XMC4500

Microcontroller Series
for Industrial Applications

XMC4000 Family

ARM® Cortex™-M4
32-bit processor core

Reference Manual

V1.0 2012-02

Microcontrollers

Edition 2012-02

Published by
Infineon Technologies AG
81726 Munich, Germany

Legal Disclaimer

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics. With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the device, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation, warranties of non-infringement of intellectual property rights of any third party.

Information

For further information on technology, delivery terms and conditions and prices, please contact the nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements, components may contain dangerous substances. For information on the types in question, please contact the nearest Infineon Technologies Office.

Infineon Technologies components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

XMC4500

Microcontroller Series
for Industrial Applications

XMC4000 Family

ARM® Cortex™-M4
32-bit processor core

Reference Manual

V1.0 2012-02

Microcontrollers

XMC4500 Reference Manual

Revision History: V1.0 2012-02

Previous Versions:	
Page	Subjects

Trademarks

C166™, TriCore™ and DAVE™ are trademarks of Infineon Technologies AG.

ARM®, ARM Powered®, Cortex® and AMBA® are registered trademarks of ARM, Limited.

CoreSight™, ETM™, Embedded Trace Macrocell™ and Embedded Trace Buffer™ are trademarks of ARM, Limited.

Synopsys™ is a trademark of Synopsys, Inc.

We Listen to Your Comments
Is there any information in this document that you feel is wrong, unclear or missing? Your feedback will help us to continuously improve the quality of this document. Please send your proposal (including a reference to this document) to:
mcdocu.comments@infineon.com

Table of Contents

About this Document

This Reference Manual is addressed to embedded hardware and software developers. It provides the reader with detailed descriptions about the behavior of the XMC4500 series functional units and their interaction.

The manual describes the functionality of the superset device of the XMC4500 microcontroller series. For the available functionality (features) of a specific XMC4500 derivative (derivative device), please refer to the respective Data Sheet. For simplicity, the various device types are referenced by the collective term XMC4500 throughout this manual.

XMC4000 Family User Documentation

The set of user documentation includes:

- **Reference Manual**
  – decribes the functionality of the superset device.
- **Data Sheets**
  – list the complete ordering information, available features and electrical characteristics of derivative devices.
- **Errata Sheets**
  – list deviations from the specifications given in the related Reference Manual or Data Sheets. Errata Sheets are provided for the superset of devices.

Attention: Please consult all parts of the documentation set to attain consolidated knowledge about your device.

Application related guidance is provided by **Users Guides** and **Application Notes**.

Please refer to **http://www.infineon.com/xmc4000** to get access to the latest versions of those documents.

Related Documentation

The following documents are referenced:

- ARM® Cortex™-M4
  – Technical Reference Manual
  – User Guide, Reference Material
- ARM®v7-M Architecture Reference Manual
- AMBA® 3 AHB-Lite Protocol Specification

Copyright Notice

- Portions of SDMMC chapter Copyright © 2010 by Arasan Chip Systems, Inc. All rights reserved. Used with permission.
- Portions of CPU chapter Copyright © 2009, 2010 by ARM, Ltd. All rights reserved. Used with permission.

- Portions of ETH, USB and GPDMA chapter Copyright © 2009, 2010 by Synopsys, Inc. All rights reserved. Used with permission.

Text Conventions

This document uses the following naming conventions:

- Functional units of the device are given in plain UPPER CASE. For example: “The USIC0 unit supports…”.
- Pins using negative logic are indicated by an overline. For example: “The $\overline{\text{WAIT}}$ input has…”.
- Bit fields and bits in registers are generally referenced as “Module_RegisterName.BitField” or “Module_RegisterName.Bit”. For example: “The USIC0_PCR.MCLK bit enables the…”. Most of the register names contain a module name prefix, separated by an underscore character “_” from the actual register name (for example, “USIC0_PCR”, where “USIC0” is the module name prefix, and “PCR” is the kernel register name). In chapters describing the kernels of the peripheral modules, the registers are mainly referenced with their kernel register names. The peripheral module implementation sections mainly refer to the actual register names with module prefixes.
- Variables used to describe sets of processing units or registers appear in mixed upper and lower cases. For example, register name “MOFCRn” refers to multiple “MOFCR” registers with variable n. The bounds of the variables are always given where the register expression is first used (for example, “n = 0-31”), and are repeated as needed in the rest of the text.
- The default radix is decimal. Hexadecimal constants are suffixed with a subscript letter “H”, as in 100_H. Binary constants are suffixed with a subscript letter “B”, as in: 111_B.
- When the extent of register fields, groups register bits, or groups of pins are collectively named in the body of the document, they are represented as “NAME[A:B]”, which defines a range for the named group from B to A. Individual bits, signals, or pins are given as “NAME[C]” where the range of the variable C is given in the text. For example: CFG[2:0] and SRPN[0].
- Units are abbreviated as follows:
  - **MHz** = Megahertz
  - **µs** = Microseconds
  - **kBaud, kbit/s** = 1000 characters/bits per second
  - **MBaud, Mbit/s, Mbps** = 1,000,000 characters/bits per second
  - **Kbyte, KB** = 1024 bytes of memory
  - **Mbyte, MB** = 1048576 bytes of memory
    In general, the k prefix scales a unit by 1000 whereas the K prefix scales a unit by 1024. Hence, the Kbyte unit scales the expression preceding it by 1024. The kBaud unit scales the expression preceding it by 1000. The M prefix scales by 1,000,000 or 1048576. For example, 1 Kbyte is 1024 bytes, 1 Mbyte is

1024 × 1024 bytes, 1 kBaud/kbit are 1000 characters/bits per second, 1 MBaud/Mbit are 1000000 characters/bits per second, and 1 MHz is 1,000,000 Hz.

- Data format quantities are defined as follows:
  - **Byte** = 8-bit quantity
  - **Half-word** = 16-bit quantity
  - **Word** = 32-bit quantity
  - **Double-word** = 64-bit quantity

Bit Function Terminology

In tables where register bits or bit fields are defined, the following conventions are used to indicate the access types.

Table 1 Bit Function Terminology

Bit Function	Description
rw	The bit or bit field can be read and written.
rwh	As rw, but bit or bit field can be also set or reset by hardware. If not otherwise documented the software takes priority in case of a write conflict between software and hardware.
r	The bit or bit field can only be read (read-only).
w	The bit or bit field can only be written (write-only). A read to this register will always give a default value back.
rh	This bit or bit field can be modified by hardware (read-hardware, typical example: status flags). A read of this bit or bit field give the actual status of this bit or bit field back. Writing to this bit or bit field has no effect to the setting of this bit or bit field.

Register Access Modes

Read and write access to registers and memory locations are sometimes restricted. In memory and register access tables, the following terms are used.

Table 2 Register Access Modes

Symbol	Description
U	Access permitted when software executes on Unprivileged level.
PV	Access permitted when software executes on Privileged level.
32	Only 32-bit word accesses are permitted to this register/address range.
NC	No change, indicated register is not changed.

Table 2 Register Access Modes (cont'd)

Symbol	Description
BE	Indicates that an access to this address range generates a Bus Error.
nBE	Indicates that no Bus Error is generated when accessing this address range.

Reserved Bits

Register bit fields named **Reserved** or **0** indicate unimplemented functions with the following behavior:

- Reading these bit fields returns 0.
- These bit fields should be written with 0 if the bit field is defined as r or rh.
- These bit fields must to be written with 0 if the bit field is defined as rw.

Abbreviations and Acronyms

The following acronyms and terms are used in this document:

ADC	Analog-to-Digital Converter
AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
ASC	Asynchronous Serial Channel
BMI	Boot Mode Index
BROM	Boot ROM
CAN	Controller Area Network
CMSIS	Cortex Microcontroller Software Interface Standard
CPU	Central Processing Unit
CRC	Cyclic Redundancy Code
CCU4	Capture Compare Unit 4
CCU8	Capture Compare Unit 8
DAC	Digital to Analog Converter
DSD	Delta Sigma Demodulator
DSRAM	Data SRAM
DMA	Direct Memory Access
EBU	External Bus Interface
ECC	Error Correction Code

ERU	Event Request Unit
ETH	Ethernet Unit
FCE	Flexible CRC Engine
FCS	Flash Command State Machine
FIM	Flash Interface and Control Module
FPU	Floating Point Unit
GPDMA	General Purpose Direct Memory Access
GPIO	General Purpose Input/Output
HMI	Human-Machine Interface
IIC	Inter Integrated Circuit (also known as I2C)
IIS	Inter-IC Sound Interface
I/O	Input / Output
JTAG	Joint Test Action Group = IEEE1149.1
LED	Light Emitting Diode
LEDTS	LED and Touch Sense (Control Unit)
LIN	Local Interconnect Network
MPU	Memory Protection Unit
MSB	Most Significant Bit
NC	Not Connected
NMI	Non-Maskable Interrupt
NVIC	Nested Vectored Interrupt Controller
OCDS	On-Chip Debug System
OTP	One Time Programmable
PBA	Peripheral Bridge AHB to AHB
PFLASH	Program Flash Memory
PLL	Phase Locked Loop
PMU	Program Memory Unit
POSIF	Position Interface
PSRAM	Program SRAM
RAM	Random Access Memory
RTC	Real Time Clock
SCU	System Control Unit

SDMMC	Secure Digital / Multi Media Card (Interface)
SDRAM	Synchronous Dynamic Random Access Memory
SFR	Special Function Register
SPI	Serial Peripheral Interface
SRAM	Static RAM
SR	Service Request
SSC	Synchronous Serial Channel
SSW	Startup Software
UART	Universal Asynchronous Receiver Transmitter
UCB	User Configuration Block
USB	Universal Serial Bus
USIC	Universal Serial Interface Channel
WDT	Watchdog Timer

Introduction

1 Introduction

The XMC4500 series belongs to the XMC4000 family of industrial microcontrollers based on the ARM Cortex-M4 processor core. The XMC4500 series devices are optimized for electrical motor control, power conversion, industrial connectivity and sense & control applications.

The growing complexity of today's energy efficient embedded control applications are demanding microcontroller solutions with higher performance CPU cores featuring DSP (Digital Signal Processing) and FPU (Floating Point Unit) capabilities as well as integrated peripherals that are optimized for performance. Complemented with a development environment designed to shorten product development time and increase productivity, the XMC4500 series of microcontrollers take advantage of Infineon's decades of experience in microcontroller design, providing an optimized solution to meet the performance challenges of today's embedded control applications.

1.1 Overview

The XMC4500 series devices combine the extended functionality and performance of the ARM Cortex-M4 core with powerful on-chip peripheral subsystems and on-chip memory units. The following key features are available in the XMC4500 series devices:

CPU Subsystem

- CPU Core
  - High Performance 32-bit ARM Cortex-M4 CPU
  - 16-bit and 32-bit Thumb2 instruction set
  - DSP/MAC instructions
  - System timer (SysTick) for Operating System support
- Floating Point Unit
- Memory Protection Unit
- Nested Vectored Interrupt Controller
- Two General Purpose DMA with up to 12 channels
- Event Request Unit (ERU) for programmable processing of external and internal service requests
- Flexible CRC Engine (FCE) for multiple bit error detection

On-Chip Memories

- 16 KB on-chip boot ROM
- 64 KB on-chip high-speed program memory
- 64 KB on-chip high speed data memory
- 32 KB on-chip high-speed communication
- 1024 KB on-chip Flash Memory with 4 KB instruction cache

Communication Peripherals

- Ethernet MAC module capable of 10/100 Mbit/s transfer rates
- Universal Serial Bus, USB 2.0 host, Full-Speed OTG, with integrated PHY
- Controller Area Network interface (MultiCAN), Full-CAN/Basic-CAN with 3 nodes, 64 message objects, data rate up to 1 Mbit/s
- Six Universal Serial Interface Channels (USIC), usable as UART, double-SPI, quadSPI, IIC, IIS and LIN interfaces
- LED and Touch-Sense Controller (LEDTS) for Human-Machine interface
- SD and Multi-Media Card interface (SDMMC) for data storage memory cards
- External Bus Interface Unit (EBU) enabling communication with external memories and off-chip peripherals like SRAM, SDRAM, NOR, NAND and Burst Flash.

Analog Frontend Peripherals

- Four Analog-Digital Converters (VADC) of 12-bit resolution, 8 channels each with input out-of-range comparators for overvoltage detection
- Delta Sigma Demodulator with four channels, digital input stage for A/D signal conversion as digital filter input stage for direct sigma-delta modulator usage
- Digital-Analogue Converter (DAC) with two channels of 12-bit resolution

Industrial Control Peripherals

- Two Capture/Compare Units 8 (CCU8) for motor control and power conversion
- Four Capture/Compare Units 4 (CCU4) for use as general purpose timers
- Two Position Interfaces (POSIF) for hall and quadratur encoders and motor positioning
- Window Watchdog Timer (WDT) for safety sensitive applications
- Die Temperature Sensor (DTS)
- Real Time Clock module with alarm support
- System Control Unit (SCU) for system configuration and control

Input/Output Lines

- Programmable port driver control module (PORTS)
- Individual bit addressability
- Tri-stated in input mode
- Push/pull or open drain output mode
- Boundary scan test support over JTAG interface

On-Chip Debug Support

- Full support for debug features: 8 breakpoints, CoreSight, trace
- Various interfaces: ARM-JTAG, SWD, single wire trace

Packages

- PG-LQFP-144
- PG-LQFP-100
- PG-LFBGA-144

Note: For details about package availability for a particular derivative please check the datasheet. For information on available delivery options for assembly support and general package see ***http://www.infineon.com/packages***

1.1.1 Block Diagram

The diagram below shows the functional blocks and their basic connectivity within the XMC4500 System.

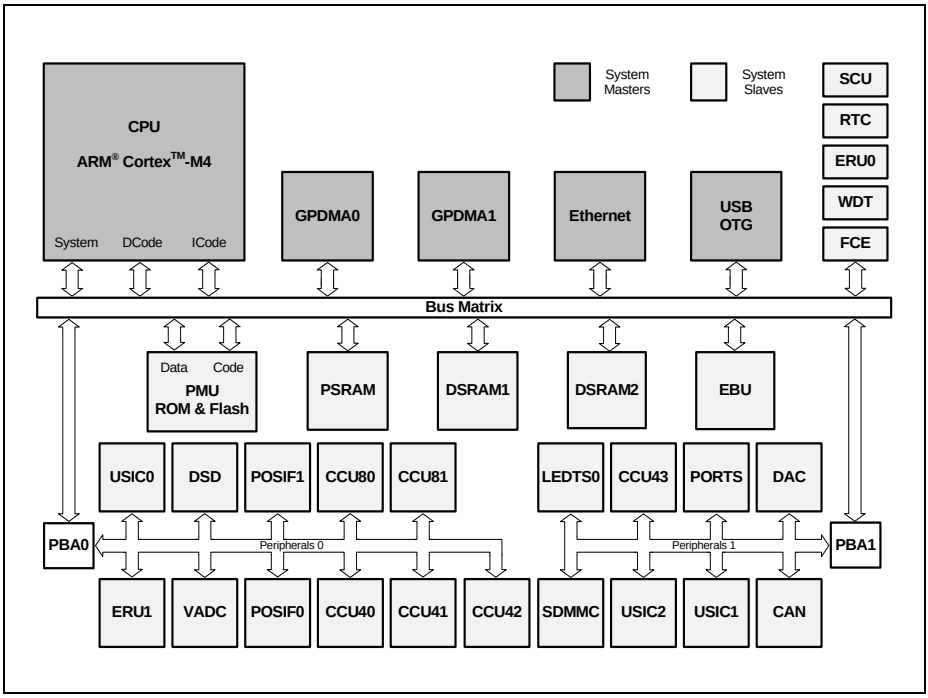


Figure 1-1 XMC4500 System

1.2 CPU Subsystem

The XMC4500 system core consists of the CPU (including FPU and MPU) and the memory interface blocks for program and data memories - PMU and EBU.

Central Processing Unit (CPU)

The Cortex-M4 processor is built on a high-performance processor core with a 3-stage pipelined Harvard architecture, making it ideal for demanding embedded applications. The processor delivers exceptional power efficiency through an efficient instruction set and a design optimized for energy efficient control applications. To address the growing complexity of embedded control it also includes a IEEE754-compliant single-precision floating-point computation and a range of single-cycle/SIMD multiplication and multiply-and-accumulate capabilities, saturating arithmetic and dedicated hardware division.

To facilitate the design of cost-sensitive devices, the Cortex-M4 processor implements tightly-coupled system components that reduce processor area while significantly improving interrupt handling and system debug capabilities.

To ensure high code density and reduced program memory requirements the processor also implements a version of the Thumb® instruction set based on Thumb-2 technology. The instruction set provides the exceptional performance expected of a modern 32-bit architecture with the high code density of 8-bit and 16-bit microcontrollers.

Floating Point Unit (FPU)

The Floating-point unit (FPU) provides IEEE754-compliant operations on single precision, 32-bit, floating-point values.

Memory Protection Unit (MPU)

The MPU improves system reliability by defining the memory attributes for different memory regions. It provides fine grain memory control, enabling applications to utilize multiple privilege levels, separating and protecting code, data and stack on a task-by-task basis. Up to eight different regions are supported as well as an optional predefined background region. These features are becoming critical to support safety requirements in many embedded applications.

Programmable Multiple Priority Interrupt System (NVIC)

The XMC4500 implements the ARM NVIC with 112 interrupt nodes and 64 priority levels. Most interrupt sources are connected to a dedicated interrupt node. In addition the XMC4500 allows to route service request directly to dedicated units like DMA, Timer and ADC. In some cases, multi-source interrupt nodes are incorporated for efficient use of system resources. These nodes can be activated by several source requests and are controlled via interrupt sub node control registers.

Direct Memory Access (GPDMA)

The GPDMA is a highly configurable DMA controller that allows high-speed data transfers between peripherals and memories. Complex data transfers can be done with minimal intervention of the processor, keeping the CPU resources free for other operations. Provides multi block, scatter/gather and linked list transfers.

Flexible CRC Engine (FCE)

The FCE provides a parallel implementation of Cyclic Redundancy Code (CRC) algorithms. It implements the IEEE 802.3 CRC32, the CCITT CRC16 and the SAE J1850 CRC8 polynomials. The primary target of FCE is to be used as a hardware acceleration engine for software applications or operating systems services using CRC signatures.

1.3 On-Chip Memories

The on-chip memories provide zero-waitstate accesses to code and data. The memories can also be accessed concurrently from various system masters.

Various types of dedicated memories are available on-chip. The suggested use of the memories aims to improve performance and system stability in most typical application cases. However, the user has the flexibility to use the memories in any other way in order to fulfill application specific requirements.

In order to meet the needs of applications where more peripherals are required the External Bus Unit (EBU) also provides means to optionally attach a broad variety of external memories.

Boot ROM (BROM)

The Boot ROM memory contains the boot code and the exception vector table. The basic system initialization sequence code, also referred to as firmware, is executed immediately after reset release.

Flash memory

The Flash is for nonvolatile code or constant data storage. The single supply Flash module is programmable at production line end and in application via built-in erase and program commands. Read and write protection mechanism are offered. A hardware error correction ensures data consistency over the whole life time under rugged industrial environment and temperatures.

The integrated cache provides an average performance boost factor of 3 in code execution compared to uncached execution.

Code RAM (PSRAM)

The Code RAM is intended for user code or Operating System data storage. The memory is accessed via the Bus Matrix and provides zero-wait-state access for the CPU for code execution or data access.

System RAM (DSRAM1)

The System RAM is intended for general user data storage. The System RAM is accessed via the Bus Matrix and provides zero-wait-state access for data.

Communication RAM (DSRAM2)

The Communication RAM is intended for use by communication interface units like the USB and Ethernet modules.

1.4 Communication Peripherals

Communication features are key requirements in today's industrial systems. The XMC4500 offers a set of peripherals supporting advanced communication protocols. Besides Ethernet, USB, CAN and the USIC the XMC4500 provides interfaces to various memories as well as a unit to realize a human-machine interface via LED and Touch Sense.

LED and Touch Sense (LEDTS)

The LEDTS module drives LEDs and controls touch pads used in human-machine interface (HMI) applications. The LEDTS can measure the capacitance of up to 8 touch pads using the relaxation oscillator (RO) topology. The module can also drive up to 64 LEDs in an LED matrix. Touch pads and LEDs can share pins to minimize the number of pins needed for such applications.

SD/MMC interface (SDMMC)

The Secure Digital/ MultiMediaCard interface (SDMMC) provides an interface between SD/SDIO/MMC cards and the system bus. It supports SD, SDIO, SDHC and MMC cards, and can operate up to 48 MHz. The SDMMC module is able to transfer a maximum of 24 MB/s for SD cards and 48 MB/s for MMC cards.

The SDMMC Host Controller handles SDIO/SD protocol at transmission level, packing data, adding cyclic redundancy check (CRC), start/end bit, and checking for transaction format correctness. Useful applications of the SDMMC interface include memory extension, data logging, and firmware update.

External Bus Unit (EBU)

The EBU supports accesses to asynchronous and synchronous external memories:

- ROMs, EPROMs
- NOR and NAND flash devices
- Static RAMs and PSRAMs
- PC133/100 compatible SDRAM
- Burst FLASH

Ethernet MAC (ETH)

The Ethernet MAC (ETH) is a major communication peripheral that supports 10/100 Mbit/s data transfer rates in compliance with the IEEE 802.3-2002 standard.

The ETH may be used to implement Internet connected applications using IPv4 and IPv6. The ETH also includes support for IEEE1588 time synchronisation to allow implimentation of Real Time Ethernet protocols.

Universal Serial Bus (USB)

The USB module is a Dual-Role Device (DRD) controller that supports both device and host functions and complies fully with the On-The-Go supplement to the USB 2.0 Specification, Revision 1.3. It can also be configured as a host-only or device-only controller, fully compliant with the USB 2.0 Specification.

The USB core's USB 2.0 configurations support full-speed (12 Mbit/s) transfers.

The USB core is optimized for the following applications and systems:

- Portable electronic devices
- Point-to-point applications (direct connection to FS device)

Universal Serial Interface Channel (USIC)

The USIC is a flexible interface module covering several serial communication protocols such as ASC, LIN, SSC, I2C, I2S. A USIC module contains two independent communication channels. Three USIC modules are implemented, hence six channels can be used in parallel. A FIFO allows transmit and result buffering for relaxing realtime conditions. Mulitple chip select signals are available for communication with mulitple devices on the same channel.

Controller Area Network (CAN)

The MultiCAN module contains three independently operating CAN nodes with Full-CAN functionality that are able to exchange Data and Remote Frames via a gateway function. Transmission and reception of CAN frames is handled in accordance with CAN specification V2.0 B (active). Each CAN node can receive and transmit standard frames with 11-bit identifiers as well as extended frames with 29-bit identifiers. Transmission rate is up to 1 Mbit/s

All CAN nodes share a common set of message objects. Each message object can be individually allocated to one of the CAN nodes. Besides serving as a storage container

for incoming and outgoing frames, message objects can be combined to build gateways between the CAN nodes or to setup a FIFO buffer.

1.5 Analog Frontend Peripherals

The XMC4500 hosts a number of interfaces to connect to the analog world.

Analog to Digital Converter (VADC)

The Versatile Analog-to-Digital Converter module consists of four independent kernels which operate according to the successive approximation principle (SAR). The resolution is programmable from 8 to 12bit with a total conversion time of less then 500ns @12bit.

Each kernel provides a versatile state machine allowing complex measurement sequences. The kernels can be synchronized and conversions may run completely in background. Multiple trigger events can be prioritized and allow the exact measurement of time critical signals. The result buffering and handling avoids data loss and ensures consistency. Selftest mechanisms can be used for plausability checks.

The basic structure supports a clean software architecture where tasks may only read valid results and do not need to care for starting conversions.

A number of out-of-range on-chip comparators serve the purpose of overvoltage monitoring for analog input pins of the VADC.

Delta- Sigma Demodulator (DSD)

The Delta-Sigma Demodulator module allows the direct usage of external Delta-Sigma Modulators for analog signal measurement.

The four input channels convert the incoming bit streams into discrete values. Each demodulator channel exists of two programmable digital filter chains (SINC/COMB type). A fast filter can be used for limit checking and a slower filter for signal measurement. An integrator stage supports carrier frequency cancellation. A special mechanism can compensate a phase delay between two channels. A built in pattern generator generates a digitized sine bitstream. This can be used for exitation of a resolver coil in motor position applications.

Digital to Analog Convertor (DAC)

The module consists of two separate 12-bit Digital-to-Analog Converters (DACs). It converts two digital input signals into two analog voltage signal outputs at a maximum conversion rate of 5 MHz.

A built-in wave generator mode allows stand alone generation of a selectable choice of wave forms. Alternatively values can be fed via CPU or DMA directly to one or both DAC

channels. Additionally an offset can be added and the amplitude can be scaled. Several time trigger sources are possible.

1.6 Industrial Control Peripherals

Core components needed for motion and motor control, power conversion and other time based applications.

Capture/Compare Unit 4 (CCU4)

The CCU4 peripheral is a major component for systems that need general purpose timers for signal monitoring/conditioning and Pulse Width Modulation (PWM) signal generation. Power electronic control systems like switched mode power supplies or uninterruptible power supplies can easily be implemented with the functions inside the CCU4 peripheral.

The internal modularity of CCU4 translates into a software friendly system for fast code development and portability between applications.

Capture/Compare Unit 8 (CCU8)

The CCU8 peripheral functions play a major role in applications that need complex Pulse Width Modulation (PWM) signal generation, with complementary high side and low side switches, multi phase control or output parity checking. The CCU8 is optimized for state of the art motor control, multi phase and multi level power electronics systems.

The internal modularity of CCU8 translates into a software friendly system for fast code development and portability between applications.

Position Interface Unit (POSIF)

The POSIF unit is a flexible and powerful component for motor control systems that use Rotary Encoders or Hall Sensors as feedback loop. The configuration schemes of the module target a very large number of motor control application requirements.

This enables the build of simple and complex control feedback loops for industrial and automotive motor applications, targeting high performance motion and position monitoring.

1.7 On-Chip Debug Support

The On-Chip Debug Support system based on the ARM CoreSight provides a broad range of debug and emulation features built into the XMC4500. The user software can therefore be debugged within the target system environment.

The On-Chip Debug Support is controlled by an external debugging device via the debug interface and an optional break interface. The debugger controls the On-Chip Debug Support via a set of dedicated registers accessible via the debug interface. Additionally,

the On-Chip Debug Support system can be controlled by the CPU, e.g. by a monitor program.

CPU Subsystem

2 Central Processing Unit (CPU)

The XMC4500 features the ARM Cortex-M4 processor. A high performance 32-bit processor designed for the microcontroller market. This CPU offers significant benefits to users, including:

- outstanding processing performance combined with fast interrupt handling
- enhanced system debug with extensive breakpoint and trace capabilities
- platform security robustness, with integrated memory protection unit (MPU).
- ultra-low power consumption with integrated sleep modes

References to ARM Documentation

The following documents can be found through **http://infocenter.arm.com**

[1] Cortex™-M4 Devices, Generic User Guide (ARM DUI 0553A)

[2] Cortex Microcontroller Software Interface Standard (CMSIS)

References to ARM Figures

[3] **http://www.arm.com**

References to IEEE Documentation

[4] IEEE Standard IEEE Standard for Binary Floating-Point Arithmetic 754-2008.

2.1 Overview

The Cortex-M4 processor is built on a high-performance processor core, with a 3-stage pipeline Harvard architecture, making it ideal for demanding embedded applications. The processor delivers exceptional power efficiency through an efficient instruction set and extensively optimized design, providing high-end processing hardware including IEEE754-compliant single-precision floating-point computation, a range of single-cycle and SIMD multiplication and multiply-with-accumulate capabilities, saturating arithmetic and dedicated hardware division.

To facilitate the design of cost-sensitive devices, the Cortex-M4 processor implements tightly-coupled system components that reduce processor area while significantly improving interrupt handling and system debug capabilities. The Cortex-M4 processor implements a version of the Thumb® instruction set based on Thumb-2 technology, ensuring high code density and reduced program memory requirements. The Cortex-M4 instruction set provides the exceptional performance expected of a modern 32-bit architecture, with the high code density of 8-bit and 16-bit microcontrollers.

The Cortex-M4 processor closely integrates a configurable NVIC, to deliver industry-leading interrupt performance. The NVIC includes a non-maskable interrupt (NMI), and provides up to 64 interrupt priority levels. The tight integration of the processor core and

NVIC provides fast execution of interrupt service routines (ISRs), dramatically reducing the interrupt latency. This is achieved through the hardware stacking of registers, and the ability to suspend load-multiple and store-multiple operations. Interrupt handlers do not require wrapping in assembler code, removing any code overhead from the ISRs. A tail-chain optimization also significantly reduces the overhead when switching from one ISR to another.

To optimize low-power designs, the NVIC integrates with the sleep modes, that include a deep sleep function that enables the entire device to be rapidly powered down while still retaining program state.

2.1.1 Features

The XMC4500 CPU features comprise

- Thumb2 instruction set combines high code density with 32-bit performance
- IEEE754-compliant single-precision FPU
- power control optimization of system components
- integrated sleep modes for low power consumption
- fast code execution permits slower processor clock or increases sleep mode time
- hardware division and fast digital-signal-processing orientated multiply accumulate
- saturating arithmetic for signal processing
- deterministic, high-performance interrupt handling for time-critical applications
- memory protection unit (MPU) for safety-critical applications
- extensive debug and trace capabilities:
  – Serial Wire Debug and Serial Wire Trace reduce the number of pins required for debugging, tracing, and code profiling.

2.1.2 Block Diagram

The Cortex-M4 core components comprise:

Processor Core

The CPU provides 16-bit and 32-bit Thumb2 instruction set and DSP/MAC instructions.

Floating-point unit

The FPU provides IEEE754-compliant operations on single-precision, 32-bit, floating-point values.

Nested Vectored Interrupt Controller

The NVIC is an embedded interrupt controller that supports low latency interrupt processing.

Memory Protection Unit

The MPU improves system reliability by defining the memory attributes for different memory regions. It provides up to eight different regions, and an optional predefined background region.

Debug Solution

The XMC4500 implements a complete hardware debug solution.

- Embedded Trace Macrocell
- Traditional JTAG port or a 2-pin Serial Wire Debug Access Port
- Trace port or Serial Wire Viewer
- Flash breakpoints and Data watchpoints

This provides high system control and visibility of the processor and memory even in small package devices.

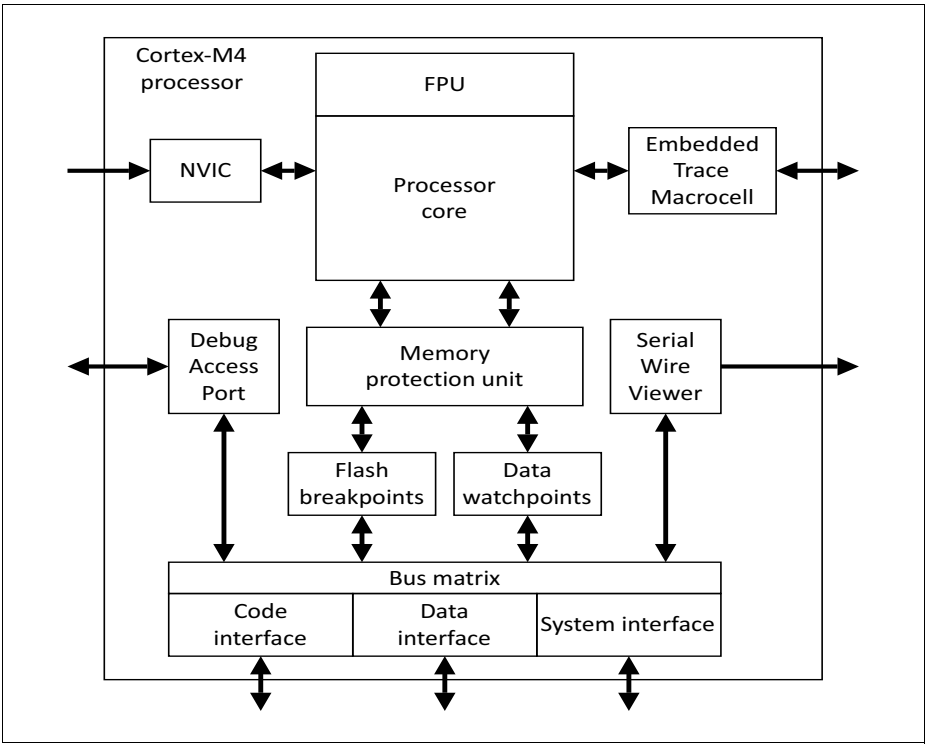


Figure 2-1 Cortex-M4 Block Diagram

System Level Interfaces

The Cortex-M4 processor provides a code, data and system interface using AMBA® technology to provide high speed, low latency accesses.

2.2 Programmers Model

This section describes the Cortex-M4 programmers model. In addition to the individual core register descriptions, it contains information about the processor modes and privilege levels for software execution and stacks.

2.2.1 Processor Mode and Privilege Levels for Software Execution

The processor modes are:

- **Thread mode**
  Used to execute application software. The processor enters Thread mode when it comes out of reset.
- **Handler mode**
  Used to handle exceptions. The processor returns to Thread mode when it has finished all exception processing.

The privilege levels for software execution are:

- **Unprivileged**
  Unprivileged software executes at the unprivileged level.
  The software:
  – has limited access to the MSR and MRS instructions, and cannot use the CPS instruction
  – cannot access the system timer, NVIC, or system control block
  – might have restricted access to memory or peripherals.
- **Privileged**
  Privileged software executes at the privileged level.
  The software can use all the instructions and has access to all resources.

In Thread mode, the CONTROL register controls whether software execution is privileged or unprivileged, see CONTROL register on **Page 2-15**. In Handler mode, software execution is always privileged.

Only privileged software can write to the CONTROL register to change the privilege level for software execution in Thread mode. Unprivileged software can use the SVC instruction to make a supervisor call to transfer control to privileged software.

2.2.2 Stacks

The processor uses a full descending stack. This means the stack pointer holds the address of the last stacked item in memory. When the processor pushes a new item onto the stack, it decrements the stack pointer and then writes the item to the new memory

location. The processor implements two stacks, the main stack and the process stack, with a pointer for each held in independent registers, see Stack Pointer on **Page 2-8**.

In Thread mode, the CONTROL register controls whether the processor uses the main stack or the process stack, see CONTROL register on **Page 2-15**. In Handler mode, the processor always uses the main stack. The options for processor operations are:

Table 2-1 Summary of processor mode, execution privilege level, and stack use options

Processor mode	Used to execute	Privilege level for software execution	Stack used
Thread	Applications	Privileged or unprivileged[1)]	Main stack or process stack[1)]
Handler	Exception handlers	Always privileged	Main stack

1) See CONTROL register on **Page 2-15**.

2.2.3 Core Registers

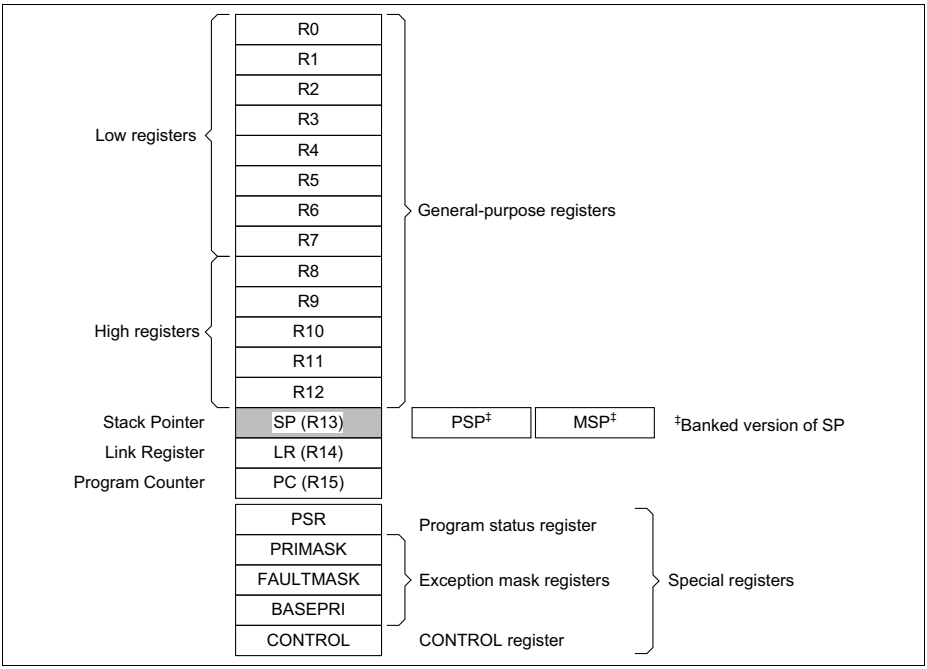


Figure 2-2 Core registers

The processor core registers are:

Table 2-2 Core register set summary

Name	Type [1]	Required privilege [2]	Reset value	Description
R0-R12	rw	Either	Unknown	General-purpose registers on **Page 2-7**
MSP	rw	Privileged	See description	Stack Pointer on **Page 2-8**
PSP	rw	Either	Unknown	Stack Pointer on **Page 2-8**
LR	rw	Either	$FFFFFFFF_H$	Link Register on **Page 2-8**
PC	rw	Either	See description	Program Counter on **Page 2-8**

Table 2-2 Core register set summary (cont'd)

Name	Type [1]	Required privilege [2]	Reset value	Description
PSR	rw	Privileged	01000000_H	Program Status Register on Page 2-9
ASPR	rw	Either	Unknown	Application Program Status Register on Page 2-9
IPSR	r	Privileged	00000000_H	Interrupt Program Status Register on Page 2-10
EPSR	r	Privileged	01000000_H	Execution Program Status Register on Page 2-11
PRIMASK	rw	Privileged	00000000_H	Priority Mask Register on Page 2-13
FAULTMASK	rw	Privileged	00000000_H	Fault Mask Register on Page 2-14
BASEPRI	rw	Privileged	00000000_H	Base Priority Mask Register on Page 2-14
CONTROL	rw	Privileged	00000000_H	CONTROL register on Page 2-15

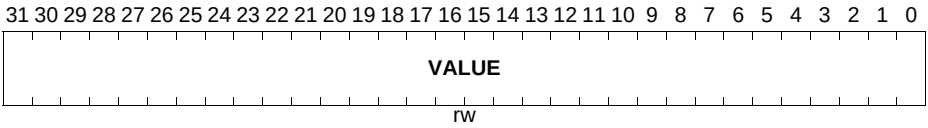
1) Describes access type during program execution in thread mode and Handler mode. Debug access can differ.

2) An entry of Either means privileged and unprivileged software can access the register.

General-purpose registers

R0-R12 are 32-bit general-purpose registers for data operations

Rx (x=0-12)
General Purpose Register Rx **Reset Value: XXXX XXXX$_H$**



Field	Bits	Type	Description
VALUE	[31:0]	rw	**Content of Register**

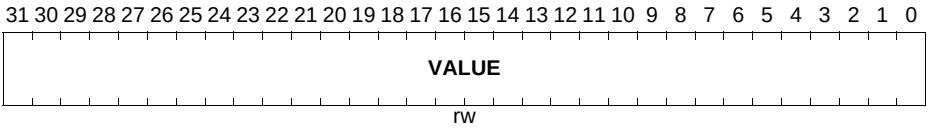
Stack Pointer

The Stack Pointer (SP) is register R13. In Thread mode, bit[1] of the CONTROL register indicates the stack pointer to use:

- 0 = Main Stack Pointer (MSP). This is the reset value.
- 1 = Process Stack Pointer (PSP).

On reset, the processor loads the MSP with the value from address 00000000_H.

SP
Stack Pointer **Reset Value: 2000 FF3C$_H$**

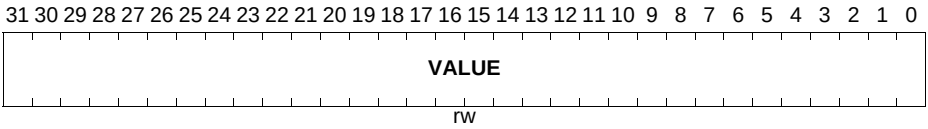


Field	Bits	Type	Description
VALUE	[31:0]	rw	**Content of Register**

Link Register

The Link Register (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions. On reset, the processor sets the LR value to $FFFFFFFF_H$.

LR
Link Register **Reset Value: FFFF FFFF$_H$**

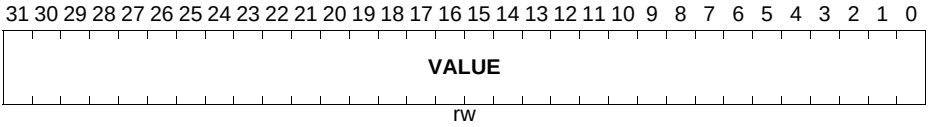


Field	Bits	Type	Description
VALUE	[31:0]	rw	**Content of Register**

Program Counter

The Program Counter (PC) is register R15. It contains the current program address. On reset, the processor loads the PC with the value of the reset vector, which is at address 00000004H. Bit[0] of the value is loaded into the EPSR T-bit at reset and must be 1.

PC
Program Counter **Reset Value: 0000 0004$_H$**



Field	Bits	Type	Description
VALUE	[31:0]	rw	**Content of Register**

Program Status Register

The Program Status Register (PSR) combines:

- Application Program Status Register (APSR)
- Interrupt Program Status Register (IPSR)
- Execution Program Status Register (EPSR)

These registers are mutually exclusive bit fields in the 32-bit PSR.

Access these registers individually or as a combination of any two or all three registers, using the register name as an argument to the MSR or MRS instructions. For example:

- read all of the registers using PSR with the MRS instruction
- write to the APSR N, Z, C, V, and Q bits using APSR_nzcvq with the MSR instruction.

The PSR combinations and attributes are:

Table 2-3 PSR register combinations

Register	Type	Combination
PSR	rw[1)2)]	APSR, EPSR, and IPSR
IEPSR	r	EPSR and IPSR
IAPSR	rw[1)]	APSR and IPSR
EAPSR	rw[2)]	APSR and EPSR

1) The processor ignores writes to the IPSR bits.

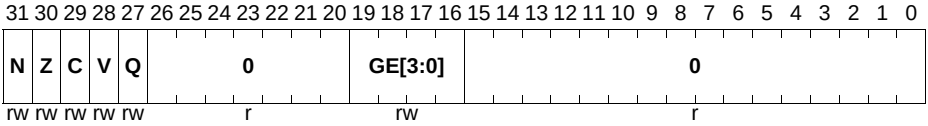
2) Reads of the EPSR bits return zero, and the processor ignores writes to the these bits

Application Program Status Register

The APSR contains the current state of the condition flags from previous instruction executions. See the register summary in **Table 2-2** on **Page 2-6** for its attributes.

APSR
Application Program Status Register

Reset Value: XXXX XXXX$_H$



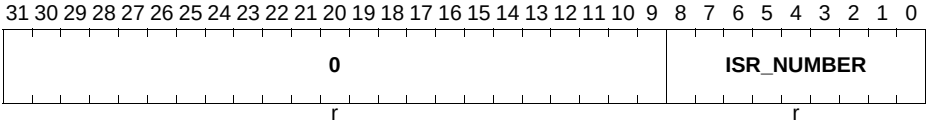
Field	Bits	Type	Description
GE[3:0]	[19:16]	rw	**Greater than or Equal flags** Please refer also to SEL instruction.
Q	27	rw	**DSP overflow and saturation flag**
V	28	rw	**Overflow flag**
C	29	rw	**Carry or borrow flag**
Z	30	rw	**Zero flag**
N	31	rw	**Negative flag**
0	[26:20], [15:0]	r	**Reserved** Read as 0; should be written with 0.

Interrupt Program Status Register

The IPSR contains the exception type number of the current Interrupt Service Routine (ISR). See the register summary in **Table 2-2** on **Page 2-6** for its attributes.

IPSR
Interrupt Program Status Register

Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
ISR_NUMBER	[8:0]	r	**Number of the current exception** 0_D Thread mode 1_D Reserved 2_D NMI 3_D HardFault 4_D MemManage 5_D BusFault 6_D UsageFault 7_D Reserved 8_D Reserved 9_D Reserved 10_D Reserved 11_D SVCall 12_D Reserved for Debug 13_D Reserved 14_D PendSV 15_D SysTick 16_D IRQ0 ... 127_D IRQ111 Values > 127_D undefined. See Exception types on **Page 2-26** for more information.
0	[31:9]	r	**Reserved** Read as 0; should be written with 0.

Execution Program Status Register

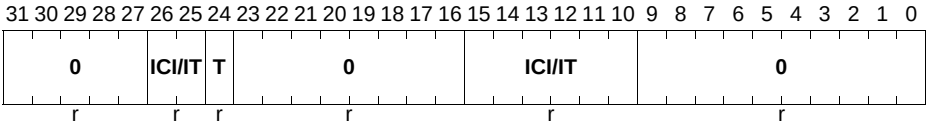
The EPSR contains the Thumb state bit, and the execution state bits for either the:

- If-Then (IT) instruction
- Interruptible-Continuable Instruction (ICI) field for an interrupted load multiple or store multiple instruction.

See the register summary in **Table 2-2** on **Page 2-6** for the EPSR attributes.

Attempts to read the EPSR directly through application software using the MSR instruction always return zero. Attempts to write the EPSR using the MSR instruction in application software are ignored.

EPSR
Execution Program Status Register **Reset Value: 0100 0000$_H$**



Field	Bits	Type	Description
ICI/IT	[26:25], [15:10]	r	**Interruptible-continuable instruction bits/Execution state bits of the IT instruction** Please refer also to IT instruction.
T	24	r	**Thumb state bit** Thumb state.
0	[31:27], [23:16], [9:0]	r	**Reserved** Read as 0; should be written with 0.

Interruptible-continuable instructions

When an interrupt occurs during the execution of an LDM, STM, PUSH, POP, VLDM, VSTM, VPUSH, or VPOP instruction, the processor:

- stops the load multiple or store multiple instruction operation temporarily
- stores the next register operand in the multiple operation to EPSR bits[15:12]

After servicing the interrupt, the processor:

- returns to the register pointed to by bits[15:12]
- resumes execution of the multiple load or store instruction.

When the EPSR holds ICI execution state, bits[26:25,11:10] are zero.

If-Then block

The If-Then block contains up to four instructions following an IT instruction. Each instruction in the block is conditional. The conditions for the instructions are either all the same, or some can be the inverse of others. See IT on page 3-122 for more information.

Thumb state

The Cortex-M4 processor only supports execution of instructions in Thumb state. The following can clear the T bit to 0:

- instructions BLX, BX and POP{PC}

- restoration from the stacked xPSR value on an exception return
- bit[0] of the vector value on an exception entry or reset.

Attempting to execute instructions when the T bit is 0 results in a fault or lockup. See Lockup on **Page 2-39** for more information.

Exception mask registers

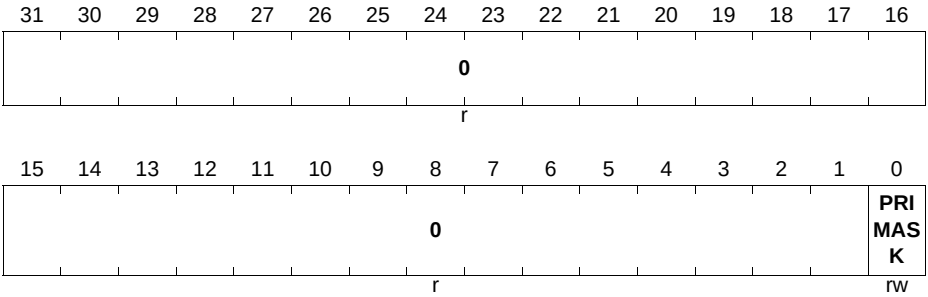
The exception mask registers disable the handling of exceptions by the processor. Disable exceptions where they might impact on timing critical tasks.

To access the exception mask registers use the MSR and MRS instructions, or the CPS instruction to change the value of PRIMASK or FAULTMASK.

Priority Mask Register

The PRIMASK register prevents activation of all exceptions with configurable priority. See the register summary in **Table 2-2** on **Page 2-6** for its attributes.

PRIMASK
Priority Mask Register **Reset Value: 0000 0000$_H$**

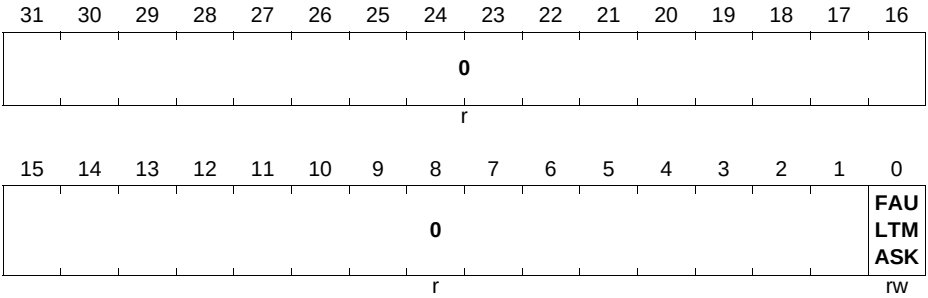


Field	Bits	Type	Description
PRIMASK	0	rw	**Priority Mask** 0_B No effect 1_B Prevents the activation of all exceptions with configurable priority.
0	[31:1]	r	**Reserved** Read as 0; should be written with 0.

Fault Mask Register

The FAULTMASK register prevents activation of all exceptions except for Non-Maskable Interrupt (NMI). See the register summary in **Table 2-2** on **Page 2-6** for its attributes.

FAULTMASK
Fault Mask Register **Reset Value: 0000 0000$_H$**



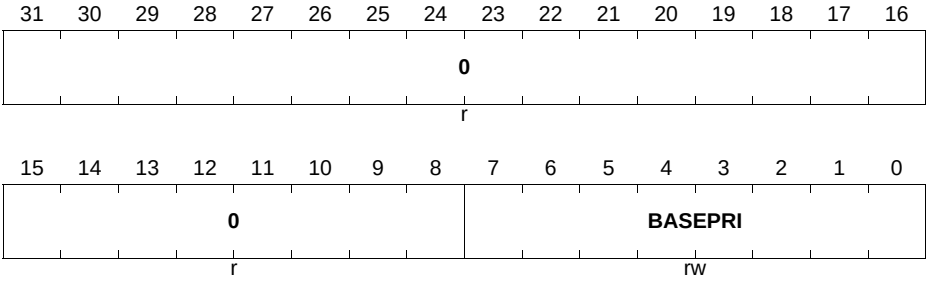
Field	Bits	Type	Description
FAULTMASK	0	rw	**Fault Mask** 0_B no effect 1_B prevents the activation of all exceptions except for NMI.
0	[31:1]	r	**Reserved** Read as 0; should be written with 0.

The processor clears the FAULTMASK bit to 0 on exit from any exception handler except the NMI handler.

Base Priority Mask Register

The BASEPRI register defines the minimum priority for exception processing. When BASEPRI is set to a nonzero value, it prevents the activation of all exceptions with the same or lower priority level as the BASEPRI value. See the register summary in **Table 2-2** on **Page 2-6** for its attributes.

BASEPRI
Base Priority Mask Register **Reset Value: 0000 0000$_H$**



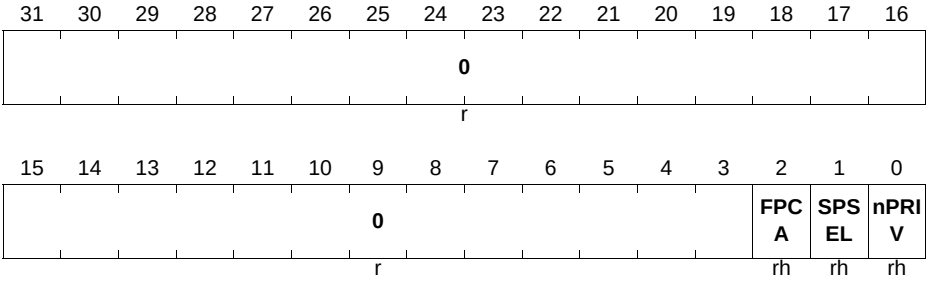
Field	Bits	Type	Description
BASEPRI[1)]	[7:0]	rw	**Priority mask bits** 0_H no effect **others**, defines the base priority for exception processing. The processor does not process any exception with a priority value greater than or equal to BASEPRI.
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

1) This field is similar to the priority fields in the interrupt priority registers. The XMC4500 implements only bits[7:2] of this field, bits[1:0] read as zero and ignore writes. See **Interrupt Priority Registers** on **Page 2-89** for more information. Remember that higher priority field values correspond to lower exception priorities.

CONTROL register

The CONTROL register controls the stack used and the privilege level for software execution when the processor is in Thread mode and indicates whether the FPU state is active. See the register summary in **Table 2-2** on **Page 2-6** for its attributes.

CONTROL
CONTROL register **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
nPRIV	0	rh	**Thread mode privilege level** 0_B Privileged 1_B Unprivileged
SPSEL	1	rh	**Currently active stack pointer** In Handler mode this bit reads as zero and ignores writes. The Cortex-M4 updates this bit automatically on exception return. 0_B MSP is the current stack pointer 1_B PSP is the current stack pointer
FPCA	2	rh	**Floating-point context currently active** 0_B No floating-point context active 1_B Floating-point context active The Cortex-M4 uses this bit to determine whether to preserve floating-point state when processing an exception.
0	[31:3]	r	**Reserved** Read as 0; should be written with 0.

Handler mode always uses the MSP, so the processor ignores explicit writes to the active stack pointer bit of the CONTROL register when in Handler mode. The exception entry and return mechanisms automatically update the CONTROL register based on the EXC_RETURN value, see **Table 2-9** on **Page 2-36**.

In an OS environment, ARM recommends that threads running in Thread mode use the process stack and the kernel and exception handlers use the main stack.

By default, Thread mode uses the MSP. To switch the stack pointer used in Thread mode to the PSP, either:

- use the MSR instruction to set the Active stack pointer bit to 1.
- perform an exception return to Thread mode with the appropriate EXC_RETURN value, see **Table 2-9** on **Page 2-36**.

Note: When changing the stack pointer, software must use an ISB instruction immediately after the MSR instruction. This ensures that instructions after the ISB instruction execute using the new stack pointer.

2.2.4 Exceptions and Interrupts

The Cortex-M4 processor supports interrupts and system exceptions. The processor and the NVIC prioritize and handle all exceptions. An exception changes the normal flow of software control. The processor uses Handler mode to handle all exceptions except for reset. See Exception entry on **Page 2-33** and Exception return on **Page 2-36** for more information.

The NVIC registers control interrupt handling. See **Page 2-43** for more information.

2.2.5 Data Types

The processor:

- supports the following data types:
  - 32-bit words
  - 16-bit halfwords
  - 8-bit bytes
- manages all data memory accesses as little-endian. See Memory regions, types and attributes on **Page 2-20** for more information.

2.2.6 The Cortex Microcontroller Software Interface Standard

For a Cortex-M4 microcontroller system, the Cortex Microcontroller Software Interface Standard (CMSIS) **[2]** defines:

- a common way to:
  - access peripheral registers
  - define exception vectors
- the names of:
  - the registers of the core peripherals
  - the core exception vectors
- a device-independent interface for RTOS kernels, including a debug channel.

The CMSIS includes address definitions and data structures for the core peripherals in the Cortex-M4 processor.

CMSIS simplifies software development by enabling the reuse of template code and the combination of CMSIS-compliant software components from various middleware vendors. Software vendors can expand the CMSIS to include their peripheral definitions and access functions for those peripherals.

This document includes the register names defined by the CMSIS, and gives short descriptions of the CMSIS functions that address the processor core and the core peripherals.

Note: This document uses the register short names defined by the CMSIS. In a few cases these differ from the architectural short names that might be used in other documents.

The following sections give more information about the CMSIS:

- Power management programming hints on **Page 2-42**
- CMSIS functions on **Page 2-18**
- Using CMSIS functions to access NVIC on **Page 2-45**

For additional information please refer to **http://www.onarm.com/cmsis**

2.2.7 CMSIS functions

ISO/IEC C code cannot directly access some Cortex-M4 instructions. This section describes intrinsic functions that can generate these instructions, provided by the CMSIS and that might be provided by a C compiler. If a C compiler does not support an appropriate intrinsic function, you might have to use inline assembler to access some instructions.

The CMSIS provides the following intrinsic functions to generate instructions that ISO/IEC C code cannot directly access:

Table 2-4 CMSIS functions to generate some Cortex-M4 instructions

Instruction	CMSIS function
CPSIE I	`void __enable_irq(void)`
CPSID I	`void __disable_irq(void)`
CPSIE F	`void __enable_fault_irq(void)`
CPSID F	`void __disable_fault_irq(void)`
ISB	`void __ISB(void)`
DSB	`void __DSB(void)`
DMB	`void __DMB(void)`
REV	`uint32_t __REV(uint32_t int value)`
REV16	`uint32_t __REV16(uint32_t int value)`

Table 2-4 CMSIS functions to generate some Cortex-M4 instructions (cont'd)

Instruction	CMSIS function
REVSH	`uint32_t __REVSH(uint32_t int value)`
RBIT	`uint32_t __RBIT(uint32_t int value)`
SEV	`void __SEV(void)`
WFE	`void __WFE(void)`
WFI	`void __WFI(void)`

The CMSIS also provides a number of functions for accessing the special registers using MRS and MSR instructions:

Table 2-5 CMSIS functions to access the special registers

Special register	Access	CMSIS function
PRIMASK	Read	`uint32_t __get_PRIMASK (void)`
	Write	`void __set_PRIMASK (uint32_t value)`
FAULTMASK	Read	`uint32_t __get_FAULTMASK (void)`
	Write	`void __set_FAULTMASK (uint32_t value)`
BASEPRI	Read	`uint32_t __get_BASEPRI (void)`
	Write	`void __set_BASEPRI (uint32_t value)`
CONTROL	Read	`uint32_t __get_CONTROL (void)`
	Write	`void __set_CONTROL (uint32_t value)`
MSP	Read	`uint32_t __get_MSP (void)`
	Write	`void __set_MSP (uint32_t TopOfMainStack)`
PSP	Read	`uint32_t __get_PSP (void)`
	Write	`void __set_PSP (uint32_t TopOfProcStack)`

2.3 Memory Model

This section describes the processor memory map and the behavior of memory accesses. The processor has a fixed default memory map that provides up to 4GB of addressable memory. The memory map is:

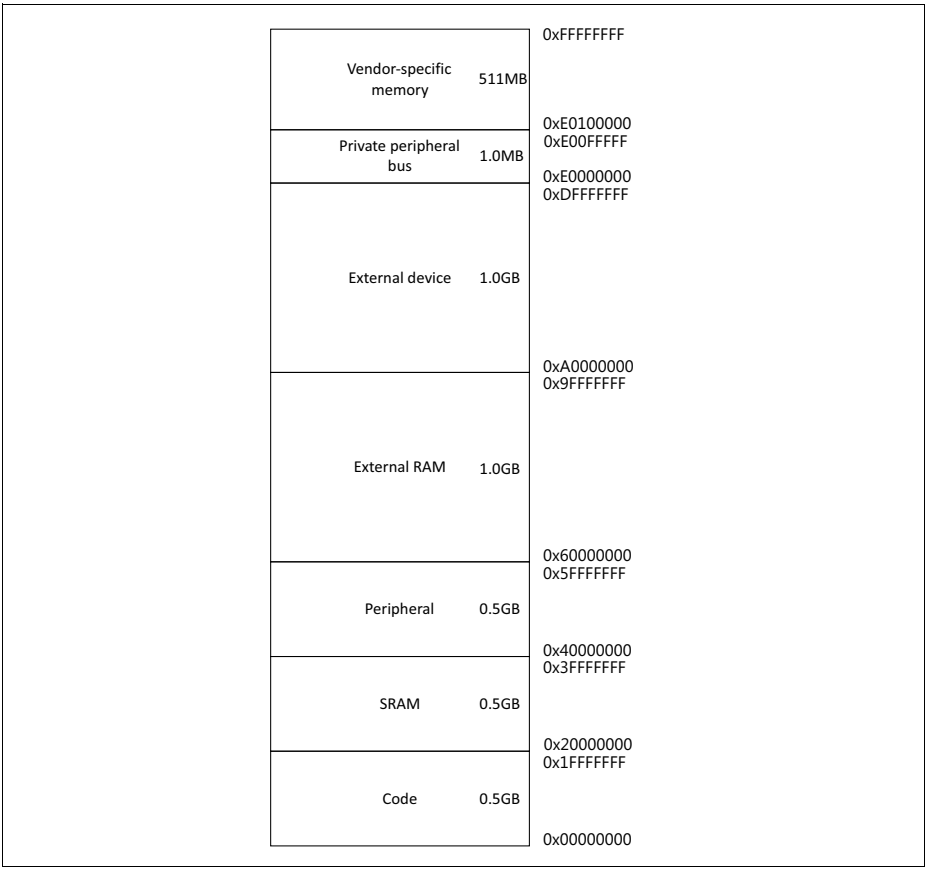


Figure 2-3 Memory map

The processor reserves regions of the Private peripheral bus (PPB) address range for core peripheral registers, see About the Private Peripherals on **Page 2-42**.

2.3.1 Memory Regions, Types and Attributes

The memory map and the programming of the MPU splits the memory map into regions. Each region has a defined memory type, and some regions have additional memory

attributes. The memory type and attributes determine the behavior of accesses to the region.

The memory types are:

Normal	The processor can re-order transactions for efficiency, or perform speculative reads.
Device	The processor preserves transaction order relative to other transactions to Device or Strongly-ordered memory.
Strongly-ordered	The processor preserves transaction order relative to all other transactions.

The different ordering requirements for Device and Strongly-ordered memory mean that the memory system can buffer a write to Device memory, but must not buffer a write to Strongly-ordered memory.

The additional memory attributes include:

Execute Never (XN)	Means the processor prevents instruction accesses. A fault exception is generated only on execution of an instruction executed from an XN region.

2.3.2 Memory System Ordering of Memory Accesses

For most memory accesses caused by explicit memory access instructions, the memory system does not guarantee that the order in which the accesses complete matches the program order of the instructions, providing this does not affect the behavior of the instruction sequence. Normally, if correct program execution depends on two memory accesses completing in program order, software must insert a memory barrier instruction between the memory access instructions. See Software ordering of memory accesses on **Page 2-23**.

However, the memory system does guarantee some ordering of accesses to Device and Strongly-ordered memory. For two memory access instructions A1 and A2, if A1 occurs before A2 in program order, the ordering of the memory accesses caused by two instructions is:

A1 \ A2	Normal access	Device access	Strongly-ordered access
Normal access	-	-	-
Device access	-	<	<
Strongly-ordered access	-	<	<

Figure 2-4 Ordering of Memory Accesses

Where:

- "-" Means that the memory system does not guarantee the ordering of the accesses.
- "<" Means that accesses are observed in program order, that is, A1 is always observed before A2.

2.3.3 Behavior of Memory Accesses

The behavior of accesses to each region in the memory map is:

Table 2-6 Memory access behavior

Address range	Memory region	Memory type[1)]	XN[1)]	Description
0x00000000-0x1FFFFFFF	Code	Normal	-	Executable region for program code. You can also put data here.
0x20000000-0x3FFFFFFF	SRAM	Normal	-	Executable region for data. You can also put code here.
0x40000000-0x5FFFFFFF	Peripheral	Device	XN	Peripherals region.
0x60000000-0x9FFFFFFF	External RAM	Normal	-	Executable region for data.
0xA0000000-0xDFFFFFFF	External device	Device	XN	External Device memory.
0xE0000000-0xE00FFFFF	Private Peripheral Bus	Strongly-ordered	XN	This region includes the NVIC, System timer, and system control block.
0xE0100000-0xFFFFFFFF	Vendor-specific device	Device	XN	Accesses to this region are to vendor-specific peripherals.

1) See Memory regions, types and attributes on **Page 2-20** for more information.

The Code, SRAM, and external RAM regions can hold programs. However, it is recommended that programs always use the Code region. This is because the processor has separate buses that enable instruction fetches and data accesses to occur simultaneously.

The MPU can override the default memory access behavior described in this section. For more information, see Memory protection unit on **Page 2-46**.

Instruction prefetch and branch prediction

The Cortex-M4 processor:

- prefetches instructions ahead of execution
- speculatively prefetches from branch target addresses.

2.3.4 Software Ordering of Memory Accesses

The order of instructions in the program flow does not always guarantee the order of the corresponding memory transactions. This is because:

- the processor can reorder some memory accesses to improve efficiency, providing this does not affect the behavior of the instruction sequence.
- The processor has multiple bus interfaces
- memory or devices in the memory map have different wait states
- some memory accesses are buffered or speculative.

Memory system ordering of memory accesses on **Page 2-21** describes the cases where the memory system guarantees the order of memory accesses. Otherwise, if the order of memory accesses is critical, software must include memory barrier instructions to force that ordering. The processor provides the following memory barrier instructions:

DMB The Data Memory Barrier (DMB) instruction ensures that outstanding memory transactions complete before subsequent memory transactions.

DSB The Data Synchronization Barrier (DSB) instruction ensures that outstanding memory transactions complete before subsequent instructions execute.

ISB The Instruction Synchronization Barrier (ISB) ensures that the effect of all completed memory transactions is recognizable by subsequent instructions.

MPU programming

Use a DSB followed by an ISB instruction or exception return to ensure that the new MPU configuration is used by subsequent instructions.

2.3.5 Memory Endianness

The processor views memory as a linear collection of bytes numbered in ascending order from zero. For example, bytes 0-3 hold the first stored word, and bytes 4-7 hold the second stored word. The XMC4500 stores information “Little-endian” format.

Little-endian format

In little-endian format, the processor stores the least significant byte of a word at the lowest-numbered byte, and the most significant byte at the highest-numbered byte. For example:

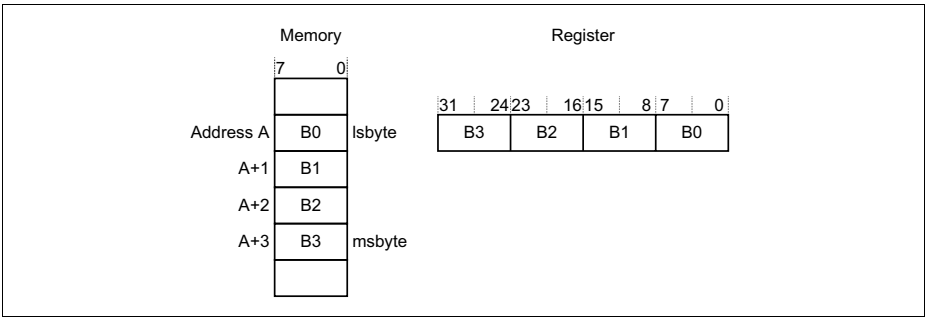


Figure 2-5 Little-endian format

2.3.6 Synchronization Primitives

The Cortex-M4 instruction set includes pairs of synchronization primitives. These provide a non-blocking mechanism that a thread or process can use to obtain exclusive access to a memory location. Software can use them to perform a guaranteed read-modify-write memory update sequence, or for a semaphore mechanism.

A pair of synchronization primitives comprises:

A Load-Exclusive instruction

Used to read the value of a memory location, requesting exclusive access to that location.

A Store-Exclusive instruction

Used to attempt to write to the same memory location, returning a status bit to a register. If this bit is:

0 it indicates that the thread or process gained exclusive access to the memory, and the write succeeds,

1 it indicates that the thread or process did not gain exclusive access to the memory, and no write was performed.

The pairs of Load-Exclusive and Store-Exclusive instructions are:

- the word instructions LDREX and STREX
- the halfword instructions LDREXH and STREXH
- the byte instructions LDREXB and STREXB.

Software must use a Load-Exclusive instruction with the corresponding Store-Exclusive instruction.

To perform an exclusive read-modify-write of a memory location, software must:

1. Use a Load-Exclusive instruction to read the value of the location.
2. Modify the value, as required.
3. Use a Store-Exclusive instruction to attempt to write the new value back to the memory location.
4. Test the returned status bit. If this bit is:
   0 The read-modify-write completed successfully.
   1 No write was performed. This indicates that the value returned at step 1 might be out of date. The software must retry the entire read-modify-write sequence.

Software can use the synchronization primitives to implement a semaphores as follows:

1. Use a Load-Exclusive instruction to read from the semaphore address to check whether the semaphore is free.
2. If the semaphore is free, use a Store-Exclusive to write the claim value to the semaphore address.
3. If the returned status bit from step 2 indicates that the Store-Exclusive succeeded then the software has claimed the semaphore. However, if the Store-Exclusive failed, another process might have claimed the semaphore after the software performed step 1.

The Cortex-M4 includes an exclusive access monitor, that tags the fact that the processor has executed a Load-Exclusive instruction.

The processor removes its exclusive access tag if:

- It executes a CLREX instruction.
- It executes a Store-Exclusive instruction, regardless of whether the write succeeds.
- An exception occurs. This means the processor can resolve semaphore conflicts between different threads.

2.3.7 Programming Hints for the Synchronization Primitives

ISO/IEC C cannot directly generate the exclusive access instructions. CMSIS provides intrinsic functions for generation of these instructions:

Table 2-7 CMSIS functions for exclusive access instructions

Instruction	CMSIS function
LDREX	`uint32_t __LDREXW (uint32_t *addr)`
LDREXH	`uint16_t __LDREXH (uint16_t *addr)`
LDREXB	`uint8_t __LDREXB (uint8_t *addr)`
STREX	`uint32_t __STREXW (uint32_t value, uint32_t *addr)`
STREXH	`uint32_t __STREXH (uint16_t value, uint16_t *addr)`
STREXB	`uint32_t __STREXB (uint8_t value, uint8_t *addr)`
CLREX	`void __CLREX (void)`

For example:

```
uint16_t value;
uint16_t *address = 0x20001002;
value = __LDREXH (address);     // load 16-bit value from memory
address 0x20001002
```

2.4 Instruction Set

The Cortex-M4 instruction set reference is available through **[1]**

2.5 Exception Model

This section describes the exception model. It describes:

- Exception states
- Exception types
- Exception handlers on **Page 2-27**
- Vector table on **Page 2-30**
- Exception priorities on **Page 2-31**
- Interrupt priority grouping on **Page 2-31**
- Exception entry and return on **Page 2-32**

2.5.1 Exception States

Each exception is in one of the following states:

Inactive	The exception is not active and not pending.
Pending	The exception is waiting to be serviced by the processor. An interrupt request from a peripheral or from software can change the state of the corresponding interrupt to pending.
Active	An exception that is being serviced by the processor but has not completed. *Note: An exception handler can interrupt the execution of another exception handler. In this case both exceptions are in the active state.*
Active and pending	The exception is being serviced by the processor and there is a pending exception from the same source.

2.5.2 Exception Types

The exception types are:

Reset	Reset is invoked on power up or a warm reset. The exception model treats reset as a special form of exception. When reset is asserted, the operation of the processor stops, potentially at any point in an instruction. When reset is deasserted, execution restarts from the address provided by the reset entry in the vector table. Execution restarts as privileged execution in Thread mode.
NMI	A NonMaskable Interrupt (NMI) can be signalled by a peripheral or triggered by software. This is the highest priority exception other than reset. It is permanently enabled and has a fixed priority of -2. NMIs cannot be: • masked or prevented from activation by any other exception • preempted by any exception other than Reset.
HardFault	A HardFault is an exception that occurs because of an error during exception processing, or because an exception cannot be managed by any other exception mechanism. HardFaults have a fixed priority of -1, meaning they have higher priority than any exception with configurable priority.
MemManage	A MemManage fault is an exception that occurs because of a memory protection related fault. The MPU or the fixed memory protection constraints determines this fault, for both instruction and data memory transactions. This fault is always used to abort instruction accesses to Execute Never (XN) memory regions.

BusFault A BusFault is an exception that occurs because of a memory related fault for an instruction or data memory transaction. This might be from an error detected on a bus in the memory system.

UsageFault A UsageFault is an exception that occurs because of a fault related to instruction execution. This includes:

- an undefined instruction
- an illegal unaligned access
- invalid state on instruction execution
- an error on exception return.

The following can cause a UsageFault when the core is configured to report them:

- an unaligned address on word and halfword memory access
- division by zero.

SVCall A supervisor call (SVC) is an exception that is triggered by the SVC instruction. In an OS environment, applications can use SVC instructions to access OS kernel functions and device drivers.

PendSV PendSV is an interrupt-driven request for system-level service. In an OS environment, use PendSV for context switching when no other exception is active.

SysTick A SysTick exception is an exception the system timer generates when it reaches zero. Software can also generate a SysTick exception. In an OS environment, the processor can use this exception as system tick.

Interrupt (IRQ) A interrupt, or IRQ, is an exception signalled by a peripheral, or generated by a software request. All interrupts are asynchronous to instruction execution. In the system, peripherals use interrupts to communicate with the processor.

Table 2-8 Properties of the different exception types

Exception number[1)]	IRQ number[1)]	Exception type	Priority	Vector address or offset[2)]	Activation
1	-	Reset	-3, the highest	0x00000004	Asynchronous
2	-14	NMI	-2	0x00000008	Asynchronous
3	-13	HardFault	-1	0x0000000C	-
4	-12	MemManage	Configurable[3)]	0x00000010	Synchronous

Table 2-8 Properties of the different exception types (cont'd)

Exception number[1]	IRQ number[1]	Exception type	Priority	Vector address or offset[2]	Activation
5	-11	BusFault	Configurable[3]	0x00000014	Synchronous when precise, asynchronous when imprecise
6	-10	UsageFault	Configurable[3]	0x00000018	Synchronous
7-10	-	Reserved	-	-	-
11	-5	SVCall	Configurable[3]	0x0000002C	Synchronous
12-13	-	Reserved	-	-	-
14	-2	PendSV	Configurable[3]	0x00000038	Asynchronous
15	-1	SysTick	Configurable[3]	0x0000003C	Asynchronous
16 and above	0 and above	Interrupt (IRQ)	Configurable[4]	0x00000040 and above[5]	Asynchronous

1) To simplify the software layer, the CMSIS only uses IRQ numbers and therefore uses negative values for exceptions other than interrupts. The IPSR returns the Exception number, see Interrupt Program Status Register on **Page 2-10**.

2) See Vector table for more information.

3) See System Handler Priority Registers on **Page 2-69**

4) See Interrupt Priority Registers on **Page 2-89**.

5) Increasing in steps of 4.

For an asynchronous exception, other than reset, the processor can execute another instruction between when the exception is triggered and when the processor enters the exception handler.

Privileged software can disable the exceptions that **Table 2-8** on **Page 2-28** shows as having configurable priority, see:

- System Handler Control and State Register on **Page 2-71**
- Interrupt Clear-enable Registers on **Page 2-87**.

For more information about HardFaults, MemManage faults, BusFaults, and UsageFaults, see Fault handling on **Page 2-36**.

2.5.3 Exception Handlers

The processor handles exceptions using:

Interrupt Service Routines (ISRs)	Interrupts IRQ0 to IRQ111 are the exceptions handled by ISRs.
Fault handlers	HardFault, MemManage fault, UsageFault, and BusFault are fault exceptions handled by the fault handlers.
System handlers	NMI, PendSV, SVCall SysTick, and the fault exceptions are all system exceptions that are handled by system handlers.

2.5.4 Vector Table

The vector table contains the reset value of the stack pointer, and the start addresses, also called exception vectors, for all exception handlers. **Figure 2-6** on **Page 2-30** shows the order of the exception vectors in the vector table. The least-significant bit of each vector must be 1, indicating that the exception handler is Thumb code, see Thumb state on **Page 2-12**.

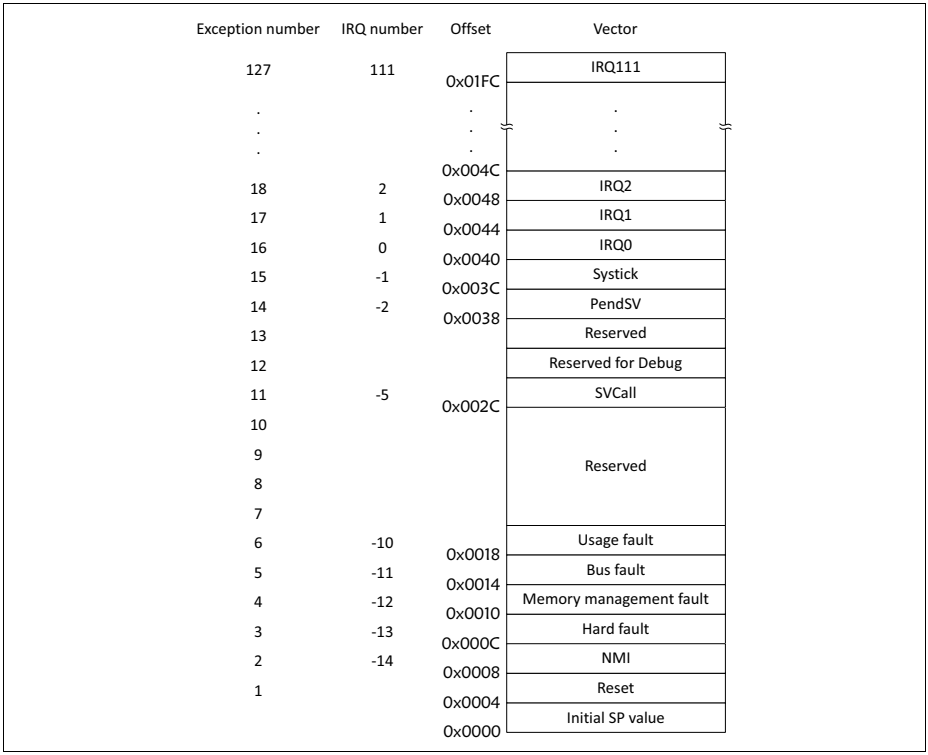


Figure 2-6 Vector table

On system reset, the vector table is fixed at address 0x00000000. Privileged software can write to the VTOR to relocate the vector table start address to a different memory location, in the range 0x00000400 to 0x3FFFFC00, see Vector Table Offset Register on **Page 2-63**.

2.5.5 Exception Priorities

As **Table 2-8** on **Page 2-28** shows, all exceptions have an associated priority, with:

- a lower priority value indicating a higher priority
- configurable priorities for all exceptions except Reset, HardFault, and NMI.

If software does not configure any priorities, then all exceptions with a configurable priority have a priority of 0. For information about configuring exception priorities see

- System Handler Priority Registers on **Page 2-69**
- Interrupt Priority Registers on **Page 2-89**.

Note: Configurable priority values are in the range 0-63. This means that the Reset, HardFault, and NMI exceptions, with fixed negative priority values, always have higher priority than any other exception.

For example, assigning a higher priority value to IRQ[0] and a lower priority value to IRQ[1] means that IRQ[1] has higher priority than IRQ[0]. If both IRQ[1] and IRQ[0] are asserted, IRQ[1] is processed before IRQ[0].

If multiple pending exceptions have the same priority, the pending exception with the lowest exception number takes precedence. For example, if both IRQ[0] and IRQ[1] are pending and have the same priority, then IRQ[0] is processed before IRQ[1].

When the processor is executing an exception handler, the exception handler is preempted if a higher priority exception occurs. If an exception occurs with the same priority as the exception being handled, the handler is not preempted, irrespective of the exception number. However, the status of the new interrupt changes to pending.

2.5.6 Interrupt Priority Grouping

To increase priority control in systems with interrupts, the NVIC supports priority grouping. This divides each interrupt priority register entry into two fields:

- an upper field that defines the group priority
- a lower field that defines a subpriority within the group.

Only the group priority determines preemption of interrupt exceptions. When the processor is executing an interrupt exception handler, another interrupt with the same group priority as the interrupt being handled does not preempt the handler,

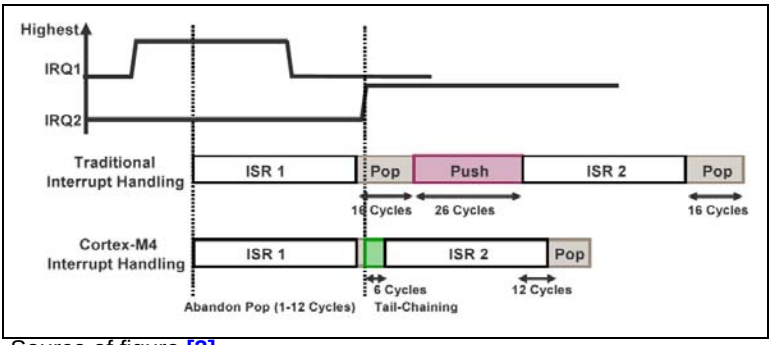
If multiple pending interrupts have the same group priority, the subpriority field determines the order in which they are processed. If multiple pending interrupts have the same group priority and subpriority, the interrupt with the lowest IRQ number is processed first.

For information about splitting the interrupt priority fields into group priority and subpriority, see Application Interrupt and Reset Control Register on **Page 2-63**.

2.5.7 Exception Entry and Return

Descriptions of exception handling use the following terms:

Preemption When the processor is executing an exception handler, an exception can preempt the exception handler if its priority is higher than the priority of the exception being handled. See Interrupt priority grouping for more information about preemption by an interrupt.
When one exception preempts another, the exceptions are called nested exceptions. See Exception entry on **Page 2-33** more information.



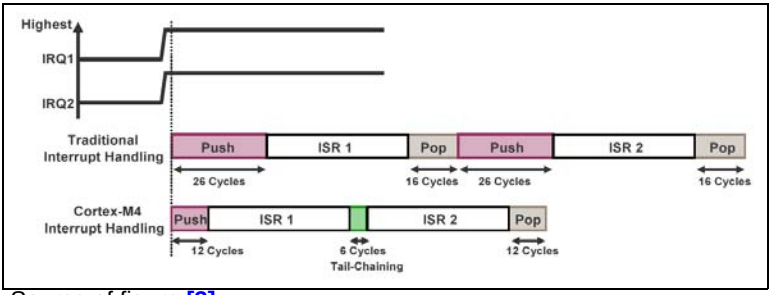
Source of figure **[3]**.

Return This occurs when the exception handler is completed, and:

- there is no pending exception with sufficient priority to be serviced
- the completed exception handler was not handling a late-arriving exception.

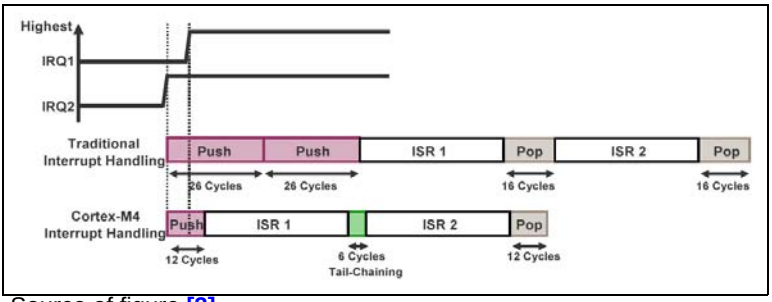
The processor pops the stack and restores the processor state to the state it had before the interrupt occurred. See Exception return on **Page 2-36** for more information.

Tail-chaining This mechanism speeds up exception servicing. On completion of an exception handler, if there is a pending exception that meets the requirements for exception entry, the stack pop is skipped and control transfers to the new exception handler.



Source of figure [3].

Late-arriving This mechanism speeds up preemption. If a higher priority exception occurs during state saving for a previous exception, the processor switches to handle the higher priority exception and initiates the vector fetch for that exception. State saving is not affected by late arrival because the state saved is the same for both exceptions. Therefore the state saving continues uninterrupted. The processor can accept a late arriving exception until the first instruction of the exception handler of the original exception enters the execute stage of the processor. On return from the exception handler of the late-arriving exception, the normal tail-chaining rules apply.



Source of figure [3].

Exception entry

Exception entry occurs when there is a pending exception with sufficient priority and either:

- the processor is in Thread mode
- the new exception is of higher priority than the exception being handled, in which case the new exception preempts the original exception.

When one exception preempts another, the exceptions are nested.

Sufficient priority means the exception has more priority than any limits set by the mask registers, see Exception mask registers on **Page 2-13**. An exception with less priority than this is pending but is not handled by the processor.

When the processor takes an exception, unless the exception is a tail-chained or a late-arriving exception, the processor pushes information onto the current stack. This operation is referred to as stacking and the structure of eight data words is referred as the stack frame.

When using floating-point routines, the Cortex-M4 processor automatically stacks the architected floating-point state on exception entry. **Figure 2-7** on **Page 2-35** shows the Cortex-M4 stack frame layout when floating-point state is preserved on the stack as the result of an interrupt or an exception.

*Note: Where stack space for floating-point state is not allocated, the stack frame is the same as that of ARMv7-M implementations without an FPU. **Figure 2-7** on **Page 2-35** shows this stack frame also.*

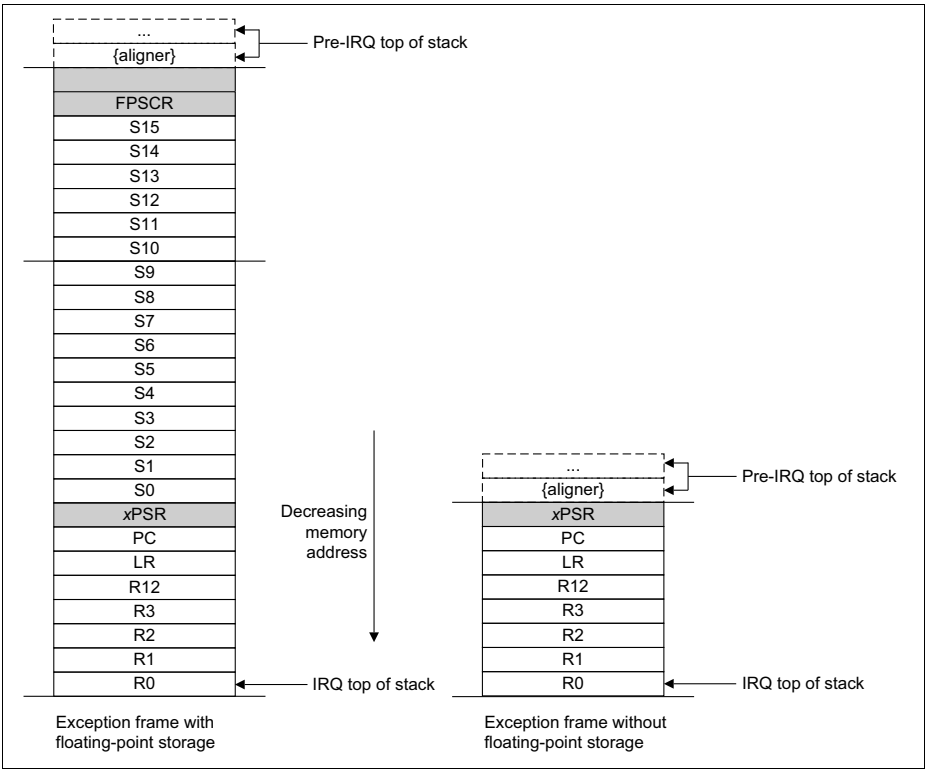


Figure 2-7 Exception stack frame

Immediately after stacking, the stack pointer indicates the lowest address in the stack frame. The alignment of the stack frame is controlled via the STKALIGN bit of the Configuration Control Register (CCR).

The stack frame includes the return address. This is the address of the next instruction in the interrupted program. This value is restored to the PC at exception return so that the interrupted program resumes.

In parallel to the stacking operation, the processor performs a vector fetch that reads the exception handler start address from the vector table. When stacking is complete, the processor starts executing the exception handler. At the same time, the processor writes an EXC_RETURN value to the LR. This indicates which stack pointer corresponds to the stack frame and what operation mode the processor was in before the entry occurred.

If no higher priority exception occurs during exception entry, the processor starts executing the exception handler and automatically changes the status of the corresponding pending interrupt to active.

If another higher priority exception occurs during exception entry, the processor starts executing the exception handler for this exception and does not change the pending status of the earlier exception. This is the late arrival case.

Exception return

Exception return occurs when the processor is in Handler mode and executes one of the following instructions to load the EXC_RETURN value into the PC:

- an LDM or POP instruction that loads the PC
- an LDR instruction with PC as the destination
- a BX instruction using any register.

EXC_RETURN is the value loaded into the LR on exception entry. The exception mechanism relies on this value to detect when the processor has completed an exception handler. The lowest five bits of this value provide information on the return stack and processor mode. **Table 2-9** shows the EXC_RETURN values with a description of the exception return behavior.

All EXC_RETURN values have bits[31:5] set to one. When this value is loaded into the PC it indicates to the processor that the exception is complete, and the processor initiates the appropriate exception return sequence.

Table 2-9 Exception return behavior

EXC_RETURN[31:0]	Description
0xFFFFFFF1	Return to Handler mode, exception return uses non-floating-point state from the MSP and execution uses MSP after return.
0xFFFFFFF9	Return to Thread mode, exception return uses non-floating-point state from MSP and execution uses MSP after return.
0xFFFFFFFD	Return to Thread mode, exception return uses non-floating-point state from the PSP and execution uses PSP after return.
0xFFFFFFE1	Return to Handler mode, exception return uses floating-point-state from MSP and execution uses MSP after return.
0xFFFFFFE9	Return to Thread mode, exception return uses floating-point state from MSP and execution uses MSP after return.
0xFFFFFFED	Return to Thread mode, exception return uses floating-point state from PSP and execution uses PSP after return.

2.6 Fault Handling

Faults are a subset of the exceptions, see Exception model on **Page 2-26**. Faults are generated by:

- a bus error on:

  - an instruction fetch or vector table load
  - a data access.
- an internally-detected error such as an undefined instruction
- attempting to execute an instruction from a memory region marked as Non-Executable (XN).
- a privilege violation or an attempt to access an unmanaged region causing an MPU fault

2.6.1 Fault Types

Table 2-10 shows the types of fault, the handler used for the fault, the corresponding fault status register, and the register bit that indicates that the fault has occurred. See Configurable Fault Status Register on page 4-24 for more information about the fault status registers.

Table 2-10 Faults

Fault	Handler	Bit name	Fault status register
Bus error on a vector read	HardFault	VECTTBL	HardFault Status Register on **Page 2-80**
Fault escalated to a hard fault		FORCED	
MPU or default memory map mismatch:	MemManage	-	-
on instruction access		IACCVIOL[1)]	MemManage Fault Address Register on **Page 2-81**
on data access		DACCVIOL	
during exception stacking		MSTKERR	
during exception unstacking		MUNSKERR	
during lazy floating-point state preservation		MLSPERR	

Table 2-10 Faults (cont'd)

Fault	Handler	Bit name	Fault status register
Bus error:	BusFault	-	-
during exception stacking		STKERR	BusFault Status Register on Page 2-73
during exception unstacking		UNSTKERR	
during instruction prefetch		IBUSERR	
during lazy floating-point state preservation		LSPERR	
Precise data bus error		PRECISERR	
Imprecise data bus error		IMPRECISERR	
Attempt to access a coprocessor	UsageFault	NOCP	UsageFault Status Register on Page 2-73
Undefined instruction		UNDEFINSTR	
Attempt to enter an invalid instruction set state[2)]		INVSTATE	
Invalid EXC_RETURN value		INVPC	
Illegal unaligned load or store		UNALIGNED	
Divide By 0		DIVBYZERO	

1) Occurs on an access to an XN region even if the processor does not include an MPU or the MPU is disabled.

2) Attempting to use an instruction set other than the Thumb instruction set or returns to a non load/store-multiple instruction with ICI continuation.

2.6.2 Fault Escalation and Hard Faults

All faults exceptions except for HardFault have configurable exception priority, see System Handler Priority Registers on page 4-21. Software can disable execution of the handlers for these faults, see System Handler Control and State Register on page 4-23.

Usually, the exception priority, together with the values of the exception mask registers, determines whether the processor enters the fault handler, and whether a fault handler can preempt another fault handler. as described in Exception model on **Page 2-26**.

In some situations, a fault with configurable priority is treated as a HardFault. This is called priority escalation, and the fault is described as escalated to HardFault. Escalation to HardFault occurs when:

- A fault handler causes the same kind of fault as the one it is servicing. This escalation to HardFault occurs because a fault handler cannot preempt itself because it must have the same priority as the current priority level.
- A fault handler causes a fault with the same or lower priority as the fault it is servicing. This is because the handler for the new fault cannot preempt the currently executing fault handler.
- An exception handler causes a fault for which the priority is the same as or lower than the currently executing exception.
- A fault occurs and the handler for that fault is not enabled.

If a BusFault occurs during a stack push when entering a BusFault handler, the BusFault does not escalate to a HardFault. This means that if a corrupted stack causes a fault, the fault handler executes even though the stack push for the handler failed. The fault handler operates but the stack contents are corrupted.

Note: Only Reset and NMI can preempt the fixed priority HardFault. A HardFault can preempt any exception other than Reset, NMI, or another HardFault.

2.6.3 Fault Status Registers and Fault Address Registers

The fault status registers indicate the cause of a fault. For BusFaults and MemManage faults, the fault address register indicates the address accessed by the operation that caused the fault, as shown in **Table 2-11**.

Table 2-11 Fault status and fault address registers

Handler	Status register name	Address register name	Register description
HardFault	HFSR	-	HardFault Status Register on **Page 2-80**
MemManage	MMFSR	MMFAR	MemManage Fault Status Register **Page 2-73** MemManage Fault Address Register **Page 2-81**
BusFault	BFSR	BFAR	BusFault Status Register on **Page 2-73** BusFault Address Register on **Page 2-82**
UsageFault	UFSR	-	UsageFault Status Register on **Page 2-73**

2.6.4 Lockup

The processor enters a lockup state if a fault occurs when executing the NMI or HardFault handlers. When the processor is in lockup state it does not execute any instructions. The processor remains in lockup state until either:

- it is reset

- an NMI occurs
- it is halted by a debugger

Note: If lockup state occurs from the NMI handler a subsequent NMI does not cause the processor to leave lockup state.

2.7 Power Management

The Cortex-M4 processor sleep modes reduce power consumption:

- Sleep mode stops the processor clock.
- Deep sleep mode stops the system clock and switches off the PLL and flash memory.

The SLEEPDEEP bit of the SCR selects which sleep mode is used, see System Control Register on **Page 2-66**. For more information about the behavior of the sleep modes see section “Power Management” in SCU chapter.

The following section describes the mechanisms for entering sleep mode, and the conditions for waking up from sleep mode.

2.7.1 Entering Sleep Mode

This section describes the mechanisms software can use to put the processor into sleep mode

The system can generate spurious wakeup events, for example a debug operation wakes up the processor. Therefore software must be able to put the processor back into sleep mode after such an event. A program might have an idle loop to put the processor back to sleep mode.

Wait for interrupt

The wait for interrupt instruction, WFI, causes immediate entry to sleep mode unless the wake-up condition is true, see Wakeup from WFI or sleep-on-exit on **Page 2-41**. When the processor executes a WFI instruction it stops executing instructions and enters sleep mode.

Wait for event

The wait for event instruction, WFE, causes entry to sleep mode depending on the value of a one-bit event register. When the processor executes a WFE instruction, it checks the value of the event register:

0 The processor stops executing instructions and enters sleep mode.
1 The processor clears the register to 0 and continues executing instructions without entering sleep mode.

If the event register is 1, this indicate that the processor must not enter sleep mode on execution of a WFE instruction. Typically, this is because an external event signal is

asserted, or a processor in the system has executed an SEV instruction, see SEV on page 3-166. Software cannot access this register directly.

Sleep-on-exit

If the SLEEPONEXIT bit of the SCR is set to 1, when the processor completes the execution of all exception handlers it returns to Thread mode and immediately enters sleep mode. Use this mechanism in applications that only require the processor to run when an exception occurs.

2.7.2 Wakeup from Sleep Mode

The conditions for the processor to wakeup depend on the mechanism that cause it to enter sleep mode.

Wakeup from WFI or sleep-on-exit

Normally, the processor wakes up only when it detects an exception with sufficient priority to cause exception entry. Some embedded systems might have to execute system restore tasks after the processor wakes up, and before it executes an interrupt handler. To achieve this set the PRIMASK bit to 1 and the FAULTMASK bit to 0. If an interrupt arrives that is enabled and has a higher priority than current exception priority, the processor wakes up but does not execute the interrupt handler until the processor sets PRIMASK to zero. For more information about PRIMASK and FAULTMASK see Exception mask registers on **Page 2-13**.

Wakeup from WFE

The processor wakes up if:

- it detects an exception with sufficient priority to cause exception entry
- it detects an external event signal, see The external event input

In addition, if the SEVONPEND bit in the SCR is set to 1, any new pending interrupt triggers an event and wakes up the processor, even if the interrupt is disabled or has insufficient priority to cause exception entry. For more information about the SCR see System Control Register on **Page 2-66**.

2.7.3 The External Event Input

The processor provides an external event input signal. Peripherals can drive this signal, either to wake the processor from WFE, or to set the internal WFE event register to one to indicate that the processor must not enter sleep mode on a later WFE instruction. See Wait for event on **Page 2-40** for more information.

2.7.4 Power Management Programming Hints

ISO/IEC C cannot directly generate the WFI and WFE instructions. The CMSIS provides the following functions for these instructions:

```
void __WFE(void)   // Wait for Event
void __WFI(void)   // Wait for Interrupt
```

2.8 Private Peripherals

The following sections are the reference material for the ARM Cortex-M4 core peripherals.

2.8.1 About the Private Peripherals

The address map of the Private Peripheral Bus (PPB) is:

Table 2-12 Core peripheral register regions

Address	Core peripheral	Description
0xE000E008-0xE000E00F	System control block	**Section 2.8.2** and **Section 2.9.1**
0xE000E010-0xE000E01F	System timer	**Section 2.8.3** and **Section 2.9.2**
0xE000E100-0xE000E4EF	Nested Vectored Interrupt Controller	**Section 2.8.4** and **Section 2.9.3**
0xE000ED00-0xE000ED3F	System control block	**Section 2.8.2** and **Section 2.9.1**
0xE000ED90-0xE000EDB8	Memory protection unit	**Section 2.8.5** and **Section 2.9.4**
0xE000EF00-0xE000EF03	Nested Vectored Interrupt Controller	**Section 2.8.4** and **Section 2.9.3**
0xE000EF30-0xE000EF44	Floating Point Unit	**Section 2.8.6** and **Section 2.9.5**

2.8.2 System control block

The System control block (SCB) provides system implementation information, and system control. This includes configuration, control, and reporting of the system exceptions. The system control block registers are:

2.8.2.1 System control block design hints and tips

Ensure software uses aligned accesses of the correct size to access the system control block registers:

- except for the CFSR and SHPR1-SHPR3, it must use aligned word accesses
- for the CFSR and SHPR1-SHPR3 it can use byte or aligned halfword or word accesses.

The processor does not support unaligned accesses to system control block registers.

In a fault handler, to determine the true faulting address:

1. Read and save the MMFAR or BFAR value.
2. Read the MMARVALID bit in the MMFSR, or the BFARVALID bit in the BFSR. The MMFAR or BFAR address is valid only if this bit is 1.

Software must follow this sequence because another higher priority exception might change the MMFAR or BFAR value. For example, if a higher priority handler preempts the current fault handler, the other fault might change the MMFAR or BFAR value.

2.8.3 System timer, SysTick

The processor has a 24-bit system timer, SysTick, that counts down from the reload value to zero, reloads, that is wraps to, the value in the **SYST_RVR** register on the next clock edge, then counts down on subsequent clocks.

Note: When the processor is halted for debugging the counter does not decrement.

2.8.3.1 SysTick design hints and tips

The SysTick counter runs on the clock selected by **SYST_CSR**.CLKSOURCE. If the selected clock signal is stopped, the SysTick counter stops.

Ensure software uses aligned word accesses to access the SysTick registers.

The SysTick counter reload and current value are undefined at reset, the correct initialization sequence for the SysTick counter is:

1. Program reload value.
2. Clear current value.
3. Program Control and Status register.

2.8.4 Nested Vectored Interrupt Controller (NVIC)

This section describes the NVIC and the registers it uses. The XMC4500 NVIC supports:

- 112 interrupts.
- A programmable priority level of 0-63 for each interrupt. A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority.
- Level and pulse detection of interrupt signals.
- Dynamic reprioritization of interrupts.

- Grouping of priority values into group priority and subpriority fields.
- Interrupt tail-chaining.
- An external Non-maskable interrupt (NMI)

The processor automatically stacks its state on exception entry and unstacks this state on exception exit, with no instruction overhead. This provides low latency exception handling. The hardware implementation of the NVIC registers is:

2.8.4.1 Level-sensitive and pulse interrupts

The processor supports both level-sensitive and pulse interrupts. Pulse interrupts are also described as edge-triggered interrupts.

A level-sensitive interrupt is held asserted until the peripheral deasserts the interrupt signal. Typically this happens because the ISR accesses the peripheral, causing it to clear the interrupt request. A pulse interrupt is an interrupt signal sampled synchronously on the rising edge of the processor clock. To ensure the NVIC detects the interrupt, the peripheral must assert the interrupt signal for at least one clock cycle, during which the NVIC detects the pulse and latches the interrupt.

When the processor enters the ISR, it automatically removes the pending state from the interrupt, see next section. For a level-sensitive interrupt, if the signal is not deasserted before the processor returns from the ISR, the interrupt becomes pending again, and the processor must execute its ISR again. This means that the peripheral can hold the interrupt signal asserted until it no longer requires servicing.

See section “Service Request Distribution” in the “Service Request Processing” chapter for details about which interrupts are level-based and which are pulsed.

Hardware and software control of interrupts

The Cortex-M4 latches all interrupts. A peripheral interrupt becomes pending for one of the following reasons:

- the NVIC detects that the interrupt signal is HIGH and the interrupt is not active
- the NVIC detects a rising edge on the interrupt signal
- software writes to the corresponding interrupt set-pending register bit, see Interrupt Set-pending Registers on **Page 2-88** or to the STIR to make an interrupt pending, see Software Trigger Interrupt Register on **Page 2-91**.

A pending interrupt remains pending until one of the following:

- The processor enters the ISR for the interrupt. This changes the state of the interrupt from pending to active. Then:
  - For a level-sensitive interrupt, when the processor returns from the ISR, the NVIC samples the interrupt signal. If the signal is asserted, the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR. Otherwise, the state of the interrupt changes to inactive.

  - For a pulse interrupt, the NVIC continues to monitor the interrupt signal, and if this is pulsed the state of the interrupt changes to pending and active. In this case, when the processor returns from the ISR the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR.
    If the interrupt signal is not pulsed while the processor is in the ISR, when the processor returns from the ISR the state of the interrupt changes to inactive.
- Software writes to the corresponding interrupt clear-pending register bit.
  For a level-sensitive interrupt, if the interrupt signal is still asserted, the state of the interrupt does not change. Otherwise, the state of the interrupt changes to inactive.
  For a pulse interrupt, state of the interrupt changes to:
  - inactive, if the state was pending
  - active, if the state was active and pending.

2.8.4.2 NVIC design hints and tips

Ensure software uses correctly aligned register accesses. The processor does not support unaligned accesses to NVIC registers. See the individual register descriptions for the supported access sizes.

A interrupt can enter pending state even if it is disabled. Disabling an interrupt only prevents the processor from taking that interrupt.

Before programming VTOR to relocate the vector table, ensure the vector table entries of the new vector table are setup for fault handlers, NMI and all enabled exception like interrupts. For more information see Vector Table Offset Register on **Page 2-63**.

2.8.4.3 Using CMSIS functions to access NVIC

CMSIS functions enable software portability between different Cortex-M profile processors. To ensure Cortex-M portability, use the functions marked for Cortex-M portability in the table below.

CMSIS provides a number of functions for NVIC control, including:

Table 2-13 CMSIS functions for NVIC control

CMSIS interrupt control function	Description	Cortex-M Portable
`void NVIC_SetPriorityGrouping( uint32_t priority_grouping)`	Set the priority grouping.	No
`uint32_t NVIC_GetPriorityGrouping( void)`	Get the priority grouping.	No
`void NVIC_EnableIRQ( IRQn_t IRQn)`	Enables IRQn.	Yes

Table 2-13 CMSIS functions for NVIC control (cont'd)

CMSIS interrupt control function	Description	Cortex-M Portable
`void NVIC_DisableIRQ( IRQn_t IRQn)`	Disables IRQn.	Yes
`uint32_t NVIC_GetPendingIRQ( IRQn_t IRQn)`	Return IRQ-Number (true) if IRQn is pending.	Yes
`void NVIC_SetPendingIRQ( IRQn_t IRQn)`	Set IRQn pending.	Yes
`void NVIC_ClearPendingIRQ( IRQn_t IRQn)`	Clear IRQn pending.	Yes
`uint32_t NVIC_GetActive( IRQn_t IRQn)`	Return the IRQ number of the active interrupt.	No
`void NVIC_SetPriority( IRQn_t IRQn, uint32_t priority)`	Set priority for IRQn.	Yes
`uint32_t NVIC_GetPriority( IRQn_t IRQn)`	Read priority of IRQn.	Yes
`uint32_t NVIC_EncodePriority( uint32_t PriorityGroup, uint32_t PreemptPriority, uint32_t SubPriority)`	Encodes the priority for an interrupt with the given priority group, preemptive priority value and sub priority value.	No
`void NVIC_DecodePriority( uint32_t Priority, uint32_t PriorityGroup, uint32_t* pPreemptPriority, uint32_t* pSubPriority)`	Decodes an interrupt priority value with the given priority group to preemptive priority value and sub priority value.	No
`void NVIC_SystemReset(void)`	Reset the system	Yes

The parameter IRQn is the IRQ number, see **Table 2-8** on **Page 2-28**. For more information about these functions see the CMSIS documentation **[4]**.

2.8.5 Memory Protection Unit (MPU)

The MPU divides the memory map into a number of regions, and defines the location, size, access permissions, and memory attributes of each region. It supports:

- independent attribute settings for each region
- overlapping regions
- export of memory attributes to the system

The memory attributes affect the behavior of memory accesses to the region. The Cortex-M4 MPU defines:

- eight separate memory regions, 0-7
- a background region

When memory regions overlap, a memory access is affected by the attributes of the region with the highest number. For example, the attributes for region 7 take precedence over the attributes of any region that overlaps region 7.

The background region has the same memory access attributes as the default memory map, but is accessible from privileged software only.

The Cortex-M4 MPU memory map is unified. This means instruction accesses and data accesses have same region settings.

If a program accesses a memory location that is prohibited by the MPU, the processor generates a MemManage fault. This causes a fault exception, and might cause termination of the process in an OS environment.

In an OS environment, the kernel can update the MPU region setting dynamically based on the process to be executed. Typically, an embedded OS uses the MPU for memory protection.

Configuration of MPU regions is based on memory types, see Memory regions, types and attributes on **Page 2-20**.

Table 2-14 shows the possible MPU region attributes.

Note: The shareability and cache attributes are not relevant to the XMC4500.

Table 2-14 Memory attributes summary

Address	Shareability	Other attributes	Description
Strongly-ordered	-	-	All accesses to Strongly-ordered memory occur in program order. All Strongly-ordered regions are assumed to be shared.
Device	Shared	-	Memory-mapped peripherals that several processors share.
	Non-shared	-	Memory-mapped peripherals that only a single processor uses.
Normal	Shared	Non-cacheable Write-through or Write-back Cacheable	Normal memory that is shared between several processors.
	Non-shared	Non-cacheable Write-through or Write-back Cacheable	Normal memory that only a single processor uses.

2.8.5.1 MPU Access Permission Attributes

This section describes the MPU access permission attributes. The access permission bits, TEX, C, B, S, AP, and XN, of the RASR, control access to the corresponding memory region. If an access is made to an area of memory without the required permissions, then the MPU generates a permission fault. **Table 2-15** shows encodings for the TEX, C, B, and S access permission bits.

Table 2-15 TEX, C, B, and S encoding

TEX	C	B	S	Memory type	Shareability	Other attributes
0b000	0	0	x	Strongly-ordered	Shareable	-
		1	x	Device	Shareable	-
	1	0	0	Normal	Not shareable	Outer and inner write-through. No write allocate.
			1		Shareable	
		1	0	Normal	Not shareable	Outer and inner write-back. No write allocate.
			1		Shareable	
0b001	0	0	0	Normal	Not shareable	Outer and inner noncacheable.
			1		Shareable	
		1	x[1)]	Reserved encoding		-
	1	0	x[1)]	Implementation defined attributes.		
		1	0	Normal	Not shareable	Outer and inner write-back. Write and read allocate.
			1		Shareable	
0b010	0	0	x[1)]	Device	Not shareable	Nonshared Device.
		1	x[1)]	Reserved encoding		-
	1	x	x[1)]	Reserved encoding		-
0b1BB	A	A	0	Normal	Not shareable	Cached memory, BB = outer policy, AA = inner policy. See **Table 2-16** on **Page 2-49** for the encoding of the AA and BB bits.
			1		Shareable	

1) The MPU ignores the value of this bit.

Table 2-16 shows the cache policy for memory attribute encodings with a TEX value is in the range 4-7.

Table 2-16 Cache policy for memory attribute encoding

Encoding, AA or BB	Corresponding cache policy
00	Non-cacheable
01	Write back, write and read allocate
10	Write through, no write allocate
11	Write back, no write allocate

MPU configuration for the XMC4500

The XMC4500 has only a single processor and no caches. However to enable portability it is recommended to program the MPU as follows:

Table 2-17 Memory region attributes for a microcontroller

Memory region	TEX	C	B	S	Memory type and attributes
Internal Flash memory	0b000	1	0	0	Normal memory, Non-shareable, write-through
Internal SRAM memories	0b000	1	0	1	Normal memory, Shareable, write-through
External memories	0b000	1	1	1	Normal memory, Shareable, write-back, write-allocate
Peripherals	0b000	0	1	1	Device memory, Shareable

Table 2-18 shows the AP encodings that define the access permissions for privileged and unprivileged software.

Table 2-18 AP encoding

AP[2:0]	Privileged permissions	Unprivileged permissions	Description
000	No access	No access	All accesses generate a permission fault
001	rw	No access	Access from privileged software only
010	rw	r	Writes by unprivileged software generate a permission fault
011	rw	rw	Full access

Table 2-18 AP encoding (cont'd)

AP[2:0]	Privileged permissions	Unprivileged permissions	Description
100	Unpredictable	Unpredictable	Reserved
101	r	No access	Reads by privileged software only
110	r	r	Read only, by privileged or unprivileged software
111	r	r	Read only, by privileged or unprivileged software

2.8.5.2 MPU Mismatch

When an access violates the MPU permissions, the processor generates a MemManage fault, see Exceptions and interrupts on **Page 2-17**. The MMFSR indicates the cause of the fault. See MemManage Fault Status Register on **Page 2-73** for more information.

2.8.5.3 Updating an MPU Region

To update the attributes for an MPU region, update the MPU_RNR, MPU_RBAR and MPU_RASR registers. You can program each register separately, or use a multiple-word write to program all of these registers. You can use the MPU_RBAR and MPU_RASR aliases to program up to four regions simultaneously using an STM instruction.

Updating an MPU region using separate words

Simple code to configure one region:

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address
LDR R0,=MPU_RNR          ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]       ; Region Number
STR R4, [R0, #0x4]       ; Region Base Address
STRH R2, [R0, #0x8]      ; Region Size and Enable
STRH R3, [R0, #0xA]      ; Region Attribute
```

Disable a region before writing new region settings to the MPU if you have previously enabled the region being changed. For example:

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address
```

```
LDR   R0,=MPU_RNR           ;  0xE000ED98,  MPU  region  number
register
STR R1, [R0, #0x0]         ; Region Number
BIC R2, R2, #1             ; Disable
STRH R2, [R0, #0x8]        ; Region Size and Enable
STR R4, [R0, #0x4]         ; Region Base Address
STRH R3, [R0, #0xA]        ; Region Attribute
ORR R2, #1                 ; Enable
STRH R2, [R0, #0x8]        ; Region Size and Enable
```

Software must use memory barrier instructions:

- before MPU setup if there might be outstanding memory transfers, such as buffered writes, that might be affected by the change in MPU settings
- after MPU setup if it includes memory transfers that must use the new MPU settings.

However, memory barrier instructions are not required if the MPU setup process starts by entering an exception handler, or is followed by an exception return, because the exception entry and exception return mechanism cause memory barrier behavior.

Software does not require any memory barrier instructions during MPU setup, because it accesses the MPU through the PPB, which is a Strongly-Ordered memory region.

For example, if you want all of the memory access behavior to take effect immediately after the programming sequence, use a DSB instruction and an ISB instruction. A DSB is required after changing MPU settings, such as at the end of context switch. An ISB is required if the code that programs the MPU region or regions is entered using a branch or call. If the programming sequence is entered using a return from exception, or by taking an exception, then you do not require an ISB.

Updating an MPU region using multi-word writes

You can program directly using multi-word writes, depending on how the information is divided. Consider the following reprogramming:

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
LDR R0, =MPU_RNR    ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]  ; Region Number
STR R2, [R0, #0x4]  ; Region Base Address
STR R3, [R0, #0x8]  ; Region Attribute, Size and Enable
```

Use an STM instruction to optimize this:

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
LDR R0, =MPU_RNR    ; 0xE000ED98, MPU region number register
```

```
STM R0, {R1-R3}     ; Region Number, address, attribute, size and
enable
```

You can do this in two words for pre-packed information. This means that the MPU_RBAR contains the required region number and had the VALID bit set to 1, see MPU Region Base Address Register on **Page 2-95**. Use this when the data is statically packed, for example in a boot loader:

```
; R1 = address and region number in one
; R2 = size and attributes in one
LDR R0, =MPU_RBAR   ; 0xE000ED9C, MPU Region Base register
STR R1, [R0, #0x0]  ; Region base address and
                    ; region number combined with VALID (bit 4)
set to 1
STR R2, [R0, #0x4]  ; Region Attribute, Size and Enable
```

Subregions

Regions of 256 bytes or more are divided into eight equal-sized subregions. Set the corresponding bit in the SRD field of the MPU_RASR to disable a subregion, see MPU Region Attribute and Size Register on **Page 2-97**. The least significant bit of SRD controls the first subregion, and the most significant bit controls the last subregion. Disabling a subregion means another region overlapping the disabled range matches instead. If no other enabled region overlaps the disabled subregion the MPU issues a fault.

Regions of 32, 64, and 128 bytes do not support subregions, With regions of these sizes, you must set the SRD field to 0x00, otherwise the MPU behavior is Unpredictable.

Example of SRD use

Two regions with the same base address overlap. Region one is 128KB, and region two is 512KB. To ensure the attributes from region one apply to the first 128KB region, set the SRD field for region two to 0b00000011 to disable the first two subregions, as the figure shows.

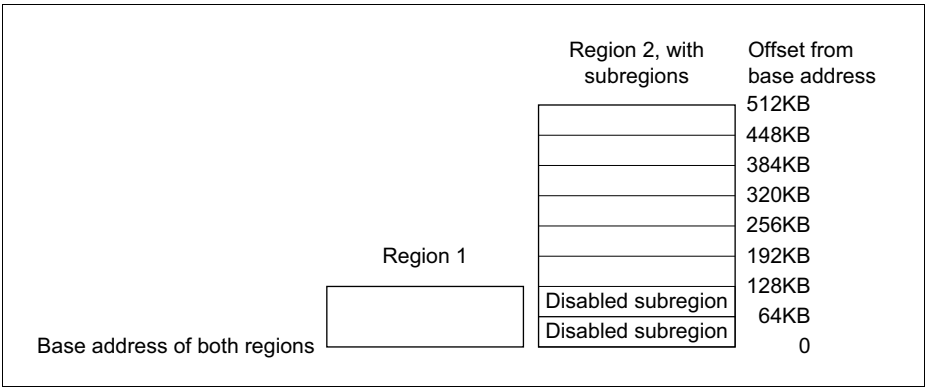


Figure 2-8 Example of SRD use

2.8.5.4 MPU Design Hints and Tips

To avoid unexpected behavior, disable the interrupts before updating the attributes of a region that the interrupt handlers might access.

Ensure software uses aligned accesses of the correct size to access MPU registers:

- except for the MPU_RASR, it must use aligned word accesses
- for the MPU_RASR it can use byte or aligned halfword or word accesses.

The processor does not support unaligned accesses to MPU registers.

When setting up the MPU, and if the MPU has previously been programmed, disable unused regions to prevent any previous region settings from affecting the new MPU setup.

In the XMC4500 the shareability and cache policy attributes do not affect the system behavior. However, using these settings for the MPU regions can make the application code more portable.

2.8.6 Floating Point Unit (FPU)

The Cortex-M4 FPU implements the FPv4-SP floating-point extension.

The FPU fully supports single-precision add, subtract, multiply, divide, multiply and accumulate, and square root operations. It also provides conversions between fixed-point and floating-point data formats, and floating-point constant instructions.

The FPU provides floating-point computation functionality that is compliant with the ANSI/IEEE Std 754-2008, IEEE Standard for Binary Floating-Point Arithmetic, referred to as the IEEE 754 standard **[4]**.

The FPU contains 32 single-precision extension registers, which you can also access as 16 doubleword registers for load, store, and move operations.

2.8.6.1 Enabling the FPU

The FPU is disabled from reset. You must enable it before you can use any floating-point instructions. The Example shows an example code sequence for enabling the FPU in both privileged and user modes. The processor must be in privileged mode to read from and write to the CPACR.

Example: Enabling the FPU

```
; CPACR is located at address 0xE000ED88
LDR.W   R0, =0xE000ED88
; Read CPACR
LDR     R1, [R0]
; Set bits 20-23 to enable CP10 and CP11 coprocessors
ORR     R1, R1, #(0xF << 20)
; Write back the modified value to the CPACR
STR     R1, [R0]; wait for store to complete
DSB
;reset pipeline now the FPU is enabled
ISB
```

2.9 PPB Registers

The CPU private peripherals registers base address is $E000E000_H$.

Table 2-19 Registers Overview

Register Short Name	Register Long Name	Offset Address	Access Mode		Description see
			Read	Write	
SCS					
ACTLR	Auxiliary Control Register	008_H	PV, 32	PV, 32	Page 2-57
CPUID	CPUID Base Register	$D00_H$	PV, 32	PV, 32	Page 2-59
ICSR	Interrupt Control and State Register	$D04_H$	PV, 32	PV, 32	Page 2-60
VTOR	Vector Table Offset Register	$D08_H$	PV, 32	PV, 32	Page 2-63
AIRCR	Application Interrupt and Reset Control Register	$D0C_H$	PV, 32	PV, 32	Page 2-63
SCR	System Control Register	$D10_H$	PV, 32	PV, 32	Page 2-66

Table 2-19 Registers Overview (cont'd)

Register Short Name	Register Long Name	Offset Address	Access Mode		Description see
			Read	Write	
CCR	Configuration and Control Register	$D14_H$	PV, 32	PV, 32	Page 2-67
SHPR1	System Handler Priority Register 1	$D18_H$	PV, 32	PV, 32	Page 2-70
SHPR2	System Handler Priority Register 2	$D1C_H$	PV, 32	PV, 32	Page 2-70
SHPR3	System Handler Priority Register 3	$D20_H$	PV, 32	PV, 32	Page 2-71
SHCRS	System Handler Control and State Register	$D24_H$	PV, 32	PV, 32	Page 2-71
CFSR	Configurable Fault Status Register	$D28_H$	PV, 32	PV, 32	Page 2-73
MMSR[1)]	MemManage Fault Status Register	$D28_H$	PV, 32	PV, 32	Page 2-73
BFSR[1)]	BusFault Status Register	$D29_H$	PV, 32	PV, 32	Page 2-73
UFSR[1)]	UsageFault Status Register	$D2A_H$	PV, 32	PV, 32	Page 2-73
HFSR	HardFault Status Register	$D2C_H$	PV, 32	PV, 32	Page 2-80
MMAR	MemManage Fault Address Register	$D34_H$	PV, 32	PV, 32	Page 2-81
BFAR	BusFault Address Register	$D38_H$	PV, 32	PV, 32	Page 2-82
AFSR	Auxiliary Fault Status Register	$D3C_H$	PV, 32	PV, 32	Page 2-82
SysTick					
SYST_CSR	SysTick Control and Status Register	010_H	PV, 32	PV, 32	Page 2-83
SYST_RVR	SysTick Reload Value Register	014_H	PV, 32	PV, 32	Page 2-84
SYST_CVR	SysTick Current Value Register	018_H	PV, 32	PV, 32	Page 2-85

Table 2-19 Registers Overview (cont'd)

Register Short Name	Register Long Name	Offset Address	Access Mode		Description see
			Read	Write	
SYST_CALIB	SysTick Calibration Value Register	$01C_H$	PV, 32	-	Page 2-85
NVIC					
NVIC_ISER0-NVIC_ISER3	Interrupt Set-enable Registers	100_H	PV, 32	PV, 32	Page 2-86
NVIC_ICER0-NVIC_ICER3	Interrupt Clear-enable Registers	180_H	PV, 32	PV, 32	Page 2-87
NVIC_ISPR0-NVIC_ISPR3	Interrupt Set-pending Registers	200_H	PV, 32	PV, 32	Page 2-88
NVIC_ICPR0-NVIC_ICPR3	Interrupt Clear-pending Registers	280_H	PV, 32	PV, 32	Page 2-88
NVIC_IABR0-NVIC_IABR3	Interrupt Active Bit Registers	300_H	PV, 32	PV, 32	Page 2-89
NVIC_IPR0-NVIC_IPR27	Interrupt Priority Registers	400_H	PV, 32	PV, 32	Page 2-89
STIR	Software Trigger Interrupt Register	$F00_H$	Configurable[2)]		Page 2-91
MPU					
MPU_TYPE	MPU Type Register	$D90_H$	PV, 32	PV, 32	Page 2-92
MPU_CTRL	MPU Control Register	$D94_H$	PV, 32	PV, 32	Page 2-92
MPU_RNR	MPU Region Number Register	$D98_H$	PV, 32	PV, 32	Page 2-95
MPU_RBAR	MPU Region Base Address Register	$D9C_H$	PV, 32	PV, 32	Page 2-95
MPU_RASR	MPU Region Attribute and Size Register	$DA0_H$	PV, 32	PV, 32	Page 2-97
MPU_RBAR_A1	Alias of RBAR, see MPU Region Base Address Register	$DA4_H$	PV, 32	PV, 32	Page 2-95
MPU_RASR_A1	Alias of RASR, see MPU Region Attribute and Size Register	$DA8_H$	PV, 32	PV, 32	Page 2-97

Table 2-19 Registers Overview (cont'd)

Register Short Name	Register Long Name	Offset Address	Access Mode		Description see
			Read	Write	
MPU_RBAR_A2	Alias of RBAR, see MPU Region Base Address Register	DAC_H	PV, 32	PV, 32	Page 2-95
MPU_RASR_A2	Alias of RASR, see MPU Region Attribute and Size Register	$DB0_H$	PV, 32	PV, 32	Page 2-97
MPU_RBAR_A3	Alias of RBAR, see MPU Region Base Address Register	$DB4_H$	PV, 32	PV, 32	Page 2-95
MPU_RASR_A3	Alias of RASR, see MPU Region Attribute and Size Register	$DB8_H$	PV, 32	PV, 32	Page 2-97
FPU					
CPACR	Coprocessor Access Control Register	$D88_H$	PV, 32	PV, 32	Page 2-100
FPCCR	Floating-point Context Control Register	$F34_H$	U, PV, 32	U, PV, 32	Page 2-101
FPCAR	Floating-point Context Address Register	$F38_H$	U, PV, 32	U, PV, 32	Page 2-103
FPSCR	Floating-point Status Control Register	-	U, PV, 32	U, PV, 32	Page 2-104
FPDSCR	Floating-point Default Status Control Register	$F3C_H$	U, PV, 32	U, PV, 32	Page 2-106
Reserved	Unused address space	All gaps	nBE	nBE	

1) A subregister of the CFSR.

2) See the register description for more information.

2.9.1 SCS Registers

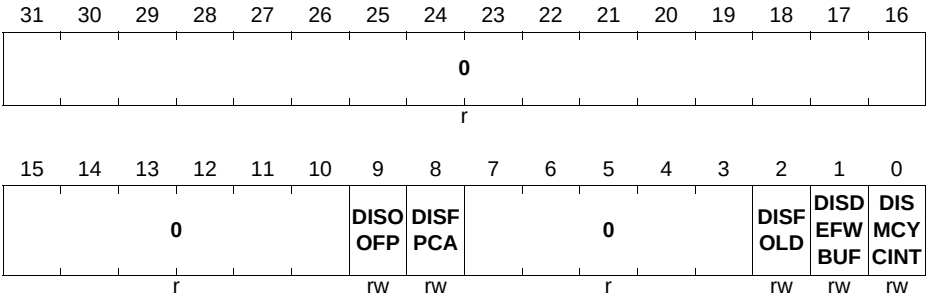
Auxiliary Control Register

The ACTLR provides disable bits for the following processor functions:

- IT folding
- write buffer use for accesses to the default memory map
- interruption of multi-cycle instructions.

By default this register is set to provide optimum performance from the Cortex-M4 processor, and does not normally require modification.

ACTLR
Auxiliary Control Register **(E000 E008$_H$)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
DISMCYCINT	0	rw	**Disable load/store multiple** When set to 1, disables interruption of load multiple and store multiple instructions. This increases the interrupt latency of the processor because any LDM or STM must complete before the processor can stack the current state and enter the interrupt handler.
DISDEFWBUF	1	rw	**Disable write buffer** When set to 1, disables write buffer use during default memory map accesses. This causes all BusFaults to be precise BusFaults but decreases performance because any store to memory must complete before the processor can execute the next instruction. *Note: This bit only affects write buffers implemented in the Cortex-M4 processor.*
DISFOLD	2	rw	**Disable IT folding** When set to 1, disables IT folding.
DISFPCA	8	rw	**Disable FPCA update** Disable automatic update of CONTROL.FPCA.

Field	Bits	Type	Description
DISOOFP	9	rw	**Disable out of order FP execution** Disables floating point instructions completing out of order with respect to integer instructions.
0	[31:10], [7:3]	r	**Reserved** Read as 0; should be written with 0.

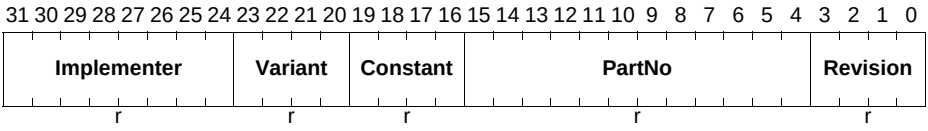
About IT folding

In some situations, the processor can start executing the first instruction in an IT block while it is still executing the IT instruction. This behavior is called IT folding, and improves performance, However, IT folding can cause jitter in looping. If a task must avoid jitter, set the DISFOLD bit to 1 before executing the task, to disable IT folding.

CPUID Base Register

The CPUID register contains the processor part number, version, and implementation information.

CPUID
CPUID Base Register **(E000 ED00$_H$)** **Reset Value: 410F C241$_H$**



Field	Bits	Type	Description
Revision	[3:0]	r	**Revision number** the y value in the “rxpy” product revision identifier 1_H Patch 1
PartNo	[15:4]	r	**Part number of the processor** $C24_H$ Cortex-M4
Constant	[19:16]	r	**Reads as 0xF**
Variant	[23:20]	r	**Variant number** the x value in the “rxpy” product revision identifier 0_H Revision 0
Implementer	[31:24]	r	**Implementer code** 41_H ARM

Interrupt Control and State Register

The ICSR:

- provides:
  - a set-pending bit for the Non-Maskable Interrupt (NMI) exception
  - set-pending and clear-pending bits for the PendSV and SysTick exceptions
- indicates:
  - the exception number of the exception being processed
  - whether there are preempted active exceptions
  - the exception number of the highest priority pending exception
  - whether any interrupts are pending.

ICSR
Interrupt Control and State Register

(E000 ED04$_H$) **Reset Value: 0000 0000$_H$**

31	30:29	28	27	26	25	24	23	22	21:20	19:18	17:16
NMIPENDSET	0	PENDSVSET	PENDSVCLR	PENDSTSET	PENDSTCLR	0	Res	ISRPENDING	0		VECTPENDING
rw	r	rw	w	rw	w	r	r	r	r		r

15:12	11	10	9:8	7:6	5:4	3:2	1:0
VECTPENDING	RETTOBASE	0	VECTACTIVE				
r	r	r	r				

Field	Bits	Type	Description
VECTACTIVE[1)]	[8:0]	r	**Active exception number** 00_H Thread mode Nonzero = The exception number of the currently active exception. *Note: Subtract 16 from this value to obtain the CMSIS IRQ number required to index into the Interrupt Clear-Enable, Set-Enable, Clear-Pending, Set-Pending, or Priority Registers.*

Field	Bits	Type	Description
RETTOBASE	11	r	**Return to Base** Indicates whether there are preempted active exceptions: 0_B there are preempted active exceptions to execute 1_B there are no active exceptions, or the currently-executing exception is the only active exception.
VECTPENDING	[17:12]	r	**Vector Pending** Indicates the exception number of the highest priority pending enabled exception: 0_H no pending exceptions Nonzero = the exception number of the highest priority pending enabled exception. The value indicated by this field includes the effect of the BASEPRI and FAULTMASK registers, but not any effect of the PRIMASK register.
ISRPENDING	22	r	**Interrupt pending flag** excluding NMI and Faults: 0_B interrupt not pending 1_B interrupt pending.
Res	23	r	**Reserved** This bit is reserved for Debug use and reads-as-zero when the processor is not in Debug.
PENDSTCLR	25	w	**SysTick exception clear-pending bit** Write: 0_B no effect 1_B removes the pending state from the SysTick exception. This bit is w. On a register read its value is Unknown.
PENDSTSET	26	rw	**SysTick exception set-pending bit** Write 0_B no effect 1_B changes SysTick exception state to pending. Read: 0_D SysTick exception is not pending 1_D SysTick exception is pending.

Field	Bits	Type	Description
PENDSVCLR	27	w	**PendSV clear-pending bit** Write: 0_B no effect 1_B removes the pending state from the PendSV exception.
PENDSVSET	28	rw	**PendSV set-pending bit** Write: 0_B no effect 1_B changes PendSV exception state to pending. Read: 0_B PendSV exception is not pending 1_B PendSV exception is pending. Writing 1 to this bit is the only way to set the PendSV exception state to pending.
NMIPENDSET	31	rw	**NMI set-pending bit** Write: 0_B no effect 1_B changes NMI exception state to pending. Read: 0_B NMI exception is not pending 1_B NMI exception is pending. Because NMI is the highest-priority exception, normally the processor enter the NMI exception handler as soon as it registers a write of 1 to this bit, and entering the handler clears this bit to 0. A read of this bit by the NMI exception handler returns 1 only if the NMI signal is reasserted while the processor is executing that handler.
0	[30:29], 24, [21:18], [10:9]	r	**Reserved** Read as 0; should be written with 0.

1) This is the same value as IPSR bits[8:0], see Interrupt Program Status Register on page 2-6.

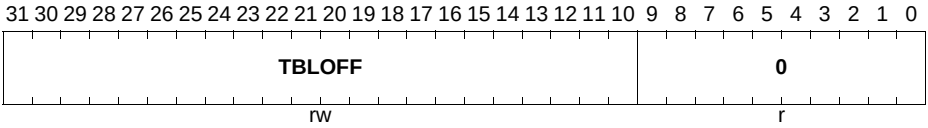
When you write to the ICSR, the effect is Unpredictable if you:

Note: write 1 to the PENDSVSET bit and write 1 to the PENDSVCLR bit write 1 to the PENDSTSET bit and write 1 to the PENDSTCLR bit.

Vector Table Offset Register

The VTOR indicates the offset of the vector table base address from memory address 0x00000000.

VTOR
Vector Table Offset Register (E000 ED08$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
TBLOFF	[31:10]	rw	**Vector table base offset field** It contains bits[29:10] of the offset of the table base from the bottom of the memory map. *Note: Bit[29] determines whether the vector table is in the code or SRAM memory region:* *0 = code* *1 = SRAM* *Bit[29] is sometimes called the TBLBASE bit.*
0	[9:0]	r	**Reserved** Read as 0; should be written with 0.

When setting TBLOFF, you must align the offset to the number of exception entries in the vector table. The XMC4500 provides 112 interrupt nodes - minimum alignment is therefore 256 words, enough for up to 128 interrupts.

Notes

1. *XMC4500 implements 112 interrupts, the remaining nodes to 128 are not used.*
2. *Table alignment requirements mean that bits[9:0] of the table offset must always be zero.*

Application Interrupt and Reset Control Register

The AIRCR provides priority grouping control for the exception model, endian status for data accesses, and reset control of the system.

To write to this register, you must write 0x5FA to the VECTKEY field, otherwise the processor ignores the write.

AIRCR
Application Interrupt and Reset Control Register
(E000 ED0C$_H$) **Reset Value: FA05 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
VECTKEY															
rw															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ENDIANNESS	0				PRIGROUP			0					SYSRESETREQ	VECTCLRACTIVE	VECTRESET
r	r				rw			r					w	w	w

Field	Bits	Type	Description
VECTRESET	0	w	**Reserved for Debug use.** This bit reads as 0. When writing to the register you must write 0 to this bit, otherwise behavior is Unpredictable
VECTCLRACTIVE	1	w	**Reserved for Debug use.** This bit reads as 0. When writing to the register you must write 0 to this bit, otherwise behavior is Unpredictable.
SYSRESETREQ	2	w	**System reset request** 0_B no system reset request 1_B asserts a signal to the outer system that requests a reset. This is intended to force a large system reset of all major components except for debug. This bit reads as 0.
PRIGROUP	[10:8]	rw	**Interrupt priority grouping field** This field determines the split of group priority from subpriority, see Binary point on **Page 2-65**.
ENDIANNESS	15	r	**Data endianness bit** 0_B Little-endian 1_B Big-endian.

Field	Bits	Type	Description
VECTKEY	[31:16]	rw	**Register key** Read: = VECTKEY, reads as 0xFA05 Write: = VECTKEYSTAT, On writes, write 0x5FA to VECTKEY, otherwise the write is ignored.
0	[14:11], [7:3]	r	**Reserved** Read as 0; should be written with 0.

Binary point

The PRIGROUP field indicates the position of the binary point that splits the PRI_n fields in the Interrupt Priority Registers into separate group priority and subpriority fields. **Table 2-20** shows how the PRIGROUP value controls this split.

Table 2-20 Priority grouping

Interrupt priority level value, PRI_N[7:0]				Number of	
PRIGROUP	**Binary point[1)]**	**Group priority bits**	**Subpriority bits**	**Group priorities**	**Sub-priorities**
0b001	bxxxxxx.00	[7:2]	None	64	1
0b010	bxxxxx.y00	[7:3]	[2]	32	2
0b011	bxxxx.yy00	[7:4]	[3:2]	16	4
0b100	bxxx.yyy00	[7:5]	[4:2]	8	8
0b101	bxx.yyyy00	[7:6]	[5:2]	4	16
0b110	bx.yyyyy00	7	[6:2]	2	32
0b111	b.yyyyyy00	None	[7:2]	1	64

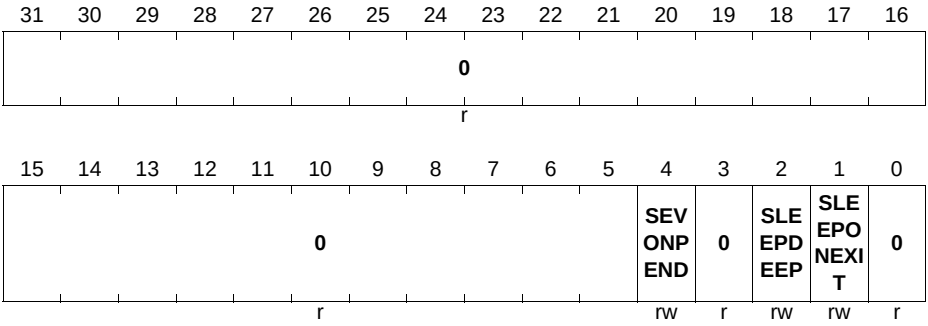
1) PRI_n[7:0] field showing the binary point. x denotes a group priority field bit, and y denotes a subpriority field bit.

*Note: Determining preemption of an exception uses only the group priority field, see Interrupt Priority Grouping on **Page 2-31**.*

System Control Register

The SCR controls features of entry to and exit from low power state.

SCR
System Control Register (E000 ED10$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
SLEEPONEXIT	1	rw	**Sleep on Exit** Indicates sleep-on-exit when returning from Handler mode to Thread mode: 0_B sleep 1_B deep sleep Setting this bit to 1 enables an interrupt driven application to avoid returning to an empty main application.
SLEEPDEEP	2	rw	**Sleep or Deep Sleep** Controls whether the processor uses sleep or deep sleep as its low power mode: 0_B sleep 1_B deep sleep

Field	Bits	Type	Description
SEVONPEND	4	rw	**Send Event on Pending bit:** 0_B only enabled interrupts or events can wakeup the processor, disabled interrupts are excluded 1_B enabled events and all interrupts, including disabled interrupts, can wakeup the processor. When an event or interrupt enters pending state, the event signal wakes up the processor from WFE. If the processor is not waiting for an event, the event is registered and affects the next WFE. The processor also wakes up on execution of an SEV instruction or an external event.
0	[31:5], 3, 0	r	**Reserved** Read as 0; should be written with 0.

Configuration and Control Register

The CCR controls entry to Thread mode and enables:

- the handlers for NMI, hard fault and faults escalated by FAULTMASK to ignore BusFaults
- trapping of divide by zero and unaligned accesses
- access to the STIR by unprivileged software, see **Software Trigger Interrupt Register** on **Page 2-91**

CCR
Configuration and Control Register

(E000 ED14$_H$) **Reset Value: 0000 0200$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						STKALIGN	BFHFNMIGN	0			DIV_0_TRP	UNALIGN_TRP	0	USERSETMPEND	NONBASETHRDENA
r						rw	rw	r			rw	rw	r	rw	rw

Field	Bits	Type	Description
NONBASETHRDENA	0	rw	**Non Base Thread Mode Enable** Indicates how the processor enters Thread mode: 0_B processor can enter Thread mode only when no exception is active. 1_B processor can enter Thread mode from any level under the control of an EXC_RETURN value, see **Exception return**.
USERSETMPEND	1	rw	**User Set Pending Enable** Enables unprivileged software access to the STIR, see **Software Trigger Interrupt Register**. 0_B disable 1_B enable
UNALIGN_TRP	3	rw	**Unaligned Access Trap Enable** Enables unaligned access traps: 0_B do not trap unaligned halfword and word accesses 1_B trap unaligned halfword and word accesses. If this bit is set to 1, an unaligned access generates a UsageFault. Unaligned LDM, STM, LDRD, and STRD instructions always fault irrespective of whether UNALIGN_TRP is set to 1.
DIV_0_TRP	4	rw	**Divide by Zero Trap Enable** Enables faulting or halting when the processor executes an SDIV or UDIV instruction with a divisor of 0: 0_B do not trap divide by 0 1_B trap divide by 0. When this bit is set to 0,a divide by zero returns a quotient of 0.

Field	Bits	Type	Description
BFHFNMIGN	8	rw	**Bus Fault Hard Fault and NMI Ignore** Enables handlers with priority -1 or -2 to ignore data BusFaults caused by load and store instructions. This applies to the hard fault, NMI, and FAULTMASK escalated handlers: 0_B data bus faults caused by load and store instructions cause a lock-up 1_B handlers running at priority -1 and -2 ignore data bus faults caused by load and store instructions. Set this bit to 1 only when the handler and its data are in absolutely safe memory. The normal use of this bit is to probe system devices and bridges to detect control path problems and fix them.
STKALIGN	9	rw	**Stack Alignment** Indicates stack alignment on exception entry: 0_B 4-byte aligned 1_B 8-byte aligned. On exception entry, the processor uses bit[9] of the stacked PSR to indicate the stack alignment. On return from the exception it uses this stacked bit to restore the correct stack alignment.
0	[31:10], [7:5], 2	r	**Reserved** Read as 0; should be written with 0.

System Handler Priority Registers

The SHPR1-SHPR3 registers set the priority level, 0 to 63 of the exception handlers that have configurable priority.

SHPR1-SHPR3 are byte accessible.

The system fault handlers and the priority field and register for each handler are:

Table 2-21 System fault handler priority fields

Handler	Field	Register description
MemManage	PRI_4	System Handler Priority Register 1 on **Page 2-70**
BusFault	PRI_5	
UsageFault	PRI_6	
SVCall	PRI_11	System Handler Priority Register 2 on **Page 2-70**

Table 2-21 System fault handler priority fields (cont'd)

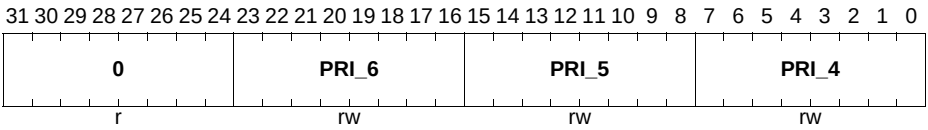
Handler	Field	Register description
PendSV	PRI_14	System Handler Priority Register 3 on **Page 2-71**
SysTick	PRI_15	

Each PRI_N field is 8 bits wide, but the XMC4500 implements only bits[7:2] of each field, and bits[1:0] read as zero and ignore writes.

System Handler Priority Register 1

SHPR1
System Handler Priority Register 1

(E000 ED18$_H$) **Reset Value: 0000 0000$_H$**

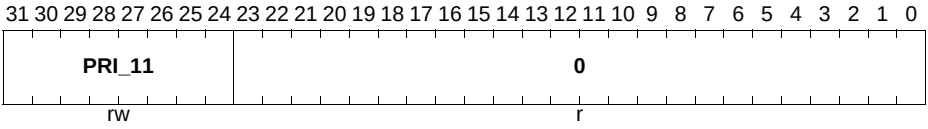


Field	Bits	Type	Description
PRI_4	[7:0]	rw	**Priority of system handler 4, MemManage**
PRI_5	[15:8]	rw	**Priority of system handler 5, BusFault**
PRI_6	[23:16]	rw	**Priority of system handler 6, UsageFault**
0	[31:24]	r	**Reserved** Read as 0; should be written with 0.

System Handler Priority Register 2

SHPR2
System Handler Priority Register 2

(E000 ED1C$_H$) **Reset Value: 0000 0000$_H$**

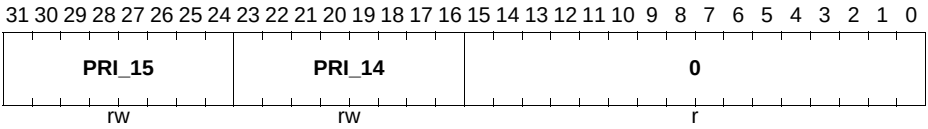


Field	Bits	Type	Description
PRI_11	[31:24]	rw	**Priority of system handler 11, SVCall**
0	[23:0]	r	**Reserved** Read as 0; should be written with 0.

System Handler Priority Register 3

SHPR3
System Handler Priority Register 3

(E000 ED20$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
PRI_14	[23:16]	rw	**Priority of system handler 14** PendSV
PRI_15	[31:24]	rw	**Priority of system handler 15** SysTick exception
0	[15:0]	r	**Reserved** Read as 0; should be written with 0.

System Handler Control and State Register

The SHCSR enables the system handlers, and indicates:

- the pending status of the BusFault, MemManage fault, and SVC exceptions
- the active status of the system handlers.

SHCSR
System Handler Control and State Register

(E000 ED24$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0													USGFAULTENA	BUSFAULTENA	MEMFAULTENA
r													rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SVCALLPENDED	BUSFAULTPENDED	MEMFAULTPENDED	USGFAULTPENDED	SYSTICKACT	PENDSVACT	0	MONITORACT	SVCALLACT	0			USGFAULTACT	0	BUSFAULTACT	MEMFAULTACT
rw	rw	rw	rw	rw	rw	r	rw	rw	r			rw	r	rw	rw

Field	Bits	Type	Description
MEMFAULTACT	0	rw	**MemManage exception active bit** Reads as 1 if exception is active.
BUSFAULTACT	1	rw	**BusFault exception active bit** Reads as 1 if exception is active.
USGFAULTACT	3	rw	**UsageFault exception active bit** Reads as 1 if exception is active.
SVCALLACT	7	rw	**SVCall active bit** Reads as 1 if SVC call is active.
MONITORACT	8	rw	**Debug monitor active bit** Reads as 1 if Debug monitor is active.
PENDSVACT	10	rw	**PendSV exception active bit** Reads as 1 if exception is active.
SYSTICKACT	11	rw	**SysTick exception active bit** Reads as 1 if exception is active.
USGFAULTPENDED	12	rw	**UsageFault exception pending bit** Reads as 1 if exception is pending.
MEMFAULTPENDED	13	rw	**MemManage exception pending bit** Reads as 1 if exception is pending.

Field	Bits	Type	Description
BUSFAULTPENDED	14	rw	**BusFault exception pending bit** Reads as 1 if exception is pending.
SVCALLPENDED	15	rw	**SVCall pending bit** Reads as 1 if exception is pending.
MEMFAULTENA	16	rw	**MemManage enable bit** Set to 1 to enable.
BUSFAULTENA	17	rw	**BusFault enable bit** Set to 1 to enable.
USGFAULTENA	18	rw	**UsageFault enable bit** Set to 1 to enable.
0	[31:19], 9, [6:4], 2	r	**Reserved** Read as 0; should be written with 0.

Notes

1. *Active bits, read as 1 if the exception is active, or as 0 if it is not active. You can write to these bits to change the active status of the exceptions, but see the Caution in this section.*
2. *Pending bits, read as 1 if the exception is pending, or as 0 if it is not pending. You can write to these bits to change the pending status of the exceptions.*
3. *Enable bits, set to 1 to enable the exception, or set to 0 to disable the exception.*

If you disable a system handler and the corresponding fault occurs, the processor treats the fault as a hard fault.

You can write to this register to change the pending or active status of system exceptions. An OS kernel can write to the active bits to perform a context switch that changes the current exception type.

Note: Software that changes the value of an active bit in this register without correct adjustment to the stacked content can cause the processor to generate a fault exception. Ensure software that writes to this register retains and subsequently restores the current active status.

Note: After you have enabled the system handlers, if you have to change the value of a bit in this register you must use a read-modify-write procedure to ensure that you change only the required bit.

Configurable Fault Status Register

The CFSR indicates the cause of a MemManage fault, BusFault, or UsageFault.

The flags in the MMFSR indicate the cause of memory access faults.

The flags in the BFSR indicate the cause of a bus access fault.

The UFSR indicates the cause of a UsageFault.

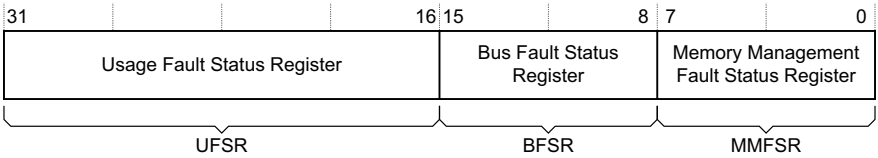


Figure 2-9 CFSR

The CFSR is byte accessible. You can access the CFSR or its subregisters as follows:

- access the complete CFSR with a word access to 0xE000ED28
- access the MMFSR with a byte access to 0xE000ED28
- access the MMFSR with a byte access to 0xE000ED28
- access the MMFSR and BFSR with a halfword access to 0xE000ED28
- access the BFSR with a byte access to 0xE000ED29
- access the UFSR with a halfword access to 0xE000ED2A

Note: The UFSR bits are sticky. This means as one or more fault occurs, the associated bits are set to 1. A bit that is set to 1 is cleared to 0 only by writing 1 to that bit, or by a reset.

CFSR
Configurable Fault Status Register

(E000 ED28$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0						DIVBYZERO	UNALIGNED	0				NOCP	INVPC	INVSTATE	UNDEFINSTR
r						rw	rw	r				rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BFARVALID	0	LSPERR	STKERR	UNSTKERR	IMPRECISERR	PRECISERR	IBUSERR	MMARVALID	0	MLSPERR	MSTKERR	MUNSTKERR	0	DACCVIOL	IACCVIOL
rw	r	rw	rw	rw	rw	rw	rw	rw	r	rw	rw	rw	r	rw	rw

Field	Bits	Type	Description
IACCVIOL	0	rw	**Instruction access violation flag** 0_B no instruction access violation fault 1_B the processor attempted an instruction fetch from a location that does not permit execution. This fault occurs on any access to an XN region, even when the MPU is disabled or not present. When this bit is 1, the PC value stacked for the exception return points to the faulting instruction. The processor has not written a fault address to the MMAR.
DACCVIOL	1	rw	**Data access violation flag** 0_B no data access violation fault 1_B the processor attempted a load or store at a location that does not permit the operation. When this bit is 1, the PC value stacked for the exception return points to the faulting instruction. The processor has loaded the MMAR with the address of the attempted access.
MUNSTKERR	3	rw	**MemManage fault on unstacking for a return from exception** 0_B no unstacking fault 1_B unstack for an exception return has caused one or more access violations. This fault is chained to the handler. This means that when this bit is 1, the original return stack is still present. The processor has not adjusted the SP from the failing return, and has not performed a new save. The processor has not written a fault address to the MMAR.
MSTKERR	4	rw	**MemManage fault on stacking for exception entry** 0_B no stacking fault 1_B stacking for an exception entry has caused one or more access violations. When this bit is 1, the SP is still adjusted but the values in the context area on the stack might be incorrect. The processor has not written a fault address to the MMAR.

Field	Bits	Type	Description
MLSPERR	5	rw	**MemManage fault during floating point lazy state preservation** 0_B No MemManage fault occurred during floating-point lazy state preservation 1_B A MemManage fault occurred during floating-point lazy state preservation
MMARVALID	7	rw	**MemManage Fault Address Register (MMFAR) valid flag** 0_B value in MMAR is not a valid fault address 1_B MMAR holds a valid fault address. If a MemManage fault occurs and is escalated to a HardFault because of priority, the HardFault handler must set this bit to 0. This prevents problems on return to a stacked active MemManage fault handler whose MMAR value has been overwritten.
IBUSERR	8	rw	**Instruction bus error** 0_B no instruction bus error 1_B instruction bus error. The processor detects the instruction bus error on prefetching an instruction, but it sets the IBUSERR flag to 1 only if it attempts to issue the faulting instruction. When the processor sets this bit is 1, it does not write a fault address to the BFAR.
PRECISERR	9	rw	**Precise data bus error** 0_B no precise data bus error 1_B a data bus error has occurred, and the PC value stacked for the exception return points to the instruction that caused the fault. When the processor sets this bit is 1, it writes the faulting address to the BFAR.

Field	Bits	Type	Description
IMPRECISERR	10	rw	**Imprecise data bus error** 0_B no imprecise data bus error 1_B a data bus error has occurred, but the return address in the stack frame is not related to the instruction that caused the error. When the processor sets this bit to 1, it does not write a fault address to the BFAR. This is an asynchronous fault. Therefore, if it is detected when the priority of the current process is higher than the BusFault priority, the BusFault becomes pending and becomes active only when the processor returns from all higher priority processes. If a precise fault occurs before the processor enters the handler for the imprecise BusFault, the handler detects both IMPRECISERR set to 1 and one of the precise fault status bits set to 1.
UNSTKERR	11	rw	**BusFault on unstacking for a return from exception** 0_B no unstacking fault 1_B stacking for an exception entry has caused one or more BusFaults. This fault is chained to the handler. This means that when the processor sets this bit to 1, the original return stack is still present. The processor does not adjust the SP from the failing return, does not performed a new save, and does not write a fault address to the BFAR.
STKERR	12	rw	**BusFault on stacking for exception entry** 0_B no stacking fault 1_B stacking for an exception entry has caused one or more BusFaults. When the processor sets this bit to 1, the SP is still adjusted but the values in the context area on the stack might be incorrect. The processor does not write a fault address to the BFAR.

Field	Bits	Type	Description
LSPERR	13	rw	**BusFault during floating point lazy state preservation** 0_B No bus fault occurred during floating-point lazy state preservation. 1_B A bus fault occurred during floating-point lazy state preservation
BFARVALID	15	rw	**BusFault Address Register (BFAR) valid flag** 0_B value in BFAR is not a valid fault address 1_B BFAR holds a valid fault address. The processor sets this bit to 1 after a BusFault where the address is known. Other faults can set this bit to 0, such as a MemManage fault occurring later. If a BusFault occurs and is escalated to a hard fault because of priority, the hard fault handler must set this bit to 0. This prevents problems if returning to a stacked active BusFault handler whose BFAR value has been overwritten.
UNDEFINSTR	16	rw	**Undefined instruction UsageFault** 0_B no undefined instruction UsageFault 1_B the processor has attempted to execute an undefined instruction. When this bit is set to 1, the PC value stacked for the exception return points to the undefined instruction. An undefined instruction is an instruction that the processor cannot decode.
INVSTATE	17	rw	**Invalid state UsageFault** 0_B no invalid state UsageFault 1_B the processor has attempted to execute an instruction that makes illegal use of the EPSR. When this bit is set to 1, the PC value stacked for the exception return points to the instruction that attempted the illegal use of the EPSR. This bit is not set to 1 if an undefined instruction uses the EPSR.

Field	Bits	Type	Description
INVPC	18	rw	**Invalid PC load UsageFault** caused by an invalid PC load by EXC_RETURN: 0_B no invalid PC load UsageFault 1_B the processor has attempted an illegal load of EXC_RETURN to the PC, as a result of an invalid context, or an invalid EXC_RETURN value. When this bit is set to 1, the PC value stacked for the exception return points to the instruction that tried to perform the illegal load of the PC.
NOCP	19	rw	**No coprocessor UsageFault** 0_B no UsageFault caused by attempting to access a coprocessor 1_B the processor has attempted to access a coprocessor.
UNALIGNED	24	rw	**Unaligned access UsageFault** 0_B no unaligned access fault, or unaligned access trapping not enabled 1_B the processor has made an unaligned memory access. Enable trapping of unaligned accesses by setting the UNALIGN_TRP bit in the CCR to 1, see Configuration and Control Register on **Page 2-67**. Unaligned LDM, STM, LDRD, and STRD instructions always fault irrespective of the setting of UNALIGN_TRP.
DIVBYZERO	25	rw	**Divide by zero UsageFault** 0_B no divide by zero fault, or divide by zero trapping not enabled 1_B the processor has executed an SDIV or UDIV instruction with a divisor of 0 When the processor sets this bit to 1, the PC value stacked for the exception return points to the instruction that performed the divide by zero. Enable trapping of divide by zero by setting the DIV_0_TRP bit in the CCR to 1, see Configuration and Control Register on **Page 2-67**.
0	[31:26], [23:20], 14, 6, 2	r	**Reserved** Read as 0; should be written with 0.

HardFault Status Register

The HFSR gives information about events that activate the HardFault handler.

This register is read, write to clear. This means that bits in the register read normally, but writing 1 to any bit clears that bit to 0. The bit assignments are:

HFSR
HardFault Status Register **(E000 ED2C$_H$)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DEBUGEVT	FORCED	0													
rw	rw	r													

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0														VECTTBL	0
r														rw	r

Field	Bits	Type	Description
VECTTBL	1	rw	**BusFault on vector table read** Indicates a BusFault on a vector table read during exception processing: 0_B no BusFault on vector table read 1_B BusFault on vector table read This error is always handled by the hard fault handler. When this bit is set to 1, the PC value stacked for the exception return points to the instruction that was preempted by the exception.
FORCED	30	rw	**Forced HardFault** Indicates a forced hard fault, generated by escalation of a fault with configurable priority that cannot be handles, either because of priority or because it is disabled: 0_B no forced HardFault 1_B forced HardFault. When this bit is set to 1, the HardFault handler must read the other fault status registers to find the cause of the fault.

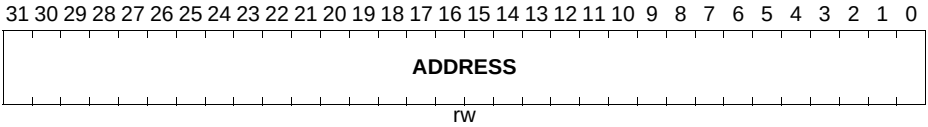
Field	Bits	Type	Description
DEBUGEVT	31	rw	**Reserved for Debug use** When writing to the register you must write 0 to this bit, otherwise behavior is Unpredictable
0	[29:2], 0	r	**Reserved** Read as 0; should be written with 0.

Note: The HFSR bits are sticky. This means as one or more fault occurs, the associated bits are set to 1. A bit that is set to 1 is cleared to 0 only by writing 1 to that bit, or by a reset.

MemManage Fault Address Register

The MMFAR contains the address of the location that generated a MemManage fault.

MMFAR
MemManage Fault Address Register
(E000 ED34$_H$) **Reset Value: XXXX XXXX$_H$**



Field	Bits	Type	Description
ADDRESS	[31:0]	rw	**Address causing the fault** When the MMARVALID bit of the MMFSR is set to 1, this field holds the address of the location that generated the MemManage fault

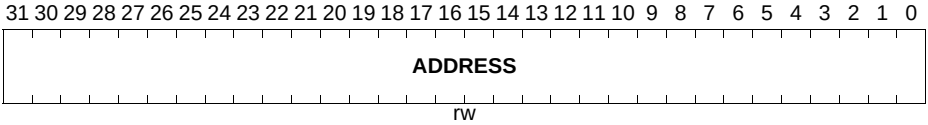
When an unaligned access faults, the address is the actual address that faulted. Because a single read or write instruction can be split into multiple aligned accesses, the fault address can be any address in the range of the requested access size.

Flags in the MMFSR indicate the cause of the fault, and whether the value in the MMFAR is valid. See MemManage Fault Status Register on **Page 2-73**.

BusFault Address Register

The BFAR contains the address of the location that generated a BusFault.

BFAR
BusFault Address Register (E000 ED38$_H$) Reset Value: XXXX XXXX$_H$



Field	Bits	Type	Description
ADDRESS	[31:0]	rw	**Address causing the fault** When the BFARVALID bit of the BFSR is set to 1, this field holds the address of the location that generated the BusFault

When an unaligned access faults the address in the BFAR is the one requested by the instruction, even if it is not the address of the fault.

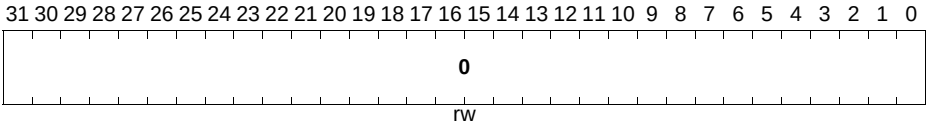
Flags in the BFSR indicate the cause of the fault, and whether the value in the BFAR is valid. See BusFault Status Register on **Page 2-73**.

Auxiliary Fault Status Register

The AFSR contains additional system fault information.

This register is read, write to clear. This means that bits in the register read normally, but writing 1 to any bit clears that bit to 0.

AFSR
Auxiliary Fault Status Register (E000 ED3C$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
0	[31:0]	rw	**Reserved** Read as 0; should be written with 0.

Each AFSR bit maps directly to an AUXFAULT input of the processor, and a single-cycle HIGH signal on the input sets the corresponding AFSR bit to one. It remains set to 1 until you write 1 to the bit to clear it to zero.

When an AFSR bit is latched as one, an exception does not occur. Use an interrupt if an exception is required.

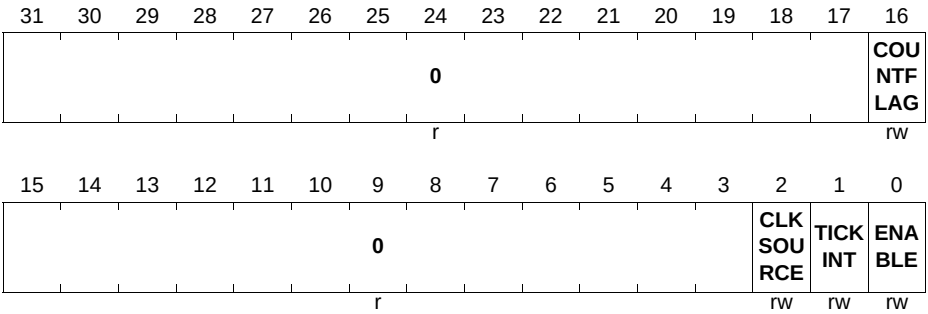
2.9.2 SysTick Registers

SysTick Control and Status Register

The SysTick SYST_CSR register enables the SysTick features.

SYST_CSR
SysTick Control and Status Register

(E000 E010$_H$) **Reset Value: 0000 0004$_H$**



Field	Bits	Type	Description
ENABLE	0	rw	**Enable** Enables the counter: 0_B counter disabled 1_B counter enabled.
TICKINT	1	rw	**Tick Interrupt Enable** Enables SysTick exception request: 0_B counting down to zero does not assert the SysTick exception request 1_B counting down to zero to asserts the SysTick exception request. Software can use COUNTFLAG to determine if SysTick has ever counted to zero.

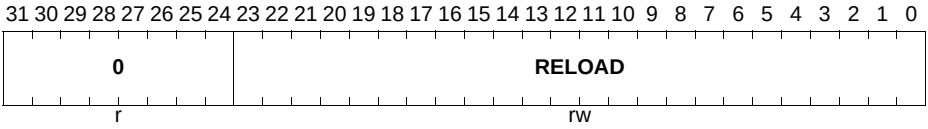
Field	Bits	Type	Description
CLKSOURCE	2	rw	**Clock source** 0_B f_{STDBY} / 2 1_B f_{CPU}
COUNTFLAG	16	rw	**Counter Flag** Returns 1 if timer counted to 0 since last time this was read.
0	[31:17], [15:3]	r	**Reserved** Read as 0; should be written with 0.

When ENABLE is set to 1, the counter loads the RELOAD value from the SYST_RVR register and then counts down. On reaching 0, it sets the COUNTFLAG to 1 and optionally asserts the SysTick depending on the value of TICKINT. It then loads the RELOAD value again, and begins counting.

SysTick Reload Value Register

The SYST_RVR register specifies the start value to load into the SYST_CVR register.

SYST_RVR
SysTick Reload Value Register (E000 E014$_H$) **Reset Value: XXXX XXXX$_H$**



Field	Bits	Type	Description
RELOAD	[23:0]	rw	**Reload Value** Value to load into the SYST_CVR register when the counter is enabled and when it reaches 0, see Calculating the RELOAD value.
0	[31:24]	r	**Reserved** Read as 0; should be written with 0.

Notes on calculating the RELOAD value

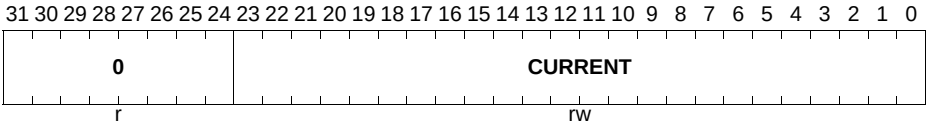
1. *The RELOAD value can be any value in the range 0x00000001-0x00FFFFFF. A start value of 0 is possible, but has no effect because the SysTick exception request and COUNTFLAG are activated when counting from 1 to 0.*

2. *The RELOAD value is calculated according to its use. For example, to generate a multi-shot timer with a period of N processor clock cycles, use a RELOAD value of N-1. If the SysTick interrupt is required every 100 clock pulses, set RELOAD to 99.*

SysTick Current Value Register

The SYST_CVR register contains the current value of the SysTick counter.

SYST_CVR
SysTick Current Value Register (E000 E018$_H$) Reset Value: XXXX XXXX$_H$

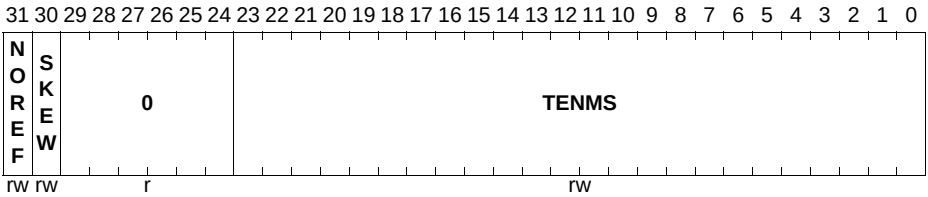


Field	Bits	Type	Description
CURRENT	[23:0]	rw	**Current Value** Reads return the current value of the SysTick counter. A write of any value clears the field to 0, and also clears the SYST_CSR COUNTFLAG bit to 0.
0	[31:24]	r	**Reserved** Read as 0; should be written with 0.

SysTick Calibration Value Register

The SYST_CALIB register indicates the SysTick calibration properties.

SYST_CALIB
SysTick Calibration Value Register
r (E000 E01C$_H$) Reset Value: C000 0000$_H$



Field	Bits	Type	Description
TENMS	[23:0]	rw	**Ten Milliseconds Reload Value** Reload value for 10ms (100Hz) timing, subject to system clock skew errors. If the value reads as zero, the calibration value is not known.
SKEW	30	rw	**Ten Milliseconds Skewed** Indicates whether the TENMS value is exact: 0_B TENMS value is exact 1_B TENMS value is inexact, or not given. An inexact TENMS value can affect the suitability of SysTick as a software real time clock.
NOREF	31	rw	**No Reference Clock** Indicates whether the device provides a reference clock to the processor: 0_B reference clock provided 1_B no reference clock provided. If your device does not provide a reference clock, the SYST_CSR.CLKSOURCE bit reads-as-one and ignores writes.
0	[29:24]	r	**Reserved** Read as 0; should be written with 0.

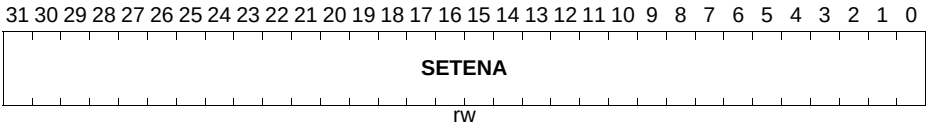
2.9.3 NVIC Registers

Interrupt Set-enable Registers

The NVIC_ISERx (x=0-3) registers enable interrupts, and show which interrupts are enabled.

NVIC_ISERx (x=0-3)
Interrupt Set-enable Register x

(E000 E100$_H$ + 4*x) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
SETENA	[31:0]	rw	**Interrupt set-enable bits** Write: 0_B no effect 1_B enable interrupt. Read: 0_B interrupt disabled 1_B interrupt enabled.

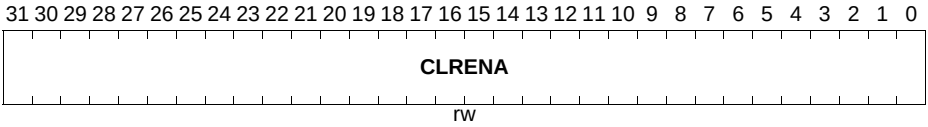
If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority. If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, but the NVIC never activates the interrupt, regardless of its priority.

Interrupt Clear-enable Registers

The NVIC_ICERx (x=0-7) registers disable interrupts, and show which interrupts are enabled.

NVIC_ISCERx (x=0-3)
Interrupt Clear-enable Register x

(E000 E180$_H$ + 4*x) **Reset Value: 0000 0000$_H$**



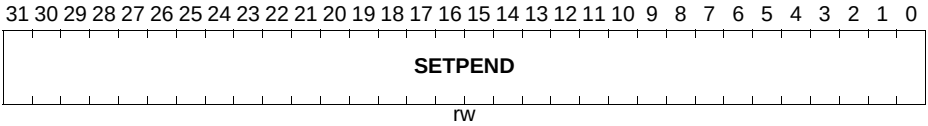
Field	Bits	Type	Description
CLRENA	[31:0]	rw	**Interrupt clear-enable bits.** Write: 0_B no effect 1_B enable interrupt. Read: 0_B interrupt disabled 1_B interrupt enabled.

Interrupt Set-pending Registers

The NVIC_ISPRx (x=0-7) registers force interrupts into the pending state, and show which interrupts are pending.

NVIC_ISSPRx (x=0-3)
Interrupt Set-pending Register x

(E000 E200$_H$ + 4*x) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
SETPEND	[31:0]	rw	**Interrupt set-pending bits.** Write: 0_B no effect 1_B changes interrupt state to pending. Read: 0_B interrupt is not pending 1_B interrupt is pending.

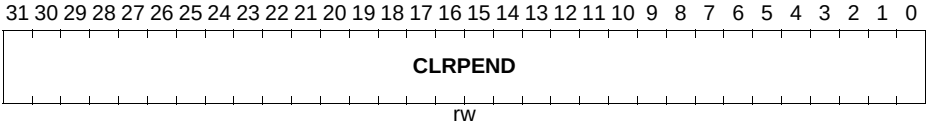
Writing 1 to the ISPR bit corresponding to:
- an interrupt that is pending has no effect
- a disabled interrupt sets the state of that interrupt to pending

Interrupt Clear-pending Registers

The NVIC_ICPRx (x=0-7) registers remove the pending state from interrupts, and show which interrupts are pending.

NVIC_ICPRx (x=0-3)
Interrupt Clear-pending Register x

(E000 E280$_H$ + 4*x) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
CLRPEND	[31:0]	rw	**Interrupt set-pending bits.** Write: 0_B no effect 1_B removes pending state an interrupt. Read: 0_B interrupt is not pending 1_B interrupt is pending.

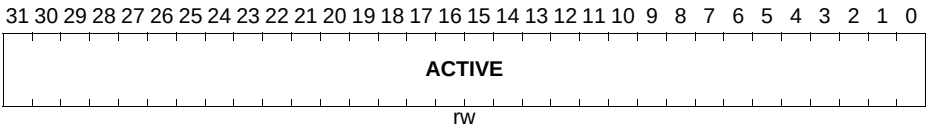
Note: Writing 1 to an ICPR bit does not affect the active state of the corresponding interrupt.

Interrupt Active Bit Registers

The NVIC_IABRx (x=0-7) registers indicate which interrupts are active.

NVIC_IABRx (x=0-3)
Interrupt Active Bit Register x

(E000 E300$_H$ + 4*x) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
ACTIVE	[31:0]	rw	**Interrupt active flags:** 0_B interrupt not active 1_B interrupt active

A bit reads as one if the status of the corresponding interrupt is active or active and pending.

Interrupt Priority Registers

The NVIC_IPRx (x=0-27) registers provide an 8-bit priority field for each interrupt. These registers are byte-accessible. Each register holds four priority fields as shown:

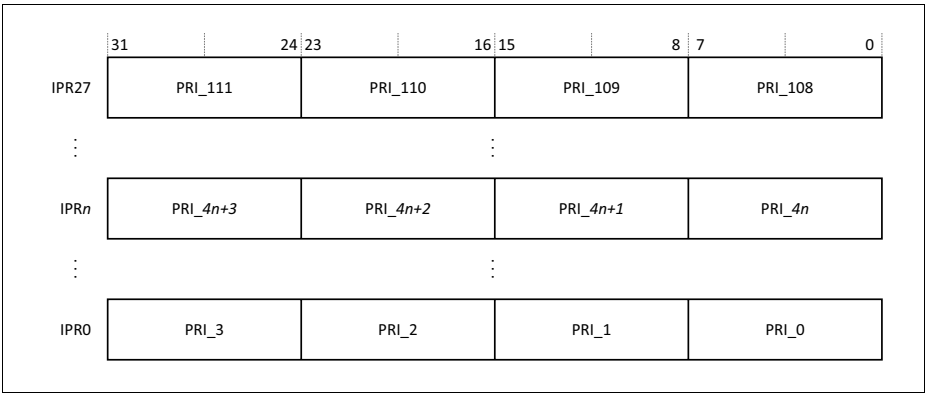


Figure 2-10 Interrupt Priority Register

NVIC_IPRx (x=0-27)
Interrupt Priority Register x

(E000 E400$_H$ + 4*x) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PRI_3								PRI_2								PRI_1								PRI_0							
rw								rw								rw								rw							

Field	Bits	Type	Description
PRI_0	[7:0]	rw	**Priority value 0**
PRI_1	[15:8]	rw	**Priority value 1**
PRI_2	[23:16]	rw	**Priority value 2**
PRI_3	[31:24]	rw	**Priority value 3** The lower the value, the greater the priority of the corresponding interrupt. The processor implements only bits[7:n] of each field, bits[n-1:0] read as zero and ignore writes.

See “Using CMSIS functions to access NVIC” on **Page 2-45** for more information about the access to the interrupt priority array, which provides the software view of the interrupt priorities.

Find the IPR number and byte offset for interrupt m as follows:

- the corresponding IPR number n, see **Figure 2-10** on **Page 2-907**, is given by
  n = m DIV 4
- the byte offset of the required Priority field in this register is m MOD 4, where:
  – byte offset 0 refers to register bits[7:0]
  – byte offset 1 refers to register bits[15:8]
  – byte offset 2 refers to register bits[23:16]
  – byte offset 3 refers to register bits[31:24].

Software Trigger Interrupt Register

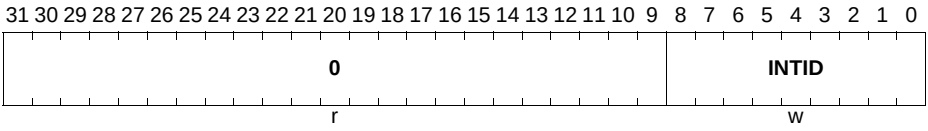
Write to the STIR to generate an interrupt from software.

When the USERSETMPEND bit in the SCR is set to 1, unprivileged software can access the STIR, see System Control Register on **Page 2-66**.

Note: Only privileged software can enable unprivileged access to the STIR.

STIR
Software Trigger Interrupt Register

(E000 EF00$_H$) **Reset Value: 0000 0000$_H$**



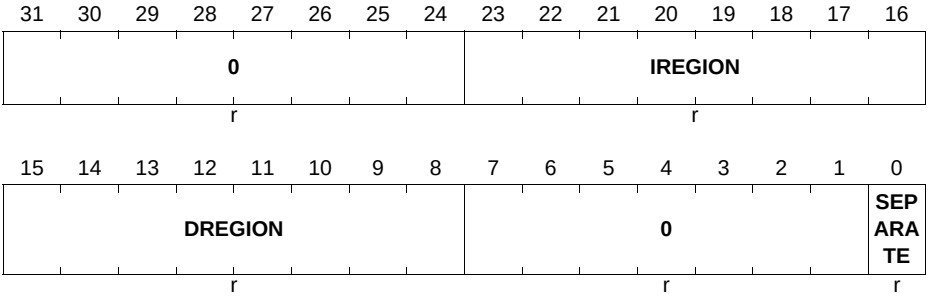
Field	Bits	Type	Description
INTID	[8:0]	w	**Interrupt ID of the interrupt to trigger** in the range 0-111. For example, a value of 0x03 specifies interrupt IRQ3.
0	[31:9]	r	**Reserved** Read as 0; should be written with 0.

2.9.4 MPU Registers

MPU Type Register

The MPU_TYPE register indicates whether the MPU is present, and if so, how many regions it supports.

MPU_TYPE
MPU Type Register **(E000 ED90$_H$)** **Reset Value: 0000 0800$_H$**



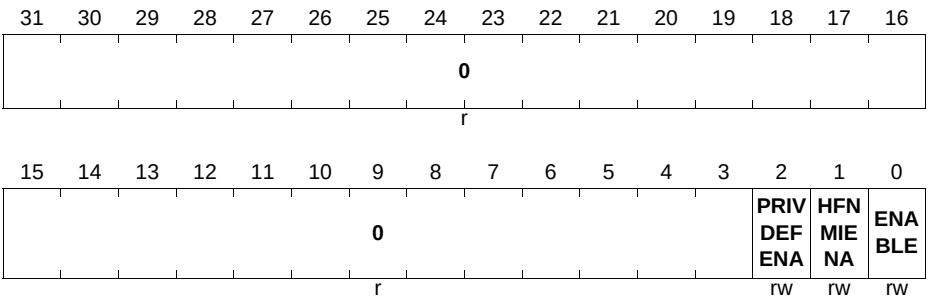
Field	Bits	Type	Description
SEPARATE	0	r	**Support for unified or separate instruction and date memory maps** 0_B unified
DREGION	[15:8]	r	**Number of supported MPU data regions** 08_H Eight MPU regions
IREGION	[23:16]	r	**Number of supported MPU instruction regions** Always contains 0x00. The MPU memory map is unified and is described by the DREGION field.
0	[31:24], [7:1]	r	**Reserved** Read as 0; should be written with 0.

MPU Control Register

The MPU_CTRL register:

- enables the MPU
- enables the default memory map background region
- enables use of the MPU when in the hard fault, Non-maskable Interrupt (NMI), and FAULTMASK escalated handlers.

MPU_CTRL
MPU Control Register **(E000 ED94$_H$)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
ENABLE	0	rw	**Enable MPU** 0_B MPU disabled 1_B MPU enabled.
HFNMIENA	1	rw	**Enable the operation of MPU during hard fault, NMI, and FAULTMASK handlers** When the MPU is enabled: 0_B MPU is disabled during hard fault, NMI, and FAULTMASK handlers, regardless of the value of the ENABLE bit 1_B the MPU is enabled during hard fault, NMI, and FAULTMASK handlers. When the MPU is disabled, if this bit is set to 1 the behavior is Unpredictable.

Field	Bits	Type	Description
PRIVDEFENA	2	rw	**Enables privileged software access to the default memory map** 0_B If the MPU is enabled, disables use of the default memory map. Any memory access to a location not covered by any enabled region causes a fault. 1_B If the MPU is enabled, enables use of the default memory map as a background region for privileged software accesses. When enabled, the background region acts as if it is region number -1. Any region that is defined and enabled has priority over this default map. f the MPU is disabled, the processor ignores this bit.
0	[31:3]	r	**Reserved** Read as 0; should be written with 0.

When ENABLE and PRIVDEFENA are both set to 1:

- For privileged accesses, the default memory map is as described in Memory model on **Page 2-20**. Any access by privileged software that does not address an enabled memory region behaves as defined by the default memory map.
- Any access by unprivileged software that does not address an enabled memory region causes a MemManage fault.

XN and Strongly-ordered rules always apply to the System Control Space regardless of the value of the ENABLE bit.

When the ENABLE bit is set to 1, at least one region of the memory map must be enabled for the system to function unless the PRIVDEFENA bit is set to 1. If the PRIVDEFENA bit is set to 1 and no regions are enabled, then only privileged software can operate.

When the ENABLE bit is set to 0, the system uses the default memory map. This has the same memory attributes as if the MPU is not implemented, see **Table 2-6** on **Page 2-22**. The default memory map applies to accesses from both privileged and unprivileged software.

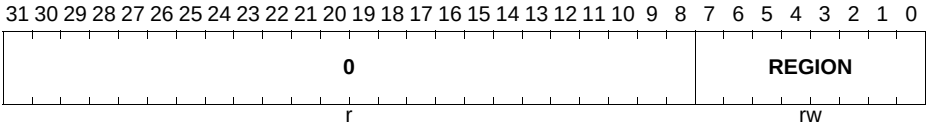
When the MPU is enabled, accesses to the System Control Space and vector table are always permitted. Other areas are accessible based on regions and whether PRIVDEFENA is set to 1.

Unless HFNMIENA is set to 1, the MPU is not enabled when the processor is executing the handler for an exception with priority –1 or –2. These priorities are only possible when handling a hard fault or NMI exception, or when FAULTMASK is enabled. Setting the HFNMIENA bit to 1 enables the MPU when operating with these two priorities.

MPU Region Number Register

The MPU_RNR selects which memory region is referenced by the MPU_RBAR and MPU_RASR registers.

MPU_RNR
MPU Region Number Register (E000 ED98$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
REGION	[7:0]	rw	**Region** Indicates the MPU region referenced by the MPU_RBAR and MPU_RASR registers. The MPU supports 8 memory regions, so the permitted values of this field are 0-7.
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

Normally, you write the required region number to this register before accessing the MPU_RBAR or MPU_RASR. However you can change the region number by writing to the MPU RBAR with the VALID bit set to 1, see MPU Region Base Address Register. This write updates the value of the REGION field.

MPU Region Base Address Register

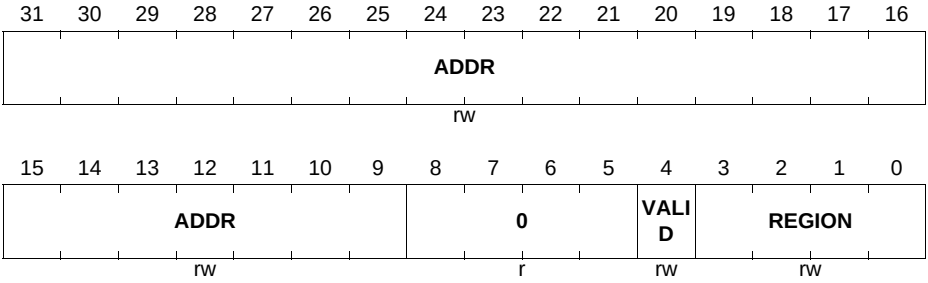
The MPU_RBAR defines the base address of the MPU region selected by the MPU_RNR, and can update the value of the MPU_RNR.

MPU_RBAR
MPU Region Base Address Register
(E000 ED9C$_H$) **Reset Value: 0000 0000$_H$**

MPU_RBAR_A1
MPU Region Base Address Register A1
(E000 EDA4$_H$) **Reset Value: 0000 0000$_H$**

MPU_RBAR_A2
MPU Region Base Address Register A2
(E000 EDAC$_H$) **Reset Value: 0000 0000$_H$**

MPU_RBAR_A3
MPU Region Base Address Register A3
(E000 EDB4$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
REGION	[3:0]	rw	**MPU region field** For the behavior on writes, see the description of the VALID field. On reads, returns the current region number, as specified by the RNR.

Field	Bits	Type	Description
VALID	4	rw	**MPU Region Number valid bit** Write: 0_B MPU_RNR not changed, and the processor: - updates the base address for the region specified in the MPU_RNR - ignores the value of the REGION field 1_B the processor: - updates the value of the MPU_RNR to the value of the REGION field - updates the base address for the region specified in the REGION field. Always reads as zero.
ADDR	[31:9]	rw	**Region base address field** The value of N (N = 9 for bit definition) depends on the region size. For more information see The ADDR field.
0	[8:5]	r	**Reserved** Read as 0; should be written with 0.

The ADDR field

The ADDR field is bits[31:N] of the MPU_RBAR. The region size, as specified by the SIZE field in the MPU_RASR, defines the value of N:

N = Log2(Region size in bytes),

If the region size is configured to 4GB, in the MPU_RASR, there is no valid ADDR field. In this case, the region occupies the complete memory map, and the base address is 0x00000000.

The base address is aligned to the size of the region. For example, a 64KB region must be aligned on a multiple of 64KB, for example, at 0x00010000 or 0x00020000.

MPU Region Attribute and Size Register

The MPU_RASR defines the region size and memory attributes of the MPU region specified by the MPU_RNR, and enables that region and any subregions.

MPU_RASR is accessible using word or halfword accesses:

- the most significant halfword holds the region attributes
- the least significant halfword holds the region size and the region and subregion enable bits.

MPU_RASR
MPU Region Attribute and Size Register
(E000 EDA0$_H$) **Reset Value: 0000 0000$_H$**

MPU_RASR_A1
MPU Region Attribute and Size Register A1
(E000 EDA8$_H$) **Reset Value: 0000 0000$_H$**

MPU_RASR_A2
MPU Region Attribute and Size Register A2
(E000 EDB0$_H$) **Reset Value: 0000 0000$_H$**

MPU_RASR_A3
MPU Region Attribute and Size Register A3
(E000 EDB8$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0			XN	0	AP			0		TEX			S	C	B
r			rw	r	rw			r		rw			rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SRD								0		SIZE					ENABLE
rw								r		rw					rw

Field	Bits	Type	Description
ENABLE	0	rw	**Region enable bit.**
SIZE	[5:1]	rw	**MPU protection region size** The minimum permitted value is 3 (0b00010), see See SIZE field values for more information.
SRD	[15:8]	rw	**Subregion disable bits** For each bit in this field: 0_B corresponding sub-region is enabled 1_B corresponding sub-region is disabled See **Subregions**on **Page 2-52** for more information. Region sizes of 128 bytes and less do not support subregions. When writing the attributes for such a region, write the SRD field as 0x00.
B	16	rw	**Memory access attribute** see **Table 2-15** on**Page 2-48**.

Field	Bits	Type	Description
C	17	rw	**Memory access attribute** see **Table 2-15** on**Page 2-48**.
S	18	rw	**Shareable bit** see **Table 2-15** on**Page 2-48**.
TEX	[21:19]	rw	**Memory access attribute** see **Table 2-15** on**Page 2-48**.
AP	[26:24]	rw	**Access permission field** see **Table 2-18** on **Page 2-49**.
XN	28	rw	**Instruction access disable bit** 0_B instruction fetches enabled 1_B instruction fetches disabled.
0	[31:29, 27, [23:22], [7:6]	r	**Reserved** Read as 0; should be written with 0.

For information about access permission, see **MPU Access Permission Attributes** on **Page 2-48**.

SIZE field values

The SIZE field defines the size of the MPU memory region specified by the RNR. as follows:

(Region size in bytes) = 2(SIZE+1)

The smallest permitted region size is 32B, corresponding to a SIZE value of 4. **Table 2-22** gives example SIZE values, with the corresponding region size and value of N in the MPU_RBAR.

Table 2-22 Example SIZE field values

SIZE value	Region size	Value of N[1)]	Note
0b00100 (4)	32B	5	Minimum permitted size
0b01001 (9)	1KB	10	-
0b10011 (19)	1MB	20	-
0b11101 (29)	1GB	30	-
0b11111 (31)	4GB	32	Maximum possible size

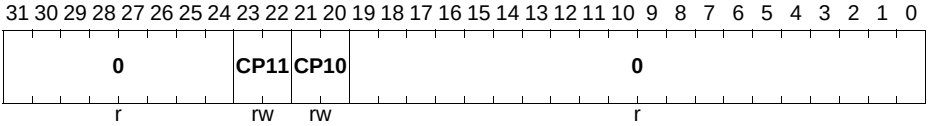
1) In the MPU_RBAR, see MPU Region Base Address Register on **Page 2-95**.

2.9.5 FPU Registers

Coprocessor Access Control Register

The CPACR register specifies the access privileges for coprocessors.

CPACR
Coprocessor Access Control Register
(E000 ED88$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
CP10	[21:20]	rw	**Access privileges for coprocessor 10** The possible values of each field are: 00_B Access denied. Any attempted access generates a NOCP UsageFault. 01_B Privileged access only. An unprivileged access generates a NOCP fault. 10_B Reserved. The result of any access is Unpredictable. 11_B Full access.
CP11	[23:22]	rw	**Access privileges for coprocessor 11** The possible values of each field are: 00_B Access denied. Any attempted access generates a NOCP UsageFault. 01_B Privileged access only. An unprivileged access generates a NOCP fault. 10_B Reserved. The result of any access is Unpredictable. 11_B Full access.
0	[31:24], [19:0]	r	**Reserved** Read as 0; should be written with 0.

Floating-point Context Control Register

The FPCCR register sets or returns FPU control data.

FPCCR
Floating-point Context Control Register
(E000 EF34$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ASPEN	LSPEN	0													
rw	rw	r													

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							MONRDY	0	BFRDY	MMRDY	HFRDY	THREAD	0	USER	LSPACT
r							rw	r	rw	rw	rw	rw	r	rw	rw

Field	Bits	Type	Description
LSPACT	0	rw	**Lazy State Preservation Active** 0_B Lazy state preservation is not active. 1_B Lazy state preservation is active. floating-point stack frame has been allocated but saving state to it has been deferred.
USER	1	rw	**User allocated Stack Frame** 0_B Privilege level was not user when the floating-point stack frame was allocated. 1_B Privilege level was user when the floating-point stack frame was allocated.
THREAD	3	rw	**Thread Mode allocated Stack Frame** 0_B Mode was not Thread Mode when the floating-point stack frame was allocated. 1_B Mode was Thread Mode when the floating-point stack frame was allocated.
HFRDY	4	rw	**HardFault Ready** 0_B Priority did not permit setting the HardFault handler to the pending state when the floating-point stack frame was allocated. 1_B Priority permitted setting the HardFault handler to the pending state when the floating-point stack frame was allocated.

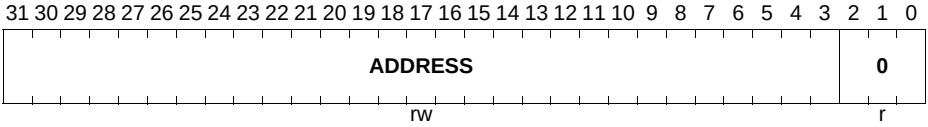
Field	Bits	Type	Description
MMRDY	5	rw	**MemManage Ready** 0_B MemManage is disabled or priority did not permit setting the MemManage handler to the pending state when the floating-point stack frame was allocated. 1_B MemManage is enabled and priority permitted setting the MemManage handler to the pending state when the floating-point stack frame was allocated.
BFRDY	6	rw	**BusFault Ready** 0_B BusFault is disabled or priority did not permit setting the BusFault handler to the pending state when the floating-point stack frame was allocated. 1_B BusFault is enabled and priority permitted setting the BusFault handler to the pending state when the floating-point stack frame was allocated.
MONRDY	8	rw	**Monitor Ready** 0_B Debug Monitor is disabled or priority did not permit setting MON_PEND when the floating-point stack frame was allocated. 1_B Debug Monitor is enabled and priority permits setting MON_PEND when the floating-point stack frame was allocated.
LSPEN	30	rw	**Lazy State Preservation Enabled** 0_B Disable automatic lazy state preservation for floating-point context. 1_B Enable automatic lazy state preservation for floating-point context.
ASPEN	31	rw	**Automatic State Preservation** Enables CONTROL setting on execution of a floating-point instruction. This results in automatic hardware state preservation and restoration, for floating-point context, on exception entry and exit. 0_B Disable CONTROL setting on execution of a floating-point instruction. 1_B Enable CONTROL setting on execution of a floating-point instruction.

Field	Bits	Type	Description
0	[29:9], 7, 2	r	**Reserved** Read as 0; should be written with 0.

Floating-point Context Address Register

The FPCAR register holds the location of the unpopulated floating-point register space allocated on an exception stack frame.

FPCAR
Floating-point Context Address Register
(E000 EF38$_H$) **Reset Value: 0000 0000$_H$**



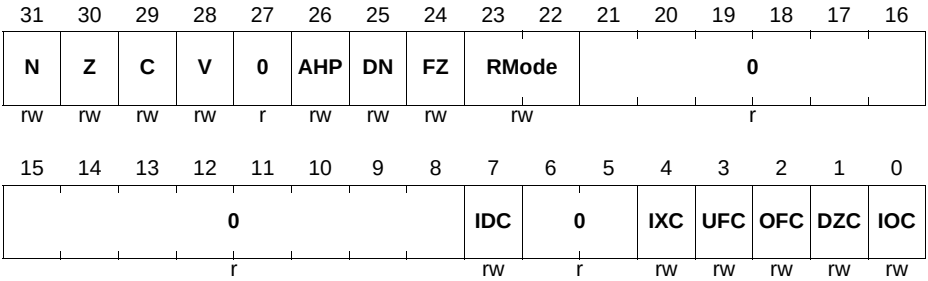
Field	Bits	Type	Description
ADDRESS	[31:3]	rw	**Address** The location of the unpopulated floating-point register space allocated on an exception stack frame.
0	[2:0]	r	**Reserved** Read as 0; should be written with 0.

Floating-point Status Control Register

The FPSCR register provides all necessary User level control of the floating-point system.

FPSCR
Floating-point Status Control Register

Reset Value: XXXX XXXX$_H$



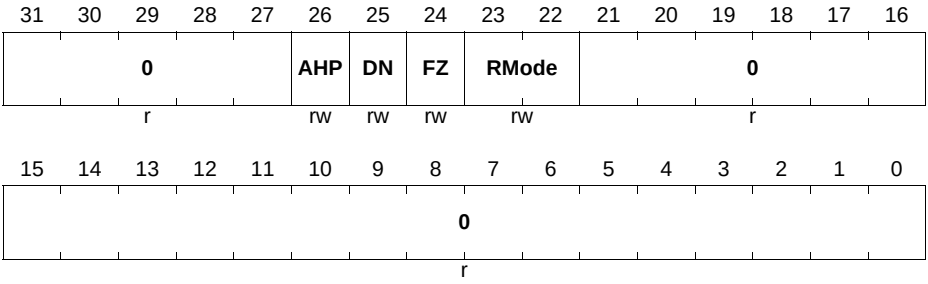
Field	Bits	Type	Description
IOC	0	rw	**Invalid Operation cumulative exception bit** IOC set to 1 indicates that the Invalid Operation cumulative exception has occurred since 0 was last written to IOC.
DZC	1	rw	**Division by Zero cumulative exception bit** DZC set to 1 indicates that the Division by Zero cumulative exception has occurred since 0 was last written to DZC.
OFC	2	rw	**Overflow cumulative exception bit** OFC set to 1 indicates that the Overflow cumulative exception has occurred since 0 was last written to OFC.
UFC	3	rw	**Underflow cumulative exception bit** UFC set to 1 indicates that the Underflow cumulative exception has occurred since 0 was last written to UFC.
IXC	4	rw	**Inexact cumulative exception bit** IXC set to 1 indicates that the Inexact cumulative exception has occurred since 0 was last written to IXC.
IDC	7	rw	**Input Denormal cumulative exception bit** see bits [4:0].

Field	Bits	Type	Description
RMode	[23:22]	rw	**Rounding Mode control field** 00_B Round to Nearest (RN) mode 01_B Round towards Plus Infinity (RP) mode 10_B Round towards Minus Infinity (RM) mode 11_B Round towards Zero (RZ) mode. The specified rounding mode is used by almost all floating-point instructions.
FZ	24	rw	**Flush-to-zero mode control bit** 0_B Flush-to-zero mode disabled. Behavior of the floating-point system is fully compliant with the IEEE 754 standard. 1_B Flush-to-zero mode enabled.
DN	25	rw	**Default NaN mode control bit** 0_B NaN operands propagate through to the output of a floating-point operation. 1_B Any operation involving one or more NaNs returns the Default NaN.
AHP	26	rw	**Alternative half-precision control bit** 0_B IEEE half-precision format selected. 1_B Alternative half-precision format selected.
V	28	rw	**Overflow condition code flag** Floating-point comparison operations update this flag.
C	29	rw	**Carry condition code flag** Floating-point comparison operations update this flag.
Z	30	rw	**Zero condition code flag** Floating-point comparison operations update this flag.
N	31	rw	**Negative condition code flag** Floating-point comparison operations update this flag.
0	27, [21:8], [6:5]	r	**Reserved** Read as 0; should be written with 0.

Floating-point Default Status Control Register

The FPDSCR register holds the default values for the floating-point status control data.

FPDSCR
Floating-point Default Status Control Register
(E000 EF3C$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
RMode	[23:22]	rw	**Default value for FPSCR.RMode**
FZ	24	rw	**Default value for FPSCR.FZ**
DN	25	rw	**Default value for FPSCR.DN**
AHP	26	rw	**Default value for FPSCR.AHP**
0	[31:27], [21:0]	r	**Reserved** Read as 0; should be written with 0.

3 Bus System

The XMC4500 is targeted for use in embedded systems. Therefore the key features are timing determinism and low latency on real time events. Bus bandwidth is required particularly for communication peripherals.

The bus system will therefore provide:

- Timing Determinism
- Low Latency
- Performance
- Throughput

3.1 Bus Interfaces

This chapter describes the features for the two kinds of interfaces.

- **Memory Interface**
- **Peripheral Interface**

All on-chip peripherals and memories are attached to the Bus Matrix, in some cases via peripheral bridges. All on-chip modules implement Little Endian data organization. The following types of transfer are supported:

- Locked Transfers
- Burst Operation
- Protection Control

Pipelining is also supported for bandwidth critical transfers.

Memory Interface

The on-chip memories capable to accept a transfer request with each bus clock cycle.

The memory interface data bus width is 32-bit. Each memory slave support 32-bit, 16-bit and 8-bit access types.

Peripheral Interface

Each slave supports 32-bit accesses. Some slaves also support 8-bit and/or 16-bit accesses.

3.2 Bus Matrix

The central part of the bus system is built up around a multilayer AHB-lite compliant matrix. By means of this technique the bus masters and bus slaves can be connected in a flexible way while maintaining high bus performance.

The Bus Martix depicted in **Figure 3-1** implements an optimized topology enabling zero wait state data accesses between the Masters and Slaves connected to it. Dedicated

arbitration scheme enables optimal access conflicts resolution resulting in improved system stability and real time behavior.

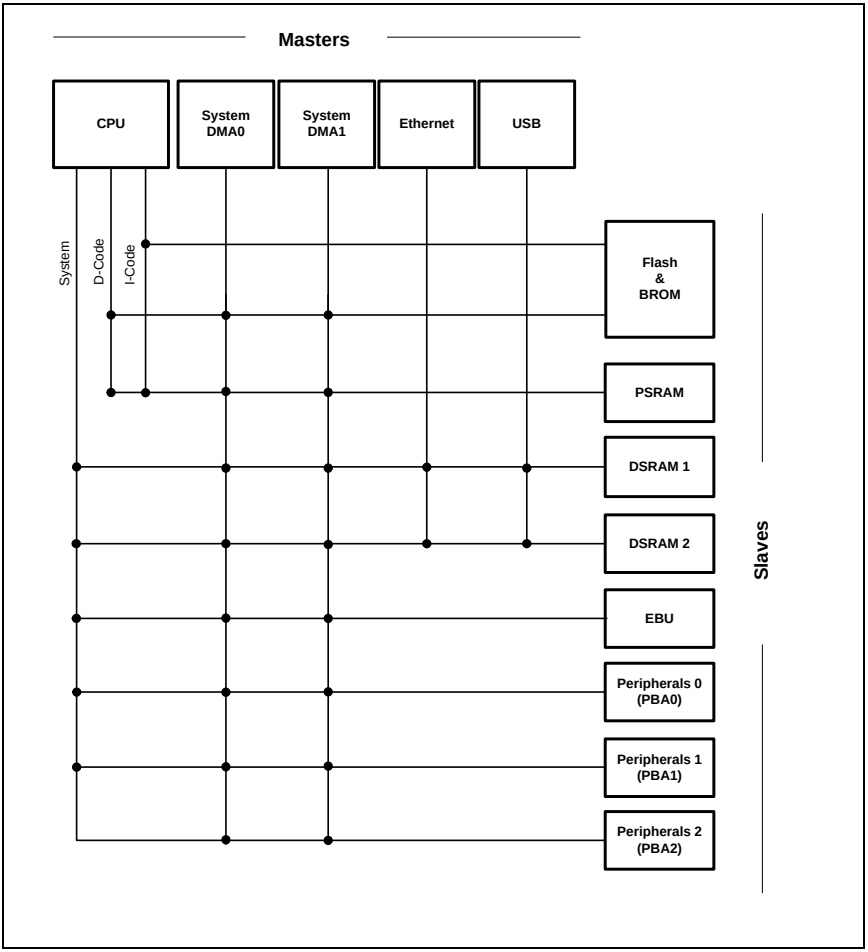


Figure 3-1 Multilayer Bus Matrix

Arbitration Priorities

In case of concurring access to the same slave the master with the highest priority is granted the bus.

Table 3-1 Access Priorities per Slave[1)]

	CPU	GPDMA0	GPDMA1	ETH	USB
PMU/FLASH	1	2	3	-	-
PSRAM	1	2	3	-	-
DSRAM1	1	2	3	4	5
DSRAM2	1	4	5	2	3
EBU	1	2	3	-	-
PBA0	1	2	3	-	-
PBA1	1	2	3	-	-
PBA2	1	2	3	-	-

1) Lower number means higher priority

The DSRAM priorities are choosen to support the application dependance of the data memories:

- DSRAM1: general purpose data storage
- DSRAM2: Ethernet and USB data storage

4 Service Request Processing

A hardware pulse or level change is called Service Request (SR) in an XMC4500 system. Service Requests are the fastest way to send trigger “messages” between connected on-chip resources.

An SR can generate any of the following requests

- Interrupt
- DMA
- Peripheral action

This chapter describes the available Service Requests and the different ways to select and process them.

Notes

1. *The CPU exception model and interrupt processing (by NVIC unit) are described in the CPU chapter.*
2. *General Purpose DMA request processing is described in the GPDMA chapter*

Table 4-1 Abbreviations

DLR	DMA Line Router
ERU	Event Request Unit
NVIC	Nested Vectored Interrupt Controller
SR	Service Request

4.1 Overview

Efficient Service Request Processing is based on the interconnect between the request sources and the request processing units. XMC4500 provides both fixed and programmable interconnect.

4.1.1 Features

The following features are provided for Service Request processing:

- Connectivity matrix between Service Requests and request processing units
  - Fixed connections
  - Programmable connections using ERU
- Event Request Unit (ERU)
  - Flexible processing of external and internal service requests
  - Programmable for edge and/or level triggering
  - Multiple inputs per channel
  - Triggers combinable from multiple inputs
  - Input and output gating

- DMA Line Router (DLR)
  – Routing and processing of DMA requests

4.1.2 Applications

The following table lists features of the Service Request Processing unit mapped to selected applications.

Table 4-2 Feature to Application Mappings

Feature	Application
TBD	TBD

4.1.3 Block Diagram

The shaded components shown in **Figure 4-1** are described in this chapter.

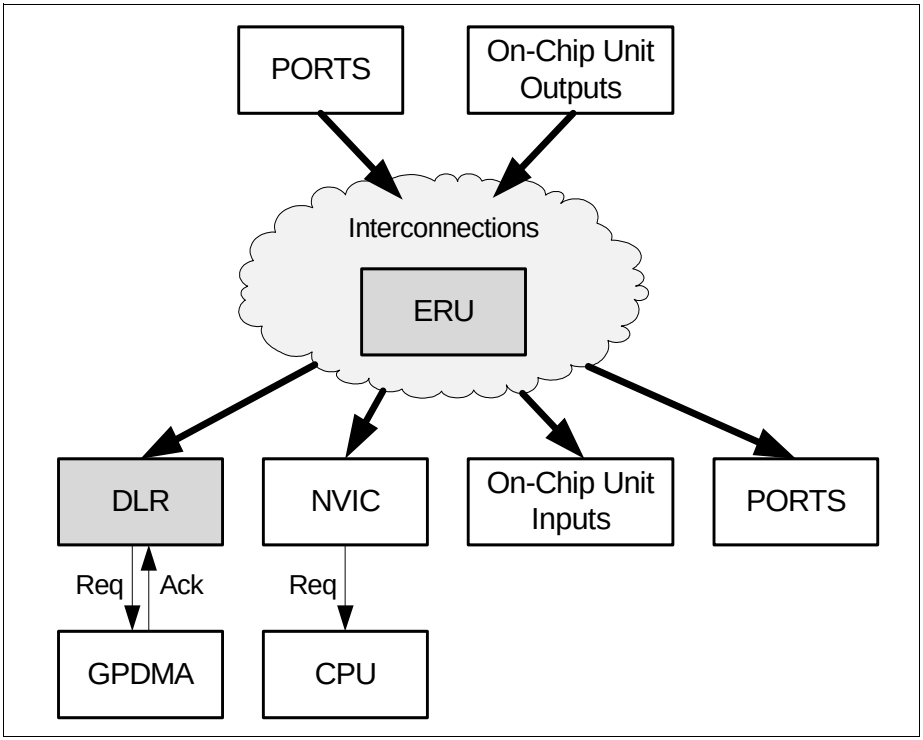


Figure 4-1 Block Diagram on Service Request Processing

4.2 Service Request Distribution

The following figure shown an example of how a service request can be distributed concurrently. To support the concurrent distribution to multiple receivers, the receiving modules are capable to enable/disable incoming requests.

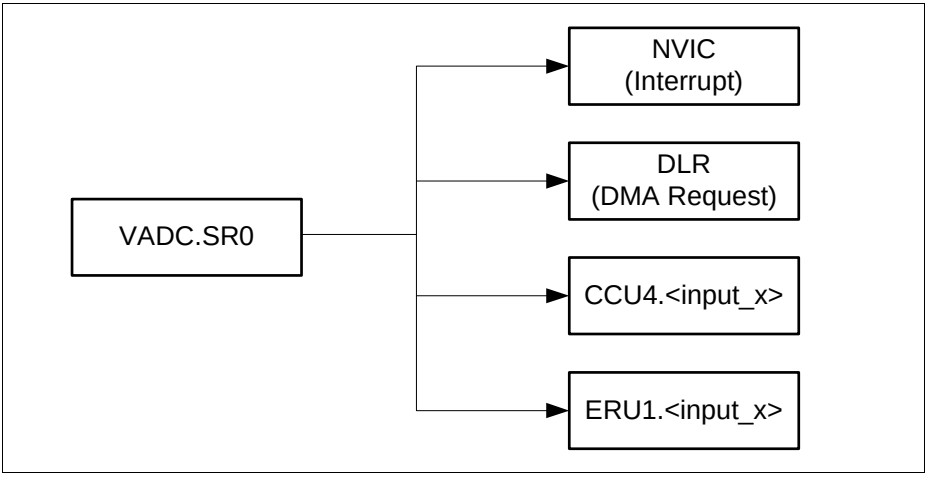


Figure 4-2 Example for Service Request Distribution

The units involved in Service Request distribution can be subdivided into

- Embedded real time services
- Interrupt and DMA services

Embedded real time services

Connectivity between On-Chip Units and PORTS is real time application and also chip package dependant. Related connectivity and availability of pins can be looked up in the

- “Interconnects” Section of the respective module(s) chapters
- “Parallel Ports” chapter and Data Sheet for PORTS
- Event Request Unit (**Section 4.5**)

Interrupt and DMA services

The following table gives an overview on the number of service requests per module and how the service requests are assigned to NVIC Interrupt and DLR/GPDMA service providers.

Service Requests can be of type “Level” or “Pulse”. The DLR/GPDMA can only process “Pulse” type of requests while the NVIC can process both. The type of Service Requests generated is listed in column “Type” in **Table 4-3**.

Table 4-3 Interrupt and DMA services per Module

Modules	Request Sources	NVIC	DLR/GPDMA	Type
VADC	20	20	20	Pulse
DSD	8	8	4	Pulse
DAC	2	2	2	Pulse
CCU40-3	16	16	8	Pulse
CCU80-1	8	8	4	Pulse
POSIF0-1	4	4	-	Pulse
CAN	8	8	4	Pulse
USIC0-2	18	18	12	Pulse
LEDTS0	1	1	-	Pulse
FCE	1	1	-	Pulse
PMU0/Flash	1	1	-	Pulse
GPDMA0-1	2	2	-	Level
SCU	1	1	-	Pulse
ERU0-1	8	8	4	Pulse & Level
SDMMC	1	1	-	Level
USB0	1	1	-	Level
ETH0	1	1	-	Level
Totals	102	102	58	-

4.3 Interrupt Service Requests

The NVIC is an integral part of the Cortex M4 processor unit. Due to a tight coupling with the CPU it allows to achieve lowest interrupt latency and efficient processing of late arriving interrupts.

NVIC Features

- 112 interrupt nodes
- Programmable priority level of 0-63 for each interrupt node. A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority
- Request source can be level or edge signal type
- Dynamic reprioritization of interrupts.
- Grouping of priority values into group priority and subpriority fields.
- Interrupt tail-chaining.

- One external Non-maskable interrupt (NMI)
- Relocatable vector table
- Software interrupt generation

Level-sensitive and pulse interrupts

The NVIC is capable to capture both level-sensitive and pulse interrupts.

- A level-sensitive interrupt is held asserted until the peripheral deasserts the interrupt signal. Deassertion is typically triggered by the interrupt service routine (ISR). It is
  - used for less frequent requests and
  - the ISR is often more complex and longer.
- A pulse interrupt is asserted and after a fixed period of time automatically deasserted. The period of time depends on the peripheral, please refer to the “Service Request Generation” section of the respective peripheral. It is
  - used for more frequent requests and
  - the ISR is often more simple and shorter.

The way to process both types of requests differs and is described in section “Level-sensitive and pulse interrupts” in the CPU chapter.

Service Request to IRQ Number Assignment

Table 4-4 lists the service request sources per on-chip unit and their assignment to NVIC IRQ numbers. The resulting exception number is calculated by adding 16 to the IRQ Number. The first 16 exception numbers are used by the Cortex M4 CPU. For calculation of the resulting exception routine address please refer to the CPU chapter.

Table 4-4 Interrupt Node assignment

Service Request	IRQ Number	Description
SCU.SR0	0	System Control
ERU0.SR0 ERU0.SR3	1...4	External Request Unit 0
ERU1.SR0 ERU1.SR3	5...8	External Request Unit 1
NC	9, 10, 11	Reserved
PMU0.SR0	12	Program Management Unit
NC	13	Reserved
VADC.C0SR0 - VADC.C0SR3	14...17	Analog to Digital Converter Common Block 0
VADC.G0SR0 - VADC.G0SR3	18...21	Analog to Digital Converter Group 0

Table 4-4 Interrupt Node assignment (cont'd)

Service Request	IRQ Number	Description
VADC.G1SR0 - VADC.G1SR3	22...25	Analog to Digital Converter Group 1
VADC.G2SR0 - VADC.G2SR3	26...29	Analog to Digital Converter Group 2
VADC.G3SR0 - VADC.G3SR3	30...33	Analog to Digital Converter Group 3
DSD.SRM0 - DSD.SRM3	34...37	Delta Sigma Demodulator Main
DSD.SRA0 - DSD.SRA3	38...41	Delta Sigma Demodulator Auxiliary
DAC.SR0 - DAC.SR1	42, 43	Digital to Analog Converter
CCU40.SR0 - CCU40.SR3	44...47	Capture Compare Unit 4 (Module 0)
CCU41.SR0 - CCU41.SR3	48...51	Capture Compare Unit 4 (Module 1)
CCU42.SR0 - CCU42.SR3	52...55	Capture Compare Unit 4 (Module 2)
CCU43.SR0 - CCU43.SR3	56...59	Capture Compare Unit 4 (Module 3)
CCU80.SR0 - CCU80.SR3	60...63	Capture Compare Unit 8 (Module 0)
CCU81.SR0 - CCU81.SR3	64...67	Capture Compare Unit 8 (Module 1)
POSIF0.SR0 - POSIF0.SR1	68...69	Position Interface (Module 0)
POSIF1.SR0 - POSIF1.SR1	70...71	Position Interface (Module 1)
NC	72...75	Reserved
CAN.SR0 - CAN.SR7	76...83	MultiCAN
USIC0.SR0 - USIC0.SR5	84...89	Universal Serial Interface Channel (Module 0)

Table 4-4 Interrupt Node assignment (cont'd)

Service Request	IRQ Number	Description
USIC1.SR0 - USIC1.SR5	90...95	Universal Serial Interface Channel (Module 1)
USIC2.SR0 - USIC2.SR5	96...101	Universal Serial Interface Channel (Module 2)
LEDTS0.SR0	102	LED and Touch Sense Control Unit (Module 0)
NC	103	Reserved
FCE.SR0	104	Flexible CRC Engine
GPDMA0.SR0	105	General Purpose DMA unit 0
SDMMC.SR0	106	Multi Media Card Interface
USB0.SR0	107	Universal Serial Bus
ETH0.SR0	108	Ethernet (Module 0)
NC	109	Reserved
GPDMA1.SR0	110	General Purpose DMA unit 1
NC	111	Reserved

4.4 DMA Line Router (DLR)

The DMA line router provides the following functionality:

- Selection of DMA request sources
- Handling of the DMA request and acknowledge handshake
- Detection of service request overruns

4.4.1 Functional Description

This unit enables the user to select 12 DMA service requests out of the set of DMA capable service request sources. It handles the Request and Acknowledge handshake to the GPDMA. Furthermore it detects service request overruns.

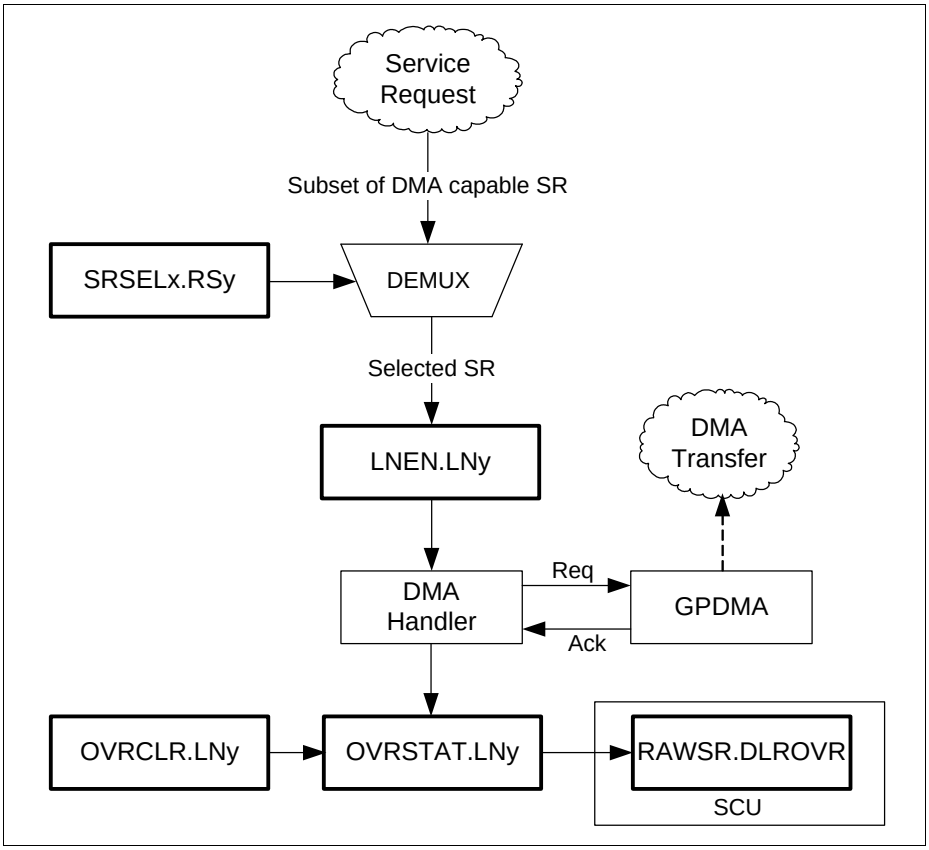


Figure 4-3 DMA Line Handler

For each DMA line the user can assign one service request source from the subset of DMA capable XMC4500 service request sources. The assignment is done by programming the SRSx bit field of register **DLR_SRSELx**.

If the selected service request pulse occurs and if the according line is enabled by the **DLR_LNEN** register, then the DMA handler forwards the request and stores it until the GPDMA responds with an acknowledge. A request pulse occurring while another transfer is ongoing is ignored and the according overrun status bit is set in the **DLR_OVRSTAT** register.

Once the overrun condition is entered the user can clear the overrun status bits by writing to the **DLR_OVRCLR** register. Additionally the pending request must be reset by successively disabling and enabling the respective line.

If any bit within the **DLR_OVRSTAT** register is set, a service request is flagged by setting the SCU_RAWSR.DLROVR bit.

The DLR unit has the following inputs:

Table 4-5 DMA Handler Service Request inputs

Service Request	# of Inputs	Description
ERU1.SR1 - ERU1.SR4	4	ERU1 (System Control) requests
VADC.C0SR0 - VADC.C0SR3	4	Analog to Digital Converter Common Block 0
VADC.G0SR0 - VADC.G0SR3	4	Analog to Digital Converter Group 0
VADC.G1SR0 - VADC.G1SR3	4	Analog to Digital Converter Group 1
VADC.G2SR0 - VADC.G2SR3	4	Analog to Digital Converter Group 2
VADC.G3SR0 - VADC.G3SR3	4	Analog to Digital Converter Group 3
DSD.SR0 - DSD.SR3	4	Delta Sigma Demodulator
DAC.SR0 - DAC.SR1	2	Digital to Analog Converter
CCU40.SR0 - CCU40.SR1	2	Capture Compare Unit 4 (Module 0)
CCU41.SR0 - CCU41.SR1	2	Capture Compare Unit 4 (Module 1)
CCU42.SR0 - CCU42.SR1	2	Capture Compare Unit 4 (Module 2)
CCU43.SR0 - CCU43.SR1	2	Capture Compare Unit 4 (Module 3)
CCU80.SR0 - CCU80.SR1	2	Capture Compare Unit 8 (Module 0)
CCU81.SR0 - CCU81.SR1	2	Capture Compare Unit 8 (Module 1)
CAN.SR0 - CAN.SR3	4	MultiCAN

Table 4-5 DMA Handler Service Request inputs (cont'd)

Service Request	# of Inputs	Description
USIC0.SR0 - USIC0.SR3	4	Universal Serial Interface Channel (Module 0)
USIC1.SR0 - USIC1.SR3	4	Universal Serial Interface Channel (Module 1)
USIC2.SR0 - USIC2.SR3	4	Universal Serial Interface Channel (Module 2)

4.4.2 DMA Service Request Source Selection

The selection of the request sources is done according to the following table by programming the **DLR_SRSELx** register. Please note that each service request source can be assigned to 2 different lines to provide maximum flexibility. For example VADC.SR0 can be assigned to line 0 and 4.

Table 4-6 DMA Request Source Selection

DMA Line	DMA Request Line	Selected by DLR_SRSEL bit field
0	ERU0.SR0	RS0 = 0000_B
	VADC.C0SR0	RS0 = 0001_B
	VADC.G0SR3	RS0 = 0010_B
	VADC.G2SR0	RS0 = 0011_B
	VADC.G2SR3	RS0 = 0100_B
	DSD.SRM0	RS0 = 0101_B
	CCU40.SR0	RS0 = 0110_B
	CCU80.SR0	RS0 = 0111_B
	Reserved[1)]	RS0 = 1000_B
	CAN.SR0	RS0 = 1001_B
	USIC0.SR0	RS0 = 1010_B
	USIC1.SR0	RS0 = 1011_B
	Reserved[1)]	RS0 = 1100_B
	VADC.G3SR3	RS0 = 1101_B
	CCU42.SR0	RS0 = 1110_B
	Reserved[1)]	RS0 = 1111_B
1	ERU0.SR3	RS1 = 0000_B

Table 4-6 DMA Request Source Selection (cont'd)

DMA Line	DMA Request Line	Selected by DLR_SRSEL bit field
	VADC.C0SR1	RS1 = 0001_B
	VADC.G0SR2	RS1 = 0010_B
	VADC.G1SR0	RS1 = 0011_B
	VADC.G2SR2	RS1 = 0100_B
	DAC.SR0	RS1 = 0101_B
	CCU40.SR0	RS1 = 0110_B
	CCU80.SR0	RS1 = 0111_B
	Reserved[1)]	RS1 = 1000_B
	CAN.SR0	RS1 = 1001_B
	USIC0.SR0	RS1 = 1010_B
	USIC1.SR0	RS1 = 1011_B
	Reserved[1)]	RS1 = 1100_B
	VADC.G3SR0	RS1 = 1101_B
	CCU42.SR0	RS1 = 1110_B
	Reserved[1)]	RS1 = 1111_B
2	ERU0.SR1	RS2 = 0000_B
	VADC.C0SR2	RS2 = 0001_B
	VADC.C0SR3	RS2 = 0010_B
	VADC.G1SR3	RS2 = 0011_B
	VADC.G2SR1	RS2 = 0100_B
	DSD.SRM1	RS2 = 0101_B
	DSD.SRM3	RS2 = 0110_B
	CCU40.SR1	RS2 = 0111_B
	CCU80.SR1	RS2 = 1000_B
	Reserved[1)]	RS2 = 1001_B
	CAN.SR1	RS2 = 1010_B
	USIC0.SR1	RS2 = 1011_B
	USIC1.SR1	RS2 = 1100_B
	VADC.G3SR2	RS2 = 1101_B
	CCU42.SR1	RS2 = 1110_B

Table 4-6 DMA Request Source Selection (cont'd)

DMA Line	DMA Request Line	Selected by DLR_SRSEL bit field
	Reserved[1)]	RS2 = 1111_B
3	ERU0.SR2	RS3 = 0000_B
	VADC.C0SR2	RS3 = 0001_B
	VADC.C0SR3	RS3 = 0010_B
	VADC.G1SR1	RS3 = 0011_B
	VADC.G1SR2	RS3 = 0100_B
	DSD.SRM2	RS3 = 0101_B
	DAC.SR1	RS3 = 0110_B
	CCU40.SR1	RS3 = 0111_B
	CCU80.SR1	RS3 = 1000_B
	Reserved[1)]	RS3 = 1001_B
	CAN.SR1	RS3 = 1010_B
	USIC0.SR1	RS3 = 1011_B
	USIC1.SR1	RS3 = 1100_B
	VADC.G3SR1	RS3 = 1101_B
	CCU42.SR1	RS3 = 1110_B
	Reserved[1)]	RS3 = 1111_B
4	ERU0.SR2	RS4 = 0000_B
	VADC.G0SR0	RS4 = 0001_B
	VADC.G0SR1	RS4 = 0010_B
	VADC.G2SR1	RS4 = 0011_B
	VADC.G2SR2	RS4 = 0100_B
	DSD.SRM2	RS4 = 0101_B
	DAC.SR1	RS4 = 0110_B
	CCU41.SR0	RS4 = 0111_B
	CCU81.SR0	RS4 = 1000_B
	Reserved[1)]	RS4 = 1001_B
	CAN.SR2	RS4 = 1010_B
	USIC0.SR0	RS4 = 1011_B
	USIC1.SR0	RS4 = 1100_B

Table 4-6 DMA Request Source Selection (cont'd)

DMA Line	DMA Request Line	Selected by DLR_SRSEL bit field
	VADC.G3SR1	RS4 = 1101_B
	CCU43.SR0	RS4 = 1110_B
	Reserved[1)]	RS4 = 1111_B
5	ERU0.SR1	RS5 = 0000_B
	VADC.G0SR0	RS5 = 0001_B
	VADC.G0SR1	RS5 = 0010_B
	VADC.G1SR2	RS5 = 0011_B
	VADC.G2SR0	RS5 = 0100_B
	DAC.SR0	RS5 = 0101_B
	CCU41.SR0	RS5 = 0110_B
	CCU81.SR0	RS5 = 0111_B
	Reserved[1)]	RS5 = 1000_B
	CAN.SR2	RS5 = 1001_B
	USIC0.SR0	RS5 = 1010_B
	USIC1.SR0	RS5 = 1011_B
	Reserved[1)]	RS5 = 1100_B
	VADC.G3SR2	RS5 = 1101_B
	CCU43.SR0	RS5 = 1110_B
	Reserved[1)]	RS5 = 1111_B
6	ERU0.SR3	RS6 = 0000_B
	VADC.C0SR1	RS6 = 0001_B
	VADC.G0SR2	RS6 = 0010_B
	VADC.G1SR1	RS6 = 0011_B
	VADC.G2SR3	RS6 = 0100_B
	DSD.SRM1	RS6 = 0101_B
	DSD.SRM3	RS6 = 0110_B
	CCU41.SR1	RS6 = 0111_B
	CCU81.SR1	RS6 = 1000_B
	Reserved[1)]	RS6 = 1001_B
	CAN.SR3	RS6 = 1010_B

Table 4-6 DMA Request Source Selection (cont'd)

DMA Line	DMA Request Line	Selected by DLR_SRSEL bit field
	USIC0.SR1	RS6 = 1011_B
	USIC1.SR1	RS6 = 1100_B
	VADC.G3SR0	RS6 = 1101_B
	CCU43.SR1	RS6 = 1110_B
	Reserved[1)]	RS6 = 1111_B
7	ERU0.SR0	RS7 = 0000_B
	VADC.C0SR0	RS7 = 0001_B
	VADC.G0SR3	RS7 = 0010_B
	VADC.G1SR0	RS7 = 0011_B
	VADC.G1SR3	RS7 = 0100_B
	DSD.SRM0	RS7 = 0101_B
	CCU41.SR1	RS7 = 0110_B
	CCU81.SR1	RS7 = 0111_B
	Reserved[1)]	RS7 = 1000_B
	CAN.SR3	RS7 = 1001_B
	USIC0.SR1	RS7 = 1010_B
	USIC1.SR1	RS7 = 1011_B
	Reserved[1)]	RS7 = 1100_B
	VADC.G3SR3	RS7 = 1101_B
	CCU43.SR1	RS7 = 1110_B
	Reserved[1)]	RS7 = 1111_B
8	ERU0.SR0	RS8 = 0000_B
	VADC.C0SR0	RS8 = 0001_B
	VADC.G3SR0	RS8 = 0010_B
	DSD.SRM0	RS8 = 0011_B
	DAC.SR0	RS8 = 0100_B
	CCU42.SR0	RS8 = 0101_B
	USIC2.SR0	RS8 = 0110_B
	USIC2.SR2	RS8 = 0111_B
	Reserved[1)]	RS8 = $1XXX_B$

Table 4-6 DMA Request Source Selection (cont'd)

DMA Line	DMA Request Line	Selected by DLR_SRSEL bit field
9	ERU0.SR1	RS9 = 0000_B
	VADC.C0SR1	RS9 = 0001_B
	VADC.G3SR1	RS9 = 0010_B
	DSD.SRM1	RS9 = 0011_B
	DAC.SR1	RS9 = 0100_B
	CCU42.SR1	RS9 = 0101_B
	USIC2.SR1	RS9 = 0110_B
	USIC2.SR3	RS9 = 0111_B
	Reserved[1)]	RS9 = $1XXX_B$
10	ERU0.SR2	RS10 = 0000_B
	VADC.C0SR2	RS10 = 0001_B
	VADC.G3SR2	RS10 = 0010_B
	DSD.SRM2	RS10 = 0011_B
	DAC.SR0	RS10 = 0100_B
	CCU43.SR0	RS10 = 0101_B
	USIC2.SR0	RS10 = 0110_B
	USIC2.SR2	RS10 = 0111_B
	Reserved[1)]	RS10 = $1XXX_B$
11	ERU0.SR3	RS11 = 0000_B
	VADC.C0SR3	RS11 = 0001_B
	VADC.G3SR3	RS11 = 0010_B
	DSD.SRM3	RS11 = 0011_B
	DAC.SR1	RS11 = 0100_B
	CCU43.SR1	RS11 = 0101_B
	USIC2.SR1	RS11 = 0110_B
	USIC2.SR3	RS11 = 0111_B
	Reserved[1)]	RS11 = $1XXX_B$

1) Reserved combinations do not result in DMA requests. The reserved multiplexer inputs should be hard wired to inactive.

4.5 Event Request Unit (ERU)

The Event Request Unit (ERU) is a versatile multiple input event detection and processing unit. The XMC4500 provides two units - ERU0 and ERU1.

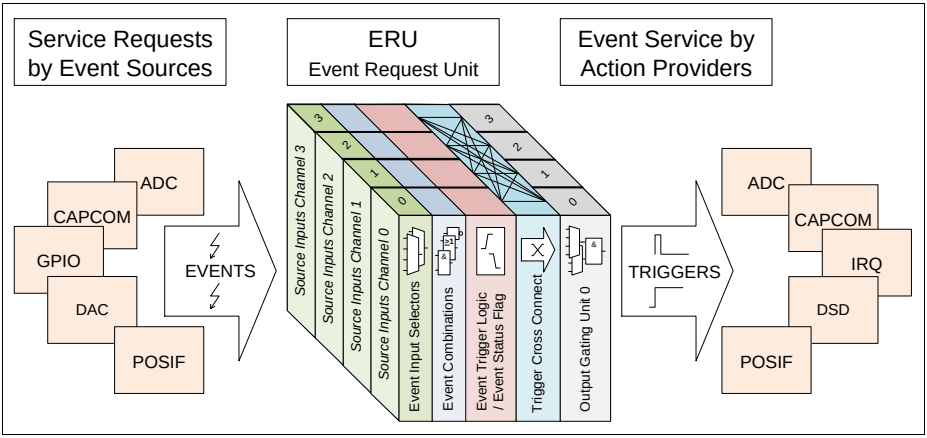


Figure 4-4 Event Request Unit Overview

Each ERU unit consists of the following blocks:

- An **Event Request Select (ERS)** unit.
  - Event Input Selectors allow the selection of one out of two inputs. For each of these two inputs, an vector of 4 possible signals is available.
  - Event Combinations allow a logical combination of two input signals to a common trigger.
- An **Event Trigger Logic (ETL)** per Input Channel allows the definition of the transition (edge selection, or by software) that lead to a trigger event and can also store this status. Here, the input levels of the selected signals are translated into events.
- The Trigger **Cross Connect Matrix** distributes the events and status flags to the Output Channels. Additionally, trigger signals from other modules are made available and can be combined with the local triggers.
- An **Output Gating Unit (OGU)** combines the trigger events and status information and gates the Output depending on a gating signal.

Note: An event of one Input can lead to reactions on several Outputs, or also events on several Inputs can be combined to a reaction on one Output.

4.5.1 Event Request Select Unit (ERS)

For each Input Channel x (x = 0-3), an ERSx unit handles the input selection for the associated ETLx unit. Each ERSx performs a logical combination of two signals (Ax, Bx)

to provide one combined output signal ERSxO to the associated ETLx. Input Ax can be selected from 4 options of the input vector ERU_xA[3:0] and can be optionally inverted. A similar structure exists for input Bx (selection from ERU_xB[3:0]).

In addition to the direct choice of either input Ax or Bx or their inverted values, the possible logical combinations for two selected inputs are a logical AND or a logical OR.

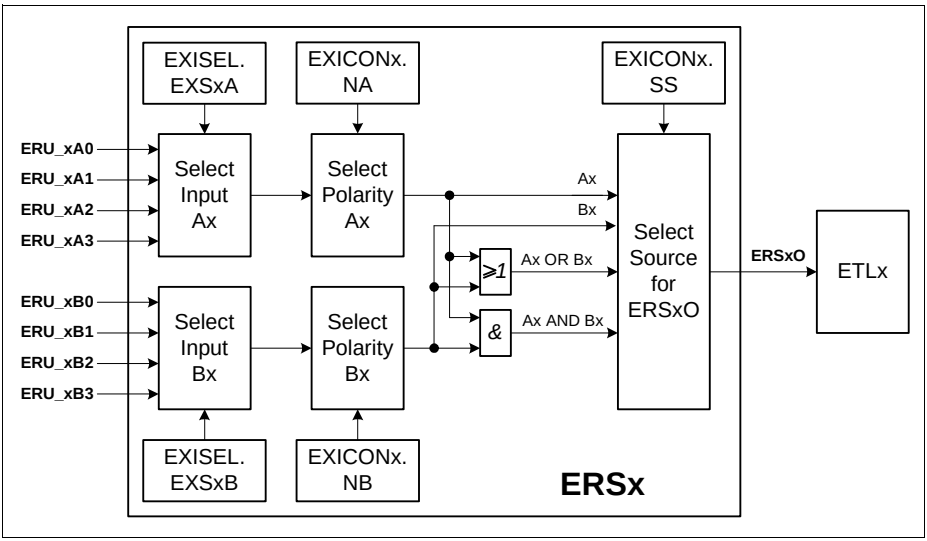


Figure 4-5 Event Request Select Unit Overview

The ERS units are controlled via register **ERU0_EXISEL** (one register for all four ERSx units) and registers EXICONx (one register for each ERSx and associated ETLx unit, e.g. **ERU0_EXICONx (x=0-3)** for Input Channel 0).

4.5.2 Event Trigger Logic (ETLx)

For each Input Channel x (x = 0-3), an event trigger logic ETLx derives a trigger event and related status information from the input ERSxO. Each ETLx is based on an edge detection block, where the detection of a rising or a falling edge can be individually enabled. Both edges lead to a trigger event if both enable bits are set (e.g. to handle a toggling input).

Each of the four ETLx units has an associated EXICONx register, that controls all options of an ETLx (the register also holds control bits for the associated ERSx unit, e.g. **ERU0_EXICONx (x=0-3)** to control ERS0 and ETL0).

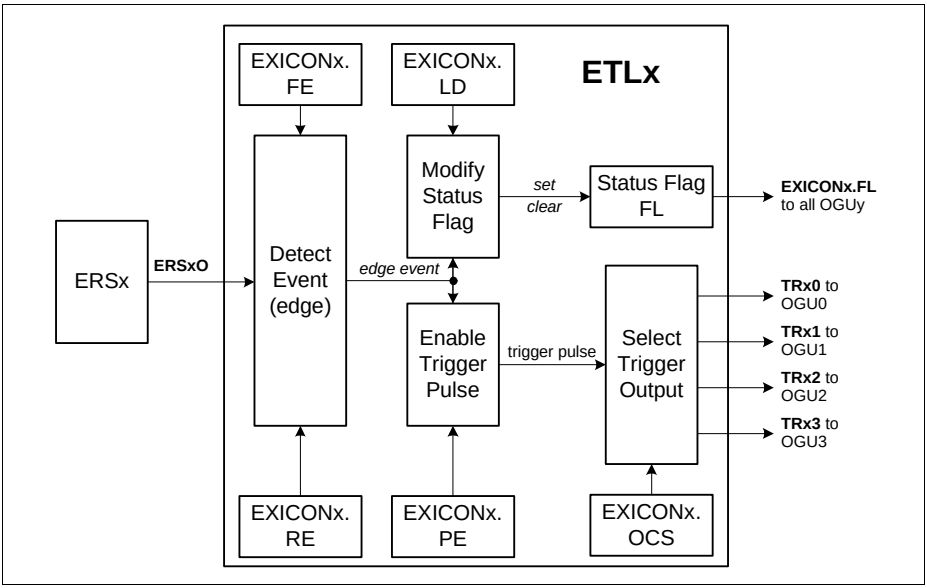


Figure 4-6 Event Trigger Logic Overview

When the selected event (edge) is detected, the status flag EXICONx.FL becomes set. This flag can also be modified by software. Two different operating modes are supported by this status flag.

It can be used as “sticky” flag, which is set by hardware when the desired event has been detected and has to be cleared by software. In this operating mode, it indicates that the event has taken place, but without indicating the actual status of the input.

In the second operating mode, it is cleared automatically if the “opposite” event is detected. For example, if only the falling edge detection is enabled to set the status flag, it is cleared when the rising edge is detected. In this mode, it can be used for pattern detection where the actual status of the input is important (enabling both edge detections is not useful in this mode).

The output of the status flag is connected to all following Output Gating Units (OGUy) in parallel (see **Figure 4-7**) to provide **pattern detection capability of all OGUy** units based on different or the same status flags.

In addition to the modification of the status flag, a trigger pulse output TRxy of ETLx can be enabled (by bit EXICONx.PE) and selected to **trigger actions in one of the OGUy** units. The target OGUy for the trigger is selected by bit field EXICON.OCS.

The trigger becomes active when the selected edge event is detected, independently from the status flag EXICONx.FL.

4.5.3 Cross Connect Matrix

The matrix shown in **Figure 4-7** distributes the trigger signals (TRxy) and status signals (EXICONx.FL) from the different ETLx units between the OGUy units. In addition, it receives peripheral trigger signals that can be OR-combined with the ETLx trigger signals in the OGUy units.

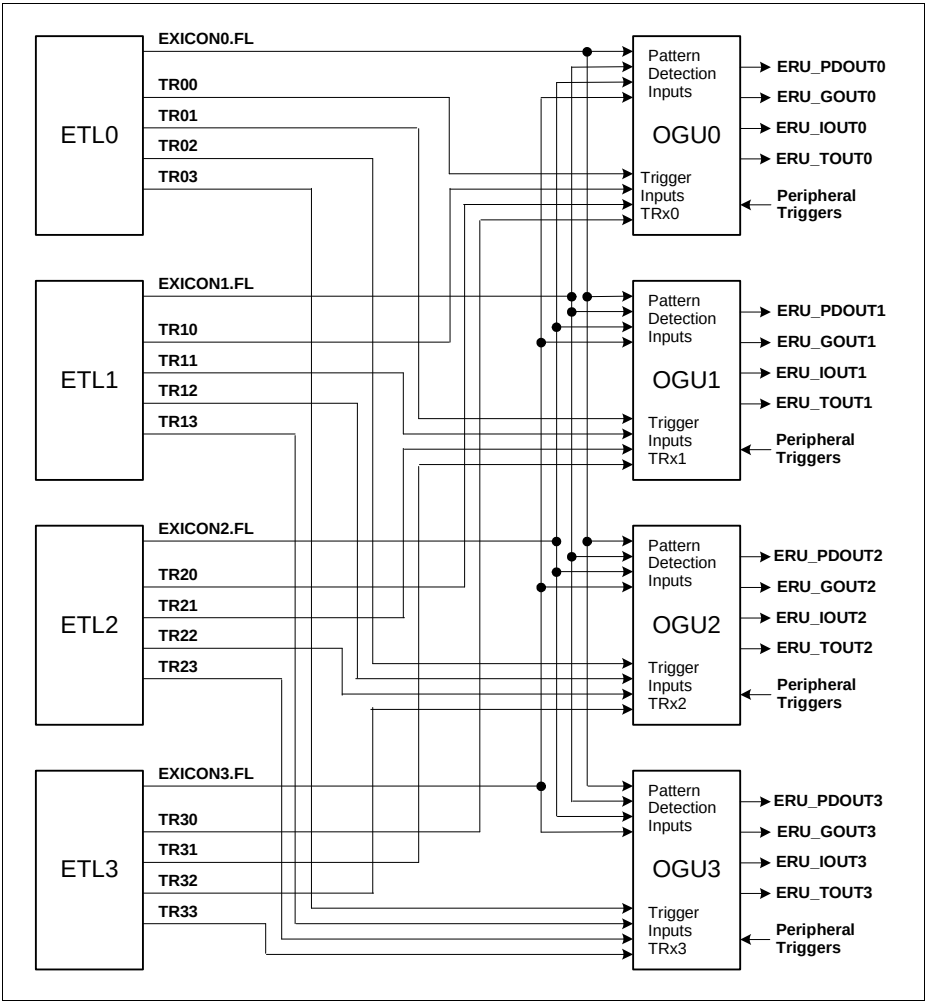


Figure 4-7 ERU Cross Connect Matrix

4.5.4 Output Gating Unit (OGUy)

Each OGUy (y = 0-3) unit combines the available trigger events and status flags from the Input Channels and distributes the results to the system. **Figure 4-8** illustrates the logic blocks within an OGUy unit. All functions of an OGUy unit are controlled by its associated EXOCONy register, e.g. **ERU0_EXOCONx (x=0-3)** for OGU0. The function of an OGUy unit can be split into two parts:

- **Trigger Combination**:
  All trigger signals TRxy from the Input Channels that are enabled and directed to OGUy, a selected peripheral-related trigger event, and a pattern change event (if enabled) are logically OR-combined.
- **Pattern Detection**:
  The status flags EXICONx.FL of the Input Channels can be enabled to take part in the pattern detection. A pattern match is detected while all enabled status flags are set.

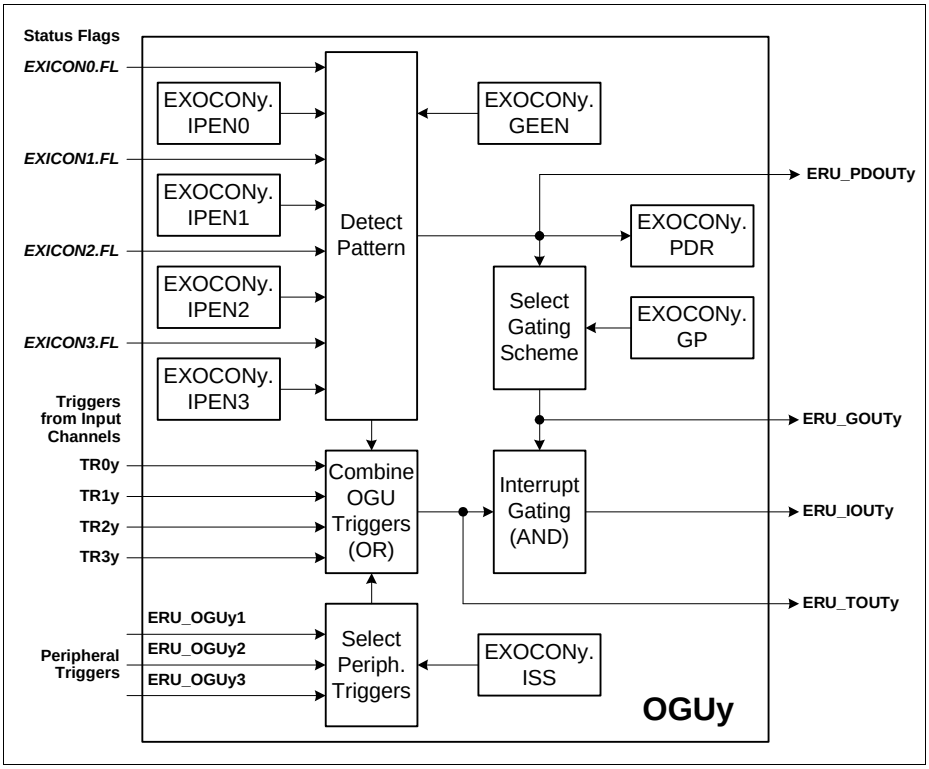


Figure 4-8 Output Gating Unit for Output Channel y

Each OGUy unit generates 4 output signals that are distributed to the system (not all of them are necessarily used):

- **ERU_PDOUTy** to directly output the pattern match information for gating purposes in other modules (pattern match = 1).
- **ERU_GOUTy** to output the pattern match or pattern miss information (inverted pattern match), or a permanent 0 or 1 under software control for gating purposes in other modules.
- **ERU_TOUTy** as combination of a peripheral trigger, a pattern detection result change event, or the ETLx trigger outputs TRxy to trigger actions in other modules.
- **ERU_IOUTy** as gated trigger output (ERU_GOUTy logical AND-combined with ERU_TOUTy) to trigger service requests (e.g. the service request generation can be gated to allow service request activation during a certain time window).

Trigger Combination

The trigger combination logically OR-combines different trigger inputs to form a common trigger ERU_TOUTy. Possible trigger inputs are:

- In each ETLx unit of the **Input Channels**, the trigger output TRxy can be enabled and the trigger event can be directed to one of the OGUy units.
- One out of three **peripheral trigger** signals per OGUy can be selected as additional trigger source. These peripheral triggers are generated by on-chip peripheral modules, such as capture/compare or timer units. The selection is done by bit field EXOCONy.ISS.
- In the case that at least one **pattern detection** input is enabled (EXOCONy.IPENx) and a change of the pattern detection result from pattern match to pattern miss (or vice-versa) is detected, a trigger event is generated to indicate a pattern detection result event (if enabled by ECOCONy.GEEN).

The trigger combination offers the possibility to program different trigger criteria for several input signals (independently for each Input Channel) or peripheral signals, and to combine their effects to a single output, e.g. to generate an service request or to start an ADC conversion. This combination capability allows the generation of a service request per OGU that can be triggered by several inputs (multitude of request sources results in one reaction).

The selection is defined by the bit fields ISS in registers **ERU0_EXOCONx (x=0-3)** (for ERU0.OGUx) and **ERU1_EXOCONy (y=0-3)** (for ERU1.OGUy).

Pattern Detection

The pattern detection logic allows the combination of the status flags of all ETLx units. Each status flag can be individually included or excluded from the pattern detection for each OGUy, via control bits EXOCONy.IPENx. The pattern detection block outputs the following pattern detection results:

- **Pattern match** (EXOCONy.PDR = 1 and ERU_PDOUTy = 1):
  A pattern match is indicated while all status flags FL that are included in the pattern detection are 1.
- **Pattern miss** (EXOCONy.PDR = 0 and ERU_PDOUTy = 0):
  A pattern miss is indicated while at least one of the status flags FL that are included in the pattern detection is 0.

In addition, the pattern detection can deliver a trigger event if the pattern detection result changes from match to miss or vice-versa (if enabled by EXOCONy.GEEN = 1). The pattern result change event is logically OR-combined with the other enabled trigger events to support service request generation or to trigger other module functions (e.g. in the ADC). The event is indicated when the pattern detection result changes and EXOCONy.PDR becomes updated.

The service request generation in the OGUy is based on the trigger ERU_TOUTy that can be gated (masked) with the pattern detection result ERU_PDOUTy. This allows an automatic and reproducible generation of service requests during a certain time window, where the request event is elaborated by the trigger combination block and the time window information (gating) is given by the pattern detection. For example, service requests can be issued on a regular time base (peripheral trigger input from capture/compare unit is selected) while a combination of input signals occurs (pattern detection based on ETLx status bits).

A programmable gating scheme introduces flexibility to adapt to application requirements and allows the generation of service requests ERU_IOUTy under different conditions:

- **Pattern match** (EXOCONy.GP = 10_B):
  A service request is issued when a trigger event occurs while the pattern detection shows a pattern match.
- **Pattern miss** (EXOCONy.GP = 11_B):
  A service request is issued when the trigger event occurs while the pattern detection shows a pattern miss.
- **Independent** of pattern detection (EXOCONy.GP = 01_B):
  In this mode, each occurring trigger event leads to a service request. The pattern detection output can be used independently from the trigger combination for gating purposes of other peripherals (independent use of ERU_TOUTy and ERU_PDOUTy with service requests on trigger events).
- **No service requests** (EXOCONy.GP = 00_B, default setting)
  In this mode, an occurring trigger event does not lead to a service request. The pattern detection output can be used independently from the trigger combination for gating purposes of other peripherals (independent use of ERU_TOUTy and ERU_PDOUTy without service requests on trigger events).

4.6 Service Request Generation

If any bit within the DLR.**DLR_OVRSTAT** register is set, a service request is flagged by setting the SCU_RAWSR.DLROVR bit.

- errors
- reaching buffer limits

Service requests can be disabled by....

A direct connection to the ADC enables the ASC to trigger an ADC conversion upon reception of a programmable data pattern.

4.7 Debug Behavior

Service request processing behavior is unchanged in debug mode.

4.8 Power, Reset and Clock

Service request processing is

- consuming power in all operating modes.
- running on f_{CPU}.
- asynchronously initialized by the system reset.

4.9 Initialization and System Dependencies

Service Requests must always be enabled at the source and at the destination. Additionally it must be checked whether it is necessary to program the ERU process and route a request.

Enabling Peripheral SRx Outputs

- Peripherals SRx outputs must be selectively enabled. This procedure depends on the individual peripheral. Please look up the section “Service Request Generation” within a peripherals chapter for details.
- Optionally ERUx must be programmed to process and route the request

Enabling External Requests

- Selected PORTS must be programmed for input
- ERUx must be programmed to process and route the external request

Note: The number of external service request inputs may be limited by the package used.

Enabling NVIC and GPDMA

Interrupt and DMA service request processing must be enabled. Please refer to the CPU and GPDMA chapters for details.

4.10 Registers

Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 4-7 Registers Address Space

Module	Base Address	End Address	Note
DLR	5000 4900$_H$	5000 49FF$_H$	
ERU0	5000 4800$_H$	5000 48FF$_H$	
ERU1	4004 4000$_H$	4004 7FFF$_H$	

Table 4-8

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
DLR Registers					
OVRSTAT	Status of DMA Service Request Overruns	000$_H$	U, PV	PV	Page 4-25
OVRCLR	Clear Status of DMA Service Request Overruns	004$_H$	U, PV	PV	Page 4-26
SRSEL0	DLR Service Request Selection 0	008$_H$	U, PV	PV	Page 4-27
LNEN	Enable DLR Line	010$_H$	U, PV	PV	Page 4-26
SRSEL1	DLR Service Request Selection 1	00C$_H$	U, PV	PV	Page 4-28
ERU Registers					
EXISEL	ERU External Input Control Selection	0000$_H$	U, PV	PV	Page 4-29
EXICON0	ERU External Input Control Selection	0010$_H$	U, PV	PV	Page 4-31
EXICON1	ERU External Input Control Selection	0014$_H$	U, PV	PV	Page 4-31
EXICON2	ERU External Input Control Selection	0018$_H$	U, PV	PV	Page 4-31

Table 4-8 (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
EXICON3	ERU External Input Control Selection	$001C_H$	U, PV	PV	Page 4-31
EXOCON0	ERU Output Control Register	0020_H	U, PV	PV	Page 4-33
EXOCON1	ERU Output Control Register	0024_H	U, PV	PV	Page 4-33
EXOCON2	ERU Output Control Register	0028_H	U, PV	PV	Page 4-33
EXOCON3	ERU Output Control Register	$002C_H$	U, PV	PV	Page 4-33

4.10.1 DLR Registers

DLR_OVRSTAT

The DLR_OVRSTAT register is used to track status of GPDMA service request overruns. Upon overrun detection, additionally a service request flag is set in the SCU_RAWSR.DLROVR bit.

DLR_OVRSTAT
Overrun Status **(00_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				LN11	LN10	LN9	LN8	LN7	LN6	LN5	LN4	LN3	LN2	LN1	LN0
r				rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

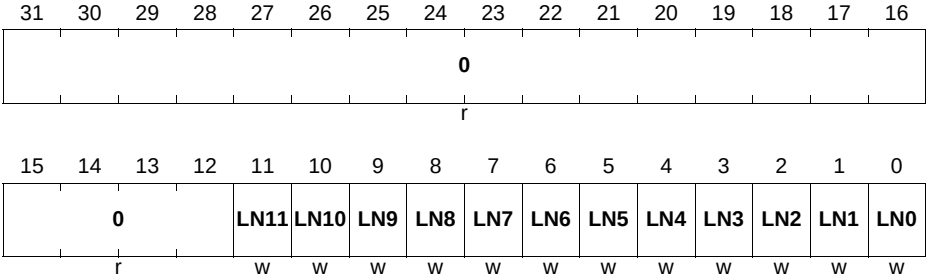
Field	Bits	Type	Description
LNx (x = 0-11)	x	rh	**Line x Overrun Status** Set if an overrun occurred on this line.

Field	Bits	Type	Description
0	[31:12]	r	**Reserved** Read as 0; should be written with 0.

DLR_OVRCLR

The DLR_OVRCLR register is used to clear the DLR_OVRSTAT register bits.

DLR_OVRCLR
Overrun Clear **(04_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
LNx **(x = 0-11)**	x	w	**Line x Overrun Status Clear** Clears the corresponding bit in the DLR_OVRSTAT register when set to 1.
0	[31:12]	r	**Reserved** Read as 0; should be written with 0.

DLR_LNEN

The DLR_LNEN register is used to enable each individual DLR line and to reset a previously stored and pending service request.

DLR_LNEN
Line Enable **(10_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				LN11	LN10	LN9	LN8	LN7	LN6	LN5	LN4	LN3	LN2	LN1	LN0
r				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
LNx (x = 0-11)	x	rw	**Line x Enable** 0_B Disables the line 1_B Enables the line and resets a pending request
0	[31:12]	r	**Reserved** Read as 0; should be written with 0.

DLR_SRSELx

The DLR_SRSEL**x** register**s** are used to select the service request source used to trigger a DMA transfer.

DLR_SRSEL0
Service Request Selection 0

(08_H) **Reset Value: 0000 0000$_H$**

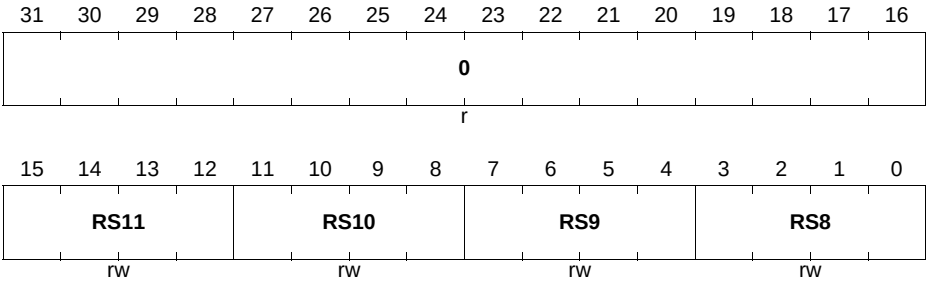
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RS7				RS6				RS5				RS4			
rw				rw				rw				rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RS3				RS2				RS1				RS0			
rw				rw				rw				rw			

Field	Bits	Type	Description
RSx (x = 0-7)	[x*4+3: x*4]	rw	**Request Source for Line x** The request source according to **Table 4-6** is selected for DMA line x.

DLR_SRSEL1
Service Request Selection 1

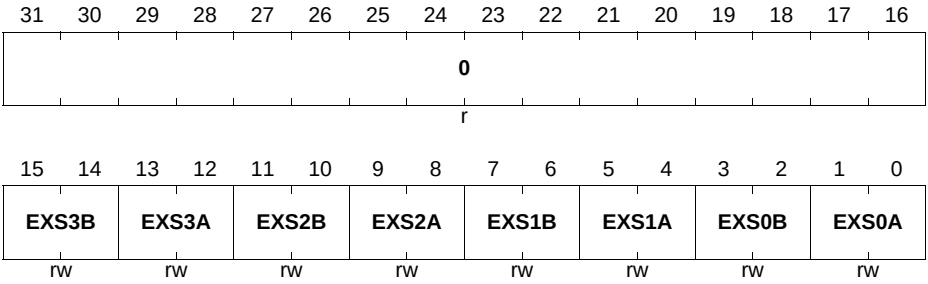
(0C$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
RS8	[3:0]	rw	**Request Source for Line 8** The request source according to **Table 4-6** is selected for DMA line x.
RS9	[7:4]	rw	**Request Source for Line 9** The request source according to **Table 4-6** is selected for DMA line x.
RS10	[11:8]	rw	**Request Source for Line 10** The request source according to **Table 4-6** is selected for DMA line x.
RS11	[15:12]	rw	**Request Source for Line 11** The request source according to **Table 4-6** is selected for DMA line x.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

4.10.2 ERU Registers

ERU0_EXISEL
Event Input Select **(00_H)** **Reset Value: 0000 0000_H**
ERU1_EXISEL
Event Input Select **(0000_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
EXS0A	[1:0]	rw	**Event Source Select for A0 (ERS0)** This bit field defines which input is selected for A0. 00_B Input ERU_0A0 is selected 01_B Input ERU_0A1 is selected 10_B Input ERU_0A2 is selected 11_B Input ERU_0A3 is selected
EXS0B	[3:2]	rw	**Event Source Select for B0 (ERS0)** This bit field defines which input is selected for B0. 00_B Input ERU_0B0 is selected 01_B Input ERU_0B1 is selected 10_B Input ERU_0B2 is selected 11_B Input ERU_0B3 is selected
EXS1A	[5:4]	rw	**Event Source Select for A1 (ERS1)** This bit field defines which input is selected for A1. 00_B Input ERU_1A0 is selected 01_B Input ERU_1A1 is selected 10_B Input ERU_1A2 is selected 11_B Input ERU_1A3 is selected

Field	Bits	Type	Description
EXS1B	[7:6]	rw	**Event Source Select for B1 (ERS1)** This bit field defines which input is selected for B1. 00_B Input ERU_1B0 is selected 01_B Input ERU_1B1 is selected 10_B Input ERU_1B2 is selected 11_B Input ERU_1B3 is selected
EXS2A	[9:8]	rw	**Event Source Select for A2 (ERS2)** This bit field defines which input is selected for A2. 00_B Input ERU_2A0 is selected 01_B Input ERU_2A1 is selected 10_B Input ERU_2A2 is selected 11_B Input ERU_2A3 is selected
EXS2B	[11:10]	rw	**Event Source Select for B2 (ERS2)** This bit field defines which input is selected for B2. 00_B Input ERU_2B0 is selected 01_B Input ERU_2B1 is selected 10_B Input ERU_2B2 is selected 11_B Input ERU_2B3 is selected
EXS3A	[13:12]	rw	**Event Source Select for A3 (ERS3)** This bit field defines which input is selected for A3. 00_B Input ERU_3A0 is selected 01_B Input ERU_3A1 is selected 10_B Input ERU_3A2 is selected 11_B Input ERU_3A3 is selected
EXS3B	[15:14]	rw	**Event Source Select for B3 (ERS3)** This bit field defines which input is selected for B3. 00_B Input ERU_3B0 is selected 01_B Input ERU_3B1 is selected 10_B Input ERU_3B2 is selected 11_B Input ERU_3B3 is selected
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

ERU0_EXICONx (x=0-3)
Event Input Control x

(10_H + 4*x) **Reset Value: 0000 0000_H**

ERU1_EXICONy (y=0-3)
Event Input Control y

(0010_H + 4*y) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				NB	NA	SS		FL	OCS			FE	RE	LD	PE
r				rw	rw	rw		rwh	rw			rw	rw	rw	rw

Field	Bits	Type	Description
PE	0	rw	**Output Trigger Pulse Enable for ETLx** This bit enables the generation of an output trigger pulse at TRxy when the selected edge is detected (set condition for the status flag FL). 0_B The trigger pulse generation is disabled 1_B The trigger pulse generation is enabled
LD	1	rw	**Rebuild Level Detection for Status Flag for ETLx** This bit selects if the status flag FL is used as “sticky” bit or if it rebuilds the result of a level detection. 0_B The status flag FL is not cleared by hardware and is used as “sticky” bit. Once set, it is not influenced by any edge until it becomes cleared by software. 1_B The status flag FL rebuilds a level detection of the desired event. It becomes automatically set with a rising edge if RE = 1 or with a falling edge if FE = 1. It becomes automatically cleared with a rising edge if RE = 0 or with a falling edge if FE = 0.

Field	Bits	Type	Description
RE	2	rw	**Rising Edge Detection Enable ETLx** This bit enables/disables the rising edge event as edge event as set condition for the status flag FL or as possible trigger pulse for TRxy. 0_B A rising edge is not considered as edge event 1_B A rising edge is considered as edge event
FE	3	rw	**Falling Edge Detection Enable ETLx** This bit enables/disables the falling edge event as edge event as set condition for the status flag FL or as possible trigger pulse for TRxy. 0_B A falling edge is not considered as edge event 1_B A falling edge is considered as edge event
OCS	[6:4]	rw	**Output Channel Select for ETLx Output Trigger Pulse** This bit field defines which Output Channel OGUy is targeted by an enabled trigger pulse TRxy. 000_B Trigger pulses are sent to OGU0 001_B Trigger pulses are sent to OGU1 010_B Trigger pulses are sent to OGU2 011_B Trigger pulses are sent to OGU3 Others: Reserved, do not use this combination
FL	7	rwh	**Status Flag for ETLx** This bit represents the status flag that becomes set or cleared by the edge detection. 0_B The enabled edge event has not been detected 1_B The enabled edge event has been detected
SS	[9:8]	rw	**Input Source Select for ERSx** This bit field defines which logical combination is taken into account as ERSxO. 00_B Input A without additional combination 01_B Input B without additional combination 10_B Input A OR input B 11_B Input A AND input B
NA	10	rw	**Input A Negation Select for ERSx** This bit selects the polarity for the input A. 0_B Input A is used directly 1_B Input A is inverted

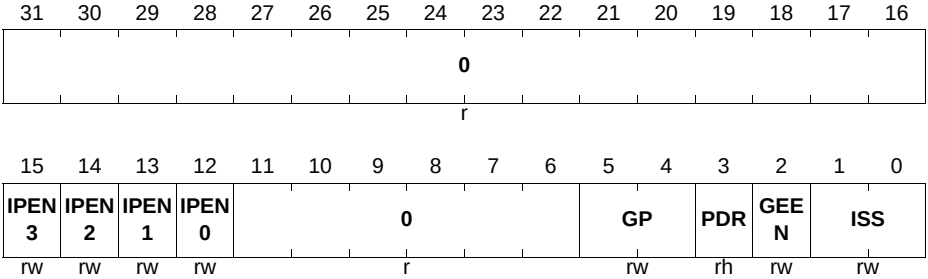
Field	Bits	Type	Description
NB	11	rw	**Input B Negation Select for ERSx** This bit selects the polarity for the input B. 0_B Input B is used directly 1_B Input B is inverted
0	[31:12]	r	**Reserved** Read as 0; should be written with 0.

ERU0_EXOCONx (x=0-3)
Event Output Trigger Control x

(20_H + 4*x) **Reset Value: 0000 0008$_H$**

ERU1_EXOCONy (y=0-3)
Event Output Trigger Control y

(0020_H + 4*y) **Reset Value: 0000 0008$_H$**



Field	Bits	Type	Description
ISS	[1:0]	rw	**Internal Trigger Source Selection** This bit field defines which input is selected as peripheral trigger input for OGUy. 00_B The peripheral trigger function is disabled 01_B Input ERU_OGUy1 is selected 10_B Input ERU_OGUy2 is selected 11_B Input ERU_OGUy3 is selected
GEEN	2	rw	**Gating Event Enable** Bit GEEN enables the generation of a trigger event when the result of the pattern detection changes from match to miss or vice-versa. 0_B The event detection is disabled 1_B The event detection is enabled

Field	Bits	Type	Description
PDR	3	rh	**Pattern Detection Result Flag** This bit represents the pattern detection result. 0_B A pattern miss is detected 1_B A pattern match is detected
GP	[5:4]	rw	**Gating Selection for Pattern Detection Result** This bit field defines the gating scheme for the service request generation (relation between the OGU output ERU_PDOUTy and ERU_GOUTy). 00_B ERU_GOUTy is always disabled and ERU_IOUTy can not be activated 01_B ERU_GOUTy is always enabled and ERU_IOUTy becomes activated with each activation of ERU_TOUTy 10_B ERU_GOUTy is equal to ERU_PDOUTy and ERU_IOUTy becomes activated with an activation of ERU_TOUTy while the desired pattern is detected (pattern match PDR = 1) 11_B ERU_GOUTy is inverted to ERU_PDOUTy and ERU_IOUTy becomes activated with an activation of ERU_TOUTy while the desired pattern is not detected (pattern miss PDR = 0)
IPENx (x = 0-3)	12+x	rw	**Pattern Detection Enable for ETLx** Bit IPENx defines whether the trigger event status flag EXICONx.FL of ETLx takes part in the pattern detection of OGUy. 0_B Flag EXICONx.FL is excluded from the pattern detection 1_B Flag EXICONx.FL is included in the pattern detection
0	[31:16], [11:6]	r	**Reserved** Read as 0; should be written with 0.

4.11 Interconnects

This section describes how the ERU0 and ERU1 modules are connected within the XMC4500 system.

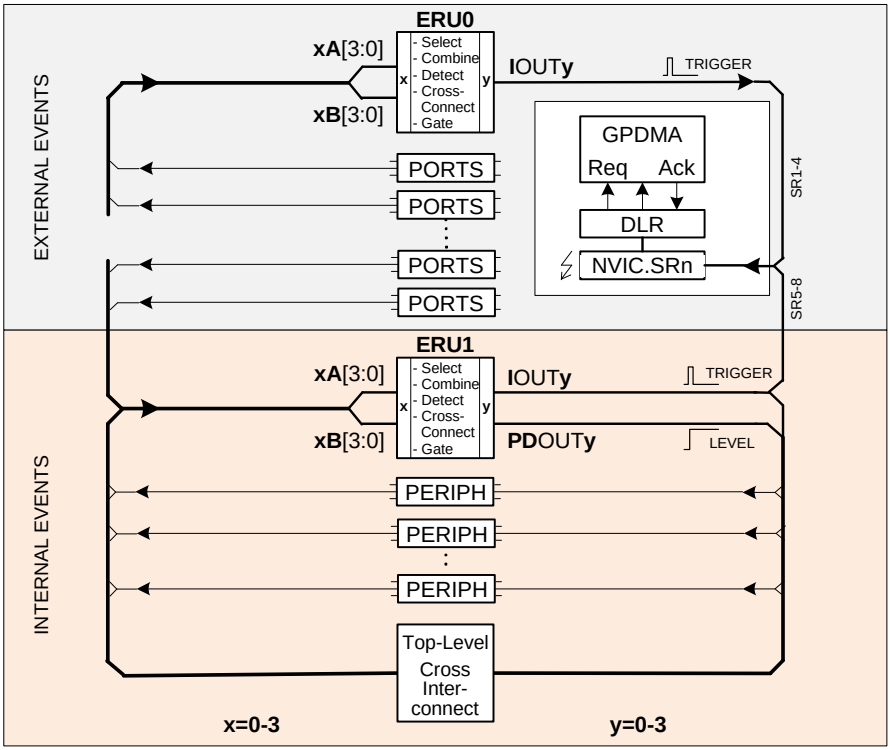


Figure 4-9 ERU Interconnects Overview

4.11.1 ERU0 Connections

The following table shows the ERU0 connections. Please refer to the ports chapter for details about PORTS connections.

Table 4-9 ERU0 Pin Connections

Global Inputs/Outputs	Connected To	I/O	Description
ERU0.0A0	PORTS	I	
ERU0.0A1	PORTS	I	
ERU0.0A2	PORTS	I	
ERU0.0A3	SCU.G0ORCOUT6	I	

Table 4-9 ERU0 Pin Connections (cont'd)

Global Inputs/Outputs	Connected To	I/O	Description
ERU0.0B0	PORTS	I	
ERU0.0B1	PORTS	I	
ERU0.0B2	PORTS	I	
ERU0.0B3	PORTS	I	
ERU0.1A0	PORTS	I	
ERU0.1A1	SCU.HIB_SR0	I	
ERU0.1A2	PORTS	I	
ERU0.1A3	SCU.G0ORCOUT7	I	
ERU0.1B0	PORTS	I	
ERU0.1B1	SCU.HIB_SR1	I	
ERU0.1B2	PORTS	I	
ERU0.1B3	PORTS	I	
ERU0.2A0	PORTS	I	
ERU0.2A1	PORTS	I	
ERU0.2A2	PORTS	I	
ERU0.2A3	SCU.G1ORCOUT6	I	
ERU0.2B0	PORTS	I	
ERU0.2B1	PORTS	I	
ERU0.2B2	PORTS	I	
ERU0.2B3	PORTS	I	
ERU0.3A0	PORTS	I	
ERU0.3A1	PORTS	I	
ERU0.3A2	PORTS	I	
ERU0.3A3	SCU.G1ORCOUT7	I	
ERU0.3B0	PORTS	I	
ERU0.3B1	PORTS	I	
ERU0.3B2	PORTS	I	
ERU0.3B3	PORTS	I	
ERU0.OGU01	0	I	
ERU0.OGU02	0	I	

Table 4-9 ERU0 Pin Connections (cont'd)

Global Inputs/Outputs	Connected To	I/O	Description
ERU0.OGU03	1	I	
ERU0.OGU11	0	I	
ERU0.OGU12	0	I	
ERU0.OGU13	1	I	
ERU0.OGU21	0	I	
ERU0.OGU22	0	I	
ERU0.OGU23	1	I	
ERU0.OGU31	0	I	
ERU0.OGU32	0	I	
ERU0.OGU33	1	I	
ERU0.PDOUT0	not connected	O	
ERU0.GOUT0	not connected	O	
ERU0.TOUT0	not connected	O	
ERU0.IOUT0	NVIC.ERU0.SR0 DLR	O	
ERU0.PDOUT1	not connected	O	
ERU0.GOUT1	not connected	O	
ERU0.TOUT1	not connected	O	
ERU0.IOUT1	NVIC.ERU0.SR1 DLR	O	
ERU0.PDOUT2	not connected	O	
ERU0.GOUT2	not connected	O	
ERU0.TOUT2	not connected	O	
ERU0.IOUT2	NVIC.ERU0.SR2 DLR	O	
ERU0.PDOUT3	not connected	O	
ERU0.GOUT3	not connected	O	
ERU0.TOUT3	not connected	O	
ERU0.IOUT3	NVIC.ERU0.SR3 DLR	O	

4.11.2 ERU1 Connections

The following table shows the ERU1 connections. Please refer to the ports chapter for details about PORTS connections.

Table 4-10 ERU1 Pin Connections

Global Inputs/Outputs	Connected To	I/O	Description
ERU1.0A0	PORTS	I	
ERU1.0A1	POSIF0.SR1	I	
ERU1.0A2	CCU40.ST0	I	
ERU1.0A3	DAC.SIGN_0	I	
ERU1.0B0	PORTS	I	
ERU1.0B1	CCU80.ST0	I	
ERU1.0B2	VADC.G0BFL3	I	
ERU1.0B3	ERU1.IOUT3	I	
ERU1.1A0	PORTS	I	
ERU1.1A1	POSIF0.SR1	I	
ERU1.1A2	CCU40.ST1	I	
ERU1.1A3	ERU1.IOUT2	I	
ERU1.1B0	PORTS	I	
ERU1.1B1	CCU80.ST1	I	
ERU1.1B2	VADC.G1BFL3	I	
ERU1.1B3	ERU1.IOUT2	I	
ERU1.2A0	PORTS	I	
ERU1.2A1	POSIF1.SR1	I	
ERU1.2A2	CCU40.ST2	I	
ERU1.2A3	DAC.SIGN_1	I	
ERU1.2B0	PORTS	I	
ERU1.2B1	CCU80.ST2	I	
ERU1.2B2	VADC.G0BFL3	I	
ERU1.2B3	not connected	I	
ERU1.3A0	PORTS	I	
ERU1.3A1	POSIF1.SR1	I	

Table 4-10 ERU1 Pin Connections (cont'd)

Global Inputs/Outputs	Connected To	I/O	Description
ERU1.3A2	CCU40.ST3	I	
ERU1.3A3	not connected	I	
ERU1.3B0	PORTS	I	
ERU1.3B1	CCU80.ST3	I	
ERU1.3B2	VADC.G1BFL3	I	
ERU1.3B3	not connected	I	
ERU1.OGU01	VADC.C0SR0	I	
ERU1.OGU02	CCU40.ST0	I	
ERU1.OGU03	1	I	
ERU1.OGU11	VADC.C0SR1	I	
ERU1.OGU12	CCU41.ST0	I	
ERU1.OGU13	1	I	
ERU1.OGU21	VADC.C0SR2	I	
ERU1.OGU22	CCU81.ST3A	I	
ERU1.OGU23	1	I	
ERU1.OGU31	VADC.C0SR3	I	
ERU1.OGU32	CCU81.ST3B	I	
ERU1.OGU33	1	I	

Table 4-10 ERU1 Pin Connections (cont'd)

Global Inputs/Outputs	Connected To	I/O	Description
ERU1.PDOUT0	CCU40.IN0J CCU41.IN0J CCU42.IN0J CCU43.IN0J CCU40.IN1D CCU40.IN2D CCU40.IN3D CCU41.IN1D CCU41.IN2D CCU41.IN3D CCU42.IN1D CCU42.IN2D CCU42.IN3D CCU43.IN1D CCU43.IN2D CCU43.IN3D CCU80.IN0J CCU80.IN1J CCU80.IN2J CCU80.IN3J VADC.G0REQGTO VADC.G1REQGTO VADC.G2REQGTO VADC.G3REQGTO VADC.BGREQGTO DSD.ITR0A DSD.ITR1A DSD.ITR2A DSD.ITR3A POSIF0.IN0D POSIF1.IN0D	O	
ERU1.GOUT0	not connected	O	
ERU1.TOUT0	not connected	O	

Table 4-10 ERU1 Pin Connections (cont'd)

Global Inputs/Outputs	Connected To	I/O	Description
ERU1.IOUT0	CCU4x.IN0K CCU8x.IN0G VADC.G0REQTRM VADC.G1REQTRM VADC.G2REQTRM VADC.G3REQTRM VADC.BGREQTRM CCU40.MCLKA CCU41.MCLKA CCU42.MCLKA CCU43.MCLKA CCU80.MCLKA CCU81.MCLKA NVIC.ERU1.SR0 POSIF0.EWHEB POSIF1.EWHEB	O	
ERU1.PDOUT1	CCU40.IN1J CCU41.IN1J CCU42.IN1J CCU43.IN1J CCU81.IN0I CCU81.IN1I CCU81.IN2I CCU81.IN3I CCU40.IN0D CCU41.IN0D CCU42.IN0D CCU43.IN0D VADC.G0REQGTP VADC.G1REQGTP VADC.BGREQGTP DSD.ITR0B DSD.ITR1B DSD.ITR2B DSD.ITR3B POSIF0.IN1D POSIF1.IN1D	O	
ERU1.GOUT1	not connected	O	

Table 4-10 ERU1 Pin Connections (cont'd)

Global Inputs/Outputs	Connected To	I/O	Description
ERU1.TOUT1	not connected	O	
ERU1.IOUT1	CCU40.IN1K CCU41.IN1K CCU42.IN1K CCU43.IN1K CCU80.IN1G CCU81.IN1G VADC.G0REQTRN VADC.G1REQTRN VADC.BGREQTRN CCU40.MCLKB CCU41.MCLKB CCU42.MCLKB CCU43.MCLKB CCU80.MCLKB CCU81.MCLKB NVIC.ERU1.SR1 POSIF0.EWHEC POSIF1.EWHEC	O	
ERU1.PDOUT2	CCU40.IN2J CCU41.IN2J CCU42.IN2J CCU43.IN2J CCU80.IN2F CCU81.IN2F DSD.ITR0C DSD.ITR1C DSD.ITR2C DSD.ITR3C DSD.SGNA VADC.G2REQGTP VADC.G3REQGTP POSIF0.IN2D POSIF1.IN2D	O	
ERU1.GOUT2	not connected	O	
ERU1.TOUT2	not connected	O	

Table 4-10 ERU1 Pin Connections (cont'd)

Global Inputs/Outputs	Connected To	I/O	Description
ERU1.IOUT2	CCU40.IN2K CCU41.IN2K CCU42.IN2K CCU43.IN2K CCU80.IN2G CCU81.IN2G VADC.G2REQTRN VADC.G3REQTRN ERU1.1A3 ERU1.1B3 NVIC.ERU1.SR2 POSIF0.MSETF POSIF1.MSETF	O	
ERU1.PDOUT3	CCU40.IN3J CCU41.IN3J CCU42.IN3J CCU43.IN3J CCU80.IN3F CCU81.IN3F DSD.ITR0D DSD.ITR1D DSD.ITR2D DSD.ITR3D DSD.SGNB	O	
ERU1.GOUT3	not connected	O	
ERU1.TOUT3	not connected	O	
ERU1.IOUT3	CCU40.IN3K CCU41.IN3K CCU42.IN3K CCU43.IN3K CCU80.IN3G CCU81.IN3G ERU1.0B3 NVIC.ERU1.SR3	O	

5 General Purpose DMA (GPDMA)

The GPDMA is a highly configurable DMA controller, that allows high-speed data transfers between peripherals and memories. Complex data transfers can be done with minimal intervention of the processor, keeping this way the CPU resources free for other operations.

Extensive support for the microcontroller peripherals, like A/D and D/A converters, Timers, Communication Interfaces (USIC) via the GPDMA, unload the CPU and increase the efficiency and parallelism, for a high arrangement of real-time applications.

Table 5-1 Abbreviations table

GPDMAx	General Purpose DMA instance x
SCU	System Control Unit
DLR	DMA Line Router
f_{DMA}	GPDMA clock frequency

5.1 Overview

The GPDMA module enables hardware or software controlled data transfers between all microcontroller modules with the exclusion of those modules which provide built-in DMA functionality (USB and Ethernet).

Each GPDMA module contains a dedicated set of highly programmable channels, that can accommodate several type of peripheral-to-peripheral, peripheral-to-memory and memory-to-memory transfers.

The link between a highly programmable channel allocation and channel priority, gives a high benefit for applications that need high efficiency and parallelism.

The built-in fast DMA request handling together with the flexible peripheral configuration, enables the implementation of very demanding application software loops.

5.1.1 Features

The GPDMA component includes the following features.

General

- Bus interfaces
  - 1 Bus master interface per each DMA unit
  - 1 Bus slave interface per each DMA unit
- Channels
  - One GPDMA0 unit with 8 channels
  - One GPDMA1 unit with 4 channels
  - Programmable channel priority

- Transfers
  - Support for memory-to-memory, memory-to-peripheral, peripheral-to-memory, and peripheral-to-peripheral DMA transfers

Channels

All channels can be programmed for the following transfer modes

- DMA triggered by software or selectable from hardware service request sources
- Programmable source and destination addresses
- Address increment, decrement, or no change

Channels 0 and 1 of GPDMA0 can be programmed for the following transfer modes

- Multi-block transfers achieved through:
  - Linked Lists (block chaining)
  - Auto-reloading of channel registers
  - Contiguous address between blocks
- Independent source and destination selection of multi-block transfer type
- Scatter/Gather - source and destination areas do not need to be in a contiguous memory space

The GPDMA0 channels 0 and 1 provide a FIFO of 32 Bytes (eight 32-bit entries). These channels can be used to execute burst transfers up to a fixed length burst size of 8. The remaining channels FIFO size is 8 Bytes.

Channel Control

- Programmable source and destination for each channel
- Programmable burst transaction size for each channel
- Programmable enable and disable of DMA channel
- Support for disabling channel without data loss
- Support for suspension of DMA operation
- Support for ERROR response
- Bus locking - programmable over transaction, block, or DMA transfer level
- Channel locking - programmable over transaction, block, or DMA transfer level
- Optional writeback of the Channel Control register at the end of every block transfer

Interrupts

- Combined and separate interrupt requests
- Interrupt generation on:
  - DMA transfer (multi-block) completion
  - Block transfer completion
  - Single and burst transaction completion
  - Error condition
- Support of interrupt enabling and masking

5.1.2 GPDMA Block Diagram

Figure 5-1 shows the following functional groupings of the main interfaces to the GPDMA block:

- DMA hardware request interface (DLR)
- Up to twelve channels
- Arbiter
- Bus Master and Slave interfaces

One channel of the GPDMA is required for each source/destination pair. The master interface reads the data from a source peripheral and writes it to a destination peripheral. Two transfers are required for each DMA data transfer; this is also known as a dual-access transfer.

The Ethernet and USB peripherals have an internal DMA controller, that enables the peripheral to act as a Master on the system.

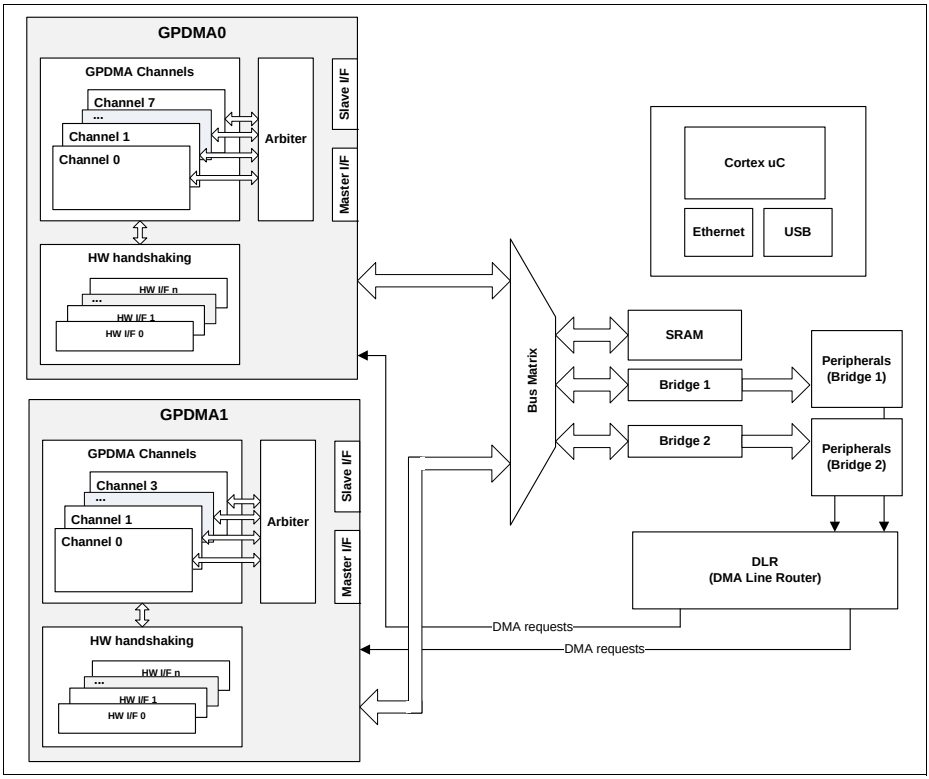


Figure 5-1 GPDMA Block Diagram

5.2 Functional Description

This chapter describes the functional details of the GPDMA component. On Section 5.2.1 a description of used terms throughout the chapters is given.

5.2.1 Basic Definitions

The following terms are concise definitions of the DMA concepts used throughout this chapter:

- **Source peripheral** - Device on a AHB layer from which the GPDMA reads data; the GPDMA then stores the data in the channel FIFO. The source peripheral teams up with a destination peripheral to form a channel.
- **Destination peripheral** - Device to which the GPDMA writes the stored data from the FIFO (previously read from the source peripheral).
- **Memory** - Source or destination that is always "ready" for a DMA transfer and does not require a handshaking interface to interact with the GPDMA.
- **Channel** - Read/write data path between a source peripheral and a destination peripheral, that occurs through the channel FIFO. If the source peripheral is not memory, then a source handshaking interface is assigned to the channel. If the destination peripheral is not memory, then a destination handshaking interface is assigned to the channel. Source and destination handshaking interfaces can be assigned dynamically by programming the channel registers.
- **Master interface** - GPDMA is a master on the AHB, reading data from the source and writing it to the destination over the bus. Each channel has to arbitrate for the master interface.
- **Slave interface** - The AHB interface over which the GPDMA is programmed.
- **Handshaking interface** - A set of signals or software registers that conform to a protocol and handshake between the GPDMA and source or destination peripheral in order to control transferring a single or burst transaction between them. This interface is used to request, acknowledge, and control a GPDMA transaction. A channel can receive a request through one of two types of handshaking interface: software, or peripheral interrupt.
  - **Software handshaking interface**- Uses software registers to control transferring a single or burst transaction between the GPDMA and the source or destination peripheral. This mode is useful for interfacing an existing peripheral to the GPDMA without modifying it. For more information about this interface, refer to Section 5.2.6.
  - **Peripheral interrupt handshaking interface** - Simple use of the hardware handshaking interface. In this mode, the interrupt line from the peripheral is tied to the dma request input of the hardware handshaking interface;
- **Flow controller** - Device that determines the length of a DMA block transfer and terminates it.

- **Transfer hierarchy** - **Figure 5-2** illustrates the hierarchy between GPDMA transfers, block transfers, transactions (single or burst), and AHB transfers (single or burst) for non-memory peripherals. **Figure 5-3** shows the transfer hierarchy for memory.

Note: Note that for memory peripherals, there is no DMA Transaction Level.

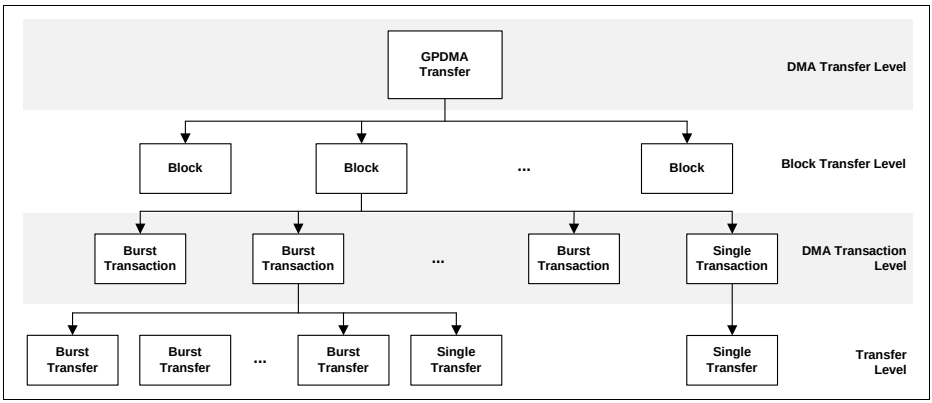


Figure 5-2 GPDMA Transfer Hierarchy for Non-Memory Peripherals

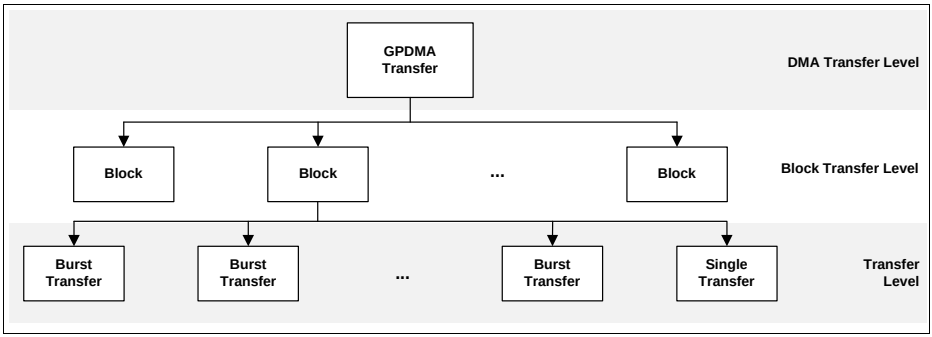


Figure 5-3 GPDMA Transfer Hierarchy for Memory

- **Block** - Block of GPDMA data, the amount of which is the block length and is determined by the flow controller. For transfers between the GPDMA and memory, a block is broken directly into a sequence of bursts and single transfers. For transfers between the GPDMA and a non-memory peripheral, a block is broken into a sequence of GPDMA transactions (single and bursts). These are in turn broken into a sequence of AHB transfers.
- **Transaction** - Basic unit of a GPDMA transfer, as determined by either the hardware or software handshaking interface. A transaction is relevant only for transfers

between the GPDMA and a source or destination peripheral if the peripheral is a non-memory device. There are two types of transactions:
  – **Single transaction** - is always converted to a single AHB transfer.
  – **Burst transaction** - Length of a burst transaction is programmed into the GPDMA. The burst transaction is converted into a sequence of AHB fixed length bursts and AHB single transfers. GPDMA executes each burst transfer by performing incremental bursts that are no longer than the maximum burst size set; the only type of burst in this kind of transaction is incremental. The burst transaction length is under program control and normally bears some relationship to the FIFO sizes in the GPDMA and in the source and destination peripherals.
- **DMA transfer** - Software controls the number of blocks in a GPDMA transfer. Once the DMA transfer has completed, the hardware within the GPDMA disables the channel and can generate an interrupt to signal the DMA transfer completion. You can then reprogram the channel for a new DMA transfer.
  – **Single-block DMA transfer** - Consists of a single block.
  – **Multi-block DMA transfer** - DMA transfer may consist of multiple GPDMA blocks. Multi-block DMA transfers are supported through block chaining (linked list pointers), auto-reloading channel registers, and contiguous blocks. The source and destination can independently select which method to use
    - **Linked lists (block chaining)** - Linked list pointer (LLP) points to the location in system memory where the next linked list item (LLI) exists. The LLI is a set of registers that describes the next block (block descriptor) and an LLP register. The GPDMA fetches the LLI at the beginning of every block when block chaining is enabled.
    LLI accesses are always 32-bit accesses aligned to 32-bit boundaries and cannot be changed or programmed to anything other than 32-bit, even if the AHB master interface of the LLI supports more than a 32-bit data width.
    - **Auto-reloading** - GPDMA automatically reloads the channel registers at the end of each block to the value when the channel was first enabled.
    - **Contiguous blocks** - Address between successive blocks is selected to be a continuation from the end of the previous block.
- **Scatter** - Relevant to destination transfers within a block. The destination address is incremented or decremented by a programmed amount when a scatter boundary is reached. The number of AHB transfers between successive scatter boundaries is under software control.
- **Gather** - Relevant to source transfers within a block. The source address is incremented or decremented by a programmed amount when a gather boundary is reached. The number of AHB transfers between successive gather boundaries is under software control.
- **Channel locking** - Software can program a channel to keep the AHB master interface by locking arbitration of the master AHB interface for the duration of a DMA transfer, block, or transaction (single or burst).

- **Bus locking** - Software can program a channel to maintain control of the AHB bus for the duration of a DMA transfer, block, or transaction (single or burst). At minimum, channel locking is asserted during bus locking.
- **FIFO mode** - Special mode to improve bandwidth. When enabled, the channel waits until the FIFO is less than half full to fetch the data from the source peripheral, and waits until the FIFO is greater than or equal to half full in order to send data to the destination peripheral. Because of this, the channel can transfer the data using bursts, which eliminates the need to arbitrate for the AHB master interface in each single AHB transfer. When this mode is not enabled, the channel waits only until the FIFO can transmit or accept a single AHB transfer before it requests the master bus interface.

5.2.2 Block Flow Controller and Transfer Type

The device that controls the length of a block is known as the flow controller.

- The block size should be programmed into the **CTL**.BLOCK_TS field.

The **CTL**.TT_FC field indicates the transfer type and flow controller for that channel. **Table 5-2** lists valid transfer types and flow controller combinations.

Table 5-2 Transfer Types and Flow Control Combinations

Transfer Type	Flow Controller
Memory to Memory	GPDMA
Memory to Peripheral	GPDMA
Peripheral to Memory	GPDMA
Peripheral to Peripheral	GPDMA

5.2.3 Handshaking Interface

Handshaking interfaces are used at the transaction level to control the flow of single or burst transactions.

The peripheral uses the handshaking interface to indicate to the GPDMA that it is ready to transfer data over the AHB bus.

A non-memory peripheral can request a DMA transfer through the GPDMA using one of two types of handshaking interfaces:

- Hardware
- Software

Software selects between the hardware or software handshaking interface on a per-channel basis. Software handshaking is accomplished through memory-mapped registers, while hardware handshaking is accomplished using a dedicated handshaking interface.

*Note: Throughout the remainder of this chapter, references to both source and destination hardware handshaking interfaces assume an active-high interface (refer to CFGx.SRC(DST)_HS_POL bits in the Channel Configuration register, **CFG**). When active-low handshaking interfaces are used, then the active level and edge are reversed from that of an active-high interface.*

*Note: Source and destination peripherals can independently select the handshaking interface type; that is, hardware or software handshaking. For more information, refer to the CFGx.HS_SEL_SRC and CFGx.HS_SEL_DST parameters in the **CFG** register.*

5.2.4 Basic Interface Definitions

The following definitions are used in this chapter:

- Source single transaction size in bytes, see **(5.1)**
- Source burst transaction size in bytes, see **(5.2)**
- Destination single transaction size in bytes, see **(5.3)**
- Destination burst transaction size in bytes, see **(5.4)**
- Block size in bytes:
  - GPDMA as flow controller: With the GPDMA as the flow controller, the processor programs the GPDMA with the number of data items (block size) of source transfer width (**CTL**.SRC_TR_WIDTH) to be transferred by the GPDMA in a block transfer; this is programmed into the **CTL**.BLOCK_TS field. Therefore, the total number of bytes to be transferred in a block is defined by **(5.5)**

$$src\_single\_size\_bytes = CTLx.SRC\_TR\_WIDTH/8 \quad (5.1)$$

$$src\_burst\_size\_bytes = CTLx.SRC\_MSIZE * src\_single\_size\_bytes \quad (5.2)$$

$$dst\_single\_size\_bytes = CTLx.DST\_TR\_WIDTH/8 \quad (5.3)$$

$$dst\_burst\_size\_bytes = CTLx.DEST\_MSIZE * dst\_single\_size\_bytes \quad (5.4)$$

$$blk\_size\_bytes\_dma = CTLx.BLOCK\_TS * src\_single\_size\_bytes \quad (5.5)$$

Note: In the above equations, references to CTLx.SRC_MSIZE, CTLx.DEST_MSIZE, CTLx.SRC_TR_WIDTH, and CTLx.DST_TR_WIDTH refer to the decoded values of the parameters; for example, CTLx.SRC_MSIZE = 001_B decodes to 4, and CTLx.SRC_TR_WIDTH = 010_B decodes to 32 bits.

5.2.5 Memory Peripherals

Figure 5-3 shows the DMA transfer hierarchy of the GPDMA for a memory peripheral. There is no handshaking interface with the GPDMA, and therefore the memory peripheral can never be a flow controller. Once the channel is enabled, the transfer proceeds immediately without waiting for a transaction request.

5.2.6 Software Handshaking

When the slave peripheral requires the GPDMA to perform a DMA transaction, it communicates this request by sending a service request to the interrupt controller. The interrupt service routine then uses the software registers, detailed in **Section 5.6.4**, to initiate and control a DMA transaction. This group of software registers is used to implement the software handshaking interface.

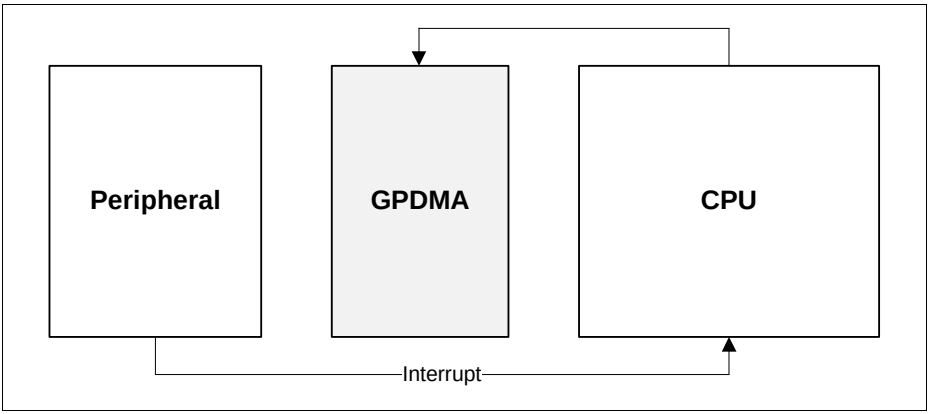


Figure 5-4 Software Controlled DMA Transfers

The HS_SEL_SRC/HS_SEL_DST bit in the **CFG** channel configuration register must be set, to enable software handshaking.

Software Controlled DMA Transfers

- Program and enable channel through **Section 5.6.2**
- After interrupt, initiate and control DMA transaction between peripherals GPDMA through **Section 5.6.4**

The software handshaking registers are:

- **REQSRCREG** - source software transaction request
- **REQDSTREG** - destination software transaction request
- **SGLREQSRCREG** - single source transaction request
- **SGLREQDSTREG** - single destination transaction request

- **LSTSRCREG** - last source transaction request
- **LSTDSTREG** - last destination transaction request

5.2.7 Handshaking Interface

The GPDMA tries to efficiently transfer the data using as little of the bus bandwidth as possible. Generally, the GPDMA tries to transfer the data using burst transactions and, where possible, fill or empty the channel FIFO in single bursts - provided that the software has not limited the burst length.

The GPDMA can also lock the arbitration for the master bus interface so that a channel is permanently granted the master bus interface. Additionally, the GPDMA can assert the lock signal to lock the system arbiter. For more information, refer to **Section 5.2.11.1**.

Before describing the handshaking interface operation, the following sections define the terms "Single Transaction Region" and "Early-Terminated Burst Transaction."

5.2.7.1 Single Transaction Region

There are cases where a DMA block transfer cannot be completed using only burst transactions. Typically this occurs when the block size is not a multiple of the burst transaction length. In these cases, the block transfer uses burst transactions up to the point where the amount of data left to complete the block is less than the amount of data in a burst transaction. At this point, the GPDMA samples the "single" status flag and completes the block transfer using single transactions.

The peripheral asserts a single status flag to indicate to the GPDMA that there is enough data or space to complete a single transaction from or to the source/destination peripheral.

*Note: For hardware handshaking, the single status flag is a signal on the hardware handshaking interface and handled automatically by the hardware (refer to **Section 5.2.7.3**). For software handshaking, the single status flag is one of the software handshaking interface registers (refer to **Section 5.2.7.4**).*

The Single Transaction Region is the time interval where the GPDMA uses single transactions to complete the block transfer (burst transactions are exclusively used outside this region).

*Note: Burst transactions can also be used in this region. For more information, refer to **Section 5.2.7.2**.*

- **Rules for entering Single Transaction Regions:**
  - The source peripheral enters the Single Transaction Region when the number of bytes left to complete in the source block transfer is less than `src_burst_size_bytes`.

If **(5.6)** is fulfilled then the source never enters this region, and the source block uses only burst transactions.

– The destination peripheral enters the Single Transaction Region when the number of bytes left to complete in the destination block transfer is less than `dst_burst_size_bytes`.
If **(5.7)** is fulfilled then the destination never enters this region, and the destination block uses only burst transactions.

$$blk\_size\_bytes/src\_burst\_size\_bytes = integer \quad (5.6)$$

$$blk\_size\_bytes/dst\_burst\_size\_bytes = integer \quad (5.7)$$

Note: The above conditions cause a peripheral to enter the Single Transaction Region. When the peripheral is outside the Single Transaction Region, then the GPDMA responds to only burst transaction requests.

5.2.7.2 Early-Terminated Burst Transaction

When a source or destination peripheral is in the Single Transaction Region, a burst transaction can still be requested. However, src_burst_size_bytes/ dst_burst_size_bytes is greater than the number of bytes left to complete in the source/destination block transfer at the time that the burst transaction is triggered. In this case, the burst transaction is started and "early-terminated" at block completion without transferring the programmed amount of data, that is, src_burst_size_bytes or dst_burst_size_bytes, but only the amount required to complete the block transfer. An Early-Terminated Burst Transaction occurs between the GPDMA and the peripheral only when the peripheral is not the flow controller.

5.2.7.3 Hardware Handshaking

Figure 5-5 illustrates the hardware handshaking interface between a peripheral (whether a destination or source) and the GPDMA.

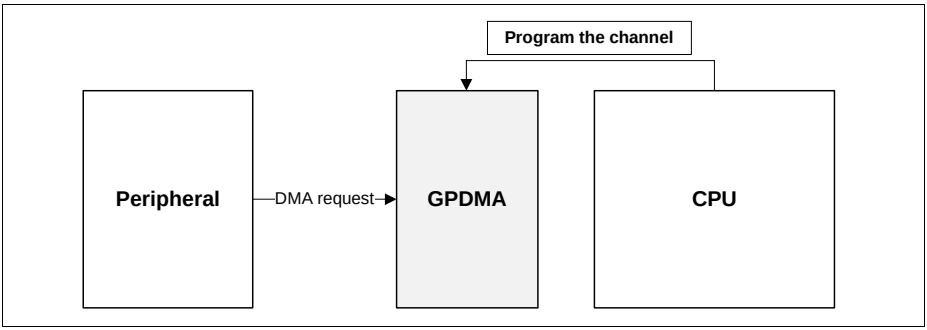


Figure 5-5 Hardware Handshaking Interface

The interface illustrated in **Figure 5-5** is the simplified version of the hardware handshaking interface. In this mode:

- The interrupt line from the peripheral is tied to the dma request input.

The interrupt line from the peripheral is tied to the dma request line (programming the DLR is needed), as shown in **Figure 5-5**.

The handshaking loop is as follows:

1. Peripheral generates an interrupt that asserts the dma request.
2. GPDMA completes the burst transaction and generates an end-of-burst transaction interrupt, IntSrcTran/IntDstTran. Interrupts must be enabled and the transaction complete interrupt unmasked.
3. The interrupt service routine clears the interrupt in the peripheral so that the dma request is de-asserted.

5.2.7.4 Software Handshaking

When the peripheral is not the flow controller, then the last transaction registers - **LSTSRCREG** and **LSTDSTREG** - are not used, and the values in these registers are ignored.

Operation - Peripheral Not In Single Transaction Region

Writing a 1 to the **REQSRCREG**[x]/**REQDSTREG**[x] register is always interpreted as a burst transaction request, where x is the channel number. However, in order for a burst transaction request to start, software must write a 1 to the **SGLREQSRCREG**[x]/ **SGLREQDSTREG**[x] register.

You can write a 1 to the SGLREQSRCREG[x]/SGLREQDSTREG[x] and REQSRCREG[x]/REQDSTREG[x] registers in any order, but both registers must be asserted in order to initiate a burst transaction. Upon completion of the burst transaction,

the hardware clears the SGLREQSRCREG[x]/SGLREQDSTREG[x] and REQSRCREG[x]/ REQDSTREG[x] registers.

Operation - Peripheral In Single Transaction Region

Writing a 1 to the **SGLREQSRCREG/SGLREQDSTREG** initiates a single transaction. Upon completion of the single transaction, both the SGLREQSRCREG/SGLREQDSTREG and **REQSRCREG/REQDSTREG** bits are cleared by hardware. Therefore, writing a 1 to the REQSRCREG/REQDSTREG is ignored while a single transaction has been initiated, and the requested burst transaction is not serviced.

Again, writing a 1 to the REQSRCREG/REQDSTREG register is always a burst transaction request. However, in order for a burst transaction request to start, the corresponding channel bit in the SGLREQSRCREG/SGLREQDSTREG must be asserted. Therefore, to ensure that a burst transaction is serviced in this region, you must write a 1 to the REQSRCREG/REQDSTREG before writing a 1 to the SGLREQSRCREG/SGLREQDSTREG register. If the programming order is reversed, a single transaction is started instead of a burst transaction. The hardware clears both the REQSRCREG/REQDSTREG and the SGLREQSRCREG/SGLREQDSTREG registers after the burst transaction request completes. When a burst transaction is initiated in the Single Transaction Region, then the block completes using an Early-Terminated Burst Transaction.

Software can poll the relevant channel bit in the SGLREQSRCREG/SGLREQDSTREG and REQSRCREG/REQDSTREG registers. When both are 0, then either the requested burst or single transaction has completed. Alternatively, the IntSrcTran or IntDstTran interrupts can be enabled and unmasked in order to generate an interrupt when the requested source or destination transaction has completed.

Note: The transaction-complete interrupts are triggered when both single and burst transactions are complete. The same transaction-complete interrupt is used for both single and burst transactions.

5.2.8 Single Transactions

Being the GPDMA the flow controller of the DMA transactions, a burst transaction can always be performed if:

- `blk_size_bytes_dma/src_burst_size_bytes = integer`

If that condition cannot be met, then a series of burst transactions followed by single transactions is needed to complete the source/destination block transfer.

5.2.9 Setting Up Transfers

Transfers are set up by programming fields of the **CTL** and **CFG** registers for that channel. As shown in **Figure 5-2**, a single block is made up of numerous transactions - single and burst - which are in turn composed of AHB transfers. A peripheral requests a transaction through the DLR to the GPDMA (for more information, refer to **Section 5.2.3**).

Table 5-3 lists the parameters that are investigated in the following examples. The effects of these parameters on the flow of the block transfer are highlighted.

Table 5-3 Parameters Used in Transfer Examples

Parameter	Description
CTLx.TT_FC	Transfer type and flow control
CTLx.BLOCK_TS	Block transfer size
CTLx.SRC_TR_WIDTH	Source transfer width
CTLx.DST_TR_WIDTH	Destination transfer width
CTLx.SRC_MSIZE	Source burst transaction length
CTLx.DEST_MSIZE	Destination burst transaction length
CFGx.MAX_ABRST	Maximum AMBA burst length
CFGx.FIFO_MODE	FIFO mode select
CFGx.FCMODE	Flow-control mode

5.2.9.1 Transfer Operation

The following examples show the effect of different settings of each parameter from **Table 5-3** on a DMA block transfer. In all examples, it is assumed that no bursts are early-terminated by the system arbiter, unless otherwise stated.

The GPDMA is programmed with the number of data items that are to be transferred for each burst transaction request, **CTL**.SRC_MSIZE /**CTL**.DEST_MSIZE. Similarly, the width of each data item in the transaction is set by the **CTL**.SRC_TR_WIDTH and **CTL**.DST_TR_WIDTH fields.

Example 1

Scenario: Example block transfer when the GPDMA is the flow controller. This example is the same for both software and hardware handshaking interfaces. **Table 5-4** lists the DMA parameters for this example (as an exemple the FIFO depth is taken as 16 bytes).

Table 5-4 Parameters in Transfer Operation - Example 1

Parameter	Description
CTLx.TT_FC = 011_B	Peripheral-to-peripheral transfer with GPDMA as flow controller
CTLx.BLOCK_TS = 12	-
CTLx.SRC_TR_WI DTH = 010_B	32 bits
CTLx. DST_TR_WI DTH = 010_B	32 bits
CTLx.SRC_MSIZE = 001_B	Source burst transaction length = 4
CTLx.DEST_MSIZE = 001_B	Destination burst transaction length = 4
CFGx.MAX_ABRST = 0_B	No limit on maximum AMBA burst length

Using **(5.5)**, a total of 48 bytes are transferred in the block (that is blk_size_bytes_dma = 48). As shown in **Figure 5-6**, this block transfer consists of three bursts of length 4 from the source, interleaved with three bursts, again of length 4, to the destination.

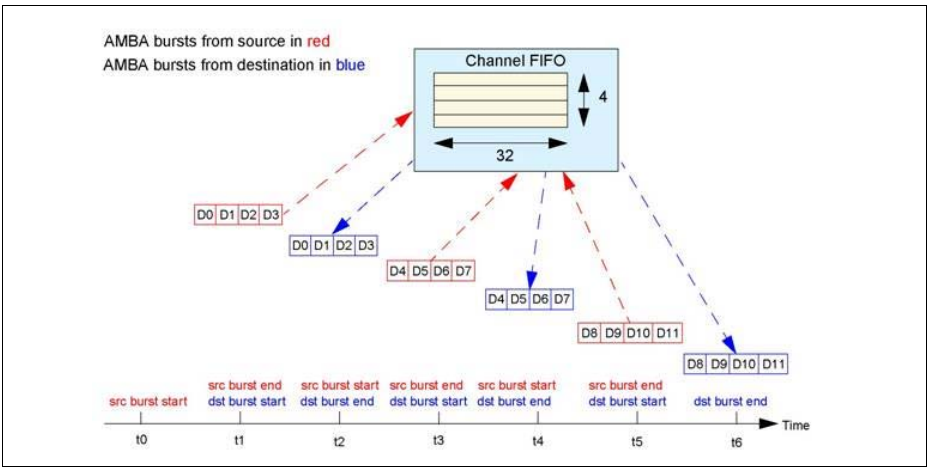


Figure 5-6 Breakdown of Block Transfer

The channel FIFO is alternatively filled by a burst from the source and emptied by a burst to the destination until the block transfer has completed, as shown in **Figure 5-7**.

Time t1	Time t2	Time t3	Time t4	Time t5	Time t6
D3	Empty	D7	Empty	D11	Empty
D2	Empty	D6	Empty	D10	Empty
D1	Empty	D5	Empty	D9	Empty
D0	Empty	D4	Empty	D8	Empty

Figure 5-7 Channel FIFO Contents

Burst transactions are completed in one burst. Additionally, because (8) and (9) are both true, neither the source or destination peripherals enter their Single Transaction Region at any stage throughout the DMA transfer, and the block transfer from the source and to the destination consists of burst transactions only.

Example 2

Scenario: Effect of the maximum AMBA burst length, CFGx.MAX_ABRST. This example is the same for both software and hardware handshaking interfaces.

If the CFGx.MAX_ABRST = 2 parameter and all other parameters are left unchanged from Example 1, **Table 5-4**, then the block transfer would look like that shown in **Figure 5-8**.

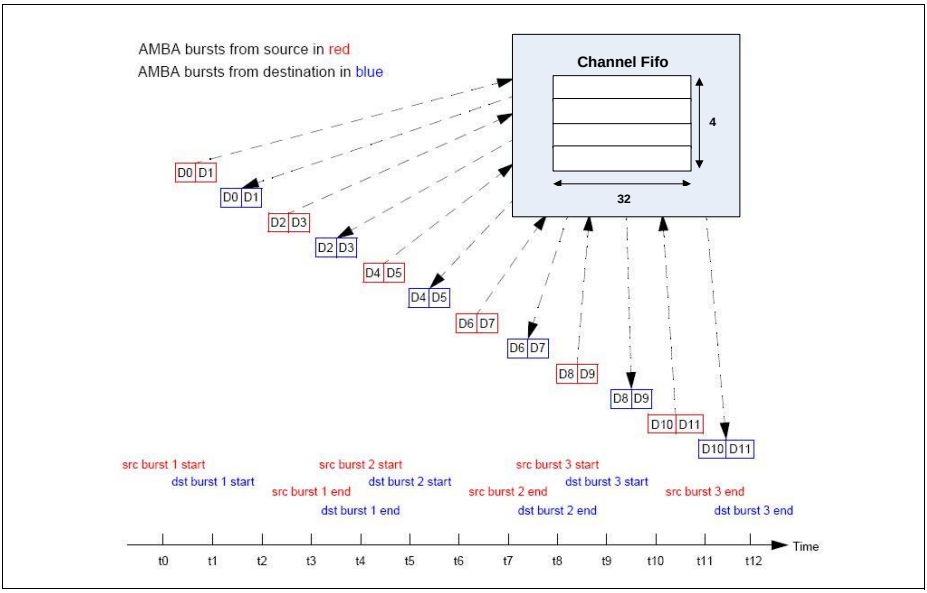


Figure 5-8 Breakdown of Block Transfer where max_abrst = 2, Case 1

The channel FIFO is alternatively half filled by a burst from the source, and then emptied by a burst to the destination until the block transfer has completed; this is illustrated in **Figure 5-9**.

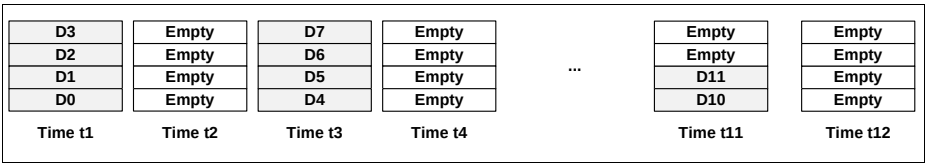


Figure 5-9 Channel FIFO Contents

In this example block transfer, each source or destination burst transaction is made up of two bursts, each of length 2. As **Figure 5-9** illustrates, the top two channel FIFO locations are redundant for this block transfer. However, this is not the general case. The block transfer could proceed as indicated in **Figure 5-10**.

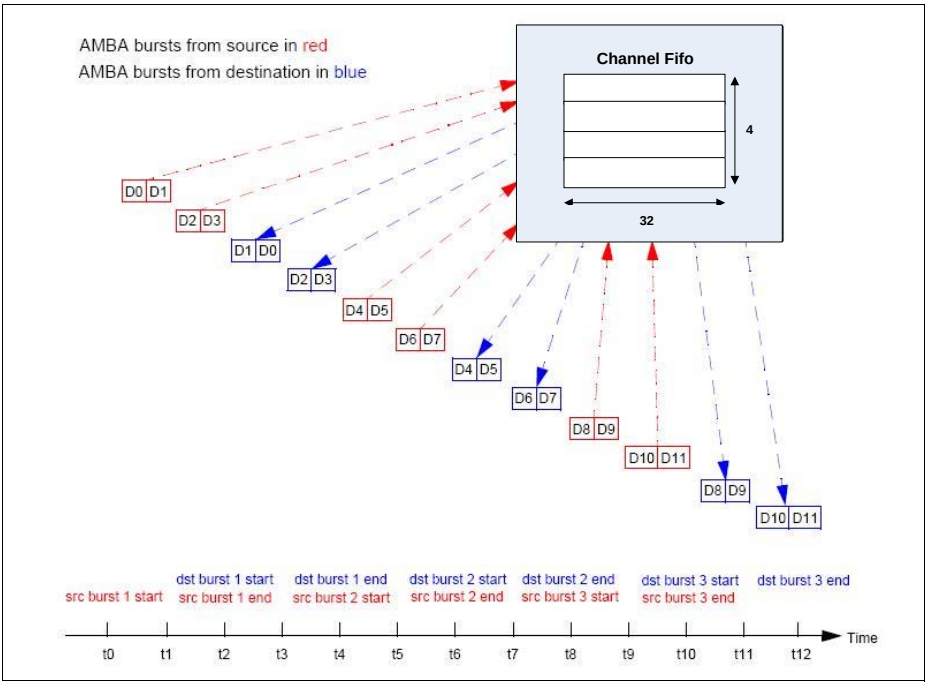


Figure 5-10 Breakdown of Block Transfer where max_abrst = 2, Case 2

This depends on the timing of the source and destination transaction requests, relative to each other. **Figure 5-11** illustrates the channel FIFO status for **Figure 5-10**.

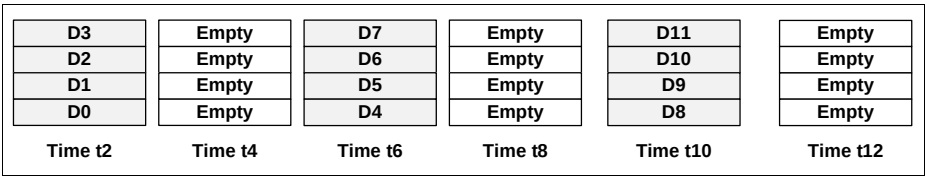


Figure 5-11 Channel FIFO Contents

Recommendation

To allow a burst transaction to complete in a single burst, the following should be true:

```
CFGx.MAX_ABRST >= max(src_burst_size_bytes,
                      dst_burst_size_bytes)
```

Adhering to the above recommendation results in a reduced number of bursts per block, which in turn results in improved bus utilization and lower latency for block transfers.

Limiting a burst to a maximum length prevents the GPDMA from saturating the AHB bus when the system arbiter is configured to only allow changing of the grant signals to bus masters at the end of an undefined length burst. It also prevents a channel from saturating a GPDMA master bus interface.

Example 3

Scenario: Source peripheral enters Single Transaction Region; the GPDMA is the flow controller.

This example is the same for both hardware and software handshaking and demonstrates how a block from the source can be completed using a series of single transactions. It also demonstrates how the watermark level that triggers a burst request in the source peripheral can be dynamically adjusted so that the block transfer from the source completes with an Early-terminated Burst Transaction. **Table 5-5** lists the parameters used in this example (as an example the FIFO depth was considered as 16 bytes).

Table 5-5 Parameters in Transfer Operation - Example 4

Parameter	Comment
CTLx.TT_FC = 011_B	Peripheral-to-peripheral transfer with GPDMA as flow controller
CTLx.BLOCK_TS = 12	-
CTLx.SRC_TR_WI DTH = 010_B	32 bits
CTLx. DST_TR_WI DTH = 010_B	32 bits
CTLx.SRC_MSIZE = 010_B	Source burst transaction length = 8

Table 5-5 Parameters in Transfer Operation - Example 4 (cont'd)

Parameter	Comment
CTLx.DEST_MSIZE = 001_B	Destination burst transaction length = 4
CFGx.MAX_ABRST = 0_B	No limit on maximum AMBA burst length

In this case, **CTL**.BLOCK_TS is not a multiple of the source burst transaction length, **CTL**.SRC_MSIZE, so near the end of a block transfer from the source, the amount of data left to be transferred is less than src_burst_size_bytes.

In this example, the block size is a multiple of the destination burst transaction length:

```
blk_size_bytes_dma/dst_burst_size_bytes = 48/16 = integer
```

The destination block is made up of three burst transactions to the destination and does not enter the Single Transaction Region.

The block size is not a multiple of the source burst transaction length:

```
blk_size_bytes_dma/src_burst_size_bytes = 48/32 != integer
```

Consider the case where the watermark level that triggers a source burst request in the source peripheral is equal to **CTL**.SRC_MSIZE = 8; that is, eight entries or more need to be in the source peripheral FIFO in order to trigger a burst request. **Figure 5-12** shows how this block transfer is broken into burst and single transactions, and bursts and single transfers.

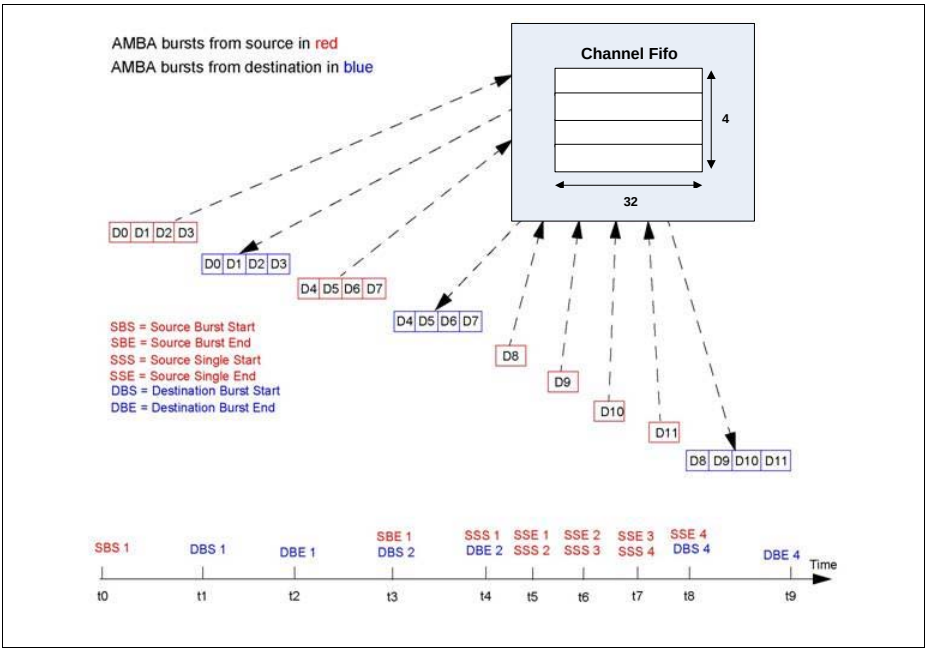


Figure 5-12 Breakdown of Block Transfer

Figure 5-13 shows the status of the source FIFO at various times throughout the source block transfer.

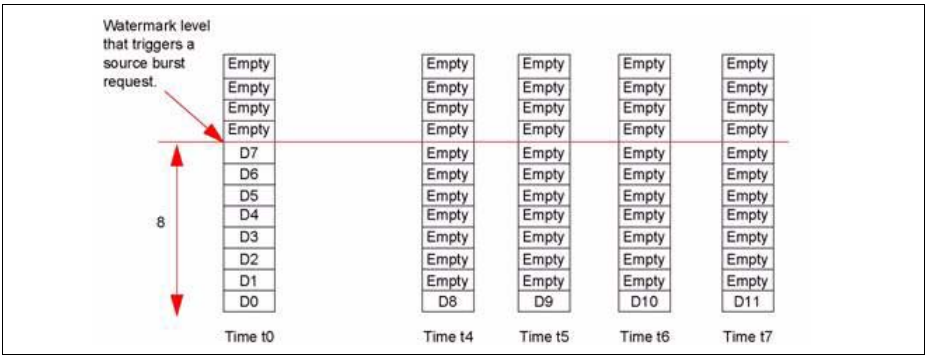


Figure 5-13 Source FIFO Contents

As shown in **Figure 5-14**, if the GPDMA does not perform single transactions, the source FIFO contains four entries at time t1. However, the source has no more data to send. Therefore, if the watermark level remains at 8 (at time t1, Case A in **Figure 5-14**), the watermark level is never reached and a new burst request is never triggered.

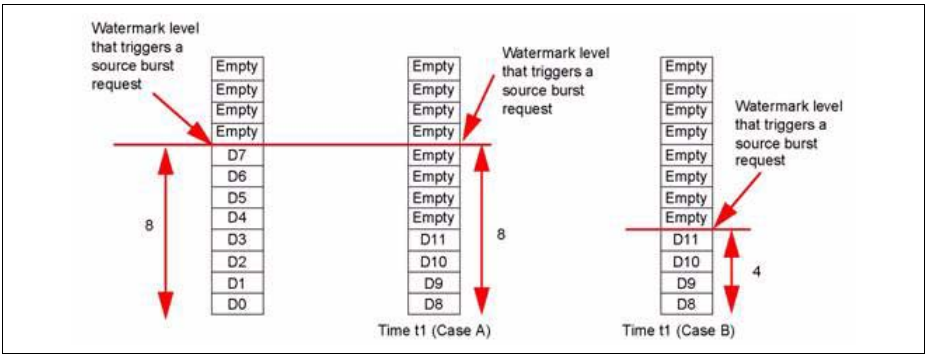


Figure 5-14 Source FIFO Contents where Watermark Level is Dynamically Adjusted

The source peripheral, not knowing the length of a block and only able to request burst transactions, sits and waits for the FIFO level to reach a watermark level before requesting a new burst transaction request. This region, where the amount of data left to transfer in the source block is less than src_burst_size_bytes, is known as the Single Transaction Region.

In the Single Transaction Region, the GPDMA performs single transactions from the source peripheral until the source block transfer has completed. In this example, the GPDMA completes the source block transfer using four single transactions from the source.

Now consider Case B in **Figure 5-14**, where the source peripheral can dynamically adjust the watermark level that triggers a burst transaction request near the end of a block. After the first source burst transaction completes, the source peripheral recognizes that it has only four data items left to complete in the block and adjusts the FIFO watermark level that triggers a burst transaction to 4. This triggers a burst request, and the block completes using a burst transaction. However, **CTL**.SRC_MSIZE = 8, and there are only four data items left to transfer in the source block. The GPDMA terminates the last source burst transaction early and fetches only four of the eight data items in the last source burst transaction. This is called an Early-Terminated Burst Transaction.

Observation: Under certain conditions, it is possible to hardcode dma_single from the source peripheral to an inactive level (hardware handshaking). Under the same conditions, it is possible for software to complete a source block transfer without initiating single transactions from the source. For more information, refer to **Section 5.2.8**.

Example 4

Scenario: The destination peripheral enters the Single Transaction Region while the GPDMA is the flow controller. This example also demonstrates how the GPDMA channel FIFO is flushed at the end of a block transfer to the destination; this example is the same for both hardware and software handshaking.

Consider the case with the parameters set to values listed in **Table 5-6** (as an example the FIFO depth was considered as 32 bytes).

Table 5-6 Parameters in Transfer Operation - Example 5

Parameter	Comment
CTLx.TT_FC = 011_B	Peripheral-to-peripheral transfer with GPDMA as flow controller
CTLx.BLOCK_TS = 44	-
CTLx.SRC_TR_WI DTH = 000_B	8 bit
CTLx. DST_TR_WI DTH = 011_B	64bit
CTLx.SRC_MSIZE = 001_B	Source burst transaction length = 4
CTLx.DEST_MSIZE = 001_B	Destination burst transaction length = 4
CFGx.MAX_ABRST = 0_B	No limit on maximum AMBA burst length

In this example, the block size is a multiple of the source burst transaction length:

```
blk_size_bytes_dma/src_burst_size_bytes = (44 * 1)/4 = 11
                                        = integer
```

The source block transfer is completed using only burst transactions, and the source does not enter the Single Transaction Region.

The block size is not a multiple of the destination burst transaction length:

```
blk_size_bytes_dma/dst_burst_size_bytes 44/32 != integer
```

So near the end of the block transfer to the destination, the amount of data left to be transferred is less than dst_burst_size_bytes and the destination enters the Single Transaction Region.

Figure 5-15 shows one way in which the block transfer to the destination can occur.

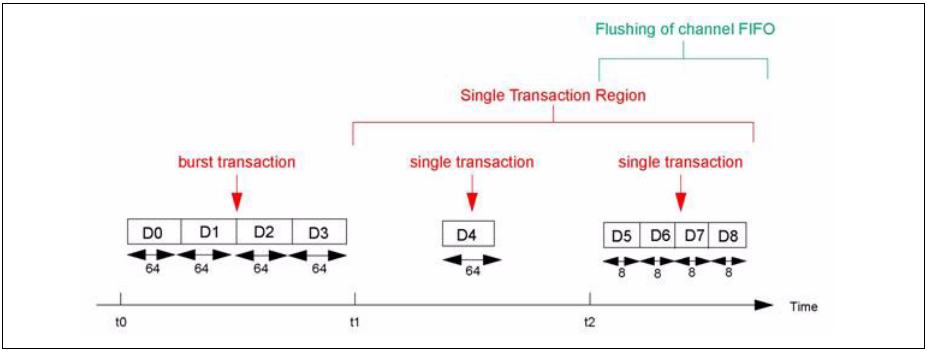


Figure 5-15 Block Transfer to Destination

After the first 32 bytes (dst_burst_size_bytes = 32) of the destination burst transaction have been transferred to the destination, there are 12 bytes (blk_size_bytes_dma - dst_burst_size_bytes = 44 - 32) left to transfer. This is less then the amount of data that is transferred in a destination burst (dst_burst_size_bytes = 32). Therefore, the destination peripheral enters the Single Transaction Region where the GPDMA can complete a block transfer to the destination using single transactions.

Notes

1. *In the Single Transaction Region, asserting dma_single initiates a single transaction for hardware handshaking. Writing a 1 to the relevant channel bit of the **SGLREQDSTREG** register initiates a single transaction for software handshaking.*
2. *The destination peripheral, not knowing the length of a block and only able to request burst transactions, sits and waits for the FIFO to fall below a watermark level before requesting a new burst transaction request.*

At time t2 in **Figure 5-15**, a single transaction to the destination has been completed. There are now only four bytes (12 - dst_single_size_bytes = 12 - 8) left to transfer in the destination block. However, **CTL**.DST_TR_WIDTH implies 64-bit AHB transfers to the destination (dst_single_size_bytes = 8 byte); therefore, the GPDMA cannot form a single word of the specified **CTL**.DST_TR_WIDTH.

The GPDMA channel FIFO has four bytes in it that must be flushed to the destination. The GPDMA switches into a "FIFO flush mode", where the block transfer to the destination is completed by changing the AHB transfer width to the destination to be equal to that of the **CTL**.SRC_TR_WIDTH; that is, byte AHB transfers in this example. Thus the last single transaction in the destination block is made up of a burst of length 4 and **CTL**.SRC_TR_WIDTH width.

When the GPDMA is in FIFO flush mode, the address is incremented by the value of **CTL**.SRC_TR_WIDTH and not **CTL**.DST_TR_WIDTH.

In cases where the **DAR** is selected to be contiguous between blocks, the DARx will need re-alignment at the start of the next block, since it is aligned to **CTL**.SRC_TR_WIDTH and not **CTL**.DST_TR_WIDTH at the end of the previous block (this is handled by hardware).

In general, channel FIFO flushing to the destination occurs if all three of the following are true:

- GPDMA or the Source peripheral are flow control peripherals
- **CTL**.DST_TR_WIDTH > **CTL**.SRC_TR_WIDTH
- Flow control device:
  - If GPDMA is flow controller:
    `blk_size_bytes_dma/dst_single_size_bytes != integer`
  - If source is flow controller:
    `blk_size_bytes_src /dst_single_size_bytes != integer`

Note: When not in FIFO flush mode, a single transaction is mapped to a single AHB transfer. However, in FIFO flush mode, a single transaction is mapped to multiple AHB transfers of CTLx.SRC_TR_WIDTH width. The cumulative total of data transferred to the destination in FIFO flush mode is less than dst_single_size_bytes.

In the above example, a burst request is not generated in the Single Transaction Region. If a burst request were generated at time t1 in **Figure 5-15**, then the burst transaction would proceed until there was not enough data left in the destination block to form a single data item of **CTL**.DST_TR_WIDTH width. The burst transaction would then be early-terminated. In this example, only one data item of the four requested (decoded value of DEST_MIZE = 4) would be transferred to the destination in the burst transaction. This is referred to as an Early-Terminated Burst Transaction. If a burst request were generated at time t2 in **Figure 5-15**, then the destination block would be completed (four byte transfers to the destination to flush the GPDMA channel FIFO) and this burst request would again be early-terminated at the end of the destination block.

Observation: If the source transfer width - **CTL**.SRC_TR_WIDTH in the channel control register (**CTL**) - is less than the destination transfer width (**CTL**.DST_TR_WIDTH), then the FIFO may need to be flushed at the end of the block transfer. This is done by setting the AHB transfer width of the last few AHB transfers of the block to the destination so that it is equal to **CTL**.SRC_TR_WIDTH and not the programmed **CTL**.DST_TR_WIDTH.

Example 5

Scenario: In all examples presented so far, none of the bursts have been early-terminated by the system arbiter. Referring to Example 1, the AHB transfers on the source and destination side look somewhat symmetric. In the examples presented so far, where the bursts are not early-terminated by the system arbiter, the traffic profile on the AHB bus would be the same, regardless of the value of CFGx.FIFO_MODE.

This example, however, considers the effect of CFGx.FIFO_MODE; it is the same for both hardware and software handshaking.

CFGx.FIFO_MODE: Determines how much space or data needs to be available in the FIFO before a burst transaction request is serviced.

0_B - Space/data available for single AHB transfer of the specified transfer width.

1_B - Data available is greater than or equal to half the FIFO depth for destination transfers and space available is greater than half the fifo depth for source transfers. The exceptions are at the end of a burst transaction request or at the end of a block transfer.

Table 5-7 lists the parameters used in this example (as an example the FIFO depth was considered as 16 bytes).

Table 5-7 Parameters in Transfer Operation - Example 6

Parameter	Comment
CTLx.TT_FC = 011_B	Peripheral-to-peripheral transfer with GPDMA as flow controller
CTLx.BLOCK_TS = 32	-
CTLx.SRC_TR_WI DTH = 010_B	32 bits
CTLx. DST_TR_WI DTH = 010_B	32 bits
CTLx.SRC_MSIZE = 010_B	Decoded value = 8
CTLx.DEST_MSIZE = 001_B	Decoded value = 4
CFGx.MAX_ABRST = 0_B	No limit on maximum AMBA burst length

The block transfer may proceed by alternately filling and emptying the GPDMA channel FIFO. Up to time t4, the transfer might proceed like that shown in **Figure 5-16**.

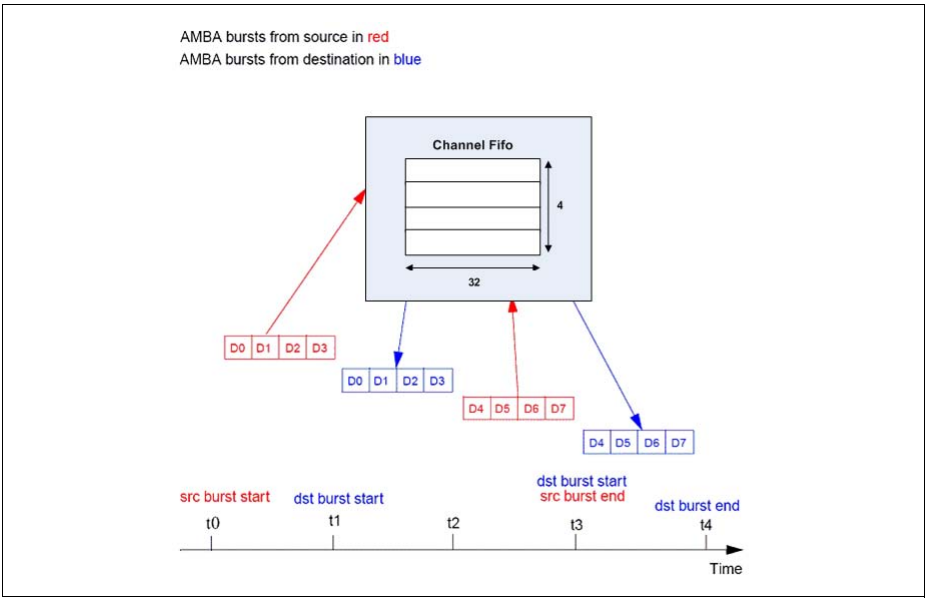


Figure 5-16 Block Transfer Up to Time "t4"

At time t4, the src, channel, and destination FIFOs might look like that shown in **Figure 5-17**

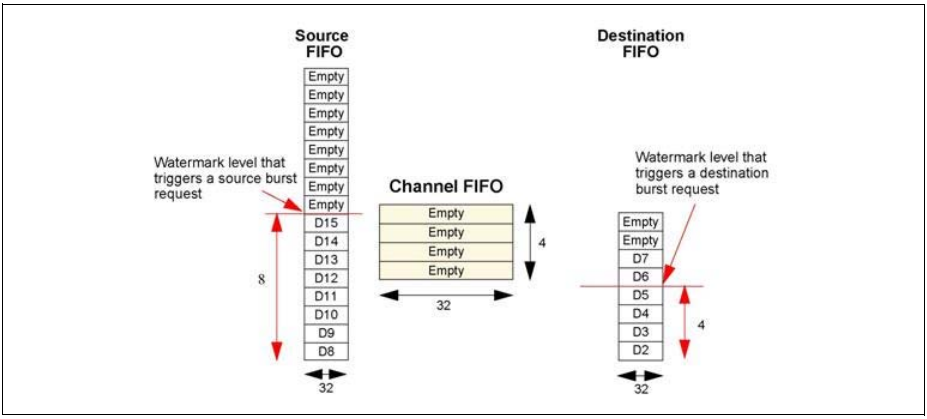


Figure 5-17 Source, GPDMA Channel and Destination FIFOs at Time 't4'

At time t4, a source burst transaction is requested, and the GPDMA attempts a burst of length 4. Suppose that this burst is early-burst terminated after three AHB transfers. The FIFO status after this burst might look like that shown in **Figure 5-18**.

Referring to **Figure 5-18**, notice that a burst request from the destination is not triggered, since the destination FIFO contents are above the watermark level. The GPDMA has space for one data item in the channel FIFO.

The GPDMA will attempt to perform a single transfer, to fill the channel FIFO, if CFGx.FIFO_MODE = 0.

If CFGx.FIFO_MODE = 1, then the GPDMA waits until the channel FIFO is less than half-full before initiating a burst from the source, as illustrated in **Figure 5-18**.

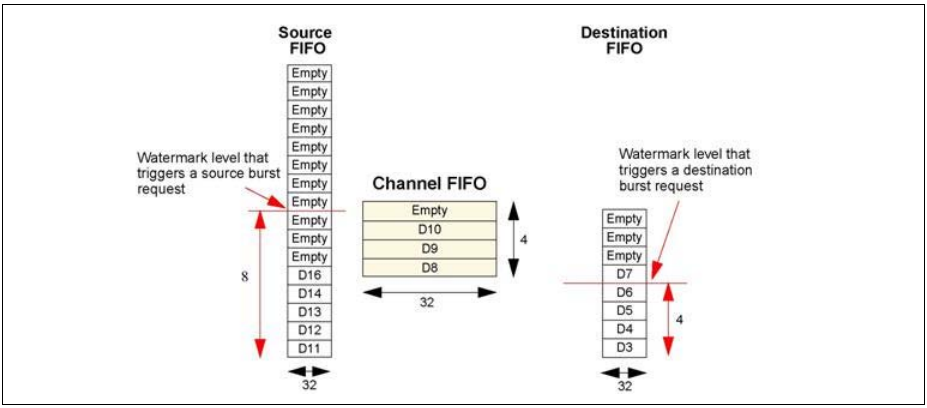


Figure 5-18 FIFO Status After Early-Terminated Burst

Observation: When CFGx.FIFO_MODE = 1, the number of bursts per block is less than when CFGx.FIFO_MODE = 0, hence, the bus utilization will improve. This setting favors longer bursts. However, the latency of DMA transfers may increase when CFGx.FIFO_MODE = 1, since the GPDMA waits for the channel FIFO contents to be less than half the FIFO depth for source transfers, or greater than or equal to half the FIFO depth for destination transfers. Therefore, system bus occupancy and usage can be improved by delaying the servicing of multiple requests until there is sufficient data/space available in the FIFO to generate a burst (rather than multiple single AHB transfers); this comes at the expense of transfer latency. For reduced block transfer latency, set CFGx.FIFO_MODE = 0. For improved bus utilization, set CFGx.FIFO_MODE = 1.

5.2.10 Flow Control Configurations

Figure 5-19 indicates three different flow control configurations using hardware handshaking interfaces - a simplified version of the interface is shown. These scenarios can also be used for software handshaking, which uses software registers instead of signals.

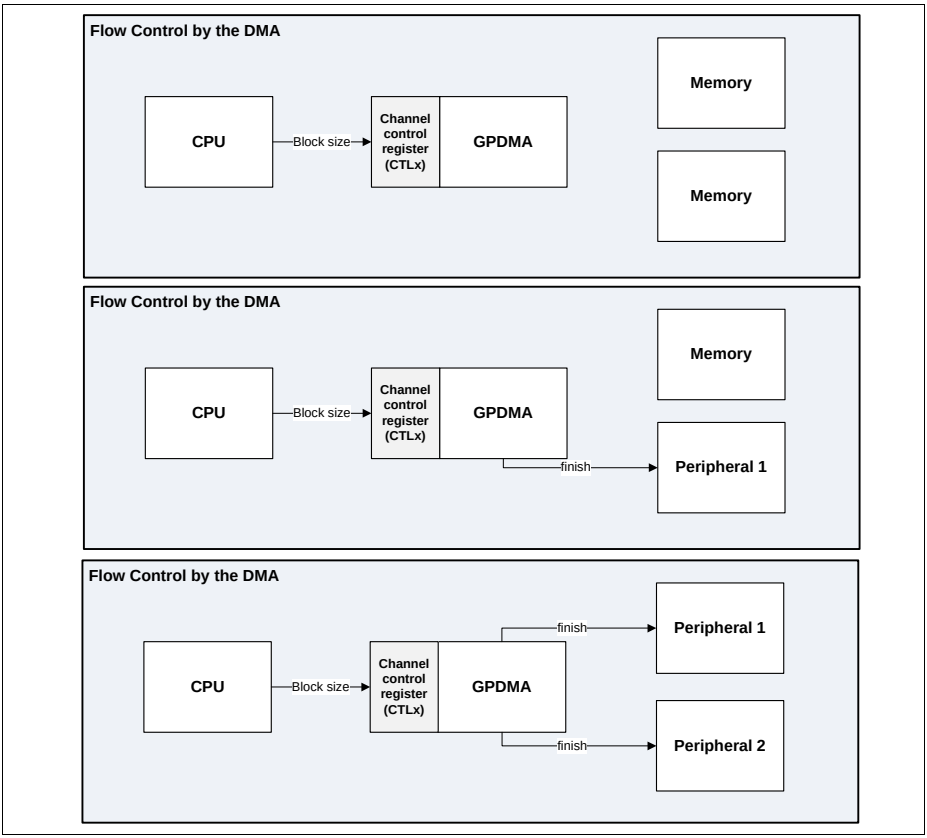


Figure 5-19 Flow Control Configurations

5.2.11 Generating Requests for the AHB Master Bus Interface

Each channel has a source state machine and destination state machine running in parallel. These state machines generate the request inputs to the arbiter, which arbitrates for the master bus interface (one arbiter per master bus interface).

When the source/destination state machine is granted control of the master bus interface, and when the master bus interface is granted control of the external AHB bus, then AHB transfers between the peripheral and the GPDMA (on behalf of the granted state machine) can take place.

AHB transfers from the source peripheral or to the destination peripheral cannot proceed until the channel FIFO is ready. For burst transaction requests and for transfers involving

memory peripherals, the criterion for "FIFO readiness" is controlled by the **FIFO_MODE** field of the **CFG** register.

The definition of FIFO readiness is the same for:

- Single transactions
- Burst transactions, where CFGx.FIFO_MODE = 0
- Transfers involving memory peripherals, where CFGx.FIFO_MODE = 0

The channel FIFO is deemed ready when the space/data available is sufficient to complete a single AHB transfer of the specified transfer width. FIFO readiness for source transfers occurs when the channel FIFO contains enough room to accept at least a single transfer of **CTL**.SRC_TR_WIDTH width. FIFO readiness for destination transfers occurs when the channel FIFO contains data to form at least a single transfer of **CTL**.DST_TR_WIDTH width.

*Note: An exception to FIFO readiness for destination transfers occurs in "FIFO flush mode" In this mode, FIFO readiness for destination transfers occurs when the channel FIFO contains data to form at least a single transfer of **CTL**.SRC_TR_WIDTH width (and not **CTL**.DST_TR_WIDTH width, as is the normal case).*

When **CFG**.FIFO_MODE = 1, then the criteria for FIFO readiness for burst transaction requests and transfers involving memory peripherals are as follows:

- A FIFO is ready for a source burst transfer when the FIFO is less than half empty.
- A FIFO is ready for a destination burst transfer when the FIFO is greater than or equal to half full.

Exceptions to this "readiness" occur. During these exceptions, a value of **CTL**. FIFO_MODE = 0 is assumed. The following are the exceptions:

- Near the end of a burst transaction or block transfer - The channel source state machine does not wait for the channel FIFO to be less than half empty if the number of source data items left to complete the source burst transaction or source block transfer is less than FIFO DEPTH/2. Similarly, the channel destination state machine does not wait for the channel FIFO to be greater than or equal to half full, if the number of destination data items left to complete the destination burst transaction or destination block transfer is less than FIFO DEPTH/2.
- In FIFO flush mode
- When a channel is suspended - The destination state machine does not wait for the FIFO to become half empty to flush the FIFO, regardless of the value of the FIFO_MODE field.

When the source/destination peripheral is not memory, the source/destination state machine waits for a single/burst transaction request. Upon receipt of a transaction request and only if the channel FIFO is "ready" for source/destination AHB transfers, a request for the master bus interface is made by the source/destination state machine.

Note: There is one exception to this, which occurs when the destination peripheral is the flow controller and CFGx.FCMODE = 1 (data pre-fetching is disabled). Then the source state machine does not generate a request for the master bus interface (even if the FIFO is "ready" for source transfers and has received a source transaction request) until the destination requests new data.

When the source/destination peripheral is memory, the source/destination state machine must wait until the channel FIFO is "ready". A request is then made for the master bus interface. There is no handshaking mechanism employed between a memory peripheral and the GPDMA.

5.2.11.1 Locked DMA Transfers

It is possible to program the GPDMA for:

- Bus locking
- Channel locking - Locks the arbitration for the AHB master interface, which grants ownership of the master bus interface to one of the requesting channel state machines (source or destination).

Bus and channel locking can proceed for the duration of a DMA transfer, a block transfer, or a single or burst transaction.

Bus Locking

If the LOCK_B bit in the channel configuration register (**CFG**) is set, then the AHB bus is locked for the duration specified in the LOCK_B_L field.

Channel Locking

If the LOCK_CH field is set, then the arbitration for the master bus interface is exclusively reserved for the source and destination peripherals of that channel for the duration specified in the LOCK_CH_L field.

If bus locking is activated for a certain duration, then it follows that the channel is also automatically locked for that duration. Three cases arise:

- CFGx.LOCK_B = 0 - Programmed values of CFGx.LOCK_CH and CFGx.LOCK_CH_L are used.
- CFGx.LOCK_B = 1 and CFGx.LOCK_CH = 0 - DMA transfer proceeds as if CFGx.LOCK_CH = 1 and CFGx.LOCK_CH_L = CFGx.LOCK_B_L. The programmed values of CFGx.LOCK_CH and CFGx.LOCK_CH_L are ignored.
- CFGx.LOCK_B = 1 and CFGx.LOCK_CH = 1 - Two cases arise:
  – CFGx.LOCK_B_L <= CFGx.LOCK_CH_L - In this case, the DMA transfer proceeds as if CFGx.LOCK_CH_L = CFGx. LOCK_B_L and the programmed value of CFGx.LOCK_CH_L is ignored. Thus, if bus locking is enabled over the DMA transfer level, then channel locking is enabled over the DMA transfer level, regardless of the programmed value of CFGx.LOCK_CH_LCFGx.

– LOCK_B_L > CFGx.LOCK_CH_L - The programmed value of CFGx.LOCK_CH_L is used. Thus, if bus locking is enabled over the DMA block transfer level and channel locking is enabled over the DMA transfer level, then channel locking is performed over the DMA transfer level.

Locking Levels

If locking is enabled for a channel, then locking of the AHB master bus interface at a programmed locking transfer level is activated when the channel is first granted the AHB master bus interface at the start of that locking transfer level. It continues until the locking transfer level has completed; that is, if channel 0 has enabled channel level locking at the block transfer level, then this channel locks the master bus interface when it is first granted the master bus interface at the start of the block transfer, and continues to lock the master bus interface until the block transfer has completed.

Source and destination block transfers occur successively in time, and a new source block cannot commence until the previous destination block has completed.

When both source and destination are on the same AHB layer, then block level locking is terminated on completion of the block to the destination. If they are on separate layers, then block-level locking is terminated on completion of the block on that layer—when the source block on the source AHB layer completes, and when the destination block on the destination AHB layer completes. The same is true for DMA transfer-level locking.

Transaction-level locking is different due to the fact that source and destination transactions occur independently in time, and the number of source and destination transactions in a DMA block or DMA transfer do not have to match. When the source and destination are on the same AHB layer, then transaction-level locking is cleared at the end of a source or destination transaction only if the opposing peripheral is not currently in the middle of a transaction.

For example, if locking is enabled at the transaction level and an end-of-source transaction is signaled, then this disables locking only if one of the following is true:

- The destination is on a different AHB layer
- The destination is on the same AHB layer, but the channel is not currently in the middle of a transaction to the destination peripheral.

The same rules apply when an end-of-destination transaction is signalled.

If channel-level or bus-level locking is enabled for a channel at the transaction level, and either the source or destination of the channel is a memory device, then the locking is ignored and the channel proceeds as if locking (bus or channel) is disabled.

Note: Since there is no notion of a transaction level for a memory peripheral, then transaction-level locking is not allowed when either source or destination is memory.

5.2.12 Arbitration for AHB Master Interface

Each GPDMA channel has two request lines that request ownership of a particular master bus interface: channel source and channel destination request lines.

Source and destination arbitrate separately for the bus. Once a source/destination state machine gains ownership of the master bus interface and the master bus interface has ownership of the AHB bus, then AHB transfers can proceed between the peripheral and GPDMA. **Figure 5-20** illustrates the arbitration flow of the master bus interface.

An arbitration scheme decides which of the request lines is granted the particular master bus interface. Each channel has a programmable priority. A request for the master bus interface can be made at any time, but is granted only after the current AHB transfer (burst or single) has completed. Therefore, if the master interface is transferring data for a lower priority channel and a higher priority channel requests service, then the master interface will complete the current burst for the lower priority channel before switching to transfer data for the higher priority channel.

To prevent a channel from saturating the master bus interface, it can be given a maximum AMBA burst length (MAX_ABRST field in **CFG** register) at channel setup time. This also prevents the master bus interface from saturating the AHB bus where the system arbiter cannot change the grant lines until the end of an undefined length burst.

The following is the interface arbitration scheme employed when no channel has locked the arbitration for the master bus interface:

- If only one request line is active at the highest priority level, then the request with the highest priority wins ownership of the AHB master bus interface; it is not necessary for the priority levels to be unique.
  If more than one request is active at the highest requesting priority, then these competing requests proceed to a second tier of arbitration.
- If equal priority requests occur, then the lower-numbered channel is granted.
  In other words, if a peripheral request attached to Channel 7 and a peripheral request attached to Channel 8 have the same priority, then the peripheral attached to Channel 7 is granted first.

Note: A channel source is granted before the destination if both have their request lines asserted when a grant decision is made. A channel source and channel destination inherit their channel priority and therefore always have the same priority.

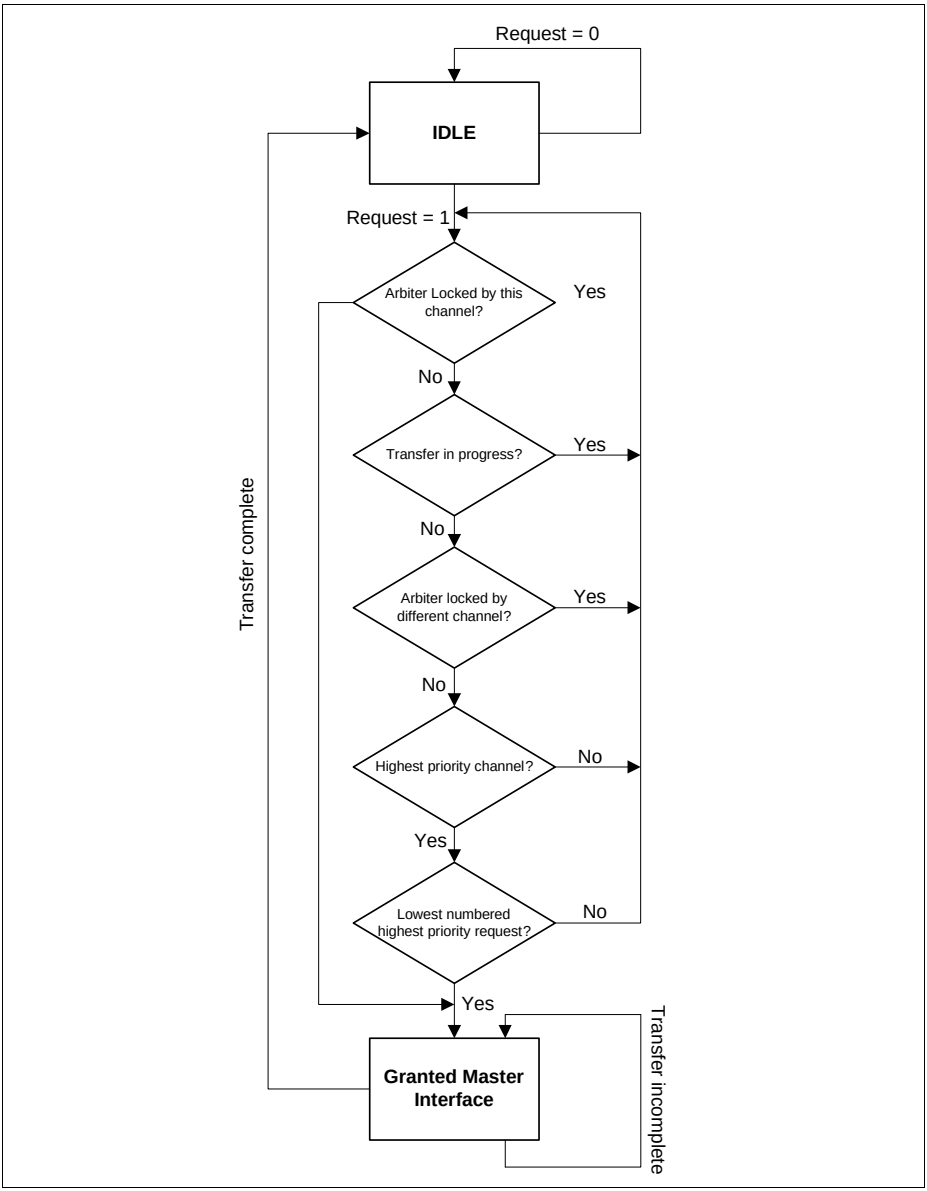


Figure 5-20 Arbitration Flow for Master Bus Interface

5.2.13 Scatter/Gather

Scatter is relevant to a destination transfer. The destination address is incremented or decremented by a programmed amount - the scatter increment - when a scatter boundary is reached. **Figure 5-21** shows an example destination scatter transfer. The destination address is incremented or decremented by the value stored in the destination scatter increment (DSRx.DSI) field (refer to **DSR**), multiplied by the number of bytes in a single AHB transfer to the destination s (decoded value of **CTL**.DST_TR_WIDTH)/8 - when a scatter boundary is reached. The number of destination transfers between successive scatter boundaries is programmed into the Destination Scatter Count (DSC) field of the DSRx register.

Scatter is enabled by writing a 1 to the **CTL**.DST_SCATTER_EN field. The **CTL**.DINC field determines if the address is incremented, decremented, or remains fixed when a scatter boundary is reached. If the **CTL**.DINC field indicates a fixed-address control throughout a DMA transfer, then the **CTL**.DST_SCATTER_EN field is ignored, and the scatter feature is automatically disabled.

Gather is relevant to a source transfer. The source address is incremented or decremented by a programmed amount when a gather boundary is reached. The number of source transfers between successive gather boundaries is programmed into the Source Gather Count (SGRx.SGC) field. The source address is incremented or decremented by the value stored in the source gather increment (SGRx.SGI) field (refer to **SGR**), multiplied by the number of bytes in a single AHB transfer from the source - (decoded value of **CTL**.SRC_TR_WIDTH)/8 - when a gather boundary is reached.

Gather is enabled by writing a 1 to the **CTL**.SRC_GATHER_EN field. The **CTL**.SINC field determines if the address is incremented, decremented, or remains fixed when a gather boundary is reached. If the **CTL**.SINC field indicates a fixed-address control throughout a DMA transfer, then the **CTL**.SRC_GATHER_EN field is ignored, and the gather feature is automatically disabled.

Note: For multi-block transfers, the counters that keep track of the number of transfers left to reach a gather/scatter boundary are re-initialized to the source gather count (SGRx.SGC) and destination scatter count (DSC), respectively, at the start of each block transfer.

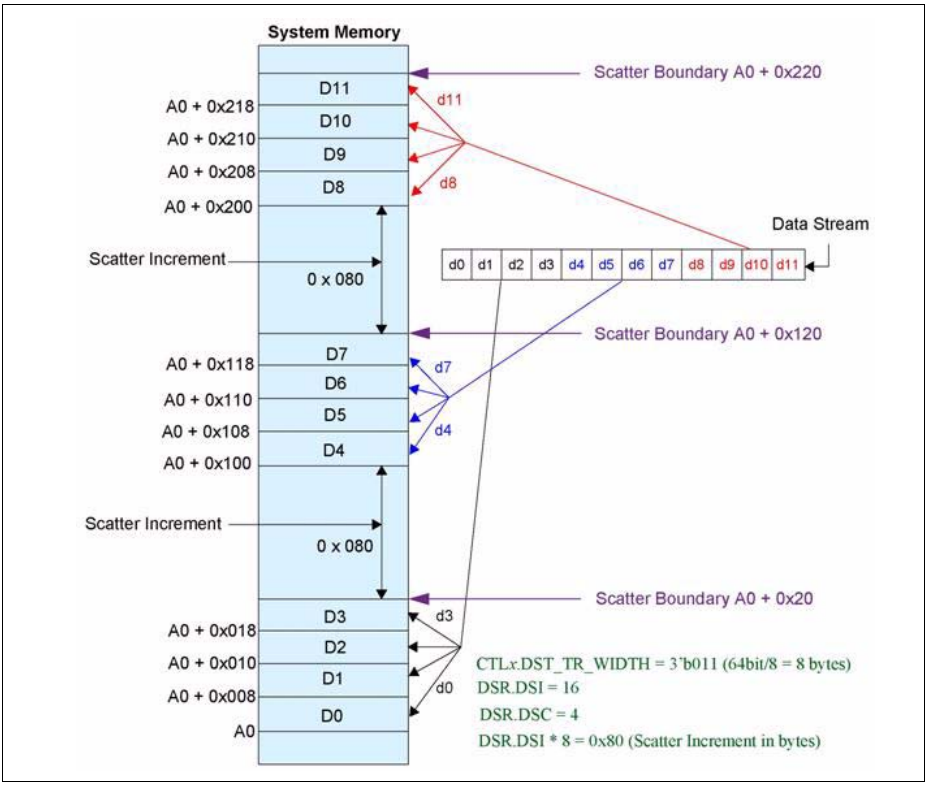


Figure 5-21 Example of Destination Scatter Transfer

As an example of gather increment, consider the following:

```
SRC_TR_WIDTH = 3'b010 (32 bits)
SGR.SGC = 0x04 (source gather count)
CTLx.SRC_GATHER_EN = 1 (source gather enabled)
SARx = A0 (starting source address)
```

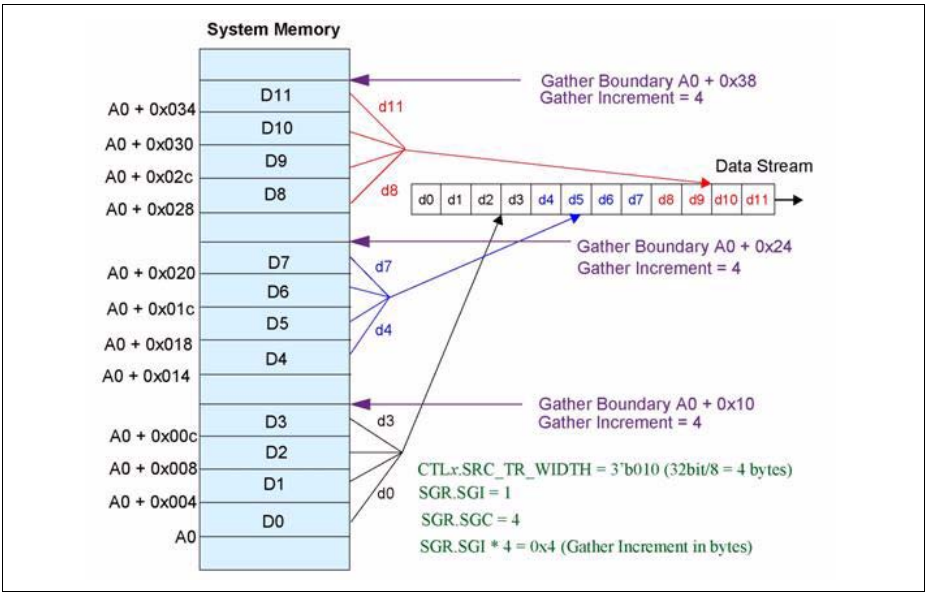


Figure 5-22 Source Gather when SGR.SGI = 0x1

In general, if the starting address is A0 and **CTL**.SINC = 00_B (increment source address control), then the transfer will be:

```
A0, AO + TWB, A0 + 2*TWB                (A0 + (SGR.SGC-1)*TWB)
<-scatter_increment-> (A0 + (SGR.SGC*TWB) + (SGR.SGI *TWB))
```

where TWB is the transfer width in bytes, decoded value of **CTL**.SRC_TR_WIDTH/8 = src_single_size_bytes.

5.3 Programming

The GPDMA can be programmed through software registers or the GPDMA low-level software driver; software registers are described in more detail in **Section 5.6**.

*Note: There are references to both software parameters throughout this chapter. The software parameters are the field names in each register description table and are prefixed by the register name; for example, the Block Transfer Size field in the Control Register for Channel x is designated as "**CTL**.BLOCK_TS."*

5.3.1 Illegal Register Access

An illegal access can be any of the following:

1. A write to the **SAR**, **DAR**, **LLP**, **CTL**, **SSTAT**, **DSTAT**, **SSTATAR**, **DSTATAR**, **SGR**, or **DSR** registers occurs when the channel is enabled.
2. A read from the Interrupt Clear Registers is attempted.
3. A write to the Interrupt Status Registers, **GPDMA0_STATUSINT**, **ID** or **VERSION** is attempted.

An illegal access (read/write) returns an AHB error response.

5.3.2 GPDMA Transfer Types

A DMA transfer may consist of

- single block transfer, supported by all channels.
- multi-block transfers, supported by channels 0 and 1.

On successive blocks of a multi-block transfer, the **SAR**/**DAR** register in the GPDMA is reprogrammed using either of the following methods:

- Block chaining using linked lists
- Auto-reloading
- Contiguous address between blocks

On successive blocks of a multi-block transfer, the **CTL** register in the GPDMA is reprogrammed using either of the following methods:

- Block chaining using linked lists
- Auto-reloading

When block chaining, using Linked Lists is the multi-block method of choice. On successive blocks, the **LLP** register in the GPDMA is reprogrammed using block chaining with linked lists.

A block descriptor consists of six registers: **SAR**, **DAR**, **LLP**, **CTL**, **SSTAT** and **DSTAT**. The first four registers, along with the **CFG** register, are used by the GPDMA to set up and describe the block transfer.

Note: The term Link List Item (LLI) and block descriptor are synonymous.

5.3.2.1 Multi-Block Transfers

Multi-block transfers—in which the source and destination are swapped during the transfer—are not supported. In a multi-block transfer, the direction must not change for the duration of the transfer.

Block Chaining Using Linked Lists

In this case, the GPDMA reprograms the channel registers prior to the start of each block by fetching the block descriptor for that block from system memory. This is known as an LLI update.

GPDMA block chaining uses a Linked List Pointer register (**LLP**) that stores the address in memory of the next linked list item. Each LLI contains the corresponding block descriptors:

1. **SAR**
2. **DAR**
3. **LLP**
4. **CTL**
5. **SSTAT**
6. **DSTAT**

To set up block chaining, you program a sequence of Linked Lists in memory.

LLI accesses are always 32-bit accesses aligned to 32-bit boundaries and cannot be changed or programmed to anything other than 32-bit, even if the AHB master interface of the LLI supports more than a 32-bit data width.

The **SAR**, **DAR**, **LLP**, and **CTL** registers are fetched from system memory on an LLI update. The updated contents of the **CTL**, **SSTAT**, and **DSTAT** registers are optionally written back to memory on block completion. **Figure 5-23** and **Figure 5-24** show how you use chained linked lists in memory to define multi-block transfers using block chaining.

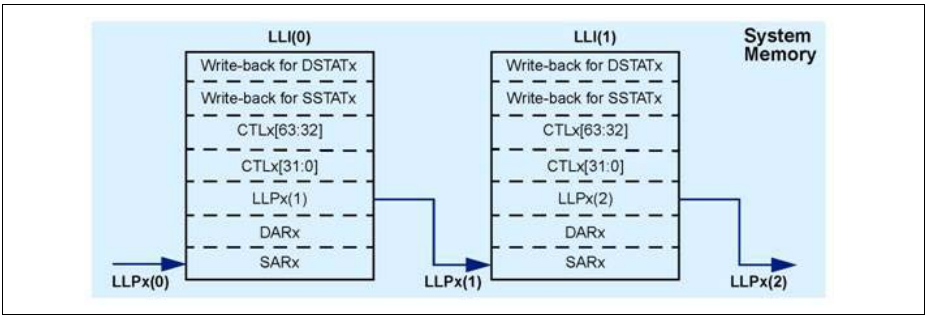


Figure 5-23 Multi-Block Transfer Using Linked Lists When CFGx.SS_UPD_EN is set to '1'

It is assumed that no allocation is made in system memory for the source status when the parameter *CFGx.SS_UPD_EN* is set to ‘0’. In this case, then the order of a Linked List item is as follows:

1. **SAR**
2. **DAR**
3. **LLP**
4. **CTL**
5. **DSTAT**

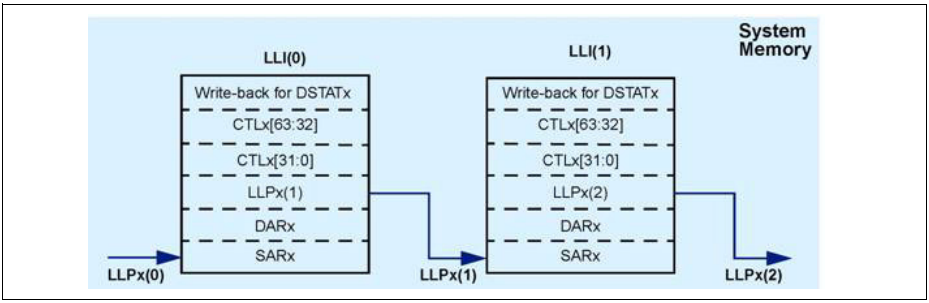


Figure 5-24 Multi-Block Transfer Using Linked Lists When CFGx.SS_UPD_EN is set to ‘0’

*Note: In order to not confuse the **SAR**, **DAR**, **LLP**, **CTL**, **SSTAT** and **DSTAT** register locations of the LLI with the corresponding GPDMA memory mapped register locations, the LLI register locations are prefixed with LLI; that is, LLI.SARx, LLI.DARx, LLI.LLPx, LLI.CTLx, LLI.SSTATx, and LLI.DSTATx.*

Figure 5-23 and **Figure 5-24** show the mapping of a Linked List Item stored in memory to the channel registers block descriptor.

Rows 6 through 10 of **Table 5-8** show the required values of LLPx, CTLx, and CFGx for multi-block DMA transfers using block chaining.

*Note: For rows 6 through 10 of **Table 5-8**, the LLI.CTLx, LLI.LLPx, LLI.SARx, and LLI.DARx register locations of the LLI are always affected at the start of every block transfer. The LLI.LLPx and LLI.CTLx locations are always used to reprogram the GPDMA LLPx and CTLx registers. However, depending on the **Table 5-8** row number, the LLI.SARx/LLI.DARx address may or may not be used to reprogram the GPDMA SARx/DARx registers.*

Table 5-8 Programming of Transfer Types and Channel Register Update Method

Transfer Type	LLP. LOC = 0	CTLx.LLP_ SRC EN	CFGx.RELOAD_SRC	CTLx.LLPDST_EN	CFGx.RELOAD_DST	CTLx, LLPx UpdateMethod	SARx UpdateMethod	DARx UpdateMethod	Write Back[1)]
1. Single-block or last transfer of multi-block.	Yes	0	0	0	0	None, user reprograms	None (single)	None (single)	No
2. Auto-reload multi-block transfer with contiguous SAR	Yes	0	0	0	1	CTLx, LLPx are reloaded from initial values.	Contig uous	Auto-Reload	No
3. Auto-reload multi-block transfer with contiguous DAR.	Yes	0	1	0	0	CTLx, LLPx are reloaded from initial values	Auto-Reload	Contig uous	No
4. Auto-reload multi-block transfer	Yes	0	1	0	1	CTLx, LLPx are reloaded from initial values	Auto-reload	Auto-Reload	No
5. Single-block or last transfer of multi-block.	No	0	0	0	0	None, user reprograms	None (single)	None (single)	Yes
6. Linked list multi-block transfer with contiguous SAR	No	0	0	1	0	CTLx, LLPx loaded from next Linked List item.	Contig uous	Linked List	Yes
7. Linked list multi-block transfer with auto-reload SAR	No	0	1	1	0	CTLx, LLPx loaded from next Linked List item.	Auto-Reload	Linked List	Yes
8. Linked list multi-block transfer with contiguous DAR	No	1	0	0	0	CTLx, LLPx loaded from next Linked List item.	Linked List	Contig uous	Yes

Table 5-8 Programming of Transfer Types and Channel Register Update Method (cont'd)

Transfer Type	LLP. LOC = 0	CTLx.LLP_ SRC EN	CFGx.RELOAD_SRC	CTLx.LLPDST_EN	CFGx.RELOAD_DST	CTLx, LLPx UpdateMethod	SARx UpdateMethod	DARx UpdateMethod	Write Back[1)]
9. Linked list multi-block transfer with auto-reload DAR	No	1	0	0	1	CTLx, LLPx loaded from next Linked List item.	Linked List	Auto-Reload	Yes
10. Linked list multi-block transfer	No	1	0	1	0	CTLx, LLPx loaded from next Linked List item.	Linked List	Linked List	Yes

1) Applicable to channels 0 and 1 only.

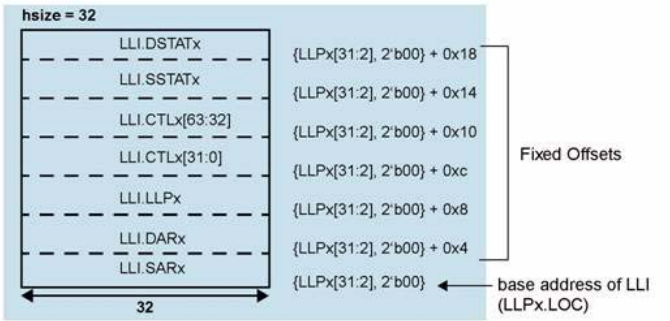


Figure 5-25 Mapping of Block Descriptor (LLI) in Memory to Channel Registers When CFG.SS_UPD_EN = 1

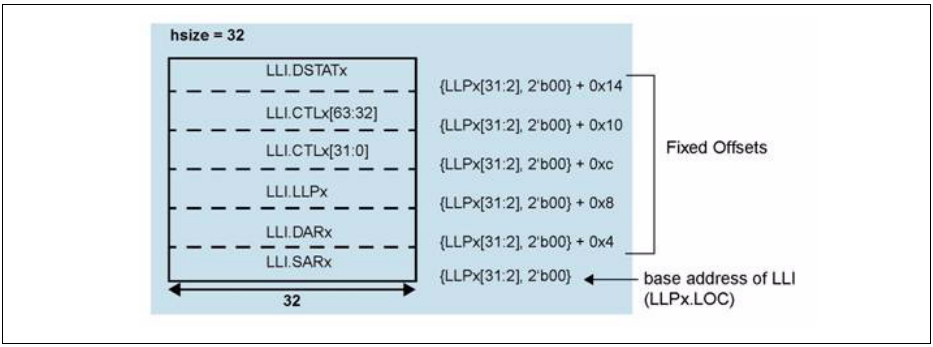


Figure 5-26 Mapping of Block Descriptor (LLI) in Memory to Channel Registers When CFG.SS_UPD_EN = 0

Notes

1. *Throughout this chapter, there are descriptions about fetching the LLI.CTLx register from the location pointed to by the LLPx register. This exact location is the LLI base address (stored in LLPx register) plus the fixed offset. For example, in **Figure 5-25** the location of the LLI.CTLx register is LLPx.LOC + 0xc.*
2. *Referring to **Table 5-8**, if the Write Back column entry is "Yes" and the channel is 0 or 1, then the CTLxH register is always written to system memory (to LLI.CTLxH) at the end of every block transfer.*
3. *The source status is fetched and written to system memory at the end of every block transfer if the Write Back column entry is "Yes" and CFGx.SS_UPD_EN is enabled.*
4. *The destination status is fetched and written to system memory at the end of every block transfer if the Write Back column entry is "Yes" and CFGx.DS_UPD_EN is enabled.*

5.3.2.2 Auto-Reloading of Channel Registers

During auto-reloading, the channel registers are reloaded with their initial values at the completion of each block and the new values used for the new block. Depending on the row number in **Table 5-8**, some or all of the SARx, DARx, and CTLx channel registers are reloaded from their initial value at the start of a block transfer.

5.3.2.3 Contiguous Address Between Blocks

In this case, the address between successive blocks is selected as a continuation from the end of the previous block.

Enabling the source or destination address to be contiguous between blocks is a function of the **CTL**.LLP_SRC_EN, **CFG**.RELOAD_SRC, **CTL**.LLP_DST_EN, and **CTL**.RELOAD_DST registers (see **Table 5-8**).

*Note: You cannot select both SARx and DARx updates to be contiguous. If you want this functionality, you should increase the size of the Block Transfer (**CTL**.BLOCK_TS), or if this is at the maximum value, use Row 10 of **Table 5-8** and set up the LLI.SARx address of the block descriptor to be equal to the end SARx address of the previous block. Similarly, set up the LLI.DARx address of the block descriptor to be equal to the end DARx address of the previous block.*

5.3.2.4 Suspension of Transfers Between Blocks

At the end of every block transfer, an end-of-block interrupt is asserted if:

1. Interrupts are enabled, **CTL**.INT_EN = 1, and
2. The channel block interrupt is unmasked, MASKBLOCK[n] = 1, where n is the channel number.

Note: The block-complete interrupt is generated at the completion of the block transfer to the destination.

For rows 6, 8, and 10 of **Table 5-8**, the DMA transfer does not stall between block transfers. For example, at the end-of-block N, the GPDMA automatically proceeds to block N + 1.

For rows 2, 3, 4, 7, and 9 of **Table 5-8** (SARx and/or DARx auto-reloaded between block transfers), the DMA transfer automatically stalls after the end-of-block interrupt is asserted, if the end-of-block interrupt is enabled and unmasked.

The GPDMA does not proceed to the next block transfer until a write to the CLEARBLOCK[n] block interrupt clear register, done by software to clear the channel block-complete interrupt, is detected by hardware.

For rows 2, 3, 4, 7, and 9 of **Table 5-8** (SARx and/or DARx auto-reloaded between block transfers), the DMA transfer does not stall if either:

- Interrupts are disabled, **CTL**.INT_EN = 0, or
- The channel block interrupt is masked, MASKBLOCK[n] = 0, where n is the channel number.

Channel suspension between blocks is used to ensure that the end-of-block ISR (interrupt service routine) of the next-to-last block is serviced before the start of the final block commences. This ensures that the ISR has cleared the **CFG**.RELOAD_SRC and/or **CFG**.RELOAD_DST bits before completion of the final block. The reload bits **CFG**.RELOAD_SRC and/or **CFG**.RELOAD_DST should be cleared in the end-of-block ISR for the next-to-last block transfer.

5.3.2.5 Ending Multi-Block Transfers

All multi-block transfers must end as shown in either Row 1 or Row 5 of **Table 5-8**. At the end of every block transfer, the GPDMA samples the row number, and if the GPDMA is in the Row 1 or Row 5 state, then the previous block transferred was the last block and the DMA transfer is terminated.

Note: Row 1 and Row 5 are used for single-block transfers or terminating multi-block transfers. Ending in the Row 5 state enables status fetch and write-back for the last block. Ending in the Row 1 state disables status fetch and write-back for the last block.

For rows 2, 3, and 4 of **Table 5-8**, (LLPx.LOC = 0 and **CFG**.RELOAD_SRC and/or **CFG**.RELOAD_DST is set), multi-block DMA transfers continue until both the **CFG**.RELOAD_SRC and **CFG**.RELOAD_DST registers are cleared by software. They should be programmed to 0 in the end-of-block interrupt service routine that services the next-to-last block transfer; this puts the GPDMA into the Row 1 state.

For rows 6, 8, and 10 of **Table 5-8** (both **CFG**.RELOAD_SRC and **CFG**.RELOAD_DST cleared), the user must set up the last block descriptor in memory so that both LLI.CTLx.LLP_SRC_EN and LLI.CTLx.LLP_DST_EN are 0. If the LLI.LLPx register of the last block descriptor in memory is non-zero, then the DMA transfer is terminated in Row 5. If the LLI.LLPx register of the last block descriptor in memory is 0, then the DMA transfer is terminated in Row 1.

*Note: The only allowed transitions between the rows of **Table 5-8** are from any row into Row 1 or Row 5. As already stated, a transition into row 1 or row 5 is used to terminate the DMA transfer; all other transitions between rows are not allowed. Software must ensure that illegal transitions between rows do not occur between blocks of a multi-block transfer. For example, if block N is in row 10, then the only allowed rows for block N +1 are rows 10, 5, or 1.*

5.3.3 Programing Examples

Three registers - **LLP**, **CTL**, and **CFG** - need to be programmed to determine whether single- or multi-block transfers occur, and which type of multi-block transfer is used. The different transfer types are shown in **Table 5-8**.

The GPDMA can be programmed to fetch the status from the source or destination peripheral; this status is stored in the **SSTAT** and **DSTAT** registers. When the GPDMA is programmed to fetch the status from the source or destination peripheral, it writes this status and the contents of the **CTL** register back to memory at the end of a block transfer. The Write Back column of **Table 5-8** shows when this occurs.

The "Update Method" columns indicate where the values of **SAR**, **DAR**, **CTL**, and **LLP** are obtained for the next block transfer when multi-block GPDMA transfers are enabled.

*Note: In **Table 5-8**, all other combinations of LLPx.LOC = 0, **CTL**.LLP_SRC_EN, CFGx.RELOAD_SRC, **CTL**.LLP_DST_EN, and CFGx.RELOAD_DST are illegal, and will cause indeterminate or erroneous behavior.*

Generic Setup of Transfer Type and Characteristics

This generic sequence is referenced by the examples further below in this section.

1. Set up the transfer type (memory or non-memory peripheral for source and destination) and flow control device by programming the TT_FC of the **CTL** register. **Table 5-13** lists the decoding for this field.
2. Set up the transfer characteristics, such as:
   a) Transfer width for the source in the SRC_TR_WIDTH field.
   Table 5-12 lists the decoding for this field.
   b) Transfer width for the destination in the DST_TR_WIDTH field.
   Table 5-12 lists the decoding for this field.
   c) Source master layer in the SMS field where the source resides.
   d) Destination master layer in the DMS field where the destination resides.
   e) Incrementing/decrementing or fixed address for the source in the SINC field.
   f) Incrementing/decrementing or fixed address for the destination in the DINC field.

5.3.3.1 Single-block Transfer (Row 1)

This section describes a single-block transfer, Row 1 in **Table 5-8**.

*Note: Row 5 in **Table 5-8** is also a single-block transfer with write-back of control and status information enabled at the end of the single-block transfer.*

1. Read the Channel Enable register to choose a free (disabled) channel.
2. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers: CLEARTFR, CLEARBLOCK, CLEARSRCTRAN, CLEARDSTTRAN, and CLEARERR. Reading the Interrupt Raw Status and Interrupt Status registers confirms that all interrupts have been cleared.
3. Program the following channel registers:
   a) Write the starting source address in the **SAR** register for channel x.
   b) Write the starting destination address in the **DAR** register for channel x.
   c) Program **CTL** and **CFG** according to Row 1, as shown in **Table 5-8**. Program the **LLP** register with 0.
   d) Write the control information for the DMA transfer in the **CTL** register for channel x.
   e) Write the channel configuration information into the **CFG** register for channel x.
      1. Designate the handshaking interface type (hardware or software) for the source and destination peripherals; this is not required for memory.
      This step requires programming the **CFG**.HS_SEL_SRC or **CFG**.HS_SEL_DST bits, respectively. Writing a 0 activates the hardware handshaking interface to handle source/destination requests. Writing a 1 activates the software handshaking interface to handle source and destination requests.

2. If the hardware handshaking interface is activated for the source or destination peripheral, assign a handshaking interface to the source and destination peripheral; this requires programming the **CFG**.SRC_PER and **CFG**.DEST_PER bits, respectively.

f) If gather is enabled (**CTL**.SRC_GATHER_EN = 1), program the **SGR** register for channel x.

g) If scatter is enabled (**CTL**.DST_SCATTER_EN = 1), program the **DSR** register for channel x.

4. After the GPDMA-selected channel has been programmed, enable the channel by writing a 1 to the **GPDMA0_CHENREG**.CH_EN bit. Ensure that bit 0 of the **GPDMA0_DMACFGREG** register is enabled.
5. Source and destination request single and burst DMA transactions in order to transfer the block of data (assuming non-memory peripherals). The GPDMA acknowledges at the completion of every transaction (burst and single) in the block and carries out the block transfer.
6. Once the transfer completes, hardware sets the interrupts and disables the channel. At this time, you can respond to either the Block Complete or Transfer Complete interrupts, or poll for the transfer complete raw interrupt status register (RAWTFR[n], n = channel number) until it is set by hardware, in order to detect when the transfer is complete. Note that if this polling is used, the software must ensure that the transfer complete interrupt is cleared by writing to the Interrupt Clear register, CLEARTFR[n], before the channel is enabled.

5.3.3.2 Multi-Block Transfer with Linked List for Source and Linked List for Destination (Row 10)

This type of transfer is supported by channels 0 and 1 only.

1. Read the Channel Enable register (see **GPDMA0_CHENREG**) to choose a free (disabled) channel.
2. Set up the chain of Linked List Items (otherwise known as block descriptors) in memory. Write the control information in the LLI.**CTL** register location of the block descriptor for each LLI in memory (see **Figure 5-23**) for channel x.
3. Write the channel configuration information into the **CFG** register for channel x.

a) Designate the handshaking interface type (hardware or software) for the source and destination peripherals; this is not required for memory.
This step requires programming the **CFG**.HS_SEL_SRC or **CFG**.HS_SEL_DST bits, respectively. Writing a 0 activates the hardware handshaking interface to handle source/destination requests for the specific channel. Writing a 1 activates the software handshaking interface to handle source/destination requests.

b) If the hardware handshaking interface is activated for the source or destination peripheral, assign the handshaking interface to the source and destination peripheral. This requires programming the **CFG**.SRC_PER and **CFG**.DEST_PER bits, respectively.

4. Make sure that the LLI.CTLx register locations of all LLI entries in memory (except the last) are set as shown in Row 10 of **Table 5-8**. The LLI.CTLx register of the last Linked List Item must be set as described in Row 1 or Row 5 of **Table 5-8**. **Figure 5-23** shows a Linked List example with two list items.
5. Make sure that the LLI.LLPx register locations of all LLI entries in memory (except the last) are non-zero and point to the base address of the next Linked List Item.
6. Make sure that the LLI.SARx/LLI.DARx register locations of all LLI entries in memory point to the start source/destination block address preceding that LLI fetch.
7. Ensure that the LLI.CTLx.DONE field of the LLI.CTLx register locations of all LLI entries in memory is cleared.
8. If source status fetching is enabled (CFGx.SS_UPD_EN is enabled), program the **SSTATAR** register so that the source status information can be fetched from the location pointed to by the **SSTATAR**. For conditions under which the source status information is fetched from system memory, refer to the Write Back column of **Table 5-8**.
9. If destination status fetching is enabled (CFGx.DS_UPD_EN is enabled), program the **DSTATAR** register so that the destination status information can be fetched from the location pointed to by the **DSTATAR** register. For conditions under which the destination status information is fetched from system memory, refer to the Write Back column of **Table 5-8**.
10. If gather is enabled (**CTL**.SRC_GATHER_EN = 1), program the **SGR** register for channel x.
11. If scatter is enabled (**CTL**.DST_SCATTER_EN = 1), program the **DSR** register for channel x.
12. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers: CLEARTFR, CLEARBLOCK, CLEARSRCTRAN, CLEARDSTTRAN, and CLEARERR. Reading the Interrupt Raw Status and Interrupt Status registers confirms that all interrupts have been cleared.
13. Program the **CTL** and **CFG** registers according to Row 10, as shown in **Table 5-8**.
14. Program the **LLP** register with LLP(0), the pointer to the first linked list item.
15. Finally, enable the channel by writing a 1 to the **GPDMA0_CHENREG**.CH_EN bit; the transfer is performed.
16. The GPDMA fetches the first LLI from the location pointed to by LLPx(0).
    Note: *The LLI.SARx, LLI.DARx, LLI.LLPx, and LLI.CTLx registers are fetched. The GPDMA automatically reprograms the* SARx, DARx, LLPx*, and CTLx channel registers from the* LLPx*(0).*
17. Source and destination request single and burst DMA transactions to transfer the block of data (assuming non-memory peripheral). The GPDMA acknowledges at the completion of every transaction (burst and single) in the block and carries out the block transfer.
18. Once the block of data is transferred, the source status information is fetched from the location pointed to by the **SSTATAR** register and stored in the **SSTAT** register if CFGx.SS_UPD_EN is enabled. For conditions under which the source status

information is fetched from system memory, refer to the Write Back column of **Table 5-8**.

The destination status information is fetched from the location pointed to by the **DSTATAR** register and stored in the **DSTAT** register if CFGx.DS_UPD_EN is enabled. For conditions under which the destination status information is fetched from system memory, refer to the Write Back column of **Table 5-8**.

19. The **CTLxH** register is written out to system memory. For conditions under which the **CTLxH** register is written out to system memory, refer to the Write Back column of **Table 5-8**.

    The **CTLxH** register is written out to the same location on the same layer (**LLP**.LMS) where it was originally fetched; that is, the location of the **CTL** register of the linked list item fetched prior to the start of the block transfer. Only the **CTLxH** register is written out, because only the **CTL**.BLOCK_TS and **CTL**.DONE fields have been updated by the GPDMA hardware. Additionally, the **CTL**.DONE bit is asserted to indicate block completion. Therefore, software can poll the LLI.CTLx.DONE bit of the **CTL** register in the LLI to ascertain when a block transfer has completed.

    Note: *Do not poll the CTLx.DONE bit in the GPDMA memory map; instead, poll the LLI.CTLx.DONE bit in the LLI for that block. If the polled LLI.CTLx.DONE bit is asserted, then this block transfer has completed. This LLI.CTLx.DONE bit was cleared at the start of the transfer (Step 7).*

20. The **SSTAT** register is now written out to system memory if CFGx.SS_UPD_EN is enabled. It is written to the **SSTAT** register location of the LLI pointed to by the previously saved LLPx.LOC register.

    The **DSTAT** register is now written out to system memory if CFGx.DS_UPD_EN is enabled. It is written to the **DSTAT** register location of the LLI pointed to by the previously saved LLPx.LOC register.

    The end-of-block interrupt, int_block, is generated after the write-back of the control and status registers has completed.

    Note: The write-back location for the control and status registers is the LLI pointed to by the previous value of the LLPx.LOC register, not the LLI pointed to by the current value of the LLPx.LOC register.

21. The GPDMA does not wait for the block interrupt to be cleared, but continues fetching the next LLI from the memory location pointed to by the current **LLP** register and automatically reprograms the **SAR**, **DAR**, **CTL**, and **LLP** channel registers. The DMA transfer continues until the GPDMA determines that the **CTL** and **LLP** registers at the end of a block transfer match the ones described in Row 1 or Row 5 of **Table 5-8** (as discussed earlier). The GPDMA then knows that the previously transferred block was the last block in the DMA transfer.

The DMA transfer might look like that shown in **Figure 5-27**.

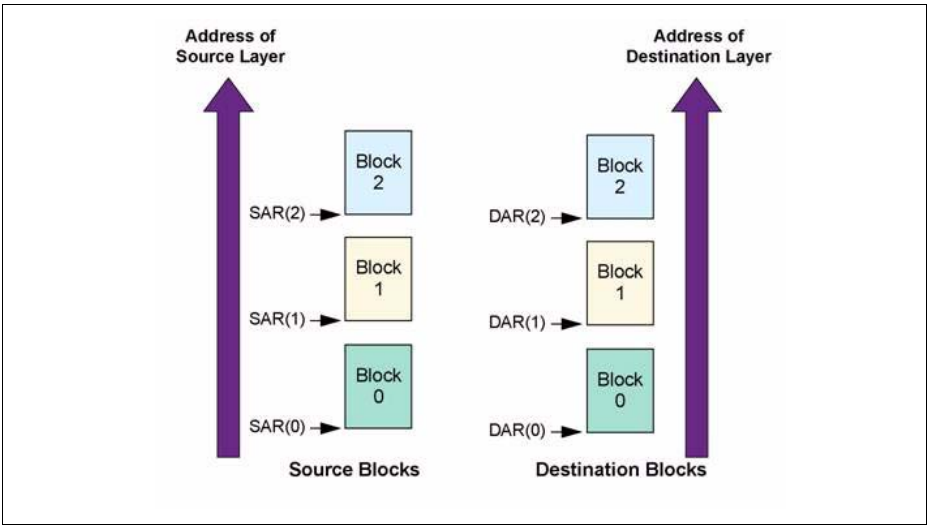


Figure 5-27 Multi-Block with Linked Address for Source and Destination

If the user needs to execute a DMA transfer where the source and destination address are contiguous, but where the amount of data to be transferred is greater than the maximum block size **CTL**.BLOCK_TS, then this can be achieved using the type of multi-block transfer shown in **Figure 5-28**.

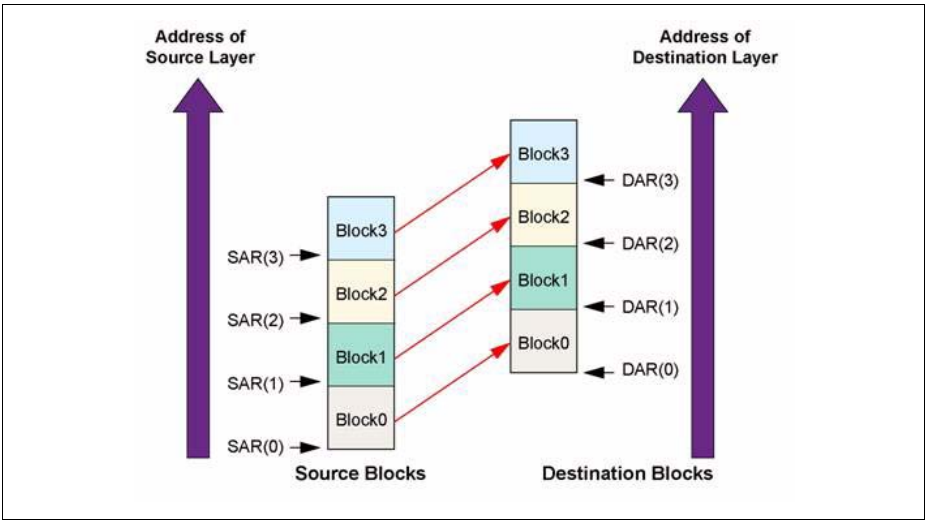


Figure 5-28 Multi-Block with Linked Address for Source and Destination Where SARx and DARx Between Successive Blocks are Contiguous

The DMA transfer flow is shown in **Figure 5-29**.

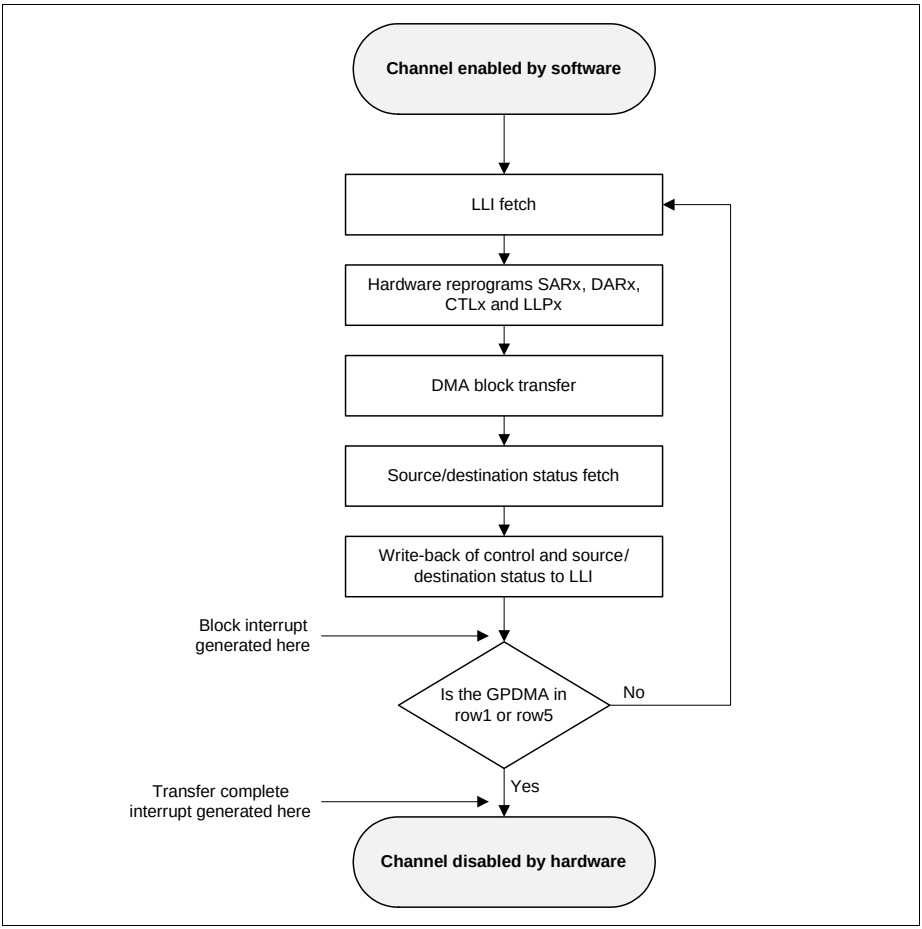


Figure 5-29 DMA Transfer Flow for Source and Destination Linked List Address

5.3.3.3 Multi-Block Transfer with Source Address Auto-Reloaded and Destination Address Auto-Reloaded (Row 4)

This type of transfer is supported by channels 0 and 1 only.

1. Read the Channel Enable register (see **GPDMA0_CHENREG**) to choose an available (disabled) channel.
2. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers: CLEARTFR, CLEARBLOCK, CLEARSRCTRAN, CLEARDSTTRAN, and CLEARERR. Reading the Interrupt Raw Status and Interrupt Status registers confirms that all interrupts have been cleared.
3. Program the following channel registers:
   a) Write the starting source address in the **SAR** register for channel x.
   b) Write the starting destination address in the **DAR** register for channel x.
   c) Program **CTL** and **CFG** according to Row 4, as shown in **Table 5-8**. Program the **LLP** register with 0.
   d) Write the control information for the DMA transfer in the **CTL** register for channel x.
   e) If gather is enabled (**CTL**.SRC_GATHER_EN = 1), program the **SGR** register for channel x.
   f) If scatter is enabled (**CTL**.DST_SCATTER_EN = 1), program the **DSR** register for channel x.
   g) Write the channel configuration information into the **CFG** register for channel x. Ensure that the reload bits, CFGx. RELOAD_SRC and CFGx.RELOAD_DST, are enabled.
      1. Designate the handshaking interface type (hardware or software) for the source and destination peripherals; this is not required for memory.
      This step requires programming the HS_SEL_SRC/HS_SEL_DST bits, respectively. Writing a 0 activates the hardware handshaking interface to handle source/destination requests for the specific channel. Writing a 1 activates the software handshaking interface to handle source/destination requests.
      2. If the hardware handshaking interface is activated for the source or destination peripheral, assign the handshaking interface to the source and destination peripheral. This requires programming the SRC_PER and DEST_PER bits, respectively.
4. After the GPDMA selected channel has been programmed, enable the channel by writing a 1 to the **GPDMA0_CHENREG**.CH_EN bit. Ensure that bit 0 of the **GPDMA0_DMACFGREG** register is enabled.
5. Source and destination request single and burst GPDMA transactions to transfer the block of data (assuming non-memory peripherals). The GPDMA acknowledges on completion of each burst/single transaction and carries out the block transfer.
6. When the block transfer has completed, the GPDMA reloads the **SAR**, **DAR**, and **CTL** registers. Hardware sets the block-complete interrupt. The GPDMA then samples the row number, as shown in **Table 5-8**. If the GPDMA is in Row 1, then the DMA transfer has completed. Hardware sets the transfer complete interrupt and

disables the channel. You can either respond to the Block Complete or Transfer Complete interrupts, or poll for the transfer complete raw interrupt status register (RAWTFR[n], where n is the channel number) until it is set by hardware, in order to detect when the transfer is complete. Note that if this polling is used, software must ensure that the transfer complete interrupt is cleared by writing to the Interrupt Clear register, CLEARTFR[n], before the channel is enabled. If the GPDMA is not in Row 1, the next step is performed.

7. The DMA transfer proceeds as follows:
   a) If interrupts are enabled (**CTL**.INT_EN = 1) and the block-complete interrupt is unmasked (MASKBLOCK[x] = 1_B, where x is the channel number), hardware sets the block-complete interrupt when the block transfer has completed. It then stalls until the block-complete interrupt is cleared by software. If the next block is to be the last block in the DMA transfer, then the block-complete ISR (interrupt service routine) should clear the reload bits in the CFGx.RELOAD_SRC and CFGx.RELOAD_DST registers. This puts the GPDMA into Row 1, as shown in **Table 5-8**. If the next block is not the last block in the DMA transfer, then the reload bits should remain enabled to keep the GPDMA in Row 4.
   b) If interrupts are disabled (**CTL**.INT_EN = 0) or the block-complete interrupt is masked (MASKBLOCK[x] = 0_B, where x is the channel number), then hardware does not stall until it detects a write to the block-complete interrupt clear register; instead, it immediately starts the next block transfer. In this case, software must clear the reload bits in the CFGx.RELOAD_SRC and CFGx.RELOAD_DST registers to put the GPDMA into Row 1 of **Table 5-8** before the last block of the DMA transfer has completed.

The transfer is similar to that shown in **Figure 5-30**.

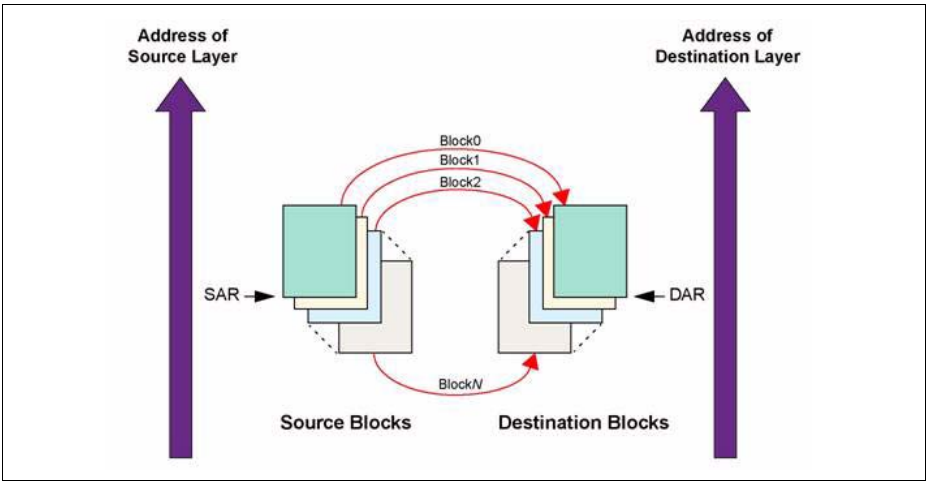


Figure 5-30 Multi-Block DMA Transfer with Source and Destination Address Auto-Reloaded

The DMA transfer flow is shown in **Figure 5-31**.

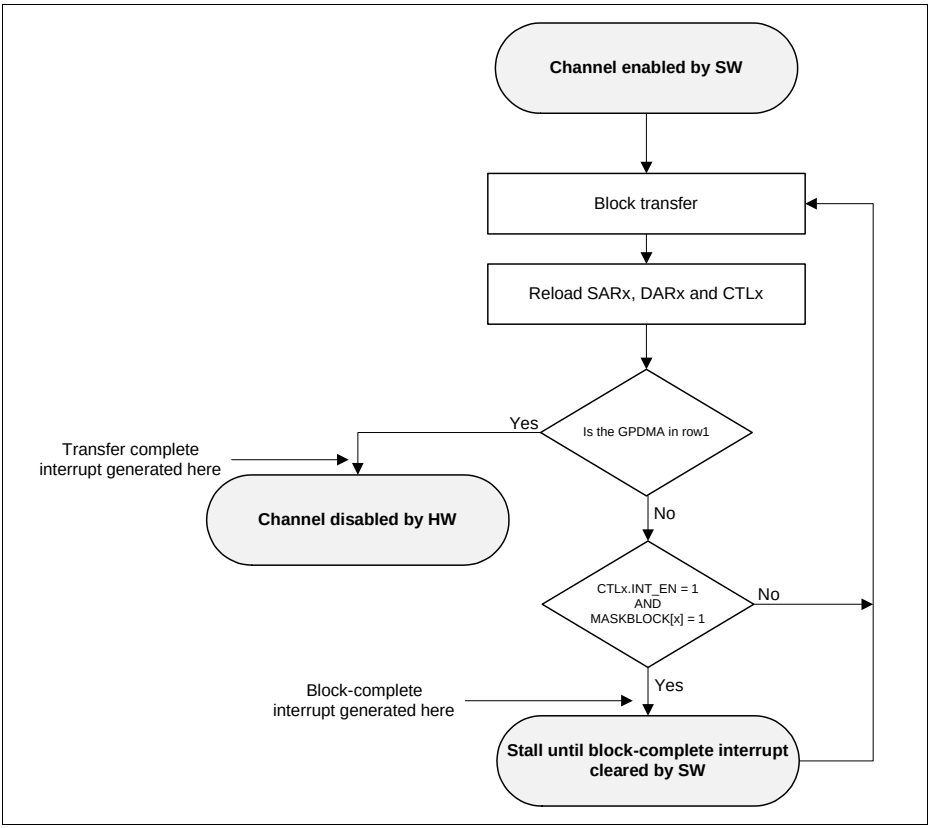


Figure 5-31 DMA Transfer Flow for Source and Destination Address Auto-Reloaded

5.3.3.4 Multi-Block Transfer with Source Address Auto-Reloaded and Linked List Destination Address (Row 7)

This type of transfer is supported by channels 0 and 1 only.

1. Read the Channel Enable register (see **GPDMA0_CHENREG**) in order to choose a free (disabled) channel.
2. Set up the chain of linked list items (otherwise known as block descriptors) in memory. Write the control information in the LLI.**CTL** register location of the block descriptor for each LLI in memory (see **Figure 5-23**) for channel x.
3. Write the starting source address in the **SAR** register for channel x.
   Note: *The values in the LLI.SARx register locations of each of the Linked List Items (LLIs) set up in memory, although fetched during an LLI fetch, are not used.*
4. Write the channel configuration information into the **CFG** register for channel x.
   a) Designate the handshaking interface type (hardware or software) for the source and destination peripherals; this is not required for memory.
      This step requires programming the HS_SEL_SRC/HS_SEL_DST bits. Writing a 0 activates the hardware handshaking interface to handle source/destination requests for the specific channel. Writing a 1 activates the software handshaking interface source/destination requests.
   b) If the hardware handshaking interface is activated for the source or destination peripheral, assign the handshaking interface to the source and destination peripheral; this requires programming the SRC_PER and DEST_PER bits, respectively.
5. Make sure that the LLI.CTLx register locations of all LLIs in memory (except the last) are set as shown in Row 7 of **Table 5-8**, while the LLI.CTLx register of the last Linked List item must be set as described in Row 1 or Row 5 of **Table 5-8**. Figure 7-1 shows a Linked List example with two list items.
6. Ensure that the LLI.LLPx register locations of all LLIs in memory (except the last) are non-zero and point to the next Linked List Item.
7. Ensure that the LLI.DARx register location of all LLIs in memory point to the start destination block address preceding that LLI fetch.
8. Ensure that the LLI.CTLx.DONE fields of the LLI.CTLx register locations of all LLIs in memory are cleared.
9. If source status fetching is enabled (CFGx.SS_UPD_EN is enabled), program the **SSTATAR** register so that the source status information can be fetched from the location pointed to by the **SSTATAR**. For conditions under which the source status information is fetched from system memory, refer to the Write Back column of **Table 5-8**.
10. If destination status fetching is enabled (CFGx.DS_UPD_EN is enabled), program the **DSTATAR** register so that the destination status information can be fetched from the location pointed to by the **DSTATAR** register. For conditions under which the destination status information is fetched from system memory, refer to the Write Back column of **Table 5-8**.

11. If gather is enabled (**CTL**.SRC_GATHER_EN = 1), program the **SGR** register for channel x.
12. If scatter is enabled (**CTL**.DST_SCATTER_EN = 1), program the **DSR** register for channel x.
13. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers: CLEARTFR, CLEARBLOCK, CLEARSRCTRAN, CLEARDSTTRAN, and CLEARERR. Reading the Interrupt Raw Status and Interrupt Status registers confirms that all interrupts have been cleared.
14. Program the **CTL** and **CFG** registers according to Row 7, as shown in **Table 5-8**.
15. Program the **LLP** register with **LLP**(0), the pointer to the first Linked List item.
16. Finally, enable the channel by writing a 1 to the **GPDMA0_CHENREG**.CH_EN bit; the transfer is performed. Ensure that bit 0 of the **GPDMA0_DMACFGREG** register is enabled.
17. The GPDMA fetches the first LLI from the location pointed to by **LLP**(0).
    Note: *The LLI.SARx, LLI.DARx, LLI.LLPx, and LLI.CTLx registers are fetched. The LLI.SARx register - although fetched - is not used.*
18. Source and destination request single and burst GPDMA transactions in order to transfer the block of data (assuming non-memory peripherals). The GPDMA acknowledges at the completion of every transaction (burst and single) in the block and carries out the block transfer.
19. Once the block of data is transferred, the source status information is fetched from the location pointed to by the **SSTATAR** register and stored in the **SSTAT** register if CFGx.SS_UPD_EN is enabled. For conditions under which the source status information is fetched from system memory, refer to the Write Back column of **Table 5-8**.
    The destination status information is fetched from the location pointed to by the **DSTATAR** register and stored in the **DSTAT** register if CFGx.DS_UPD_EN is enabled. For conditions under which the destination status information is fetched from system memory, refer to the Write Back column of **Table 5-8**.
20. The **CTLxH** register is written out to system memory. For conditions under which the **CTLxH** register is written out to system memory, refer to the Write Back column of **Table 5-8**.
    The **CTLxH** register is written out to the same location on the same layer (**LLP**.LMS) where it was originally fetched; that is, the location of the **CTL** register of the linked list item fetched prior to the start of the block transfer. Only the **CTLxH** register is written out, because only the **CTL**.BLOCK_TS and **CTL**.DONE fields have been updated by hardware within the GPDMA. The LLI.CTLx.DONE bit is asserted to indicate block completion. Therefore, software can poll the LLI.**CTL**.DONE bit field of the **CTL** register in the LLI to ascertain when a block transfer has completed.
    Note: *Do not poll the CTLx.DONE bit in the GPDMA memory map. Instead, poll the LLI.CTLx.DONE bit in the LLI for that block. If the polled LLI.CTLx.DONE bit is asserted, then this block transfer has completed. This LLI.CTLx.DONE bit was cleared at the start of the transfer (Step 8).*

21. The **SSTAT** register is now written out to system memory if CFGx.SS_UPD_EN is enabled. It is written to the SSTATx register location of the LLI pointed to by the previously saved LLPx.LOC register.
    The **DSTAT** register is now written out to system memory if CFGx.DS_UPD_EN is enabled. It is written to the DSTATx register location of the LLI pointed to by the previously saved LLPx.LOC register.
    The end-of-block interrupt, int_block, is generated after the write-back of the control and status registers has completed.
    Note: *The write-back location for the control and status registers is the LLI pointed to by the previous value of the LLPx.LOC register, not the LLI pointed to by the current value of the LLPx.LOC register.*
22. The GPDMA reloads the **SAR** register from the initial value. Hardware sets the block-complete interrupt. The GPDMA samples the row number, as shown in **Table 5-8**. If the GPDMA is in Row 1 or Row 5, then the DMA transfer has completed. Hardware sets the transfer complete interrupt and disables the channel. You can either respond to the Block Complete or Transfer Complete interrupts, or poll for the transfer complete raw interrupt status register (RAWTFR[n], n = channel number) until it is set by hardware, in order to detect when the transfer is complete. Note that if this polling is used, software must ensure that the transfer complete interrupt is cleared by writing to the Interrupt Clear register, CLEARTFR[n], before the channel is enabled. If the GPDMA is not in Row 1 or Row 5 as shown in **Table 5-8**, the following steps are performed.
23. The DMA transfer proceeds as follows:
    a) If interrupts are enabled (**CTL**.INT_EN = 1) and the block-complete interrupt is unmasked (MASKBLOCK[x] = 1_B, where x is the channel number), hardware sets the block-complete interrupt when the block transfer has completed. It then stalls until the block-complete interrupt is cleared by software. If the next block is to be the last block in the DMA transfer, then the block-complete ISR (interrupt service routine) should clear the CFGx.RELOAD_SRC source reload bit. This puts the GPDMA into Row 1, as shown in **Table 5-8**. If the next block is not the last block in the DMA transfer, then the source reload bit should remain enabled to keep the GPDMA in Row 7, as shown in **Table 5-8**.
    b) If interrupts are disabled (**CTL**.INT_EN = 0) or the block-complete interrupt is masked (MASKBLOCK[x] = 0_B, where x is the channel number), then hardware does not stall until it detects a write to the block-complete interrupt clear register; instead, it immediately starts the next block transfer. In this case, software must clear the source reload bit, CFGx.RELOAD_SRC in order to put the device into Row 1 of **Table 5-8** before the last block of the DMA transfer has completed.
24. The GPDMA fetches the next LLI from memory location pointed to by the current **LLP** register and automatically reprograms the **DAR**, **CTL**, and **LLP** channel registers. Note that the **SAR** is not reprogrammed, since the reloaded value is used for the next DMA block transfer. If the next block is the last block of the DMA transfer, then the

CTL and **LLP** registers just fetched from the LLI should match Row 1 or Row 5 of **Table 5-8**.

The DMA transfer might look like that shown in **Figure 5-32**.

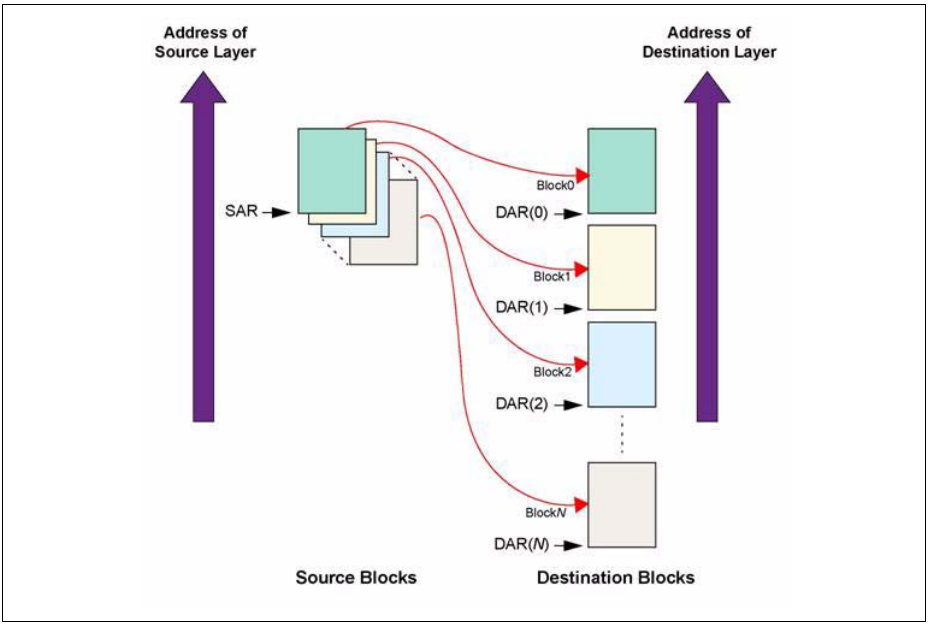


Figure 5-32 Multi-Block DMA Transfer with Source Address Auto-Reloaded and Linked List Destination Address

The DMA transfer flow is shown in **Figure 5-33**.

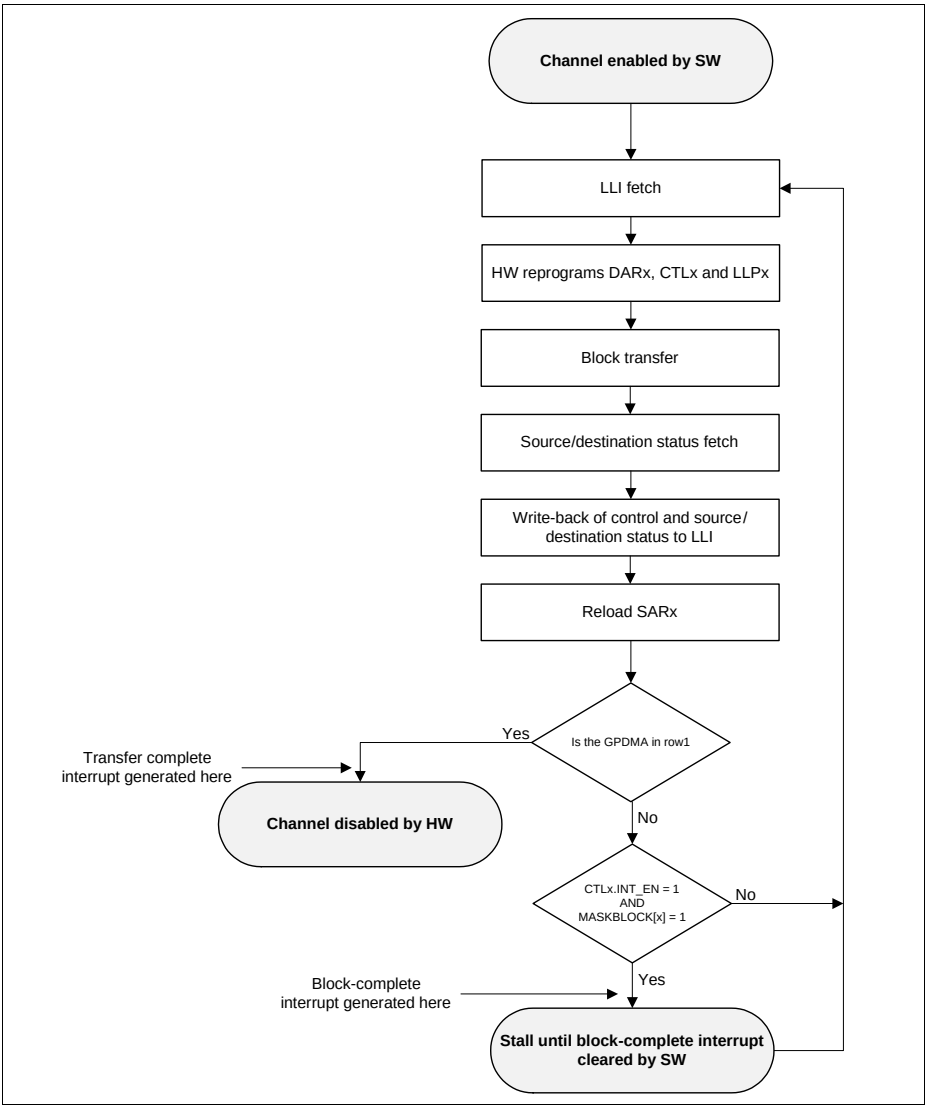


Figure 5-33 DMA Transfer Flow for Source Address Auto-Reloaded and Linked List Destination Address

5.3.3.5 Multi-Block Transfer with Source Address Auto-Reloaded and Contiguous Destination Address (Row 3)

This type of transfer is supported by channels 0 and 1 only.

1. Read the Channel Enable register (see **GPDMA0_CHENREG**) to choose a free (disabled) channel.
2. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers: CLEARTFR, CLEARBLOCK, CLEARSRCTRAN, CLEARDSTTRAN, and CLEARERR. Reading the Interrupt Raw Status and Interrupt Status registers confirms that all interrupts have been cleared.
3. Program the following channel registers:
   a) Write the starting source address in the **SAR** register for channel x.
   b) Write the starting destination address in the **DAR** register for channel x.
   c) Program **CTL** and **CFG** according to Row 3, shown in **Table 5-8**. Program the **LLP** register with 0.
   d) Write the control information for the DMA transfer in the **CTL** register for channel x.
   e) If gather is enabled (**CTL**.SRC_GATHER_EN = 1), program the **SGR** register for channel x.
   f) If scatter is enabled (**CTL**.DST_SCATTER_EN = 1), program the **DSR** register for channel x.
   g) Write the channel configuration information into the **CFG** register for channel x.
      1. Designate the handshaking interface type (hardware or software) for the source and destination peripherals; this is not required for memory.
      This step requires programming the HS_SEL_SRC/HS_SEL_DST bits, respectively. Writing a 0 activates the hardware handshaking interface to handle source/destination requests for the specific channel. Writing a 1 activates the software handshaking interface to handle source/destination requests.
      2. If the hardware handshaking interface is activated for the source or destination peripheral, assign the handshaking interface to the source and destination peripheral. This requires programming the SRC_PER and DEST_PER bits, respectively.
4. After the GPDMA channel has been programmed, enable the channel by writing a 1 to the **GPDMA0_CHENREG**.CH_EN bit. Ensure that bit 0 of the **GPDMA0_DMACFGREG** register is enabled.
5. Source and destination request single and burst GPDMA transactions to transfer the block of data (assuming non-memory peripherals). The GPDMA acknowledges at the completion of every transaction (burst and single) in the block and carries out the block transfer.
6. When the block transfer has completed, the GPDMA reloads the **SAR** register; the **DAR** register remains unchanged. Hardware sets the block-complete interrupt. The GPDMA then samples the row number, as shown in **Table 5-8**. If the GPDMA is in Row 1, then the DMA transfer has completed. Hardware sets the transfer-complete interrupt and disables the channel. You can either respond to the Block Complete or

Transfer Complete interrupts, or poll for the transfer complete raw interrupt status register (RAWTFR[n], n = channel number) until it is set by hardware, in order to detect when the transfer is complete. Note that if this polling is used, software must ensure that the transfer complete interrupt is cleared by writing to the Interrupt Clear register, CLEARTFR[n], before the channel is enabled. If the GPDMA is not in Row 1, the next step is performed.

7. The DMA transfer proceeds as follows:
   a) If interrupts are enabled (**CTL**.INT_EN = 1) and the block-complete interrupt is unmasked (MASKBLOCK[x] = 1_B, where x is the channel number), hardware sets the block-complete interrupt when the block transfer has completed. It then stalls until the block-complete interrupt is cleared by software. If the next block is to be the last block in the DMA transfer, then the block-complete ISR (interrupt service routine) should clear the source reload bit, CFGx.RELOAD_SRC. This puts the GPDMA into Row 1, as shown in **Table 5-8**. If the next block is not the last block in the DMA transfer, then the source reload bit should remain enabled to keep the GPDMA in Row 3, as shown in **Table 5-8**.
   b) If interrupts are disabled (**CTL**.INT_EN = 0) or the block-complete interrupt is masked (MASKBLOCK[x] = 0_B, where x is the channel number), then hardware does not stall until it detects a write to the block-complete interrupt clear register; instead, it starts the next block transfer immediately. In this case, software must clear the source reload bit, CFGx.RELOAD_SRC, to put the device into Row 1 of **Table 5-8** before the last block of the DMA transfer has completed.

The transfer is similar to that shown in **Figure 5-34**.

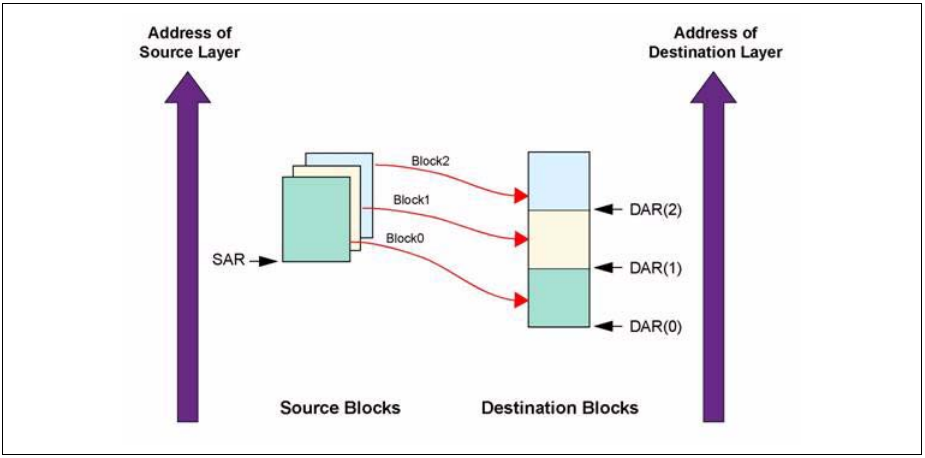


Figure 5-34 Multi-Block DMA Transfer with Source Address Auto-Reloaded and Contiguous Destination Address

The DMA transfer flow is shown in **Figure 5-35**.

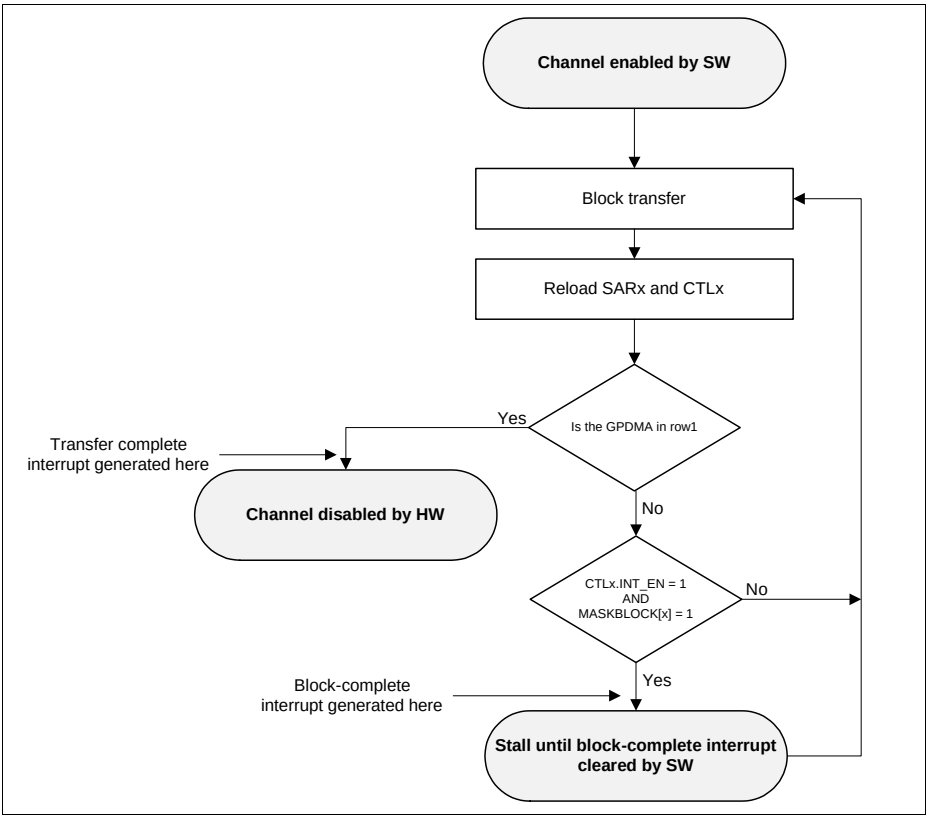


Figure 5-35 DMA Transfer Flow for Source Address Auto-Reloaded and Linked List Destination Address

5.3.3.6 Multi-Block DMA Transfer with Linked List for Source and Contiguous Destination Address (Row 8)

This type of transfer is supported by channels 0 and 1 only.

1. Read the Channel Enable register (see **GPDMA0_CHENREG**) to choose a free (disabled) channel.
2. Set up the linked list in memory. Write the control information in the LLI.**CTL** register location of the block descriptor for each LLI in memory (see **Figure 5-23**) for channel x.
3. Write the starting destination address in the **DAR** register for channel x.
   Note: *The values in the LLI.DARx register location of each Linked List Item (LLI) in memory, although fetched during an LLI fetch, are not used.*
4. Write the channel configuration information into the **CFG** register for channel x.
   1. Designate the handshaking interface type (hardware or software) for the source and destination peripherals; this is not required for memory.
   This step requires programming the HS_SEL_SRC/HS_SEL_DST bits. Writing a 0 activates the hardware handshaking interface to handle source/destination requests for the specific channel. Writing a 1 activates the software handshaking interface to handle source/destination requests.
   2. If the hardware handshaking interface is activated for the source or destination peripheral, assign the handshaking interface to the source and destination peripherals. This requires programming the SRC_PER and DEST_PER bits, respectively.
5. Ensure that all LLI.CTLx register locations of the LLI (except the last) are set as shown in Row 8 of **Table 5-8**, while the LLI.CTLx register of the last Linked List item must be set as described in Row 1 or Row 5 of **Table 5-8**. **Figure 5-23** shows a Linked List example with two list items.
6. Ensure that the LLI.LLPx register locations of all LLIs in memory (except the last) are non-zero and point to the next Linked List Item.
7. Ensure that the LLI.SARx register location of all LLIs in memory point to the start source block address preceding that LLI fetch.
8. Ensure that the LLI.CTLx.DONE fields of the LLI.CTLx register locations of all LLIs in memory are cleared.
9. If source status fetching is enabled (CFGx.SS_UPD_EN is enabled), program the **SSTATAR** register so that the source status information can be fetched from the location pointed to by **SSTATAR**. For conditions under which the source status information is fetched from system memory, refer to the Write Back column of **Table 5-8**.
10. If destination status fetching is enabled (CFGx.DS_UPD_EN is enabled), program the **DSTATAR** register so that the destination status information can be fetched from the location pointed to by the **DSTATAR** register. For conditions under which the destination status information is fetched from system memory, refer to the Write Back column of **Table 5-8**.

11. If gather is enabled (**CTL**.SRC_GATHER_EN = 1), program the **SGR** register for channel x.
12. If scatter is enabled (**CTL**.DST_SCATTER_EN = 1), program the **DSR** register for channel x.
13. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers: CLEARTFR, CLEARBLOCK, CLEARSRCTRAN, CLEARDSTTRAN, and CLEARERR. Reading the Interrupt Raw Status and Interrupt Status registers confirms that all interrupts have been cleared.
14. Program the **CTL** and **CFG** registers according to Row 8, as shown in **Table 5-8**.
15. Program the **LLP** register with **LLP**(0), the pointer to the first Linked List item.
16. Finally, enable the channel by writing a 1 to the **GPDMA0_CHENREG**.CH_EN bit; the transfer is performed. Ensure that bit 0 of the **GPDMA0_DMACFGREG** register is enabled.
17. The GPDMA fetches the first LLI from the location pointed to by **LLP**(0).
    Note: *The LLI.SARx, LLI.DARx, LLI.LLPx, and LLI.CTLx registers are fetched. The LLI.DARx register location of the LLI - although fetched - is not used. The* **DAR** *register in the GPDMA remains unchanged.*
18. Source and destination request single and burst GPDMA transactions to transfer the block of data (assuming non-memory peripherals). The GPDMA acknowledges at the completion of every transaction (burst and single) in the block and carries out the block transfer.
19. Once the block of data is transferred, the source status information is fetched from the location pointed to by the **SSTATAR** register and stored in the **SSTAT** register if CFGx.SS_UPD_EN is enabled. For conditions under which the source status information is fetched from system memory, refer to the Write Back column of **Table 5-8**.The destination status information is fetched from the location pointed to by the **DSTATAR** register and stored in the **DSTAT** register if CFGx.DS_UPD_EN is enabled. For conditions under which the destination status information is fetched from system memory, refer to the Write Back column of **Table 5-8**.
20. The **CTLxH** register is written out to system memory. For conditions under which the **CTLxH** register is written out to system memory, refer to the Write Back column of **Table 5-8**.The **CTLxH** register is written out to the same location on the same layer (LLPx.LMS) where it was originally fetched; that is, the location of the **CTL** register of the linked list item fetched prior to the start of the block transfer. Only the second word of the **CTL** register is written out, **CTLxH**, because only the **CTL**.BLOCK_TS and **CTL**.DONE fields have been updated by hardware within the GPDMA. Additionally, the **CTL**.DONE bit is asserted to indicate block completion. Therefore, software can poll the LLI.**CTL**.DONE bit field of the **CTL** register in the LLI to ascertain when a block transfer has completed.
    Note: Do not poll the **CTL**.DONE bit in the GPDMA memory map. Instead, poll the LLI.CTLx.DONE bit in the LLI for that block. If the polled LLI.CTLx.DONE bit is asserted, then this block transfer has completed. This LLI.CTLx.DONE bit was cleared at the start of the transfer (Step 8).

21. The **SSTAT** register is now written out to system memory if CFGx.SS_UPD_EN is enabled. It is written to the **SSTAT** register location of the LLI pointed to by the previously saved LLPx.LOC register.The **DSTAT** register is now written out to system memory if CFGx.DS_UPD_EN is enabled. It is written to the **DSTAT** register location of the LLI pointed to by the previously saved LLPx.LOC register.The end-of-block interrupt, int_block, is generated after the write-back of the control and status registers has completed.
    Note: The write-back location for the control and status registers is the LLI pointed to by the previous value of the LLPx.LOC register, not the LLI pointed to by the current value of the LLPx.LOC register.
22. The GPDMA does not wait for the block interrupt to be cleared, but continues and fetches the next LLI from the memory location pointed to by the current **LLP** register and automatically reprograms the **SAR**, **CTL**, and **LLP** channel registers. The **DAR** register is left unchanged. The DMA transfer continues until the GPDMA samples that the **CTL** and **LLP** registers at the end of a block transfer match those described in Row 1 or Row 5 of **Table 5-8** (as discussed earlier). The GPDMA then knows that the previously transferred block was the last block in the DMA transfer.

The GPDMA transfer might look like that shown in **Figure 5-36**. Note that the destination address is decrementing.

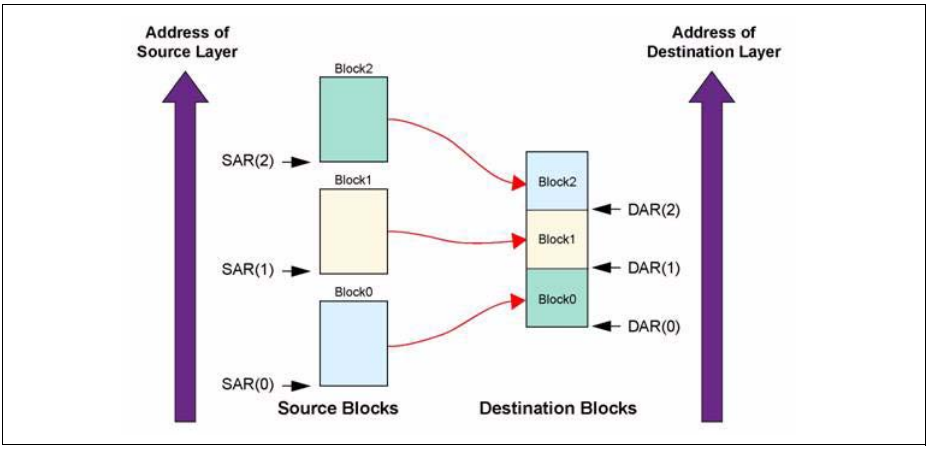


Figure 5-36 Multi-Block DMA Transfer with Linked List Source Address and Contiguous Destination Address

The DMA transfer flow is shown in **Figure 5-37**.

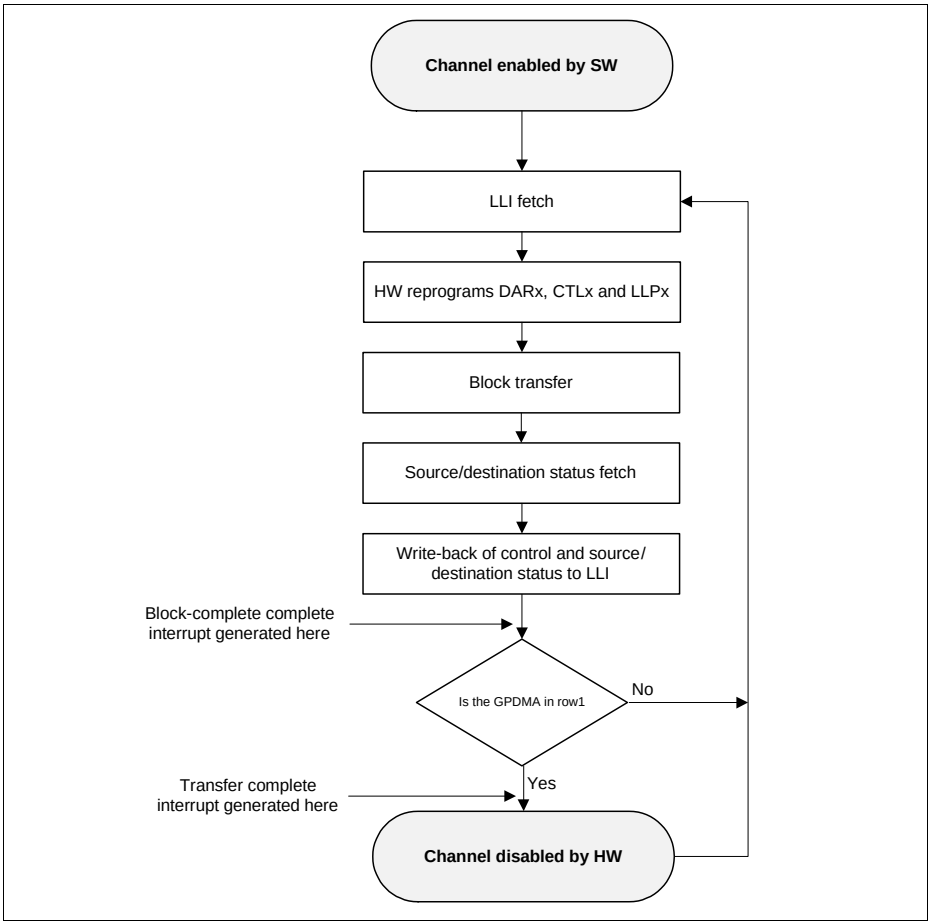


Figure 5-37 DMA Transfer Flow for Source Address Auto-Reloaded and Linked List Destination Address

5.3.3.7 Programming Example for Linked List Multi-Block Transfer

The flow diagram in **Figure 5-38** shows an overview of programming the DMA.

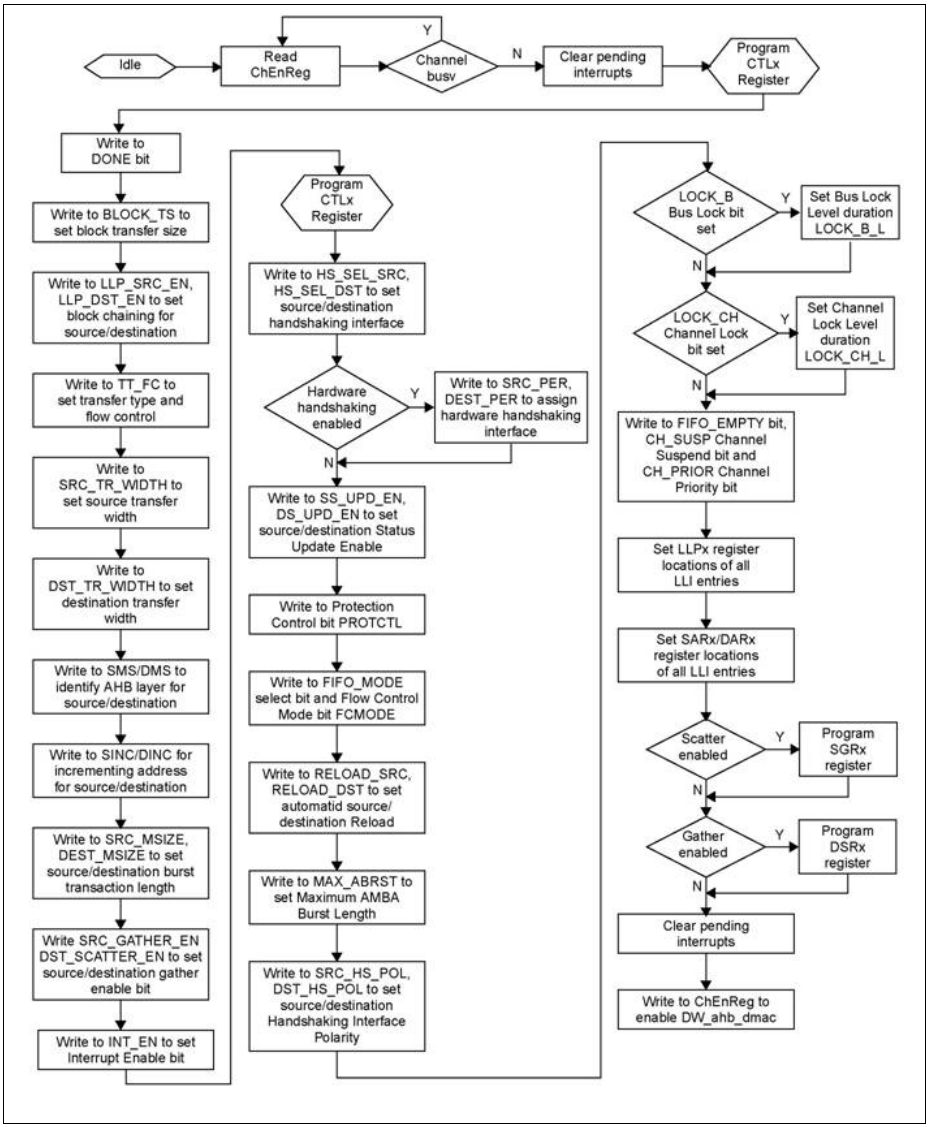


Figure 5-38 Flowchart for DMA Programming Example

This section explains the step-by-step programming of the GPDMA. The example demonstrates row 10 of **Table 5-8** for Multi-Block Transfer with Linked List for Source

and Linked List for Destination. This example uses the GPDMA to move four blocks of contiguous data from source to destination memory using the Linked List feature.

1. Set up the chain of Linked List items — otherwise known as block descriptors — in memory. Write the control information in the LLI.CTLx register location of the block descriptor for each LLI in memory for Channel 1. In the LLI.CTLx register, the following is programmed:
   a) Set up the transfer type for a memory-to-memory transfer:
      - `ctlx[22:20] = 000;`
   b) Set up the transfer characteristics:
      1. Transfer width for the source in the SRC_TR_WIDTH field
      - `ctlx[6:4] = 001;`
      2. Transfer width for the destination in the DST_TR_WIDTH field
      - `ctlx[3:1] = 001;`
      3. Source master layer in the SMS field where the source resides
      - `ctlx[26:25] = 00;`
      4. Destination master layer in the DMS field where the destination resides
      - `ctlx[24:23] = 00;`
      5. Incrementing address for the source in the SINC field
      - `ctlx[10:9] = 00;`
      6. Incrementing address for the destination in the DINC field
      - `ctlx[8:7] = 00;`
2. Write the channel configuration information into the CFGx register for Channel 1:
   a) HS_SEL_SRC/HS_SEL_DST bits select which of the handshaking interfaces—hardware or software—is active for source requests on this channel.
      - `cfgx[11] = 0;`
      - `cfgx[10] = 0;`

      These settings are ignored because both the source and destination are memory types
   b) If the hardware handshaking interface is activated for the source or destination peripheral, assign the handshaking interface to the source and destination peripheral by programming the SRC_PER and DEST_PER bits:
      - `cfgx[46:43] = 0;`
      - `cfgx[42:39] = 0;`

      These settings are ignored because both the source and destination are memory types.
3. The following For loop, shown as a programming example, sets the following:
   a) LLI.LLPx register locations of all LLI entries in memory (except the last) to non-zero and point to the base address of the next Linked List Item
   b) LLI.SARx/LLI.DARx register locations of all LLI entries in memory point to the start source/destination block address preceding that LLI fetch. The For statement below configures the LLPx entries:

      ```
      for(i=0 ; i < 4 ; i=i+1) begin
      ```

```
if (i == 3)llpx = 0; // end of LLI
elsellpx = llp_addr + 20; // start of next LLI

//-: Program SAR"AHB_MASTER.write(0, llp_addr, sarx,
AhbWord32Attrb, handle[0]);

//-: Program DAR"AHB_MASTER.write(0, (llp_addr + 4), darx,
AhbWord32Attrb, handle[0]);

//-: Program LLP"AHB_MASTER.write(0, (llp_addr + 8), llpx,
AhbWord32Attrb, handle[0]);

//-: Program CTL"AHB_MASTER.write(0, (llp_addr + 12),
ctlx[31:0], AhbWord32Attrb, handle[0]);"AHB_MASTER.write(0,
(llp_addr + 16), ctlx[63:32], AhbWord32Attrb, handle[0]);//
update pointersllp_addr = llp_addr + 20; // start of next LLI
// 4 16-bit words each with scatter/gather interval in each
block
// ( will work only with scatter_gather count of 2)
sarx = sarx + 24;
darx = darx + 24;
end
```

4. If Gather is enabled (**CTL**.SRC_GATHER_EN = 1), program the SGRx register for Channel 1.
5. If Scatter is enabled (**CTL**.DST_SCATTER_EN = 1) program the DSRx register for Channel 1.
6. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers.
7. Finally, enable the channel by writing a 1 to the CHENREG.CH_EN bit; the transfer is performed

5.3.4 Abnormal Transfer Termination

A GPDMA DMA transfer may be terminated abruptly by software by clearing the channel enable bit, **CHENREG**.CH_EN or by clearing the global enable bit in the GPDMA Configuration Register (**DMACFGREG**[0]).

If a transfer is in progress while a channel is disabled, abnormal transfer termination and data corruption occurs. Also the transfer acknowledge may be lost. Therefore this must be avoided.

Attention: *Disabling a channel via software prior to completing a transfer is not supported.*

5.4 Power, Reset and Clock

The GPDMA unit is inside the power core domain, therefore no special considerations about power up or power down sequences need to be taken. For an explanation about the different power domains, please address the SCU (System Control Unit) chapter.

Additionally, if a GPDMA unit is not needed, it can be set in reset via the PRSET2.DMAyRS bitfield (address the SCU chapter for a full description).

The clock used for the GPDMA unit is described on the SCU chapter as f_{DMA}. Please address the specific section under the SCU chapter for a detailed description on the clock configuration schemes.

5.5 Initialization and System Dependencies

The generic initialization sequence for an application that is using the GPDMA, should be the following:

1st Step: Release reset of the GPDMA, via the specific SCU bitfield on the PRCLR2 register.

2nd Step: If the GPDMA is already under use (step 1 was not performed) do the following steps:

- read the channel Enable register to choose a free channel, **CHENREG**. Clear also any pending requests of the specific channel, by writing into the **CLEARTFR**, **CLEARBLOCK**, **CLEARSRCTRAN**, **CLEARDSTTRAN** and **CLEARERR**
- confirm that all the interrupts have been cleared via the Status and RAW registers.

3rd Step: Configure the GPDMA channels accordingly with the wanted transfer type:

- Configure the starting source address and starting destination address, on the **SAR** and **DAR**, respectively.
- Configure the type of transfer that are going to be used via the **LLP**, **CTL** and **CFG** registers.

4th Step: Enable the GPDMA channel, by setting the specific bitfield on the **CHENREG**.

5th Step: Configure the DLR (DMA Line Router) block to map the DMA requests from the peripherals to the wanted DMA request lines (if not previously done).

6th Step: Configure the peripherals that are linked with DMA requests.

7th Step: Enable the specific Service requests on the peripheral blocks.

8th Step: Start the peripheral(s)

*Note: This is a generic channel initialization example. Please address **Section 5.3** for a complete description and examples of how to control the complete flow for a GPDMA* channel.

5.6 Registers

This chapter includes information on how to program the GPDMA.

Notes

1. *There are references to software parameters throughout this chapter. The software parameters are the field names in each register description table and are prefixed by the register name; for example, the Block Transfer Size field in the Control register for channel x of GPDMA0 is designated as "GPDMA0_CTLxH.BLOCK_TS"*

Table 5-9 Registers Address Space

Module	Base Address	End Address	Note
GPDMA0_CH0	5001 4000$_H$	5001 4054$_H$	
GPDMA0_CH1	5001 4058$_H$	5001 40AC$_H$	
GPDMA0_CH2	5001 40B0$_H$	5001 4104$_H$	
GPDMA0_CH3	5001 4108$_H$	5001 415C$_H$	
GPDMA0_CH4	5001 4160$_H$	5001 41B4$_H$	
GPDMA0_CH5	5001 41B8$_H$	5001 420C$_H$	
GPDMA0_CH6	5001 4210$_H$	5001 4264$_H$	
GPDMA0_CH7	5001 4268$_H$	5001 42BC$_H$	
GPDMA0	5001 42C0$_H$	5001 7FFF$_H$	
GPDMA1_CH0	5001 8000$_H$	5001 8054$_H$	
GPDMA1_CH1	5001 8058$_H$	5001 80AC$_H$	
GPDMA1_CH2	5001 80B0$_H$	5001 8104$_H$	
GPDMA1_CH3	5001 8108$_H$	5001 815C$_H$	
GPDMA1	5001 82C0$_H$	5001 FFFF$_H$	

Table 5-10 Register Overview

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
ChannelRegisters					
SARx	Source Address Register	0000$_H$ + x*58$_H$	U, PV	U, PV	Page 5-83
DARx	Destination Address Register	0008$_H$ + x*58$_H$	U, PV	U, PV	Page 5-84

Table 5-10 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
Control Registers					
CTLxH	Control Register High	$001C_H$ + x*$5C_H$	U, PV	U, PV	Page 5-88
CTLxL	Control Register Low	0018_H+ x*58_H	U, PV	U, PV	Page 5-90
LLPx	Linked List Pointer Register	0010_H+ x*58_H	U, PV	U, PV	Page 5-86
SSTATx	Source Status Register	0020_H+ x*58_H	U, PV	U, PV	Page 5-97
DSTATx	Destination Status Register	0028_H+ x*58_H	U, PV	U, PV	Page 5-98
SSTATARx	Source Status Register	0030_H+ x*58_H	U, PV	U, PV	Page 5-99
DSTATARx	Destination Status Register	0038_H+ x*58_H	U, PV	U, PV	Page 5-100
CFGxH	Configuration Register High	0044_H+ x*$5C_H$	U, PV	U, PV	Page 5-101
CFGxL	Configuration Register Low	0040_H+ x*58_H	U, PV	U, PV	Page 5-106
SGRx	Source Gather Register	0048_H+ x*58_H	U, PV	U, PV	Page 5-113
DSRx	Destination Scatter Register	0050_H+ x*58_H	U, PV	U, PV	Page 5-114
Interrupt Registers					
RAW* with *TFR, *BLOCK, *SRCTRAN, *DSTTRAN, *ERR	Interrupt Raw Status Registers	$02C0_H$ - $02E0_H$	U, PV	U, PV	Page 5-117

Table 5-10 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
STATUS* with *TFR, *BLOCK, *SRCTRAN, *DSTTRAN, *ERR	Interrupt Status Registers	$02E8_H$ - 0308_H	U, PV	U, PV	Page 5-120
MASK* with *TFR, *BLOCK, *SRCTRAN, *DSTTRAN, *ERR	Interrupt Mask Registers	0310_H - 0330_H	U, PV	U, PV	Page 5-122
CLEAR* with *TFR, *BLOCK, *SRCTRAN, *DSTTRAN, *ERR	Interrupt Clear Registers	0338_H - 0358_H	U, PV	U, PV	Page 5-125
STATUSINT	Combined Interrupt Status Register	0360_H	U, PV	U, PV	Page 5-126
Software Handshaking Registers					
REQSRCREG	Source Software Transaction Request Register	0368_H	U, PV	U, PV	Page 5-128
REQDSTREG	Destination Software Transaction Request Register	0370_H	U, PV	U, PV	Page 5-129
SGLREQSRCREG	Single Source Transaction Request Register	0378_H	U, PV	U, PV	Page 5-131
SGLREQDSTREG	Single Destination Transaction Request Register	0380_H	U, PV	U, PV	Page 5-132
LSTSRCREG	Last Source Transaction Request Register	0388_H	U, PV	U, PV	Page 5-134

Table 5-10 Register Overview (cont'd)

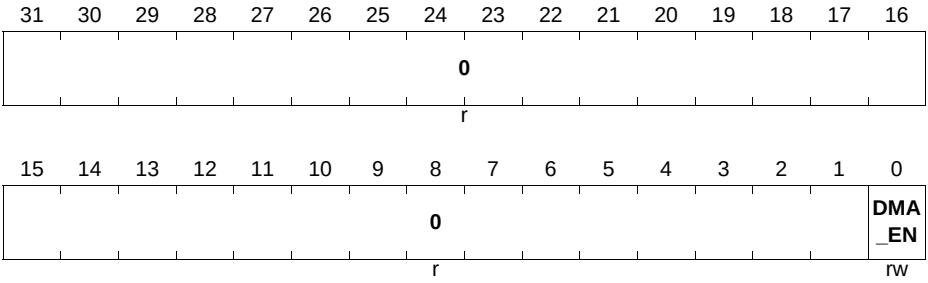
Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
LSTDSTREG	Last Destination Transaction Request Register	0390_H	U, PV	U, PV	Page 5-135
Configuration and Channel Enable Registers					
DMACFGREG	Configuration Register	0398_H	U, PV	U, PV	Page 5-80
CHENREG	Channel Enable Register	$03A0_H$	U, PV	U, PV	Page 5-80
Miscellaneous GPDMA Registers					
ID	GPDMA Module ID	$03A8_H$	U, PV	U, PV	Page 5-138
Reserved	Reserved	$03B0_H$ - $03F4_H$	nBE	nBE	
TYPE	GPDMA Component Type	$03F8_H$	U, PV	U, PV	Page 5-138
VERSION	GPDMA Component Version	$03FC_H$	U, PV	U, PV	Page 5-139
Reserved	Reserved	0400_H - $7FFC_H$	nBE	nBE	

5.6.1 Configuration and Channel Enable Registers

DMACFGREG

This register is used to enable the GPDMA, which must be done before any channel activity can begin.

GPDMA0_DMACFGREG
GPDMA Configuration Register (398_H) **Reset Value: $0000\ 0000_H$**
GPDMA1_DMACFGREG
GPDMA Configuration Register (398_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
0	[31:1]	r	**Reserved**
DMA_EN	0	rw	**GPDMA Enable bit.** 0_B GPDMA Disabled 1_B GPDMA Enabled.

If the global channel enable bit is cleared while any channel is still active, then DMACFGREG.DMA_EN still returns 1 to indicate that there are channels still active until hardware has terminated all activity on all channels, at which point the DMACFGREG.DMA_EN bit returns 0.

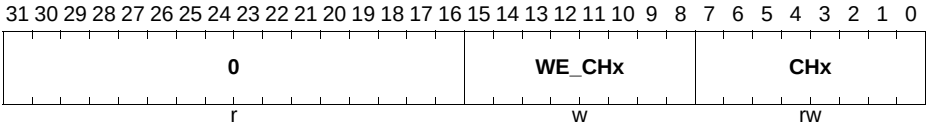
CHENREG

This is the GPDMA “Channel Enable Register”. If software needs to set up a new channel, then it can read this register in order to find out which channels are currently inactive; it can then enable an inactive channel with the required priority.

All bits of this register are cleared to 0 when the global GPDMA channel enable bit, **DMACFGREG**[0], is 0. When the global channel enable bit is 0, then a write to the **CHENREG** register is ignored and a read will always read back 0.

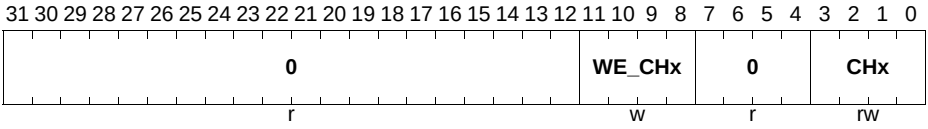
The channel enable bit, **CHENREG**.CH_EN, is written only if the corresponding channel write enable bit, **CHENREG**.CH_EN_WE, is asserted on the same AHB write transfer. For example, writing hex 01x1 writes a 1 into **CHENREG**[0], while **CHENREG**[7:1] remains unchanged. Writing hex 00xx leaves **CHENREG**[7:0] unchanged. Note that a read-modified write is not required.

GPDMA0_CHENREG
GPDMA Channel Enable Register (3A0$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
0	[31:16]	r	**Reserved**
WE_CHx	[15:8]	w	**Channel enable write enable**
CHx	[7:0]	rw	**Enables/Disables the channel** Setting this bit enables a channel; clearing this bit disables the channel. 0_B Disable the Channel 1_B Enable the Channel The CHENREG.CH_EN bit is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer.

GPDMA1_CHENREG
GPDMA Channel Enable Register (3A0$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
0	[31:12]	r	**Reserved**
WE_CHx	[11:8]	w	**Channel enable write enable**
0	[7:4]	r	**Reserved**
CHx	[3:0]	rw	**Enables/Disables the channel** Setting this bit enables a channel; clearing this bit disables the channel. 0_B Disable the Channel 1_B Enable the Channel The CHENREG.CH_EN bit is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer.

5.6.2 Channel Registers

The **SAR**, **DAR**, **LLP**, **CTL**, and **CFG** channel registers should be programmed prior to enabling the channel. However, if an LLI update occurs before commencing data transfer, **SAR** and **DAR** may not need to be programmed prior to enabling the channel; refer to rows 6 to 10 in **Table 5-8** . It is an illegal register access (see **Section 5.3.1**) when a write to the **SAR**, **DAR**, **LLP**, **CTL**, **SSTAT**, **DSTAT**, **SSTATAR**, **DSTATAR**, **SGR**, or **DSR** registers occurs when the channel is enabled.

SAR

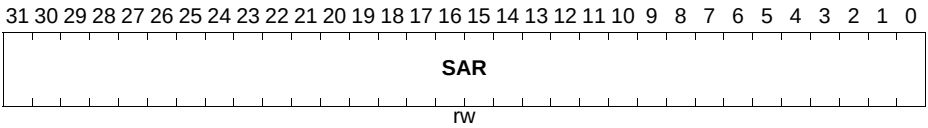
The starting source address is programmed by software before the DMA channel is enabled, or by an LLI update before the start of the DMA transfer. While the DMA transfer is in progress, this register is updated to reflect the source address of the current AHB transfer.

*Note: You must program the SAR address to be aligned to **CTL**.SRC_TR_WIDTH.*

For information on how the SARx is updated at the start of each DMA block for multi-block transfers, refer to **Table 5-8**.

GPDMA0_CHx_SAR (x=0-7)
Source Address Register for Channel x
(00_H + x*58_H) **Reset Value: 0000 0000_H**

GPDMA1_CHx_SAR (x=0-3)
Source Address Register for Channel x
(00_H + x*58_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
SAR	[31:0]	rw	**Current Source Address of DMA transfer** Updated after each source transfer. The SINC field in the **CTL** register determines whether the address increments, decrements, or is left unchanged on every source transfer throughout the block transfer. Reset: 0_D

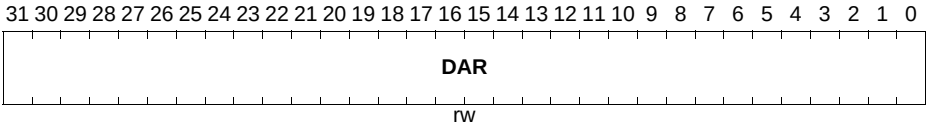
DAR

The starting destination address is programmed by software before the DMA channel is enabled, or by an LLI update before the start of the DMA transfer. While the DMA transfer is in progress, this register is updated to reflect the destination address of the current AHB transfer.

*Note: You must program the DAR to be aligned to **CTL**.DST_TR_WIDTH.*

GPDMA0_CHx_DAR (x=0-7)
Destination Address Register for Channel x
(08_H + x*58_H) **Reset Value: 0000 0000_H**

GPDMA1_CHx_DAR (x=0-3)
Destination Address Register for Channel x
(08_H + x*58_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
DAR	[31:0]	rw	**Current Destination address of DMA transfer** Updated after each destination transfer. The DINC field in the **CTL** register determines whether the address increments, decrements, or is left unchanged on every destination transfer throughout the block transfer. Reset: 0_D

Hardware Realignment of SAR/DAR Registers

In a particular circumstance, during contiguous multi-block DMA transfers, the destination address can become misaligned between the end of one block and the start of the next block. When this situation occurs, GPDMA re-aligns the destination address before the start of the next block.

Consider the following example. If the block length is 9, the source transfer width is 16 (halfword), and the destination transfer width is 32 (word) — the destination is programmed for contiguous block transfers — then the destination performs four word transfers followed by a halfword transfer to complete the block transfer to the destination. At the end of the destination block transfer, the address is aligned to a 16-bit transfer as the last AMBA transfer is halfword. This is misaligned to the programmed transfer size of 32 bits for the destination. However, for contiguous destination multi-block transfers, GPDMA re-aligns the DAR address to the nearest 32-bit address (next 32-bit address

upwards if address control is incrementing or next address downwards if address control is decrementing).

The destination address is automatically realigned by the GPDMA in the following DMA transfer setup scenario:

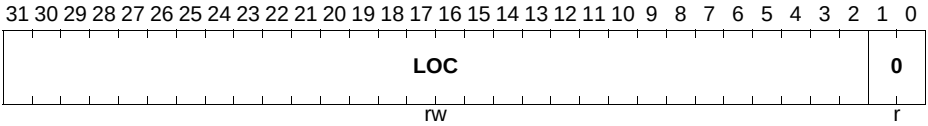
- Contiguous multi-block transfers on destination side, AND
- DST_TR_WIDTH > SRC_TR_WIDTH, AND
- (BLOCK_TS * SRC_TR_WIDTH)/DST_TR_WIDTH != integer (where SRC_TR_WIDTH, DST_TR_WIDTH is byte width of transfer)

LLP

You need to program this register to point to the first Linked List Item (LLI) in memory prior to enabling the channel if block chaining is enabled.

GPDMA0_CHx_LLP (x = 0-1)
Linked List Pointer Register for Channel x

(10_H + x*58_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
LOC	[31:2]	rw	**Starting Address In Memory** of next LLI if block chaining is enabled. Note that the two LSBs of the starting address are not stored because the address is assumed to be aligned to a 32-bit boundary. LLI accesses are always 32-bit accesses (Hsize = 2) aligned to 32-bit boundaries and cannot be changed or programmed to anything other than 32bit.
0	[1:0]	r	**Reserved**

The LLP register has two functions:

- The logical result of the equation LLP.LOC != 0 is used to set up the type of DMA transfer — single or multi-block. **Table 5-8** shows how the method of updating the channel registers is a function of LLP.LOC != 0. If LLP.LOC is set to 0, then transfers using linked lists are not enabled. This register must be programmed prior to enabling the channel in order to set up the transfer type.
- LLP.LOC != 0 contains the pointer to the next LLI for block chaining using linked lists. The LLPx register can also point to the address where write-back of the control and source/destination status information occur after block completion.

CTL

These registers contain fields that control the DMA transfer.

The CTLxH and CTLxL registers are part of the block descriptor (linked list item - LLI) when block chaining is enabled. It can be varied on a block-by-block basis within a DMA transfer when block chaining is enabled.

If status write-back is enabled, the upper control register, CTLxH, is written to the control register location of the LLI in system memory at the end of the block transfer.

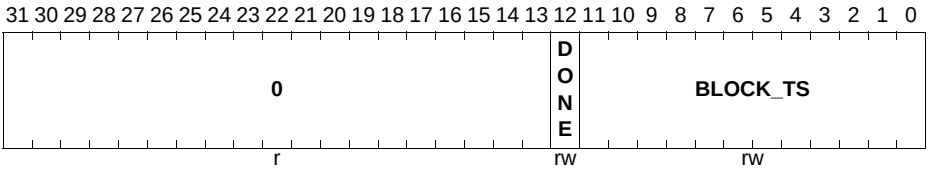
Note: You need to program these registers prior to enabling the channel.

CTLxH

Control Register High.

GPDMA0_CHx_CTLH (x=0-7)
Control Register High for Channel x
($1C_H$ + x*58_H) **Reset Value: 0000 0002$_H$**

GPDMA1_CHx_CTLH (x=0-3)
Control Register High for Channel x
($1C_H$ + x*58_H) **Reset Value: 0000 0002$_H$**



Field	Bits	Type	Description
0	[31:13]	r	**Reserved**

Field	Bits	Type	Description
DONE	12	rw	**Done bit** If status write-back is enabled, the upper word of the control register, CTLX[63:32], is written to the control register location of the Linked List Item (LLI) in system memory at the end of the block transfer with the done bit set. Software can poll the LLI CTLX.DONE bit to see when a block transfer is complete. The LLI CTLX.DONE bit should be cleared when the linked lists are set up in memory prior to enabling the channel. LLI accesses are always 32-bit accesses (Hsize = 2) aligned to 32-bit boundaries and cannot be changed or programmed to anything other than 32-bit.
BLOCK_TS	[11:0]	rw	**Block Transfer Size** When the GPDMA is the flow controller, the user writes this field before the channel is enabled in order to indicate the block size. The number programmed into BLOCK_TS indicates the total number of single transactions to perform for every block transfer; a single transaction is mapped to a single AMBA beat. **Width**: The width of the single transaction is determined by **CTL**.SRC_TR_WIDTH. Once the transfer starts, the read-back value is the total number of data items already read from the source peripheral, regardless of what is the flow controller.

CTLxL

Control Register Low.

GPDMA0_CHx_CTLL (x=0-1)
Control Register Low for Channel x

(18_H + x*58_H) **Reset Value: 0030 4801_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0			LLP_SRC_EN	LLP_DST_EN	0				TT_FC			0	DST_SCATTER_EN	SRC_GATHER_EN	SRC_MSIZE
r			rw	rw	r				rw			r	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SRC_MSIZE		DEST_MSIZE			SINC		DINC		SRC_TR_WIDTH			DST_TR_WIDTH			INT_EN
rw		rw			rw		rw		rw			rw			rw

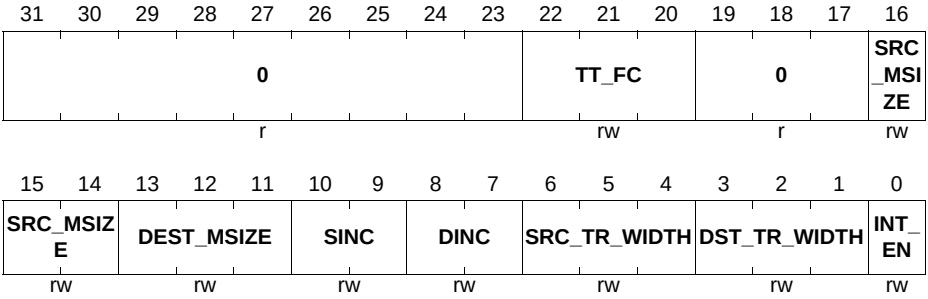
Field	Bits	Type	Description
0	[31:29]	r	**Reserved**
LLP_SRC_EN	28	rw	**Linked List Pointer for Source Enable** Block chaining is enabled on the source side only if the LLP_SRC_EN field is high and LLPx.LOC is non-zero.
LLP_DST_EN	27	rw	**Linked List Pointer for Destination Enable** Block chaining is enabled on the destination side only if the LLP_DST_EN field is high and LLPx.LOC is non-zero.
0	[26:23]	r	**Reserved**
TT_FC	[22:20]	rw	**Transfer Type and Flow Control** The following transfer types are supported. • Memory to Memory • Memory to Peripheral • Peripheral to Memory • Peripheral to Peripheral Flow Control can be assigned to the GPDMA, the source peripheral, or the destination peripheral. **Table 5-13** lists the decoding for this field.

Field	Bits	Type	Description
0	19	r	**Reserved**
DST_SCATTER_EN	18	rw	**Destination scatter enable** 0_B Scatter disabled 1_B Scatter enabled Scatter on the destination side is applicable only when the DINC bit indicates an incrementing or decrementing address control.
SRC_GATHER_EN	17	rw	**Source gather enable** 0_B Gather disabled 1_B Gather enabled Gather on the source side is applicable only when the SINC bit indicates an incrementing or decrementing address control.
SRC_MSIZE	[16:14]	rw	**Source Burst Transaction Length** Number of data items, each of width SRC_TR_WIDTH, to be read from the source every time a source burst transaction request is made from either the corresponding hardware or software handshaking interface. **Table 5-11** lists the decoding for this field; *Note: This value is not related to the AHB bus master HBURST bus.*
DEST_MSIZE	[13:11]	rw	**Destination Burst Transaction Length** Number of data items, each of width DST_TR_WIDTH, to be written to the destination every time a destination burst transaction request is made from either the corresponding hardware or software handshaking interface. **Table 5-11** lists the decoding for this field. *Note: This value is not related to the AHB bus master HBURST bus.*

Field	Bits	Type	Description
SINC	[10:9]	rw	**Source Address Increment** Indicates whether to increment or decrement the source address on every source transfer. If the device is fetching data from a source peripheral FIFO with a fixed address, then set this field to "No change". 00_B Increment 01_B Decrement $1x_B$ No change *Note: Incrementing or decrementing is done for alignment to the next CTLx.SRC_TR_WIDTH boundary.*
DINC	[8:7]	rw	**Destination Address Increment** Indicates whether to increment or decrement the destination address on every destination transfer. If your device is writing data to a destination peripheral FIFO with a fixed address, then set this field to "No change". 00_B Increment 01_B Decrement $1x_B$ No change *Note: Incrementing or decrementing is done for alignment to the next CTLx.DST_TR_WIDTH boundary.*
SRC_TR_WIDTH	[6:4]	rw	**Source Transfer Width** Table 5-12 lists the decoding for this field.
DST_TR_WIDTH	[3:1]	rw	**Destination Transfer Width** Table 5-12 lists the decoding for this field.
INT_EN	0	rw	**Interrupt Enable Bit** If set, then all interrupt-generating sources are enabled. Functions as a global mask bit for all interrupts for the channel; Raw* interrupt registers still assert if INT_EN = 0.

GPDMA0_CHx_CTLL (x=2-7)
Control Register Low for Channel x
(18_H + x*58_H) **Reset Value: 0030 4801$_H$**

GPDMA1_CHx_CTLL (x=0-3)
Control Register Low for Channel x
(18_H + x*58_H) **Reset Value: 0030 4801$_H$**



Field	Bits	Type	Description
0	[31:23]	r	**Reserved**
TT_FC	[22:20]	rw	**Transfer Type and Flow Control** The following transfer types are supported. • Memory to Memory • Memory to Peripheral • Peripheral to Memory • Peripheral to Peripheral Flow Control can be assigned to the GPDMA, the source peripheral, or the destination peripheral. **Table 5-13** lists the decoding for this field.
0	[19:17]	r	**Reserved**
SRC_MSIZE	[16:14]	rw	**Source Burst Transaction Length** Number of data items, each of width SRC_TR_WIDTH, to be read from the source every time a source burst transaction request is made from either the corresponding hardware or software handshaking interface. **Table 5-11** lists the decoding for this field. *Note: This value is not related to the AHB bus master HBURST bus.*

Field	Bits	Type	Description
DEST_MSIZE	[13:11]	rw	**Destination Burst Transaction Length** Number of data items, each of width DST_TR_WIDTH, to be written to the destination every time a destination burst transaction request is made from either the corresponding hardware or software handshaking interface. **Table 5-11** lists the decoding for this field. *Note: This value is not related to the AHB bus master HBURST bus.*
SINC	[10:9]	rw	**Source Address Increment** Indicates whether to increment or decrement the source address on every source transfer. If the device is fetching data from a source peripheral FIFO with a fixed address, then set this field to "No change". 00_B Increment 01_B Decrement $1x_B$ No change *Note: Incrementing or decrementing is done for alignment to the next CTLx.SRC_TR_WIDTH boundary.*
DINC	[8:7]	rw	**Destination Address Increment** Indicates whether to increment or decrement the destination address on every destination transfer. If your device is writing data to a destination peripheral FIFO with a fixed address, then set this field to "No change". 00_B Increment 01_B Decrement $1x_B$ No change *Note: Incrementing or decrementing is done for alignment to the next CTLx.DST_TR_WIDTH boundary.*
SRC_TR_WIDTH	[6:4]	rw	**Source Transfer Width** **Table 5-12** lists the decoding for this field.

Field	Bits	Type	Description
DST_TR_WIDTH	[3:1]	rw	**Destination Transfer Width** Table 5-12 lists the decoding for this field.
INT_EN	0	rw	**Interrupt Enable Bit** If set, then all interrupt-generating sources are enabled. Functions as a global mask bit for all interrupts for the channel; Raw* interrupt registers still assert if INT_EN = 0.

Table 5-11 CTLx.SRC_MSIZE and CTLx.DST_MSIZE Field Decoding

CTLx.SRC_MSIZE / CTLx.DEST_MSIZE	Number of data items to be transferred(of width CTLx.SRC_TR_WIDTH or CTLx.DST_TR_WIDTH)
000_B	1
001_B	4
010_B	8
others	reserved

Table 5-12 CTLx.SRC_TR_WIDTH and CTLx.DST_TR_WIDTH Field Decoding

CTLx.SRC_TR_WIDTH / CTLx.DST_TR_WIDTH	Size (bits)
000_B	8
001_B	16
010_B	32
others	reserved

Table 5-13 CTLx.TT_FC Field Decoding

CTLx.TT_FC Field	Transfer Type	Flow Controller
000_B	Memory to Memory	GPDMA
001_B	Memory to Peripheral	GPDMA
010_B	Peripheral to Memory	GPDMA
011_B	Peripheral to Peripheral	GPDMA
100_B	Peripheral to Memory	Peripheral
101_B	Peripheral to Peripheral	Source Peripheral

Table 5-13 CTLx.TT_FC Field Decoding (cont'd)

CTLx.TT_FC Field	Transfer Type	Flow Controller
110_B	Memory to Peripheral	Peripheral
111_B	Peripheral to Peripheral	Destination Peripheral

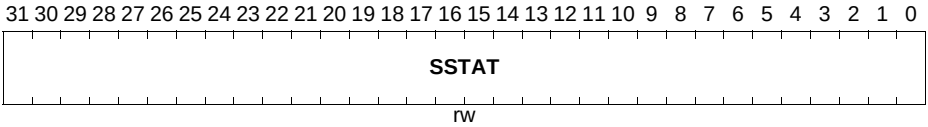
SSTAT

After each block transfer completes, hardware can retrieve the source status information from the address pointed to by the contents of the **SSTATAR** register. This status information is then stored in the SSTATx register and written out to the SSTATx register location of the LLI before the start of the next block.

Note: This register is a temporary placeholder for the source status information on its way to the SSTATx register location of the LLI. The source status information should be retrieved by software from the SSTATx register location of the LLI, and not by a read of this register over the GPDMA slave interface.

GPDMA0_CHx_SSTAT (x=0-1)
Source Status Register for Channel x

(20_H + x*58_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
SSTAT	[31:0]	rw	**Source Status** retrieved by hardware from the address pointed to by the contents of the **SSTATAR** register.

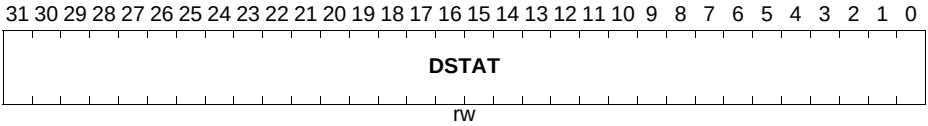
DSTAT

After the completion of each block transfer, hardware can retrieve the destination status information from the address pointed to by the contents of the **DSTATAR** register. This status information is then stored in the DSTATx register and written out to the DSTATx register location of the LLI before the start of the next block. This register does only exist for channels 0 and 1, for other channels the read-back value is always 0.

Note: This register is a temporary placeholder for the destination status information on its way to the DSTATx register location of the LLI. The destination status information should be retrieved by software from the DSTATx register location of the LLI and not by a read of this register over the GPDMA slave interface.

GPDMA0_CHx_DSTAT (x=0-1)
Destination Status Register for Channel x

(28_H + x*58_H) **Reset Value: 0000 0000_H**

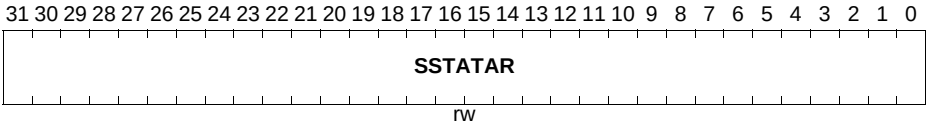


Field	Bits	Type	Description
DSTAT	[31:0]	rw	**Destination Status** retrieved by hardware from the address pointed to by the contents of the **DSTATAR** register.

SSTATAR

After the completion of each block transfer, hardware can retrieve the source status information from the address pointed to by the contents of the SSTATARx register.

GPDMA0_CHx_SSTATAR (x=0-1)
Source Status Address Register for Channel x
(30_H + x*58_H) **Reset Value: 0000 0000_H**

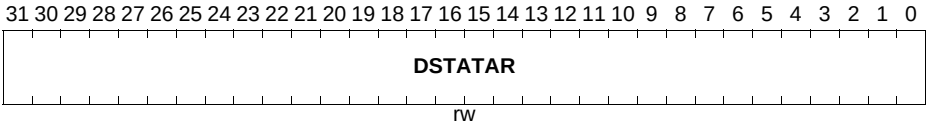


Field	Bits	Type	Description
SSTATAR	[31:0]	rw	**Source Status Address** Pointer from where hardware can fetch the source status information, which is registered in the **SSTAT** register and written out to the SSTATx register location of the LLI before the start of the next block.

DSTATAR

After the completion of each block transfer, hardware can retrieve the destination status information from the address pointed to by the contents of the DSTATARx register.

GPDMA0_CHx_DSTATAR (x=0-1)
Destination Status Address Register for Channel x
(38_H + x*58_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
DSTATAR	[31:0]	rw	**Destination Status Address** Pointer from where hardware can fetch the destination status information, which is registered in the **DSTAT** register and written out to the DSTATx register location of the LLI before the start of the next block.

CFG

These registers contain fields that configure the DMA transfer. The channel configuration register remains fixed for all blocks of a multi-block transfer.

Note: You need to program this register prior to enabling the channel.

GPDMA0_CHx_CFGH (x=0-1)
Configuration Register High for Channel x

(44_H + x*58_H) **Reset Value: 0000 0004_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	DEST_PER				SRC_PER				SS_UPD_EN	DS_UPD_EN	PROTCTL			FIFO_MODE	FCMODE
r	rw				rw				rw	rw	rw			rw	rw

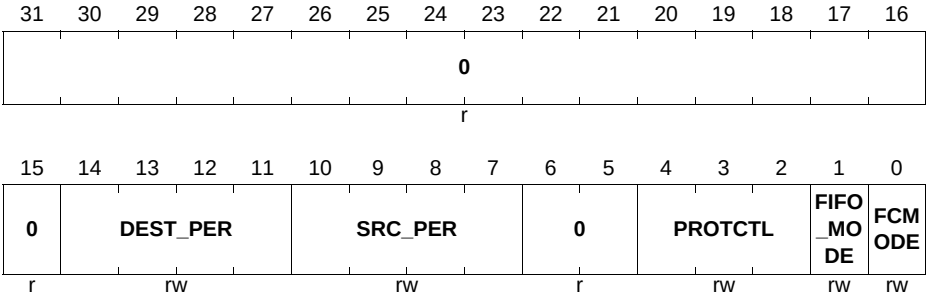
Field	Bits	Type	Description
0	[31:15]	r	**Reserved** Reset: 0_D
DEST_PER	[14:11]	rw	**Destination Peripheral** Assigns a DLR line as hardware handshaking interface to the destination of channel x if the CFGLx.HS_SEL_DST field is 0; otherwise, this field is ignored. The channel can then communicate with the destination peripheral connected to that interface through the assigned hardware handshaking interface. *Note: For correct DMA operation, only one peripheral (source or destination) should be assigned to the same handshaking interface.*

Field	Bits	Type	Description
SRC_PER	[10:7]	rw	**Source Peripheral** Assigns a DLR line as hardware handshaking interface to the source of channel x if the CFGLx.HS_SEL_SRC field is 0; otherwise, this field is ignored. The channel can then communicate with the source peripheral connected to that interface through the assigned hardware handshaking interface. *Note: For correct GPDMA operation, only one peripheral (source or destination) should be assigned to the same handshaking interface.*
SS_UPD_EN	6	rw	**Source Status Update Enable** Source status information is fetched only from the location pointed to by the **SSTATAR** register, stored in the **SSTAT** register and written out to the **SSTAT** location of the LLI if SS_UPD_EN is high.
DS_UPD_EN	5	rw	**Destination Status Update Enable** Destination status information is fetched only from the location pointed to by the **DSTATAR** register, stored in the **DSTAT** register and written out to the DSTATx location of the LLI if DS_UPD_EN is high.
PROTCTL	[4:2]	rw	**Protection Control** Used to drive the AHB HPROT[3:1] bus. The AMBA Specification recommends that the default value of HPROT indicates a non-cached, non-buffered, privileged data access. The reset value is used to indicate such an access. HPROT[0] is tied high because all transfers are data accesses, as there are no opcode fetches. There is a one-to-one mapping of these register bits to the HPROT[3:1] master interface signals. **Table 5-14** shows the mapping of bits in this field to the AHB HPROT[3:1] bus.

Field	Bits	Type	Description
FIFO_MODE	1	rw	**FIFO Mode Select** Determines how much space or data needs to be available in the FIFO before a burst transaction request is serviced. 0_B Space/data available for single AHB transfer of the specified transfer width. 1_B Data available is greater than or equal to half the FIFO depth for destination transfers and space available is greater than half the fifo depth for source transfers. The exceptions are at the end of a burst transaction request or at the end of a block transfer.
FCMODE	0	rw	**Flow Control Mode** Determines when source transaction requests are serviced when the Destination Peripheral is the flow controller. 0_B Source transaction requests are serviced when they occur. Data pre-fetching is enabled. 1_B Source transaction requests are not serviced until a destination transaction request occurs. In this mode, the amount of data transferred from the source is limited so that it is guaranteed to be transferred to the destination prior to block termination by the destination. Data pre-fetching is disabled.

GPDMA0_CHx_CFGH (x=2-7)
Configuration Register High for Channel x
(44_H + x*58_H) **Reset Value: 0000 0004_H**

GPDMA1_CHx_CFGH (x=0-3)
Configuration Register High for Channel x
(44_H + x*58_H) **Reset Value: 0000 0004_H**



Field	Bits	Type	Description
0	[31:15]	r	**Reserved**
DEST_PER	[14:11]	rw	**Destination Peripheral** Assigns a DLR line as hardware handshaking interface to the destination of channel x if the CFGLx.HS_SEL_DST field is 0; otherwise, this field is ignored. The channel can then communicate with the destination peripheral connected to that interface through the assigned hardware handshaking interface. *Note: For correct DMA operation, only one peripheral (source or destination) should be assigned to the same handshaking interface.*
SRC_PER	[10:7]	rw	**Source Peripheral** Assigns a DLR line as hardware handshaking interface to the source of channel x if the CFGLx.HS_SEL_SRC field is 0; otherwise, this field is ignored. The channel can then communicate with the source peripheral connected to that interface through the assigned hardware handshaking interface. *Note: For correct GPDMA operation, only one peripheral (source or destination) should be assigned to the same handshaking interface.*

Field	Bits	Type	Description
0	[6:5]	r	**Reserved**
PROTCTL	[4:2]	rw	**Protection Control** Used to drive the AHB HPROT[3:1] bus. The AMBA Specification recommends that the default value of HPROT indicates a non-cached, non-buffered, privileged data access. The reset value is used to indicate such an access. HPROT[0] is tied high because all transfers are data accesses, as there are no opcode fetches. There is a one-to-one mapping of these register bits to the HPROT[3:1] master interface signals. **Table 5-14** shows the mapping of bits in this field to the AHB HPROT[3:1] bus.
FIFO_MODE	1	rw	**FIFO Mode Select** Determines how much space or data needs to be available in the FIFO before a burst transaction request is serviced. 0_B Space/data available for single AHB transfer of the specified transfer width. 1_B Data available is greater than or equal to half the FIFO depth for destination transfers and space available is greater than half the fifo depth for source transfers. The exceptions are at the end of a burst transaction request or at the end of a block transfer.
FCMODE	0	rw	**Flow Control Mode** Determines when source transaction requests are serviced when the Destination Peripheral is the flow controller. 0_B Source transaction requests are serviced when they occur. Data pre-fetching is enabled. 1_B Source transaction requests are not serviced until a destination transaction request occurs. In this mode, the amount of data transferred from the source is limited so that it is guaranteed to be transferred to the destination prior to block termination by the destination. Data pre-fetching is disabled.

GPDMA0_CHx_CFGL (x=0-1)
Configuration Register Low for Channel x

(40_H + x*58_H) **Reset Value: 0000 0EX0$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RELOAD_DST	RELOAD_SRC	MAX_ABRST										SRC_HS_POL	DST_HS_POL	LOCK_B	LOCK_CH
rw	rw	rw										rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LOCK_B_L		LOCK_CH_L		HS_SEL_SRC	HS_SEL_DST	FIFO_EMPTY	CH_SUSP	CH_PRIOR			0				
rw		rw		rw	rw	r	rw	rw			r				

Field	Bits	Type	Description
RELOAD_DST	31	rw	**Automatic Destination Reload** The **DAR** register can be automatically reloaded from its initial value at the end of every block for multi-block transfers. A new block transfer is then initiated. For conditions under which this occurs, refer to **Table 5-8**.
RELOAD_SRC	30	rw	**Automatic Source Reload** The **SAR** register can be automatically reloaded from its initial value at the end of every block for multi-block transfers. A new block transfer is then initiated. For conditions under which this occurs, refer to **Table 5-8**.
MAX_ABRST	[29:20]	rw	**Maximum AMBA Burst Length** Maximum AMBA burst length that is used for DMA transfers on this channel. A value of 0 indicates that software is not limiting the maximum AMBA burst length for DMA transfers on this channel.
SRC_HS_POL	19	rw	**Source Handshaking Interface Polarity** 0_B Active high 1_B Active low For information on this, refer to **Section 5.2.3**.

Field	Bits	Type	Description
DST_HS_POL	18	rw	**Destination Handshaking Interface Polarity** 0_B Active high 1_B Active low For information on this, refer to **Section 5.2.3**.
LOCK_B	17	rw	**Bus Lock Bit** When active, the AHB bus master signal hlock is asserted for the duration specified in CFGLx.LOCK_B_L. For more information, refer to **Section 5.2.11.1**.
LOCK_CH	16	rw	**Channel Lock Bit** When the channel is granted control of the master bus interface and if the CFGLx.LOCK_CH bit is asserted, then no other channels are granted control of the master bus interface for the duration specified in CFGLx.LOCK_CH_L. Indicates to the master bus interface arbiter that this channel wants exclusive access to the master bus interface for the duration specified in CFGLx.LOCK_CH_L.
LOCK_B_L	[15:14]	rw	**Bus Lock Level** Indicates the duration over which CFGLx.LOCK_B bit applies. 00_B Over complete DMA transfer 01_B Over complete DMA block transfer $1x_B$ Over complete DMA transaction
LOCK_CH_L	[13:12]	rw	**Channel Lock Level** Indicates the duration over which CFGLx.LOCK_CH bit applies. 00_B Over complete DMA transfer 01_B Over complete DMA block transfer $1x_B$ Over complete DMA transaction

Field	Bits	Type	Description
HS_SEL_SRC	11	rw	**Source Software or Hardware Handshaking Select** This register selects which of the handshaking interfaces - hardware or software - is active for source requests on this channel. 0_B Hardware handshaking interface. Software-initiated transaction requests are ignored. 1_B Software handshaking interface. Hardware-initiated transaction requests are ignored. If the source peripheral is memory, then this bit is ignored.
HS_SEL_DST	10	rw	**Destination Software or Hardware Handshaking Select** This register selects which of the handshaking interfaces - hardware or software - is active for destination requests on this channel. 0_B Hardware handshaking interface. Software-initiated transaction requests are ignored. 1_B Software handshaking interface. Hardware-initiated transaction requests are ignored. If the destination peripheral is memory, then this bit is ignored.
FIFO_EMPTY	9	r	**Indicates if there is data left in the channel FIFO** Can be used in conjunction with CFGLx.CH_SUSP to cleanly disable a channel. 1_B Channel FIFO empty 0_B Channel FIFO not empty
CH_SUSP	8	rw	**Channel Suspend** Suspends all DMA data transfers from the source until this bit is cleared. There is no guarantee that the current transaction will complete. Can also be used in conjunction with CFGLx.FIFO_EMPTY to cleanly disable a channel without losing any data. 0_B Not suspended. 1_B Suspend DMA transfer from the source.

Field	Bits	Type	Description
CH_PRIOR	[7:5]	rw	**Channel priority** A priority of 7 is the highest priority, and 0 is the lowest. The value programmed to this field must be within 0 and 7. A programmed value outside this range will cause erroneous behavior. Reset: Channel Number For example: Chan0 = 000_B Chan1 = 001_B
0	[4:0]	r	**Reserved**

GPDMA0_CHx_CFGL (x=2-7)
Configuration Register Low for Channel x
(40_H + x*58_H) **Reset Value: 0000 0EX0$_H$**

GPDMA1_CHx_CFGL (x=0-3)
Configuration Register Low for Channel x
(40_H + x*58_H) **Reset Value: 0000 0EX0$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		MAX_ABRST										SRC_HS_POL	DST_HS_POL	LOCK_B	LOCK_CH
r		rw										rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LOCK_B_L		LOCK_CH_L		HS_SEL_SRC	HS_SEL_DST	FIFO_EMPTY	CH_SUSP	CH_PRIOR			0				
rw		rw		rw	rw	r	rw	rw			r				

Field	Bits	Type	Description
0	[31:30]	r	**Reserved**
MAX_ABRST	[29:20]	rw	**Maximum AMBA Burst Length** Maximum AMBA burst length that is used for DMA transfers on this channel. A value of 0 indicates that software is not limiting the maximum AMBA burst length for DMA transfers on this channel.

Field	Bits	Type	Description
SRC_HS_POL	19	rw	**Source Handshaking Interface Polarity** 0_B Active high 1_B Active low For information on this, refer to **Section 5.2.3**.
DST_HS_POL	18	rw	**Destination Handshaking Interface Polarity** 0_B Active high 1_B Active low For information on this, refer to **Section 5.2.3**.
LOCK_B	17	rw	**Bus Lock Bit** When active, the AHB bus master signal hlock is asserted for the duration specified in CFGLx.LOCK_B_L. For more information, refer to **Section 5.2.11.1**.
LOCK_CH	16	rw	**Channel Lock Bit** When the channel is granted control of the master bus interface and if the CFGLx.LOCK_CH bit is asserted, then no other channels are granted control of the master bus interface for the duration specified in CFGLx.LOCK_CH_L. Indicates to the master bus interface arbiter that this channel wants exclusive access to the master bus interface for the duration specified in CFGLx.LOCK_CH_L.
LOCK_B_L	[15:14]	rw	**Bus Lock Level** Indicates the duration over which CFGLx.LOCK_B bit applies. 00_B Over complete DMA transfer 01_B Over complete DMA block transfer $1x_B$ Over complete DMA transaction
LOCK_CH_L	[13:12]	rw	**Channel Lock Level** Indicates the duration over which CFGLx.LOCK_CH bit applies. 00_B Over complete DMA transfer 01_B Over complete DMA block transfer $1x_B$ Over complete DMA transaction

Field	Bits	Type	Description
HS_SEL_SRC	11	rw	**Source Software or Hardware Handshaking Select** This register selects which of the handshaking interfaces - hardware or software - is active for source requests on this channel. 0_B Hardware handshaking interface. Software-initiated transaction requests are ignored. 1_B Software handshaking interface. Hardware-initiated transaction requests are ignored. If the source peripheral is memory, then this bit is ignored.
HS_SEL_DST	10	rw	**Destination Software or Hardware Handshaking Select** This register selects which of the handshaking interfaces - hardware or software - is active for destination requests on this channel. 0_B Hardware handshaking interface. Software-initiated transaction requests are ignored. 1_B Software handshaking interface. Hardware-initiated transaction requests are ignored. If the destination peripheral is memory, then this bit is ignored.
FIFO_EMPTY	9	r	**Indicates if there is data left in the channel FIFO** Can be used in conjunction with CFGLx.CH_SUSP to cleanly disable a channel. 1_B Channel FIFO empty 0_B Channel FIFO not empty
CH_SUSP	8	rw	**Channel Suspend** Suspends all DMA data transfers from the source until this bit is cleared. There is no guarantee that the current transaction will complete. Can also be used in conjunction with CFGLx.FIFO_EMPTY to cleanly disable a channel without losing any data. 0_B Not suspended. 1_B Suspend DMA transfer from the source.

Field	Bits	Type	Description
CH_PRIOR	[7:5]	rw	**Channel priority** A priority of 7 is the highest priority, and 0 is the lowest. The value programmed to this field must be within 0 and 7. A programmed value outside this range will cause erroneous behavior. Reset: Channel Number For example: Chan0 = 000_B Chan1 = 001_B
0	[4:0]	r	**Reserved**

Table 5-14 PROTCTL field to HPROT Mapping

1_B	HPROT[0]
CFGHx.PROTCTL[1]	HPROT[1]
CFGHx.PROTCTL[2]	HPROT[2]
CFGHx.PROTCTL[3]	HPROT[3]

SGR

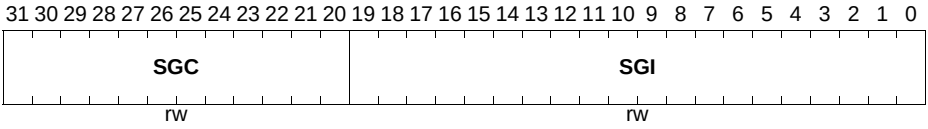
The Source Gather register contains two fields:

- Source gather count field (SGRx.SGC) - Specifies the number of contiguous source transfers of **CTL**.SRC_TR_WIDTH between successive gather intervals. This is defined as a gather boundary.
- Source gather interval field (SGRx.SGI) - Specifies the source address increment/decrement in multiples of **CTL**.SRC_TR_WIDTH on a gather boundary when gather mode is enabled for the source transfer.

The **CTL**.SINC field controls whether the address increments or decrements. When the **CTL**.SINC field indicates a fixed-address control, then the address remains constant throughout the transfer and the SGRx register is ignored. For more information, see **Section 5.2.13**.

GPDMA0_CHx_SGR (x=0-1)
Source Gather Register for Channel x

(48_H + x*58_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
SGC	[31:20]	rw	**Source gather count** Source contiguous transfer count between successive gather boundaries.
SGI	[19:0]	rw	**Source gather interval**

DSR

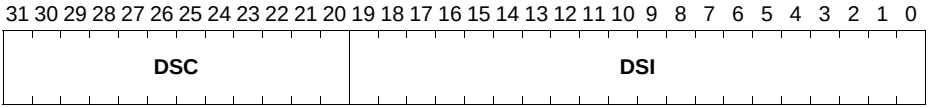
The Destination Scatter register contains two fields:

- Destination scatter count field (DSRx.DSC) - Specifies the number of contiguous destination transfers of **CTL**.DST_TR_WIDTH between successive scatter boundaries.
- Destination scatter interval field (DSRx.DSI) - Specifies the destination address increment/decrement in multiples of **CTL**.DST_TR_WIDTH on a scatter boundary when scatter mode is enabled for the destination transfer.

The **CTL**.DINC field controls whether the address increments or decrements. When the **CTL**.DINC field indicates a fixed address control, then the address remains constant throughout the transfer and the DSRx register is ignored. For more information, see **Section 5.2.13**.

GPDMA0_CHx_DSR (x=0-1)
Destination Scatter Register for Channel x

(50_H + x*58_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
DSC	[31:20]	rw	**Destination scatter count** Destination contiguous transfer count between successive scatter boundaries.
DSI	[19:0]	rw	**Destination scatter interval**

5.6.3 Interrupt Registers

The following sections describe the registers pertaining to interrupts, their status, and how to clear them. For each channel, there are five types of interrupt sources:

- IntBlock - Block Transfer Complete Interrupt. This interrupt is generated on DMA block transfer completion to the destination peripheral.
- IntDstTran - Destination Transaction Complete Interrupt
  This interrupt is generated after completion of the last AHB transfer of the requested single/burst transaction from the handshaking interface (either the hardware or software handshaking interface) on the destination side.

Note: If the destination for a channel is memory, then that channel will never generate the IntDstTran interrupt. Because of this, the corresponding bit in this field will not be set.

- IntErr - Error Interrupt
  This interrupt is generated when an ERROR response is received from an AHB slave on the HRESP bus during a DMA transfer. In addition, the DMA transfer is cancelled and the channel is disabled.
- IntSrcTran - Source Transaction Complete Interrupt
  This interrupt is generated after completion of the last AHB transfer of the requested single/burst transaction from the handshaking interface (either the hardware or software handshaking interface) on the source side.

Note: If the source or destination is memory, then IntSrcTran/IntDstTran interrupts should be ignored, as there is no concept of a "DMA transaction level" for memory.

- IntTfr - DMA Transfer Complete Interrupt
  This interrupt is generated on DMA transfer completion to the destination peripheral.

There are several groups of interrupt-related registers:

- **Interrupt Raw Status Registers**
- **Interrupt Status Registers**
- **Interrupt Mask Registers**
- **Interrupt Clear Registers**
- **Combined Interrupt Status Register**

When a channel has been enabled to generate interrupts, the following is true:

- Interrupt events are stored in the Raw Status registers.
- The contents of the Raw Status registers are masked with the contents of the Mask registers.
- The masked interrupts are stored in the Status registers.
- The contents of the Status registers are used to drive the int_* port signals.
- Writing to the appropriate bit in the Clear registers clears an interrupt in the Raw Status registers and the Status registers on the same clock cycle.

The contents of each of the five Status registers is ORed to produce a single bit for each interrupt type in the Combined Status register; that is, STATUSINT.

Note: For interrupts to propagate past the raw interrupt register stage, **CTL**.INT_EN must be set to 1_B, and the relevant interrupt must be unmasked in the mask* interrupt register.*

Interrupt Raw Status Registers

Interrupt events are stored in these Raw Interrupt Status registers before masking: RAWBLOCK, RawDstTran, RawErr, RawSrcTran, and RAWTFR. Each Raw Interrupt Status register has a bit allocated per channel; for example, RAWTFR[2] is the Channel 2 raw transfer complete interrupt.

Each bit in these registers is cleared by writing a 1 to the corresponding location in the CLEARTFR, CLEARBLOCK, CLEARSRCTRAN, CLEARDSTTRAN, CLEARERR registers.

Note: Write access is available to these registers for software testing purposes only. Under normal operation, writes to these registers are not recommended.

RAWTFR

Raw DMA Transfer Complete Interrupt Status.

RAWBLOCK

Raw Block Transfer Complete Interrupt Status.

RAWSRCTRAN

Raw Source Transaction Complete Interrupt Status.

RAWDSTTRAN

Raw Block Transfer Complete Interrupt Status.

RAWERR

Raw Error Interrupt Status.

GPDMA0_RAWTFR
Raw IntTfr Status **($2C0_H$)** **Reset Value: 0000 0000$_H$**
GPDMA0_RAWBLOCK
Raw IntBlock Status **($2C8_H$)** **Reset Value: 0000 0000$_H$**
GPDMA0_RAWSRCTRAN
Raw IntSrcTran Status **($2D0_H$)** **Reset Value: 0000 0000$_H$**
GPDMA0_RAWDSTTRAN
Raw IntBlock Status **($2D8_H$)** **Reset Value: 0000 0000$_H$**
GPDMA0_RAWERR
Raw IntErr Status **($2E0_H$)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0																								CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
r																								rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
0	[31:8]	r	**Reserved**
CHx (x=0-7)	x	rw	**Raw Interrupt Status for channel x**

GPDMA1_RAWTFR
Raw IntTfr Status **($2C0_H$)** **Reset Value: 0000 0000$_H$**
GPDMA1_RAWBLOCK
Raw IntBlock Status **($2C8_H$)** **Reset Value: 0000 0000$_H$**
GPDMA1_RAWSRCTRAN
Raw IntSrcTran Status **($2D0_H$)** **Reset Value: 0000 0000$_H$**
GPDMA1_RAWDSTTRAN
Raw IntBlock Status **($2D8_H$)** **Reset Value: 0000 0000$_H$**
GPDMA1_RAWERR
Raw IntErr Status **($2E0_H$)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0																												CH3	CH2	CH1	CH0
r																												rw	rw	rw	rw

Field	Bits	Type	Description
0	[31:4]	r	**Reserved**
CHx (x=0-3)	x	rw	**Raw Interrupt Status for channel x**

Interrupt Status Registers

All interrupt events from all channels are stored in these Interrupt Status registers after masking: STATUSBLOCK, STATUSDSTTRAN, STATUSERR, STATUSSRCTRAN, and STATUSTFR. Each Interrupt Status register has a bit allocated per channel; for example, STATUSTFR[2] is the Channel 2 status transfer complete interrupt. The contents of these registers are used to generate the interrupt signals (int or int_n bus, depending on interrupt polarity) leaving the GPDMA.

STATUSTFR

DMA Transfer Complete Interrupt Status.

STATUSBLOCK

Block Transfer Complete Interrupt Status.

STATUSSRCTRAN

Source Transaction Complete Interrupt Status.

STATUSDSTTRAN

Block Transfer Complete Interrupt Status.

STATUSERR

Error Interrupt Status.

GPDMA0_STATUSTFR
IntTfr Status **($2E8_H$)** **Reset Value: $0000\ 0000_H$**

GPDMA0_STATUSBLOCK
IntBlock Status **($2F0_H$)** **Reset Value: $0000\ 0000_H$**

GPDMA0_STATUSSRCTRAN
IntSrcTran Status **($2F8_H$)** **Reset Value: $0000\ 0000_H$**

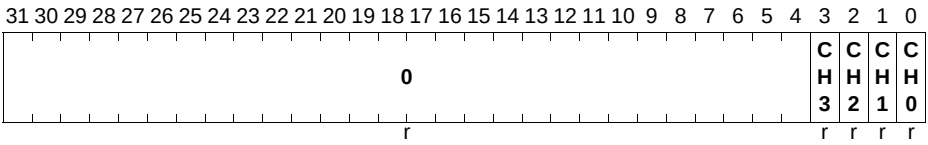
GPDMA0_STATUSDSTTRAN
IntBlock Status **(300_H)** **Reset Value: $0000\ 0000_H$**

GPDMA0_STATUSERR
IntErr Status **(308_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0																								CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
r																								r	r	r	r	r	r	r	r

Field	Bits	Type	Description
0	[31:8]	r	**Reserved**
CHx (x=0-7)	x	r	**Interrupt Status for channel x**

GPDMA1_STATUSTFR
IntTfr Status **($2E8_H$)** **Reset Value: 0000 0000$_H$**
GPDMA1_STATUSBLOCK
IntBlock Status **($2F0_H$)** **Reset Value: 0000 0000$_H$**
GPDMA1_STATUSSRCTRAN
IntSrcTran Status **($2F8_H$)** **Reset Value: 0000 0000$_H$**
GPDMA1_STATUSDSTTRAN
IntBlock Status **(300_H)** **Reset Value: 0000 0000$_H$**
GPDMA1_STATUSERR
IntErr Status **(308_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
0	[31:4]	r	**Reserved**
CHx (x=0-3)	x	r	**Interrupt Status for channel x**

Interrupt Mask Registers

The contents of the Raw Status registers are masked with the contents of the Mask registers: MASKBLOCK, MASKDSTTRAN, MASKERR, MASKSRCTRAN, and MASKTFR. Each Interrupt Mask register has a bit allocated per channel; for example, MASKTFR[2] is the mask bit for the Channel 2 transfer complete interrupt.

When the source peripheral of DMA channel i is memory, then the source transaction complete interrupt, MASKSRCTRAN[z], must be masked to prevent an erroneous triggering of an interrupt on the int_combined signal. Similarly, when the destination peripheral of DMA channel i is memory, then the destination transaction complete interrupt, MASKDSTTRAN[i], must be masked to prevent an erroneous triggering of an interrupt on the int_combined(_n) signal.

A channel INT_MASK bit will be written only if the corresponding mask write enable bit in the INT_MASK_WE field is asserted on the same AHB write transfer. This allows software to set a mask bit without performing a read-modified write operation. For example, writing hex 01x1 to the MASKTFR register writes a 1 into MASKTFR[0], while MASKTFR[7:1] remains unchanged. Writing hex 00xx leaves MASKTFR[7:0] unchanged.

Writing a 1 to any bit in these registers unmasks the corresponding interrupt, thus allowing the GPDMA to set the appropriate bit in the Status registers and int_* port signals.

MASKTFR

Mask for Raw DMA Transfer Complete Interrupt Status.

MASKBLOCK

Mask for Raw Block Transfer Complete Interrupt Status.

MASKSRCTRAN

Mask for Raw Source Transaction Complete Interrupt Status.

MASKDSTTRAN

Mask for Raw Block Transfer Complete Interrupt Status.

MASKERR

Mask for Raw Error Interrupt Status.

GPDMA0_MASKTFR
Mask for Raw IntTfr Status **(310_H)** **Reset Value: 0000 0000$_H$**

GPDMA0_MASKBLOCK
Mask for Raw IntBlock Status **(318_H)** **Reset Value: 0000 0000$_H$**

GPDMA0_MASKSRCTRAN
Mask for Raw IntSrcTran Status **(320_H)** **Reset Value: 0000 0000$_H$**

GPDMA0_MASKDSTTRAN
Mask for Raw IntBlock Status **(328_H)** **Reset Value: 0000 0000$_H$**

GPDMA0_MASKERR
Mask for Raw IntErr Status **(330_H)** **Reset Value: 0000 0000$_H$**

31:16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	WE_CH7	WE_CH6	WE_CH5	WE_CH4	WE_CH3	WE_CH2	WE_CH1	WE_CH0	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
r	w	w	w	w	w	w	w	w	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
0	[31:16]	r	**Reserved**
WE_CHx (x=0-7)	8+x	w	**Write enable for mask bit of channel x** 0_B write disabled 1_B write enabled
CHx (x=0-7)	x	rw	**Mask bit for channel x** 0_B masked 1_B unmasked

GPDMA1_MASKTFR
Mask for Raw IntTfr Status **(310_H)** **Reset Value: 0000 0000$_H$**

GPDMA1_MASKBLOCK
Mask for Raw IntBlock Status **(318_H)** **Reset Value: 0000 0000$_H$**

GPDMA1_MASKSRCTRAN
Mask for Raw IntSrcTran Status **(320_H)** **Reset Value: 0000 0000$_H$**

GPDMA1_MASKDSTTRAN
Mask for Raw IntBlock Status **(328_H)** **Reset Value: 0000 0000$_H$**

GPDMA1_MASKERR
Mask for Raw IntErr Status **(330_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0																				WE_CH3	WE_CH2	WE_CH1	WE_CH0	0				CH3	CH2	CH1	CH0
r																				w	w	w	w	r				rw	rw	rw	rw

Field	Bits	Type	Description
0	[31:12]	r	**Reserved**
WE_CHx (x=0-3)	8+ x	w	**Write enable for mask bit of channel x** 0_B write disabled 1_B write enabled
0	[7:4]	r	**Reserved**
CHx (x=0-3)	x	rw	**Mask bit for channel x** 0_B masked 1_B unmasked

Interrupt Clear Registers

Each bit in the Raw Status and Status registers is cleared on the same cycle by writing a 1 to the corresponding location in the Clear registers: CLEARBLOCK, CLEARDSTTRAN, CLEARERR, CLEARSRCTRAN, and CLEARTFR. Each Interrupt Clear register has a bit allocated per channel; for example, CLEARTFR[2] is the clear bit for the Channel 2 transfer complete interrupt. Writing a 0 has no effect. These registers are not readable.

CLEARTFR

Clear DMA Transfer Complete Interrupt Status and Raw Status.

CLEARBLOCK

Clear Block Transfer Complete Interrupt Status and Raw Status.

CLEARSRCTRAN

Clear Source Transaction Complete Interrupt Status and Raw Status.

CLEARDSTTRAN

Clear Block Transfer Complete Interrupt Status and Raw Status.

CLEARERR

Clear Error Interrupt Status and Raw Status.

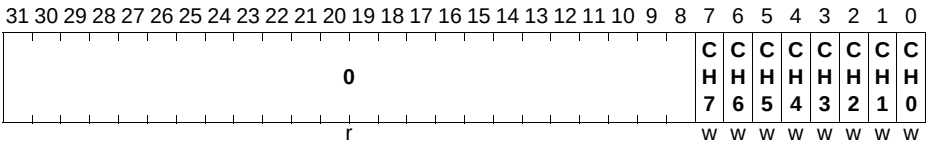
GPDMA0_CLEARTFR
IntTfr Status **(338_H)** **Reset Value: $0000\ 0000_H$**

GPDMA0_CLEARBLOCK
IntBlock Status **(340_H)** **Reset Value: $0000\ 0000_H$**

GPDMA0_CLEARSRCTRAN
IntSrcTran Status **(348_H)** **Reset Value: $0000\ 0000_H$**

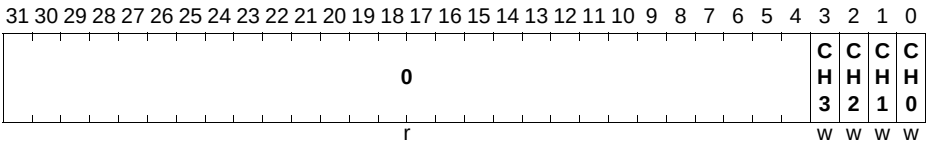
GPDMA0_CLEARDSTTRAN
IntBlock Status **(350_H)** **Reset Value: $0000\ 0000_H$**

GPDMA0_CLEARERR
IntErr Status **(358_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
0	[31:8]	r	**Reserved**
CHx (x=0-7)	x	w	**Clear Interrupt Status and Raw Status for channel x** 0_B no effect 1_B clear status

GPDMA1_CLEARTFR
IntTfr Status **(338_H)** **Reset Value: $0000\ 0000_H$**
GPDMA1_CLEARBLOCK
IntBlock Status **(340_H)** **Reset Value: $0000\ 0000_H$**
GPDMA1_CLEARSRCTRAN
IntSrcTran Status **(348_H)** **Reset Value: $0000\ 0000_H$**
GPDMA1_CLEARDSTTRAN
IntBlock Status **(350_H)** **Reset Value: $0000\ 0000_H$**
GPDMA1_CLEARERR
IntErr Status **(358_H)** **Reset Value: $0000\ 0000_H$**

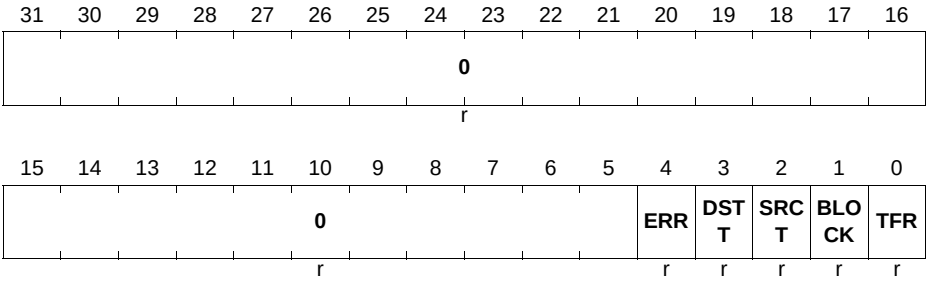


Field	Bits	Type	Description
0	[31:4]	r	**Reserved**
CHx (x=0-3)	x	w	**Clear Interrupt Status and Raw Status for channel x** 0_B no effect 1_B clear status

Combined Interrupt Status Register

The contents of each of the five Status registers - STATUSTFR, STATUSBLOCK, STATUSSRCTRAN, STATUSDSTTRAN, STATUSERR - is ORed to produce a single bit for each interrupt type in the Combined Status register (STATUSINT). This register is read-only.

GPDMA0_STATUSINT
Combined Interrupt Status Register (360_H) **Reset Value: $0000\ 0000_H$**
GPDMA1_STATUSINT
Combined Interrupt Status Register (360_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
0	[31:5]	r	**Reserved**
ERR	4	r	**OR of the contents of STATUSERR register**
DSTT	3	r	**OR of the contents of STATUSDSTTRAN register**
SRCT	2	r	**OR of the contents of STATUSSRCTRAN register**
BLOCK	1	r	**OR of the contents of STATUSBLOCK register**
TFR	0	r	**OR of the contents of STATUSTFR register**

5.6.4 Software Handshaking Registers

The registers that comprise the software handshaking registers allow software to initiate single or burst transaction requests in the same way that handshaking interface signals do in hardware.

Setting **CFG**.HS_SEL_SRC to 1 enables software handshaking on the source of channel x. Setting **CFG**.HS_SEL_DST to 1 enables software handshaking on the destination of channel x.

REQSRCREG

A bit is assigned for each channel in this register. REQSRCREG[n] is ignored when software handshaking is not enabled for the source of channel n.

A channel SRC_REQ bit is written only if the corresponding channel write enable bit in the SRC_REQ_WE field is asserted on the same AHB write transfer, and if the channel is enabled in the **CHENREG** register. For example, writing hex 0101 writes a 1 into REQSRCREG[0], while REQSRCREG[7:1] remains unchanged. Writing hex 00xx leaves REQSRCREG[7:0] unchanged. This allows software to set a bit in the REQSRCREG register without performing a read-modified write operation.

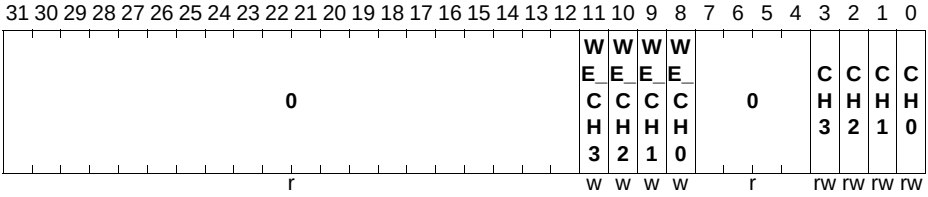
The functionality of this register depends on whether the source is a flow control peripheral or not. For a description of when the source is not a flow controller, refer to **Section 5.2.7.4**.

GPDMA0_REQSRCREG
Source Software Transaction Request Register
(368_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0																WE_CH7	WE_CH6	WE_CH5	WE_CH4	WE_CH3	WE_CH2	WE_CH1	WE_CH0	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
r																w	w	w	w	w	w	w	w	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
0	[31:16]	r	**Reserved**
WE_CHx (x=0-7)	8+x	w	**Source request write enable for channel x** 0_B write disabled 1_B write enabled
CHx (x=0-7)	x	rw	**Source request for channel x**

GPDMA1_REQSRCREG
Source Software Transaction Request Register
(368_H) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
0	[31:12]	r	**Reserved**
WE_CHx (x=0-3)	8+x	w	**Source request write enable for channel x** 0_B write disabled 1_B write enabled
0	[7:4]	r	**Reserved**
CHx (x=0-3)	x	rw	**Source request for channel x**

REQDSTREG

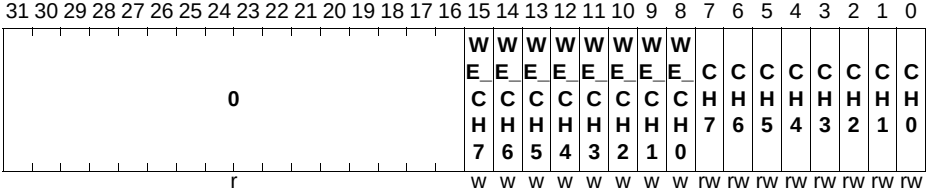
A bit is assigned for each channel in this register. REQDSTREG[n] is ignored when software handshaking is not enabled for the source of channel n.

A channel DST_REQ bit is written only if the corresponding channel write enable bit in the DST_REQ_WE field is asserted on the same AHB write transfer, and if the channel is enabled in the **CHENREG** register.

The functionality of this register depends on whether the destination is a flow control peripheral or not. For a description of when the destination is not a flow controller, refer to **Section 5.2.7.4**.

GPDMA0_REQDSTREG
Destination Software Transaction Request Register

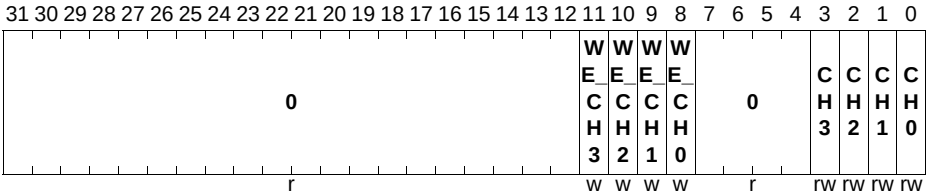
(370$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
0	[31:16]	r	**Reserved**
WE_CHx (x=0-7)	8+x	w	**Source request write enable for channel x** 0_B write disabled 1_B write enabled
CHx (x=0-7)	x	rw	**Source request for channel x**

GPDMA1_REQDSTREG
Destination Software Transaction Request Register

(370$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
0	[31:12]	r	**Reserved**
WE_CHx (x=0-3)	8+x	w	**Source request write enable for channel x** 0_B write disabled 1_B write enabled

Field	Bits	Type	Description
0	[7:4]	r	**Reserved**
CHx (x=0-3)	x	rw	**Source request for channel x**

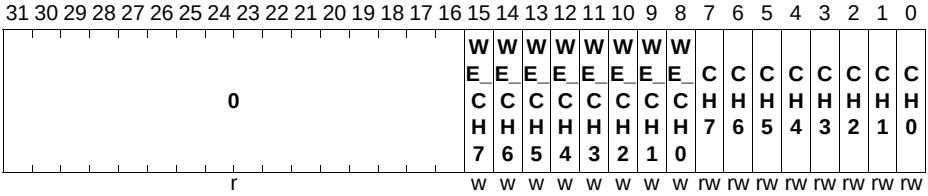
SGLREQSRCREG

A bit is assigned for each channel in this register. SGLREQSRCREG[n] is ignored when software handshaking is not enabled for the source of channel n.

A channel SRC_SGLREQ bit is written only if the corresponding channel write enable bit in the SRC_SGLREQ_WE field is asserted on the same AHB write transfer, and if the channel is enabled in the **CHENREG** register.

The functionality of this register depends on whether the source is a flow control peripheral or not. For a description of when the source is not a flow controller, refer to **Section 5.2.7.4**.

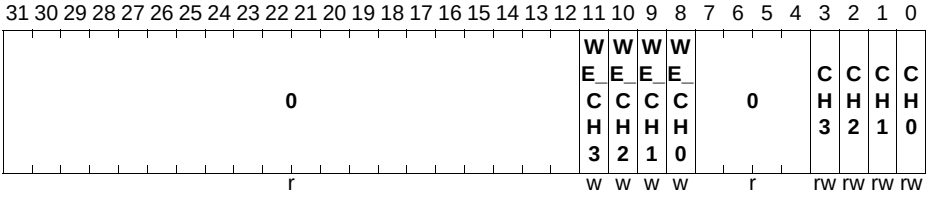
GPDMA0_SGLREQSRCREG
Single Source Transaction Request Register
(378_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
0	[31:16]	r	**Reserved**
WE_CHx (x=0-7)	8+x	w	**Source request write enable for channel x** 0_B write disabled 1_B write enabled
CHx (x=0-7)	x	rw	**Source request for channel x**

GPDMA1_SGLREQSRCREG
Single Source Transaction Request Register

(378$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
0	[31:12]	r	**Reserved**
WE_CHx (x=0-3)	8+x	w	**Source request write enable for channel x** 0_B write disabled 1_B write enabled
0	[7:4]	r	**Reserved**
CHx (x=0-3)	x	rw	**Source request for channel x**

SGLREQDSTREG

A bit is assigned for each channel in this register. SGLREQDSTREG[n] is ignored when software handshaking is not enabled for the destination of channel n.

A channel DST_SGLREQ bit is written only if the corresponding channel write enable bit in the DST_SGLREQ_WE field is asserted on the same AHB write transfer, and if the channel is enabled in the **CHENREG** register.

The functionality of this register depends on whether the destination is a flow control peripheral or not. For a description of when the destination is not a flow controller, refer to **Section 5.2.7.4**.

GPDMA0_SGLREQDSTREG
Single Destination Transaction Request Register

(380_H) **Reset Value: 0000 0000$_H$**

31:16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	WE_CH7	WE_CH6	WE_CH5	WE_CH4	WE_CH3	WE_CH2	WE_CH1	WE_CH0	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
r	w	w	w	w	w	w	w	w	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
0	[31:16]	r	**Reserved**
WE_CHx (x=0-7)	8+x	w	**Source request write enable for channel x** 0_B write disabled 1_B write enabled
CHx (x=0-7)	x	rw	**Source request for channel x**

GPDMA1_SGLREQDSTREG
Single Destination Transaction Request Register

(380_H) **Reset Value: 0000 0000$_H$**

31:12	11	10	9	8	7:4	3	2	1	0
0	WE_CH3	WE_CH2	WE_CH1	WE_CH0	0	CH3	CH2	CH1	CH0
r	w	w	w	w	r	rw	rw	rw	rw

Field	Bits	Type	Description
0	[31:12]	r	**Reserved**
WE_CHx (x=0-3)	8+x	w	**Source request write enable for channel x** 0_B write disabled 1_B write enabled

Field	Bits	Type	Description
0	[7:4]	r	**Reserved**
CHx (x=0-3)	x	rw	**Source request for channel x**

LSTSRCREG

A bit is assigned for each channel in this register. LSTSRCREG[n] is ignored when software handshaking is not enabled for the source of channel n, or when the source of channel n is not a flow controller.

A channel LSTSRC bit is written only if the corresponding channel write enable bit in the LSTSRC_WE field is asserted on the same AHB write transfer, and if the channel is enabled in the **CHENREG** register.

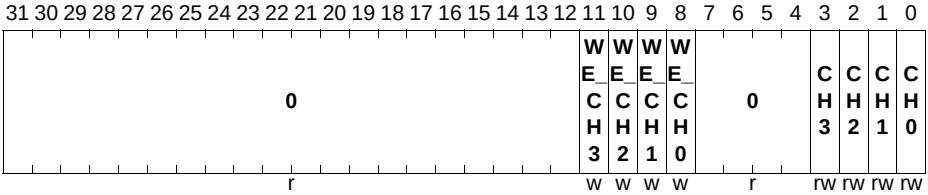
GPDMA0_LSTSRCREG
Last Source Transaction Request Register

(388_H) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0																WE_CH7	WE_CH6	WE_CH5	WE_CH4	WE_CH3	WE_CH2	WE_CH1	WE_CH0	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
r																w	w	w	w	w	w	w	w	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
0	[31:16]	r	**Reserved**
WE_CHx (x=0-7)	8+x	w	**Source last transaction request write enable for channel x** 0_B write disabled 1_B write enabled
CHx (x=0-7)	x	rw	**Source last request for channel x** 0_B Not last transaction in current block 1_B Last transaction in current block

GPDMA1_LSTSRCREG
Last Source Transaction Request Register
(388_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
0	[31:12]	r	**Reserved**
WE_CHx (x=0-3)	8+x	w	**Source last transaction request write enable for channel x** 0_B write disabled 1_B write enabled
0	[7:4]	r	**Reserved**
CHx (x=0-3)	x	rw	**Source last request for channel x** 0_B Not last transaction in current block 1_B Last transaction in current block

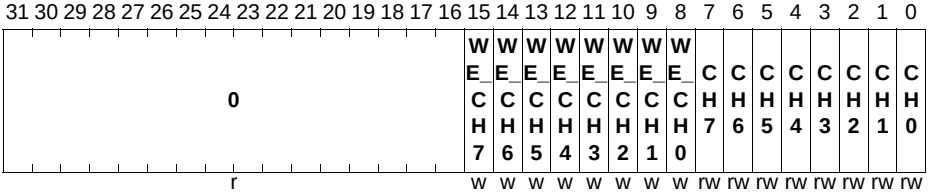
LSTDSTREG

A bit is assigned for each channel in this register. LSTDSTREG[n] is ignored when software handshaking is not enabled for the destination of channel n or when the destination of channel n is not a flow controller.

A channel LSTDST bit is written only if the corresponding channel write enable bit in the LSTDST_WE field is asserted on the same AHB write transfer, and if the channel is enabled in the **CHENREG** register.

GPDMA0_LSTDSTREG
Last Destination Transaction Request Register

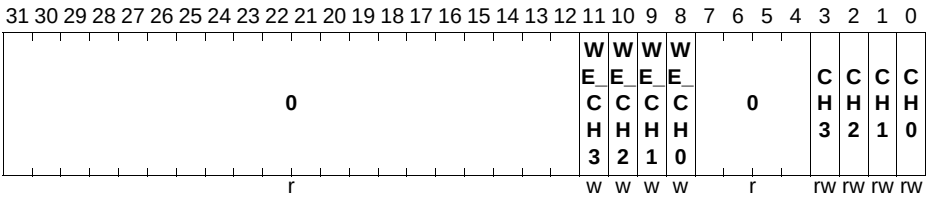
(390$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
0	[31:16]	r	**Reserved**
WE_CHx (x=0-7)	8+x	w	**Destination last transaction request write enable for channel x** 0_B write disabled 1_B write enabled
CHx (x=0-7)	x	rw	**Destination last request for channel x** 0_B Not last transaction in current block 1_B Last transaction in current block

GPDMA1_LSTDSTREG
Last Destination Transaction Request Register

(390$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
0	[31:12]	r	**Reserved**
WE_CHx (x=0-3)	8+x	w	**Destination last transaction request write enable for channel x** 0_B write disabled 1_B write enabled

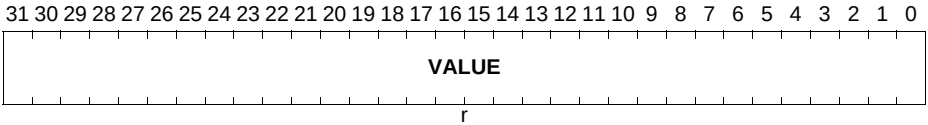
Field	Bits	Type	Description
0	[7:4]	r	**Reserved**
CHx (x=0-3)	x	rw	**Destination last request for channel x** 0_B Not last transaction in current block 1_B Last transaction in current block

5.6.5 Miscellaneous GPDMA Registers

ID

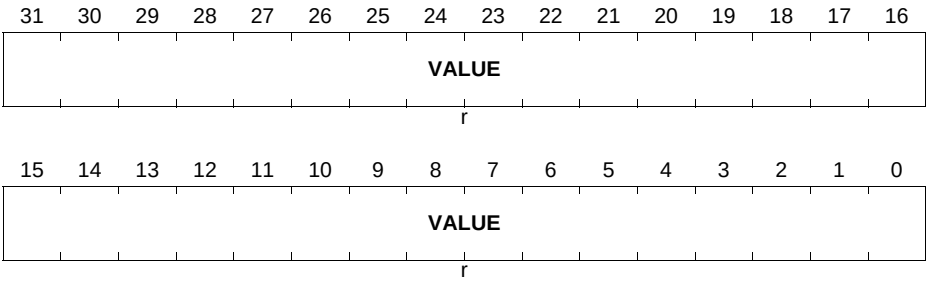
This is the GPDMA ID register, which is a read-only register that reads back the hardcoded module ID number.

GPDMA0_ID
GPDMA0 ID Register **($3A8_H$)** **Reset Value: $00AF\ C0XX_H$**



Field	Bits	Type	Description
VALUE	[31:0]	r	**Hardcoded GPDMA Peripheral ID**

GPDMA1_ID
GPDMA1 ID Register **($3A8_H$)** **Reset Value: $00B0\ C0XX_H$**

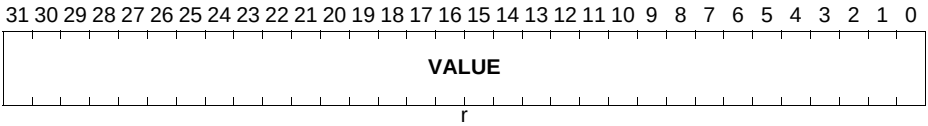


Field	Bits	Type	Description
VALUE	[31:0]	r	**Hardcoded GPDMA Peripheral ID**

TYPE

This is the GPDMA Component Type register, which is a read-only register that specifies the type of the packaged component.

GPDMA0_TYPE
GPDMA Component Type **($3F8_H$)** **Reset Value: $4457\ 1110_H$**
GPDMA1_TYPE
GPDMA Component Type **($3F8_H$)** **Reset Value: $4457\ 1110_H$**

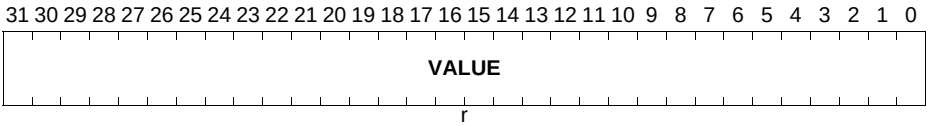


Field	Bits	Type	Description
VALUE	[31:0]	r	**Component Type** number = 44_57_11_10.

VERSION

This is the GPDMA Component Version register, which is a read-only register that specifies the version of the packaged component.

GPDMA0_VERSION
DMA Component Version **($3FC_H$)** **Reset Value: $3231\ 342A_H$**
GPDMA1_VERSION
DMA Component Version **($3FC_H$)** **Reset Value: $3231\ 342A_H$**



Field	Bits	Type	Description
VALUE	[31:0]	r	**Version number of the component**

6 Flexible CRC Engine (FCE)

The FCE provides a parallel implementation of Cyclic Redundancy Code (CRC) algorithms. The current FCE version for the XMC4500 microcontroller implements the IEEE 802.3 ethernet CRC32, the CCITT CRC16 and the SAE J1850 CRC8 polynomials. The primary target of FCE is to be used as an hardware acceleration engine for software applications or operating systems services using CRC signatures.

The FCE operates as a standard peripheral bus slave and is fully controlled through a set of configuration and control registers. The different CRC algorithms are independent from each other, they can be used concurrently by different software tasks.

*Note: The FCE kernel register names described in **"Registers" on Page 6-11** are referenced in a product Reference Manual by the module name prefix "FCE_".*

Input documents

[5] A painless guide to CRC Error Detection Algorithms, Ross N. Williams

[6] 32-Bit Cyclic Redundancy Codes for Internet Applications, Philip Koopman, International Conference on Dependable Systems and Networks (DSN), 2002

Related standards and norms

[7] IEEE 802.3 Ethernet 32-bits CRC

Table 6-1 FCE Abbreviations

CRC	Cyclic Redundancy Checksum
FCE	Flexible CRC Engine
IR	Input Register
RES	Result
STS	Status
CFG	Configuration

6.1 Overview

This chapter provides on overview of the features, applications and architecture of the FCE module.

6.1.1 Features

The FCE provides the following features:

- The FCE implements the following CRC polynomials:

  - CRC kernel 0 and 1: IEEE 802.3 CRC32 ethernet polynomial: 0x04C11DB7[1] - $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1$
  - CRC kernel 2: CCITT CRC16 polynomial: 0x1021 - $x^{16}+x^{12}+x^{5}+1$
  - CRC kernel 3: SAE J1850 CRC8 polynomial: 0x1D - $x^{8}+x^{4}+x^{3}+x^{2}+1$
- Parallel CRC implementation
  - Data blocks to be computed by FCE shall be a multiple of the polynomial degree
  - Start address of Data blocks to be computed by FCE shall be aligned to the polynomial degree
- Register Interface:
  - Input Register
  - CRC Register
  - Configuration Registers enabling to control the CRC operation and perform automatic checksum checks at the end of a message.
  - Extended register interface to control reliability of FCE execution in safety applications.
- Error notification scheme via dedicated interrupt node for:
  - Transient error detection: error interrupt generation (maskable) with local status register (cleared by software)
  - Checksum failure: error interrupt generation (maskable) with local status register (cleared by software)
- FCE provides one interrupt line to the interrupt system. Each CRC engine has its own set of flag registers.

6.1.2 Application Mapping

Among other applications, CRC algorithms are commonly used to calculate message signatures to:

- Check message integrity during transport over communication channels like internal buses or interfaces between microcontrollers
- Sign blocks of data residing in variable or invariable storage elements
- Compute signatures for program flow monitoring

One important property to be taken into account by the application when choosing a polynomial is the hamming distance: see **Chapter 6.9**.

6.1.3 Block Diagram

The FCE is a standard peripheral slave module which is controlled over a set of memory mapped registers. The FCE is fully synchronous with the CPU clock and runs with a 1:1 clock ratio.

1) The polynomial hexadecimal representation covers the coefficients (degree - 1) down to 0.

Depending on the hardware configuration the FCE may implement more CRC kernels with different CRC polynomials. The specific configuration for the XMC4500 microcontroller is shown in the **Figure 6-1 “FCE Block Diagram” on Page 6-3**.

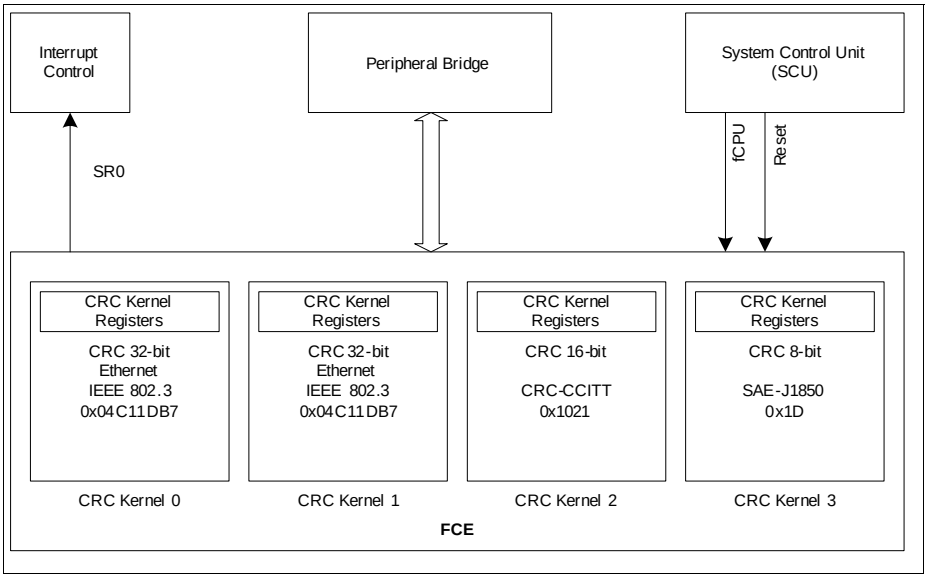


Figure 6-1 FCE Block Diagram

Every CRC kernel will present the same hardware and software architecture. The rest of this document will focus only on the description of the generic CRC kernel architecture.

In a multi-kernel implementation the interrupt lines are ored together, the FCE only presents a single interrupt node to the system. Each CRC kernel implements a status register that enables the software to identify which interrupt source is active. Please refer to the **STSm (m = 0-3)** register for a detailed description of the status and interrupt handling.

6.2 Functional Description

.

A checksum algorithm based on CRC polynomial division is characterized by the following properties:

1. polynomial degree (e.g. 32, that represents the highest power of two of the polynomial)
2. polynomial (e.g. 0x04C11DB7: the 33rd bit is omitted because always equal to 1)
3. init value: the initial value of the CRC register

4. input data reflected: indicates if each byte of the input parallel data is reflected before being used to compute the CRC
5. result data reflected: indicates if the final CRC value is reflected or not
6. XOR value: indicates if a final XOR operation is done before returning the CRC result

All the properties are fixed once a polynomial has been chosen. However the FCE provides the capability to control the two reflection steps and the final XOR through the CFG register. The reset values are compatible with the implemented algorithm. The final XOR control enables to select either 0xFFFFFFFF or 0x00000000 to be XORed with the POST_CRC1 value. These two values are those used by the most common CRC polynomials.

Note: The reflection steps and final XOR do not modify the properties of the CRC algorithm in terms of error detection, only the CRC final signature is affected.

The next two figures provides an overview of the control and status features of a CRC kernel.

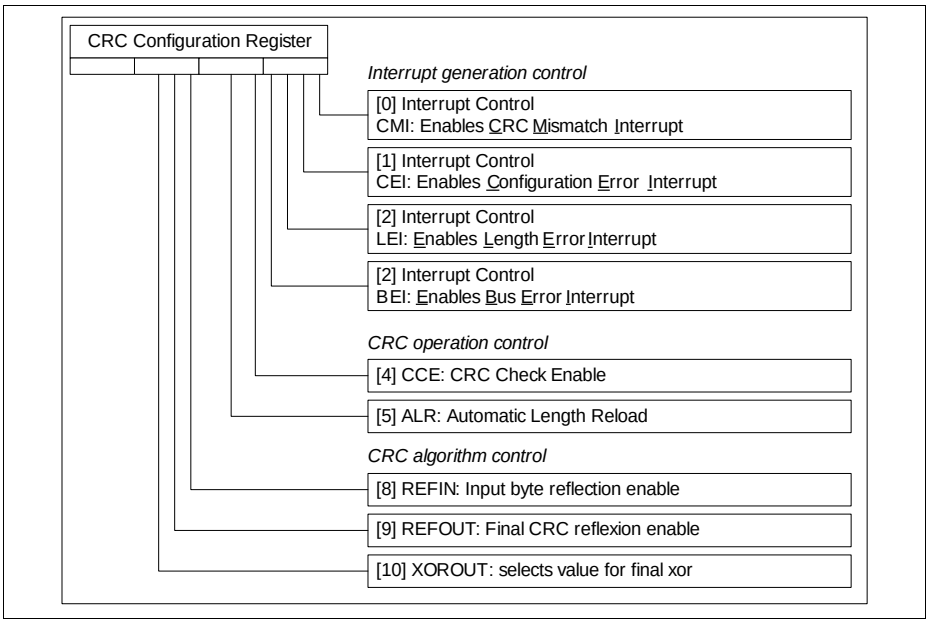


Figure 6-2 CRC kernel configuration register

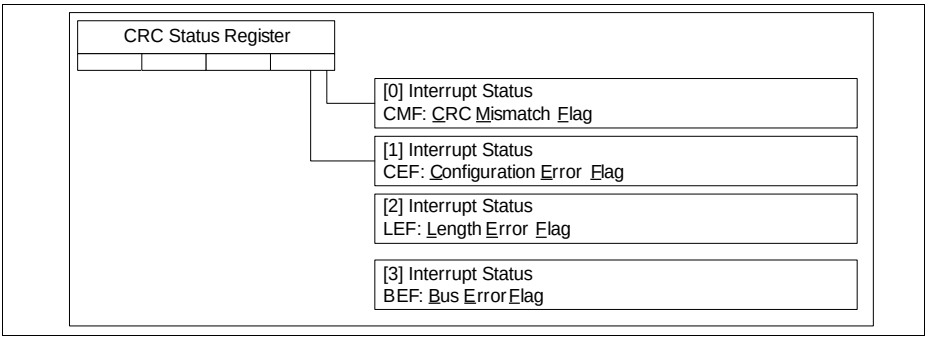


Figure 6-3 CRC kernel status register

6.2.1 Basic Operation

The software must first ensure that the CRC kernel is properly configured, especially the initial CRC value written via the **CRC** register. Then, it writes as many times as necessary into the **IR** register according to the length of the message. The resulting signature is stored in the CRC engine result register, **RESm**, which can be read by the software.

Depending on the CRC kernel accesses by software the following rules apply:

- When accessing a CRC kernel of degree <N> only the bits N-1 down to 0 are used by the CRC kernel. The upper bits are ignored on write. When reading from a CRC kernel register the non-used upper bits are set to 0.

6.2.2 Automatic Signature Check

The automatic signature check compares the signature at the end of a message with the expected signature configured in the CHECK register. In case of a mismatch, an event is generated (see **Section 6.3**. This feature is enabled by the CFG.CCE bit field (see **CFGm (m = 0-3)** register).

If the software whishes to use this feature, the LENGTH register and CHECK registers must be configured with respectively the length as number of words of the message and the expected signature (CHECK). The word length is defined by the degree of the polynomial used. The CHECK value takes into account the final CRC reflection and XOR operation.

When the **CFG.CCE** bit field is set, every time the IR register is written, the LENGH register is decremented by one until it reaches zero. The hardware monitors the transition of the LENGTH register from 1 to 0 to detect the end of the message and proceed with the comparison of the result register RESvalue with the CHECK register value. If the automatic length reload feature is enabled by the CFG.ALR bit field (see

CFGm (m = 0-3)), the LENGTH register is reinitialized with the previously configured value. This feature is especially suited when the FCE is used in combination with a DMA engine.

In the case the automatic length reload feature is not enabled, if LENGTH is already at zero but software still writes to IR (by mistake) every bit of the LENGTH should be set to 1 and hold this value until software initializes it again for the processing of a new message. In such case the STS.LEF (Length Error Flag) should be set and an interrupt generated if the CFG.LEI (Length Error Interrupt) is set.

Usually, the CRC signature of a message M0 is computed and appended to M0 to form the message M1 which is transmitted. One interesting property of CRCs is that the CRC signature of M1 shall be zero. This property is particularly useful when automatically checking the signature of data blocks of fixed length with the automatic length reload enabled. LENGTH should be loaded with the length of M1 and CHECK with 0.

6.2.3 Register protection and monitoring methods

Register Monitoring: applied to CFG and CHECK registers

Because CFG and CHECK registers are critical to the CRC operation, some mechanisms to detect and log transient errors are provided. Early detection of transient failures enables to improve the failure detection time and assess the severity of the failure. The monitoring mechanisms are implemented using two redundant instances as presented in **Figure 6-4**.

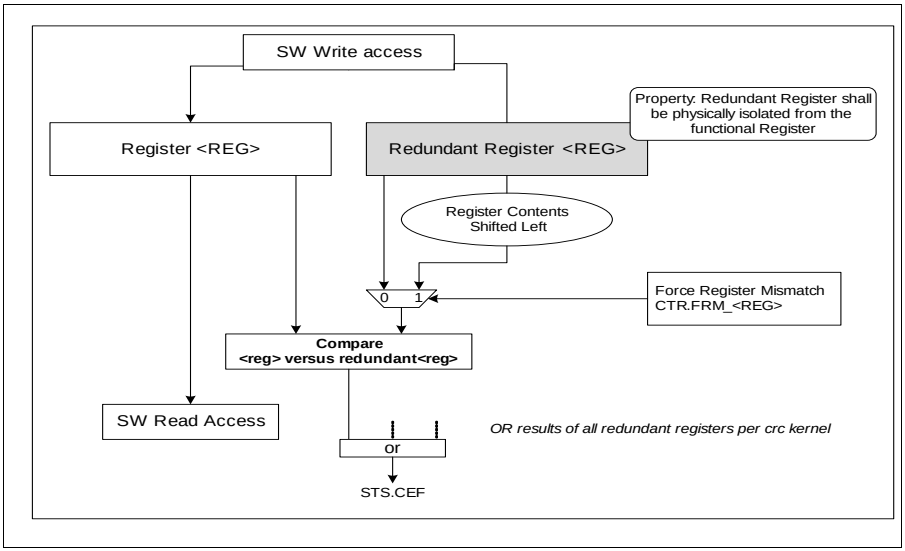


Figure 6-4 Register monitoring scheme

Let <REG> designate either CFG or CHECK registers. When a write to <REG> takes place the redundant register is also updated. **Redundant registers are not visible to software**. Bits of <REG> reserved have no storage and are not used for redundancy. A compare logic continuously compares the two stored values and provides a signal that indicates if the compare is successful or not. The result of all compare blocks are ored together to provide a single flag information. If a mismatch is detected the **STS.CEF** (Configuration Error Flag) bit is set. For run-time validation of the compare logic a Force Register Mismatch bit field (**CTR.FRM_<REG>**) is provided. When set to 1 by software the contents of the redundant register is shifted left by one bit position (redundant bit 0 position is always replaced by a logical 0 value) and is given to the compare logic instead of the redundant register value. This enables to check the compare logic is functional. Using a walking bit pattern, the software can completely check the full operation of the compare logic. Software needs to clear the **CTR.FRM_<REG>** bit to ‘0’ to be able to trigger again a new comparison error interrupt.

Register Access Protection: applies to LENGTH and CHECK registers

In order to reduce the probability of a mis-configuration of the CHECK and LENGTH registers (in the case the automatic check is used), the write access to the CHECK and LENGTH registers must follow a procedure:

Let **<REG>** designate **CHECK** or **LENGTH** registers. Before being able to configure a new <value> value into the <REG> register of a CRC kernel, software must first write the

0xFACECAFE value to the <REG> address. The 0xFACECAFE is not written into the <REG> register. The next write access will proceed as a normal bus write access. The write accesses shall use full 32-bit access only. This procedure will then be repeated every time software wants to configure a new <REG> value. If software reads the CHECK register just after writing 0xFACECAFE it returns the current <REG> contents and not 0xFACECAFE. **A read access to <REG> has no effect on the protection mechanism**.

The following C-code shows write accesses to the CHECK and LENGTH registers following this procedure:

```
//set CHECK register
FCE_CHECK0.U = 0xFACECAFE;
FCE_CHECK0.U = 0;

//set LENGTH register
FCE_LENGTH0.U = 0xFACECAFE;
FCE_LENGTH0.U = 256;
```

6.3 Service Request Generation

Each FCE CRC kernel provides one internal interrupt source. The interrupt lines from each CRC kernel are ored together to be sent to the interrupt system. The system interrupt is an active high pulse with the duration of one cycle (of the peripheral clock). The FCE interrupt handler can use the status information located within the **STS** status register of each CRC kernel.

Each CRC kernel provides the following interrupt sources:

- CRC Mismatch Interrupt controlled by **CFG.CMI** bit field and observable via the status bit field **STS.CMF** (CRC Mismatch Flag).
- Configuration Error Interrupt controlled by **CFG.CEI** bit field and observable via the status bit field **STS.CEF** (Configuration Error Flag).
- Length Error Interrupt controlled by **CFG.LEI** bit field and observable via the status bit field **STS.LEF** (Length Error Flag).
- Bus Error Interrupt controlled by **CFG.BEI** bit field and observable via the status bit field **STS.BEF** (Bus Error Flag).

Interrupt generation rules

- A status flag shall be cleared by software by writing a **1** to the corresponding bit position.
- If an status flag is set and a new hardware condition occurs, no new interrupt is generated by the kernel: the STS.<FLAG> bit field masks the generation of a new

interrupt from the same source. If a SW access to clear the interrupt status bit takes place and in the same cycle the hardware wants to set the bit, the hardware condition wins the arbitration.

As all the interrupts are caused by an error condition, the interrupt shall be handled by a Error Management software layer. The software services using the FCE as acceleration engine may not directly deal with error conditions but let the upper layer using the service to deal with the error handling.

6.4 Debug Behavior

The FCE has no specific debug feature.

6.5 Power, Reset and Clock

The FCE is inside the power core domain, therefore no special considerations about power up or power down sequences need to be taken. For an explanation about the different power domains, please address the SCU (System Control Unit) chapter.

A power down mode can be achieved by disabling the module using the **Clock Control Register (CLC)**.

The FCE module has one reset source. This reset source is handled at system level and it can be generated independently via a system control register (address SCU chapter for full description).

After release, the complete IP is set to default configuration. The default configuration for each register field is addressed on **Section 6.7**.

The FCE uses the CPU clock, fCPU (address SCU chapter for more details on clocking).

6.6 Initialization and System Dependencies

The FCE may have dependencies regarding the bus clock frequency. This dependencies should be addressed in the SCU and System Architecture Chapters.

Initialization:

The FCE is enabled by writing 0x0 to the CLC register. Software must first ensure that the CRC kernel is properly configured, especially the initial CRC register value written via the CRC register, the input and result reflection as well as the final xored value via the CFG register. The following source code is an example of initialization for the basic operation of the FCE kernel 0:

```
//enable FCE
FCE_CLC.U = 0x0;
//final result to be xored with 0xFFFFFFFF, no reflection
```

```
FCE_CFG0.U = 0x400;
//set CRC initial value (seed)
FCE_CRC0.U = 0xFFFFFFFF;
```

6.7 Registers

Table 6-3 show all registers associated with a FCE CRC-kernel. All FCE kernel register names are described in this section. They should get the prefix “FCE_” when used in the context of a product specification.

The registers are numbered by one index to indicate the related FCE CRC Kernel (m = 0-3). Some kernel registers are adapted to the degree of the polynomial implemented by the kernel.

Table 6-2 Registers Address Space - FCE Module

Module	Base Address	End Address	Note
FCE	$5002\ 0000_H$	$5002\ 3FFF_H$	

Table 6-3 Registers Overview - CRC Kernel Registers

Short Name	Description	Offset Addr.[1)]	Access Mode		Reset Class	Description See
			Read	Write		
System Registers						
CLC	Clock Control Register	00_H	U, PV	SV,E	3	Page 6-12
ID	Module Identification Register	08_H	U, PV	BE	3	Page 6-12
Generic CRC Engine Registers						
IRm	Input Register m	$20_H + m*20_H$	U, PV	U, PV	3	Page 6-14
RESm	CRC Result Register m	$24_H + m*20_H$	U, PV	BE	3	Page 6-15
CFGm	CRC Configuration Register m	$28_H + m*20_H$	U, PV	PV	3	Page 6-17
STSm	CRC Status Register m	$2C_H + m*20_H$	U, PV	U, PV	3	Page 6-19
LENGTH m	CRC Length Register m	$30_H + m*20_H$	U, PV	U, PV	3	Page 6-20
CHECKm	CRC Check Register m	$34_H + m*20_H$	U, PV	U, PV	3	Page 6-20
CRCm	CRC Register m	$38_H + m*20_H$	U, PV	U, PV	3	Page 6-22

Table 6-3 Registers Overview - CRC Kernel Registers (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Reset Class	Description See
			Read	Write		
CTRm	CRC Test Register m	$3C_H + m*20_H$	U, PV	U, PV	3	Page 6-23
CTRm	CRC Test Register m	$3C_H + m*20_H$	U, PV	U, PV	3	Page 6-23

1) The absolute register byte address for each CRC kernel m is calculated as follows:
CRC kernel register base Address (Table 6-2) + m*20H, m = 0-3

Disabling the FCE

The FCE module can be disabled using the **CLC** register.

When the disable state is requested all pending transactions running on the bus slave interface must be completed before the disabled state is entered. The CLC Register Module Disable Bit Status CLC.DISS indicates whether the module is currently disabled (DISS == 1). Any attempt to write any register with the exception of the CLC Register will generate a bus error. A read operation is allowed and does not generate a bus error.

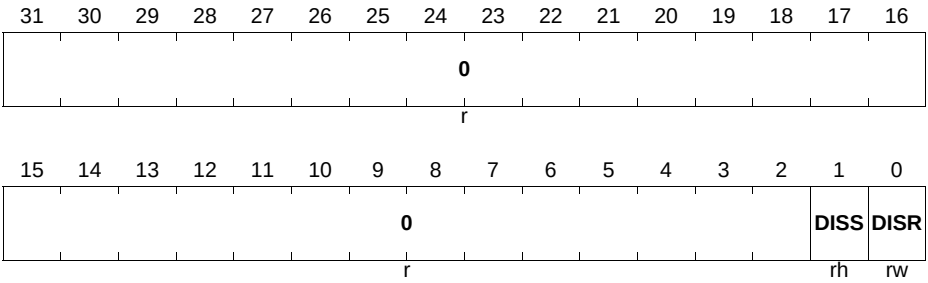
6.7.1 System Registers description

This section describes the registers related to the product system architecture.

Clock Control Register (CLC)

The Clock Control Register allows the programmer to adapt the functionality and power consumption of the module to the requirements of the application.

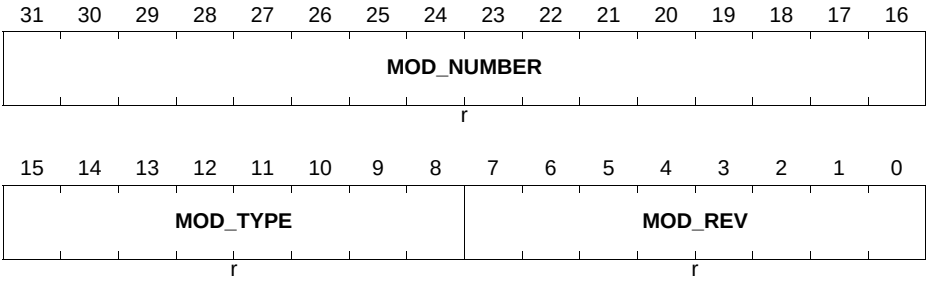
CLC
Clock Control Register **(00_H)** **Reset Value: $0000\ 0003_H$**



Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module.
DISS	1	rh	**Module Disable Status Bit** Bit indicates the current status of the module.
0	[31:2]	r	**Reserved** Read as 0; should be written with 0.

Module Identification Register

ID
Module Identification Register **(08_H)** **Reset Value: $00CA\ C001_H$**

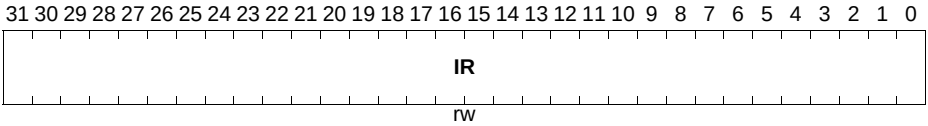


Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** This bit field defines the module revision number. The value of a module revision starts with 01_H (first revision). The current revision number is 01_H
MOD_TYPE	[15:8]	r	**Module Type** The bit field is set to $C0_H$ which defines the module as a 32-bit module.
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines a module identification number. The value for the FCE module is $00CA_H$.

6.7.2 CRC Kernel Control/Status Registers

CRC Engine Input Register

IRm (m = 0-1)
Input Register m **(20_H + m*20_H)** **Reset Value: 0000 0000$_H$**

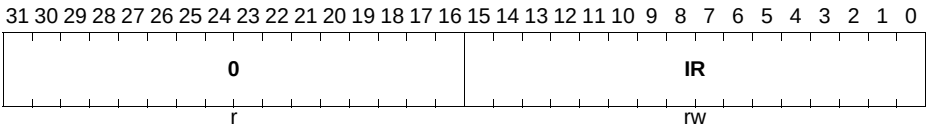


Field	Bits	Type	Description
IR	[7:0]	r	**Input Register** This bit field holds the 32-bit data to be computed

A write to IRm triggers the CRC kernel to update the message checksum according to the IR contents and to the current CRC register contents. Only 32-bit write transactions are allowed to this IRm registers, any other bus write transaction will lead to a Bus Error.

CRC Engine Input Register

IRm (m = 2-2)
Input Register m **(20_H + m*20_H)** **Reset Value: 0000 0000$_H$**



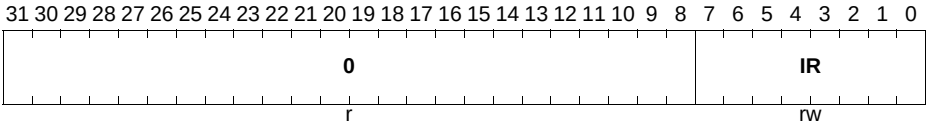
Field	Bits	Type	Description
IR	[15:0]	r	**Input Register** This bit field holds the 16-bit data to be computed
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

A write to IRm triggers the CRC kernel to update the message checksum according to the IR contents and to the current CRC register contents. Only 32-bit or 16-bit write

transactions are allowed to this IRm register, any other bus write transaction will lead to a Bus Error. Only the lower 16-bit of the write transactions will be used.

CRC Engine Input Register

IRm (m = 3-3)
Input Register m **(20_H + m*20_H)** **Reset Value: 0000 0000$_H$**

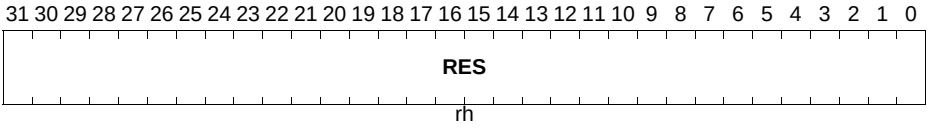


Field	Bits	Type	Description
IR	[7:0]	rw	**Input Register** This bit field holds the 8-bit data to be computed
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

A write to IRm triggers the CRC kernel to update the message checksum according to the IR contents and to the current CRC register contents. Any write transaction is allowed to this IRm register. Only the lower 8-bit of the write transactions will be used.

CRC Engine Result Register

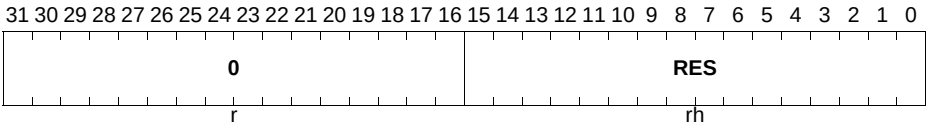
RESm (m = 0-1)
CRC Result Register m **(24_H + m*20_H)** **Reset Value: FFFF FFFF$_H$**



Field	Bits	Type	Description
RES	[31:0]	rh	**Result Register** Returns the final CRC value including CRC reflection and final XOR according to the CFG register configuration. Writing to this register has no effect.

CRC Engine Result Register

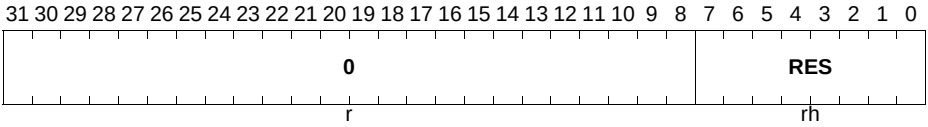
RESm (m = 2-2)
CRC Result Register m **(24_H + m*20_H)** **Reset Value: 0000 FFFF$_H$**



Field	Bits	Type	Description
RES	[15:0]	rh	**Result Register** Returns the final CRC value including CRC reflection and final XOR according to the CFG register configuration. Writing to this register has no effect.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

CRC Engine Result Register

RESm (m = 3-3)
CRC Result Register m **(24_H + m*20_H)** **Reset Value: 0000 00FF$_H$**



Field	Bits	Type	Description
RES	[7:0]	rh	**Result Register** Returns the final CRC value including CRC reflection and final XOR according to the CFG register configuration. Writing to this register has no effect.
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

CRC Engine Configuration Register

CFGm (m = 0-3)
CRC Configuration Register m

(28_H + m*20_H) **Reset Value: 0000 0700_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

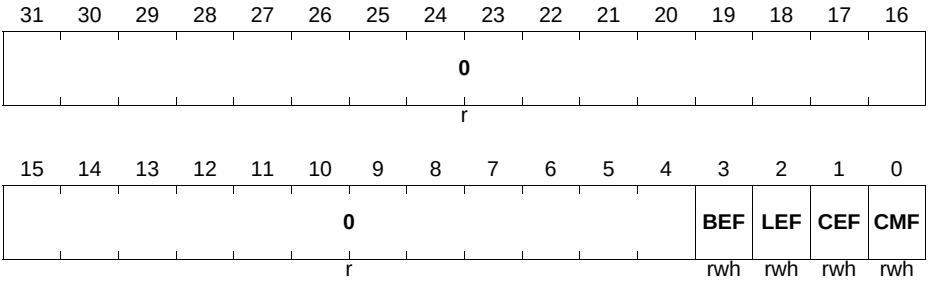
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					XSEL	REFOUT	REFIN	0		ALR	CCE	BEI	LEI	CEI	CMI
r					rw	rw	rw	r		rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
CMI	0	rw	**CRC Mismatch Interrupt** 0_B CRC Mismatch Interrupt is disabled 1_B CRC Mismatch Interrupt is enabled
CEI	1	rw	**Configuration Error Interrupt** When enabled, a Configuration Error Interrupt is generated whenever a mismatch is detected in the CFG and CHECK redundant registers. 0_B Configuration Error Interrupt is disabled 1_B Configuration Error Interrupt is enabled
LEI	2	rw	**Length Error Interrupt** When enabled, a Length Error Interrupt is generated if software writes to IR register with LENGTH equal to 0 and CFG.CCE is set to 1. 0_B Length Error Interrupt is disabled 1_B Length Error Interrupt is enabled
BEI	3	rw	**Bus Error Interrupt** When enabled, an interrupt is generated if a bus write transaction with an access width smaller than the kernel width is issued to the input register. 0_B Bus Error Interrupt is disabled 1_B Bus Error Interrupt is enabled

Field	Bits	Type	Description
CCE	4	rw	**CRC Check Comparison** 0_B CRC check comparison at the end of a message is disabled 1_B CRC check comparison at the end of a message is enabled
ALR	5	rw	**Automatic Length Reload** 0_B Disables automatic reload of the LENGTH field. 1_B Enables automatic reload of the LENGTH field at the end of a message.
REFIN	8	rw	**IR Byte Wise Reflection** 0_B IR Byte Wise Reflection is disabled 1_B IR Byte Wise Reflection is enabled
REFOUT	9	rw	**CRC 32-Bit Wise Reflection** 0_B CRC 32-bit wise is disabled 1_B CRC 32-bit wise is enabled
XSEL	10	rw	**Selects the value to be xored with the final CRC** 0_B 0x00000000 1_B 0xFFFFFFFF
0	[7:6], [31:11]	r	**Reserved** Read as 0; should be written with 0.

CRC Engine Status Register

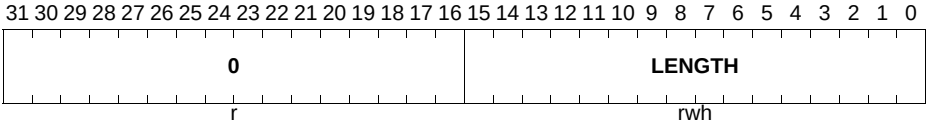
STSm (m = 0-3)
CRC Status Register m **($2C_H$ + m*20_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
CMF	0	rwh	**CRC Mismatch Flag** This bit is set per hardware only. To clear this bit, software must write a 1 to this bit field location. Writing 0 per software has no effect.
CEF	1	rwh	**Configuration Error Flag** This bit is set per hardware only. To clear this bit, software must write a 1 to this bit field location. Writing 0 per software has no effect.
LEF	2	rwh	**Length Error Flag** This bit is set per hardware only. To clear this bit, software must write a 1 to this bit field location. Writing 0 per software has no effect.
BEF	3	rwh	**Bus Error Flag** This bit is set per hardware only. To clear this bit, software must write a 1 to this bit field location. Writing 0 per software has no effect.
0	[31:4]	r	**Reserved** Read as 0; should be written with 0.

CRC Engine Length Register

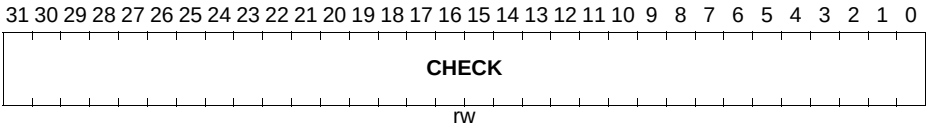
LENGTHm (m = 0-3)
CRC Length Register m **(30_H + m*20_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
LENGTH	[15:0]	rwh	**Message Length Register** Number of words building the message over which the CRC checksum is calculated. This bit field is modified by the hardware: every write to the IR register decrements the value of the LENGTH bit field. If the CFG.ALR field is set to 1, the LENGTH field shall be reloaded with its configuration value at the end of the cycle where LENGTH reaches 0.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

CRC Engine Check Register

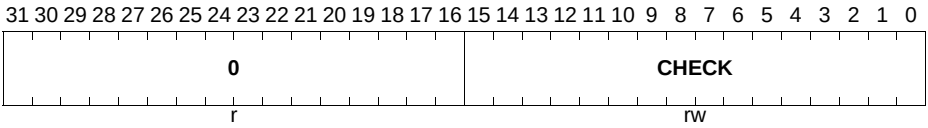
CHECKm (m = 0-1)
CRC Check Register m **(34_H + m*20_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
CHECK	[31:0]	rw	**CHECK Register** Expected CRC value to be checked by the hardware upon detection of a 1 to 0 transition of the LENGTH register. The comparison is enabled by the CFG.CCE bit field

CRC Engine Check Register

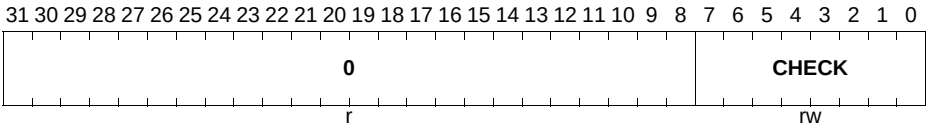
CHECKm (m = 2-2)
CRC Check Register m **(34_H + m*20_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
CHECK	[15:0]	rw	**CHECK Register** Expected CRC value to be checked by the hardware upon detection of a 1 to 0 transition of the LENGTH register. The comparison is enabled by the CFG.CCE bit field
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

CRC Engine Check Register

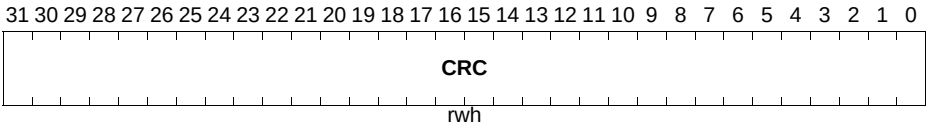
CHECKm (m = 3-3)
CRC Check Register m **(34_H + m*20_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
CHECK	[7:0]	rw	**CHECK Register** Expected CRC value to be checked by the hardware upon detection of a 1 to 0 transition of the LENGTH register. The comparison is enabled by the CFG.CCE bit field
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

CRC Engine Initialization Register

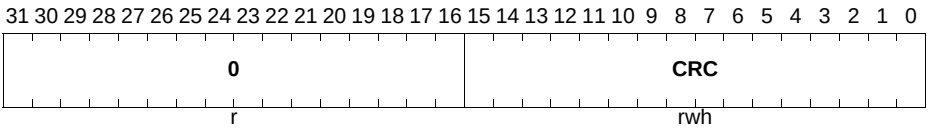
CRCm (m = 0-1)
CRC Register m **(38_H + m*20_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
CRC	[31:0]	rwh	**CRC Register** This register enables to directly access the internal CRC register

CRC Engine Initialization Register

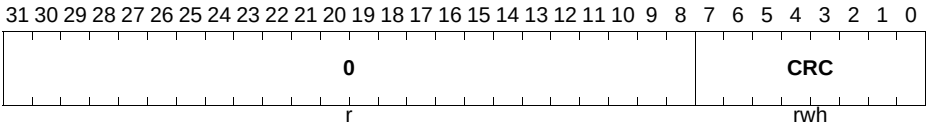
CRCm (m = 2-2)
CRC Register m **(38_H + m*20_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
CRC	[15:0]	rwh	**CRC Register** This register enables to directly access the internal CRC register
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

CRC Engine Initialization Register

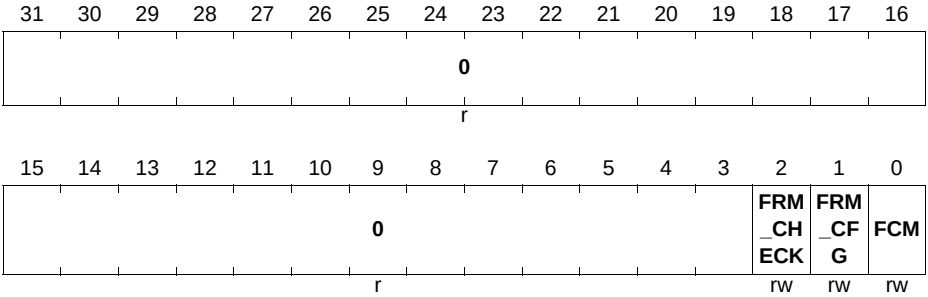
CRCm (m = 3-3)
CRC Register m **(38_H + m*20_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
CRC	[7:0]	rwh	**CRC Register** This register enables to directly access the internal CRC register
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

CRC Test Register

CTRm (m = 0-3)
CRC Test Register m **($3C_H$ + m*20_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
FCM	0	rw	**Force CRC Mismatch** Forces the CRC compare logic to issue an error regardless of the CHECK and CRC values. The hardware detects a 0 to 1 transition of this bit field and triggers a CRC Mismatch interrupt

Field	Bits	Type	Description
FRM_CFG	1	rw	**Force CFG Register Mismatch** This field is used to control the error injection mechanism used to check the compare logic of the redundant CFG registers. This is a one shot operation. When the hardware detects a 0 to 1 transition of this bit field it triggers a Configuration Mismatch interrupt (if enabled by the corresponding CFGm register).
FRM_CHECK	2	rw	**Force Check Register Mismatch** This field is used to control the error injection mechanism used to check the compare logic of the redundant CHECK registers. This is a one shot operation. The hardware detects a 0 to 1 transition of this bit field and triggers a Check Register Mismatch interrupt (if enabled by the corresponding CFGm register).
0	[31:3]	r	**Reserved** Read as 0; should be written with 0.

6.8 Interconnects

The interfaces of the FCE module shall be described in the module design specification.

The **Table 6-4** shows the services requests of the FCE module.

Table 6-4 FCE Service Requests

Inputs/Outputs	I/O	Connected To	Description
FCE.SR0	O	NVIC	Service request line

6.9 Properties of CRC code

Hamming Distance

The Hamming distance defines the error detection capability of a CRC polynomial. A cyclic code with a Hamming Distance of D can detect all D-1 bit errors. **Table 6-5 “Hamming Distance as a function of message length (bits)” on Page 6-25** shows the dependency of the Hamming Distance with the length of the message.

Table 6-5 Hamming Distance as a function of message length (bits)[1)]

Hamming Distance	IEEE-802.3 CRC32	CCITT CRC16	J1850 CRC8
15	8 - 10	Information not available	Information not available
14	8 - 10		
13	8 - 10		
12	11 - 12		
11	13 - 21		
10	22 - 34		
9	35 - 57		
8	58 - 91		
7	92 - 171		
6	172 - 268		
5	269 - 2974		
4	2973 - 91607		
3	91607 - 131072		

1) Data from technical paper “32-Bit Cyclic Redundancy Codes for Internet Applications” by Philip Koopman, Carnegie Mellon University, 2002

On-Chip Memories

7 Memory Organization

This chapter provides description of the system Memory Organization and basic information related to Parity Testing and Parity Error handling.

References

[8] Cortex®-M4 User Guide, ARM DUI 0508B (ID062910)

7.1 Overview

The Memory Map is intended to balance decoding cost at various level of the system bus infrastructure.

7.1.1 Features

The Memory Map implements the following features:

- Compatibility with standard ARM Cortex-M4 CPU **[8]**
- Compatibility across entire XMC4000 Family
- Optimal functional module address spaces grouping

7.1.2 Cortex-M4 Address Space

The system memory map defines several regions. Address boundaries of each of the regions are determined by the Cortex-M4 core architecture.

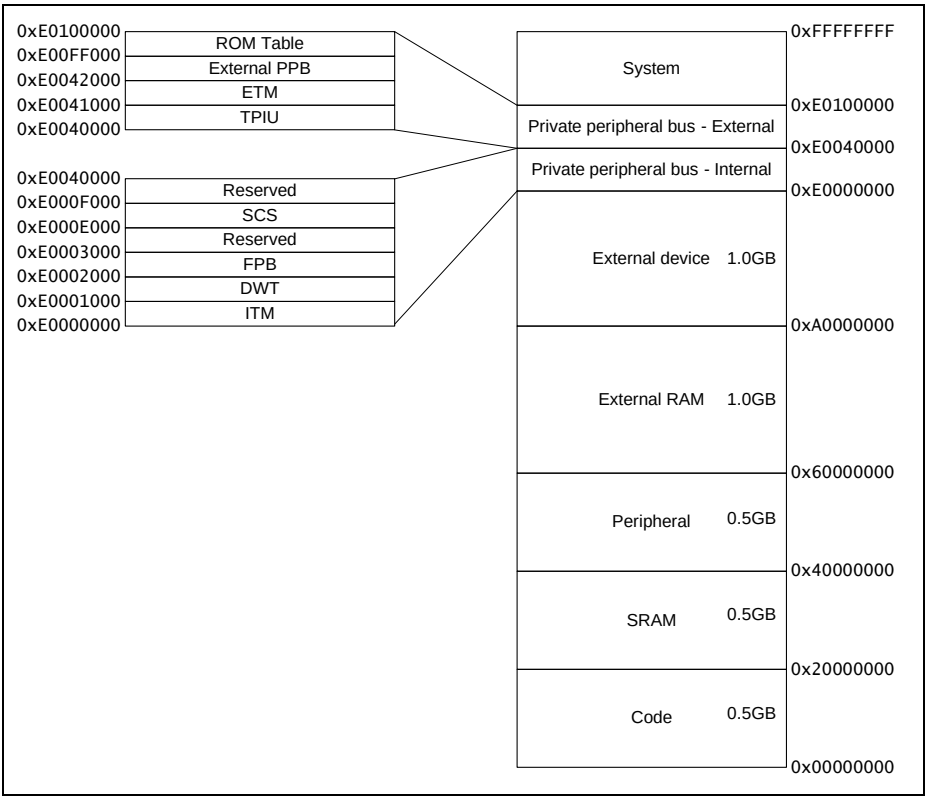


Figure 7-1 Cortex-M4 processor address space

7.2 Memory Regions

The XMC4500 device specific address map assumes presence of internal and external memories and peripherals. The memory regions for XMC4500 are described in Table 7-1.

Table 7-1 Memory Regions

Start	End	Size (hex)	Space name	Usage
00000000	1FFFFFFF	20000000	Code	Boot ROM Flash Program SRAM
20000000	3FFFFFFF	20000000	SRAM	Fast internal SRAMs
40000000	47FFFFFF	08000000	Peripheral 0	Internal Peripherals group 0
48000000	4FFFFFFF	08000000	Peripheral 1	Internal Peripherals group 1
50000000	57FFFFFF	08000000	Peripheral 2	Internal Peripherals group 2
58000000	5FFFFFFF	08000000	Peripheral 3	Internal Peripherals group 3
60000000	9FFFFFFF	40000000	External SRAM	External Memories
A0000000	DFFFFFFF	40000000	External Device	External Devices
E0000000	E00FFFFF	00100000	Private Peripheral Bus	CPU
E0100000	EFFFFFFF	0FF00000	Vedor specific 1	reserved
F0000000	FFFFFFFF	10000000	Vedor specific 2	reserved

7.3 Memory Map

Table 7-2 defines detailed system memory map of XMC4500 where each individual peripheral or memory instance implement its own address spaces. For detailed register description of the system components and peripherals please refer to respective chapters of this document.

Table 7-2 Memory Map

Addr space	Start Address (hex)	End Address (hex)	Modules
Code	00000000	00003FFF	BROM (PMU ROM)
	00004000	07FFFFFF	reserved
	08000000	080FFFFF	PMU/FLASH (cached)
	08100000	09E1FFFF	reserved
	09E20000	09E23FFF	reserved
	09E24000	0BFFFFFF	reserved
	0C000000	0C0FFFFF	PMU/FLASH (uncached)
	0C100000	0FFFFFFF	reserved
	0DE20000	0DE23FFF	reserved
	0DE24000	0FFFFFFF	reserved
	10000000	1000FFFF	PSRAM (code)
	10010000	1FFFFFFF	reserved
SRAM	20000000	2000FFFF	DSRAM1 (system)
	20010000	2FFFFFFF	reserved
	30000000	30007FFF	DSRAM2 (comm)
	30008000	3FFFFFFF	reserved

Table 7-2 Memory Map (cont'd)

Addr space	Start Address (hex)	End Address (hex)	Modules
Peripherals 0	40000000	40003FFF	PBA0
	40004000	40007FFF	VADC
	40008000	4000BFFF	DSD
	4000C000	4000FFFF	CCU40
	40010000	40013FFF	CCU41
	40014000	40017FFF	CCU42
	40018000	4001BFFF	reserved
	4001C000	4001FFFF	reserved
	40020000	40023FFF	CCU80
	40024000	40027FFF	CCU81
	40028000	4002BFFF	POSIF0
	4002C000	4002FFFF	POSIF1
	40030000	40033FFF	USIC0
	40034000	40037FFF	reserved
	40038000	4003BFFF	reserved
	4003C000	4003FFFF	reserved
	40044000	40047FFF	ERU1
	40048000	47FFFFFF	reserved
Peripherals 1	48000000	48003FFF	PBA1
	48004000	48007FFF	CCU43
	48008000	4800BFFF	reserved
	4800C000	4800FFFF	reserved
	48010000	48013FFF	LEDTS0
	48014000	48017FFF	CAN
	48018000	4801BFFF	DAC
	4801C000	4801FFFF	SDMMC
	48020000	48023FFF	USIC1
	48024000	48027FFF	USIC2
	48028000	4802BFFF	PORTS
	4802C000	4FFFFFFF	reserved

Table 7-2 Memory Map (cont'd)

Addr space	Start Address (hex)	End Address (hex)	Modules
Peripherals 2	50000000	50003FFF	PBA2
	50004000	50007FFF	SCU & RTC
	50008000	5000BFFF	WDT
	5000C000	5000FFFF	ETH
	50010000	50013FFF	reserved
	50014000	50017FFF	DMA0
	50018000	5001BFFF	DMA1
	5001C000	5001FFFF	reserved
	50020000	50023FFF	FCE
	50024000	5003FFFF	reserved
	50040000	5007FFFF	USB
	50080000	57FFFFFF	reserved
Peripherals 3	58000000	58003FFF	PMU0 registers
	58004000	58007FFF	PMU0 prefetch
	58008000	5800BFFF	EBU registers
	5800C000	5800FFFF	reserved
	58010000	58013FFF	reserved
	58014000	58017FFF	reserved
	58018000	5FFFFFFF	reserved
External SRAM	60000000	63FFFFFF	EBU memory CS0
	64000000	67FFFFFF	EBU memory CS1
	68000000	6BFFFFFF	EBU memory CS2
	6C000000	6FFFFFFF	EBU memory CS3
	70000000	9FFFFFFF	reserved
External Device	A0000000	A3FFFFFF	EBU devices CS0
	A4000000	A7FFFFFF	EBU devices CS1
	A8000000	ABFFFFFF	EBU devices CS2
	AC000000	AFFFFFFF	EBU devices CS3
	B0000000	DFFFFFFF	reserved

Table 7-2 Memory Map (cont'd)

Addr space	Start Address (hex)	End Address (hex)	Modules
Private Peripheral Bus	E0000000	E0000FFF	ITM
	E0001000	E0001FFF	DWT
	E0002000	E0002FFF	FPB
	E0003000	E000DFFF	reserved
	E000E000	E000EFFF	SCS
	E000E010	E000E01C	SysTick
	E000EF34	E000EF47	FPU
	E000F000	E003FFFF	reserved
	E0040000	E0040FFF	TPIU
	E0041000	E0041FFF	ETM
	E0042000	E00FEFFF	reserved
	E00FF000	E00FFFFF	ROM Table
Vedor specific 1	E0100000	EFFFFFFF	reserved
Vedor specific 2	F0000000	FFFFFFFF	reserved

7.4 Service Request Generation

Memory modules and other system components are capable of generating error reposnses indicated to the CPU as bus error exceptions or interrupts.

Types of error causes

- Unsupported Access Mode
- Access to Invalid Address
- Parity Error (memories only)
- Bufferable Write Accesses to Peripheral

Errors that cannot be indicated with bus errors get indicated with service requests that get propagated to CPU as interrupts. Typically lack of bus error response capability applies to memory modules that lack of direct access from the system bus This applies to memories that serve the purpose of internal FIFOs and local storage buffers.

Unsupported Access Modes

Unsupported access modes can be clasiffied in various ways and are usually specific to the module that access is performed to. The typical examples of unsupported access modes are read access to write-only or write access to read-only type of address

mapped ressources, unsupported access data witdhs, protected memory regions. For module specific limitations please refer to individual module chapters.

Invalid Address

Accesses to invalid addresses result in error responses. Invalid addresses are defined as those that do not mapped to any valid ressources. This applies to single addresses and to wider address ranges. Some invalid addresses within valid module address ranges may not produce error responses and this is specific to individual modules.

Parity Errors

Parity test is performed on the XMC4500 memories in normal functional mode. Parity errors are generated in case of failure of parity test performed inside of each of the memory module. The mechnism of parity testing depends on memory data width and access mode, i.e. memory modules that are accessible byte-wise implement parity check for each data byte individually while for memory modules that are accessible double-word-wise it is sufficient to perform joint check for all bits.

The occurrence of a parity error gets signalized to the system with system bus error or an interrupt (parity trap). For details on parity errror generation control and handling please refer to the SCU chapter. For more details please refer to **Table 7-3**.

Table 7-3 Parity Test Enabled Memories and Supported Parity Error Indication

Memory	Number of Parity Bits	Parity Test Granularity	Bus Error	Parity Trap
Program SRAM (PSRAM)	1	4 bytes	yes	yes
System SRAM (DSRAM1)	4	1 byte	yes	yes
Communication SRAM (DSRAM2)	4	1 byte	yes	yes
USIC 0 Buffer Memory	1	4 bytes	no	yes
USIC 1 Buffer Memory	1	4 bytes	no	yes
USIC 2 Buffer Memory	1	4 bytes	no	yes
MultiCAN Buffer Memory	1	4 bytes	no	yes
PMU Prefetch Buffer Memory	1	4 bytes	no	yes
USB Buffer Memory	1	4 bytes	no	yes
ETH 0 TX Buffer Memory	1	4 bytes	no	yes
ETH 0 RX Buffer Memory	1	4 bytes	no	yes
SDMMC Buffer Memory 0	1	4 bytes	no	yes
SDMMC Buffer Memory 1	1	4 bytes	no	yes

Bufferable Write Access to Peripheral

Bufferable writes to peripheral may result in error reponses as described above. Bus error responses from modules attached to peripheral bridges PBA0 and PBA1 trigger service request from the respective bridge that will result in NMI to the CPU. Error status and access address that caused the service request get stored in dedicated registers of the peripheral bridges. For detail please refer to **Registers**.

7.5 Debug Behavior

The bus system in debug mode allows debug probe access to all system ressources except for the Flash sectors protected with a dedicated protection mechanism (for more details please refer to Flash Memory chapter). No special handling of HALT mode is implemented and all interfaces respond with a valid bus response upon accesses.

7.6 Power, Reset and Clock

The bus system clocking scheme enables stable system operation and accesses to system ressources for all valid system clock rates. Some parts of the system may also run at a half of the system clock rate and no special handling is required as appropriate alignment of the bus system protocol is provided on the clock domain boundary (for details please refer to clocking system description in SCU chapter).

7.7 Initialization and System Dependencies

No initialization is required for the memory system from user point of view. All valid memories are available after reset. Some peripherals may need to be initialized (e.g. released from reset state) before accessed. For details please refer to individual peripheral chapters.

7.8 Registers

This section describes registers of the Peripheral Bridges. The purpose of the registers is handling of errors signalized during bufferable accesses to peripherals connected to the respective bridges. Aactive errors on bufferable writes trigger interrupt requests geberated from the Peripheral Bridges that can be monitored and cleared in the register defiled in this chapter.

Table 7-4 Registers Address Space

Module	Base Address	End Address	Note
PBA0	$4000\ 0000_H$	$4000\ 3FFF_H$	Peripheral Bridge 0
PBA1	$4800\ 0000_H$	$4800\ 3FFF_H$	Peripheral Bridge 1

Table 7-5 Registers Overview

Register Short Name	Register Long Name	Offset Address	Access Mode		Description
			Read	Write	
PBA0_STS	PBA 0 Status Register	0000_H	U, PV	PV	Page 7-10
PBA0_WADDR	PBA 0 Write Error Address	0004_H	U, PV		Page 7-11
PBA1_STS	PBA 1 Status Register	0000_H	U, PV	PV	Page 7-12
PBA1_WADDR	PBA 1 Write Error Address	0004_H	U, PV		Page 7-12

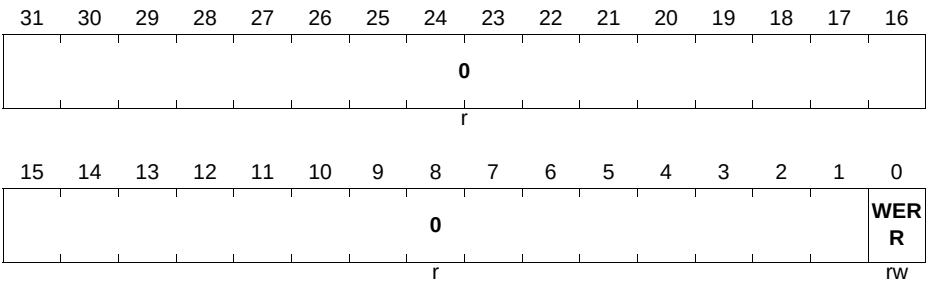
PBA0_STS

The status register of PBA0 bridge indicates bus error occurrence for write access. Is meant to be used for errors triggered upon buffered writes. The bit gets set and interrupt request has been generated to the SCU.

Write one to clear, writing zero has no effect.

PBA0_STS

Peripheral Bridge Status Register (0000$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
WERR	0	rw	**Bufferable Write Access Error** 0_B no write error occurred. 1_B write error occurred, interrupt request is pending.
0	[31:1]	r	**Reserved bits. Write zeros**

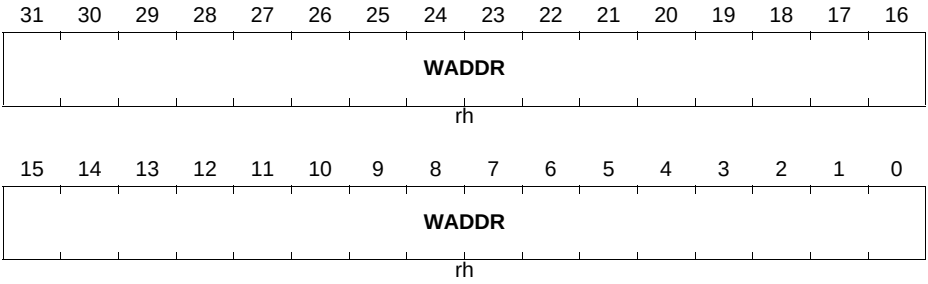
PBA0_WADDR

The Write Error Address Register keeps write access address that caused a bus error upon bufferable write attempt to a peripheral connected to PBA0 bridge. This register store the address that of the bufferable write access attempt that caused error resulting in setting WERR bit of the **PBA0_STS** register.

This register value remains unchanged when WERR bit of **PBA0_STS** register is set.

PBA0_WADDR

PBA Write Error Address Register (0004$_H$) Reset Value: 0000 0000$_H$



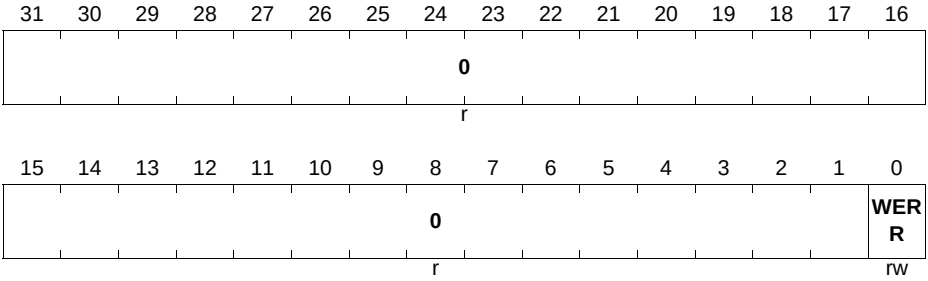
Field	Bits	Type	Description
WADDR	[31:0]	rh	**Write Error Address** Address of the write access that caused a bus error on the bridge Master port.

PBA1_STS

The status register of PBA1 bridge indicates bus error occurrence for write access. Is meant to be used for errors triggered upon buffered writes. The bit gets set and interrupt request has been generated to the SCU.

Write one to clear, writing zero has no effect.

PBA1_STS
Peripheral Bridge Status Register ($(0000)_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
WERR	0	rw	**Bufferable Write Access Error** 0_B no write error occurred. 1_B write error occurred, interrupt request is pending.
0	[31:1]	r	**Reserved bits. Write zeros**

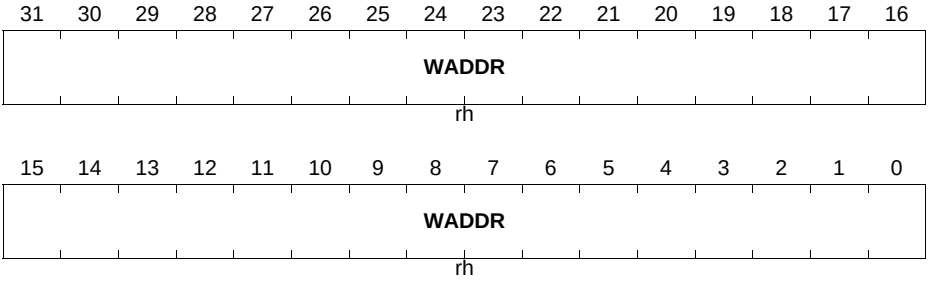
PBA1_WADDR

The Write Error Address Register keeps write access address that caused a bus error upon bufferable write attempt to to a peripheral connected to PBA1 bridge. This register store the address that of the bufferable write access attempt that caused error resulting in setting WERR bit of the **PBA1_STS** register.

This register value remains unchanged when WERR bit of **PBA1_STS** register is set.

PBA1_WADDR

PBA Write Error Address Register (0004_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
WADDR	[31:0]	rh	**Write Error Address** Address of the write access that caused a bus error on the bridge Master port.

8 Flash and Program Memory Unit (PMU)

The Program Memory Unit (PMU) controls the Flash memory and the BROM and connects these to the system. The Prefetch unit maximizes system performance with higher system frequencies, by buffering instruction and data accesses to the Flash.

8.1 Overview

In the XMC4500, the PMU controls the following interfaces:

- The Flash command and fetch control interface for Program Flash
- The Boot ROM interface
- The PMU interfaces via the Prefetch unit to the Bus Matrix

Following memories are controlled by and belong to the PMU:

- 1.0 Mbyte of Program Flash memory (PFLASH)
- 16 Kbyte of BROM (BROM)
- 4 Kbyte of Instruction Cache memory in the Prefetch unit
- 256-bit Data Buffer in the Prefetch unit

8.1.1 Block Diagram

The PMU block diagram is shown in **Figure 8-1**.

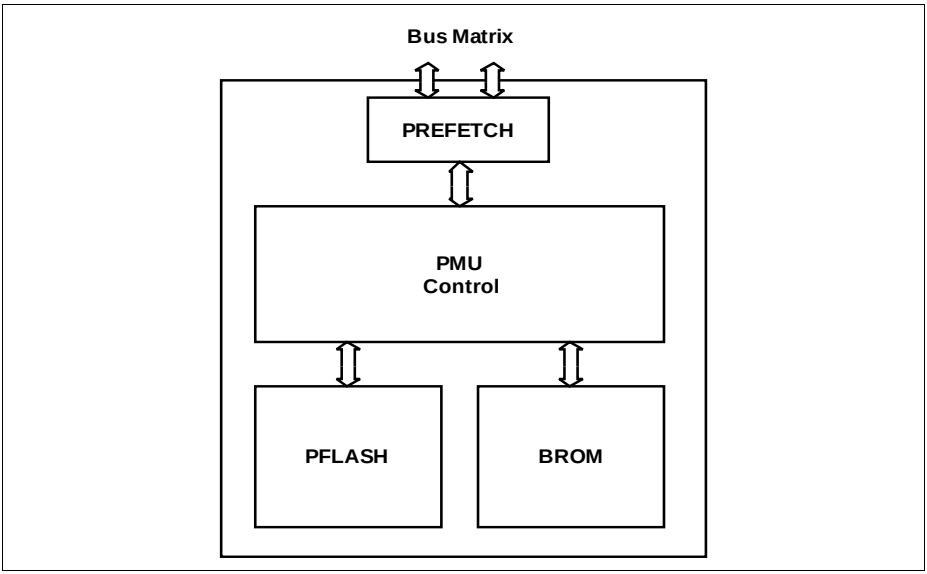


Figure 8-1 PMU Block Diagram

8.2 Boot ROM (BROM)

The Boot ROM in PMU0 has a capacity of 16 KB. The BROM contains the Firmware with:

- startup routines
- bootstrap loading software.

Details on the operations of the BROM are given in the chapter “Startup Modes”.

8.2.1 BROM Addressing

The BROM is visible at one location, as can be seen in the memory map:

- (non-cached space) starting at location $0000\ 0000_H$

After any reset, the hardware-controlled start address is $0000\ 0000_H$. At this location, the startup procedure is stored and started. As no other start location after reset is supported, the startup software within the BROM is always executed first after any reset.

8.3 Prefetch Unit

The purpose of the Prefetch unit is to reduce the Flash latency gap at higher system frequencies to increase the instruction per cycle performance.

8.3.1 Overview

The Prefetch unit separates between instruction and data accesses to the Flash with the following configuration:

- 4 Kbyte Instruction Buffer
  - 2-way set associative
  - Least-Recently-Used (LRU) replacement policy
  - Cache line size: 256 bits
  - Critical word first
  - Streaming[1)]
  - Line wrap around
  - Parity, 32-bit granularity
  - Buffer can be bypassed
  - Buffer can be globally invalidated
- 256-bit Data Buffer
  - Single line
  - Critical word first
  - Streaming[1)]
  - Line Wrap around

1) The first 32-bit data from Flash gets immediately forwarded to the CPU

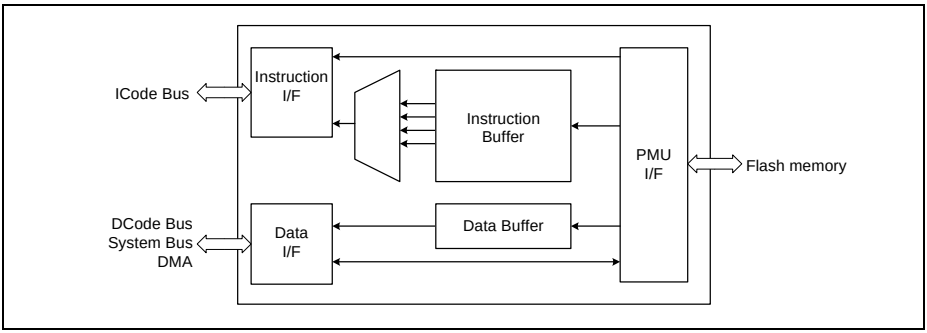


Figure 8-2 Prefetch Unit

8.3.2 Operation

8.3.2.1 Instruction Buffer

The instruction buffer acts like a regular instruction cache with the characteristics described in the overview, optimized for minimum latency via the dedicated instruction interface. Instruction fetches to the non-cacheable address space bypass the instruction buffer. For software development and benchmarking purposes the cacheable accesses can also bypass the instruction buffer by setting **PREF_PCON**.IBYP to 1_B.

Prefetch buffer hits are without any penalty i.e. single cycle access rate. This ensures a minimized latency.

The instruction buffer may be invalidated by writing a 1_B to **PREF_PCON**.IINV. After system reset, the instruction buffer is automatically invalidated.

A parity error during a buffer read operation is automatically turned into a buffer miss, triggering a refill operation of the cache line.

Note: The complete invalidation operation is performed in a single cycle.

Note: The parity information is generated on the fly during the cache refill operation. Parity is checked for each read operation targeting the instruction buffer.

The streaming operation is on the fly - it does not cause any additional latency.

8.3.2.2 Data Buffer

The characteristics of the data buffer are described in the overview. It is used for data read requests from the CPU using the DCode interface and for data read requests from

the DMA. CPU read accesses to the prefetch buffer are without any penalty i.e. single cycle access rate. The miss latency is minimized.

The data interface is shared between DMA requests, CPU DCode bus requests and CPU System bus requests. The CPU System bus is attached to the Prefetch unit to access configuration and status registers within the Prefetch unit and the PMU and Flash. All read requests outside the cacheable address space and all write accesses bypass the data buffer.

Note: The streaming operation is on the fly - it does not cause any additional latency.

8.3.2.3 PMU Interface

Each Flash read access returns 256 bits, intermediately stored in a “global read buffer” in the Flash (**Chapter 8.4.4**). The Prefetch unit reads from this buffer via a 64-bit interface. Cacheable read accesses that are not yet stored in the Prefetch buffer (cache miss) trigger a refill operation by a 4x64-bit burst transfer. By that burst transfer the data from the global buffer is copied, refilling the instruction buffer (code fetch) or data buffer (data fetch) respectively.
Only the initial Flash read access is affected by the Flash latency. The subsequent read accesses of the burst transfer are serviced by the global read buffer with no additional delay. An additional prefetch mechanism in the PFLASH further reduces the latency for linear Flash accesses (**Chapter 8.4.4**).

Non-cacheable accesses benefit from the global read buffer in the same way, as long as its content is not “trashed” by a new Flash read access (e.g. from a different bus master).

Accesses to the BROM and register address spaces and write operations are ignored by the Prefetch buffers.

8.4 Program Flash (PFLASH)

This chapter describes the embedded Flash module of the XMC4500 and its software interface.

8.4.1 Overview

The embedded Flash module of XMC4500 includes 1.0 MB of Flash memory for code or constant data (called Program Flash).

8.4.1.1 Features

The following list gives an overview of the features implemented in the Program Flash. Absolute values can be found in the “Data Sheet”.

- Consists of one bank.
- Commonly used for instructions and constant data.
- High throughput burst read based on a 256-bit Flash access.
- Application optimized sector structure with sectors ranging from 16 Kbytes to 256 Kbytes.
- High throughput programming of a 256 byte page (see Data Sheet t_{PRP}).
- Sector-wise erase on logical and physical sectors (see Data Sheet t_{ERP}).
- Write protection separately configurable for groups of sectors.
- Hierarchical write protection control with 3 levels of which 2 are password based and 1 is a one-time programmable one.
- Password based read protection combined with write protection for the whole Flash.
- Separate configuration sector containing the protection configuration and boot configuration (BMI).
- All Flash operations initiated by command sequences as protection against unintended operation.
- Erase and program performed by a Flash specific control logic independent of the CPU.
- End of erase and program operations reported by interrupt.
- Dynamic Error Correcting Code (ECC) with Single-bit Error Correction and Double-bit Error Detection (“SEC-DED”).
- Error reporting by bus error, interrupts and status flags.
- Margin reads for quality assurance.
- Delivery in the erased state.
- Configurable wait state configuration for optimum read performance depending on CPU frequency (see **FCON**.WSPFLASH).
- High endurance and long retention.
- Pad supply voltage used for program and erase.

8.4.2 Definition of Terms

The description of Flash memories uses a specific terminology for operations and the hierarchical structure.

Flash Operation Terms

- **Erasing**: The erased state of a Flash cell is logical ‘0’. Forcing a cell to this state is called “erasing”. Depending on the Flash area always complete physical sectors, logical sectors are erased. All Flash cells in this area incur one “cycle” that counts for the “endurance”.
- **Programming**: The programmed state of a cell is logical ‘1’. Changing an erased Flash cell to this state is called “programming”. The 1-bits of a page are programmed concurrently.
- **Retention**: This is the time during which the data of a Flash cell can be read reliably. The retention time is a statistical figure that depends on the operating conditions of the device (e.g. temperature profile) and is affected by operations on other Flash cells in the same word-line and physical sector. With an increasing number of program/erase cycles (see endurance) the retention is lowered. Figures are documented in the Data Sheet separately for physical sectors (t_{RET}) and UCBs (t_{RTU}).
- **Endurance**: The maximum number of program/erase cycles of each Flash cell is called “endurance”. The endurance is a statistical figure that depends on operating conditions and the use of the flash cells and also on the required quality level. The endurance is documented in the Data Sheet as a condition to the retention parameters.

Flash Structure Terms

- **Flash Module**: The PMU contains one “Flash module” with its own operation control logic.
- **Bank**: A “Flash module” may contain separate “banks”. “Banks” support concurrent operations (read, program, erase) with some limitations due to common logic.
- **Physical Sector**: A Flash “bank” consists of “physical sectors” ranging from 64 Kbytes to 256 Kbytes. The Flash cells of different “physical sectors” are isolated from each other. Therefore cycling Flash cells in one physical sectors does not affect the retention of Flash cells in other physical sectors. A “physical sector” is the largest erase unit.
- **Logical Sector**: A “logical sector” is a group of word-lines of one physical sector. They can be erased with a single operation but other Flash cells in the same physical sector are slightly disturbed.
- **Sector**: The plain term “sector” means “logical sector” when a physical sector is divided in such, else it means the complete physical sector.
- **User Configuration Block “UCB”**: A “UCB” is a specific logical sector contained in the configuration sector. It contains the protection settings and other data configured

by the user. The “UCBs” are the only part of the configuration sector that can be programmed and erased by the user.

- **Word-Line**: A “word-line” consists of two pages, an even one and an odd one. In the PFLASH a word-line contains aligned 512 bytes.
- **Page**: A “page” is a part of a word-line that is programmed at once. In PFLASH a page is an aligned group of 256 bytes.

8.4.3 Flash Structure

The PMU contains one PFLASH bank, accessible via the cacheable or non-cacheable address space. The offset address of each sector is relative to the base address of its bank which is given in **Table 8-1**.

Derived devices (see Data Sheet) can have less Flash memory. The PFLASH bank shrinks by cutting-off higher numbered physical sectors.

Table 8-1 Flash Memory Map

Range Description	Size	Start Address
PMU0 Program Flash Bank non-cached	1.0 Mbyte	$0C00\ 0000_H$
PMU0 Program Flash Bank cached space (different address space for the same physical memory, mapped in the non-cached address space)	1.0 Mbyte	$0800\ 0000_H$
PMU0 UCB User Configuration Blocks	3 Kbyte	$0C00\ 0000_H$
PMU0 Flash Registers	1 Kbyte	$5800\ 2000_H$

PFLASH

All addresses offset to the start addresses given in **Table 8-1**. All sectors from S9 on have a size of 256 Kbyte.

Table 8-2 Sector Structure of PFLASH

Sector	Phys. Sector	Size	Offset Address
S0	PS0	16 KB	$00'0000_H$
S1		16 KB	$00'4000_H$
S2		16 KB	$00'8000_H$
S3		16 KB	$00'C000_H$

Table 8-2 Sector Structure of PFLASH (cont'd)

Sector	Phys. Sector	Size	Offset Address
S4	PS4	16 KB	$01'0000_H$
S5		16 KB	$01'4000_H$
S6		16 KB	$01'8000_H$
S7		16 KB	$01'C000_H$
S8	–	128 KB	$02'0000_H$
S9	–	256 KB	$04'0000_H$
S10	–	256 KB	$08'0000_H$
S11	–	256 KB	$0C'0000_H$

UCB

All addresses offset to the start addresses given in **Table 8-1**. As explained before the UCBx are logical sectors.

Table 8-3 Structure of UCB Area

Sector	Size	Offset Address
UCB0	1 KB	$00'0000_H$
UCB1	1 KB	$00'0400_H$
UCB2	1 KB	$00'0800_H$

8.4.4 Flash Read Access

Flash banks that are active and in read mode can be directly read like a ROM.

The wait cycles for the Flash read access must be configured based on the CPU frequency f_{CPU} (incl. PLL jitter) in relation to the Flash access time t_a defined in the Data Sheet. The follwing formula applies for **FCON**.WSPFLASH > 0_H[1]:

$$WSPFLASH \times (1 / f_{CPU}) \geq t_a \quad (8.1)$$

The PFLASH delivers 256 bits per read access. All read data from the PFLASH passes through a 256-bit "global read buffer".

The PMU allows 4x64-bit burst accesses to the cached address space and single 32-bit read accesses to the non-cached address space of the PFLASH.

1) WSPFLASH = 0_H deviates from this formula and results in the same timing as WSPFLASH = 1_H.

The Prefetch generates the 4x64-bit bursts for code and data fetches from the cached address range in order to fill one cache line or the data buffer respectively. Data reads from the non-cached address range are performed with single 32-bit transfers.

Following an inital Flash access, the PFLASH automatically starts a prefetch of the next linear address (even before it has been requested). Has the content of the global read buffer been read completely (e.g. by a burst from the Prefetch unit), the new prefetched data is copied to the read buffer and another prefetch to the PFLASH is started. This significantly reduces the Flash latency for mostly linearly accessed code or data sections. To avoid additional wait states due to these prefetches, they can be aborted in case a new (initial) read access is requested from a different address. For power saving purposes these prefetch operations can be disabled by **FCON**.IDLE (**Idle Read Path**).

Read accesses from Flash can be blocked by the read protection (see **Chapter 8.4.8**).

ECC errors can be detected and corrected (see **Chapter 8.4.9**).

8.4.5 Flash Write and Erase Operations

Flash write and erase operations are triggered by **Command Sequences** to avoid harm to the stored data by “accidential” accesses from faulty code. Erase operations are executed on sectors, write operations on pages.

Attention: Flash write and erase operations must be executed to the non-cacheable address space.

8.4.6 Modes of Operation

A Flash module can be in one of the following states:

- Active (normal) mode.
- Sleep mode (see **Chapter 8.6.2**).

In sleep mode write and read accesses to all Flash ranges of this PMU are refused with a bus error.

When the Flash module is in active mode the Flash bank can be in one of these modes:

- Read mode.
- Command mode.

In read mode a Flash bank can be read and command sequences are interpreted. In read mode a Flash bank can additionally enter page mode which enables it to receive data for programming.

In command mode an operation is performed. During its execution the Flash bank reports BUSY in **FSR**. In this mode read accesses to this Flash bank are refused with a bus error. At the end of an operation the Flash bank returns to read mode and BUSY is cleared. Only operations with a significant duration (shown in the command documentation) set BUSY.

Register read and write accesses are not affected by these modes.

8.4.7 Command Sequences

All Flash operations except read are performed with command sequences. When a Flash bank is in read mode or page mode all write accesses to its reserved address range are interpreted as command cycle belonging to a command sequence. Write accesses to a busy bank cause a sequence error (SQER).

Attention: For the proper execution of the command sequences and the triggered operations f_{CPU} must be equal or above 1 MHz.

Command sequences consist of 1 to 6 command cycles. The command interpreter checks that a command cycle is correct in the current state of command interpretation. Else a SQER is reported.

When the command sequence is accepted the last command cycle finishes read mode and the Flash bank transitions into command mode.

These write accesses must be single transfers and must address the non-cacheable address range.

Generally when the command interpreter detects an error it reports a sequence error by setting **FSR**.SQER. Then the command interpreter is reset and a page mode is left. The next command cycle must be the 1st cycle of a command sequence. The only exception is “Enter Page Mode” when a bank is already in page mode (see below).

8.4.7.1 Command Sequence Definitions

Table 8-4 gives an overview of the supported command sequence, with the following nomenclature:

The parameter “addr” can be one of the following:

- **$CCCC_H$**: The “addr” must point into the bank that performs the operation. The last 16 address bits must match $CCCC_H$. It is recommended to use as address the base address of the bank incremented by $CCCC_H$.
- **PA**: Absolute start address of the Flash page.
- **UCPA**: Absolute start address of a user configuration block page.
- **SA**: Absolute start address of a Flash sector. Allowed are the PFLASH sectors Sx.
- **PSA**: Absolute start address of a physical sector. Allowed are the PFLASH physical sectors PSx.
- **UCBA**: Absolute start address of a user configuration block.

The parameter “data” can be one of the following:

- **WD**: 64-bit or 32-bit write data to be loaded into the page assembly buffer.
- **xxYY**: 8-bit write data as part of a command cycle. Only the byte “YY” is used for command interpretation. The higher order bytes “xx” are ignored.
  - **xx5y**: Specific case for “YY”. The “y” can be “0_H” for selecting the PFLASH bank.

- **UL**: User protection level ($xxx0_H$ or $xxx1_H$ for user levels 0 and 1).
- **PWx**: 32-bit password.

When using for command cycles 64-bit transfers the “data” is expected in the correct 32-bit word as indicated by the address “addr”.

Command Sequence Overview Table

The **Table 8-4** summarizes all commands sequences. The following sections describe each command sequence in detail.

Table 8-4 Command Sequences for Flash Control

Command Sequence		1. Cycle	2. Cycle	3. Cycle	4. Cycle	5. Cycle	6. Cycle
Reset to Read	Address Data	.5554 ..xx**F0**					
Enter Page Mode	Address Data	.5554 ..xx**5y**					
Load Page	Address Data	.55F0 **WD**					
Write Page	Address Data	.5554 ..xx**AA**	.AAA8 ..xx**55**	.5554 ..xx**A0**	**PA** ..xx**AA**		
Write User Configuration Page	Address Data	.5554 ..xx**AA**	.AAA8 ..xx**55**	.5554 ..xx**C0**	**UCPA** ..xx**AA**		
Erase Sector	Address Data	.5554 ..xx**AA**	.AAA8 ..xx**55**	.5554 ..xx**80**	.5554 ..xx**AA**	.AAA8 ..xx**55**	**SA** ..xx**30**
Erase Physical Sector	Address Data	.5554 ..xx**AA**	.AAA8 ..xx**55**	.5554 ..xx**80**	.5554 ..xx**AA**	.AAA8 ..xx**55**	**SA** ..xx**40**
Erase User Configuration Block	Address Data	.5554 ..xx**AA**	.AAA8 ..xx**55**	.5554 ..xx**80**	.5554 ..xx**AA**	.AAA8 ..xx**55**	**UCBA** ..xx**C0**
Disable Sector Write Protection	Address Data	.5554 ..xx**AA**	.AAA8 ..xx**55**	.553C **UL**	.AAA8 **PW 0**	.AAA8 **PW 1**	.5558 ..xx**05**
Disable Read Protection	Address Data	.5554 ..xx**AA**	.AAA8 ..xx**55**	.553C ..xx**00**	.AAA8 **PW 0**	.AAA8 **PW 1**	.5558 ..xx**08**
Resume Protection	Address Data	.5554 ..xx**5E**					
Clear Status	Address Data	.5554 ..xx**F5**					

Reset to Read

This function resets the command interpreter to its initial state (i.e. the next command cycle must be the 1st cycle of a sequence). A page mode is aborted.

This command is the only one that is accepted without generating a SQER when the command interpreter has already received command cycles of a different sequence but is still not in command mode. Thus “Reset to Read” can cancel every command sequence before its last command cycle has been received.

The error flags of **FSR** (PFOPER, SQER, PROER, PFDBER, ORIER, VER) are cleared. The flags can be also cleared in the status registers without command sequence.

If any Flash bank is busy this command is executed but the flag SQER is set.

Enter Page Mode

The PFLASH enters page mode. The selection of the PFLASH assembly buffer (256 bytes) is additionally done by the parameter “$y_H = 0_H$”.

The write pointer of the page assembly buffer is set to 0, its previous content is maintained.

The page mode is signalled by the flag PAGEx in the FSR.

If a new “Enter Page Mode” command sequence is received while any Flash bank is already in page mode SQER is set but this sequence is correctly executed (i.e. in this case the command interpreter is not reset).

Load Page

Loads the data “WD” into the page assembly buffer and increments the write pointer to the next position[1)].

All WD transfers for one page must have the same width (either all 32-bit or all 64-bit). Else the transfer is refused with SQER.

The addressed bank must be in page mode, else SQER is issued.

If “Load Page” is called more often than necessary for filling the page SQER is issued and if configured an interrupt is triggered. The overflow data is discarded. The page mode is not left.

Write Page

This function starts the programming process for one page with the data transferred previously by “Load Page” commands. Upon entering command mode the page mode

1) More specifically: after “Load Page” has transferred 64 bits (i.e. two command with 32-bit WD or one command with one 64-bit WD) the ECC is calculated and the result is transferred to the assembly buffer.

is finished (indicated by clearing the corresponding PAGE flag) and the BUSY flag of the bank is set.

This command is refused with SQER when the addressed Flash bank is not in page mode.

SQER is also issued when PA addresses an unavailable Flash range or when PA does not point to a legal page start address.

If after “Enter Page Mode” too few data or no data was transferred to the assembly buffer with “Load Page” then “Write Page” programs the page but sets SQER. The missing data is programmed with the previous content of the assembly buffer.

When the page “PA” is located in a sector with active write protection or the Flash module has an active global read protection the execution fails and PROER is set.

Write User Configuration Page

As for “Write Page”, except that the page “UCPA” is located in a user configuration block. This changes the Flash module’s protection configuration.

When the page “UCPA” is located in an UCB with active write protection or the Flash module has an active global read protection the execution fails and PROER is set.

When UCPA is not the start address of a page in a valid UCB the command fails with SQER.

Erase Sector

The sector “SA” is erased.

SQER is returned when SA does not point to the base address of a correct sector (as specified at the beginning of this section) or to an unavailable sector.

When SA has an active write protection or the Flash module has an active global read protection the execution fails and PROER is set.

Erase Physical Sector

The physical sector “PSA” is erased.

SQER is returned when PSA does not point to the base address of a correct sector (as specified at the beginning of this section) or an unavailable sector.

When PSA has an active write protection or the Flash module has an active global read protection the execution fails and PROER is set.

Erase User Configuration Block

The addressed user configuration block “UCB” is erased.

When the UCB has an active write protection or the Flash module has an active global read protection the execution fails and PROER is set.

The command fails with SQER when UCBA is not the start address of a valid UCB.

Disable Sector Write Protection

The sector write protection belonging to user level “UL” is temporarily disabled by setting FSR.WPRODIS when the passwords PW0 and PW1 match their configured values in the corresponding UCB.

The command fails by setting PROER when any of PW0 and PW1 does not match. In this case until the next application reset all further calls of “Disable Sector Write Protection” and “Disable Read Protection” fail with PROER independent of the supplied password.

Disable Read Protection

The Flash module read protection including the derived module wide write protection are temporarily disabled by setting FSR.RPRODIS when the passwords PW0 and PW1 match their configured values in the UCB0.

The command fails by setting PROER when any of PW0 and PW1 does not match. In this case until the next application reset all further calls of “Disable Sector Write Protection” and “Disable Read Protection” fail with PROER independent of the supplied password.

Resume Protection

This command clears all FSR.WPRODISx and the FSR.RPRODIS effectively enabling again the Flash protection as it was configured.

A FSR.WPRODISx is not cleared when corresponding UCBx is not in the “confirmed” state (see **Chapter 8.4.8.1**).

Clear Status

The flags FSR.PROG and FSR.ERASE and the error flags of **FSR** (PFOPER, SQER, PROER, PFDBER, ORIER, VER) are cleared. These flags can be also cleared in the status registers without command sequence.

When any Flash bank is busy this command fails by setting additionally SQER.

8.4.8 Flash Protection

The Flash memory can be read and write protected. The protection is configured by programming the User Configuration Blocks “UCB”.

For an effective IP protection the Flash read protection must be activated. This ensures system wide that the Flash cannot be read from external or changed without authorization.

8.4.8.1 Configuring Flash Protection in the UCB

As indicated above the effective protection is determined by the content of the **Protection Configuration Indication** PROCON0–2 registers. These are loaded during startup from the UCB0–2. Each UCB comprises 1 Kbyte of Flash organized in 4 UC pages of 256 bytes. The UCBs have the following structure:

Table 8-5 UCB Content

UC Page	Bytes	UCB0	UCB1	UCB2
0	[1:0]	PROCON0	PROCON1	PROCON2
	[7:2]	unused	unused	unused
	[9:8]	PROCON0 (copy)	PROCON1 (copy)	PROCON2 (copy)
	[15:10]	unused	unused	unused
	[19:16]	PW0 of User 0	PW0 of User 1	unused
	[23:20]	PW1 of User 0	PW1 of User 1	unused
	[27:24]	PW0 of User 0 (copy)	PW0 of User 1 (copy)	unused
	[31:28]	PW1 of User 0 (copy)	PW1 of User 1 (copy)	unused
	others	unused	unused	unused
1		unused	unused	BMI and configuration data (details in Startup Mode chapter)
2	[3:0]	confirmation code	confirmation code	confirmation code
	[11:8]	confirmation code (copy)	confirmation code (copy)	confirmation code (copy)
	others	unused	unused	unused
3	unused	unused	unused	unused

If the confirmation code field is programmed with 8AFE 15C3$_H$ the UCB content is “confirmed” otherwise it is “unconfirmed”. The status flags **FSR**.PROIN, **FSR**.RPROIN and **FSR**.WPROIN0–2 indicate this confirmation state:

- **FSR**.PROIN: set when any UCB is in the confirmed state.
- **FSR**.RPROIN: set when **PROCON0**.RPRO is ‘1’ and the UCB0 is in “confirmed” state.
- **FSR**.WPROIN0–2: set when their UCB0–2 is in “confirmed” state.

An UCB can be erased with the command “Erase User Configuration Block”. An UCB page can be programmed with the command “Write User Configuration Page”. These commands fail with PROER when the UCB is write-protected.

An UCB is write-protected if:

- UCB0: (FSR.RPROIN and not **FSR**.RPRODIS) or (**FSR**.WPROIN0 and not **FSR**.WPRODIS0)
- UCB1: **FSR**.WPROIN1 and not **FSR**.WPRODIS1.
- UCB2: **FSR**.WPROIN2

So when the UCB2 is in the “confirmed” state its protection can not be changed anymore. Therefore this realizes a one-time programmable protection.

Changing UCBs

The protection installation is modified by erasing and programming the UCBs with dedicated command sequences, described in **Chapter 8.4.7.1**. These operations need to be performed with care as described in the following.

Aborting an “Erase UC Block” operation (e.g. due to reset or power failure) must be avoided at all means, as it can result in an unusable device.

UCBs are logical sectors, and as such the allowed number of program/erase cycles of the UCBs must not be exceeded. Over-cycling the UCBs can also lead to an unusable device.

The installation of the protection and its confirmation on different pages of the UCB offers the possibility to check the installation before programming the confirmation. First the protection needs to be programmed, then an application reset must be triggered to trigger the reading of the UCBs by the PMU and after that the protection can be verified (e.g. “Disable … Protection” to check the password and by checking PROCONs and **FCON**). The application reset is inevitable because the PMU reads the UCBs only during the startup phase.

8.4.8.2 Flash Read Protection

Read protection can be activated for the whole Flash module.

Read Protection Status

A read access to PFLASH fails with bus error under the following conditions:

- Code fetch: **FCON**.DCF and **FCON**.RPA.
- Data read: **FCON**.DDF and **FCON**.RPA.

The read protection bit **FCON**.RPA is determined during startup by the protection configuration of UCB0. It can be temporarily modified by the command sequences “Disable Read Protection” and “Resume Protection” which modify **FSR**.RPRODIS. **FCON**.RPA is determined by the following equation:

- **FCON**.RPA = **PROCON0**.RPRO and not **FSR**.RPRODIS.

The bits **FCON**.DDF and **FCON**.DCF are initialized by the startup software depending on the configured protection and the startup mode. They can also be directly modified by the user software under conditions noted in the description of **FCON**.

Initializing Read Protection

Installation of read protection is performed with the “Write User Configuration Page” operation, controlled by the user 0. With this command, user 0 writes the protection configuration bits RPRO, and the two 32-bit keywords into the UCB0-page 0. Additionally, with a second “Write User Configuration Page” command, a special 32-bit confirmation (lock-) code is written into the UCB0-page2. Only this confirmation code enables the protection and thus the keywords. The confirmation write operation to the second wordline of the User Configuration Block shall be executed only after check of keyword-correctness (with command “Disable Read Protection” after next reset). The confirmed state and thus the installation of protection is indicated with the FSR-bit PROIN in Flash Status Register FSR and for read protection with bit RPROIN in FSR. If read protection is not correctly confirmed and thus not enabled, the bits PROIN and RPROIN in the FSR are not set. The configured read protection as fetched from UCB0 is indicated in the protection configuration register **PROCON0**.

For safety of the information stored in the UCB pages, all keywords, lock bits and the confirmation code are stored two-times in the two wordlines. In case of a disturbed original data detected during ramp up, its copy is used**FSR**. Layout of the four UC pages belonging to the user’s UC block is shown in **Table 8-5**, the command “Write User Configuration Page” is described in **Chapter 8.4.7.1**.

Disabling Read Protection

With the command sequence “Disable Read Protection” short-term disabling of read protection is possible. This command disables the Flash protection (latest until next reset) for user controlled erase and re-program operations as well as for clearing of DCF and DDF control bits after external program execution. The “Disable Read Protection” command sequence is a protected command, which is only processed by the command state machine, if the included two passwords are identical to the two keywords of user 0. The disabled state of read protection is controlled with the **FCON**.RPA=’0’ and indicated in the Flash Status Register **FSR** with the RPRODIS bit (see **Chapter 8.7.3.1**). As long as read protection is disabled (and thus not active), the **FCON**-bits DDF and DCF can be cleared.

Resumption of read protection after disablement is performed with the “Resume Read/Write Protection” command. After execution of this single cycle command, read protection (if installed) is again active, indicated by the **FCON** bit RPA=’1’.

Generally, Flash read protection will remain installed as long as it is confirmed (locked) in the User Configuration Block 0. Erase of UC block and re-program of UC pages can

be performed up to 4 times. But note, after execution of the Erase UC block command (which is protected and therefore requires the preceding disable command with the user's specific passwords), all keywords and all protection installations of user 0 are erased; thus, the Flash is no more read protected (beginning with next reset) until re-programming the UC pages. But the division and separation of the protection configuration data and of the confirmation data into two different UCB-wordlines guarantees, that a disturb of keywords can be discovered and corrected before the protection is confirmed. For this reason, the command sequence "Disable Read Protection" can also be used when protection is programmed (configured) but not confirmed; wrong keywords are then indicated by the error flag PROER.

Read protection can be combined with sector specific write protection. In this case, after execution of the command 'Disable Read Protection' only those sectors are unlocked for write accesses, which are not separately write protected.

8.4.8.3 Flash Write and OTP Protection

A range of Flash can be write protected by several means:

- The complete PFLASH can be write protected by the read protection.
- Groups of sectors of PFLASH can be write-protected by three different "users", i.e. UCBs:
  - UCB0: Write protection that can be disabled with the password of UCB0.
  - UCB1: Write protection that can be disabled with the password of UCB1.
  - UCB2: Write protection that can not be disabled anymore (ROM or OTP function: "One-Time Programmable").

Write and OTP Protection Status

An active write protection is indicated by WPROIN bits in **FSR** register. It causes the program and erase command sequences to fail with a PROER.

A range "x" (i.e. a group of sectors, see **PROCON0**) of the PFLASH is write protected if any of the following conditions is true:

- **FCON**.RPA
- **PROCON2**.SxROM
- **PROCON0**.SxL and not(**FSR**.WPRODIS0)
- **PROCON1**.SxL and not(**PROCON0**.SxL) and not(**FSR**.WPRODIS1)

Thus with the password of UCB0 the write protection of sectors protected by user 0 and user 1 can be disabled, however with the password of UCB1 only those sectors that are only protected by user 1. The write protection of user 2 (OTP) can be obviously not disabled. The global write protection caused by the read protection can be disabled as described above by using the password of UCB0 to disable the read protection.

Initialization of Write and OTP Protection

Installation of write protection is performed with the “Write User Configuration Page” operation, controlled by the user. With this command, the user defines and writes into the UCBx page 0 the write protection configuration bits for all sectors, which shall be locked by the specific user, and the user-specific two keywords (not necessary for user 2). The position of sector lock bits is identical as defined for the PROCON registers. The correctness of keywords shall then (after next reset) be checked with the command ‘Disable Sector Write Protection’, which delivers a protection error PROER in case of wrong passwords. Only if the keywords are correct, the special 32-bit confirmation code must be written into the page 2 of UCBx with a second “Write User Configuration Page” command. Only this confirmation code enables the write protection of the User Control Block UCBx, and only in this case the installation bit(s) in **FSR** is (are) set during ramp up.

Note: If the write protection is configured in the user’s UCB page 0 but not confirmed via page 2 (necessary for check of keywords), the state after next reset is as follows:
- The selected sector(s) are protected (good for testing of protection, also of OTP)
- The UCBx is not protected, thus it can be erased without passwords
*- The related WPROINx bit in **FSR** is not set*
- The Disable Write Protection command sets the WPRODISx bit
- The Resume command does not clear the WPRODISx bit.

The structure and layout of the three UC blocks is shown in **Table 8-3** below, the command “Write User Configuration Page” is described in **Chapter 8.4.7.1**.

Disabling Write Protection (not applicable to OTP)

With the command sequence “Disable Sector Write Protection” short-term disabling of write protection for user 0 or user 1 is possible. This command unlocks temporarily all locked sectors belonging to the user. The “Disable Sector Write Protection” command sequence is a protected command, which is only processed by the command state machine, if the included two passwords are correct. The disabled state of sector protection is indicated in the Flash Status Register **FSR** with the WPRODIS bit of the user 0 or/and user 1 (see **Chapter 8.7.3.1**). For user 2 who owns the sectors with ROM functionality, a disablement of write protection and thus re-programming is not possible.

Resumption of write protection after disablement is performed with the “Resume Read/Write Protection” command, which is identical for user 0 and user 1.

Generally, sector write protection will remain installed as long as it is configured and confirmed in the User Configuration Block belonging to the user. Erase of UC block and re-program of UC pages can be performed up to 4 times, for user 0 and user 1 only. But note, after execution of the Erase UC block command (which is still protected and therefore requires the preceding disablement of write protection with the user’s passwords), the complete protection configuration including the keywords of the specific user (not user 2) is erased; thus, the sectors belonging to the user are unprotected until

the user's UC pages are re-programmed. Only exception: sectors protected by user 2 are locked for ever because the UCB2 can no more be erased after installation of write protection in UCB2.

8.4.8.4 System Wide Effects of Flash Protection

An active Flash read protection needs to be respected in the complete system.

The startup software (SSW) checks if the Flash read protection is active in the PMU, if yes:

- If the selected boot mode executes from internal PFLASH.
  - The SSW clears the DCF and DDF.
  - The SSW leaves the debug interface locked.
- If the selected boot mode does not execute from internal PFLASH:
  - The SSW either leaves DCF and DDF set or actively sets them again in the PMU after evaluating the configuration sector.
  - The debug interface is unlocked.

If the read protection is inactive in the PMU the DCF and DDF flags are cleared by the SSW and the debug interface is unlocked.

Note: Full Flash analysis of an FAR device is only possible when the customer has removed all installed protections or delivers the necessary passwords with the device. As the removal of an OTP protection in UCB2 is not possible the OTP protection inevitably limits analysis capabilities.

8.4.9 Data Integrity and Safety

The data in Flash is stored with error correcting codes "ECC" in order to protect against data corruption. The healthiness of Flash data can be checked with margin checks.

8.4.9.1 Error-Correcting Code (ECC)

The data in the PFLASH is stored with ECC codes. These are automatically generated when the data is programmed. When data is read these codes are evaluated. Data in PFLASH uses an ECC code with SEC-DED (Single Error Correction, Double Error Detection) capabilities. Each block of 64 data bits is accompanied with 8 ECC bits.

Standard PFLASH ECC

In the standard PFLASH ECC the 8-bit ECC value is calculated over 64 data bits. An erased data block (all bits '0') has an ECC value of 00_H. Therefore an erased sector is free of ECC errors. A data block with all bits '1' has an ECC value of FF_H.

The ECC is automatically generated when programming the PFLASH.

The ECC is automatically evaluated when reading data.

This algorithm has the following capabilities:

- Single-bit error:
  - Is noted in **FSR**.PFSBER.
  - Data and ECC value are corrected.
  - Interrupt is triggered if enabled with **FCON**.PFSBERM.
- Double-bit error:
  - Is noted in **FSR**.PFDBER.
  - Causes a bus error if not disabled by **MARP**.TRAPDIS.
  - Interrupt is triggered if enabled with **FCON**.PFDBERM. This interrupt shall only be used for margin check, when the bus error is disabled.

8.4.9.2 Margin Checks

The Flash memory offers a “margin check feature”: the limit which defines if a Flash cell is read as logic ‘0’ or logic ‘1’ can be shifted. This is controlled by the register **MARP**. The Margin Control Register **MARP** is used to change the margin levels for read operations to find problematic array bits. The array area to be checked is read with more restrictive margins. “Problematic” bits will result in a single or double-bit error that is reported to the CPU by an error interrupt or a bus error trap. The double-bit error trap can be disabled for margin checks and also redirected to an error interrupt.

After changing the read margin at least $t_{FL\_MarginDel}$ have to be waited before reading the affected Flash module. During erase or program operation only the standard (default) margins are allowed.

8.5 Service Request Generation

Access and/or operational errors (e.g. wrong command sequences) may be reported to the user by interrupts, and they are indicated by flags in the Flash Status Register **FSR**. Additionally, bus errors may be generated resulting in CPU traps.

8.5.1 Interrupt Control

The PMU and Flash module supports immediate error and status information to the user by interrupt generation. One CPU interrupt request is provided by the Flash module.

The Flash interrupt can be issued because of following events:

- End of busy state: program or erase operation finished
- Operational error (OPER): program or erase operation aborted
- Verify error (VER): program or erase operation not correctly finished
- Protection error
- Sequence error
- Single-bit error: corrected read data from PFLASH delivered
- Double-bit error in Program Flash.

Note: In case of an OPER or VER error, the error interrupt is issued not before the busy state of the Flash is deactivated.

The source of interrupt is indicated in the Flash Status Register **FSR** by the error flags or by the PROG or ERASE flag in case of end of busy interrupt. An interrupt is also generated for a new error event, even if the related error flag is still set from a previous error interrupt.

Every interrupt source is masked (disabled) after reset and can be enabled via dedicated mask bits in the Flash Configuration Register **FCON**.

8.5.2 Trap Control

CPU traps are executed because of bus errors, generated by the PMU in case of erroneous Flash accesses. Bus errors are generated synchronously to the bus cycle requesting the not allowed Flash access or the disturbed Flash read data. Bus errors are issued because of following events:

- Not correctable double-bit error of 64-bit read data from PFLASH (if not disabled for margin check)
- Not allowed write access to read only register (see **Table 8-11**)
- Not allowed write access to Privileged Mode protected register (see **Table 8-11**)
- Not allowed data or instruction read access in case of active read protection
- Access to not implemented addresses within the register or array space.
- Read-modify-write access to the Flash array.

Write accesses to the Flash array address space are interpreted as command cycles and initiate not a bus error but a sequence error if the address or data pattern is not correct. However, command sequence cycles, which address a busy Flash bank, are serviced with busy cycles, not with a sequence error.

If the trap event is a double-bit error in PFLASH, it is indicated in the **FSR**. With exception of this error trap event, all other trap sources cannot be disabled within the PMU.

*Note: A double-bit error trap during margin check can be disabled (via **MARP** register) and redirected to an interrupt request.*

8.5.3 Handling Errors During Operation

The previous sections described shortly the functionality of “error indicating” bits in the flash status register **FSR**. This section elaborates on this with more in-depth explanation of the error conditions and recommendations how these should be handled by customer software. This first part handles error conditions occurring during operation (i.e. after issuing command sequences) and the second part (**Section 8.5.3.6**) error conditions detected during startup.

8.5.3.1 SQER “Sequence Error”

Fault conditions:

- Improper command cycle address or data, i.e. incorrect command sequence.
- New “Enter Page” in Page Mode.
- “Load Page” and not in Page Mode.
- “Load Page” results in buffer overflow.
- First “Load Page” addresses 2. word.
- “Write Page” with buffer underflow.
- “Write Page” and not in Page Mode.
- “Write Page” to wrong Flash type.
- Byte transfer to password or data.
- “Clear Status” or “Reset to Read” while busy[1)].
- Erase UCB with wrong UCBA.

New state:

Read mode is entered with following exceptions:

- “Enter Page” in Page Mode re-enters Page Mode.
- “Write Page” with buffer underflow is executed.
- After “Load Page” causing a buffer overflow the Page Mode is not left, a following “Write Page” is executed.

Proposed handling by software:

Usually this bit is only set due to a bug in the software. Therefore in development code the responsible error tracer should be notified. In production code this error should not occur. It is however possible to clear this flag with “Clear Status” or “Reset to Read” and simply issue the corrected command sequence again.

8.5.3.2 PFOPER “Operation Error”

Fault conditions:

ECC double-bit error detected in Flash module internal SRAM during a program or erase operation in PFLASH. This can be a transient event due to alpha-particles or illegal operating conditions or it is a permanent error due to a hardware defect. This situation will practically not occur.

Attention: these bits can also be set during startup (see **Chapter 8.5.3.6**).

New state:

The Flash operation is aborted, the BUSY flag is cleared and read mode is entered.

Proposed handling by software:

1) When the command addresses the busy Flash bank, the access is serviced withbusy cycles.

The flag should be cleared with “Clear Status”. The last operation can be determined from the PROG and ERASE flags. In case of an erase operation the affected physical sector must be assumed to be in an invalid state, in case of a program operation only the affected page. Other physical sectors can still be read. New program or erase commands must not be issued before the next reset.

Consequently a reset must be performed. This performs a new Flash ramp up with initialization of the microcode SRAM. The application must determine from the context which operation failed and react accordingly. Mostly erasing the addressed sector and re-programming its data is most appropriate. If a “Program Page” command was affected and the sector can not be erased the wordline could be invalidated if needed by marking it with all-one data and the data could be programmed to another empty wordline.

Only in case of a defective microcode SRAM the next program or erase operation will incur again this error.

Note: Although this error indicates a failed operation it is possible to ignore it and rely on a data verification step to determine if the Flash memory has correct data. Before re-programming the Flash the flow must ensure that a new reset is applied.

*Note: Even when the flag is ignored it is recommended to clear it. Otherwise all following operations — including “sleep” — could trigger an interrupt even when they are successful (see **Chapter 8.5.1**, interrupt because of operational error).*

8.5.3.3 PROER “Protection Error”

Fault conditions:

- Password failure.
- Erase/Write to protected sector.
- Erase UCB and protection active.
- Write UC-Page to protected UCB.

Attention: a protection violation can even occur when a protection was not explicitly installed by the user. This is the case when the Flash startup detects an error and starts the user software with read-only Flash (see **Chapter 8.5.3.6**). Trying to change the Flash memory will then cause a PROER.

New state:

Read mode is entered. The protection violating command is not executed.

Proposed handling by software:

Usually this bit is only set during runtime due to a bug in the software. In case of a password failure a reset must be performed in the other cases the flag can be cleared with “Clear Status” or “Reset to Read”. After that the corrected sequence can be executed.

8.5.3.4 VER “Verification Error”

Fault conditions:

This flag is a warning indication and not an error. It is set when a program or erase operation was completed but with a suboptimal result. This bit is already set when only a single bit is left over-erased or weakly programmed which would be corrected by the ECC anyhow.

However, excessive VER occurrence can be caused by operating the Flash out of the specified limits, e.g. incorrect voltage or temperature. A VER after programming can also be caused by programming a page whose sector was not erased correctly (e.g. aborted erase due to power failure).

Under correct operating conditions a VER after programming will practically not occur. A VER after erasing is not unusual.

Attention: this bit can also be set during startup (see **Chapter 8.5.3.6**).

New state:

No state change. Just the bit is set.

Proposed handling by software:

This bit can be ignored. It should be cleared with “Clear Status” or “Reset to Read”. In-spec operation of the Flash memory must be ensured.

If the application allows (timing and data logistics), a more elaborate procedure can be used to get rid of the VER situation:

- VER after program: erase the sector and program the data again. This is only recommended when there are more than 3 program VERs in the same sector. When programming the Flash in field ignoring program VER is normally the best solution because its most likely cause are violated operating conditions. Take care that never a sector is programmed in which the erase was aborted.
- VER after erase: the erase operation can be repeated until VER disappears. Repeating the erase more than 3 times consecutively for the same sector is not recommended. After that it is better to ignore the VER, program the data and check its readability. Again its most likely cause are violated operating conditions. Therefore it is recommended to repeat the erase at most once or ignore it altogether.

For optimizing the quality of Flash programming see the following section about handling single-bit ECC errors.

*Note: Even when this flag is ignored it is recommended to clear it. Otherwise all following operations — including “sleep” — could trigger an interrupt even when they are successful (see **Chapter 8.5.1**, interrupt because of verify error).*

8.5.3.5 PFSBER/DFSBER “Single-Bit Error”

Fault conditions:

When reading data or fetching code from PFLASH the ECC evaluation detected a single-bit error (“SBE”) which was corrected.

This flag is a warning indication and not an error. A certain amount of single-bit errors must be expected because of known physical effects.

New state:

No state change. Just the bit is set.

Proposed handling by software:

This flag can be used to analyze the state of the Flash memory. During normal operation it should be ignored. In order to count single-bit errors it must be cleared by “Clear Status” or “Reset to Read” after each occurrence[1].

Usually it is sufficient after programming data to compare the programmed data with its reference values ignoring the SBE bits. When there is a comparison error the sector is erased and programmed again.

When programming the PFLASH (end-of-line programming or SW updates) customers can further reduce the probability of future read errors by performing the following check after programming:

- Change the read margin to “high margin 0”.
- Verify the data and count the number of SBEs.
- When the number of SBEs exceeds a certain limit (e.g. 10 in 2 Mbyte) the affected sectors could be erased and programmed again.
- Repeat the check for “high margin 1”.
- Each sector should be reprogrammed at most once, afterwards SBEs can be ignored.

Due to the specificity of each application the appropriate usage and implementation of these measures (together with the more elaborate VER handling) must be chosen according to the context of the application.

8.5.3.6 Handling Flash Errors During Startup

During startup, a fatal error during Flash ramp up forces the Firmware to terminate the startup process and to end in the Debug Monitor Mode (see Firmware chapter).

The reason for a failed Flash startup can be a hardware error or damaged configuration data.

1) Further advice: remember that the ECC is evaluated when the data is read from the PMU. When counting single-bit errors use always the non-cached address range otherwise the error count can depend on cache hit or miss and it refers to the complete cache line. As the ECC covers a block of 64 data bits take care to evaluate the **FSR** only once per 64-bit block.

FSR bits set after startup are of informative warning nature.

FSR.PFOPER can indicate a problem of a program/erase operation before the last system reset or an error when restoring the Flash module internal SRAM content after the last reset. In both cases it is advised to clear the flag with the command sequence “Clear Status” and trigger a system reset. If the error shows up again it is an indication for a permanent fault which will limit the Flash operation to read accesses. Under this condition program and erase operations are forbidden (but not blocked by hardware!).

8.6 Power, Reset and Clock

The following chapters describe the required power supplies, the power consumption and its possible reduction, the control of Flash Sleep Mode and the basic control of Reset.

8.6.1 Power Supply

The Flash module uses the standard V_{DDP} I/O power supply to generate the voltages for both read and write functions. Internally generated and regulated voltages are provided for the program and erase operations as well as for read operations. The standard V_{DDC} is used for all digital control functions.

8.6.2 Power Reduction

The “Flash Sleep Mode” can be used to drastically reduce power consumption while the Flash is not accessed for longer periods of time.

The “Idle Read Path” slightly reduces the dynamic power consumption during normal operation with marginal impact on the Flash read performance.

Flash Sleep Mode

As power reduction feature, the Flash module provides the Flash Sleep mode which can be selected by the user individually for the Flash. The Sleep mode can be requested by:

- Programming 1_B to the bit **FCON**.SLEEP.
- “External” sleep mode by the SCU (see “Flash Power Control” in the SCU). Only executed by the Flash when **FCON**.ESLDIS = 0_B.

Attention: f_{CPU} must be equal or above 1 MHz when Sleep mode is requested until the Sleep mode is indicated in FSR.SLM, and when a wake-up request is triggered, until FSR.PBUSY is cleared.

The requested Sleep mode is only taken if the Flash is in idle state and when all pending or active requests are processed and terminated. Only then, the Flash array performs the ramp down into the Sleep mode: the sense amplifiers are switched off and the voltages are ramped down.

During ramp down to Sleep mode **FSR**.PBUSY is set.

As long as the Flash is in Sleep mode, this state is indicated by the bit **FSR**.SLM. The **FSR**.PBUSY stays set as well.

Wake-up from sleep is controlled with clearing of bit **FCON**.SLEEP, if selected via this bit, or wake-up is initiated by releasing the “external” sleep signal from SCU. After wake-up, the Flash enters read mode and is available again after the wake-up time t_{WU}. During the wake-up phase the **FSR**.PBUSY is set until the wake-up process is completed.

Note: During sleep and wake-up, the Flash is reported to be busy. Thus, read and write accesses to the Flash in Sleep mode are acknowledged with busy' and should therefore be avoided; those accesses make sense only during wake-up, when waiting for the Flash read mode.

2. *The wake-up time t_{WU} is documented in the Data Sheet. This time may fully delay the interrupt response time in Sleep mode.*
3. *Note: A wake-up is only accepted by the Flash if it is in Sleep mode. The Flash will first complete the ramp down to Sleep mode before reacting to a wake-up trigger.*

Idle Read Path

An additional power saving feature is enabled by setting **FCON**.IDLE. In this case the PFLASH read path (**Flash Read Access**) is switched off when no read access is pending. System performance for sequential accesses is slightly reduced because internal linear prefetches of the PFLASH are disabled. Non-sequential read accesses requested by the CPU or any other bus master see no additional delayed.

8.6.3 Reset Control

All PMU and Flash functionality is reset with the system reset with the exception of the register bits: **FSR**.PROG, **FSR**.ERASE, **FSR**.PFOPER. These bits are reset with the power-on reset.

The flash will be automatically reset to the read mode after every reset.

8.6.3.1 Resets During Flash Operation

A reset or power failure during an ongoing Flash operation (i.e. program or erase) must be considered as violation of stable operating conditions. However the Flash was designed to prevent damage to non-addressed Flash ranges when the reset is applied as defined in the data sheet. The exceptions are erasing logical sectors and UCBs. Aborting an erase process of a logical sector can leave the complete physical sector unreadable.When an UCB erase is aborted the complete Flash can become unusable. So UCBs must be only erased in a controlled environment. The addressed Flash range is left in an undefined state.

When an erase operation is aborted the addressed logical or physical sector can contain any data. It can even be in a state that doesn’t allow this range to be programmed.

When a page programming operation is aborted the page can still appear as erased (but contain slightly programmed bits), it can appear as being correctly programmed (but the data has a lowered retention) or the page contains garbage data. It is also possible that the read data is instable so that depending on the operating conditions different data is read.

For the detection of an aborted Flash process the flags **FSR**.PROG and **FSR**.ERASE could be used as indicator but only when the reset was a System Reset. Power-on resets can not be determined from any flags. It is not possible to detect an aborted operation simply by reading the Flash range. Even the margin reads don't offer a reliable indication.

When erasing or programming the PFLASH usually an external instance can notice the reset and simply restart the operation by erasing the Flash range and programming it again.

However, in cases where this external instance is not existing, a common solution is detecting an abort by performing two operations in sequence and determine after reset from the correctness of the second the completeness of the first operation.

E.g. after erasing a sector a page is programmed. After reset the existence of this page proves that the erase process was performed completely.

The detection of aborted programming processes can be handled similarly. After programming a block of data an additional page is programmed as marker. When after reset the block of data is readable and the marker is existent it is ensured that the block of data was programmed without interruption.

If a complete page can be spent as marker, the following recipe allows to reduce the marker size to 8 bytes. This recipe violates the rule that a page may be programmed only once. This violation is only allowed for this purpose and only when the algorithm is robust against disturbed pages (see also recommendations for handling single-bit errors) by repeating a programming step when it detects a failure.

Robust programming of a page of data with an 8 byte marker:

1. After reset program preferably always first to an even page ("Target Page").
2. If the Other Page on the same wordline contains active data save it to SRAM (the page can become disturbed because of the 4 programming operations per wordline).
3. Program the data to the Target Page.
4. Perform strict check of the Target Page (see below).
5. Program 8 byte marker to Target Page.
6. Perform strict check of the Target Page.
7. In case of any error of the strict check go to the next wordline and program the saved data and the target data again following the same steps.
8. Ensure that the algorithm doesn't repeat unlimited in case of a violation of operating conditions.

Strict checking of programmed data:

1. Ignore single-bit errors and the VER flag.
2. Switch to tight margin 0.
3. If the data (check the complete page) is not equal to the expected data report an error.
4. If a double-bit error is detected report an error.

After reset the algorithm has to check the last programmed page if it was programmed completely:

1. Read with normal read level. Ignore single-bit errors.
2. Read 8-byte marker and check for double-bit error.
3. Read data part and verify its consistency (e.g. by evaluating a CRC). Check for double-bit error.
4. If the data part is defective don't use it (e.g. by invalidating the page).
5. If the data part is ok:
   a) If the marker is erased the data part could have been programmed incompletely. Therefore the data part should not be used or alternatively it could be programmed again to a following page.
   b) If the marker contains incorrect data the data part was most likely programmed correctly but the marker was programmed incompletely. The page could be used as is or alternatively the data could be programmed again to a following page.
   c) If the marker is ok the data part was programmed completely and has the full retention. However this is not ensured for the marker part itself. Therefore the algorithm must be robust against the case that the marker becomes unreadable later.

8.6.4 Clock

The Flash interface is operating at the same clock speed as the CPU, f_{CPU}. Depending on the frequency, wait states must be inserted in the Flash accesses. Further details onthe wait states configuration are give in **Chapter 8.4.4**.

For proper operation of command sequences and when entering or waking up from Sleep mode, f_{CPU} must be equal or above 1 MHz.

8.7 Registers

The register set consists of the PMU ID register (**Chapter 8.7.1**), the Prefetch Control register (**Chapter 8.7.2**). The other registers control Flash functionality (**Chapter 8.7.3**).

All accesses prevented due to access mode restrictions fail with a bus error.

Also accesses to unoccupied register addresses fail with a bus error.

8.7.1 PMU Registers

The PMU only contains the ID register.

Table 8-6 Registers Address Space

Module	Base Address	End Address	Note
reserved	5800 0000$_H$	5800 04FF$_H$	Bus Error
PMU0	5800 0500$_H$	5800 05FF$_H$	
reserved	5800 0600$_H$	5800 0FFF$_H$	Bus Error
reserved	5800 2400$_H$	5800 3FFF$_H$	Bus Error

Table 8-7 Registers Overview

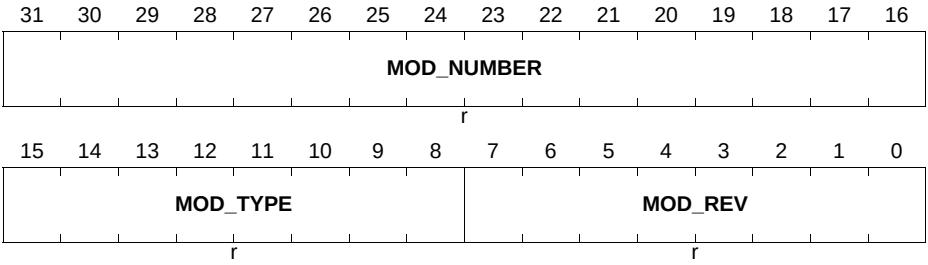
Short Name	Description	Offset Addr.[1]	Access Mode		Reset Class	Page Number
			Read	Write		
ID	Module Identification	08$_H$	U, PV	BE	System Reset	8-31

1) The absolute register address is calculated as follows:
Module Base Address (**Table 8-6**) + Offset Address (shown in this column)

8.7.1.1 PMU ID Register

The PMU0_ID register is a read-only register, thus write accesses lead to a bus error trap. Read accesses are permitted in Privileged Mode PV and in User Mode. The PMU0_ID register is defined as follows:

PMU0_ID
PMU0 Identification Register (5800 0508$_H$) Reset Value: 00A1 C0XX$_H$



Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** MOD_REV defines the module revision number. The value of a module revision starts with 01_H (first rev.).
MOD_TYPE	[15:8]	r	**Module Type** This bit field is $C0_H$. It defines the module as a 32-bit module.
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines the module identification number for PMU0.

8.7.2 Prefetch Registers

This section describes the register of the Prefetch unit.

Table 8-8 Registers Address Space

Module	Base Address	End Address	Note
PREF	5800 4000_H	5800 $7FFF_H$	Prefetch Module Registers

Table 8-9 Registers Overview

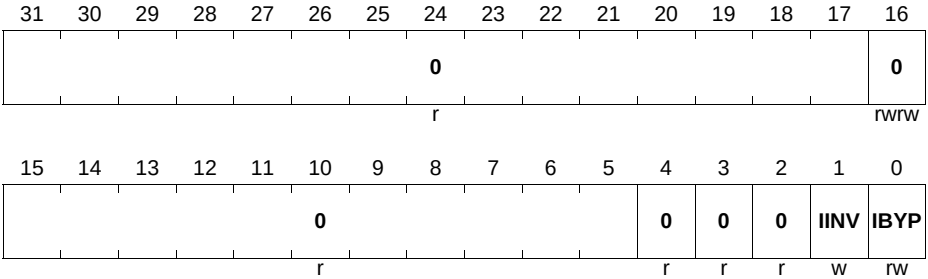
Short Name	Description	Offset Addr.[1)]	Access Mode		Reset Class	Page Number
			Read	Write		
PCON	Prefetch Configuration Register	0_H	U, PV	U, PV	System Reset	Page 8-33

1) The absolute register address is calculated as follows:
Module Base Address (**Table 8-6**) + Offset Address (shown in this column)

8.7.2.1 Prefetch Configuration Register

This register provides control bits for instruction buffer invalidation and bypass.

PREF_PCON
Prefetch Configuration Register (5800 4000$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
IBYP	0	rw	**Instruction Prefetch Buffer Bypass** 0_B Instruction prefetch buffer not bypassed. 1_B Instruction prefetch buffer bypassed.
IINV	1	w	**Instruction Prefetch Buffer Invalidate** Write Operation: 0_B No effect. 1_B Initiate invalidation of entire instruction cache.
0	16	rw	**Reserved** Must be written with 0.
0	[15:5], 4, 3, 2, [31:17]	r	**Reserved** returns 0 if read; should be written with 0.

8.7.3 Flash Registers

All register addresses are word aligned, independently of the register width. Besides word-read/write accesses, also byte or half-word read/write accesses are supported.

The absolute address of a Flash register is calculated by the base address from **Table 8-10** plus the offset address of this register from **Table 8-11**.

Table 8-10 Registers Address Space

Module	Base Address	End Address	Note
FLASH0	5800 1000$_H$	5800 23FF$_H$	Flash registers of PMU0

The following table shows the addresses, the access modes and reset types for the Flash registers in PMU0:

Table 8-11 Addresses of Flash0 Registers

Short Name	Description	Address	Access Mode		Reset	See
			Read	Write		
–	Reserved	5800 2000$_H$ – 5800 2004$_H$	BE	BE	–	–
FLASH0_ ID	Flash Module Identification Register	5800 2008$_H$	U, PV	BE	System Reset	Page 8-44
–	Reserved	5800 200C$_H$	BE	BE	–	–
FLASH0_ FSR	Flash Status Register	5800 2010$_H$	U, PV	BE	System + PORST	Page 8-35
FLASH0_ FCON	Flash Configuration Register	5800 2014$_H$	U, PV	PV	System Reset	Page 8-41
FLASH0_ MARP	Flash Margin Control Register PFLASH	5800 2018$_H$	U, PV	PV	System Reset	Page 8-45
FLASH0_ PROCON0	Flash Protection Configuration User 0	5800 2020$_H$	U, PV	BE	System Reset	Page 8-46
FLASH0_ PROCON1	Flash Protection Configuration User 1	5800 2024$_H$	U, PV	BE	System Reset	Page 8-47
FLASH0_ PROCON2	Flash Protection Configuration User 2	5800 2028$_H$	U, PV	BE	System Reset	Page 8-48
–	Reserved	5800 202C$_H$ – 5800 23FC$_H$	BE	BE	–	

8.7.3.1 Flash Status Definition

The Flash Status Register FSR reflects the overall status of the Flash module after Reset and after reception of the different commands. Sector specific protection states are not indicated in the FSR, but in the registers PROCON0, PROCON1 and PROCON2. The status register is a read-only register. Only the error flags and the two status flags (PROG, ERASE) are affected with the “Clear Status” command. The error flags are also cleared with the “Reset to Read” command.

The FSR is defined as follows:

FSR
Flash Status Register **(1010_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
VER	**X**	**0**	**SLM**	**0**	**WPRODIS1**	**WPRODIS0**	**0**	**WPROIN2**	**WPROIN1**	**WPROIN0**	**0**	**RPRODIS**	**RPROIN**	**0**	**PROIN**
rh	rh	r	rh	r	rh	rh	r	rh	rh	rh	r	rh	rh	r	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**PFDBER**	**0**	**PFSBER**	**PROER**	**SQER**	**0**	**PFOPER**	**0**	**PFPAGE**	**ERASE**	**PROG**	**0**	**0**	**FABUSY**	**PBUSY**
r	rh	r	rh	rh	rh	r	rh	r	rh	rh	rh	r	r	rh	rh

Field	Bits	Type	Description
PBUSY[1)]	0	rh	**Program Flash Busy** HW-controlled status flag. 0_B PFLASH ready, not busy; PFLASH in read mode. 1_B PFLASH busy; PFLASH not in read mode. Indication of busy state of PFLASH because of active execution of program or erase operation; PFLASH busy state is also indicated during Flash recovery (after reset) and in power ramp-up state or in sleep mode; while in busy state, the PFLASH is not in read mode.
FABUSY[1)]	1	rh	**Flash Array Busy** Internal busy flag for testing purposes. Must be ignored by application software, which must use PBUSY instead.

Field	Bits	Type	Description
PROG[3)4)]	4	rh	**Programming State** HW-controlled status flag. 0_B There is no program operation requested or in progress or just finished. 1_B Programming operation (write page) requested (from FIM) or in action or finished. Set with last cycle of Write Page command sequence, cleared with Clear Status command (if not busy) or with power-on reset. If one BUSY flag is coincidently set, PROG indicates the type of busy state. If xOPER is coincidently set, PROG indicates the type of erroneous operation. Otherwise, PROG indicates, that operation is still requested or finished.
ERASE[3)4)]	5	rh	**Erase State** HW-controlled status flag. 0_B There is no erase operation requested or in progress or just finished 1_B Erase operation requested (from FIM) or in action or finished. Set with last cycle of Erase command sequence, cleared with Clear Status command (if not busy) or with power-on reset. Indications are analogous to PROG flag.
PFPAGE[1)2)]	6	rh	**Program Flash in Page Mode** HW-controlled status flag. 0_B Program Flash not in page mode 1_B Program Flash in page mode; assembly buffer of PFLASH (256 byte) is in use (being filled up) Set with Enter Page Mode for PFLASH, cleared with Write Page command *Note: Concurrent page and read modes are allowed*
PFOPER[2)3)4)]	8	rh	**Program Flash Operation Error** 0_B No operation error reported by Program Flash 1_B Flash array operation aborted, because of a Flash array failure, e.g. an ECC error in microcode. This bit is not cleared with System Reset, but with power-on reset. Registered status bit; must be cleared per command

Field	Bits	Type	Description
SQER[1)2)3)]	10	rh	**Command Sequence Error** 0_B No sequence error 1_B Command state machine operation unsuccessful because of improper address or command sequence. A sequence error is not indicated if the Reset to Read command aborts a command sequence. Registered status bit; must be cleared per command
PROER[1)2)3)]	11	rh	**Protection Error** 0_B No protection error 1_B Protection error. A Protection Error is reported e.g. because of a not allowed command, for example an Erase or Write Page command addressing a locked sector, or because of wrong password(s) in a protected command sequence such as “Disable Read Protection” Registered status bit; must be cleared per command
PFSBER[1)2)3)]	12	rh	**PFLASH Single-Bit Error and Correction** 0_B No Single-Bit Error detected during read access to PFLASH 1_B Single-Bit Error detected and corrected Registered status bit; must be cleared per command
PFDBER[1)2)3)]	14	rh	**PFLASH Double-Bit Error** 0_B No Double-Bit Error detected during read access to PFLASH 1_B Double-Bit Error detected in PFLASH Registered status bit; must be cleared per command
PROIN	16	rh	**Protection Installed** 0_B No protection is installed 1_B Read or/and write protection for one or more users is configured and correctly confirmed in the User Configuration Block(s). HW-controlled status flag

Field	Bits	Type	Description
RPROIN	18	rh	**Read Protection Installed** 0_B No read protection installed 1_B Read protection and global write protection is configured and correctly confirmed in the User Configuration Block 0. Supported only for the master user (user zero). HW-controlled status flag
RPRODIS[1)5)]	19	rh	**Read Protection Disable State** 0_B Read protection (if installed) is not disabled 1_B Read and global write protection is temporarily disabled. Flash read with instructions from other memory, as well as program or erase on not separately write protected sectors is possible. HW-controlled status flag
WPROIN0	21	rh	**Sector Write Protection Installed for User 0** 0_B No write protection installed for user 0 1_B Sector write protection for user 0 is configured and correctly confirmed in the User Configuration Block 0. HW-controlled status flag
WPROIN1	22	rh	**Sector Write Protection Installed for User 1** 0_B No write protection installed for user 1 1_B Sector write protection for user 1 is configured and correctly confirmed in the User Configuration Block 1. HW-controlled status flag
WPROIN2	23	rh	**Sector OTP Protection Installed for User 2** 0_B No OTP write protection installed for user 2 1_B Sector OTP write protection with ROM functionality is configured and correctly confirmed in the UCB2. The protection is locked for ever. HW-controlled status flag

Field	Bits	Type	Description
WPRODIS0[1)5)]	25	rh	**Sector Write Protection Disabled for User 0** 0_B All protected sectors of user 0 are locked if write protection is installed 1_B All write-protected sectors of user 0 are temporarily unlocked, if not coincidently locked by user 2 or via read protection. Hierarchical protection control: User-0 sectors are also unlocked, if coincidently protected by user 1. But not vice versa. HW-controlled status flag
WPRODIS1[1)5)]	26	rh	**Sector Write Protection Disabled for User 1** 0_B All protected sectors of user 1 are locked if write protection is installed 1_B All write-protected sectors of user 1 are temporarily unlocked, if not coincidently locked by user 0 or user 2 or via read protection. HW-controlled status flag
SLM[1)]	28	rh	**Flash Sleep Mode** HW-controlled status flag. Indication of Flash sleep mode taken because of global or individual sleep request; additionally indicates when the Flash is in shut down mode. 0_B Flash not in sleep mode 1_B Flash is in sleep or shut down mode
X	30	rh	**Reserved** Value undefined
VER[1)3)]	31	rh	**Verify Error** 0_B The page is correctly programmed or the sector correctly erased. All programmed or erased bits have full expected quality. 1_B A program verify error or an erase verify error has been detected. Full quality (retention time) of all programmed (“1”) or erased (“0”) bits cannot be guaranteed. See **Chapter 8.5.3** and **Chapter 8.5.3.6** for proper reaction. Registered status bit; must be cleared per command

Field	Bits	Type	Description
0	2,3,7, 9,13, 15,17, 20,24, 27, 29	r	**Reserved** Read zero, no write

Note: The footnote numbers of FSR bits describe the specific reset conditions:
1)Cleared with System Reset
2)Cleared with command “Reset to Read”
3)Cleared with command “Clear Status”
4)Cleared with power-on reset (PORST)
5)Cleared with command “Resume Protection”

Note: The xBUSY flags as well as the protection flags cannot be cleared with the “Clear Status” command or with the “Reset to Read” command. These flags are controlled by HW.

*Note: The reset value above is indicated after correct execution of Flash ramp up. Additionally, errors are possible after ramp up (see **Chapter 8.5.3.6**).*

8.7.3.2 Flash Configuration Control

The Flash Configuration Register FCON reflects and controls the following general Flash configuration functions:

- Number of wait states for Flash accesses.
- Indication of installed and active read protection.
- Instruction and data access control for read protection.
- Interrupt mask bits.
- Power reduction and shut down control.

FCON is a Privileged Mode protected register. It is defined as follows:

FCON
Flash Configuration Register (1014$_H$) Reset value: 0007 0006$_H$[1)]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EOBM	**0**	**PFDBERM**	**0**	**PFSBERM**	**PROERM**	**SQERM**	**VOPERM**	**0**	**0**	**0**	**0**	**0**	**DDF**	**DCF**	**RPA**
rw	r	rw	r	rw	rw	rw	rw	r	r	r	r	r	rwh	rwh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SLEEP	**ESLDIS**	**IDLE**	**0**								**WSECPF**	**WSPFLASH**			
rw	rw	rw	r			r			r		rw	rw			

1) After Flash ramp up and execution of the startup software in BROM (after firmware exit), the initial value is 000X 0006H.

Field	Bits	Type	Description
WSPFLASH	[3:0]	rw	**Wait States for read access to PFLASH** This bit field defines the number of wait states n, which are used for an initial read access to the Program Flash memory area, with WSPFLASH x (1 / f_{CPU}) ≥ t_a[1)]. 0000_B PFLASH access in one clock cycle 0001_B PFLASH access in one clock cycle 0010_B PFLASH access in two clock cycles 0011_B PFLASH access in three clock cycles .... PFLASH access in four up to fourteen clock cycles. 1111_B PFLASH access in fifteen clock cycles.
WSECPF	4	rw	**Wait State for Error Correction of PFLASH** 0_B No additional wait state for error correction 1_B One additional wait state for error correction during read access to Program Flash. If enabled, this wait state is only used for the first transfer of a burst transfer. Set this bit only when requested by Infineon.
IDLE	13	rw	**Dynamic Flash Idle** 0_B Normal/standard Flash read operation 1_B Dynamic idle of Program Flash enabled for power saving; static prefetching disabled

Field	Bits	Type	Description
ESLDIS	14	rw	**External Sleep Request Disable** 0_B External sleep request signal input is enabled 1_B Externally requested Flash sleep is disabled The ‘external’ signal input is connected with a global power-down/sleep request signal from SCU.
SLEEP	15	rw	**Flash SLEEP** 0_B Normal state or wake-up 1_B Flash sleep mode is requested Wake-up from sleep is started with clearing of the SLEEP-bit.
RPA	16	rh	**Read Protection Activated** This bit monitors the status of the Flash-internal read protection. This bit can only be ‘0’ when read protection is not installed or while the read protection is temporarily disabled with password sequence. 0_B The Flash-internal read protection is not activated. Bits DCF, DDF are not taken into account. Bits DCF, DDFx can be cleared 1_B The Flash-internal read protection is activated. Bits DCF, DDF are enabled and evaluated.
DCF	17	rwh	**Disable Code Fetch from Flash Memory** This bit enables/disables the code fetch from the internal Flash memory area. Once set, this bit can only be cleared when RPA=’0’. This bit is automatically set with reset and is cleared during ramp up, if no RP installed, and during startup (BROM) in case of internal start out of Flash. 0_B Code fetching from the Flash memory area is allowed. 1_B Code fetching from the Flash memory area is not allowed. This bit is not taken into account while RPA=’0’.

Field	Bits	Type	Description
DDF	18	rwh	**Disable Any Data Fetch from Flash** This bit enables/disables the data read access to the Flash memory area (Program Flash and Data Flash). Once set, this bit can only be cleared when RPA='0'. This bit is automatically set with reset and is cleared during ramp up, if no RP installed, and during startup (BROM) in case of internal start out of Flash. 0_B Data read access to the Flash memory area is allowed. 1_B Data read access to the Flash memory area is not allowed. This bit is not taken into account while RPA='0'.
VOPERM	24	rw	**Verify and Operation Error Interrupt Mask** 0_B Interrupt not enabled 1_B Flash interrupt because of Verify Error or Operation Error in Flash array (FSI) is enabled
SQERM	25	rw	**Command Sequence Error Interrupt Mask** 0_B Interrupt not enabled 1_B Flash interrupt because of Sequence Error is enabled
PROERM	26	rw	**Protection Error Interrupt Mask** 0_B Interrupt not enabled 1_B Flash interrupt because of Protection Error is enabled
PFSBERM	27	rw	**PFLASH Single-Bit Error Interrupt Mask** 0_B No Single-Bit Error interrupt enabled 1_B Single-Bit Error interrupt enabled for PFLASH
PFDBERM	29	rw	**PFLASH Double-Bit Error Interrupt Mask** 0_B Double-Bit Error interrupt for PFLASH not enabled 1_B Double-Bit Error interrupt for PFLASH enabled. Especially intended for margin check
EOBM	31	rw	**End of Busy Interrupt Mask** 0_B Interrupt not enabled 1_B EOB interrupt is enabled
0	[12:5], [23:19], 28, 30	r	**Reserved** Always read/write zero

1) WSPFLASH = 0_H deviates from this formula and results in the same timing as WSPFLASH = 1_H.

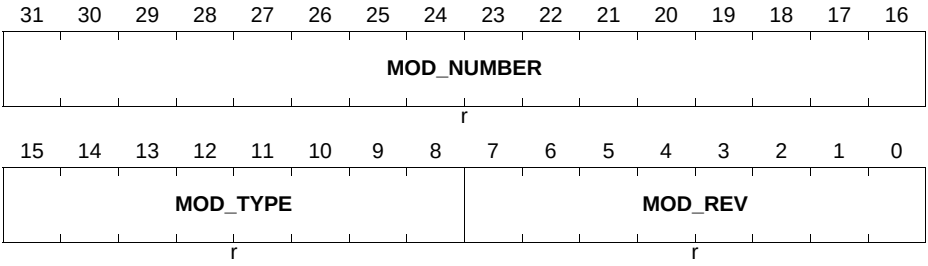
Note: The default numbers of wait states represent the slow cases. This is a general proceeding and additionally opens the possibility to execute higher frequencies without changing the configuration.

Note: After reset and execution of Firmware, the read protection control bits are coded as follows:
DDF, DCF, RPA = "110": No read protection installed
DDF, DCF, RPA = "001": Read protection installed; start in internal Flash
DDF, DCF, RPA = "111": Read protection installed; start not in internal Flash.

8.7.3.3 Flash Identification Register

The module identification register of Flash module is directly accessible by the CPU via PMU access. This register is mapped into the space of the Flash Interface Module's registers (see **Table 8-11**).

FLASH0_ID
Flash Module Identification Register (1008_H) **Reset Value: 00A2 C0XX$_H$**

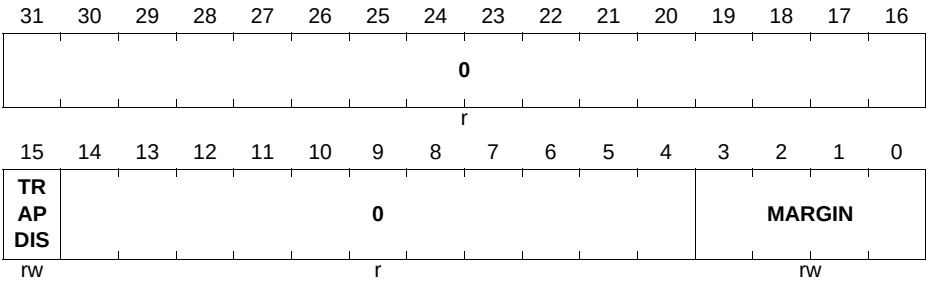


Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** MOD_REV defines the module revision number. The value of a module revision starts with 01_H (first revision).
MOD_TYPE	[15:8]	r	**Module Type** This bit field is $C0_H$. It defines the module as a 32-bit module.

Field	Bits	Type	Description
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines a module identification number. For the XMC4500 Flash0 this number is $00A2_H$.

8.7.3.4 Margin Check Control Register

MARP
Margin Control Register PFLASH (1018_H) **Reset Value: $0000\ 8000_H$**



Field	Bits	Type	Description
MARGIN	[3:0]	rw	**PFLASH Margin Selection** 0000_B **Default**, Standard (default) margin. 0001_B **Tight0**, Tight margin for 0 (low) level. Suboptimal 0-bits are read as 1s. 0100_B **Tight1**, Tight margin for 1 (high) level. Suboptimal 1-bits are read as 0s. – Reserved.
TRAPDIS	15	rw	**PFLASH Double-Bit Error Trap Disable** 0_B If a double-bit error occurs in PFLASH, a bus error trap is generated[1]. 1_B The double-bit error trap is disabled. Shall be used only during margin check
0	[14:4], [31:16]	r	**Reserved** Always read as 0; should be written with 0.

1) After Boot ROM exit, double-bit error traps are enabled (TRAPDIS = 0).

8.7.3.5 Protection Configuration Indication

The configuration of read/write/OTP protection is indicated with registers PROCON0, PROCON1 and PROCON2, thus separately for every user, and it is generally indicated in the status register FSR.

If write protection is installed for user 0 or 1 or OTP protection for user 2, for each sector of the Program Flash it is indicated in the user-specific Protection Configuration register PROCONx, if it is locked or unlocked for program or erase operations.

The Flash Protection Configuration registers PROCONx are loaded by the FIM state machine out of the user's configuration block directly after reset during ramp up. The three PROCONx registers are read-only registers.

PROCON0
Flash Protection Configuration Register User 0
(1020_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R PRO	**0**	**0**	**0**	**0**	**S10_ S11L**	**S9L**	**S8L**	**S7L**	**S6L**	**S5L**	**S4L**	**S3L**	**S2L**	**S1L**	**S0L**
rh	r	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
SnL (n=0-9)	n	rh	**Sector n Locked for Write Protection by User 0** These bits indicate whether PFLASH sector n is write-protected by user 0 or not. 0_B No write protection is configured for sector n. 1_B Write protection is configured for sector n.

Field	Bits	Type	Description
S10_S11L	10	rh	**Sectors 10 and 11 Locked for Write Protection by User 0** This bit is only used if PFLASH has more than 0.5 Mbyte. It indicates whether PFLASH sectors 10+11 (together 512 KB) are write-protected by user 0 or not. 0_B No write protection is configured for sectors 10+11. 1_B Write protection is configured for sectors 10+11.
RPRO	15	rh	**Read Protection Configuration** This bit indicates whether read protection is configured for PFLASH by user 0. 0_B No read protection configured 1_B Read protection and global write protection is configured by user 0 (master user)
0	13, 12, 11	rh	**Reserved** deliver the corresponding UCB0 entry. Shall be configured to 0.
0	[31:16], 14	r	**Reserved** Always reads as 0.

PROCON1
Flash Protection Configuration Register User 1

(1024_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		**0**	**0**	**0**	**S10_ S11L**	**S9L**	**S8L**	**S7L**	**S6L**	**S5L**	**S4L**	**S3L**	**S2L**	**S1L**	**S0L**
r		rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
SnL (n=0-9)	n	rh	**Sector n Locked for Write Protection by User 1** These bits indicate whether PFLASH sector n is write-protected by user 1 or not. 0_B No write protection is configured for sector n. 1_B Write protection is configured for sector n.
S10_S11L	10	rh	**Sectors 10 and 11 Locked for Write Protection by User 1** This bit is only used if PFLASH has more than 0.5 Mbyte. It indicates whether PFLASH sectors 10+11 (together 512 KB) are write-protected by user 1 or not. 0_B No write protection is configured for sectors 10+11. 1_B Write protection is configured for sectors 10+11.
0	13, 12, 11	rh	**Reserved** Deliver the corresponding UCB1 entry. Shall be configured to 0.
0	[31:16], 15, 14	r	**Reserved** Always reads as 0.

PROCON2
Flash Protection Configuration Register User 2

(1028_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**S10_S11 ROM**	**S9 ROM**	**S8 ROM**	**S7 ROM**	**S6 ROM**	**S5 ROM**	**S4 ROM**	**S3 ROM**	**S2 ROM**	**S1 ROM**	**S0 ROM**
r	r	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
SnROM (n=0-9)	n	rh	**Sector n Locked Forever by User 2** These bits indicate whether PFLASH sector n is an OTP protected sector with read-only functionality, thus if it is locked for ever. 0_B No ROM functionality configured for sector n. 1_B ROM functionality is configured for sector n. Re-programming of this sector is no longer possible.
S10_S11ROM	10	rh	**Sectors 10 and 11 Locked Forever by User 2** This bit is only used if PFLASH has more than 0.5 Mbyte. It indicates whether PFLASH sectors 10+11 (together 512 KB) are read-only sectors or not. 0_B No ROM functionality is configured for sectors 10+11. 1_B ROM functionality is configured for sectors 10+11.
0	13, 12, 11	r	**Reserved** Deliver the corresponding UCB2 entry. Shall be configured to 0.
0	[31:16], 15, 14	r	**Reserved** Always reads as 0.

System Control

9 Window Watchdog Timer (WDT)

Purpose of the Window Watchdog Timer module is an improvement of the system integrity. The WDT triggers the system reset or other corrective actions like e.g. non-maskable interrupt if the main program, due to some fault condition, neglects to regularly service the watchdog (also referred to as “kicking the dog”, “petting the dog”, “feeding the watchdog” or “waking the watchdog”). The intention is to bring the system back from unresponsive state into normal operation.

References

[9] Cortex-M4 User Guide, ARM DUI 0508B (ID062910)

9.1 Overview

A successful servicing of the WDT results in a pulse on the signal wdt_service. The signal is offered also as an alternate function output. It can be used to show to an external watchdog that the system is alive.

The WDT timer is a 32-bit counter, which counts up from 0_H. It can be serviced while the counter value is within the window boundary, i.e. between the lower and the upper boundary value. Correct servicing results in a reset of the counter to 0_H. A so called “Bad Service” attempt results in a system reset request.

The timer block is running on the f_{WDT} clock which is independent from the bus clock. The timer value is updated in the corresponding register **TIM**, whenever the timer value increments. This mechanism enables immediate response on a read access from the bus.

The WDT module provides a register interface for configuration. A write to writable registers is only allowed, when the access is in privileged mode. A write access in user mode results in a bus error response.

9.1.1 Features

The watchdog timer (WDT) is an independent window watchdog timer.

The features are:

- Triggers system reset when not serviced on time or serviced in a wrong way
- Servicing restricted to be within boundaries of a user definable refresh window
- Can run from an independent clock
- Provides service indication to an external pin
- Can be suspended in HALT mode
- Provides optional pre-warning alarm before reset

Table 9-1 Application Features

Feature	Purpose/Application
System reset upon Bad Servicing	Triggered to restore system stable operation and ensure system integrity
Servicing restricted to be within defined boundaries of refresh window	Allows to consider minimum and maximum software timing
Independent clocks	To ensure that WDT counts even in case of the system clock failure
Service indication on external pin	For dual-channel watchdog solution, additional external control of system integrity
Suspending in HALT mode	Enables safe debugging with productive code
Pre-warning alarm	Software recovery to allow corrective action via software recovery routine bringing system back from the unresponsive state into normal operation

9.1.2 Block Diagram

The WDT block diagram is shown in **Figure 9-1**.

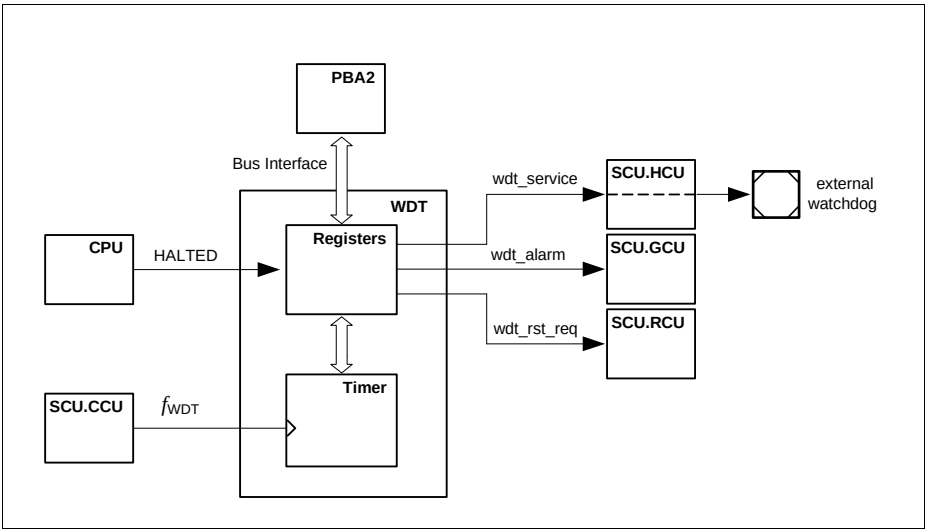


Figure 9-1 Watchdog Timer Block Diagram

9.2 Time-Out Mode

An overflow results in an immediate reset request going to the RCU of the SCU via the signal wdt_rst_req whenever the counter crosses the upper boundary it triggers an overflow event pre-warning is not enabled with **CTR** register. A successful servicing performed with writing a unique value, referred to as “Magic Word” to the **SRV** register of the WDT within the valid servicing window, results in a pulse on the signal wdt_service and reset of the timer counter.

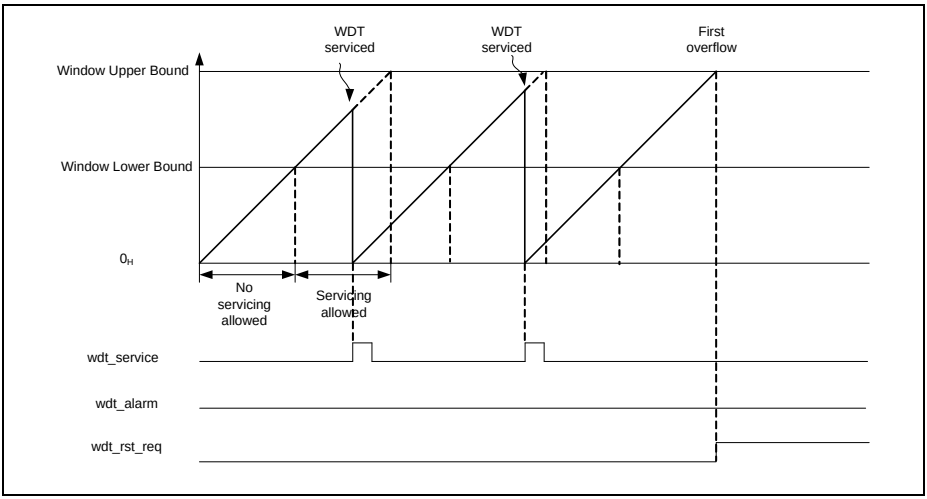


Figure 9-2 Reset without pre-warning

The example scenario depicted in **Figure 9-2** shows two consecutive service pulses generated from WDT module as the result of successful servicing within valid time windows. The situation where no service has been performed immediately triggers generation of reset request on the wdt_rst_req output after the counter value has exceeded window upper bound value.

9.3 Pre-warning Mode

While in prewarning mode the effect of the overflow event is different with and without pre-warning enabled. The first crossing of the upper bound triggers the outgoing alarm signal wdt_alarm when pre-warning is enabled. Only the next overflow results a reset request. The alarm status is shown via register **WDTSTS** and can be cleared via register **WDTCLR**. A clear of the alarm status will bring the WDT back to normal state. The alarm signal is routed as request to the SCU, where it can be promoted to NMI.

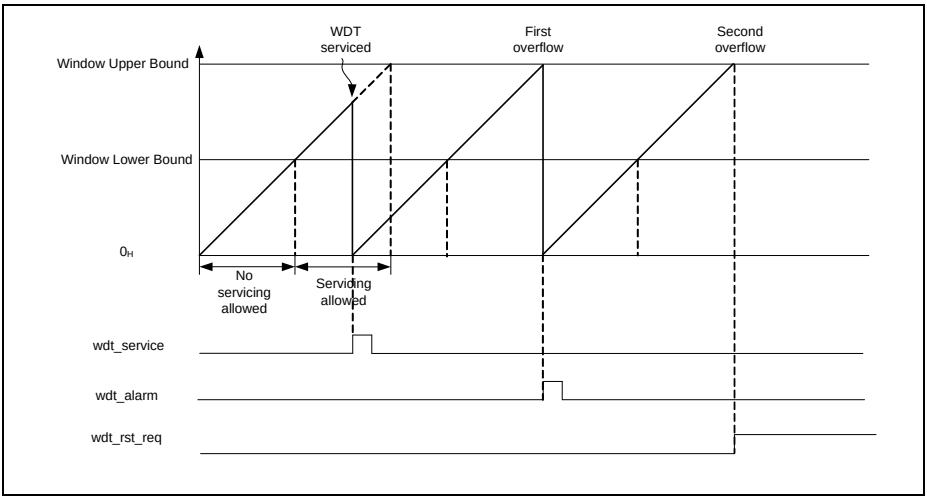


Figure 9-3 Reset after pre-warning

The example scenario depicted in **Figure 9-3** shows service pulse generated from WDT module as the result of successful servicing within valid time window. WDT generates alarm pulse on wdt_alarm upon first missing servicing. The alarm signal is routed as interrupt request to the SCU, where it can be promoted to NMI. Within this alarm service request the user can clear the WDT status bit and give a proper WDT service before it overflows next time. Otherwise WDT generates reset request on wdt_rstn upon the second missing service.

9.4 Bad Service Operation

A bad service attempt results in a reset request. A bad service attempt can be due to servicing outside the window boundaries or servicing with an invalid Magic Word.

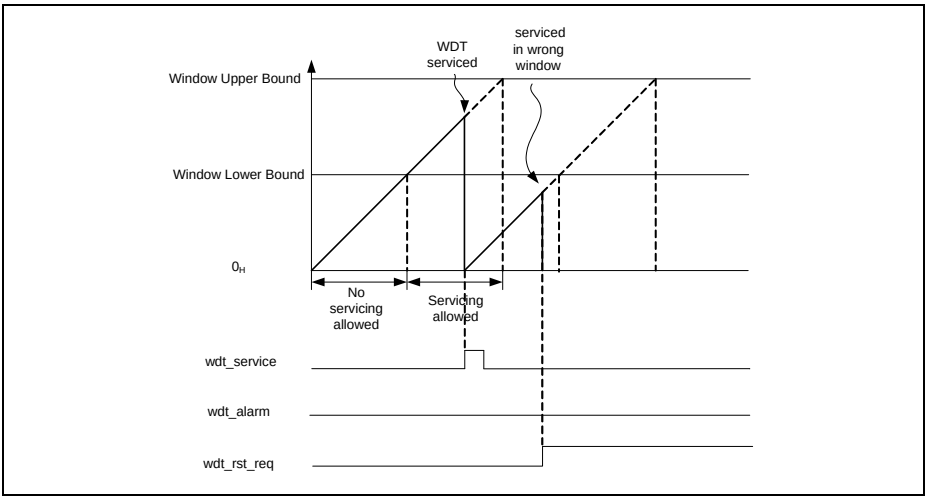


Figure 9-4 Reset upon servicing in a wrong window

The example in **Figure 9-4** shows servicing performed outside of valid servicing window. Attempt to service WDT while counter value remains below the Window Lower Bound results in immediate reset request on wdt_rst_req signal.

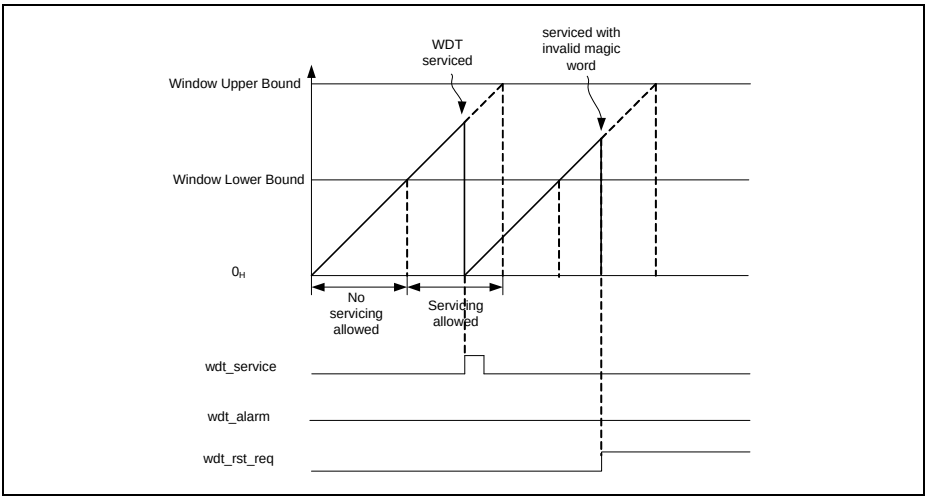


Figure 9-5 Reset upon servicing with a wrong magic word

The example in **Figure 9-5** shows servicing performed within a valid servicing window but with an invalid Magic Word. Attempt to write a wrong word to the **SRV** register results in immediate reset request on wdt_rst_req signal.

9.5 Service Request Processing

The WDT generates watchdog alarm service requests via wdt_alarm output signal upon first counter overflow over Watchdog Upper Bound when pre-warning mode is enabled. The alarm service request may be promoted by the SCU in two alternative modes:

- service request
- trap request causing NMI interrupt

Service requests can be disabled i SCU with service request mask or trap request disable registers respectively.

9.6 Debug Behavior

The WDT function can be suspended when the CPU enters HALT mode. WDT debug function is controlled by DSP bit field in **CTR** register.

9.7 Power, Reset and Clock

The WDT module is a part of the core domain and supplied with VDDC voltage.

All WDT registers get reset with the system reset.

A sticky bit in the RSSTAT register of SCU/RCU module indicates whether the last system reset has been triggered by the WDT module. This bit does not get reset with system reset.

The input clock of the WDT counter can be selected by the user between system PLL output, direct output of the internal system oscillator or 32kHz clock of hibernate domain, independently from the AHB interface clock. Selection of the WDT input clock is performed in SCU using WDTCLKCR register (for details please refer to the SCU/CCU chapter).

9.8 Initialization and Control Sequence

The programming model of the WDT module assumes several scenarios where different control sequences apply.

Note: Some of the scenarios described in this chapter require operations on system level the that are not in the scope of the WDT module description, therefore for detailed information please refer to relevant chapters of this document.

9.8.1 Initialization & Start of Operation

Complete WDT module initialization is required after system reset.

- check reason for last system reset in order to determine power state
  - read out SCU_RSTSTAT.RSTSTAT register bit field to determine last system reset cause
  - perform appropriate operations dependent on the last system reset cause
- WDT software initialization sequence
  - enable WDT clock with SCU_CLKSET.WDTCEN register bit field
  - release WDT reset with SCU_PRCLR2.WDTRS register bit field
  - set lower window bound with WDT_WLB register
  - set upper window bound with WDT_WUB register
  - configure external watchdog service indication (optional, please refer to SCU/HCU chapter)
  - select and enable WDT input clock with SCU_WDTCLKCR register
  - enable system trap for pre-warning alarm on system level with SCU_NMIREQEN register (optional, used in WDT pre-warning mode only)
- software start sequence
  - select mode (Time-Out or Pre-warning) and enable WDT module with WDT_CTR register
- service the watchdog
  - check current timer value in WDT_TIM register against programmed time window
  - write magic word to WDT_SRV register within valid time window

9.8.2 Reconfiguration & Restart of Operation

Reset and initialization of the WDT module is required in order to update its settings.

- software initialization sequence
  - assert WDT reset with SCU_PRSET2.WDTCEN register bit field
  - release WDT reset with SCU_PRCLR2.WDTRS register bit field register
  - set lower window bound with WDT_WLB register
  - set upper window bound with WDT_WUB register
  - configure external watchdog service indication (optional, please refer to SCU/HCU chapter)
  - select and enable WDT input clock (if change of the clock settings required) with SCU_WDTCLKCR register
  - enable system trap for pre-warning alarm on system level with SCU_NMIREQEN register (optional, used in WDT pre-warning mode only)
- software start sequence
  - select mode (Time-Out or Pre-warning) and enable WDT module with WDT_CTR register

- service the watchdog
  - check current timer value in WDT_TIM register against programmed time window
  - write magic word to WDT_SRV register within valid time window

9.8.3 Software Stop & Resume Operation

The WDT module can be stopped and re-started at any point of time for e.g. debug purpose using software sequence.

- software stop sequence
  - disable WDT module with WDT_CTR register
- perform any user operations
- software start (resume) sequence
  - enable WDT module with WDT_CTR register with WDT_CTR register
- service the watchdog
  - check current timer value in WDT_TIM register against programmed time window
  - write magic word to WDT_SRV register within valid time window

9.8.4 Enter Sleep/Deep Sleep & Resume Operation

The WDT counter clock can be configured to stop while in sleep or deep-sleep mode. No direct software interaction with the WDT is required in those modes and no watchdog time-out will fire if the WDT clock is configured to stop while CPU is sleeping.

- software configuration sequence for sleep/deep-sleep mode
  - configure WDT behavior with SCU register SLEEPCR or DSLEEPCR
- enter sleep/deep-sleep mode software sequence
  - select sleep or deep-sleep mode in CPU (for details please refer to Cortex-M4 documentation **[9]**)
  - enter selected mode (for details please refer to Cortex-M4 documentation **[9]**)
- wait for a wake-up event (no software interaction, CPU stopped)
- resume operation (CPU clock restarted automatically on an event)
- service the watchdog
  - check current timer value in WDT_TIM register against programmed time window
  - write magic word to WDT_SRV register within valid time window

9.8.5 Prewarning Alarm Handling

The WDT will fire prewarning alarm before requesting system reset while in pre-warning mode and not serviced within valid time window. The WDT status register indicating alarm must be cleared before the timer counter value crosses the upper bound for the second time after firing the alarm. After clearing of the alarm status regular watchdog servicing must be performed within valid time window.

- alarm event
  - exception routine (system trap or service request) clearing WDT_WDTSTAT register with WDT_WDTCLR register
- service the watchdog
  - check current timer value in WDT_TIM register against programmed time window
  - write magic word to WDT_SRV register within valid time window

9.9 Registers

Registers Overview

All these registers can be read in User Mode, but can only be written in Supervisor Mode.

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 9-2 Registers Address Space

Module	Base Address	End Address	Note
WDT	5000 8000$_H$	5000 BFFF$_H$	Watchdog Timer Registers

Table 9-3 Register Overview

Short Name	Register Long Name	Offset Addr.	Access Mode		Description
			Read	Write	
WDT Kernel Registers					
ID	Module ID Register	00$_H$	U, PV	PV	Page 9-11
CTR	Control Register	04$_H$	U, PV	PV	Page 9-12
SRV	Service Register	08$_H$	BE	PV	Page 9-13
TIM	Timer Register	0C$_H$	U, PV	BE	Page 9-14
WLB	Window Lower Bound	10$_H$	U, PV	PV	Page 9-14
WUB	Window Upper Bound	14$_H$	U, PV	PV	Page 9-14
WDTSTS	Watchdog Status Register	18$_H$	U, PV	PV	Page 9-15
WDTCLR	Watchdog Status Clear Register	1C$_H$	U, PV	PV	Page 9-16

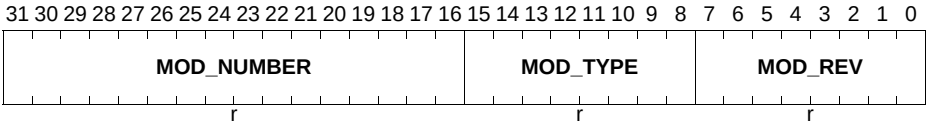
9.9.1 Registers Description

ID

The module ID register.

ID

WDT ID Register **(00_H)** **Reset Value: 00AD C0XX$_H$**



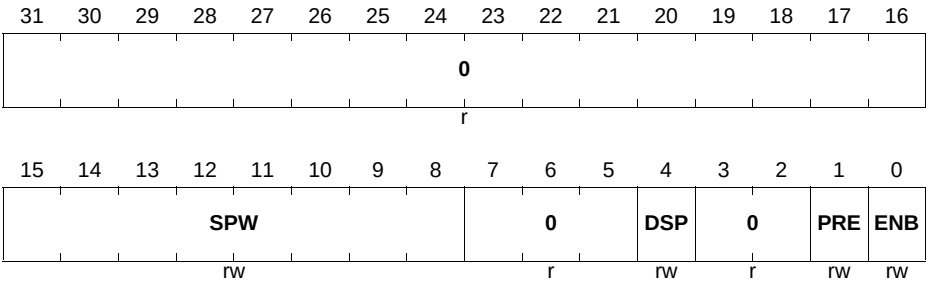
Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision** Indicates the revision number of the implementation. This information depends on the design step.
MOD_TYPE	[15:8]	r	**Module Type** This internal marker is fixed to $C0_H$.
MOD_NUMBER	[31:16]	r	**Module Number** Indicates the module identification number

CTR

The operation mode control register.

CTR

WDT Control Register **(04_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
ENB	0	rw	**Enable** 0_B disables watchdog timer, 1_B enables watchdog timer

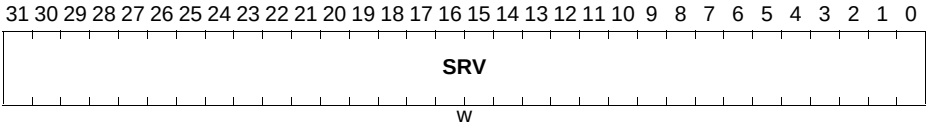
Field	Bits	Type	Description
PRE	1	rw	**Pre-warning** 0_B disables pre-warning 1_B enables pre-warning,
DSP	4	rw	**Debug Suspend** 0_B watchdog timer is stopped during halting mode debug, 1_B watchdog timer is not stopped during halting mode debug
SPW	[15:8]	rw	**Service Indication Pulse Width** Pulse width (SPW+1) of service indication in f_{WDT} cycles
0	[3:2], [7:5], [31:16]	r	**Reserved**

SRV

The WDT service register. Software must write a magic word while the timer value is within the valid window boundary. Writing the magic word while the timer value is within the window boundary will service the watchdog and result a reload of the timer with 0H.

Upon writing data different than the magic word within valid time window or writing even correct Magic Word but outside of the valid time window no servicing will be performed. Instead will request an immediate system reset request.

SRV
WDT Service Register **(08_H)** **Reset Value: 0000 0000_H**

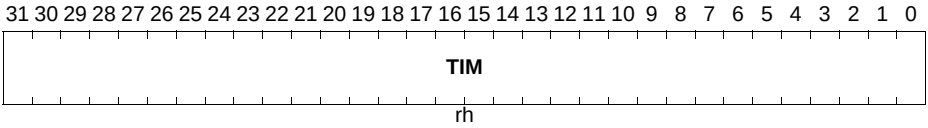


Field	Bits	Type	Description
SRV	[31:0]	w	**Service** Writing the magic word $ABADCAFE_H$ while the timer value is within the window boundary will service the watchdog.

TIM

The actual watchdog timer register count value. This register can be read by software in order to determine current position in the WDT time window.

TIM
WDT Timer Register (0C$_H$) Reset Value: 0000 0000$_H$

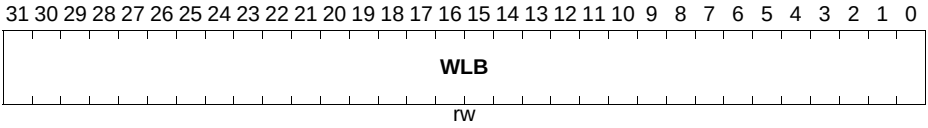


Field	Bits	Type	Description
TIM	[31:0]	rh	**Timer Value** Actual value of watchdog timer value.

WLB

The Window Lower Bound register defines the lower bound for servicing window. Servicing of the watchdog has only effect within the window boundary

WLB
WDT Window Lower Bound Register (10$_H$) Reset Value: 0000 0000$_H$



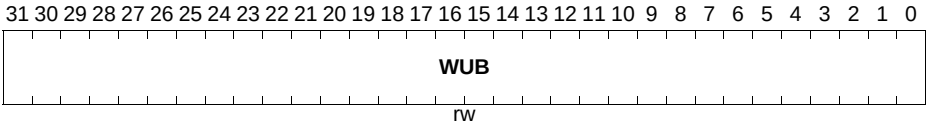
Field	Bits	Type	Description
WLB	[31:0]	rw	**Window Lower Bound** Lower bound for servicing window. Setting the lower bound to 0$_H$ disables the window mechanism.

WUB

The Window Upper Bound register defines the upper bound for servicing window. Servicing of the watchdog has only effect within the window boundary.

WUB

WDT Window Upper Bound Register (14$_H$) **Reset Value: FFFF FFFF$_H$**



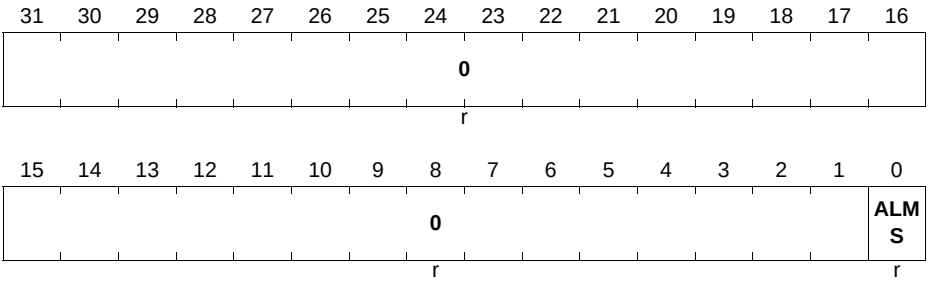
Field	Bits	Type	Description
WUB	[31:0]	rw	**Window Upper Bound** Upper Bound for servicing window. The WDT triggers an reset request when the timer is crossing the upper bound value without pre-warning enabled. With pre-warning enabled the first crossing triggers a watchdog alarm and the second crossing triggers a system reset.

WDTSTS

The status register contains sticky bit indicating occurrence of alarm condition.

WDTSTS

WDT Status Register (0018$_H$) **Reset Value: 00000000$_H$**



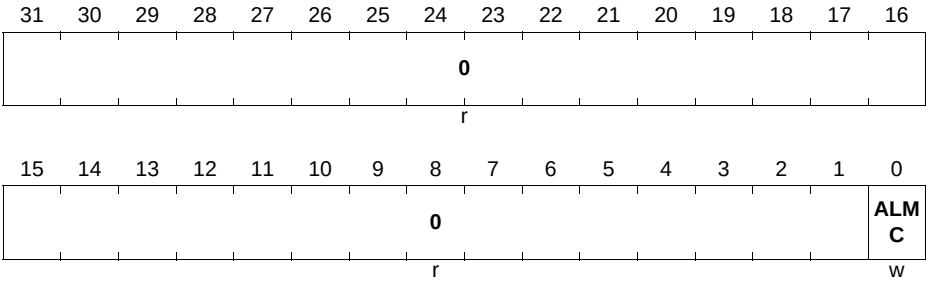
Field	Bits	Type	Description
ALMS	0	r	**Pre-warning Alarm** 1_B pre-warning alarm occurred, 0_B no pre-warning alarm occurred

Field	Bits	Type	Description
0	[31:1]	r	**Reserved**

WDTCLR

The status register contains sticky bit field indicating occurrence of alarm condition.

WDTCLR
WDT Clear Register **($001C_H$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
ALMC	0	w	**Pre-warning Alarm** 1_B clears pre-warning alarm 0_B no-action
0	[31:1]	r	**Reserved**

9.10 Interconnects

Table 9-4 Pin Table

Input/Output	I/O	Connected To	Description
Clock and Reset Signals			
f_{WDT}	I	SCU.CCU	timer clock
Timer Signals			
wdt_service	O	SCU.HCU	service indication to external watchdog

Table 9-4 Pin Table (cont'd)

Input/Output	I/O	Connected To	Description
HALTED	I	CPU	In halting mode debug. HALTED remains asserted while the core is in debug.
Service Request Connectivity			
wdt_alarm	O	SCU.GCU	pre-warning alarm
wdt_rst_req	O	SCU.RCU	reset request

10 Real Time Clock (RTC)

Real-time clock (RTC) is a clock that keeps track of the current time. RTCs are present in almost any electronic device which needs to keep accurate time in a digital format for clock displays and real-time actions.

10.1 Overview

The RTC module tracks time with separate registers for hours, minutes, and seconds. The calendar registers track date, day of the week, month and year with automatic leap year correction.

The RTC is capable of running from an alternate source of power, so it can continue to keep time while the primary source of power is off or unavailable. The timer remains operational when the core domain is in power-down. The kernel part of the RTC keeps running as long as the hibernate domain is powered with an alternate supply source. The alternate source can be for example a lithium battery or a supercapacitor.

10.1.1 Features

The features of the Real Time Clock (RTC) module are:

- Precise real time keeping with
  - 32.768 kHz external crystal clock
  - 32.768 kHz high precision internal clock
- Periodic time-based interrupt
- Programmable alarm interrupt on time match
- Supports wake-up mechanism from hibernate state

Table 10-1 Application Features

Feature	Purpose/Application
Precise real-time keeping	Reduced need for time adjustments
Periodic time-based interrupt	Scheduling of operations performed on precisely defined intervals
Programmable alarm interrupt on time match	Scheduling of operations performed on precisely defined times
Supports wake-up mechanism from hibernate state	Autonomous wake up from hibernate for system state control and maintaintenance routine operations

10.1.2 Block Diagram

The RTC block diagram is shown in **Figure 10-1**.

The main building blocks of the RTC is Time Counter implementing real time counter and RTC registers containing multi-field registers for the time counter and alarm programming register. Dedicated fields represent values for elapsing second, minutes, hours, days, days of week, months and years.

The kernel of the RTC module is instantiated in the hibernate domain.

The RTC registers are instantiated in hibernate domain and mirrored in SCU. Access to the RTC registers is performed via register mirror updated over serial interface.

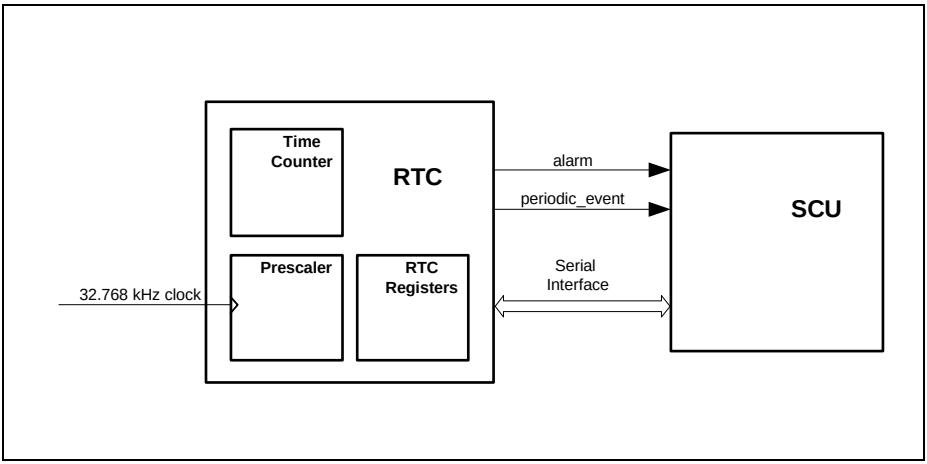


Figure 10-1 Real-Time Clock Block Diagram Structure

10.2 RTC Operation

The RTC timer counts seconds, minutes, hours, days of month, days of week, months and years each in a separate field (see **Figure 10-2**). Individual bit fields of the RTC counter can be programmed and read with software over serial interface via mirror registers in SCU module. For details of the serial communication please refer to SCU chapter.

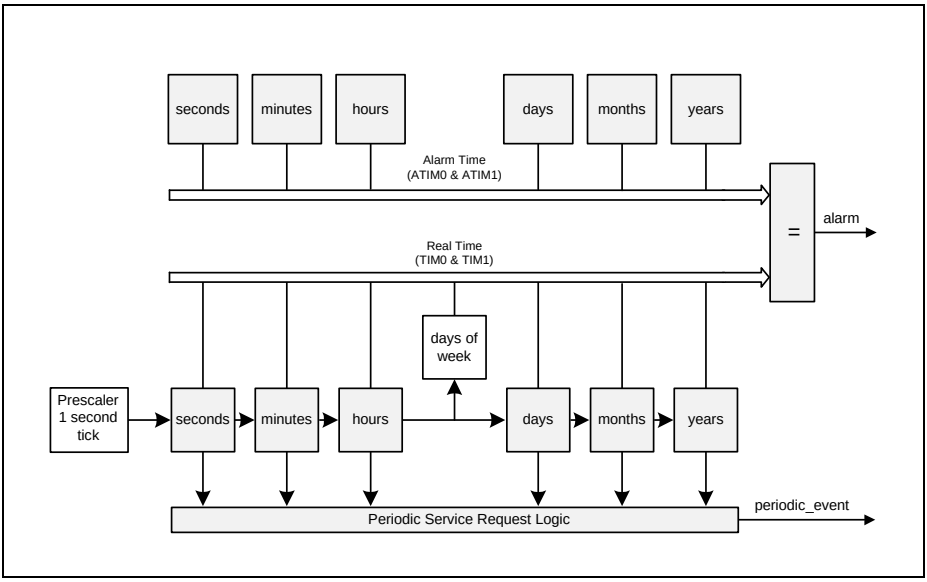


Figure 10-2 Block Diagram of RTC Time Counter

Occurrence of an internal timer event is stored in the service request raw status register **RAWSTAT**. The values of the status register **RAWSTAT** drive the outgoing service request lines alarm and periodic_event.

10.3 Register Access Operations

The RTC module is a part of SCU from programmming model perepcetive and shares register address space for configuration with other sub-modulesof SCU. RTC registers are instantiated in hibernate domain are mirrored in SCU. The registers get updated in both clock domains over serial interface running at 32kHz clock rate.

Any update of the registers is performed with some delay required for data to propagate to and from the mirror registers over serial interface. Accesses to the RTC registers in core domain must not block the bus interface of SCU module. For details of the register mirror and serial communication handling please refer to SCU chapter.

A write to writable registers is only allowed, when the access is in privileged mode. A write access in user mode results in a bus error response.

For consistent write to the timer registers **TIM0** and **TIM1**, the register **TIM0** has to be written before the register **TIM1**.

After wake-up from hibernate state the content of the mirror registers **TIM0** and **TIM1** is undefined until the first update of the corresponding RTC timers occurs and is propagated to the registers.

For consistent read-out of the timer registers **TIM0** and **TIM1**, the register **TIM0** has to be read before the register **TIM1**. The value of **TIM1** is stored in a shadow register upon each read of **TIM0** before they get copied to the mirror register in core domain.

10.4 Service Request Processing

The RTC generates service requests upon:

- periodic timer events
- configured alarm condition

The service requests can be processed in the core domain as regular service requests or as wake-up triggers from hibernate mode in HCU module in hibernate domain (for more details please refer to hibernate control description in SCU chapter).

10.4.1 Periodic Service Request

The periodic timer service request is raised whenever a non-masked field of the timer counter gets updated. Masking of the bits is performed using **MSKSR** register. Periodic Service requests can be disabled with **MSKSR**.

10.4.2 Timer Alarm Service Request

The alarm interrupt is triggered when **TIM0** and **TIM1** bit fields values match all corresponding bit fields values of **ATIM0**, **ATIM1** registers selected with **CTR** register. Timer Alarm Service requests can be disabled with **CTR**.

10.5 Debug behavior

The RTC clock does not implement dedicated debug mechanisms.

10.6 Power, Reset and Clock

RTC is instantiated entirely in hibernate domain and remains powered up when hibernate domain is powered up. Supply voltage is passed either from VDDP or VBAT pin as specified in the SCU chapter.

The RTC module remains in reset state along with entire hibernate domain after initial power up of hibernate domain until reset released with software.

The RTC timer is running from ether internal or external 32.768 kHz clock selectable with HDCR control register of SCU/HCU module. The prescaler setting of $7FFF_H$ results in an once per second update of the RTC timer.

10.7 Initialization and Control Sequence

Programming model of the RTC module assumes several scenarios where different control sequences apply.

Note: Some of the scenarios described in this chapter require operations on system level the that are not in the scope of the RTC module description, therefore for detailed information please refer to relevant chapters of this document.

10.7.1 Initialization & Start of Operation

Complete RTC module initialization is required upon hibernate domain reset. The hibernate domain needs to be enabled before any programming of RTC registers takes place. Accesses to RTC registers are performed via dedicated mirror registers (for more details please refer to SCU chapter)

- enable hibernate domain (if not disabled)
  - write one to SCU_PWRSET.HIB
- release reset of hibernate domain reset (if asserted)
  - write one to SCU_RSTCLR.HIBRS
- enable RTC module to start counting time
  - write one to RTC_CTR.ENB
- program RTC_TIM0 and RTC_TIM1 registers with current time
  - check SCU_MIRRSTS to ensure that no transfer over serial interface is pending to the RTC_TIM0 register
  - write a new value to the RTC_TIM0 register
  - check SCU_MIRRSTS to ensure that no transfer over serial interface is pending to the RTC_TIM1 register
  - write a new value to the RTC_TIM1 register

10.7.2 Re-configuration & Re-start of Operation

Reset and re-initialization of the RTC module may be required without complee power up sequence of the hibernate domain.

- apply and release reset of hibernate domain reset
  - write one to SCU_RSTSET.HIBRS
  - write one to SCU_RSTCLR.HIBRS
- enable RTC module to start counting time
  - write one to RTC_CTR.ENB
- program RTC_TIM0 and RTC_TIM1 registers with current time
  - check SCU_MIRRSTS to ensure that no transfer over serial interface is pending to the RTC_TIM0 register
  - write a new value to the RTC_TIM1 register
  - check SCU_MIRRSTS to ensure that no transfer over serial interface is pending to the RTC_TIM1 register

– write a new value to the RTC_TIM1 register

10.7.3 Configure and Enable Periodic Alarm

The RTC periodic alarm configuration require programming in order to enable intrrupt request generation out upon a change of value in the corresponding bit fields.

- enable service request for periodic timer events in RTC module
  – check SCU_MIRRSTS to ensure that no transfer over serial interface is pending to the RTC_MSKSR register
  – set MAI bit field of RTC_MSKSR register in order enable individual periodic timer events
- enable service request for periodic timer events in RTC module
  – set PI bit field of SCU_SRMSK register in order enable generation of interrupts upon periodic timer events

10.7.4 Configure and Enable Timer Alarm

The RTC periodic alarm configuration require programming in order to enable intrrupt request generation out upon compare match of values in the corresponding bit fields of TIM0 and TIM1 against ATIM0 and ATIM1 respectively.

- program compare values in individual bit fields of ATIM0 and ATIM1 in RTC module
  – check SCU_MIRRSTS to ensure that no transfer over serial interface is pending to the RTC_ATIM0 register
  – write to RTC_ATIM0 register bit fields
  – check SCU_MIRRSTS to ensure that no transfer over serial interface is pending to the RTC_ATIM1 register
  – write to RTC_ATIM1 register bit fields
- enable service request for timer alarm events in RTC module
  – check SCU_MIRRSTS to ensure that no transfer over serial interface is pending to the RTC_CTR register
  – set TAE bit field of RTC_CTR register in order enable individual periodic timer events
- enable service request for timer alarm events in RTC module
  – write one to AI bit field of SCU_SRMSK register in order enable generation of interrupts upon periodic timer events

10.8 Registers

Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 10-2 Registers Address Space

Module	Base Address	End Address	Note
RTC	5000 4A00$_H$	5000 4BFF$_H$	Accessible via Mirror Registers

Table 10-3 Register Overview

Short Name	Register Long Name	Offset Addr.	Access Mode		Description
			Read	Write	
RTC Kernel Registers					
ID	ID Register	0000$_H$	U, PV	BE	Page 10-7
CTR	Control Register	0004$_H$	U, PV	PV	Page 10-8
RAWSTAT	Raw Service Request Register	0008$_H$	U, PV	BE	Page 10-9
STSSR	Status Service Request Register	000C$_H$	U, PV	BE	Page 10-10
MSKSR	Mask Service Request Register	0010$_H$	U, PV	PV	Page 10-11
CLRSR	Clear Service Request Register	0014$_H$	BE	PV	Page 10-13
ATIM0	Alarm Time Register 0	0018$_H$	U,PV	PV	Page 10-14
ATIM1	Alarm Time Register 1	001C$_H$	U,PV	PV	Page 10-15
TIM0	Time Register 0	0020$_H$	U, PV	PV	Page 10-16
TIM1	Time Register 1	0024$_H$	U, PV	PV	Page 10-18

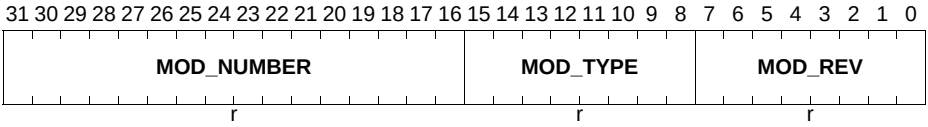
10.8.1 Registers Description

ID

Read-only ID register of the RTC module containing unique identification code of the RTC module.

ID

RTC ID Register **(00_H)** **Reset Value: 00A3 C0XX$_H$**



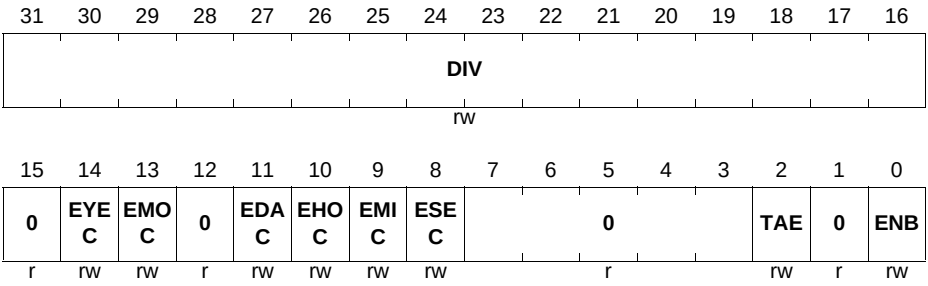
Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision** Indicates the revision number of the implementation. This information depends on the design step.
MOD_TYPE	[15:8]	r	**Module Type** This internal marker is fixed to $C0_H$.
MOD_NUMBER	[31:16]	r	**Module Number** Indicates the module identification number

CTR

RTC Control Register providing control means of the operation mode of the module.

CTR

RTC Control Register **(04_H)** **Reset Value: 7FFF 0000$_H$**



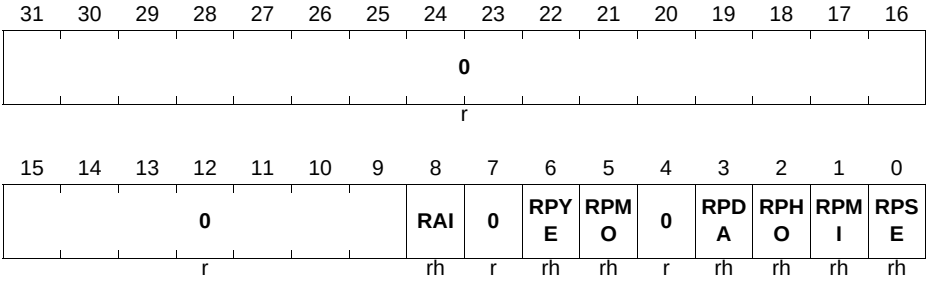
Field	Bits	Type	Description
ENB	0	rw	**RTC Module Enable** 0_B disables RTC module 1_B enables RTC module

Field	Bits	Type	Description
TAE	2	rw	**Timer Alarm Enable** 0_B disable timer alarm 1_B enable timer alarm
ESEC	8	rw	**Enable Seconds Comparison** 0_B disabled 1_B enabled
EMIC	9	rw	**Enable Minutes Comparison** 0_B disabled 1_B enabled
EHOC	10	rw	**Enable Hours Comparison** 0_B disabled 1_B enabled
EDAC	11	rw	**Enable Days Comparison** 0_B disabled 1_B enabled
EMOC	13	rw	**Enable Months Comparison** 0_B disabled 1_B enabled
EYEC	14	rw	**Enable Years Comparison** 0_B disabled 1_B enabled
DIV	[31:16]	rw	**Divider Value** reload value of RTC prescaler. Clock is divided by DIV+1. $7FFF_H$ is default value for RTC mode with 32.768 kHz crystal or internal clock
0	1,[7:3], 12,15	r	**Reserved** Read as 0; should be written with 0

RAWSTAT

RTC Raw Service Request Register contains raw status info i.e. before status mask takes effect on generation of service requests. This register serves debug purpose but can be also used for polling of the status without generating serice requests.

RAWSTAT
RTC Raw Service Request Register (08_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
RPSE	0	rh	**Raw Periodic Seconds Service Request** Set whenever seconds count increments
RPMI	1	rh	**Raw Periodic Minutes Service Request** Set whenever minutes count increments
RPHO	2	rh	**Raw Periodic Hours Service Request** Set whenever hours count increments
RPDA	3	rh	**Raw Periodic Days Service Request** Set whenever days count increments
RPMO	5	rh	**Raw Periodic Months Service Request** Set whenever months count increments
RPYE	6	rh	**Raw Periodic Years Service Request** Set whenever years count increments
RAI	8	rh	**Raw Periodic Alarm Service Request** Set whenever count value matches compare value
0	4, 7, [31:9]	r	**Reserved**

STSSR

RTC Service Request Status Register contains status info reflecting status mask effect on generation of service requests. This register needs to be accessed by software in order to determine the actual cause of an event.

STSSR

RTC Service Request Status Register ($0C_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

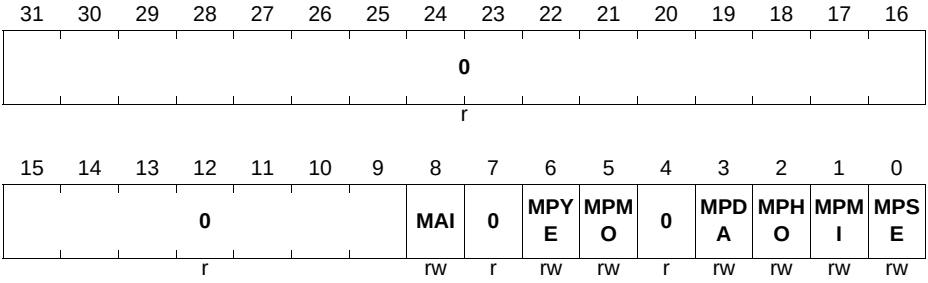
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							SAI	0	SPYE	SPMO	0	SPDA	SPHO	SPMI	SPSE
r							rh	r	rh	rh	r	rh	rh	rh	rh

Field	Bits	Type	Description
SPSE	0	rh	**Periodic Seconds Service Request Status after Masking**
SPMI	1	rh	**Periodic Minutes Service Request Status after Masking**
SPHO	2	rh	**Periodic Hours Service Request Status after Masking**
SPDA	3	rh	**Periodic Days Service Request Status after Masking**
SPMO	5	rh	**Periodic Months Service Request Status after Masking**
SPYE	6	rh	**Periodic Years Service Request Status after Masking**
SAI	8	rh	**Periodic Alarm Service Request Status after Masking**
0	4, 7, [31:9]	r	**reserved**

MSKSR

RTC Service Request Mask Register contains masking value for generation control of service requests or interrupts.

MSKSR
RTC Service Request Mask Register (10$_H$) **Reset Value: 0000 0000$_H$**

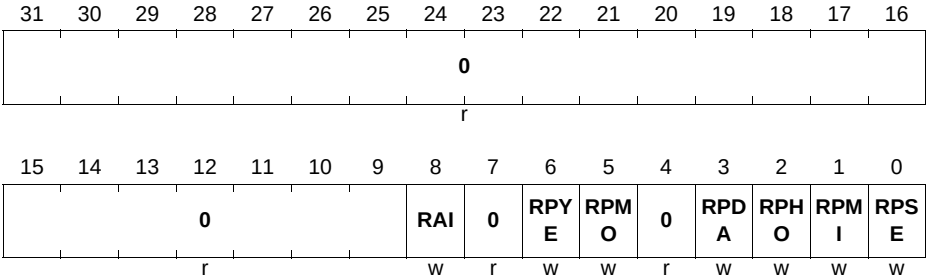


Field	Bits	Type	Description
MPSE	0	rw	**Periodic Seconds Interrupt Mask** 0_B disable 1_B enable
MPMI	1	rw	**Periodic Minutes Interrupt Mask** 0_B disable 1_B enable
MPHO	2	rw	**Periodic Hours Interrupt Mask** 0_B disable 1_B enable
MPDA	3	rw	**Periodic Days Interrupt Mask** 0_B disable 1_B enable
MPMO	5	rw	**Periodic Months Interrupt Mask** 0_B disable 1_B enable
MPYE	6	rw	**Periodic Years Interrupt Mask** 0_B disable 1_B enable
MAI	8	rw	**Periodic Alarm Interrupt Mask** 0_B disable 1_B enable
0	4, 7, [31:9]	r	**Reserved**

CLRSR

RTC Clear Service Request Register serves purpose of clearing sticky bits of **RAWSTAT** and **STSSR** registers. Write one to a bit in order to clear it is set. Writing zero has no effect on the set nor reset bits.

CLRSR
RTC Clear Service Request Register (14$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
RPSE	0	w	**Raw Periodic Seconds Interrupt Clear** 0_B no effect 1_B clear raw status bit
RPMI	1	w	**Raw Periodic Minutes Interrupt Clear** 0_B no effect 1_B clear raw status bit
RPHO	2	w	**Raw Periodic Hours Interrupt Clear** 0_B no effect 1_B clear raw status bit
RPDA	3	w	**Raw Periodic Days Interrupt Clear** 0_B no effect 1_B clear raw status bit
RPMO	5	w	**Raw Periodic Months Interrupt Clear** 0_B no effect 1_B clear raw status bit
RPYE	6	w	**Raw Periodic Years Interrupt Clear** 0_B no effect 1_B clear raw status bit

Field	Bits	Type	Description
RAI	8	w	**Raw Alarm Interrupt Clear** 0_B no effect 1_B clear raw status bit
0	4, 7, [31:9]	r	**Reserved**

ATIM0

RTC Alarm Time Register 0 serves purpose of programming single alarm time at a desired point of time reflecting comparison configuration in the **CTR** for individual fields against **TIM0** register. The register contains portion of bit fields for seconds, minutes, hours and days. Upon attempts to write an invalid value to a bit field e.g. exceeding maximum value default value gets programmed as described for each individual bit fields.

ATIM0
RTC Alarm Time Register 0 **(18_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0			**ADA**					**0**			**AHO**				
r			rw					r			rw				

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			**AMI**					**0**		**ASE**					
r			rw					r		rw					

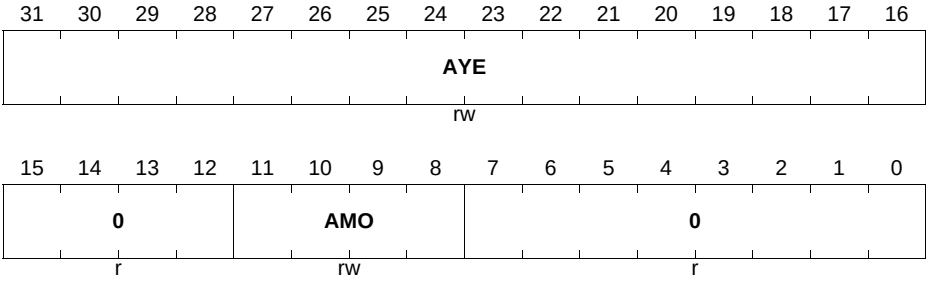
Field	Bits	Type	Description
ASE	[5:0]	rw	**Alarm Seconds Compare Value** Match of seconds timer count to this value triggers alarm seconds interrupt. Setting value equal or above $3C_H$ results in setting the field value to 0_H

Field	Bits	Type	Description
AMI	[13:8]	rw	**Alarm Minutes Compare Value** Match of minutes timer count to this value triggers alarm minutes interrupt. Setting value equal or above $3C_H$ results in setting the field value to 0_H
AHO	[20:16]	rw	**Alarm Hours Compare Value** Match of hours timer count to this value triggers alarm hours interrupt. Setting value equal or above 18_H results in setting the field value to 0_H
ADA	[28:24]	rw	**Alarm Days Compare Value** Match of days timer count to this value triggers alarm days interrupt. Setting valueequal above $1F_H$ results in setting the field value to 0_H
0	[7:6], [15:14], [23:21], [31:29]	r	**Reserved**

ATIM1

RTC Alarm Time Register 1 serves purpose of programming single alarm time at a desired point of time reflecting comparison configuration in the **CTR** for individual fields against **TIM1** register. The ATM1 register contains portion of bit fields for days of week, months and years. Upon attempts to write an invalid value to a bit field e.g. exceeding maximum value default value gets programmed as described for each individual bit fields.

ATIM1
RTC Alarm Time Register 1 **($1C_H$)** **Reset Value: 0000 0000$_H$**

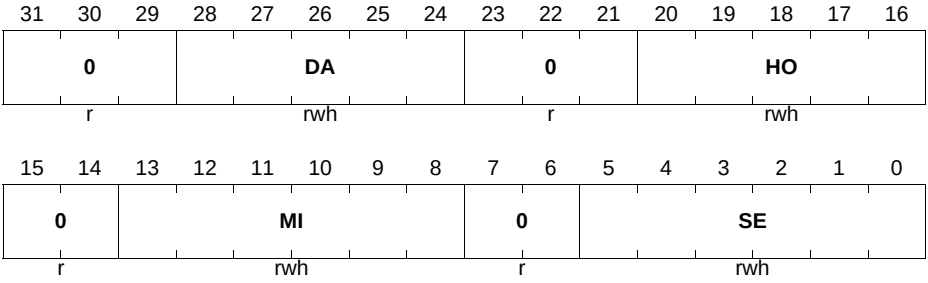


Field	Bits	Type	Description
AMO	[11:8]	rw	**Alarm Month Compare Value** Match of months timer count to this value triggers alarm month interrupt. Setting value equal or above the number of days of the actual month count results in setting the field value to 0_H
AYE	[31:16]	rw	**Alarm Year Compare Value** Match of years timer count to this value triggers alarm years interrupt.
0	[7:0], [15:12]	r	**Reserved**

TIM0

RTC Time Register 0 contains current time value for seconds, minutes, hours and days. The bit fields get updated in intervals corresponding with their meaning accordingly. The register needs to be programmed to reflect actual time after initial power up and will continue counting time also while in hibernate mode. Upon attempts to write an invalid value to a bit bield e.g. exceeding maximum value a default value gets programmed as described for each individual bit fields.

TIM0
RTC Time Register 0 **(20_H)** **Reset Value: 0000 0000$_H$**



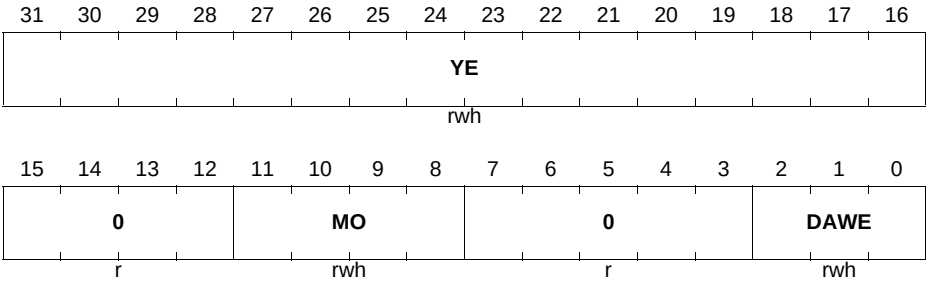
Field	Bits	Type	Description
SE	[5:0]	rwh	**Seconds Time Value** Setting value equal or above $3C_H$ results in setting the field value to 0_H. Value can only be written, when RTC is disabled. After wake-up from hibernate, value is undefined until first update of RTC.
MI	[13:8]	rwh	**Minutes Time Value** Setting value equal or above $3C_H$ results in setting the field value to 0_H. Value can only be written, when RTC is disabled. After wake-up from hibernate, value is undefined until first update of RTC.
HO	[20:16]	rwh	**Hours Time Value** Setting value equal or above 18_H results in setting the field value to 0_H Value can only be written, when RTC is disabled. After wake-up from hibernate, value is undefined until first update of RTC.

Field	Bits	Type	Description
DA	[28:24]	rwh	**Days Time Value** Setting value equal or above the number of days of the actual month count results in setting the field value to 0_H Value can only be written, when RTC is disabled. After wake-up from hibernate, value is undefined until first update of RTC. Days counter starts with value 0 for the first day of month.
0	[7:6], [15:14], [23:21], [31:29]	r	**Reserved**

TIM1

RTC Time Register 1 contains current time value for days of week, months and years. The bit fields get updated in intervals corresponding with their meaning accordingly. The register needs to be programmed to reflect actual time after initial power up and will continue counting time also while in hibernate mode. Upon attempts to write an invalid value to a bit bield e.g. exceeding maximum value a default value gets programmed as described for each individual bit fields.

TIM1
RTC Time Register 1 **(24_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
DAWE	[2:0]	rwh	**Days of Week Time Value** Setting value above 6_H results in setting the field value to 0_H Value can only be written, when RTC is disabled. After wake-up from hibernate, value is undefined until first update of RTC. Days counter starts with value 0 for the first day of week.
MO	[11:8]	rwh	**Month Time Value** Setting value equal or above C_H results in setting the field value to 0_H. Value can only be written, when RTC is disabled. After wake-up from hibernate, value is undefined until first update of RTC. Months counter starts with value 0 for the first month of year.
YE	[31:16]	rwh	**Year Time Value** Value can only be written, when RTC is disabled. After wake-up from hibernate, value is undefined until first update of RTC.
0	[7:3], [15:12]	r	**Reserved**

10.9 Interconnects

Table 10-4 Pin Connections

Input/Output	I/O	Connected To	Description
Clock Signals			
f_{RTC}	I	SCU.HCU	32.768 kHz clock selected in hibernate domain
Service Request Connectivity			
periodic_event	O	SCU.GCU	Timer periodic service request
alarm	O	SCU.GCU	Alarm service request

11 System Control Unit (SCU)

The SCU is the SoC power, reset and a clock manager with additional responsibility of providing system stability protection and other auxiliary functions.

11.1 Overview

The functionality of the SCU described in this chapter is organized in the following sub-chapters, representing different aspects of system control:

- Miscellaneous control functions, **Chapter 11.2**
- Power Control, **Chapter 11.3**
- Hibernate Control, **Chapter 11.4**
- Reset Control, **Chapter 11.5**
- Clock Control, **Chapter 11.6**

11.1.1 Features

The following features are provided for monitoring and controlling the system:

- General Control
  - Boot Mode Detection
  - Memory Content Protection
  - Trap Generation
  - Die Temperature Measurement
  - Retention Memory Support
- Power Control
  - Power Sequencing
  - EVR Control
  - Supply Watchdog
  - Voltage Monitoring
  - Power Validation
  - Power State Indication
  - Flash Power Control
- Hibernate Control
  - Hibernate Mode Control
  - Wake-up from Hibernate Mode
  - Hibernate Domain Control
- Reset Control
  - Reset assertion on various reset request sources
  - System Reset Generation
  - Inspection of Reset Sources After Reset
  - Selective Module reset
- Clock Control

- Input clock selection
- Clock Generation
- Clock Distribution
- Clock Supervision
- Power Management
- RTC Clock

11.1.2 Block Diagram

The block diagram shown in **Figure 11-1** reflects logical organization of the System Control Unit.

- Power Control Unit (PCU)
- Hibernate Control Unit (HCU)
- Reset Control Unit (RCU)
- General Control Unit (GCU)
- Clock Control Unit (CCU)

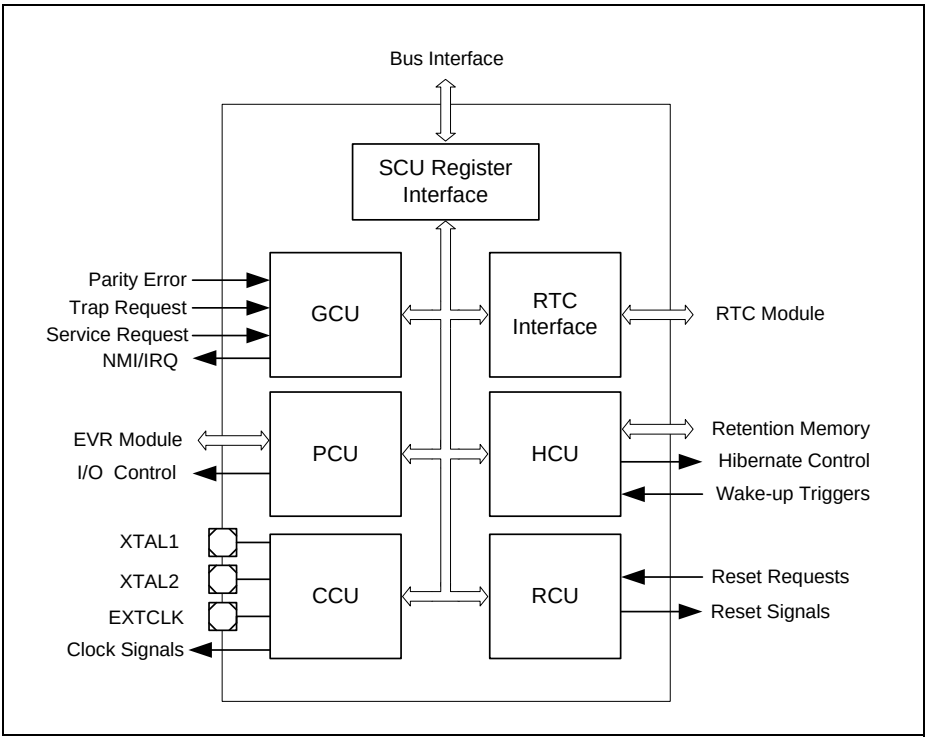


Figure 11-1 SCU Block Diagram

Interface of General Control Unit

The General Control Unit GCU has a memory fault interface to the memory validation logic of each on-chip SRAM and the Flash to receive memory fault events, as parity errors. NMI request are routed to the unit to be gated and combined. The GCU provides the start up protection to all units, which require an additional level of protection.

Interface of Power Control Unit

The Power Control Unit PCU has an interface to the Embedded Voltage Regulator (EVR) and an interface to the PORTS module. The PCU related signals are described in more detail in **Chapter 11.3**.

Interface of Reset Control Unit

The Reset Control Unit RCU has an interface to the Embedded Voltage Regulator (EVR). The RCU receives from the EVR the power-on reset and the reset information for

each power related reset. Reset requests are coming to the unit from the watchdog, the CPU and the test control unit. The RCU is providing the reset signals to all other units of the chip in the Core power domain. The RCU related signals are described in more detail in **Chapter 11.5**.

Interface of Clock Control Unit

The Clock Control Unit (CCU) receives the external clock source via the crystal pins XTAL1 and XTAL2. As further clock source the CCU receives the standby clock f_{STDBY} from the Hibernate domain. The CCU drives an external clock output, where internal clocks can be routed out. The CCU provides the clock signals to all other units of the chip.

Interface of Hibernate Power Domain

The interface to the Hibernate domain provides mirror registers updated via a shared serial interface to the Retention Memory, RTC module registers and Hibernate domain control registers. Update of the mirror registers over the serial is controlled using **MIRRSTS**, **RMDATA** and **RMACR** registers. End of update can also trigger service requests via **SRSTAT** register. Refresh of the Hibernate domain registers in the register mirror are performed continuously, as fast as possible in order to instantly reflect any register state change on both sides. The serial interface is inactivated while in hibernate mode in order to reduce power.

Interface of Retention Memory

Access to the Retention Memory is served over shared serial interface identical to the one used to access Hibernate domain registers, for detail please refer to **“Interface of Hibernate Power Domain” on Page 11-4**.

Interface of RTC

Access to the RTC module is served over shared serial interface identical to the one used to access Hibernate domain registers, for detail please refer to **“Interface of Hibernate Power Domain” on Page 11-4**. The RTC module functionality is described in separate RTC chapter.

11.2 Miscellaneous control functions

System Control implements system management functions accessible via GCU registers. General system control including various auxiliary function is performed in General Control Unit (GCU).

11.2.1 Startup Software Support

Externally driven boot mode pins decide the boot mode after a power on reset. It also possible for applications to decide the boot mode. Ability to determine boot mode values on boot mode pins and user application desired boot modes is provided by SCU.

11.2.2 Service Requests

Service request events listed in **Table 11-1** can result in assertion of a regular interrupt or an NMI. Please refer to **SRMSK** and **NMIREQEN** register description.

The interrupt structure is shown in **Figure 11-2**. The interrupt request or the corresponding interrupt set bit (in register **SRSET**) can trigger the interrupt generation at the selected interrupt node x. The service request pulse is generated independently from the interrupt flag in register **SRSTAT**. The interrupt flag can be cleared by software by writing to the corresponding bit in register **SRCLR**. In addition several service requests can be promoted to NMI trigger level using **NMIREQEN** register

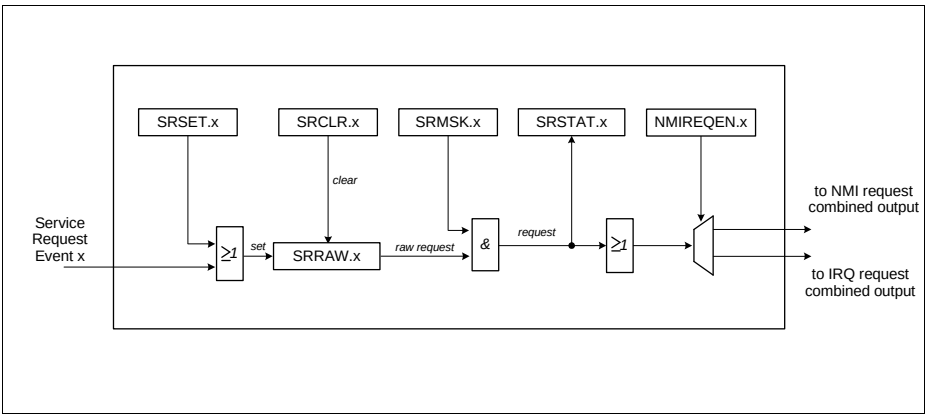


Figure 11-2 Service Request Subsystem

The trap flag in register **SRSTAT** can be cleared by software by writing to the corresponding bit in register **SRCLR**. All trap requests are combined to one common line and connected to a regular interrupt node or NMI node of NVIC.

The trap requests have a sticky flag in register **SRRAW**.

Note: When servicing an SCU service request, make sure that all related request flags are cleared after the identified request has been handled.

11.2.2.1 Service Request Sources

The SCU supports service request sources listed in **Table 11-2** and reflected in the **SRSTAT**, **SRRAW**, **SRMSK**, **SRCLR** and **SRSET** registers.

Table 11-1 Service Requests

Service Request Name	Service Request Short Name
WDT pre-warning	PRWARN
RTC Periodic Event	PI
RTC Alarm	AI
DLR Request Overrun	DLROVR
HDCTAT Mirror Register Updated	HDSTAT
HDCLR Mirror Register Updated	HDCLR
HDSET Mirror Register Updated	HDSET
HDCR Mirror Register Updated	HDCR
OSCSICTRL Mirror Register Updated	OSCSICTRL
OSCULSTAT Mirror Register Updated	OSCULSTAT
OSCULCTRL Mirror Register Updated	OSCULCTRL
RTC CTR Mirror Register Updated	RTC_CTR
RTC ATIM0 Mirror Register Updated	RTC_ATIM0
RTC ATIM1 Mirror Register Updated	RTC_ATIM1
RTC TIM0 Mirror Register Updated	RTC_TIM0
RTC TIM1 Mirror Register Updated	RTC_TIM1
Retention Memory Mirror Register Updated	RMX

11.2.3 Memory Content Protection

For supervising the content of the on-chip memories, the following mechanism is provided:

All on-chip SRAMs provide protection of content via parity checking. The parity logic generates additional parity bits which are stored along with each data word at a write operation. A read operation implies checking of the previous stored parity information.

An occurrence of a parity error is observable at the memory error raw status register. It is configurable whether a memory error should trigger an NMI or system reset.

11.2.3.1 Parity Error Handling

The on-chip RAM modules check parity information during read accesses and in case of an error a signal can be generated if enabled with **PEEN** register. Two modes of parity error signalling are implemented:

- bus error
- parity error trap (NMI)

The bus error generation applies to memories that can be accessed directly from the bus system level. Apart from that, all memories, including those that are not accessible directly and are internal to peripherals are capable of generating system traps resulting in NMI. Parity trap requests get enabled with **PETE** register implementing individual control for each memory. Parity error signalling with trap generation is not recommended to be used for memories capable of bus error generation and therefore should be disabled.

Parity error trap generation mechanism can be also used to generate system reset if enabled with **PERSTEN** register in conjunction with the **PETE** register configuration. For more details of the parity error generation scheme please refer to **Figure 11-3**.

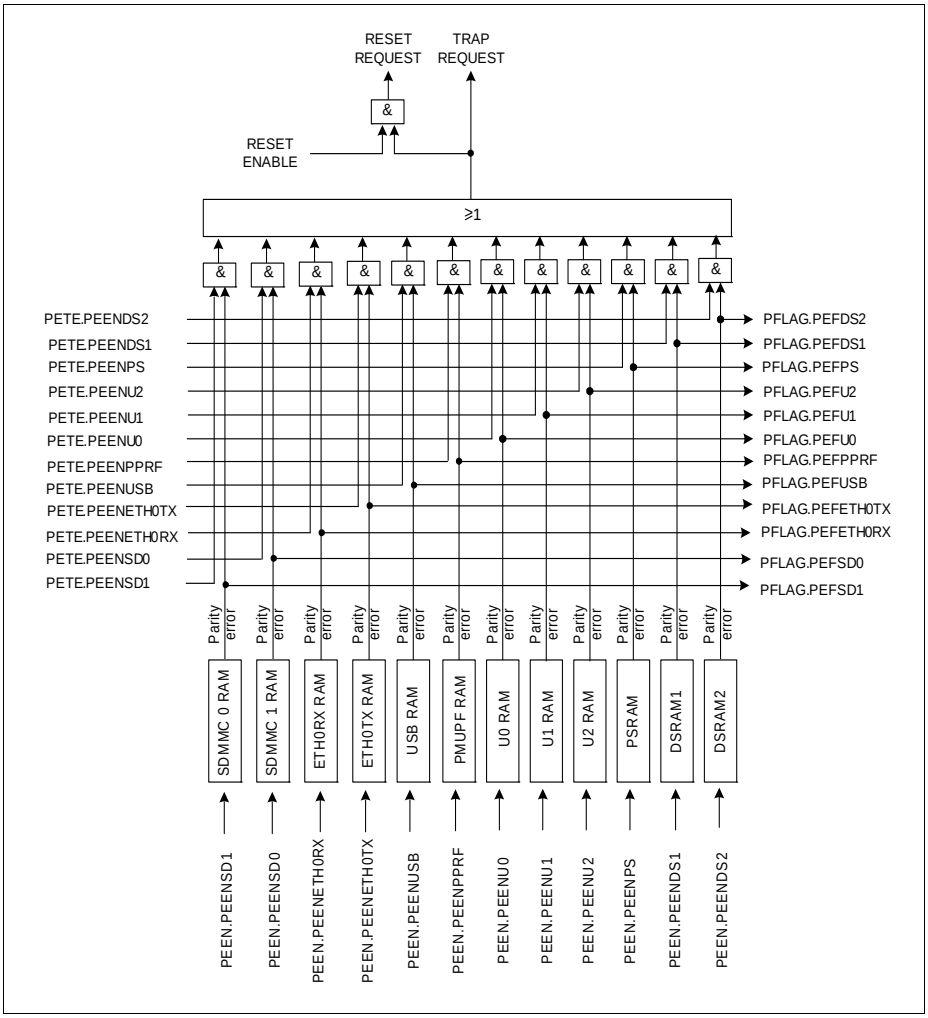


Figure 11-3 XMC4500 Parity Error Control Logic

The logic is controlled by registers **PMTSR** and **PMTPR**. Via bit field **PMTPR**.PWR a parity value can be written to any address of every memory for software driven testing purpose. The parity control software test update has to be enabled with bit **PMTSR** for each memory individually. Otherwise a write to the parity control has no effect. With each read access to a memory the parity from the memory parity control is stored in register **PMTPR** and accessible with software.

Note: Test software should be located in external memory.

11.2.4 Trap Generation

Several abnormal events listed in **Table 11-2** can result in assertion of NMI. Please refer to **TRAPDIS** register description.

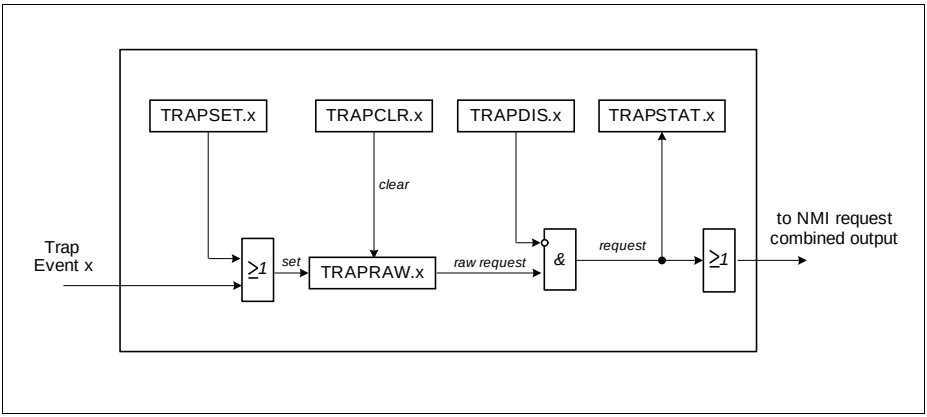


Figure 11-4 Trap Subsystem

The trap flag in register **TRAPSTAT** can be cleared by software by writing to the corresponding bit in register **TRAPCLR**. All trap requests are combined to one common line and connected to NMI node of NVIC.

The trap requests have a sticky flag in register **TRAPRAW**.

Note: When servicing an SCU trap request, make sure that all related request flags are cleared after the identified request has been handled.

11.2.4.1 Trap Sources

The SCU supports trap sources listed in **Table 11-2** and reflected in the **TRAPSTAT**, **TRAPRAW**, **TRAPDIS**, **TRAPCLR** and **TRAPSET** registers.

Table 11-2 Trap Requests

Source of Trap	Short Trap Name
System PLL Trap	SOSCWDGT
USB VCO Lock Trap	SVCOLCKT
System VCO Lock Trap	UVCOLCKT
Parity Error Trap	PET
Brownout Trap	BRWNT

Table 11-2 Trap Requests (cont'd)

Source of Trap	Short Trap Name
OSCULP Watchdog Trap	ULPWDGT
Peripheral Bus 0 Write Error Trap	BWERR0T
Peripheral Bus 1 Write Error Trap	BWERR1T

11.2.5 Die Temperature Measurement

The Die Temperature Sensor (DTS) generates a measurement result that indicates directly the current temperature. The result of the measurement is displayed via bit field **DTSSTAT**.RESULT. In order to start one measurement bit **DTSCON**.START needs to be set.

The DTS has to be enabled before it can be used via bit **DTSCON**.PWD. When the DTS is powered temperature measurement can be started.

In order to adjust production variations of temperature measurement accuracy bit field **DTSCON**.BGTRIM is provided. **DTSCON**.BGTRIM can be programmed by the user software.

Measurement data is available certain time after measurement started. Register **DTSSTAT**.RDY bit indicated that the DTS is ready to start a measurement. If a started measurement is finished or still in progress is indicated via the status bit **DTSSTAT**.BUSY.

The formula to calculate the die temperature is defined in the Target Data Sheet.

Note: The first measurement after the DTS was powered delivers a result without calibration adjustment and should be ignored.

11.2.6 Retention Memory

Retention memory for context store/restore is available hibernate mode support. The retention memory area is located in Hibernate domain. Any content is retained in hibernate mode, when the core might be without power supply. The user software can store some context critical data in the memory before entering hibernate mode and access the data after booting up if a wake-up from hibernate bode is signalized in **RSTSTAT** register.

Access to the 64 Bytes of retention memory is provided via **RMDATA** register, controlled with **RMACR** register. The purpose of **RMACR** register is addressing of the memory cells and issuing read/write command. The **RMDATA** is read or written by software according to the access direction after data transfer has been completed as indicated with RMX bit field in **MIRRSTS**.

11.2.7 Out of Range Comparator Control

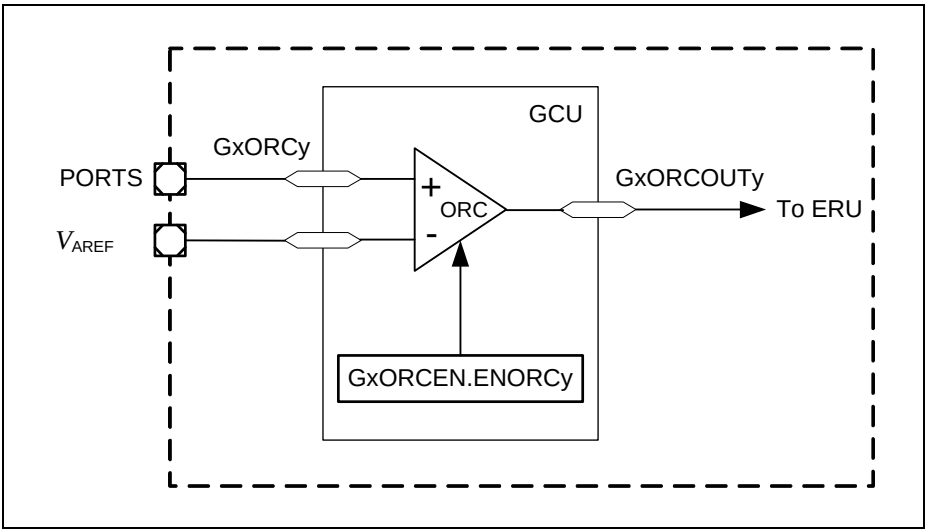


Figure 11-5 Out od Range Comparator Control

The out of range comparator serves the purpose of overvoltage monitoring for analog input pins of the chip. A number of analog channels are associated with dedicated pads connected to inputs of analog modules. They get supervised by dedicated circuits constituted of analog comparator controlled by digital signals as depicted in **Figure 11-5**. Detection of input voltage exceeding V_{AREF} triggers a service request to the ERU. Digital control signals are generated in GCU submodule of SCU. Dedicated registers **G0ORCEN** and **G1ORCEN** provide control means for enabling and disabling monitoring of analog channels.

11.3 Power Management

Power management control is performed in the Power Control Unit (PCU).

11.3.1 Functional Description

The XMC4500 is running from a single external power supply (V_{DDP}). The main supply voltage is supervised by a supply watchdog.

The I/Os and the main part of the flash block are running directly from the external supply voltage. The core voltage (V_{DDC}) is generated by an on-chip Embedded Voltage

Regulator (EVR). The safe voltage range of the core voltage is supervised by a power validation circuit, which is part of the EVR.

Logic in the hibernate domain, mainly the real-time clock RTC, hibernate control and retention memory, is supplied by an auxiliary power supply using an additional power pad called. The auxiliary V_{BAT} voltage, supplied from e.g. a coin battery, enables the RTC to operate while the main supply is switched off.

11.3.2 System States

The system has the following general system states:

- Active
- Sleep
- Deep-Sleep
- Hibernate

Figure 11-6 shows the state diagram and the transitions between these power modes. The additional state power-up is only a transient state which is passed on cold or warm start-up from Off state.

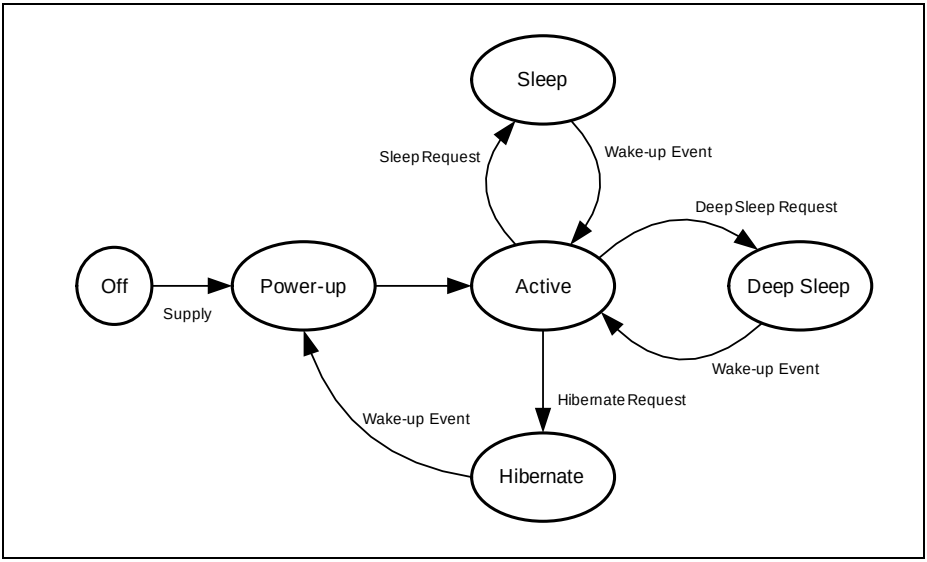


Figure 11-6 System States Diagram

Active State

The Active state is the normal operation state. The system is fully powered. The CPU is usually running from a high-speed clock. Depending on the application the system clock

might be slowed down. The PLL output clock or another clock can be selected as clock source. Unused peripherals might be stopped. Stopping a peripheral means that the peripheral i put into reset and the clock to this peripherals is disabled.

After a cold start the hibernate domain stays disabled until activated by user code.

Sleep State

The Sleep state of the system corresponds to the Sleep state of the CPU. The state is entered via WFI or WFE instruction of the CPU. In this state the clock to the CPU is stopped. The source of the system clock may be altered. Peripherals clocks are gated according to the **SLEEPCR** register.

Peripherals can continue to operate unaffected and eventually generate an event to wake-up the CPU. Any interrupt to the NVIC will bring the CPU back to operation. The clock tree upon exit from SLEEP state is restored to what it was before entry into SLEEP state.

Deep-Sleep State

The Deep-Sleep State is entered on the same mechanism as the Sleep state with the addition that user code has enabled the Deep Sleep state in system control register. In Deep-Sleep state the OSCHP and the PLL may be switched off. The wake-up logic in the NVIC is still clocked by a free-running clock. Peripherals are only clocked when configured to stay enabled in the **DSLEEPCR** register. Configuration of peripherals and any SRAM content is preserved.

The Flash can be put into low-power mode to achieve a further power reduction. On wake-up Flash will be restarted again before instructions or data access is possible.

Any interrupt will bring the system back to operation via the NVIC.The clock setup before entering Deep Sleep state is restored upon wake-up.

Hibernate State

In Hibernate mode the power supply to the core is switched off. Additionally the power to the analog domain and the main supply V_{DDP} can be switched off. Only the Hibernate power domain will stay powered. The power supply of the Hibernate domain is switched automatically to the auxiliary supply when the main supply is no longer present.

The Hibernate State is entered using control register **HDCR** of HCU in the Hibernate domain that will drive the external Voltage Regulator with HIBOUT signal to switch off power to the chip (see **System Level Power Control Example**).

Depending on configuration the following wake-up sources will wake-up the system to normal operation:

- Edge detection on external WKUP signal

- RTC Alarm Event
- RTC Periodic Event
- OSCULP Watchdog Event

The system can only wake-up from Hibernate if V_{DDP} is present. An external power supply can be switched on by the HIBOUT signal of the Hibernate Control Unit.

All blocks outside of the Hibernate domain will see a complete power-up sequence upon wake-up.

11.3.3 Hibernate Domain Operating Modes

The standard use case of the Hibernate domain is the time keeping function with V_{BAT} available, keeping the Real Time Counter active while the System Supply is powered off.

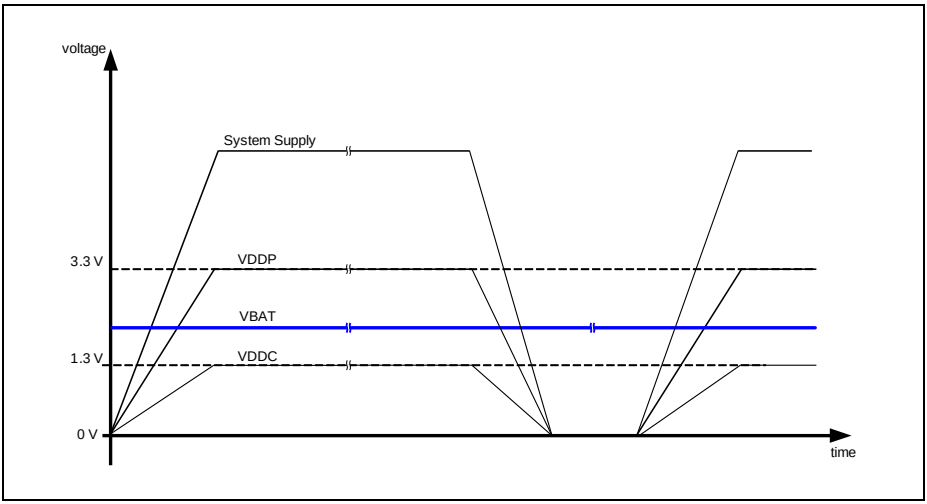


Figure 11-7 Hibernate state in time keeping mode

A special case of the Hibernate domain assumes that the V_{BAT} supply voltage is available only after the core domain power-up. This may occur if, for example, the battery gets plugged in or replaced after the core domain startup, or, if only a capacitor is available to hold V_{BAT} voltage for some limited time in case of absence of the main supply voltage in order to keep the Real Time Counter unaffected. The Hibernate domain requires to be switched on once after core domain power up with the dedicated register **PWRSET**.HIB. In this application case even switching off the main supply of the board does not affect availability of the V_{HIB} voltage required to keep RTC running.

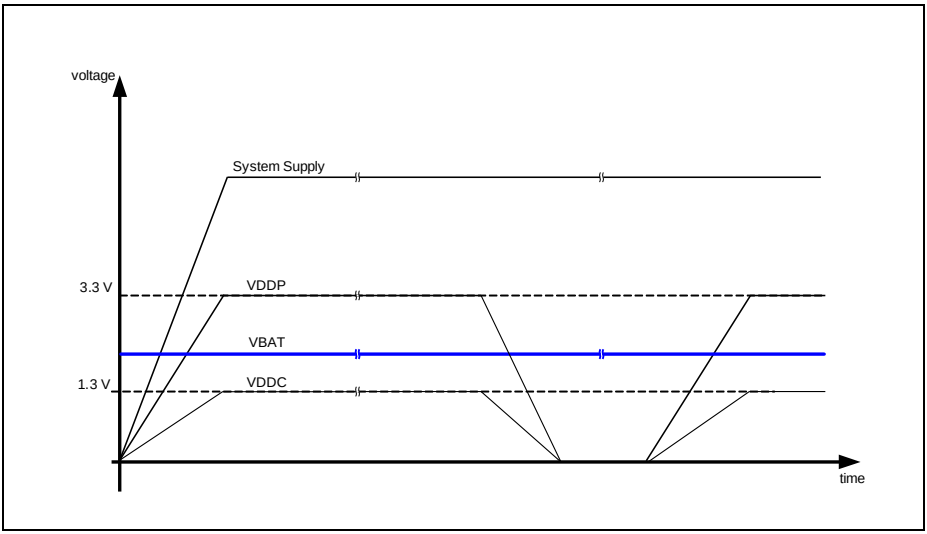


Figure 11-8 Hibernate controlled with external voltage regulator

The system power-down mode is controlled via HIB pin and external power supply device. The main supply voltage of the board stays active allowing selective disabling of the XMC4500 in order to save power while other devices of on the board remain active. At the time HIB pin indicates Hibernate mode the dedicated pass device connected to the chip will stop generating V_{DDP} and voltage and in effect complete chip except the Hibernate domain will be powered off. On a wake-up event the Hibernate domain will dessert HIB control signal and enable generation of V_{DDP} voltage, hence complete power-up sequence of the core domain.

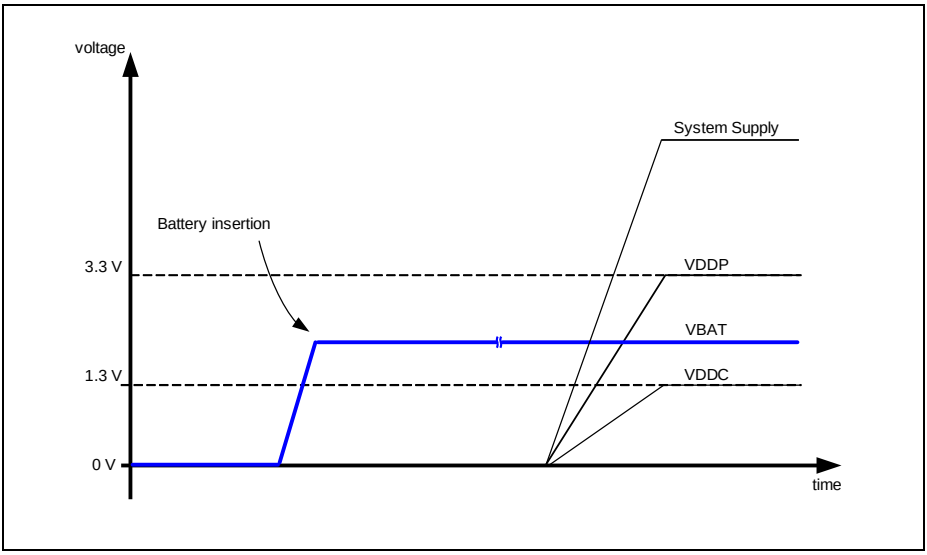


Figure 11-9 Initial power-up sequence

One of the valid power-up scenarios assumes that battery will be installed, possibly soldered, before core supply is available (see **Figure 11-9**). That would be a common situation after PCB assembly, before the complete device is installed in the target system. The battery voltage V_{BAT} gets connected before the main supply of the board is available and only the Hibernate domain is supplied. No current (only leakage is allowed) is drawn from the battery before explicit switching on Hibernate domain with software after Core domain power up using dedicated register **PWRSET**.HIBEN. This feature allows to keep the device in a storage for a long time before shipment to the end user, without significant loss of charge in the battery.

11.3.4 Embedded Voltage Regulator (EVR)

The EVR generates the core voltage V_{DDC} out of the external supplied voltage V_{DDP}. The EVR provides a supply watchdog (SWD) for the input voltage V_{DDP}. The generated core voltage V_{DDC} is monitored by a power validation circuit (PV).

11.3.5 Supply Watchdog (SWD)

Figure 11-10 shows the operation of the supply monitor. The supply watchdog compares the supply voltage against the reset threshold V_{POR}. The Data Sheet defines the nominal value and applied hysteresis.

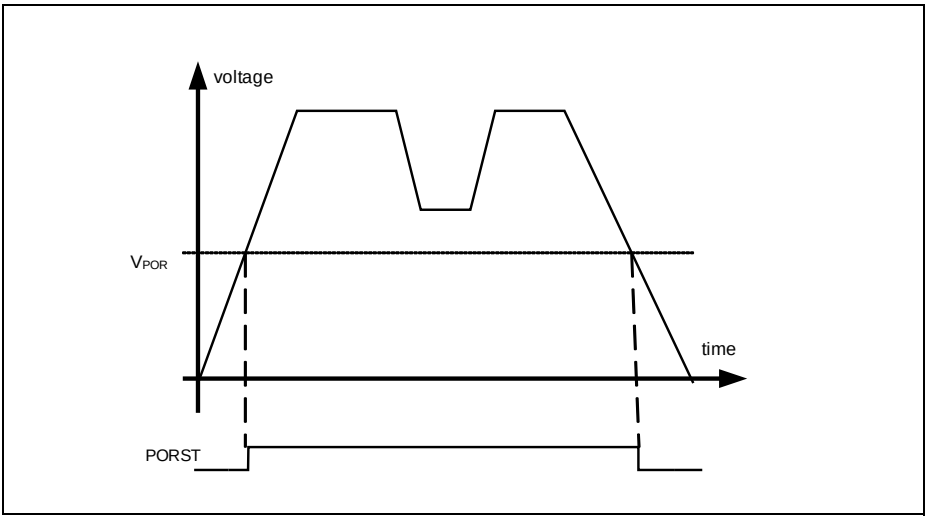


Figure 11-10 Supply Voltage Monitoring

While the supply voltage is below V_{POR} the device is held in reset. As soon as the voltage falls below V_{POR} a power on reset is triggered.

11.3.6 Power Validation (PV)

A power validation (PV) circuit monitors the internal core supply voltage of the core domain. It monitors that the core voltage is above the voltage threshold V_{PV} which guarantees save operation. Whenever the voltage falls below the threshold level a power-on reset is generated.

11.3.7 Supply Voltage Brown-out Detection

Brown-out detection is an additional voltage monitoring feature that enables the user software to perform some corrective action that bring the chip into safe operation in case a critical supply voltage drop and avoids system reset generated by the Supply Voltage Monitoring.

A drop of supply voltage to a critical threshold level programmed by the user can be signalized to the CPU with an NMI. An emergency corrective action may involve e.g. reduction of current consumption by switching of some modules or some interaction with external devices that should result in recovery of the supply voltage level.

Automatic monitoring of the voltage against programmed voltage allows efficient operation without a need for software interaction if the supply voltage remains at a safe

level. The supply voltage can be monitored also directly by software in register **EVRVADCSTAT**. The threshold value and the inspection interval is configured at **PWRMON**.

11.3.8 Hibernate Domain Power Management

The switch for the supply voltage of the Hibernate domain is depending of the voltage relation between V_{DDP} and V_{BAT}. It is strongly recommended to avoid supplying hibernate domain from a battery when valid V_{DDP} supply voltage is available in order to extend the battery life time. For XMC4500 product external supply voltage switching solution is based on Shottky diodes is proposed (see **Figure 11-11**).

11.3.9 Flash Power Control

The Flash module can be configured to operate in low power mode while in Deep Sleep state. Control of the Flash power mode it is performed using the **DSLEEPCR** register prior to entering the Deep Sleep mode. The user needs to trade the reduced leakage current against longer startup time.

11.4 Hibernate Control

Hibernate control is performed with Hibernate Control Unit (HCU).

In the operation mode with lowest power consumption only the hibernate domain remains powered up. In this state the hibernate domain enable switching off externally the main power supply (see **Figure 11-11**).

The HCU can request a complete power down of the core logic by programming HIB bit field of **HDCR** register.

The pin that controls external voltage regulator has a defined voltage level under all conditions. The pin remains at a default level if the hibernate domain is not enabled or not correctly powered.

Externally Controlled Hibernate Mode

External Voltage Regulator is controlled from Hibernate domain and V_{DDP} gets switched off when hibernate mode is entered. No reset of hibernate control occurs in hardware upon power-up but the chip will boot up as normal since in this mode no internal state of the chip is affected except for the hibernate control pin from hibernate domain to the External Voltage Regulator. Re-initialization of is possible with software upon boot-up.

Hibernation Support

The entry of the hibernate state is configured via the register **HDCR** by setting of the HIB bit. The HIBOUT bit in conjunction with selected HIBIOnPOL bit of **HDCR** register drives HIB_IO_n pad.

The hibernation control signal HIBOUT is connected to an open-drain pad to enable control of an external power regulator for V_{DDP}.

Wake-up from Hibernate Mode

The hibernation control supports multiple wake-up sources. Occurrence of any of the Wake-up from hibernate triggers resets of **HDCR**.HIB bit which results resetting of the HIBOUT signal that controls the External Voltage Regulator.

The additional wake-up sources shown as part of the Wake-up Unit will not be implemented in the XMC4500 and are shown here for future enhancements.

System Level Integration

The Power Control scheme require additional external devices in order to fully support defined functionality as shown in **Figure 11-11**.

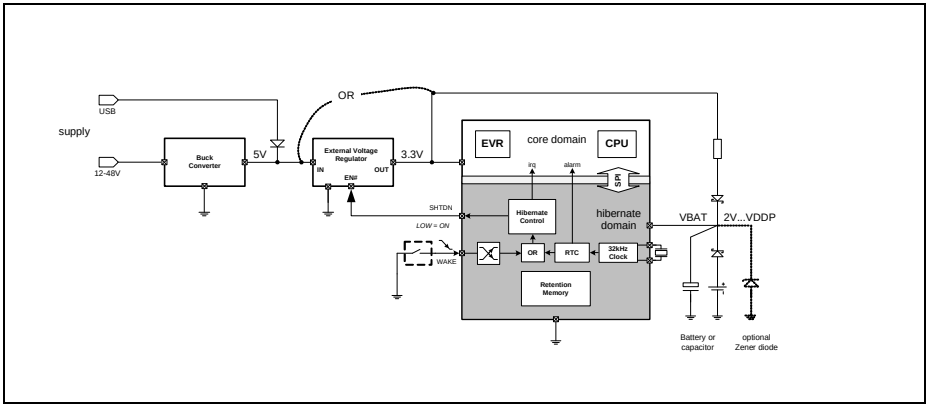


Figure 11-11 System Level Power Control Example

The external switch based on Shottky diode prevents discharging battery when voltage is supplied to the chip from the external power supply.

Hibernate Domain Pin Functions

Hibernate domain control register **HDCR** implements fields for alternate pin function selection.

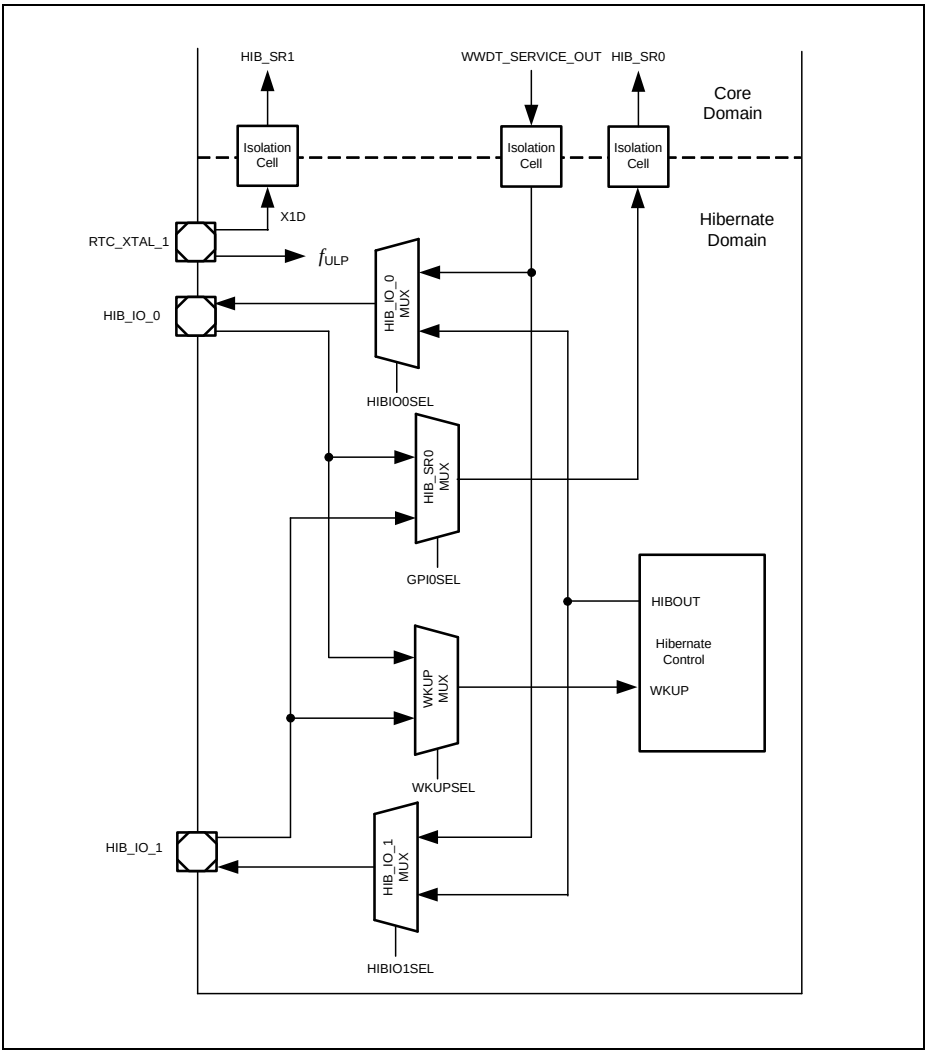


Figure 11-12 Alternate function selection of HIB_IO_0 and HIB_IO_1 pins of Hibernate Domain

The HIB_IO_0 pad is configured as open drain low and HIB_IO_1 as input after reset. This configuration enables use of low-enabled external voltage regulator as depicted in the **Figure 11-11**. In case of high-active external voltage regulator and battery presence

in the pins may be swapped to ensure safe startup of the system (additional external pull-up required on HIB_IO_1)

11.5 Reset Control

Reset Control Unit (RCU) performs control of all reset related functionality including:

- Reset assertion on various reset request sources
- Inspection of reset sources after reset
- Selective reset of peripherals

11.5.1 Supported Reset types

The XMC4500 has the following reset types:

- Power-on Reset, PORESETn
- System Reset, SYSRESETn
- Standby Reset, STDBYRESETn
- Debug Reset, DBGRESETn

Power-on Reset, PORESETn

A complete reset of the core domain of the device is executed upon power-up. Whenever the supply V_{DDP} is ramped-up and crossing the PORST voltage threshold the power-on reset is released. Additional a power-on reset is triggered by asserting the external PORST pin.

A Power-on reset is asserted again whenever the V_{DDP} voltage or the V_{DDC} voltage falls below defined reset thresholds.

System Reset, SYSRESETn

System reset is triggered by sources:

- Power-on reset
- Software reset via Cortex-M4 Application Interrupt and Reset Control Register (AIRCR)
- Lockup signal from Cortex-M4 when enabled at RCU
- Watchdog reset
- Memory Parity Error

A system reset resets almost all logic in the core domain. The only exceptions in the core domain are RCU Registers and Debug Logic.

Standby Reset, STDBYRESETn

The Hibernate domain including the RTC is only reset by a standby reset. A standby reset is triggered by a power-on reset specific to the Hibernate domain. Additionally a standby reset can be activated by software:

- Power-on reset specific to the Hibernate domain
- Software reset via **RSTSET** register

Debug Reset, DBGRESETn

Debug reset is triggered by the following sources:

- Debug Reset request from DAP while in mission mode and with debug probe present
- System Reset while in normal mode and debug probe not present

The Debug Reset is triggered by System Reset while in normal operation mode while debug probe is not present. When debug probe is present System Reset does not affect Debug Reset generation.

11.5.2 Peripheral Reset Control

Software can activate the reset of all peripherals individually via the registers **PRSET0**, **PRSET1**, **PRSET2** and **PRSET3**. The default state is that all peripherals are in reset after power-up. A return to the default state of a peripheral can be performed by forcing it to reset state by a separate reset.

The user needs to properly configure the port values before resetting a module to assure that the default output values of a peripheral do not harm external circuitry. Similarly the user has to take care of top-level interconnects, which will not be affected by a module specific reset.

11.5.3 Reset Status

The EVR provides the cause of a power reset to the RCU. The reset cause can be inspected after resuming operation by reading register **RSTSTAT**.

All register of the RCU undergo reset only by a power-on reset PORESETn.

Table 11-3 shows an overview of the reset signals their source and effects on the various parts of the system.

Table 11-3 Reset Overview

Name	Source	Core Domain	PAD Domain	Hibernate Domain	Debug & Trace System
PORESETn	EVR	yes	yes	no	yes
SYSRESETn	PORESETn Software WDT	yes	no	no	no
STDBYRESETn	Power-on Software	no	no	yes	no
DBGRESETn	DAP Software	no	no	no	yes

11.6 Clock Control

Clock generation and control is performed in the Clock Control Unit (CCU).

11.6.1 Block Diagram

The clock control unit has as major sub blocks the main Clock Generation Unit (CGU), the Standby Clock Generation Unit (SCGU) in the Hibernate power domain and the clock selection unit (CSU). The clock generation unit is connected to the external crystal pins XTAL1 and XTAL2. The clock generation unit provides in parallel three clocks to the clock selection unit, i.e. the USB PLL clock f_{USB}, the PLL output clock f_{PLL} and a fast internal generated clock f_{OFI} generated by the backup clock source. The clock selection unit receives as additional clocks sources a slow standby clock from the hibernate domain. The f_{STDBY} clock is one of the two clock sources of the Standby Clock Generation Unit (SGCU) in the hibernate domain. The standby clock is sourced by either the 32 kHz external crystal oscillator or by an internal 32 kHz clock source.

The clock selection unit provides various clocks to the modules of the system. The CSU allows to select the source of the clocks and to apply scaling of the frequency.

Major clock sources can be selected for driving the external clock pin EXTCLK to be observed externally.

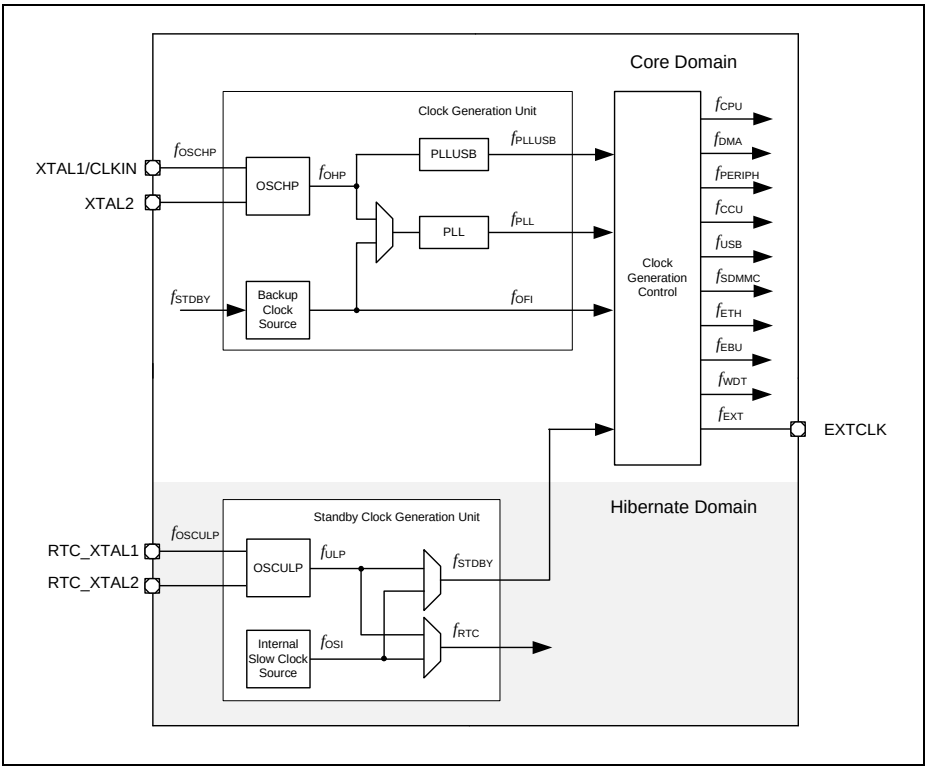


Figure 11-13 Clock Control Unit

11.6.2 Clock Sources

The system has four clock sources distributed over the core power domain and the hibernate power domain. The clock sources in the core power domain are:

- OSCHP, a high speed crystal oscillator to be used with an external clock signal or a crystal
- The backup clock source, f_{OFI}

During start-up, first the backup clock f_{OFI} is used. The f_{OFI} clock may be used during normal operation for low power operation.

The high precision oscillator OSCHP can be used with an external crystal or accepts an external clock source via pin XTAL1/CLKIN. Both the clock signals from OSCHP and the f_{OFI} can be selected as input clock source for the main PLL. The main PLL provides a maximum frequency of 520 MHz.

The clock sources in the hibernate power domain are:

- OSCULP, an ultra low power oscillator to be used with external crystal - 32kHz clock f_{ULP}
- Internal Slow Clock source 32 kHz clock f_{OSI}

The clocks drive the logic in the hibernate domain. In addition one of the two can be used as standby clock for low power operation in the core power domain.

To generate the required high precision f_{PLLUSB} clock for operation of the USB and the MMC/SD modules additional PLL can be optionally used if the desired clock frequency cannot be derived from the system PLL output f_{PLL}. The dedicated PLL referred to as PLLUSB, shown the block diagram in **Figure 11-14**.

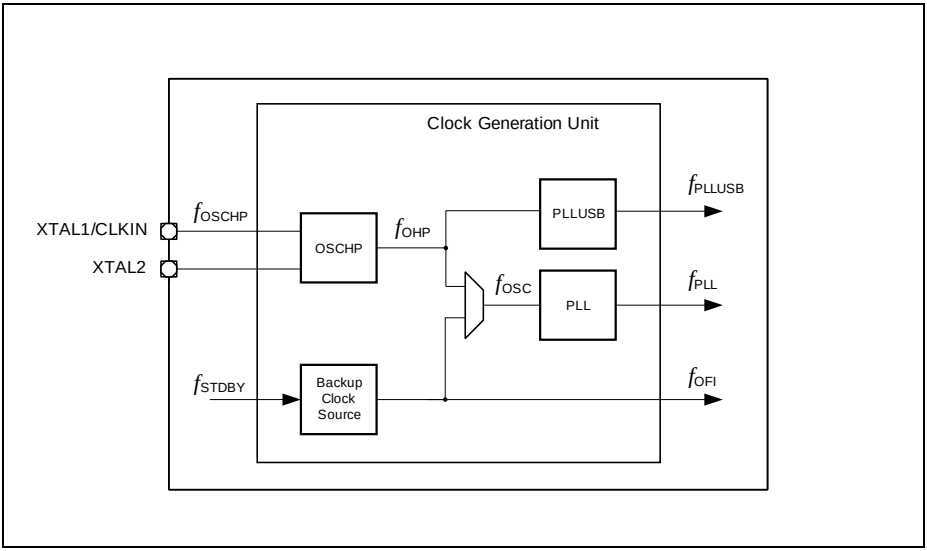


Figure 11-14 Clock Generation Block Diagram

11.6.3 Clock System Overview

The clock selection unit CSU provides the following clocks to the system:

Table 11-4 Clock Signals

Clock name	From/to module or pin	Description
System Clock Signals		
f_{SYS}	SCU.CCU	System master clock

Table 11-4 Clock Signals (cont'd)

Clock name	From/to module or pin	Description
f_{CPU}	CPU	CPU and NVIC clock
f_{DMA}	DMA0, DMA1	DMA clock
f_{PERIPH}	PBA0 PBA1	Peripheral clock for modules connected to peripheral bridges PBA0 and PBA1
f_{CCU}	CCU4 CCU8 POSIF	Clock for CCU4, CCU8 and POSIF modules
f_{USB}	USB	UTMI clock of USB
f_{ETH}	ETH0	ETH clock
f_{EBU}	EBU	EBU clock
f_{WDT}	WDT	Clock for independent watchdog timer
Internal Clock Signals		
f_{PLL}	System PLL	System PLL output clock
f_{PLLUSB}	USB PLL	USB PLL output clock
f_{OHP}	OSCHP	External crystal oscillator output clock
f_{OFI}	Internal Backup Clock Source	System Backup Block
f_{ULP}	OSCULP	External crystal slow oscillator output clock
f_{OSI}	Slow Internal Backup Clock Source	32 kHz backup clock for Hibernate domain
f_{OSC}	PLL input	PLL input clock
f_{STDBY}	Hibernate Domain	32kHz clock, optional WDT independent clock
f_{FLASH}	FLASH	Internal Flash clock
External Clock Signals		
f_{OSCHP}	XTAL1/CLKIN	External crystal input, optionally also direct input clock to system PLL

Table 11-4 Clock Signals (cont'd)

Clock name	From/to module or pin	Description
f_{OSCULP}	RTC_XTAL1	External crystal input, optionally also direct input clock to Hibernate domain and RTC module
f_{EXT}	Clock output to pin EXTCLK	Chip output clock

The clock sources of the clock selection unit are the four clocks from the clock generation unit, i.e. the USB clock f_{USB}, the PLL output clocks f_{PLL} and a fast internal generated clock f_{OFI}.The clock selection unit receives as additional clocks source a slow standby clock from the hibernate domain.

11.6.3.1 Clock System Architecture

The system clock f_{SYS} drives the CPU clock and all peripheral modules. It can be selected from the following clock sources:

- f_{PLL}, main PLL output clock divided by a 8-bit divider
- f_{OFI}, fast internal clock, bypassing PLL
- f_{OSCHP}, external clock, bypassing PLL
- f_{OHP}, external crystal oscillator clock, bypassing PLL

Please note that in dependence of the PLL mode setting the PLL output clock f_{PLL} is either a scaled version of the VCO clock in normal mode or a scaled version of one of the input clocks.

In Deep Sleep mode the system clocks can be switched to a clock which does not require PLL and thereby allowing the power down of the system PLL. This is either the fast internal clock or the slow standby clock. The **DSLEEPCR** register controls the clock settings for Deep Sleep mode.

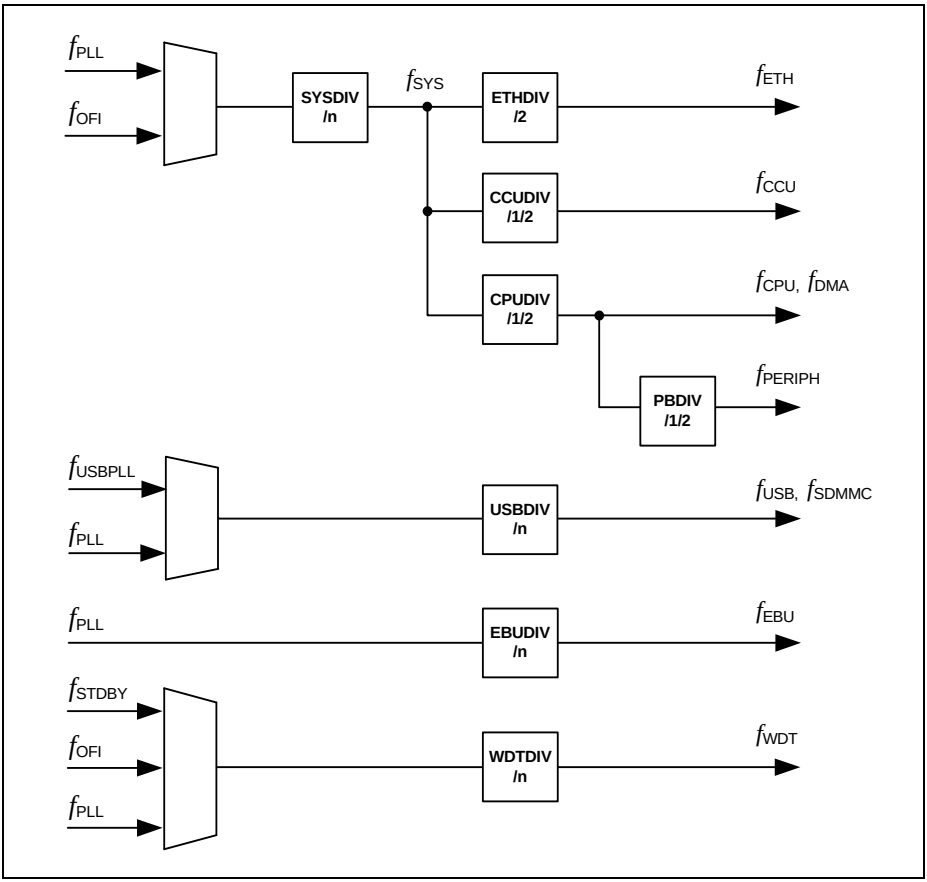


Figure 11-15 Clock Generation Control

Some limitations apply on clock ratio combinations between f_{CCU}, f_{CPU} and f_{PERIPH}. Only divider setting listed in the table **Table 11-5** are allowed for the f_{CCU}, f_{CPU} and f_{PERIPH} clocks. All other clock dividers settings must be prohibited by software applications in order to avoid invalid clock ratios leading to system malfunctions. The **Table 11-5** table shows also example clock rate values assuming f_{SYS} rate of 120 MHz.

Table 11-5 Valid values of clock divide registers for f_{CCU}, f_{CPU} and f_{PERIPH} clocks

CCUCLKCR.CCUDIV	CPUCLKCR.CPUDIV	PBCLKCR.PBDIV
0 (120 MHz)	0 (120 MHz)	0 (120 MHz)
0 (120 MHz)	0 (120 MHz)	1 (60 MHz)

Table 11-5 Valid values of clock divide registers for f_{CCU}, f_{CPU} and f_{PERIPH} clocks

CCUCLKCR.CCUDIV	CPUCLKCR.CPUDIV	PBCLKCR.PBDIV
0 (120 MHz)	1 (60 MHz)	0 (60 MHz)
1 (60 MHz)	0 (120 MHz)	1 (60 MHz)
1 (60 MHz)	1 (60 MHz)	0 (60 MHz)

USB and MMC/SD Clock Selection

The clock for the USB module and the clock for the MMC/SD module are both derived from the clock f_{USB} or from the clock f_{PLL} of the main PLL. The later is only feasible, when the main PLL is providing a frequency which allows to generate the 48MHz with a 3-bit divider.

The USB clock is automatically gated and the USB PLL put in power-down by the USB suspend signal.

The clock divider and the MUX must only be configured while the USB and MMC/SD are not enabled.

Ethernet Clock Selection

The Ethernet module receives SRAM clock f_{SYS} which has to be twice the frequency of the ETH MAC internal clock f_{ETH}. The SRAM clock rate must always be above 100 MHz which guarantee that the clock of ETH MAC is of rate above its acceptable minimum of 50 MHz.

Both clocks are disabled with software in Wake-On-Lan mode. The clocks have to be enabled via software when the related interrupt the end of Wake-On-Lan mode.

The clock divider and the clock MUX must only be configured while ETH module is not enabled.

CPU Clock Selection

The CPU clock f_{CPU} may be equal to or a half of the system clock f_{SYS}.

Peripheral Bus Clock Selection

The Peripheral Bus clock is derived from the f_{CPU} clock and may be equal t or a half of the f_{CPU}. There are some limitations imposed on peripheral bus configuration in term of clock ratio with respect to the f_{CCU} clock. The Peripheral Bus clock rate must never be less than half and more than the f_{CCU} clock.

CAPCOM and POSIF Clock Selection

The capture compare blocks CCU4, CCU8 and POSIF use as timer clock the clock f_{CCU} derived from system f_{SYS} via CCUDIV clock divider. The clock divider allows to adjust clock frequency of the timers with respect to the rest of the system. Relation between f_{CCU} clock and other clocks in the system is constrained as described in the **Table 11-5**.

The f_{CCU} clock frequency must only be configured while all CAPCOM and POSIF modules are not enabled.

Watchdog Clock Selection

The watchdog module uses as independent clock either the internal fast clock f_{OFI} or the slow clock f_{STDBY}. The system clock f_{PLL} is available as additional clock source.

Both the clock divider and the clock MUX must only be configured while the WDT is not enabled.

External Clock Output

An external clock is provided via clock pin EXTCLK. All clock sources can be selected as clock signal for the external clock output. Optionally a divider can be used before bringing the system clock to the outside to stay within the limit of the supported frequencies for the pads.

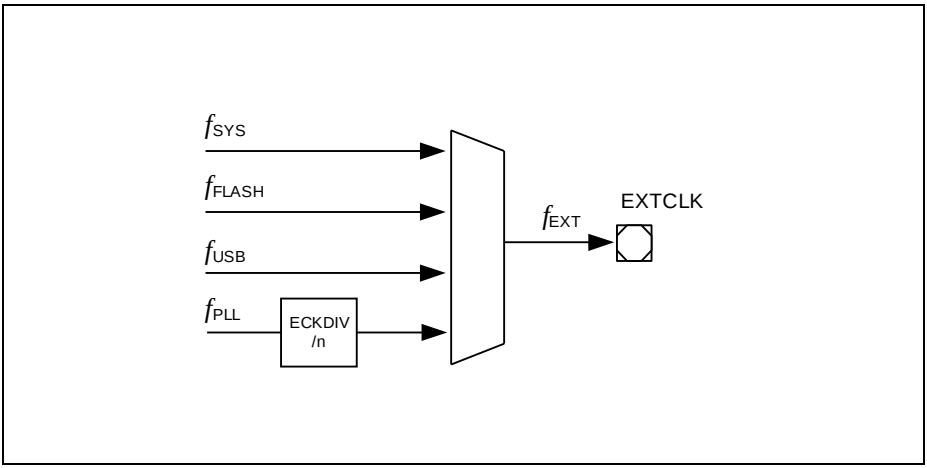


Figure 11-16 External Clock Selection

11.6.4 High Precision Oscillator Circuit (OSCHP)

The high precision oscillator circuit can drive an external crystal or accepts an external clock source. It consists of an inverting amplifier with XTAL1 as input, and XTAL2 as output.

Figure 11-18 and **Figure 11-17** show the recommended external circuitries for both operating modes, External Crystal Mode and External Input Clock Mode.

External Input Clock Mode

In this usage an external clock signal is supplied directly not using an external crystal and bypassing the amplifier of the oscillator. The input frequency must be in the range from 4 to 40 MHz.When using an external clock signal it must be connected to XTAL1. XTAL2 is left open i.e. unconnected.

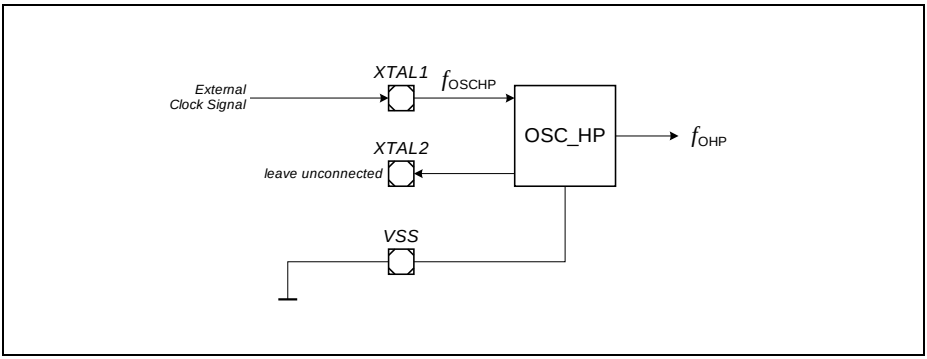


Figure 11-17 External Clock Input Mode for the High-Precision Oscillator

External Crystal Mode

For the external crystal mode an external oscillator load circuitry is required. The circuitry must be connected to both pins, XTAL1 and XTAL2. It consists normally of the two load capacitances C1 and C2. For some crystals a series damping resistor might be necessary. The exact values and related operating range depend on the crystal and have to be determined and optimized together with the crystal vendor using the negative resistance method.

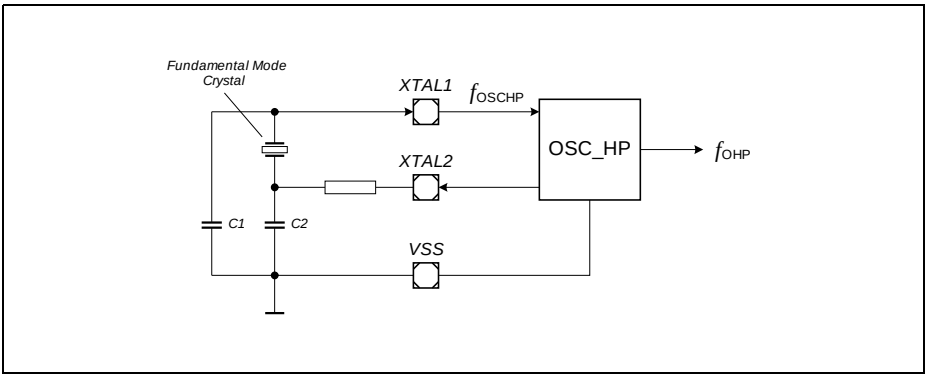


Figure 11-18 External Crystal Mode Circuitry for the High-Precision Oscillator

11.6.5 Backup Clock Source

The backup clock f_{OFI} generated internally is the default clock after start-up. It is used for bypassing the PLL for startup of the system without external clock. Furthermore it can be used as independent clock source for the watchdog module or even as system clock

source during normal operation. While in prescaler mode this clock is automatically used as emergency clock if the external clock failure is detected.

Clock adjustment is required to reach desired level of f_{OFI} precision. The backup clock source provides two adjustment procedures:

- loading of adjustment value during start-up
- continuos adjustment using the high-precision f_{STDBY} clock as reference

11.6.6 Main PLL

The main PLL converts a low-frequency external clock signal to a high-speed internal clock. The PLL also has fail-safe logic that detects degenerative external clock behavior such as abnormal frequency deviations or a total loss of the external clock. The PLL triggers autonomously emergency action if it loses its lock on the external clock and switches to the Fast Internal Backup Clock.

This module is a phase locked loop for integer frequency synthesis. It allows the use of input and output frequencies of a wide range by varying the different divider factors.

11.6.6.1 Features

- VCO lock detection
- 4-bit input divider **P:** (divide by PDIV+1)
- 7-bit feedback divider **N**: (multiply by NDIV+1)
- 7-bit output dividers **K1 or K2:** (divide by KxDIV+1)
- Oscillator Watchdog
  - Detecting too low input frequencies
  - Detecting too high input frequencies
  - Spike detection for the OSC input frequency
- Different operating modes
  - Bypass Mode
  - Prescaler Mode
  - Normal Mode
- VCO Power Down
- PLL Power Down
- Glitch less switching between K-Dividers
- Switching between Normal Mode and Prescaler Mode

11.6.6.2 System PLL Functional Description

The PLL consists of a Voltage Controlled Oscillator (VCO) with a feedback path. A divider in the feedback path (N-Divider) divides the VCO frequency down. The resulting frequency is then compared with the externally provided and divided frequency (P-Divider). The phase detection logic determines the difference between the two clocks

and accordingly controls the frequency of the VCO (f_{VCO}). A PLL lock detection unit monitors and signals this condition. The phase detection logic continues to monitor the two clocks and adjusts the VCO clock if required.

The following figure shows the PLL block structure.

Clock Source Control

The input clock for the PLL f_{OSC} can be one of the following two clock sources:

- The internal generated fast clock f_{OFI}
- The clock f_{OHP} sourced by the crystal oscillator OSCHP

The PLL clock f_{PLL} is generated from the input clock in one of two software selectable operation modes:

- Normal Mode, using VCO output clock
- Prescaler Mode, using input clock directly

The PLL output clock f_{PLL} is derived from either

- VCO clock divided by the K2-Divider, normal mode
- external oscillator clock divided by the K1-Divider, prescaler mode
- backup clock divided by the K1-Divider, prescaler mode

The PLL clock f_{PLL} is generated in emergency from one of two sources:

- Free running VCO if Emergency entered from Normal Mode of PLL
- Backup Clock if emergency entered from Prescaler Mode of PLL

Configuration and Operation of the Normal Mode

In Normal Mode, the PLL is running at the frequency f_{OSC} and f_{PLL} is divided down by a factor P, multiplied by a factor N and then divided down by a factor K2.

The output frequency is given by:

(11.1)

$$f_{PLL} = \frac{N}{P \cdot K_2} \cdot f_{OSC}$$

The maximum frequency in normal mode is 520 MHz corresponding to the maximum supported VCO frequency.

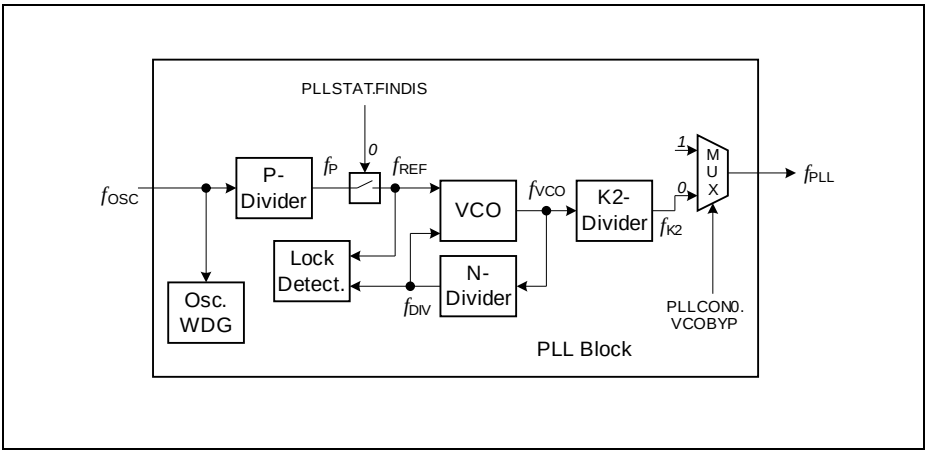


Figure 11-19 PLL Normal Mode

The Normal Mode is selected by the following settings

- Register Setting
  - PLLCON0.VCOBYP = 0
  - PLLCON0.FINDIS = 0

The Normal Mode is entered when the following requirements are all together valid:

- Register Values
  - PLLCON0.FINDIS = 0
  - PLLSTAT.VCOBYST = 0
  - PLLSTAT.VCOLOCK = 1
  - PLLSTAT.PLLLV = 1
  - PLLSTAT.PLLLH = 1

Operation on the Normal Mode does require an input clock frequency of f_{OSC}. Therefore it is recommended to check and monitor if an input frequency f_{OSC} is available at all by checking PLLSTAT.PLLLV. For a better monitoring also the upper frequency can be monitored via PLLSTAT.PLLHV.

For the Normal Mode there is the following requirement regarding the frequency of f_{OSC}.

A modification of the two dividers P and N has a direct influence to the VCO frequency and lead to a loss of the VCO Lock status. A modification of the K2-divider has no impact on the VCO Lock status.

When the frequency of the Normal Mode should be modified or entered the following sequence needs to be followed:

- Configure and enter Prescaler Mode
- Disable NMI trap generation for the VCO Lock

- Configure Normal Mode
- Wait for a positive VCO Lock status (PLLSTAT.VCOLOCK = 1).
- Switch to Normal Mode by clearing PLLCON.VCOBYP

The Normal Mode is entered when the status bit PLLSTAT.VCOBYST is cleared.

When the Normal Mode is entered, the NMI status flag for the VCO Lock trap should be cleared and then enabled again. The intended PLL output target frequency can now be configured by changing only the K2-Divider. This can result in multiple changes of the K2-Divider to avoid to big frequency changes. Between the update of two K2-Divider values 6 cycles of f_{PLL} should be waited. For ramping up PLL output frequency in Normal Mode the following steps are required:

- The first target frequency of the Normal Mode should be selected in a way that it matches or is only slightly higher as the one used in the Prescaler Mode. This avoids big changes in the system operation frequency and therefore power consumption when switching later from Prescaler Mode to Normal Mode.
- Selecting P and N in a way that f_{VCO} is in the upper area of its allowed values leads to a slightly increased power consumption but to a slightly reduced jitter
- Selecting P and N in a way that f_{VCO} is in the lower area of its allowed values leads to a slightly reduced power consumption but to a slightly increased jitter
- It is recommended to reset the f_{VCO} Lock detection (PLLCON0.RESLD = 1) after the new values of the dividers are configured to get a defined VCO lock check time.

Depending on the selected divider value of the K2-Divider the duty cycle of the clock is selected. This can have an impact for the operation with an external communication interface. The duty cycles values for the different K2-divider values are defined in the Data Sheet.

PLL VCO Lock Detection

The PLL has a lock detection that supervises the VCO part of the PLL in order to differentiate between stable and unstable VCO circuit behavior. The lock detector marks the VCO circuit and therefore the output f_{VCO} of the VCO as instable if the two inputs f_{REF} and f_{DIV} differ too much. Changes in one or both input frequencies below a level are not marked by a loss of lock because the VCO can handle such small changes without any problem for the system.

PLL VCO Loss-of-Lock Event

The PLL may become unlocked, caused by a break of the crystal or the external clock line. In such a case, an NMI trap is generated if it were enabled. Additionally, the OSC clock input f_{OSC} is disconnected from the PLL VCO to avoid unstable operation due to noise or sporadic clock pulses coming from the oscillator circuit. Without a clock input f_{OSC}, the PLL gradually slows down to its VCO base frequency and remains there. This default feature can be disabled by setting bit PLLCON0.OSCDISCDIS. If this bit is set the OSC clock remains connected to the VCO.

11.6.6.3 Configuration and Operation of the Prescaler Mode

In Prescaler Mode, the PLL is running at the external frequency f_{OSC} and f_{PLL1} is derived from f_{OSC} only by the K1-Divider.

The output frequency is given by

$$f_{PLL} = \frac{f_{OSC}}{K_1} \quad (11.2)$$

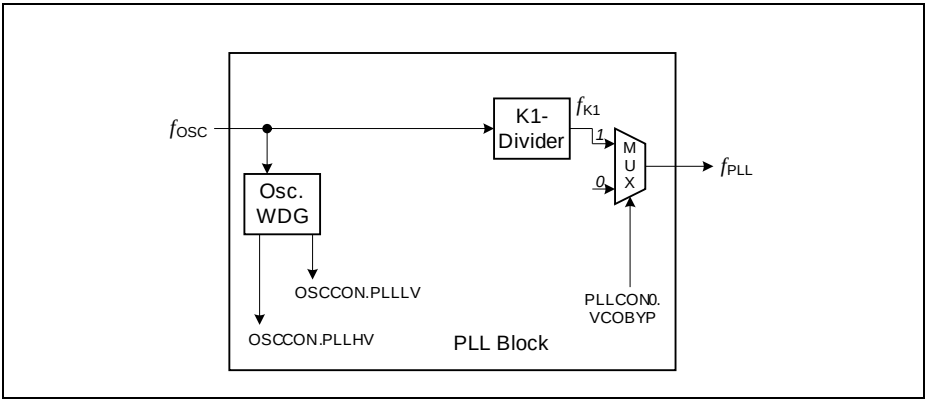


Figure 11-20 PLL Prescaler Mode Diagram

The Prescaler Mode is selected by the following settings

- PLLCON0.VCOBYP = 1

The Prescaler Mode is entered when the following requirements are all together valid:

- PLLSTAT.VCOBYST = 1
- PLLSAT.PLLLV = 1

Operation on the Prescaler Mode does require an input clock frequency of f_{OSC}. Therefore it is recommended to check and monitor if an input frequency f_{OSC} is available at all by checking PLLSAT.PLLLV. For a better monitoring also the upper frequency can be monitored via PLLSAT.PLLHV.

For the Prescaler Mode there are no requirements regarding the frequency of f_{OSC}.

The system operation frequency is controlled in the Prescaler Mode by the value of the K1-Divider. When the value of PLLCON1.DIV was changed the next update of this value should not be done before bit PLLSTAT.K1RDY is set.

Depending on the selected divider value of the K1-Divider the duty cycle of the clock is selected. This can have an impact for the operation with an external communication interface. The duty cycles values for the different K1-divider values are defined in the Data Sheet.

The Prescaler Mode is requested from the Normal Mode by setting bit PLLCON.VCOBYP. The Prescaler Mode is entered when the status bit PLLSTAT.VCOBYST is set. Before the Prescaler Mode is requested the K1-Divider should be configured with a value generating a PLL output frequency f_{PLL} that matches the one generated by the Normal Mode as much as possible. In this way the frequency change resulting out of the mode change is reduced to a minimum.

The Prescaler Mode is requested to be left by clearing bit PLLCON.VCOBYP. The Prescaler Mode is left when the status bit PLLSTAT.VCOBYST is cleared.

11.6.6.4 Bypass Mode

The bypass mode is used only for testing purposes. In Bypass Mode the input clock f_{OSC} is directly connected to the PLL output f_{PLL}.

The output frequency is given by:

(11.3)

$$f_{PLL} = f_{OSC}$$

11.6.6.5 System Oscillator Watchdog (OSC_WDG)

The oscillator watchdog monitors the incoming clock frequency f_{OSC} from OSCHP or f_{OFI}. A stable and defined input frequency is a mandatory requirement for operation in both Prescaler Mode and Normal Mode. In addition for the Normal Mode it is required that the input frequency f_{OSC} is in a certain frequency range to obtain a stable master clock from the VCO part.

The expected input frequency is selected via the bit field **OSCHPCTRL**.OSCVAL. The OSC_WDG checks for spikes, too low frequencies, and for too high frequencies.

The frequency that is monitored is f_{OSCREF} which is derived from f_{OSC}.

(11.4)

$$f_{OSCREF} = \frac{f_{OSC}}{OSCVAL + 1}$$

The divider value **OSCHPCTRL**.OSCVAL has to be selected in a way that f_{OSCREF} is 2.5 MHz.

Note: f_{OSCREF} has to be within the range of 2 MHz to 3 MHz and should be as close as possible to 2.5 MHz.

The monitored frequency is too low if it is below 1.25 MHz and too high if it is above 7.5 MHz. This leads to the following two conditions:

- Too low: f_{OSC} < 1.25 MHz × (**OSCHPCTRL**.OSCVAL+1)
- Too high: f_{OSC} > 7.5 MHz × (**OSCHPCTRL**.OSCVAL+1)

Before configuring the OSC_WDG function all the trap options should be disabled in order to avoid unintended traps. Thereafter the value of **OSCHPCTRL**.OSCVAL can be changed. Then the OSC_WDG should be reset by setting PLLCON0.OSCVAL. This requests the start of OSC_WDG monitoring with the new configuration. When the expected positive monitoring results of PLLSAT.PLLLV and / or PLLSAT.PLLHV are set the input frequency is within the expected range. As setting PLLCON0.OSCVAL clears all three bits PLLSAT.PLLSP, PLLSAT.PLLLV, and PLLSAT.PLLHV all three trap status flags will be set. Therefore all three flags should be cleared before the trap generation is enabled again. The trap disabling-clearing-enabling sequence should also be used if only bit PLLCON0.OSCVAL is set without any modification of **OSCHPCTRL**.OSCVAL.

11.6.6.6 VCO Power Down Mode

The PLL offers a VCO Power Down Mode. This mode can be entered to save power within the PLL. The VCO Power Down Mode is entered by setting bit PLLCON0.VCOPWD. While the PLL is in VCO Power Down Mode only the Prescaler Mode is operable. Please note that selecting the VCO Power Down Mode does not automatically switch to the Prescaler Mode. So before the VCO Power Down Mode is entered the Prescaler Mode must be active.

11.6.6.7 PLL Power Down Mode

The PLL offers a Power Down Mode. This mode can be entered to save power if the PLL is not required. The Power Down Mode is entered by setting bit PLLCON0.PLLPWD. While the PLL is in Power Down Mode no PLL output frequency is generated.

11.6.7 USB PLL

The USB PLL serves the special purpose to provide an accurate 48MHz clock for USB 2.0 full-speed operation. The basic functionality of the USB PLL is similar to the main PLL (see **Figure 11-21**).

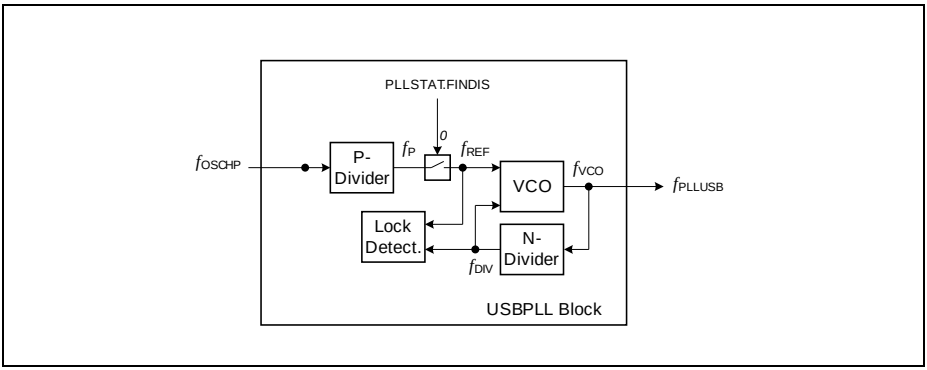


Figure 11-21 USB PLL Block Diagram

Below table shows the required values of N divider for different crystal values.

Table 11-6 USB support and N Divider Values for crystal frequencies

Crystal frequency [MHz]	N Divider Value
4	24
6	16
8	12
12	8
16	6

The USB PLL is put automatically in power-down by the USB suspend signal, when the clock is not used for SD/MMC operation.

Note: Reconfiguration of the P-Divider before USB-PLL has locked must be avoided.

11.6.8 Ultra Low Power Oscillator

The ultra low power oscillator is providing a real time clock source of 32 kHz when paired with an external crystal. It operates in the supply voltage range of the Hibernate power domain. However a secure start-up of the oscillator requires a power-up of the device with a power supply above 3.0 V present. The crystal pads are always powered by V_{BAT} or V_{DDP}. It is further used for continuos adjustment of the fast internal backup clock source f_{OFI}.

When paired with an appropriate crystal the real time clock has a very high precision of +/- 1 sec. / week.

The precise and stable 32kHz clock propagated to Core domain as f_{STDBY} can be used for continuos adjustments the fast internal backup clock source f_{OFI}.

11.6.9 Slow Internal Clock Source

The slow internal clock source provides a clock f_{OSI} of 32 kHz. This clock can be used as independent clock source for WDT module and as the clock for periodic wake-up events in power saving modes.

11.6.9.1 OSCULP Oscillator Watchdog (ULPWDG)

The slow oscillator watchdog monitors the incoming clock frequency f_{ULP} from OSCULP. A reliable32kHz clock is required in the Hibernate domain in Hibernate state in order to perform system wake-up upon occurrence of configured events. In order to ensure that a clock watchdog is required to continuously monitor the enabled clock sources. In case of external crystal failure the clock source must switch to the It is further used for continuos adjustments the fast internal backup clock source f_{OFI}. Occurrence of the switch must be indicated in the **HDSTAT** register.

11.6.10 Clock Gating Control

The clock to peripherals can be individually gated and parts of the system stopped by these means.

Clock gating in Sleep and Deep-Sleep modes

Global power management related to clock generation and selection is supported for the sleep modes of the system. The user has full control on the clock configuration for these modes. The registers **SLEEPCR** and **DSLEEPCR** control which clocks should remain active and which are to be gated by entering the corresponding Sleep or Deep Sleep mode. Furthermore the system clock can be switched to a slow standby clock and the PLLs put into power-down mode in Deep Sleep mode.

11.7 Debug Behavior

The SCU module does not get affected with the HALTED signal from SCU upon debug activities performed using external debug probe.

11.8 Power, Reset and Clock

The SCU module implements functions that involve various types of modules controlled directly or via dedicated interfaces that are instantiated in different power, clock and reset domains. These modules are functionally considered parts of the SCU and therefore SCU is also considered a multi domain circuit in this sense.

Power domains:

Power domains get separated with appropriate power separation cells.

- Core domain supplied with V_{DDC} voltage

- Pad domain supplied with V_{DDP} voltage
- Hibernate domain supplied with V_{BAT} (optional external battery, capacitor) voltage

Clock domains:

All cross-domain interfaces implement signal synchronization.

- internal SCU clock is f_{SYS} always identical to the CPU clock and selectable within SCU
- HCU, RTC and register mirror interface clock is 32.786 kHz clock generated in hibernate domain

Reset domains:

All resets get internally synchronized to respective clocks.

- System Reset (SYSRESETn) resets most of the logic in SCU and can be triggered from various sources (please refer to **Reset Control** chapter for more details)
- Power-on Reset (PORESETn) resets directly a few registers and contributes in generation of the System Reset an gets triggered upon power-up sequence of the Core domain
- Standby Reset (STDBYRESETn) resets HCU part of SCU, get triggered by power-up sequence of Hibernate domain and is not affected by power-up sequence of the Core domain
- Debug Reset (DBGRESETn) is used in various debug or test related scenarios (please refer to **Reset Control** chapter for more details)

11.9 Initialization and System Dependencies

The following system dependencies should be considered.

Initialization

The system initialization requires that all required clocks are enabled and the relevant modules get released from reset state. After system reset release a number of modules clocked with f_{PERIPH} still remain in reset state. Reset release of those modules is controlled with **PRCLR0**, **PRCLR1**, **PRCLR2** and **PRCLR3**.

The system relevant clocks are configured with the dedicated registers **SYSCLKCR**, **CPUCLKCR** and **PBCLKCR**. Some modules require explicit clock enable with **CLKSET** register. For details please refer to **“Reset Control” on Page 11-22**

After reset release the system is clocked with a clock derived from the Backup Clock source. If a PLL output clock is required as the system clock source then it is necessary to initialize the respective PLL with a software routine. For details please refer to PLL section in **“Main PLL” on Page 11-34**.

Auxiliary functionality of the system control, like Memory Parity, Die Temperature Sensor, Out of Range Comparators for analog inputs can be activated with the registers defined in GCU section of the **“GCU Registers” on Page 11-50**

The Watchdog Timer requires the activation of an independent clock. For more details please refer to WDT chapter.

Hibernate Control logic and the RTC module needs to be activated with **PWRSET** register before it can be used. This initialization has to be performed only once if the V_{BAT} is newly applied and it stays enabled until V_{BAT} is removed. After power off of the chips the hibernate domain will remain intact while V_{BAT} is still supplied. For details of hibernate control please refer to **“Hibernate Control” on Page 11-19**. For details of RTC module control please refer o RTC chapter.

11.10 Registers

This section describes the registers of SCU. Most of the registers are reset SYSRESETn reset signal but some of the registers can be reset only with PORST reset.

Table 11-7 Base Addresses of sub-sections of SCU registers

Short Name	Description	Offset Addr.[1)]
GCU Registers	Offset address of General Control Unit	0000_H
PCU Registers	Offset address of Power Control Unit	0200_H
HCU Registers	Offset address of Hibernate Control Unit	0300_H
RCU Registers	Offset address of Reset Control Unit	0400_H
CCU Registers	Offset address of Clock Control Unit	0600_H

1) The absolute register address is calculated as follows:
Module Base Address + Sub-Module Offset Address (shown in this column) + Register Offset Address

Following access result an AHB error response:

- Read or write access to undefined address
- Write access in user mode to registers which allow only privileged mode write
- Write access to read-only registers
- Write access to startup protected registers

Table 11-8 Registers Address Space

Module	Base Address	End Address	Note
SCU	5000 4000_H	5000 $7FFF_H$	System Control Unit Registers

Table 11-9 Registers Overview

Short Name	Register Long Name	Offset Addr. 1)	Access Mode		Description
			Read	Write	
General SCU Registers					
GCU Registers					
ID	Module Identification Register	0000_H	U, PV	BE	Page 11-50
IDCHIP	Chip ID	0004_H	U, PV	BE	Page 11-51

Table 11-9 Registers Overview (cont'd)

Short Name	Register Long Name	Offset Addr. [1]	Access Mode		Description
			Read	Write	
IDMANUF	Manufactory ID	0008_H	U, PV	BE	Page 11-51
STCON	Start-up Control	0010_H	U, PV	PV	Page 11-52
GPR0	General Purpose Register 0	$002C_H$	U, PV	PV	Page 11-53
GPR1	General Purpose Register 1	0030_H	U, PV	PV	Page 11-53
ETH0_CON	Ethernet 0 Port Control	0040_H	U, PV	PV	Page 11-54
CCUCON	CCUx Global Start Control Register	$004C_H$	U, PV	U	Page 11-56
SRSTAT	Service Request Status	0074_H	U, PV	U, PV	Page 11-58
SRRAW	RAW Service Request Status	0078_H	U, PV	BE	Page 11-60
SRMSK	Service Request Mask	$007C_H$	U, PV	PV	Page 11-62
SRCLR	Service Request Clear	0080_H	nBE	PV	Page 11-64
SRSET	Service Request Set	0084_H	nBE	PV	Page 11-66
NMIREQEN	Enable Promoting Events to NMI Request	0088_H	U, PV	PV	Page 11-68
DTSCON	DTS Control	$008C_H$	U, PV	PV	Page 11-69
DTSSTAT	DTS Status	0090_H	U, PV	BE	Page 11-71
SDMMCDEL	SD-MMC Delay Control Register	$009C_H$	U, PV	U	Page 11-72
G0ORCEN	Out-Of-Range Comparator Enable Register	$00A0_H$	U, PV	U, PV	Page 11-72
G1ORCEN	Out-Of-Range Comparator Enable Register	$00A4_H$	U, PV	U, PV	Page 11-73
MIRRSTS	Mirror Update Status Register	$00C4_H$	U, PV	BE	Page 11-74
RMACR	Retention Memory Access Control Register	$00C8_H$	U, PV	U, PV	Page 11-76

Table 11-9 Registers Overview (cont'd)

Short Name	Register Long Name	Offset Addr. [1]	Access Mode		Description
			Read	Write	
RMADATA	Retention Memory Access Data Register	$00CC_H$	U, PV	U, PV	Page 11-77
PEEN	Parity Error Enable Register	$013C_H$	U, PV	PV	Page 11-78
MCHKCON	Memory Checking Control Register	0140_H	U, PV	PV	Page 11-79
PETE	Parity Error Trap Enable Register	0144_H	U, PV	PV	Page 11-81
PERSTEN	Reset upon Parity Error Enable Register	0148_H	U, PV	PV	Page 11-83
PEFLAG	Parity Error Control Register	0150_H	U, PV	PV	Page 11-84
PMTPR	Parity Memory Test Pattern Register	0154_H	U, PV	PV	Page 11-86
PMTSR	Parity Memory Test Select Register	0158_H	U, PV	PV	Page 11-87
TRAPSTAT	Trap Status Register	0160_H	U, PV	BE	Page 11-89
TRAPRAW	Trap Raw Status Register	0164_H	U, PV	BE	Page 11-91
TRAPDIS	Trap Mask Register	0168_H	U, PV	PV	Page 11-92
TRAPCLR	Trap Clear Register	$016C_H$	nBE	PV	Page 11-93
TRAPSET	Trap Set Register	0170_H	nBE	PV	Page 11-95
PCU Registers					
PWRSTAT	Power Status Register	0000_H	U, PV		Page 11-96
PWRSET	Power Set Control Register	0004_H	nBE	PV	Page 11-97
PWRCLR	Power Clear Control Register	0008_H	nBE	PV	Page 11-98
EVRSTAT	EVR Status Register	0010_H	U, PV	BE	Page 11-99
EVRVADCSTAT	EVR VADC Status Register	0014_H	U, PV	BE	Page 11-99
PWRMON	Power Monitor Value	$002C_H$	U, PV	PV	Page 11-100

Table 11-9 Registers Overview (cont'd)

Short Name	Register Long Name	Offset Addr. [1]	Access Mode		Description
			Read	Write	
HCU Registers					
HDSTAT	Hibernate Domain Status Register	0000_H	U, PV	PV	Page 11-101
HDCLR	Hibernate Domain Status Clear Register	0004_H	U, PV	PV	Page 11-102
HDSET	Hibernate Domain Status Set Register	0008_H	U, PV	PV	Page 11-103
HDCR	Hibernate Domain Control Register	$000C_H$	U, PV	PV	Page 11-104
OSCSICTRL	Internal 32 kHz Clock Source Control Register	0014_H	U, PV	PV	Page 11-107
OSCULSTAT	OSCULP Status Register	0018_H	U, PV	BE	Page 11-107
OSCULCTRL	OSCULP Control Register	$001C_H$	U, PV	PV	Page 11-108
RCU Registers					
RSTSTAT	System Reset Status	0000_H	U, PV	BE	Page 11-109
RSTSET	Reset Set Register	0004_H	nBE	PV	Page 11-110
RSTCLR	Reset Clear Register	0008_H	nBE	PV	Page 11-111
PRSTAT0	Peripheral Reset Status Register 0	$000C_H$	U, PV	PV	Page 11-112
PRSET0	Peripheral Reset Set Register 0	0010_H	nBE	PV	Page 11-114
PRCLR0	Peripheral Reset Clear Register 0	0014_H	nBE	PV	Page 11-116
PRSTAT1	Peripheral Reset Status Register 1	0018_H	U, PV	BE	Page 11-117
PRSET1	Peripheral Reset Set Register 1	$001C_H$	nBE	PV	Page 11-119
PRCLR1	Peripheral Reset Clear Register 1	0020_H	nBE	PV	Page 11-120
PRSTAT2	Peripheral Reset Status Register 2	0024_H	U, PV	BE	Page 11-121

Table 11-9 Registers Overview (cont'd)

Short Name	Register Long Name	Offset Addr. [1]	Access Mode		Description
			Read	Write	
PRSET2	Peripheral Reset Set Register 2	0028_H	nBE	PV	Page 11-123
PRCLR2	Peripheral Reset Clear Register 2	$002C_H$	nBE	PV	Page 11-124
PRSTAT3	Peripheral Reset Status Register 3	0030_H	U, PV	BE	Page 11-125
PRSET3	Peripheral Reset Set Register 3	0034_H	BE	PV	Page 11-126
PRCLR3	Peripheral Reset Clear Register 3	0038_H	BE	PV	Page 11-126
CCU Registers					
CLKSTAT	Clock Status Register	0000_H	U,PV	BE	Page 11-127
CLKSET	Clock Set Control Register	0004_H	nBE	PV	Page 11-128
CLKCLR	Clock clear Control Register	0008_H	nBE	PV	Page 11-129
SYSCLKCR	System Clock Control	$000C_H$	U, PV	PV	Page 11-130
CPUCLKCR	CPU Clock Control	0010_H	U, PV	PV	Page 11-131
PBCLKCR	Peripheral Bus Clock Control	0014_H	U, PV	PV	Page 11-132
USBCLKCR	USB Clock Control	0018_H	U, PV	PV	Page 11-132
EBUCLKCR	EBU Clock Control	$001C_H$	U, PV	PV	Page 11-133
CCUCLKCR	CCU Clock Control	0020_H	U, PV	PV	Page 11-134
WDTCLKCR	WDT Clock Control	0024_H	U, PV	PV	Page 11-134
EXTCLKCR	External clock Control Register	0028_H	U, PV	PV	Page 11-135
SLEEPCR	Sleep Control Register	0030_H	U, PV	PV	Page 11-136
DSLEEPCR	Deep Sleep Control Register	0034_H	U, PV	PV	Page 11-138
OSCHPSTAT	OSCHP Status Register	0100_H	U, PV	BE	Page 11-139
OSCHPCTRL	OSCHP Control Register	0104_H	U, PV	PV	Page 11-140

Table 11-9 Registers Overview (cont'd)

Short Name	Register Long Name	Offset Addr. [1]	Access Mode		Description
			Read	Write	
PLLSTAT	PLL Status Register	0110_H	U, PV	BE	Page 11-141
PLLCON0	PLL Configuration 0 Register	0114_H	U, PV	PV	Page 11-143
PLLCON1	PLL Configuration 1 Register	0118_H	U, PV	PV	Page 11-145
PLLCON2	PLL Configuration 2 Register	$011C_H$	U, PV	PV	Page 11-145
USBPLLSTAT	USB_PLL Status Register	0120_H	U, PV	PV	Page 11-146
USBPLLCON	USB_PLL Control Register	0124_H	U, PV	PV	Page 11-147
CLKMXSTAT	Clock Multiplexing Status Register	0138_H	U, PV	BE	Page 11-149

1) The absolute register address is calculated as follows:
Module Base Address + Sub-Module Offset Address + Offset Address (shown in this column)

11.10.1 GCU Registers

ID

Register containing unique ID of the module.

ID
SCU Module ID Register **(0000_H)** **Reset Value: 00A0 C0XX$_H$**

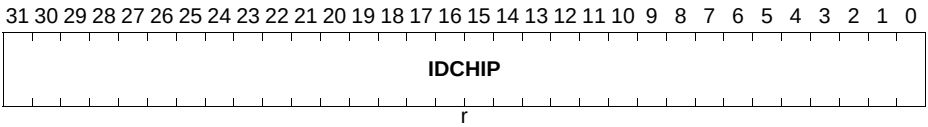
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
MOD_NUMBER	MOD_TYPE	MOD_REV
r	r	r

Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision** Indicates the revision number of the implementation. This information depends on the design step.
MOD_TYPE	[15:8]	r	**Module Type** This internal marker is fixed to $C0_H$.
MOD_NUMBER	[31:16]	r	**Module Number** Indicates the module identification number

IDCHIP

Register containing unique ID of the chip.

IDCHIP
Chip ID Register **(0004_H)** **Reset Value: XXXX XXXX$_H$**

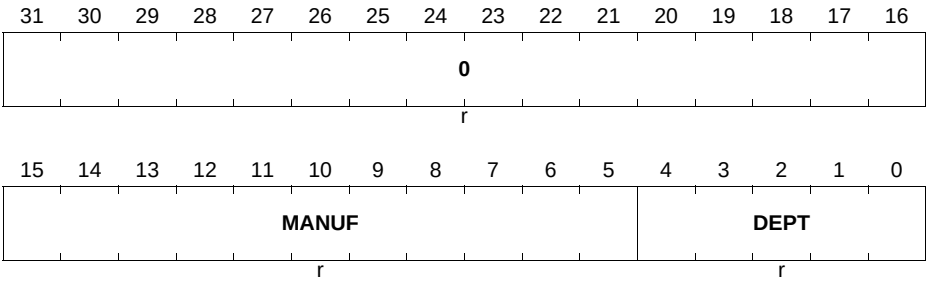


Field	Bits	Type	Description
IDCHIP	[31:0]	r	**Chip ID**

IDMANUF

Register containing unique manufactory ID of the chip.

IDMANUF
Manufactory ID Register **(0008_H)** **Reset Value: 0000 1820$_H$**

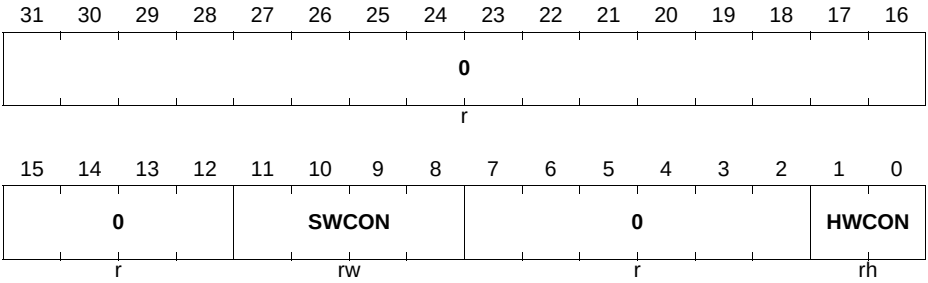


Field	Bits	Type	Description
DEPT	[4:0]	r	**Department Identification Number** DEPT indicated department within Infineon Technologies.
MANUF	[15:5]	r	**Manufacturer Identification Number** JEDEC normalized Manufacturer code. MANUF = $C1_H$ stands for Infineon Technologies.
0	[31:16]	r	**Reserved**

STCON

Startup configuration register determining boot process of the chip.

STCON
Startup Configuration Register (0010_H) Reset Value: $0000\ 0000_H$



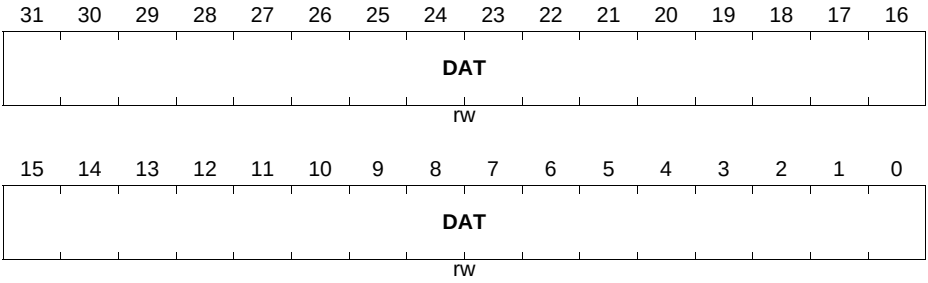
Field	Bits	Type	Description
HWCON	[1:0]	rh	**HW Configuration** At PORESET the following values are latched HWCON.0 = not (TMS) HWCON.1 = TCK 00_B Normal mode, JTAG 01_B ASC BSL enabled 10_B BMI customized boot enabled 11_B CAN BSL enabled

Field	Bits	Type	Description
SWCON	[11:8]	rw	**SW Configuration** Bit[9:8] is copy of Bit[1:0] after PORESET 0000_B Normal mode, boot from Boot ROM 0001_B ASC BSL enabled 0010_B BMI customized boot enabled 0011_B CAN BSL enabled 0100_B Boot from Code SRAM 1000_B Boot from alternate Flash Address 0 1100_B Boot from alternate Flash Address 1 1110_B Enable fallback Alternate Boot Mode (ABM) *Note: Only reset with Power-on Reset*
0	[7:2], [31:12]	r	**Reserved** Read as 0; should be written with 0.

GPRx

Software support registers. Can be reset only with PORST reset.

GPRx (x=0-1)
General Purpose Register x **($002C_H$+ x*4)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
DAT	[31:0]	rw	**User Data** 32-bit data *Note: GPRx registers can be reset with PORST reset only*

ETH0_CON

ETH0 module configuration register.

ETH0_CON
Ethernet 0 Port Control Register (50004040$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0					INFSEL	0		MDIO		0		CLK_TX		COL	
r					rw	r		rw		r		rw		rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RXER		CRS		CRS_DV		CLK_RMII		RXD3		RXD2		RXD1		RXD0	
rw		rw		rw		rw		rw		rw		rw		rw	

Field	Bits	Type	Description
RXD0	[1:0]	rw	**MAC Receive Input 0** This bit field indicates the receive input position of the RXD0 signal. 00_B Data input RXD0A is selected 01_B Data input RXD0B is selected 10_B Data input RXD0C is selected 11_B Data input RXD0D is selected
RXD1	[3:2]	rw	**MAC Receive Input 1** This bit field indicates the receive input position of the RXD1 signal. 00_B Data input RXD1A is selected 01_B Data input RXD1B is selected 10_B Data input RXD1C is selected 11_B Data input RXD1D is selected
RXD2	[5:4]	rw	**MAC Receive Input 2** This bit field indicates the receive input position of the RXD2 signal. 00_B Data input RXD2A is selected 01_B Data input RXD2B is selected 10_B Data input RXD2C is selected 11_B Data input RXD2D is selected

Field	Bits	Type	Description
RXD3	[7:6]	rw	**MAC Receive Input 3** This bit field indicates the receive input position of the RXD3 signal. 00_B Data input RXD3A is selected 01_B Data input RXD3B is selected 10_B Data input RXD3C is selected 11_B Data input RXD3D is selected
CLK_RMII	[9:8]	rw	**RMII clock input** This bit field indicates the receive input position of the RMII clock input signal. 00_B Data input RMIIA is selected 01_B Data input RMIIB is selected 10_B Data input RMIIC is selected 11_B Data input RMIID is selected
CRS_DV	[11:10]	rw	**CRS_DV input** This bit field indicates the receive input position of the CRS_DV input signal. 00_B Data input CRS_DVA is selected 01_B Data input CRS_DVB is selected 10_B Data input CRS_DVC is selected 11_B Data input CRS_DVD is selected
CRS	[13:12]	rw	**CRS input** This bit field indicates the receive input position of the CRS input signal. 00_B Data input CRSA 01_B Data input CRSB 10_B Data input CRSC 11_B Data input CRSD
RXER	[15:14]	rw	**RXER Input** This bit field indicates the receive input position of the RXER input signal. 00_B Data input RXERA is selected 01_B Data input RXERB is selected 10_B Data input RXERC is selected 11_B Data input RXERD is selected

Field	Bits	Type	Description
COL	[17:16]	rw	**COL input** This bit field indicates the receive input position of the COL clock input signal. 00_B Data input COLA is selected 01_B Data input COLB is selected 10_B Data input COLC is selected 11_B Data input COLD is selected
CLK_TX	[19:18]	rw	**CLK_TX input** This bit field indicates the receive input position of the CLK_TX input signal. 00_B Data input CLK_TXA is selected 01_B Data input CLK_TXB is selected 10_B Data input CLK_TXC is selected 11_B Data input CLK_TXD is selected
MDIO	[23:22]	rw	**MDIO Input Select** This bit field selects the input position of the MDI signal. 00_B Data input MDIA is selected 01_B Data input MDIB is selected 10_B Data input MDIC is selected 11_B Data input MDID is selected
INFSEL	26	rw	**Ethernet MAC Interface Selection** This bit selects Ethernet MAC interface to PHY. 0_B MII 1_B RMII
0	[21:20], [25:24], [31:27]	r	**Reserved** Read as 0; should be written with 0.

CCUCON

CAPCOM module control register. Individual signals signal is generated with f_{CCU} clock.

CCUCON
CCU Control Register **(004C$_H$)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0							0	0							
r							r	r							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						GSC81	GSC80	0				GSC43	GSC42	GSC41	GSC40
r						rw	rw	r				rw	rw	rw	rw

Field	Bits	Type	Description
GSC40	0	rw	**Global Start Control CCU40** 0_B Disable 1_B Enable
GSC41	1	rw	**Global Start Control CCU41** 0_B Disable 1_B Enable
GSC42	2	rw	**Global Start Control CCU42** 0_B Disable 1_B Enable
GSC43	3	rw	**Global Start Control CCU43** 0_B Disable 1_B Enable
GSC80	8	rw	**Global Start Control CCU80** 0_B Disable 1_B Enable
GSC81	9	rw	**Global Start Control CCU81** 0_B Disable 1_B Enable
0	[7:4], [23:10], 24, [31:25]	r	**Reserved** Read as 0; should be written with 0.

SRSTAT

Service request status reflecting masking with SRMSK mask register. Write one to a bit in SRCLR register to clear a bit or SRSET to set a bit. Writing zero has no effect. Outputs of this register are used to trigger interrupts or service requests.

SRSTAT
SCU Service Request Status **(0074_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		RMX	RTC_TIM1	RTC_TIM0	RTC_ATIM1	RTC_ATIM0	RTC_CTR	OSCULCTRL	OSCULSTAT	OSCSICTRL	0	HDCR	HDSET	HDCLR	HDSTAT
r		rh	rh	rh	rh	rh	rh	rh	rh	rh	r	rh	rh	rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					0					0		DLROVR	AI	PI	PRWARN
r					r					r		rh	rh	rh	rh

Field	Bits	Type	Description
PRWARN	0	rh	**WDT pre-warning Interrupt Status** 0_B Inactive 1_B Active
PI	1	rh	**RTC Periodic Interrupt Status** Set whenever periodic counter increments
AI	2	rh	**Alarm Interrupt Status** Set whenever count value matches compare value
DLROVR	3	rh	**DLR Request Overrun Interrupt Status** Set whenever DLR overrun condition occurs.
HDSTAT	16	rh	**HDSTAT Mirror Register Update Status** 0_B Not updated 1_B Update completed
HDCLR	17	rh	**HDCLR Mirror Register Update Status** 0_B Not updated 1_B Update completed
HDSET	18	rh	**HDSET Mirror Register Update Status** 0_B Not updated 1_B Update completed

Field	Bits	Type	Description
HDCR	19	rh	**HDCR Mirror Register Update Status** 0_B Not updated 1_B Update completed
OSCSICTRL	21	rh	**OSCSICTRL Mirror Register Update Status** 0_B Not updated 1_B Update completed
OSCULSTAT	22	rh	**OSCULSTAT Mirror Register Update Status** 0_B Not updated 1_B Update completed
OSCULCTRL	23	rh	**OSCULCTRL Mirror Register Update Status** 0_B Not updated 1_B Update completed
RTC_CTR	24	rh	**RTC CTR Mirror Register Update Status** 0_B Not updated 1_B Update completed
RTC_ATIM0	25	rh	**RTC ATIM0 Mirror Register Update Status** 0_B Not updated 1_B Update completed
RTC_ATIM1	26	rh	**RTC ATIM1 Mirror Register Update Status** 0_B Not updated 1_B Update completed
RTC_TIM0	27	rh	**RTC TIM0 Mirror Register Update Status** 0_B Not updated 1_B Update completed
RTC_TIM1	28	rh	**RTC TIM1 Mirror Register Update Status** 0_B Not updated 1_B Update completed
RMX	29	rh	**Retention Memory Mirror Register Update Status** 0_B Not updated 1_B Update completed
0	[5:4], [14:6], 15, 20, [31:30]	r	**Reserved**

SRRAW

Service request status without masking. Write one to a bit in **SRCLR** register to clear a bit or **SRSET** to set a bit. Writing zero has no effect.

SRRAW
SCU Raw Service Request Status (0078$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		RMX	RTC_TIM1	RTC_TIM0	RTC_ATIM1	RTC_ATIM0	RTC_CTR	OSCULCTRL	OSCULSTAT	OSCSICTRL	0	HDCR	HDSET	HDCLR	HDSTAT
r		rh	rh	rh	rh	rh	rh	rh	rh	rh	r	rh	rh	rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					0					0		DLROVR	AI	PI	PRWARN
r					r					r		rh	rh	rh	rh

Field	Bits	Type	Description
PRWARN	0	rh	**WDT pre-warning Interrupt Status Before Masking** 0_B Inactive 1_B Active
PI	1	rh	**RTC Raw Periodic Interrupt Status Before Masking** Set whenever periodic counter increments
AI	2	rh	**RTC Raw Alarm Interrupt Status Before Masking** Set whenever count value matches compare value
DLROVR	3	rh	**DLR Request Overrun Interrupt Status Before Masking** Set whenever DLR overrun condition occurs.
HDSTAT	16	rh	**HDSTAT Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
HDCLR	17	rh	**HDCLR Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed

Field	Bits	Type	Description
HDSET	18	rh	**HDSET Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
HDCR	19	rh	**HDCR Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
OSCSICTRL	21	rh	**OSCSICTRL Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
OSCULSTAT	22	rh	**OSCULSTAT Mirror Register Update Status** 0_B Not updated 1_B Update completed
OSCULCTRL	23	rh	**OSCULCTRL Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
RTC_CTR	24	rh	**RTC CTR Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
RTC_ATIM0	25	rh	**RTC ATIM0 Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
RTC_ATIM1	26	rh	**RTC ATIM1 Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
RTC_TIM0	27	rh	**RTC TIM0 Mirror Register Update Before Masking Status** 0_B Not updated 1_B Update completed

Field	Bits	Type	Description
RTC_TIM1	28	rh	**RTC TIM1 Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
RMX	29	rh	**Retention Memory Mirror Register Update Status Before Masking** 0_B Not updated 1_B Update completed
0	[5:4], [14:6], 15, 20, [31:30]	r	**Reserved**

SRMSK

Service request mask used to mask outputs of **SRRAW** register outputs connected to **SRSTAT** register.

SRMSK
SCU Service Request Mask **($007C_H$)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		**RMX**	**RTC_TIM1**	**RTC_TIM0**	**RTC_ATIM1**	**RTC_ATIM0**	**RTC_CTR**	**OSCULCTRL**	**OSCULSTAT**	**OSCSICTRL**	**0**	**HDCR**	**HDSET**	**HDCLR**	**HDSTAT**
r		rw	rw	rw	rw	rw	rw	rw	rw	rw	r	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					**0**					**0**		**DLROVR**	**AI**	**PI**	**PRWARN**
r					r					r		rw	rw	rw	rw

Field	Bits	Type	Description
PRWARN	0	rw	**WDT pre-warning Interrupt Mask** 0_B Disabled 1_B Enabled

Field	Bits	Type	Description
PI	1	rw	**RTC Periodic Interrupt Mask** 0_B Disabled 1_B Enabled
AI	2	rw	**RTC Alarm Interrupt Mask** 0_B Disabled 1_B Enabled
DLROVR	3	rw	**DLR Request Overrun Interrupt Mask** 0_B Disabled 1_B Enabled
HDSTAT	16	rw	**HDSTAT Mirror Register Update Mask** 0_B Disabled 1_B Enabled
HDCLR	17	rw	**HDCLR Mirror Register Update Mask** 0_B Disabled 1_B Enabled
HDSET	18	rw	**HDSET Mirror Register Update Mask** 0_B Disabled 1_B Enabled
HDCR	19	rw	**HDCR Mirror Register Update Mask** 0_B Disabled 1_B Enabled
OSCSICTRL	21	rw	**OSCSICTRL Mirror Register Update Mask** 0_B Disabled 1_B Enabled
OSCULSTAT	22	rw	**OSCULSTAT Mirror Register Update Mask** 0_B Disabled 1_B Enabled
OSCULCTRL	23	rw	**OSCULCTRL Mirror Register Update Mask** 0_B Disabled 1_B Enabled
RTC_CTR	24	rw	**RTC CTR Mirror Register Update Mask** 0_B Disabled 1_B Enabled
RTC_ATIM0	25	rw	**RTC ATIM0 Mirror Register Update Mask** 0_B Disabled 1_B Enabled

Field	Bits	Type	Description
RTC_ATIM1	26	rw	**RTC ATIM1 Mirror Register Update Mask** 0_B Disabled 1_B Enabled
RTC_TIM0	27	rw	**RTC TIM0 Mirror Register Update Mask** 0_B Disabled 1_B Enabled
RTC_TIM1	28	rw	**RTC TIM1 Mirror Register Update Mask** 0_B Disabled 1_B Enabled
RMX	29	rw	**Retention Memory Mirror Register Update Mask** 0_B Disabled 1_B Enabled
0	[5:4], [14:6], 15, 20, [31:30]	r	**Reserved**

SRCLR

Clear service request bits of registers **SRRAW** and **SRSTAT**. Write one to clear corresponding bits. Writing zeros has no effect.

SRCLR

SCU Service Request Clear **(0080_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		**RMX**	**RTC_TIM1**	**RTC_TIM0**	**RTC_ATIM1**	**RTC_ATIM0**	**RTC_CTR**	**OSCULCTRL**	**OSCULSTAT**	**OSCSICTRL**	**0**	**HDCR**	**HDSET**	**HDCLR**	**HDSTAT**
r		w	w	w	w	w	w	w	w	w	r	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					**0**					**0**		**DLROVR**	**AI**	**PI**	**PRWARN**
r					r					r		w	w	w	w

Field	Bits	Type	Description
PRWARN	0	w	**WDT pre-warning Interrupt Clear** 0_B No effect 1_B Clear the status bit
PI	1	w	**RTC Periodic Interrupt Clear** 0_B No effect 1_B Clear the status bit
AI	2	w	**RTC Alarm Interrupt Clear** 0_B No effect 1_B Clear the status bit
DLROVR	3	w	**DLR Request Overrun Interrupt clear** 0_B No effect 1_B Clear the status bit
HDSTAT	16	w	**HDCTAT Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
HDCLR	17	w	**HDCLR Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
HDSET	18	w	**HDSET Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
HDCR	19	w	**HDCR Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
OSCSICTRL	21	w	**OSCSICTRL Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
OSCULSTAT	22	w	**OSCULSTAT Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
OSCULCTRL	23	w	**OSCULCTRL Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
RTC_CTR	24	w	**RTC CTR Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit

Field	Bits	Type	Description
RTC_ATIM0	25	w	**RTC ATIM0 Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
RTC_ATIM1	26	w	**RTC ATIM1 Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
RTC_TIM0	27	w	**RTC TIM0 Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
RTC_TIM1	28	w	**RTC TIM1 Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
RMX	29	w	**Retention Memory Mirror Register Update Clear** 0_B No effect 1_B Clear the status bit
0	[5:4], [14:6], 15, 20, [31:30]	r	**Reserved**

SRSET

Set service request fits of registers **SRRAW** and **SRSTAT**. Write one to clear corresponding bits. Writing zeros has no effect.

SRSET
SCU Service Request Set **(0084_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		**RMX**	**RTC_TIM1**	**RTC_TIM0**	**RTC_ATIM1**	**RTC_ATIM0**	**RTC_CTR**	**OSCULCTRL**	**OSCULSTAT**	**OSCSICTRL**	**0**	**HDCR**	**HDCRSET**	**HDCRCLR**	**HDSTAT**
r		w	w	w	w	w	w	w	w	w	r	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					**0**					**0**		**DLROVR**	**AI**	**PI**	**PRWARN**
r					r					r		w	w	w	w

Field	Bits	Type	Description
PRWARN	0	w	**WDT pre-warning Interrupt Set** 0_B No effect 1_B set the status bit
PI	1	w	**RTC Periodic Interrupt Set** 0_B No effect 1_B set the status bit
AI	2	w	**RTC Alarm Interrupt Set** 0_B No effect 1_B set the status bit
DLROVR	3	w	**DLR Request Overrun Interrupt Set** 0_B No effect 1_B set the status bit
HDSTAT	16	w	**HDSTAT Mirror Register Update Set** 0_B No effect 1_B set the status bit
HDCRCLR	17	w	**HDCRCLR Mirror Register Update Set** 0_B No effect 1_B set the status bit
HDCRSET	18	w	**HDCRSET Mirror Register Update Set** 0_B No effect 1_B set the status bit
HDCR	19	w	**HDCR Mirror Register Update Set** 0_B No effect 1_B set the status bit
OSCSICTRL	21	w	**OSCSICTRL Mirror Register Update Set** 0_B No effect 1_B set the status bit
OSCULSTAT	22	w	**OSCULSTAT Mirror Register Update Set** 0_B No effect 1_B set the status bit
OSCULCTRL	23	w	**OSCULCTRL Mirror Register Update Set** 0_B No effect 1_B set the status bit
RTC_CTR	24	w	**RTC CTR Mirror Register Update Set** 0_B No effect 1_B set the status bit

Field	Bits	Type	Description
RTC_ATIM0	25	w	**RTC ATIM0 Mirror Register Update Set** 0_B No effect 1_B set the status bit
RTC_ATIM1	26	w	**RTC ATIM1 Mirror Register Update Set** 0_B No effect 1_B set the status bit
RTC_TIM0	27	w	**RTC TIM0 Mirror Register Update Set** 0_B No effect 1_B set the status bit
RTC_TIM1	28	w	**RTC TIM1 Mirror Register Update Set** 0_B No effect 1_B set the status bit
RMX	29	w	**Retention Memory Mirror Register Update Set** 0_B No effect 1_B set the status bit
0	[5:4], [14:6], 15, 20, [31:30]	r	**Reserved**

NMIREQEN

The **NMIREQEN** register serves purpose of promoting service requests to NMI requests. Is a bit is set then corresponding service request reflected in **SRSTAT** otherwise will be mirrored in the **TRAPSTAT** register instead.

NMIREQEN
SCU Service Request Mask **(0088_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0												ERU03	ERU02	ERU01	ERU00
r												rw	rw	rw	rw

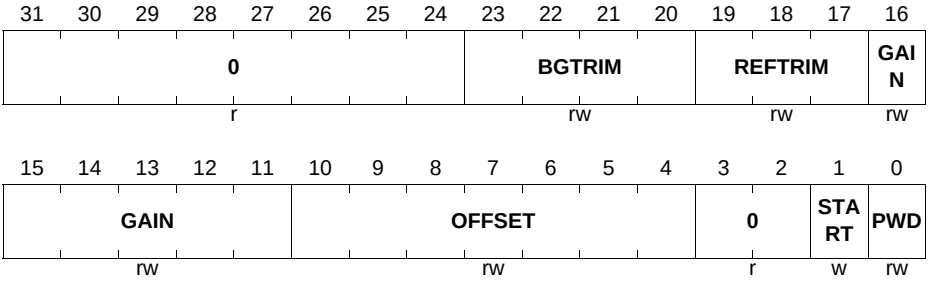
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0													AI	PI	PRWARN
r													rw	rw	rw

Field	Bits	Type	Description
PRWARN	0	rw	**Promote Pre-Warning Interrupt Request to NMI Request** 0_B Disabled 1_B Enabled
PI	1	rw	**Promote RTC Periodic Interrupt request to NMI Request** 0_B Disabled 1_B Enabled
AI	2	rw	**Promote RTC Alarm Interrupt Request to NMI Request** 0_B Disabled 1_B Enabled
ERU00	16	rw	**Promote Channel 0 Interrupt of ERU0 Request to NMI Request** 0_B Disabled 1_B Enabled
ERU01	17	rw	**Promote Channel 1 Interrupt of ERU0 Request to NMI Request** 0_B Disabled 1_B Enabled
ERU02	18	rw	**Promote Channel 2 Interrupt of ERU0 Request to NMI Request** 0_B Disabled 1_B Enabled
ERU03	19	rw	**Promote Channel 3 Interrupt of ERU0 Request to NMI Request** 0_B Disabled 1_B Enabled
0	[15:3], [31:20]	r	**Reserved**

DTSCON

Die temperature sensor control register

DTSCON
Die Temperature Sensor Control Register ($008C_H$) **Reset Value: $0000\ 0001_H$**



Field	Bits	Type	Description
PWD	0	rw	**Sensor Power Down** This bit defines the DTS power state. 0_B The DTS is powered 1_B The DTS is not powered
START	1	w	**Sensor Measurement Start** This bit starts a measurement of the DTS. 0_B No DTS measurement is started 1_B A DTS measurement is started If set this bit is automatically cleared. This bit always reads as zero.
OFFSET	[10:4]	rw	**Offset Calibration Value** This bit field interfaces the offset calibration values to the DTS. The calibration values are forwarded to the DTS by setting bit START.
GAIN	[16:11]	rw	**Gain Calibration Value** This bit field interfaces the gain calibration values to the DTS. The calibration values are forwarded to the DTS by setting bit START.
REFTRIM	[19:17]	rw	**Reference Trim Calibration Value** This bit field interfaces the reference trim calibration values to the DTS. The calibration values are forwarded to the DTS by setting bit START.

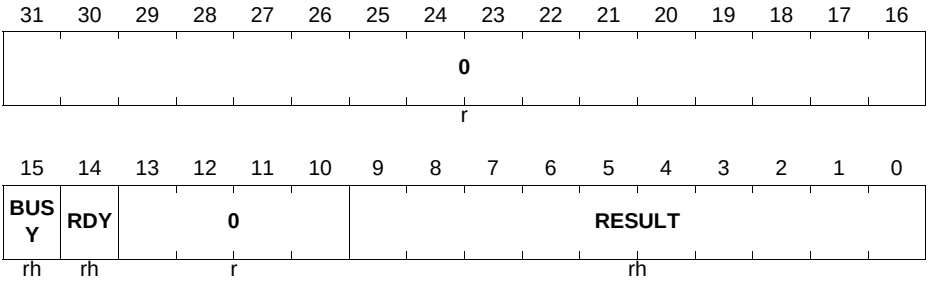
Field	Bits	Type	Description
BGTRIM	[23:20]	rw	**Bandgap Trim Calibration Value** This bit field interfaces the bandgap trim calibration values to the DTS. The calibration values are forwarded to the DTS by setting bit START.
0	[3:2], [31:24]	r	**Reserved** Read as 0; should be written with 0.

DTSSTAT

Die temperature status register

DTSSTAT

Die Temperature Sensor Status Register (0090_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
RESULT	[9:0]	rh	**Result of the DTS Measurement** This bit field shows the result of the DTS measurement. The value given is directly related to the die temperature.
RDY	14	rh	**Sensor Ready Status** This bit indicate the DTS is ready or not. 0_B The DTS is not ready 1_B The DTS is ready

Field	Bits	Type	Description
BUSY	15	rh	**Sensor Busy Status** This bit indicate if the DTS is currently busy. If the sensor is busy a measurement is still running and the result should not be used. 0_B not busy 1_B busy
0	[13:10], [31:16]	r	**Reserved** Read as 0; should be written with 0.

SDMMCDEL

Delay control register for SD-MMC module.

SDMMCDEL
SD-MMC Delay Control Register (009C$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								TAPDEL				0			TAPEN
r								rw				r			rw

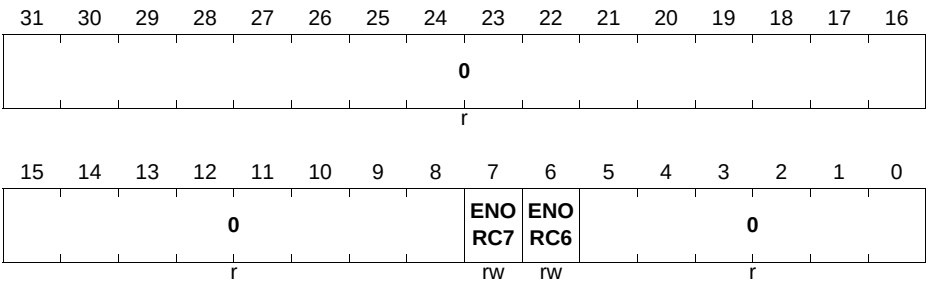
Field	Bits	Type	Description
TAPEN	0	rw	**Enable delay on the CMD/DAT out lines** 0_B Disabled 1_B Enabled
TAPDEL	[7:4]	rw	**Number of Delay Elements Select** number of delay elements of (TAPDEL+1),
0	[3:1], [31:8]	r	**Reserved** Read as 0; should be written with 0.

G0ORCEN

Enable register for out-of-range comparators of group 0 of analog input channels.

G0ORCEN

Out of Range Comparator Enable Register 0 ($00A0_H$) **Reset Value: $0000\ 0000_H$**



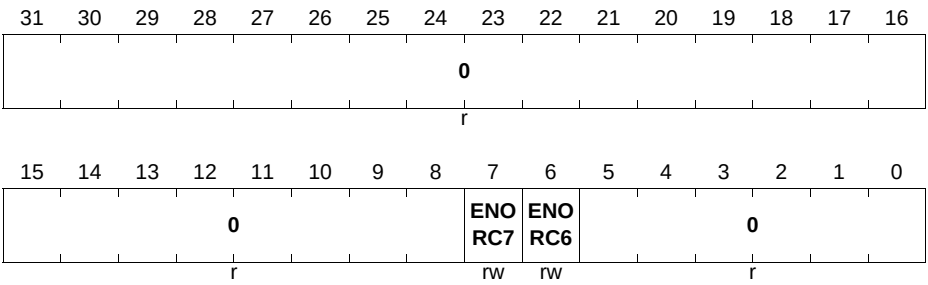
Field	Bits	Type	Description
ENORC6	6	rw	**Enable Out of Range Comparator, Channel 6** Each bit (when set) enables the out of range comparator of the associated channel 0_B Disabled 1_B Enabled
ENORC7	7	rw	**Enable Out of Range Comparator, Channel 7** Each bit (when set) enables the out of range comparator of the associated channel 0_B Disabled 1_B Enabled
0	[5:0], [31:8]	r	**Reserved** returns 0 if read; should be written with 0;

G1ORCEN

Enable register for out-of-range comparators of group 1 of analog input channels.

G1ORCEN

Out of Range Comparator Enable Register 1 ($00A4_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
ENORC6	6	rw	**Enable Out of Range Comparator, Channel 6** Each bit (when set) enables the out of range comparator of the associated channel 0_B Disabled 1_B Enabled
ENORC7	7	rw	**Enable Out of Range Comparator, Channel 7** Each bit (when set) enables the out of range comparator of the associated channel 0_B Disabled 1_B Enabled
0	[5:0], [31:8]	r	**Reserved** returns 0 if read; should be written with 0;

MIRRSTS

Mirror status register for control of communication between SCU and other modules in hibernate domain.

MIRRSTS
Mirror Update Status Register **($00C4_H$)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0							0								
r							r								

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		RMX	RTC_TIM1	RTC_TIM0	RTC_ATIM1	RTC_ATIM0	RTC_CTR	OSCULCTRL	OSCULSTAT	OSCSICTRL	0	HDCR	HDSET	HDCLR	HDSTAT
r		rh	rh	rh	rh	rh	rh	rh	rh	rh	r	rh	rh	rh	rh

Field	Bits	Type	Function
HDSTAT	0	rh	**HDSTAT Mirror Register Update Status** 0_B No update pending 1_B Update pending
HDCLR	1	rh	**HDCLR Mirror Register Update Status** 0_B No update pending 1_B Update pending
HDSET	2	rh	**HDSET Mirror Register Update Status** 0_B No update pending 1_B Update pending
HDCR	3	rh	**HDCR Mirror Register Update Status** 0_B No update pending 1_B Update pending
OSCSICTRL	5	rh	**OSCSICTRL Mirror Register Update Status** 0_B No update pending 1_B Update pending
OSCULSTAT	6	rh	**OSCULSTAT Mirror Register Update Status** 0_B No update pending 1_B Update pending
OSCULCTRL	7	rh	**OSCULCTRL Mirror Register Update Status** 0_B No update pending 1_B Update pending
RTC_CTR	8	rh	**RTC CTR Mirror Register Update Status** 0_B No update pending 1_B Update pending

Field	Bits	Type	Function
RTC_ATIM0	9	rh	**RTC ATIM0 Mirror Register Update Status** 0_B No update pending 1_B Update pending
RTC_ATIM1	10	rh	**RTC ATIM1 Mirror Register Update Status** 0_B No update pending 1_B Update pending
RTC_TIM0	11	rh	**RTC TIM0 Mirror Register Update Status** 0_B No update pending 1_B Update pending
RTC_TIM1	12	rh	**RTC TIM1 Mirror Register Update Status** 0_B No update pending 1_B Update pending
RMX	13	rh	**Retention Memory Access Register Update Status** This fields indicates status of retention memory update from **RMDATA** register to Hibernate domain retention memory or from Hibernate domain to **RMDATA** 0_B No update pending 1_B Update pending
0	4, [15:14], [24:16], [31:25]	r	**Reserved** Read as 0; should be written with 0.

RMACR

Access control to retention memory in hibernate domain.

RMACR

Retention Memory Access Control Register ($00C8_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0												ADDR			
r												rw			

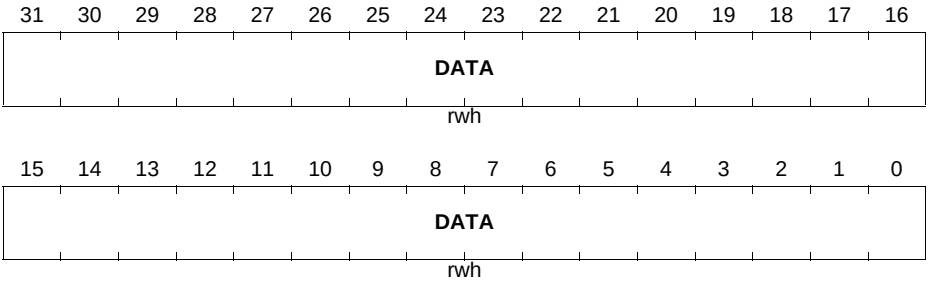
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0															RDWR
r															rw

Field	Bits	Type	Function
RDWR	0	rw	**Hibernate Retention Memory Register Update Control** This field controls access to Retention Memory using address selected in ADDR slice 0_B transfer data from Retention Memory in Hibernate domain to **RMDATA** register 1_B transfer data from **RMDATA** into Retention Memory in Hibernate domain
ADDR	[19:16]	rw	**Hibernate Retention Memory Register Address Select** This field selects Retention Memory address of 0 to 15 for read or write access.
0	[15:1], [31:20]	r	**Reserved** Read as 0; should be written with 0.

RMDATA

Access data of retention memory in hibernate domain.

RMDATA
Retention Memory Access Data Register ($00CC_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Function
DATA	[31:0]	rwh	**Hibernate Retention Memory Data** This field data of selected of Retention Memory using address. The address of 0-15 is selected with **RMACR** register.

PEEN

The following register enables parity check mechanism on peripheral modules.

PEEN
Parity Error Enable Register **($013C_H$)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0											PEENSD1	PEENSD0	PEENETH0RX	PEENETH0TX	PEENUSB
r											rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		PEENPPRF	PEENMC	0	PEENU2	PEENU1	PEENU0	0					PEENDS2	PEENDS1	PEENPS
r		rw	rw	r	rw	rw	rw	r					rw	rw	rw

Field	Bits	Type	Description
PEENPS	0	rw	**Parity Error Enable for PSRAM** 0_B Disabled 1_B Enabled
PEENDS1	1	rw	**Parity Error Enable for DSRAM1** 0_B Disabled 1_B Enabled
PEENDS2	2	rw	**Parity Error Enable for DSRAM2** 0_B Disabled 1_B Enabled
PEENU0	8	rw	**Parity Error Enable for USIC0 Memory** 0_B Disabled 1_B Enabled
PEENU1	9	rw	**Parity Error Enable for USIC1 Memory** 0_B Disabled 1_B Enabled
PEENU2	10	rw	**Parity Error Enable for USIC2 Memory** 0_B Disabled 1_B Enabled

Field	Bits	Type	Description
PEENMC	12	rw	**Parity Error Enable for MultiCAN Memory** 0_B Disabled 1_B Enabled
PEENPPRF	13	rw	**Parity Error Enable for PMU Prefetch Memory** 0_B Disabled 1_B Enabled
PEENUSB	16	rw	**Parity Error Enable for USB Memory** 0_B Disabled 1_B Enabled
PEENETH0TX	17	rw	**Parity Error Enable for ETH TX Memory** 0_B Disabled 1_B Enabled
PEENETH0RX	18	rw	**Parity Error Enable for ETH RX Memory** 0_B Disabled 1_B Enabled
PEENSD0	19	rw	**Parity Error Enable for SDMMC Memory 0** 0_B Disabled 1_B Enabled
PEENSD1	20	rw	**Parity Error Enable for SDMMC Memory 1** 0_B Disabled 1_B Enabled
0	[7:3], 11, [15:14], [31:21]	r	**Reserved** Should be written with 0.

MCHKCON

The following register enables the functional parity check mechanism for testing purpose. MCHKCON register is used to support access to parity bits of SRAM modules for various types of peripherals. The SRAM modules providing direct access natively need to be selected in order to enable direct write to parity bits using **PMTPR** register.

MCHKCON
Memory Checking Control Register (0140_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0											SELSD1	SELSD0	SELETH0RX	SELETH0TX	SELUSB
r											rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		PPRFDRA	MCANDRA	0	USIC2DRA	USIC1DRA	USIC0DRA	0					SELDS2	SELDS1	SELPS
r		rw	rw	r	rw	rw	rw	r					rw	rw	rw

Field	Bits	Type	Description
SELPS	0	rw	**Select Memory Check for PSRAM** 0_B Not selected 1_B Selected
SELDS1	1	rw	**Select Memory Check for DSRAM1** 0_B Not selected 1_B Selected
SELDS2	2	rw	**Select Memory Check for DSRAM2** 0_B Not selected 1_B Selected
USIC0DRA	8	rw	**Select Memory Check for USIC0** 0_B Not selected 1_B Selected
USIC1DRA	9	rw	**Select Memory Check for USIC1** 0_B Not selected 1_B Selected
USIC2DRA	10	rw	**Select Memory Check for USIC2** 0_B Not selected 1_B Selected
MCANDRA	12	rw	**Select Memory Check for MultiCAN** 0_B Not selected 1_B Selected

Field	Bits	Type	Description
PPRFDRA	13	rw	**Select Memory Check for PMU** 0_B Not selected 1_B Selected
SELUSB	16	rw	**Select Memory Check for USB SRAM** 0_B Not selected 1_B Selected
SELETH0TX	17	rw	**Select Memory Check for ETH0 TX SRAM** 0_B Not selected 1_B Selected
SELETH0RX	18	rw	**Select Memory Check for ETH0 RX SRAM** 0_B Not selected 1_B Selected
SELSD0	19	rw	**Select Memory Check for SDMMC SRAM 0** 0_B Not selected 1_B Selected
SELSD1	20	rw	**Select Memory Check for SDMMC SRAM 1** 0_B Not selected 1_B Selected
0	[7:3], 11, [15:14], [31:21]	r	**Reserved** Should be written with 0.

PETE

The following register enables the functional parity error trap generation mechanism. The trap flag gets reflected in **TRAPRAW** register and needs to be enabled with **TRAPDIS** register before can be effectively used to generate an NMI. The same tap flag can be configured with **PERSTEN** register to generate system reset instead of an NMI.

PETE

Parity Error Trap Enable Register (0144$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0											PETESD1	PETESD0	PETEETH0RX	PETEETH0TX	PETEUSB
r											rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		PETEPPRF	PETEMC	0	PETEU2	PETEU1	PETEU0	0					PETEDS2	PETEDS1	PETEPS
r		rw	rw	r	rw	rw	rw	r					rw	rw	rw

Field	Bits	Type	Description
PETEPS	0	rw	**Parity Error Trap Enable for PSRAM** 0_B Disabled 1_B Enabled
PETEDS1	1	rw	**Parity Error Trap Enable for DSRAM1** 0_B Disabled 1_B Enabled
PETEDS2	2	rw	**Parity Error Trap Enable for DSRAM2** 0_B Disabled 1_B Enabled
PETEU0	8	rw	**Parity Error Trap Enable for USIC0 Memory** 0_B Disabled 1_B Enabled
PETEU1	9	rw	**Parity Error Trap Enable for USIC1 Memory** 0_B Disabled 1_B Enabled
PETEU2	10	rw	**Parity Error Trap Enable for USIC2 Memory** 0_B Disabled 1_B Enabled
PETEMC	12	rw	**Parity Error Trap Enable for MultiCAN Memory** 0_B Disabled 1_B Enabled

Field	Bits	Type	Description
PETEPPRF	13	rw	**Parity Error Trap Enable for PMU Prefetch Memory** 0_B Disabled 1_B Enabled
PETEUSB	16	rw	**Parity Error Trap Enable for USB Memory** 0_B Disabled 1_B Enabled
PETEETH0TX	17	rw	**Parity Error Trap Enable for ETH 0TX Memory** 0_B Disabled 1_B Enabled
PETEETH0RX	18	rw	**Parity Error Trap Enable for ETH0 RX Memory** 0_B Disabled 1_B Enabled
PETESD0	19	rw	**Parity Error Trap Enable for SDMMC SRAM 0 Memory** 0_B Disabled 1_B Enabled
PETESD1	20	rw	**Parity Error Trap Enable for SDMMC SRAM 1 Memory** 0_B Disabled 1_B Enabled
0	[7:3], 11, [15:14], [31:21]	r	**Reserved** Should be written with 0.

PERSTEN

The following register enables reset upon parity error flag from the functional parity check mechanism indicated in **PEFLAG** register.

PERSTEN

Parity Error Reset Enable Register (0148$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0															RSEN
r															rw

Field	Bits	Type	Description
RSEN	0	rw	**System Reset Enable upon Parity Error Trap** 0_B Reset request disabled 1_B Reset request enabled
0	[31:1]	r	**Reserved** Should be written with 0.

PEFLAG

The PEFLAG register controls the functional parity check mechanism.

The register bits can only get set by corresponding parity error assertion if enabled and can only be cleared via software. Writing a zero to this bit does not change the content. Writing a one to this bit does clear the bit.

PEFLAG

Parity Error Flag Register (0150$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0											PESD1	PESD0	PEETH0RX	PEETH0TX	PEUSB
r											rwh	rwh	rwh	rwh	rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		PEFPPRF	PEFMC	0	PEFU2	PEFU1	PEFU0	0					PEFDS2	PEFDS1	PEFPS
r		rwh	rwh	r	rwh	rwh	rwh	r					rwh	rwh	rwh

Field	Bits	Type	Description
PEFPS	0	rwh	**Parity Error Flag for PSRAM** 0_B No parity error detected 1_B Parity error detected
PEFDS1	1	rwh	**Parity Error Flag for DSRAM1** 0_B No parity error detected 1_B Parity error detected
PEFDS2	2	rwh	**Parity Error Flag for DSRAM2** 0_B No parity error detected 1_B Parity error detected
PEFU0	8	rwh	**Parity Error Flag for USIC0 Memory** 0_B No parity error detected 1_B Parity error detected
PEFU1	9	rwh	**Parity Error Flag for USIC1 Memory** 0_B No parity error detected 1_B Parity error detected
PEFU2	10	rwh	**Parity Error Flag for USIC2 Memory** 0_B No parity error detected 1_B Parity error detected
PEFMC	12	rwh	**Parity Error Flag for MultiCAN Memory** 0_B No parity error detected 1_B Parity error detected
PEFPPRF	13	rwh	**Parity Error Flag for PMU Prefetch Memory** 0_B No parity error detected 1_B Parity error detected
PEUSB	16	rwh	**Parity Error Flag for USB Memory** 0_B No parity error detected 1_B Parity error detected
PEETH0TX	17	rwh	**Parity Error Flag for ETH TX Memory** 0_B No parity error detected 1_B Parity error detected
PEETH0RX	18	rwh	**Parity Error Flag for ETH RX Memory** 0_B No parity error detected 1_B Parity error detected
PESD0	19	rwh	**Parity Error Flag for SDMMC Memory 0** 0_B No parity error detected 1_B Parity error detected

Field	Bits	Type	Description
PESD1	20	rwh	**Parity Error Flag for SDMMC Memory 1** 0_B No parity error detected 1_B Parity error detected
0	[7:3], 11, [15:14], [31:21]	r	**Reserved** Should be written with 0.

PMTPR

The following register provides direct access to parity bits of a selected module.

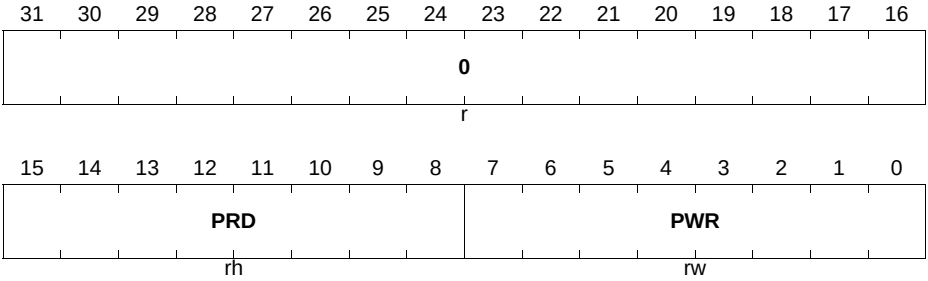
The width and therefore the valid bits in register **PMTPR** is listed in **Table 11-10**.

Table 11-10 Memory Widths

Memory	Number of Parity Bits	Valid Bits in PWR/PRD
Program SRAM (PSRAM)	4	PWR[3:0]/PRD[11:8]
System SRAM (DSRAM1)	4	PWR[3:0]/PRD[11:8]
Communication SRAM (DSRAM2)	4	PWR[3:0]/PRD[11:8]
USIC 0 Buffer Memory (U0)	1	PWR[0]/PRD[8]
USIC 1 Buffer Memory (U1)	1	PWR[0]/PRD[8]
USIC 2 Buffer Memory (U2)	1	PWR[0]/PRD[8]
MultiCAN Buffer Memory (MC)	1	PWR[0]/PRD[8]
PMU Prefetch Buffer Memory (PPRF)	1	PWR[0]/PRD[8]
USB Buffer Memory (USB)	1	PWR[0]/PRD[8]
ETH 0 TX Buffer Memory (ETH0TX)	1	PWR[0]/PRD[8]
ETH 0 RX Buffer Memory (ETH0RX)	1	PWR[0]/PRD[8]
SDMMC Buffer Memory 0 (SD0)	1	PWR[0]/PRD[8]
SDMMC Buffer Memory 1 (SD1)	1	PWR[0]/PRD[8]

PMTPR

Parity Memory Test Pattern Register (0154$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
PRD	[15:8]	rh	**Parity Read Values for Memory Test** For each byte of a memory module the parity bits generated during the most recent read access are indicated here.
PWR	[7:0]	rw	**Parity Write Values for Memory Test** For each byte of a memory module the parity bits corresponding to the next write access are stored here.
0	[31:16]	r	**Reserved** Should be written with 0.

PMTSR

This register selects parity test output from a memory instance that will be reflected in PRD bit field of **PMTPR** register.

*Note: Only one bit shall be set at the same time in register **PMTPR**. Otherwise the result of the parity software test is unpredictable.*

PMTSR
Parity Memory Test Select Register (0158$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0											MTSD1	MTSD0	MTETH0RX	MTETH0TX	MTUSB
r											rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		MTEPPRF	MTEMC	0	MTEU2	MTEU1	MTEU0	0					MTENDS2	MTENDS1	MTENPS
r		rwh	rwh	r	rwh	rwh	rwh	r					rw	rw	rw

Field	Bits	Type	Description
MTENPS	0	rw	**Test Enable Control for PSRAM** 0_B Standard operation 1_B Parity bits under test
MTENDS1	1	rw	**Test Enable Control for DSRAM1** 0_B Standard operation 1_B Parity bits under test
MTENDS2	2	rw	**Test Enable Control for DSRAM2** 0_B Standard operation 1_B Parity bits under test
MTEU0	8	rwh	**Test Enable Control for USIC0 Memory** 0_B Standard operation 1_B Parity bits under test
MTEU1	9	rwh	**Test Enable Control for USIC1 Memory** 0_B Standard operation 1_B Parity bits under test
MTEU2	10	rwh	**Test Enable Control for USIC2 Memory** 0_B Standard operation 1_B Parity bits under test
MTEMC	12	rwh	**Test Enable Control for MultiCAN Memory** 0_B Standard operation 1_B Parity bits under test

Field	Bits	Type	Description
MTEPPRF	13	rwh	**Test Enable Control for PMU Prefetch Memory** 0_B Standard operation 1_B Parity bits under test
MTUSB	16	rw	**Test Enable Control for USB Memory** 0_B Standard operation 1_B Parity bits under test
MTETH0TX	17	rw	**Test Enable Control for ETH TX Memory** 0_B Standard operation 1_B Parity bits under test
MTETH0RX	18	rw	**Test Enable Control for ETH RX Memory** 0_B Standard operation 1_B Parity bits under test
MTSD0	19	rw	**Test Enable Control for SDMMC Memory 0** 0_B Standard operation 1_B Parity bits under test
MTSD1	20	rw	**Test Enable Control for SDMMC Memory 1** 0_B Standard operation 1_B Parity bits under test
0	[7:3], 11, [15:14], [31:21]	r	**Reserved** Should be written with 0.

TRAPSTAT

This register contains the status flags for all trap request trigger sources of the SCU. A trap flag is set when a corresponding emergency event occurs. Trap mechanism supports testing and debug of these status bits by software using registers **TRAPSET** and **TRAPCLR**. This register reflects masking with **TRAPDIS** register.

TRAPSTAT

Trap Status Register **(0160_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		0	0	0			BWERR1T	BWERR0T	ULPWDGT	BRWNT	PET	UVCOLCKT	SVCOLCKT	0	SOSCWDGT
r		r	r	r			rh	rh	rh	rh	rh	rh	rh	r	rh

Field	Bits	Type	Description
SOSCWDGT	0	rh	**System OSC WDT Trap Status** 0_B No pending trap request 1_B Pending trap request
SVCOLCKT	2	rh	**System VCO Lock Trap Status** 0_B No pending trap request 1_B Pending trap request
UVCOLCKT	3	rh	**USB VCO Lock Trap Status** 0_B No pending trap request 1_B Pending trap request
PET	4	rh	**Parity Error Trap Status** 0_B No pending trap request 1_B Pending trap request
BRWNT	5	rh	**Brown Out Trap Status** 0_B No pending trap request 1_B Pending trap request
ULPWDGT	6	rh	**OSCULP WDG Trap Status** 0_B No pending trap request 1_B Pending trap request
BWERR0T	7	rh	**Peripheral Bridge 0 Trap Status** This trap flags error responses for buffered write operations on the Peripheral Bridge 0 0_B No pending trap request 1_B Pending trap request

Field	Bits	Type	Description
BWERR1T	8	rh	**Peripheral Bridge 1 Trap Status** This trap flags error responses for buffered write operations on the Peripheral Bridge 1 0_B No pending trap request 1_B Pending trap request
0	1, [11:9], 12,13, [31:14]	r	**Reserved** Read as 0; should be written with 0.

TRAPRAW

This register contains the status flags for all trap request trigger sources of the SCU before masking with **TRAPDIS**.
A trap flag is set when a corresponding emergency event occurs. For setting and clearing of these status bits by software see registers **TRAPSET** and **TRAPCLR**, respectively.

TRAPRAW
Trap Raw Status Register **(0164_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		0	0	0			BWERR1T	BWERR0T	ULPWDGT	BRWNT	PET	UVCOLCKT	SVCOLCKT	0	SOSCWDGT
r		r	r	r			rh	rh	rh	rh	rh	rh	rh	r	rh

Field	Bits	Type	Description
SOSCWDGT	0	rh	**System OSC WDT Trap Raw Status** 0_B No pending trap request 1_B Pending trap request
SVCOLCKT	2	rh	**System VCO Lock Trap Raw Status** 0_B No pending trap request 1_B Pending trap request

Field	Bits	Type	Description
UVCOLCKT	3	rh	**USB VCO Lock Trap Raw Status** 0_B No pending trap request 1_B Pending trap request
PET	4	rh	**Parity Error Trap Raw Status** 0_B No pending trap request 1_B Pending trap request
BRWNT	5	rh	**Brown Out Trap Raw Status** 0_B No pending trap request 1_B Pending trap request
ULPWDGT	6	rh	**OSCULP WDG Trap RAW Status** 0_B No pending trap request 1_B Pending trap request
BWERR0T	7	rh	**Peripheral Bridge 0 Trap Raw Status** 0_B No pending trap request 1_B Pending trap request
BWERR1T	8	rh	**Peripheral Bridge 1 Trap Raw Status** 0_B No pending trap request 1_B Pending trap request
0	1, [11:9], 12,13, [31:14]	r	**Reserved** Read as 0; should be written with 0.

TRAPDIS

Disable corresponding traps.

TRAPDIS
Trap Disable Register **(0168_H)** **Reset Value: 0000 01FF$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		0	0	0			BWERR1T	BWERR0T	ULPWDGT	BRWNT	PET	UVCOLCKT	SVCOLCKT	0	SOSCWDGT
r		r	r	r			rw	rw	rw	rw	rw	rw	rw	r	rw

Field	Bits	Type	Description
SOSCWDGT	0	rw	**System OSC WDT Trap Disable** 0_B Trap request enabled 1_B Trap request disabled
SVCOLCKT	2	rw	**System VCO Lock Trap Disable** 0_B Trap request enabled 1_B Trap request disabled
UVCOLCKT	3	rw	**USB VCO Lock Trap Disable** 0_B Trap request enabled 1_B Trap request disabled
PET	4	rw	**Parity Error Trap Disable** 0_B Trap request enabled 1_B Trap request disabled
BRWNT	5	rw	**Brown Out Trap Disable** 0_B Trap request enabled 1_B Trap request disabled
ULPWDGT	6	rw	**Wake-up Trap Disable** 0_B Trap request enabled 1_B Trap request disabled
BWERR0T	7	rw	**Peripheral Bridge 0 Trap Disable** 0_B Trap request enabled 1_B Trap request disabled
BWERR1T	8	rw	**Peripheral Bridge 1 Trap Disable** 0_B Trap request enabled 1_B Trap request disabled
0	1, [11:9], 12,13, [31:14]	r	**Reserved** Read as 0; should be written with 0.

TRAPCLR

This register contains the software clear control for the trap status flags in register **TRAPRAW** and **TRAPSTAT**.

TRAPCLR
Trap Clear Register **($016C_H$)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		0	0	0			BWERR1T	BWERR0T	ULPWDGT	BRWNT	PET	UVCOLCKT	SVCOLCKT	0	SOSCWDGT
r		r	r	r			w	w	w	w	w	w	w	r	w

Field	Bits	Type	Description
SOSCWDGT	0	w	**System OSC WDT Trap Clear** 0_B No effect 1_B Clear trap request
SVCOLCKT	2	w	**System VCO Lock Trap Clear** 0_B No effect 1_B Clear trap request
UVCOLCKT	3	w	**USB VCO Lock Trap Clear** 0_B No effect 1_B Clear trap request
PET	4	w	**Parity Error Trap Clear** 0_B No effect 1_B Clear trap request
BRWNT	5	w	**Brown Out Trap Clear** 0_B No effect 1_B Clear trap request
ULPWDGT	6	w	**OSCULP WDG Trap Clear** 0_B No effect 1_B Clear trap request
BWERR0T	7	w	**Peripheral Bridge 0 Trap Clear** 0_B No effect 1_B Clear trap request
BWERR1T	8	w	**Peripheral Bridge 1 Trap Clear** 0_B No effect 1_B Clear trap request

Field	Bits	Type	Description
0	1, [11:9], 12,13, [31:14]	r	**Reserved** Read as 0; should be written with 0.

TRAPSET

This register contains the software set control for the trap status flags in register TRAPRAW.

TRAPSET
Trap Set Register **(0170_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		**0**	**0**	**0**			**BWERR1T**	**BWERR0T**	**ULPWDT**	**BRWNT**	**PET**	**UVCOLCKT**	**SVCOLCKT**	**0**	**SOSCWDGT**
r		r	r	r			w	w	w	w	w	w	w	r	w

Field	Bits	Type	Description
SOSCWDGT	0	w	**System OSC WDT Trap Set** 0_B No effect 1_B Set trap request
SVCOLCKT	2	w	**System VCO Lock Trap Set** 0_B No effect 1_B Set trap request
UVCOLCKT	3	w	**USB VCO Lock Trap Set** 0_B No effect 1_B Set trap request
PET	4	w	**Parity Error Trap Set** 0_B No effect 1_B Set trap request

Field	Bits	Type	Description
BRWNT	5	w	**Brown Out Trap Set** 0_B No effect 1_B Set trap request
ULPWDT	6	w	**OSCULP WDG Trap Set** 0_B No effect 1_B Set trap request
BWERR0T	7	w	**Peripheral Bridge 0 Trap Set** 0_B No effect 1_B Set trap request
BWERR1T	8	w	**Peripheral Bridge 1 Trap Set** 0_B No effect 1_B Set trap request
0	1, [11:9], 12,13, [31:14]	r	**Reserved** Read as 0; should be written with 0.

11.10.2 PCU Registers

PWRSTAT

Power status register.

PWRSTAT
PCU Status Register **(0200_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0													USBPUWQ	USBOTGEN	USBPHYPDQ
r													r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0														0	HIBEN
r														r	r

Field	Bits	Type	Description
HIBEN	0	r	**Hibernate Domain Enable Status** 0_B Inactive 1_B Active
USBPHYPDQ	16	r	**USB PHY Transceiver State** 0_B Power-down 1_B Active
USBOTGEN	17	r	**USB On-The-Go Comparators State** 0_B Power-down 1_B Active
USBPUWQ	18	r	**USB Weak Pull-Up at PADN State** 0_B Pull-up active 1_B Pull-up not active
0	1, [15:2], [31:19]	r	**Reserved**

PWRSET

Power control register. Write one to set, writing zeros have no effect.

PWRSET
PCU Set Control Register **(0204_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
						0							**USBPUWQ**	**USBOTGEN**	**USBPHYPDQ**
						r							w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
							0								**HIB**
							r								w

Field	Bits	Type	Description
HIB	0	w	**Set Hibernate Domain Enable** 0_B No effect 1_B Enable Hibernate domain

Field	Bits	Type	Description
USBPHYPDQ	16	w	**Set USB PHY Transceiver Disable** 0_B No effect 1_B Active
USBOTGEN	17	w	**Set USB On-The-Go Comparators Enable** 0_B No effect 1_B Active
USBPUWQ	18	w	**Set USB Weak Pull-Up at PADN Enable** 0_B No effect 1_B Pull-up not active
0	[15:1], [31:19]	r	**Reserved**

PWRCLR

Power control register. Write one to clear, writing zeros have no effect.

PWRCLR
PCU Clear Control Register **(0208_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
						0							**USBPUWQ**	**USBOTGEN**	**USBPHYPDQ**
						r							w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
							0								**HIB**
							r								w

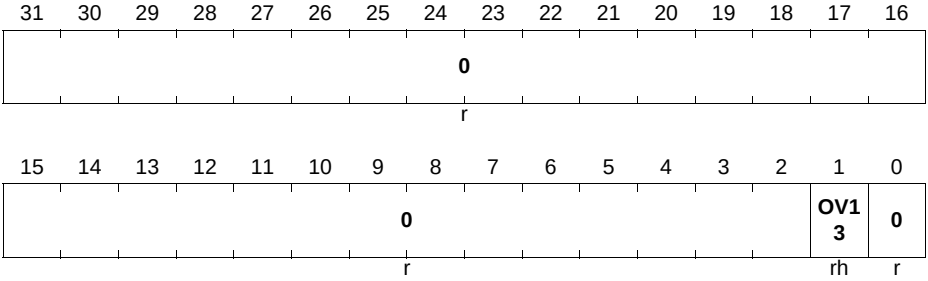
Field	Bits	Type	Description
HIB	0	w	**Clear Disable Hibernate Domain** 0_B No effect 1_B Disable Hibernate domain
USBPHYPDQ	16	w	**Clear USB PHY Transceiver Disable** 0_B No effect 1_B Power-down

Field	Bits	Type	Description
USBOTGEN	17	w	**Clear USB On-The-Go Comparators Enable** 0_B No effect 1_B Power-down
USBPUWQ	18	w	**Clear USB Weak Pull-Up at PADN Enable** 0_B No effect 1_B Pull-up active
0	[15:1], [31:19]	r	**Reserved**

EVRSTAT

EVR status register.

EVRSTAT
EVR Status Register **(0210_H)** **Reset Value: 0000 0000$_H$**

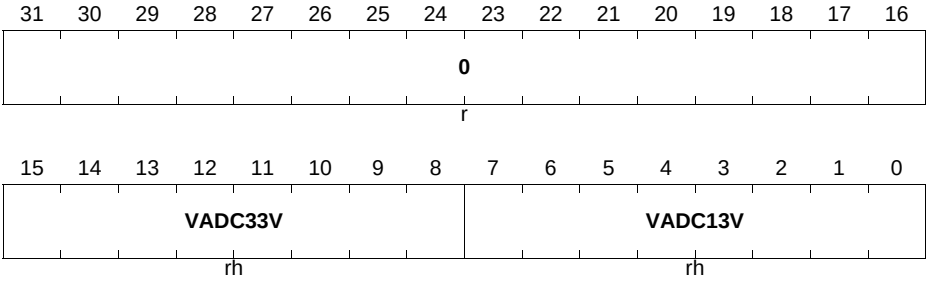


Field	Bits	Type	Description
OV13	1	rh	**Regulator Overvoltage for 1.3 V** 0_B No overvoltage condition 1_B Regulator is in overvoltage
0	0, [31:2]	r	**Reserved** Read as 0; should be written with 0.

EVRVADCSTAT

Supply voltage monitor register.

EVRVADCSTAT
EVR VADC Status Register **(0214_H)** **Reset Value: 0000 0000$_H$**

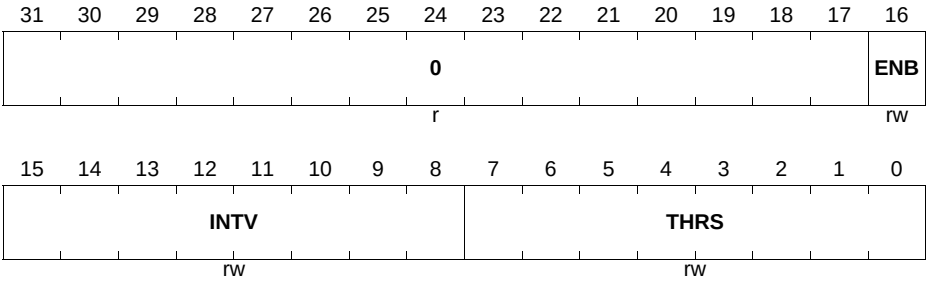


Field	Bits	Type	Description
VADC13V	[7:0]	rh	**VADC 1.3 V Conversion Result** This bit field contains the last conversion result of the VADC for the EVR13.
VADC33V	[15:8]	rh	**VADC 3.3 V Conversion Result** This bit field contains the last conversion result of the VADC for the EVR33. The value is used for brown-out detection
0	[31:16]	r	**Reserved** Read as 0.

PWRMON

Power monitoring control register for brown-out detection.

PWRMON
Power Monitor Control **(022C$_H$)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
THRS	[7:0]	rw	**Threshold** Threshold value for comparison to V_{DDP} for brown-out detection
INTV	[15:8]	rw	**Interval** Interval value for comparison to V_{DDP} expressed in cycles of system clock
ENB	16	rw	**Enable** Enable of comparison and interrupt generation
0	[31:17]	r	**Reserved**

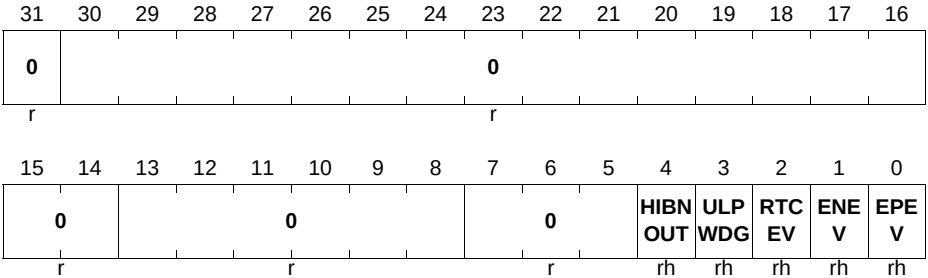
11.10.3 HCU Registers

HDSTAT

Hibernate domain status register

HDSTAT

Hibernate Domain Status Register (0300$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
EPEV	0	rh	**Wake-up Pin Event Positive Edge** 0_B Wake-up on positive edge pin event inactive 1_B Wake-up on positive edge pin event active
ENEV	1	rh	**Wake-up Pin Event Negative Edge** 0_B Wake-up on negative edge pin event inactive 1_B Wake-up on negative edge pin event active
RTCEV	2	rh	**RTC Event** 0_B Wake-up on RTC event inactive 1_B Wake-up on RTC event active
ULPWDG	3	rh	**ULP WDG Alarm Status** 0_B Watchdog alarm did not occur 1_B Watchdog alarm occurred
HIBNOUT	4	rh	**Hibernate Control Status** 0_B Hibernate not driven active to pads 1_B Hibernate driven active to pads
0	[7:5], [13:8], [30:14], 31	r	**Reserved** Read as 0; should be written with 0.

HDCLR

Hibernate domain clear status register. Write one to clear, writing zeros has no effect.

HDCLR

Hibernate Domain Status Clear Register (0304_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								0							
r								r							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				0				0				ULP WDG	RTC EV	ENE V	EPE V
r				r				r				w	w	w	w

Field	Bits	Type	Description
EPEV	0	w	**Wake-up Pin Event Positive Edge Clear** 0_B No effect 1_B Clear wake-up event
ENEV	1	w	**Wake-up Pin Event Negative Edge Clear** 0_B No effect 1_B Clear wake-up event
RTCEV	2	w	**RTC Event Clear** 0_B No effect 1_B Clear wake-up event
ULPWDG	3	w	**ULP WDG Alarm Clear** 0_B No effect 1_B Clear watchdog alarm
0	[7:4], [13:8], [30:14], 31	r	**Reserved** Read as 0; should be written with 0.

HDSET

Hibernate domain set status register. Write one to set, writing zeros has no effect.

HDSET

Hibernate Domain Status Set Register (0308_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								0							
r								r							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				0				0				ULPWDG	RTCEV	ENEV	EPEV
r				r				r				w	w	w	w

Field	Bits	Type	Description
EPEV	0	w	**Wake-up Pin Event Positive Edge Set** 0_B No effect 1_B Set wake-up event
ENEV	1	w	**Wake-up Pin Event Negative Edge Set** 0_B No effect 1_B Set wake-up event
RTCEV	2	w	**RTC Event Set** 0_B No effect 1_B Set wake-up event
ULPWDG	3	w	**ULP WDG Alarm Set** 0_B No effect 1_B Set watchdog alarm
0	[7:4], [13:8], [30:14], 31	r	**Reserved** Read as 0; should be written with 0.

HDCR

Hibernate domain configuration register.

HDCR
Hibernate Domain Control Register (030C$_H$) **Reset Value: 000C 2000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		0						HIBIO1SEL				HIBIO0SEL			
r		r						rw				rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		HIBIO1POL	HIBIO0POL	0	GPI0SEL	0	WKUPSEL	STDBYSEL	RCS	0	HIB	ULPWDGEN	RTCE	WKPEN	WKPEP
r		rw	rw	r	rw	r	rw	rw	rw	r	rwh	rw	rw	rw	rw

Field	Bits	Type	Description
WKPEP	0	rw	**Wake-Up on Pin Event Positive Edge Enable** 0_B Wake-up event disabled 1_B Wake-up event enabled
WKPEN	1	rw	**Wake-up on Pin Event Negative Edge Enable** 0_B Wake-up event disabled 1_B Wake-up event enabled
RTCE	2	rw	**Wake-up on RTC Event Enable** 0_B Wake-up event disabled 1_B Wake-up event enabled
ULPWDGEN	3	rw	**ULP WDG Alarm Enable** 0_B Wake-up event disabled 1_B Wake-up event enabled
HIB	4	rwh	**Hibernate Request Value Set** 0_B External hibernate request inactive 1_B External hibernate request active *Note: This bit get automatically cleared by hardware upon occurrence of any wake-up event enabled in this register*
RCS	6	rw	**f_{RTC} Clock Selection** 0_B f_{OSI} selected 1_B f_{ULP} selected
STDBYSEL	7	rw	**f_{STDBY} Clock Selection** 0_B f_{OSI} selected 1_B f_{ULP} selected

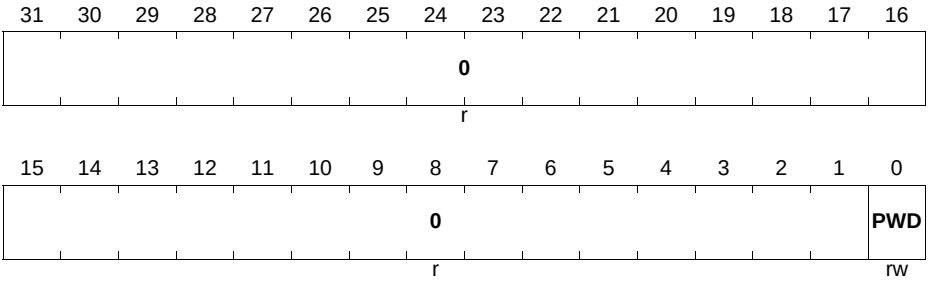
Field	Bits	Type	Description
WKUPSEL	8	rw	**Wake-Up from Hibernate Trigger Input Selection** 0_B HIB_IO_1 pin selected 1_B HIB_IO_0 pin selected
GPI0SEL	10	rw	**General Purpose Input 0 Selection** This bit field selects input to ERU0 module that optionally can be used with software as a general purpose input. 0_B HIB_IO_1 pin selected 1_B HIB_IO_0 pin selected
HIBIO0POL	12	rw	**HIBIO0 Polarity Set** Selects the output polarity of the HIBIO0 0_B Direct value 1_B Inverted value
HIBIO1POL	13	rw	**HIBIO1 Polarity Set** Selects the output polarity of the HIBIO1 0_B Direct value 1_B Inverted value
HIBIO0SEL	[19:16]	rw	**HIB_IO_0 Pin I/O Control (default HIBOUT)** This bit field determines the Port n line x functionality. 0000_B Direct input, No input pull device connected 0001_B Direct input, Input pull-down device connected 0010_B Direct input, Input pull-up device connected 1000_B Push-pull HIB Control output 1001_B Push-pull WDT service output 1010_B Push-pull GPIO output 1100_B Open-drain HIB Control output 1101_B Open-drain WDT service output 1110_B Open-drain GPIO output
HIBIO1SEL	[23:20]	rw	**HIB_IO_1 Pin I/O Control (Default WKUP)** This bit field determines the Port n line x functionality. 0000_B Direct input, No input pull device connected 0001_B Direct input, Input pull-down device connected 0010_B Direct input, Input pull-up device connected 1000_B Push-pull HIB Control output 1001_B Push-pull WDT service output 1010_B Push-pull GPIO output 1100_B Open-drain HIB Control output 1101_B Open-drain WDT service output 1110_B Open-drain GPIO output

Field	Bits	Type	Description
0	5,9, 11, [15:14], [29:24], [31:30]	r	**Reserved** Read as 0; should be written with 0.

OSCSICTRL

Control register for f_{OSI} clock source. A special mechanism keeps the f_{OSI} clock active if the external crystal oscillator is switched off, regardless of the value of the PWD bit field. The f_{OSI} can be switched off only if the external crystal oscillator is enabled and the f_{ULP} clock toggling.

OSCSICTRL
f_{OSI} Control Register **(0314_H)** **Reset Value: $0000\ 0001_H$**

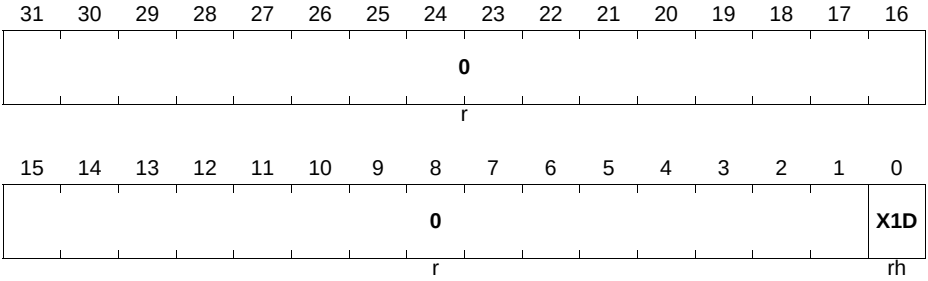


Field	Bits	Type	Description
PWD	0	rw	**Tun OFF the f_{OSI} Clock Source** 0_B Enabled 1_B Disabled
0	[31:1]	r	**Reserved** Read as 0; should be written with 0.

OSCULSTAT

Status register of the OSCULP oscillator.

OSCULSTAT
OSCULP Status Register **(0318_H)** **Reset Value: $0000\ 0000_H$**

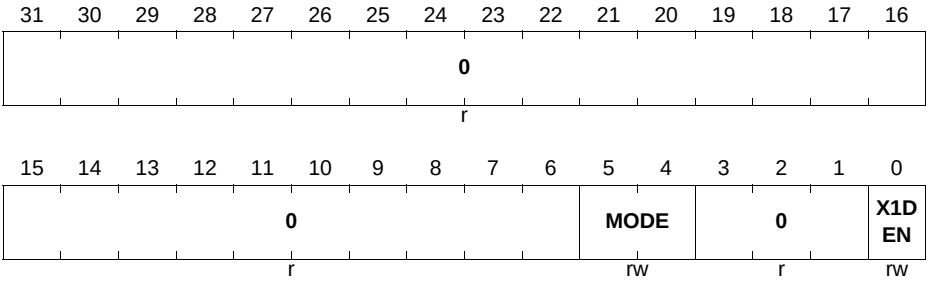


Field	Bits	Type	Description
X1D	0	rh	**XTAL1 Data Value** This bit monitors the value (level) of pin XTAL1. If XTAL1 is not used as clock input it can be used as GPI pin. This bit is only updated if X1DEN is set.
0	[31:1]	r	**Reserved** Read as 0; should be written with 0.

OSCULCTRL

Control register for OSCULP oscillator. This register allows selection of clock generation with external crystal, direct clock input, or power down mode. Alternate GPI function of the pin is also controlled with this register.

OSCULCTRL
OSCULP Control Register **($031C_H$)** **Reset Value: $0000\ 0020_H$**



Field	Bits	Type	Description
X1DEN	0	rw	**XTAL1 Data General Purpose Input Enable** The GPI data can be monitored with X1D bit of **OSCULSTAT** register 0_B Data input inactivated, power down 1_B Data input active *Note: It is strongly recommended to keep this function inactivated if the XTAL1 input is used as clock source*
MODE	[5:4]	rw	**Oscillator Mode** 00_B Oscillator is enabled, in operation 01_B Oscillator is enabled, in bypass mode 10_B Oscillator in power down 11_B Oscillator in power down, can be used as GPI *Note: Use of the oscillator input require that X1DEN bit is activated*
0	[3:1], [31:6]	r	**Reserved** Read as 0; should be written with 0.

11.10.4 RCU Registers

RSTSTAT

Reset status register. This register needs to be checked after system startup in order to determine last reset reason.

RSTSTAT
RCU Reset Status **(0400$_H$)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				0	LCK EN	HIBR S	HIB WK	RSTSTAT							
r				r	r	r	rh	rh							

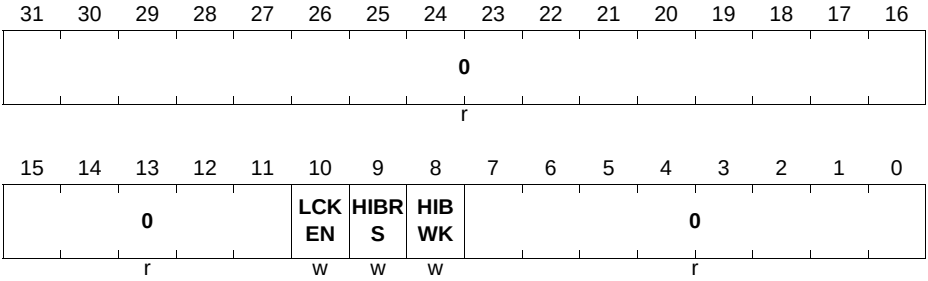
Field	Bits	Type	Description
RSTSTAT	[7:0]	rh	**Reset Status Information** Provides reason of last reset 00000001_B PORST reset 00000010_B SWD reset 00000100_B PV reset 00001000_B CPU system reset 00010000_B CPU lockup reset 00100000_B WDT reset 01000000_B Reserved 10000000_B Parity Error reset
HIBWK	8	rh	**Hibernate Wake-up Status** 0_B No Wake-up 1_B Wake-up event *Note: Field is cleared with enable of Hibernate mode*
HIBRS	9	r	**Hibernate Reset Status** 0_B Reset de-asserted 1_B Reset asserted
LCKEN	10	r	**Enable Lockup Status** 0_B Reset by Lockup disabled 1_B Reset by Lockup enabled
0	11, [31:12]	r	**Reserved**

RSTSET

Selective configuration of reset behavior in the system. Write one to set selected bit, writing zeros has no effect.

RSTSET

RCU Reset Set Register **(0404_H)** **Reset Value: 0000 0000$_H$**

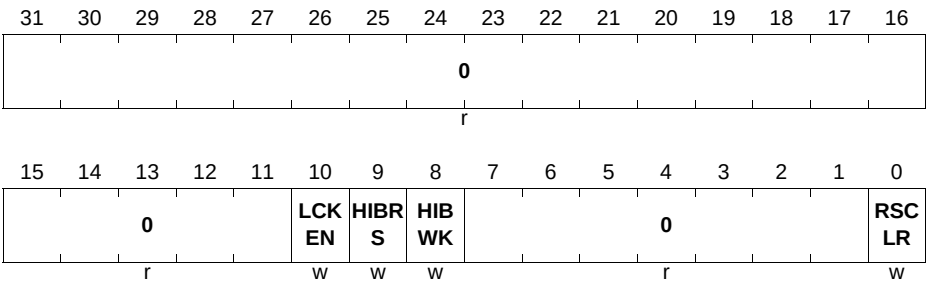


Field	Bits	Type	Description
HIBWK	8	w	**Set Hibernate Wake-up Reset Status** 0_B No effect 1_B Assert reset status bit
HIBRS	9	w	**Set Hibernate Reset** 0_B No effect 1_B Assert reset
LCKEN	10	w	**Enable Lockup Reset** 0_B No effect 1_B Enable reset when Lockup gets asserted
0	[7:0], [31:11]	r	**Reserved**

RSTCLR

Selective configuration of reset behavior in the system. Write one to clear selected bit, writing zeros has no effect.

RSTCLR
RCU Reset Clear Register **(0408_H)** **Reset Value: 0000 0000$_H$**

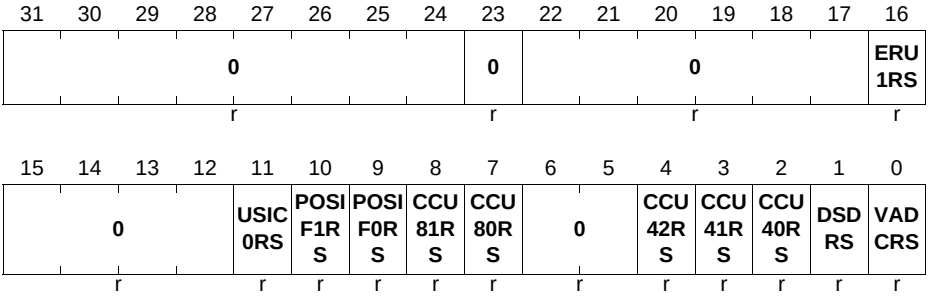


Field	Bits	Type	Description
RSCLR	0	w	**Clear Reset Status** 0_B No effect 1_B Clears field **RSTSTAT**.**RSTSTAT**
HIBWK	8	w	**Clear Hibernate Wake-up Reset Status** 0_B No effect 1_B De-assert reset status bit
HIBRS	9	w	**Clear Hibernate Reset** 0_B No effect 1_B De-assert reset
LCKEN	10	w	**Enable Lockup Reset** 0_B No effect 1_B Disable reset when Lockup gets asserted
0	[7:1], [31:11]	r	**Reserved**

PRSTAT0

Selective reset status register for peripherals for Peripherals 0.

PRSTAT0
RCU Peripheral 0 Reset Status **($040C_H$)** **Reset Value: 0001 0F9F$_H$**



Field	Bits	Type	Description
VADCRS	0	r	**VADC Reset Status** 0_B Reset de-asserted 1_B Reset asserted
DSDRS	1	r	**DSD Reset Status** 0_B Reset de-asserted 1_B Reset asserted
CCU40RS	2	r	**CCU40 Reset Status** 0_B Reset de-asserted 1_B Reset asserted
CCU41RS	3	r	**CCU41 Reset Status** 0_B Reset de-asserted 1_B Reset asserted
CCU42RS	4	r	**CCU42 Reset Status** 0_B Reset de-asserted 1_B Reset asserted
CCU80RS	7	r	**CCU80 Reset Status** 0_B Reset de-asserted 1_B Reset asserted
CCU81RS	8	r	**CCU81 Reset Status** 0_B Reset de-asserted 1_B Reset asserted
POSIF0RS	9	r	**POSIF0 Reset Status** 0_B Reset de-asserted 1_B Reset asserted

Field	Bits	Type	Description
POSIF1RS	10	r	**POSIF1 Reset Status** 0_B Reset de-asserted 1_B Reset asserted
USIC0RS	11	r	**USIC0 Reset Status** 0_B Reset de-asserted 1_B Reset asserted
ERU1RS	16	r	**ERU1 Reset Status** 0_B Reset de-asserted 1_B Reset asserted
0	[6:5], [15:12], [22:17], 23, [31:24]	r	**Reserved**

PRSET0

Selective reset assert register for peripherals for Peripherals 0. Write one to assert selected reset, writing zeros has no effect.

PRSET0
RCU Peripheral 0 Reset Set **(0410_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								0	0						ERU1RS
r								r	r						w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				USIC0RS	POSIF1RS	POSIF0RS	CCU81RS	CCU80RS	0		CCU42RS	CCU41RS	CCU40RS	DSDRS	VADCRS
r				w	w	w	w	w	r		w	w	w	w	w

Field	Bits	Type	Description
VADCRS	0	w	**VADC Reset Assert** 0_B No effect 1_B Assert reset
DSDRS	1	w	**DSD Reset Assert** 0_B No effect 1_B Assert reset
CCU40RS	2	w	**CCU40 Reset Assert** 0_B No effect 1_B Assert reset
CCU41RS	3	w	**CCU41 Reset Assert** 0_B No effect 1_B Assert reset
CCU42RS	4	w	**CCU42 Reset Assert** 0_B No effect 1_B Assert reset
CCU80RS	7	w	**CCU80 Reset Assert** 0_B No effect 1_B Assert reset
CCU81RS	8	w	**CCU81 Reset Assert** 0_B No effect 1_B Assert reset
POSIF0RS	9	w	**POSIF0 Reset Assert** 0_B No effect 1_B Assert reset
POSIF1RS	10	w	**POSIF1 Reset Assert** 0_B No effect 1_B Assert reset
USIC0RS	11	w	**USIC0 Reset Assert** 0_B No effect 1_B Assert reset
ERU1RS	16	w	**ERU1 Reset Assert** 0_B No effect 1_B Assert reset

Field	Bits	Type	Description
0	[6:5], [15:12], [22:17], 23, [31:24]	r	**Reserved**

PRCLR0

Selective reset de-assert register for peripherals for Peripherals 0. Write one to de-assert selected reset, writing zeros has no effect.

PRCLR0
RCU Peripheral 0 Reset Clear **(0414_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								0	0						ERU1RS
r								r	r						w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				USIC0RS	POSIF1RS	POSIF0RS	CCU81RS	CCU80RS	0		CCU42RS	CCU41RS	CCU40RS	DSDRS	VADCRS
r				w	w	w	w	w	r		w	w	w	w	w

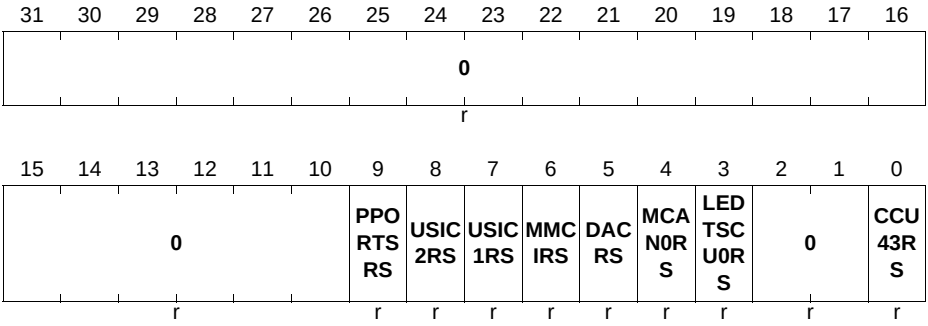
Field	Bits	Type	Description
VADCRS	0	w	**VADC Reset Clear** 0_B No effect 1_B De-assert reset
DSDRS	1	w	**DSD Reset Clear** 0_B No effect 1_B De-assert reset
CCU40RS	2	w	**CCU40 Reset Clear** 0_B No effect 1_B De-assert reset

Field	Bits	Type	Description
CCU41RS	3	w	**CCU41 Reset Clear** 0_B No effect 1_B De-assert reset
CCU42RS	4	w	**CCU42 Reset Clear** 0_B No effect 1_B De-assert reset
CCU80RS	7	w	**CCU80 Reset Clear** 0_B No effect 1_B De-assert reset
CCU81RS	8	w	**CCU81 Reset Clear** 0_B No effect 1_B De-assert reset
POSIF0RS	9	w	**POSIF0 Reset Clear** 0_B No effect 1_B De-assert reset
POSIF1RS	10	w	**POSIF1 Reset Clear** 0_B No effect 1_B De-assert reset
USIC0RS	11	w	**USIC0 Reset Clear** 0_B No effect 1_B De-assert reset
ERU1RS	16	w	**ERU1 Reset Clear** 0_B No effect 1_B De-assert reset
0	[6:5], [15:12], [22:17], 23, [31:24]	r	**Reserved**

PRSTAT1

Selective reset status register for peripherals for Peripherals 1.

PRSTAT1
RCU Peripheral 1 Reset Status **(0418_H)** **Reset Value: 0000 01F9$_H$**



Field	Bits	Type	Description
CCU43RS	0	r	**CCU43 Reset Status** 0_B Reset de-asserted 1_B Reset asserted
LEDTSCU0RS	3	r	**LEDTS Reset Status** 0_B Reset de-asserted 1_B Reset asserted
MCAN0RS	4	r	**MultiCAN Reset Status** 0_B Reset de-asserted 1_B Reset asserted
DACRS	5	r	**DAC Reset Status** 0_B Reset de-asserted 1_B Reset asserted
MMCIRS	6	r	**MMC Interface Reset Status** 0_B Reset de-asserted 1_B Reset asserted
USIC1RS	7	r	**USIC1 Reset Status** 0_B Reset de-asserted 1_B Reset asserted
USIC2RS	8	r	**USIC2 Reset Status** 0_B Reset de-asserted 1_B Reset asserted

Field	Bits	Type	Description
PPORTSRS	9	r	**PORTS Reset Status** 0_B Reset de-asserted 1_B Reset asserted
0	[2:1], [31:10]	r	**Reserved**

PRSET1

Selective reset assert register for peripherals for Peripherals 1. Write one to assert selected reset, writing zeros has no effect.

PRSET1
RCU Peripheral 1 Reset Set **($041C_H$)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						**PPORTSRS**	**USIC2RS**	**USIC1RS**	**MMCIRS**	**DACRS**	**MCAN0RS**	**LEDTSCU0RS**	**0**		**CCU43RS**
r						w	w	w	w	w	w	w	r		w

Field	Bits	Type	Description
CCU43RS	0	w	**CCU43 Reset Assert** 0_B No effect 1_B Assert reset
LEDTSCU0RS	3	w	**LEDTS Reset Assert** 0_B No effect 1_B Assert reset
MCAN0RS	4	w	**MultiCAN Reset Assert** 0_B No effect 1_B Assert reset

Field	Bits	Type	Description
DACRS	5	w	**DAC Reset Assert** 0_B No effect 1_B Assert reset
MMCIRS	6	w	**MMC Interface Reset Assert** 0_B No effect 1_B Assert reset
USIC1RS	7	w	**USIC1 Reset Assert** 0_B No effect 1_B Assert reset
USIC2RS	8	w	**USIC2 Reset Assert** 0_B No effect 1_B Assert reset
PPORTSRS	9	w	**PORTS Reset Assert** 0_B No effect 1_B Assert reset
0	[2:1], [31:10]	r	**Reserved**

PRCLR1

Selective reset de-assert register for peripherals for Peripherals 1. Write one to de-assert selected reset, writing zeros has no effect.

PRCLR1
RCU Peripheral 1 Reset Clear **(0420_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

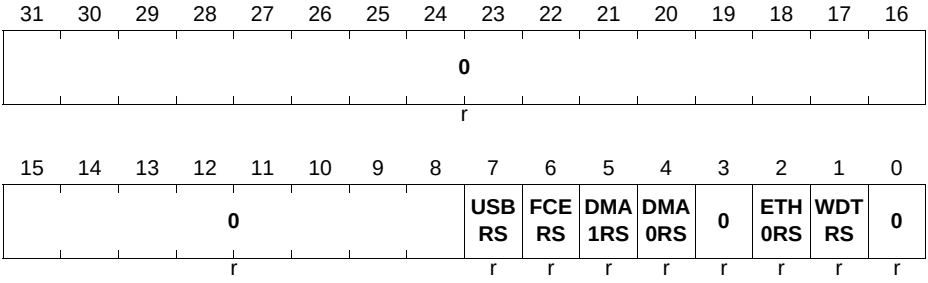
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						PPORTSRS	USIC2RS	USIC1RS	MMCIRS	DACRS	MCAN0RS	LEDTSCU0RS	0		CCU43RS
r						w	w	w	w	w	w	w	r		w

Field	Bits	Type	Description
CCU43RS	0	w	**CCU43 Reset Clear** 0_B No effect 1_B De-assert reset
LEDTSCU0RS	3	w	**LEDTS Reset Clear** 0_B No effect 1_B De-assert reset
MCAN0RS	4	w	**MultiCAN Reset Clear** 0_B No effect 1_B De-assert reset
DACRS	5	w	**DAC Reset Clear** 0_B No effect 1_B De-assert reset
MMCIRS	6	w	**MMC Interface Reset Clear** 0_B No effect 1_B De-assert reset
USIC1RS	7	w	**USIC1 Reset Clear** 0_B No effect 1_B De-assert reset
USIC2RS	8	w	**USIC2 Reset Clear** 0_B No effect 1_B De-assert reset
PPORTSRS	9	w	**PORTS Reset Clear** 0_B No effect 1_B De-assert reset
0	[2:1], [31:10]	r	**Reserved**

PRSTAT2

Selective reset status register for peripherals for Peripherals 2.

PRSTAT2
RCU Peripheral 2 Reset Status **(0424_H)** **Reset Value: 0000 00F6$_H$**



Field	Bits	Type	Description
WDTRS	1	r	**WDT Reset Status** 0_B Reset de-asserted 1_B Reset asserted
ETH0RS	2	r	**ETH0 Reset Status** 0_B Reset de-asserted 1_B Reset asserted
DMA0RS	4	r	**DMA0 Reset Status** 0_B Reset de-asserted 1_B Reset asserted
DMA1RS	5	r	**DMA1 Reset Status** 0_B Reset de-asserted 1_B Reset asserted
FCERS	6	r	**FCE Reset Status** 0_B Reset de-asserted 1_B Reset asserted
USBRS	7	r	**USB Reset Status** 0_B Reset de-asserted 1_B Reset asserted
0	0, 3, [31:8]	r	**Reserved**

PRSET2

Selective reset assert register for peripherals for Peripherals 2. Write one to assert selected reset, writing zeros has no effect.

PRSET2
RCU Peripheral 2 Reset Set **(0428_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								USBRS	FCERS	DMA1RS	DMA0RS	0	ETH0RS	WDTRS	0
r								w	w	w	w	r	w	w	r

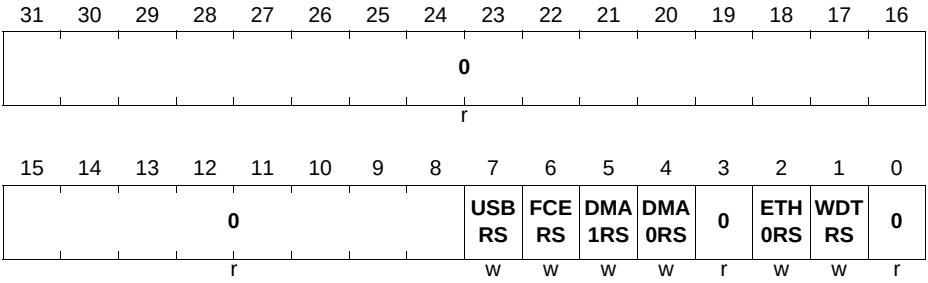
Field	Bits	Type	Description
WDTRS	1	w	**WDT Reset Assert** 0_B No effect 1_B Assert reset
ETH0RS	2	w	**ETH0 Reset Assert** 0_B No effect 1_B Assert reset
DMA0RS	4	w	**DMA0 Reset Assert** 0_B No effect 1_B Assert reset
DMA1RS	5	w	**DMA1 Reset Assert** 0_B No effect 1_B Assert reset
FCERS	6	w	**FCE Reset Assert** 0_B No effect 1_B Assert reset
USBRS	7	w	**USB Reset Assert** 0_B No effect 1_B Assert reset

Field	Bits	Type	Description
0	0, 3, [31:8]	r	**Reserved**

PRCLR2

Selective reset de-assert register for peripherals for Peripherals 2. Write one to de-assert selected reset, writing zeros has no effect.

PRCLR2
RCU Peripheral 2 Reset Clear **($042C_H$)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
WDTRS	1	w	**WDT Reset Clear** 0_B No effect 1_B De-assert reset
ETH0RS	2	w	**ETH0 Reset Clear** 0_B No effect 1_B De-assert reset
DMA0RS	4	w	**DMA0 Reset Clear** 0_B No effect 1_B De-assert reset
DMA1RS	5	w	**DMA1 Reset Clear** 0_B No effect 1_B De-assert reset
FCERS	6	w	**FCE Reset Clear** 0_B No effect 1_B De-assert reset

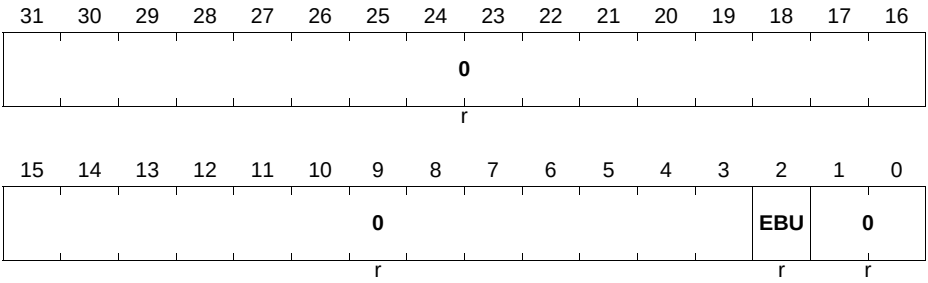
Field	Bits	Type	Description
USBRS	7	w	**USB Reset Clear** 0_B No effect 1_B De-assert reset
0	0, 3, [31:8]	r	**Reserved**

PRSTAT3

Selective reset status register for peripherals for Peripherals 3.

Note: Reset release must be effectively prevented for unless module clock is gated or off in cases where kernel clock and bus interface clocks are shared, in order to avoid system hang-ups.

PRSTAT3
RCU Peripheral 3 Reset Status **(0430_H)** **Reset Value: 0000 0004$_H$**



Field	Bits	Type	Description
EBU	2	r	**EBU Reset Status** 0_B Reset de-asserted 1_B Reset asserted
0	[1:0], [31:3]	r	**Reserved**

PRSET3

Selective reset assert register for peripherals for Peripherals 3. Write one to assert selected reset, writing zeros has no effect.

PRSET3
RCU Peripheral 3 Reset Set **(0434_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0													EBU	0	
r													w	r	

Field	Bits	Type	Description
EBU	2	w	**EBU Reset Assert** 0_B No effect 1_B Assert reset
0	[1:0], [31:3]	r	**Reserved**

PRCLR3

Selective reset de-assert register for peripherals for Peripherals 3. Write one to de-assert selected reset, writing zeros has no effect.

PRCLR3
RCU Peripheral 3 Reset Clear **(0438_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0													EBU	0	
r													w	r	

Field	Bits	Type	Description
EBU	2	w	**EBU Reset Assert** 0_B No effect 1_B De-assert reset
0	[1:0], [31:3]	r	**Reserved**

11.10.5 CCU Registers

CLKSTAT

Global clock status register.

CLKSTAT
Clock Status Register **(0600_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0										WDTCST	CCUCST	EBUCST	ETH0CST	MMCCST	USBCST
r										r	r	r	r	r	r

Field	Bits	Type	Description
USBCST	0	r	**USB Clock Status** 0_B Clock disabled 1_B Clock enabled
MMCCST	1	r	**MMC Clock Status** 0_B Clock disabled 1_B Clock enabled
ETH0CST	2	r	**Ethernet Clock Status** 0_B Clock disabled 1_B Clock enabled

Field	Bits	Type	Description
EBUCST	3	r	**EBU Clock Status** 0_B Clock disabled 1_B Clock enabled
CCUCST	4	r	**CCU Clock Status** 0_B Clock disabled 1_B Clock enabled
WDTCST	5	r	**WDT Clock Status** 0_B Clock disabled *Note: WDT clock can be put on hold in debug mode when this behavior is enabled at the watchdog* 1_B Clock enabled
0	[31:6]	r	**Reserved** Read as 0.

CLKSET

Global clock enable register. Write one to enable selected clock, writing zeros has no effect.

CLKSET
CLK Set Register **(0604_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0										WDTCEN	CCUCEN	EBUCEN	ETH0CEN	MMCCEN	USBCEN
r										w	w	w	w	w	w

Field	Bits	Type	Description
USBCEN	0	w	**USB Clock Enable** 0_B No effect 1_B Enable

Field	Bits	Type	Description
MMCCEN	1	w	**MMC Clock Enable** 0_B No effect 1_B Enable
ETH0CEN	2	w	**Ethernet Clock Enable** 0_B No effect 1_B Enable
EBUCEN	3	w	**EBU Clock Enable** 0_B No effect 1_B Enable
CCUCEN	4	w	**CCU Clock Enable** 0_B No effect 1_B Enable
WDTCEN	5	w	**WDT Clock Enable** 0_B No effect 1_B Enable
0	[31:6]	r	**Reserved** Read as 0.

CLKCLR

Global clock disable register. Write one to disable selected clock, writing zeros has no effect.

CLKCLR
CLK Clear Register **(0608_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

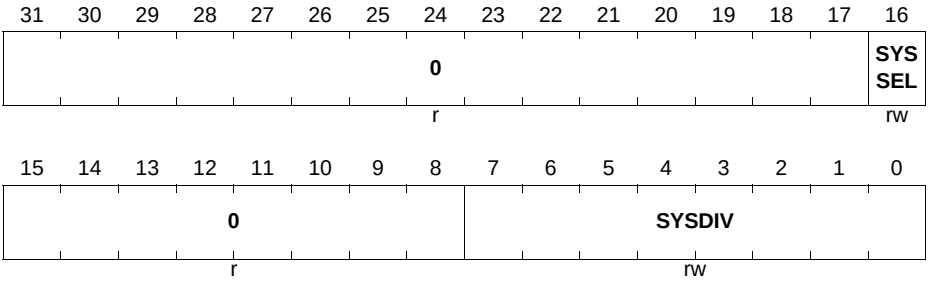
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0										WDT CDI	CCU CDI	EBU CDI	ETH 0CDI	MMC CDI	USB CDI
r										w	w	w	w	w	w

Field	Bits	Type	Description
USBCDI	0	w	**USB Clock Disable** 0_B No effect 1_B Disable clock
MMCCDI	1	w	**MMC Clock Disable** 0_B No effect 1_B Disable clock
ETH0CDI	2	w	**Ethernet Clock Disable** 0_B No effect 1_B Disable clock
EBUCDI	3	w	**EBU Clock Disable** 0_B No effect 1_B Disable clock
CCUCDI	4	w	**CCU Clock Disable** 0_B No effect 1_B Disable clock
WDTCDI	5	w	**WDT Clock Disable** 0_B No effect 1_B Disable clock
0	[31:6]	r	**Reserved** Read as 0.

SYSCLKCR

System clock control register.

SYSCLKCR
System Clock Control Register (060C$_H$) Reset Value: 0000 0000$_H$

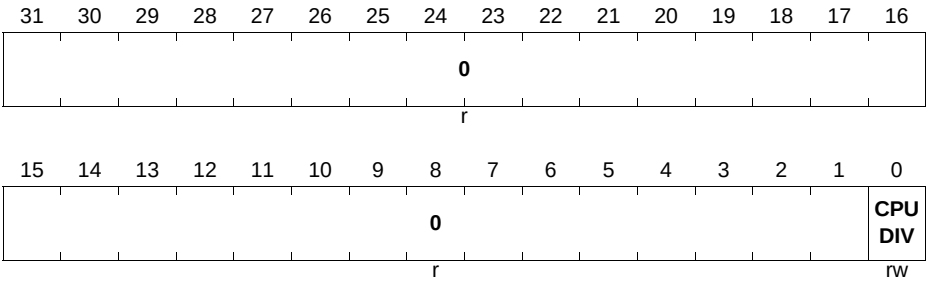


Field	Bits	Type	Function
SYSDIV	[7:0]	rw	**System Clock Division Value** The value the divider operates is (SYSDIV+1).
SYSSEL	16	rw	**System Clock Selection Value** 0_B f_{OFI} clock 1_B f_{PLL} clock
0	[15:8], [31:17]	r	**Reserved** Read as 0; should be written with 0.

CPUCLKCR

CPU clock control register.

CPUCLKCR
CPU Clock Control Register **(0610_H)** **Reset Value: $0000\ 0000_H$**



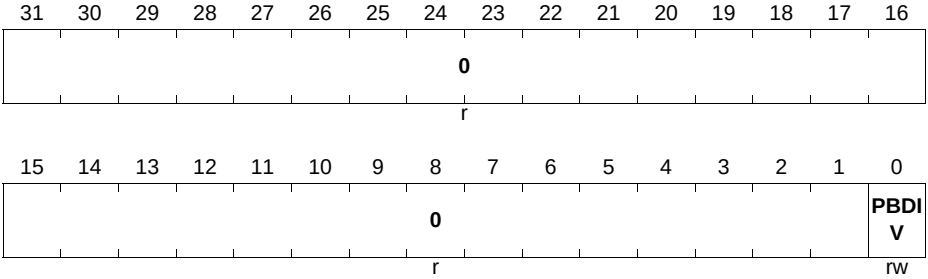
Field	Bits	Type	Function
CPUDIV	0	rw	**CPU Clock Divider Enable** This bit enables division of f_{SYS} clock to produce f_{CPU} clock. 0_B $f_{CPU} = f_{SYS}$ 1_B $f_{CPU} = f_{SYS} / 2$ *Note: Some clock division settings are not allowed. See **Table 11-5** for more details.*
0	[31:1]	r	**Reserved** Read as 0; should be written with 0.

PBCLKCR

Peripheral clock control register.

PBCLKCR

Peripheral Bus Clock Control Register (0614_H) **Reset Value: $0000\ 0000_H$**



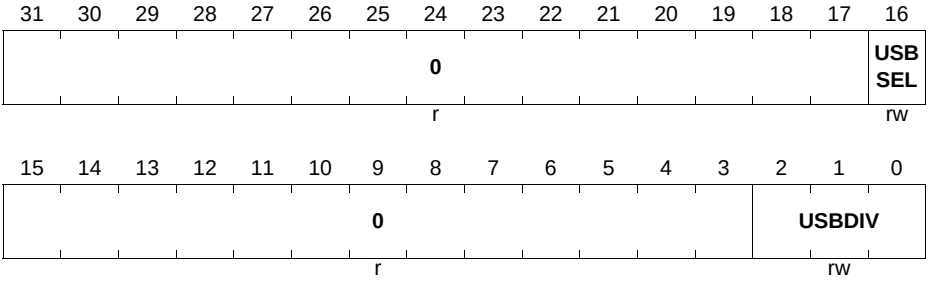
Field	Bits	Type	Function
PBDIV	0	rw	**PB Clock Divider Enable** This bit enables division of f_{SYS} clock to produce f_{PERIPH} clock. 0_B $f_{PERIPH} = f_{CPU}$ 1_B $f_{PERIPH} = f_{CPU} / 2$ *Note: Some clock division settings are not allowed. See **Table 11-5** for more details.*
0	[31:1]	r	**Reserved** Read as 0; should be written with 0.

USBCLKCR

USB clock control register.

USBCLKCR

USB Clock Control Register **(0618_H)** **Reset Value: 0000 0000$_H$**



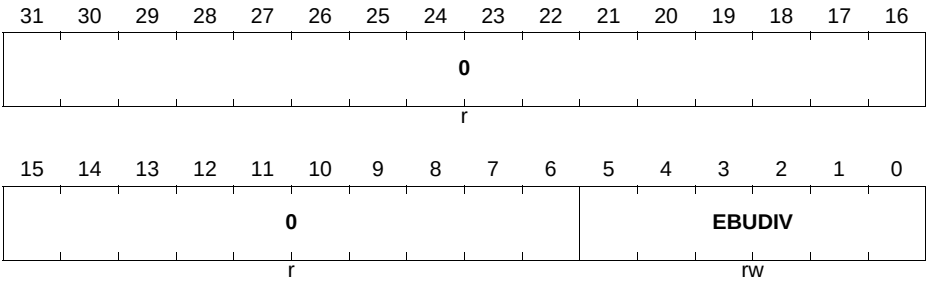
Field	Bits	Type	Function
USBDIV	[2:0]	rw	**USB Clock Divider Value** PLL clock is divided by USBDIV + 1 Must only be programmed, when clock is not used
USBSEL	16	rw	**USB Clock Selection Value** 0_B USB PLL Clock 1_B PLL Clock
0	[15:3], [31:17]	r	**Reserved** Read as 0; should be written with 0.

EBUCLKCR

EBU clock control register.

EBUCLKCR

EBU Clock Control Register **($061C_H$)** **Reset Value: 0000 0000$_H$**

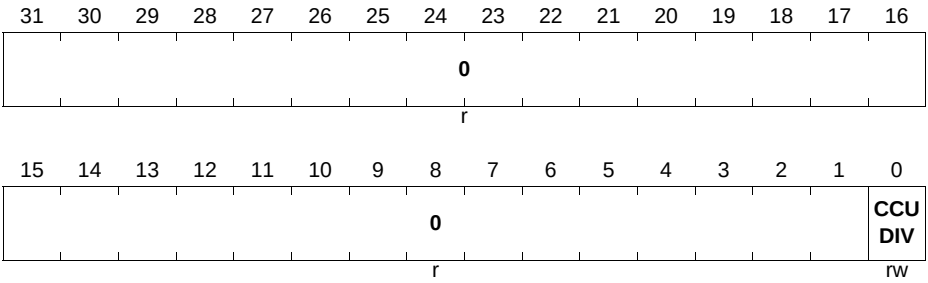


Field	Bits	Type	Function
EBUDIV	[5:0]	rw	**EBU Clock Divider Value** PLL clock is divided by EBUDIV + 1 Must only be programmed, when clock is not used
0	[31:6]	r	**Reserved** Read as 0; should be written with 0.

CCUCLKCR

CCUx clock control register.

CCUCLKCR
CCU Clock Control Register **(0620_H)** **Reset Value: $0000\ 0000_H$**

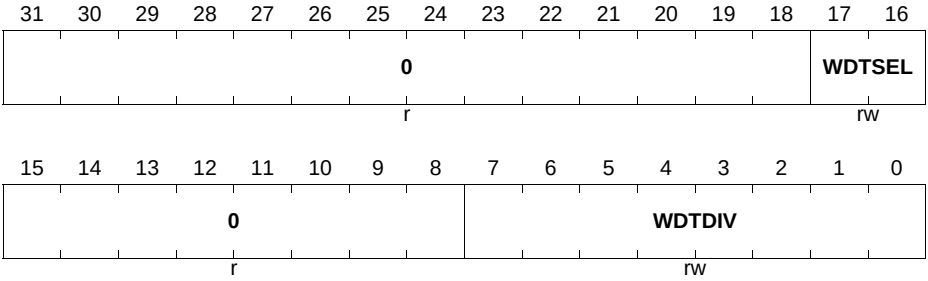


Field	Bits	Type	Function
CCUDIV	0	rw	**CCU Clock Divider Enable** This bit enables division of f_{SYS} clock to produce f_{CCU} clock. 0_B $f_{CCU} = f_{SYS}$ 1_B $f_{CCU} = f_{SYS} / 2$ *Note: Some clock division settings are not allowed. See **Table 11-5** for more details.*
0	[31:1]	r	**Reserved** Read as 0; should be written with 0.

WDTCLKCR

System watchdog (WDT) clock control register.

WDTCLKCR
WDT Clock Control Register (0624$_H$) Reset Value: 0000 0000$_H$

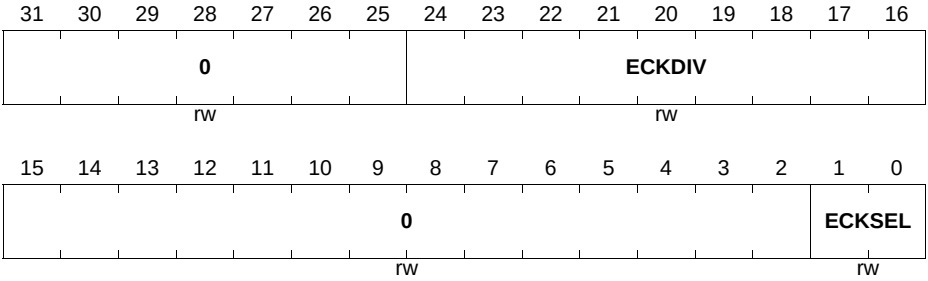


Field	Bits	Type	Function
WDTDIV	[7:0]	rw	**WDT Clock Divider Value** WDT is divided by WDTDIV + 1 Must only be programmed, when clock is not used
WDTSEL	[17:16]	rw	**WDT Clock Selection Value** 00_B f_{OFI} clock 01_B f_{STDBY} clock 10_B f_{PLL} clock 11_B Reserved
0	[15:8], [31:18]	r	**Reserved** Read as 0; should be written with 0.

EXTCLKCR

External clock control register. Use this register to select output clock.

EXTCLKCR
External Clock Control **(0628_H)** **Reset Value: $0000\ 0000_H$**

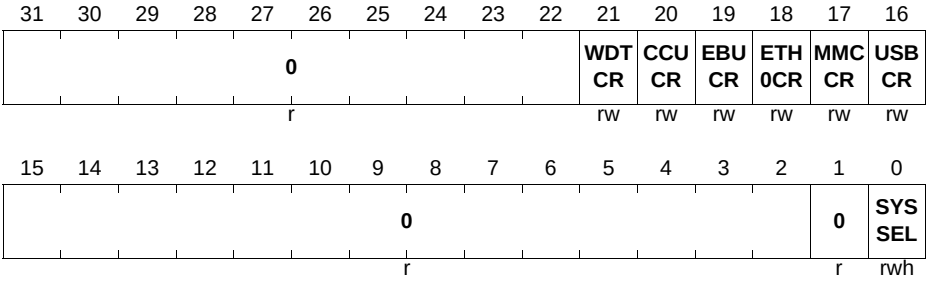


Field	Bits	Type	Description
ECKSEL	[1:0]	rw	**External Clock Selection Value** 00_B f_{SYS} clock 01_B f_{FLASH} clock 10_B f_{USB} clock 11_B f_{PLL} clock divided according to ECKDIV bit field configuration
ECKDIV	[24:16]	rw	**External Clock Divider Value** PLL clock is divided by ECKDIV + 1 Must only be programmed, when clock is not used
0	[15:2], [31:25]	rw	**Reserved** Read as 0.

SLEEPCR

Configuration register that defines some system behavior aspects while in sleep mode. The original system state gets restored upon wake-up from sleep mode.

SLEEPCR
Sleep Control Register **(0630_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
SYSSEL	0	rwh	**System Clock Selection Value** 0_B f_{OFI} clock 1_B f_{PLL} clock
USBCR	16	rw	**USB Clock Control** 0_B Disable 1_B Enable
MMCCR	17	rw	**MMC Clock Control** 0_B Disable 1_B Enable
ETH0CR	18	rw	**Ethernet Clock Control** 0_B Disable 1_B Enable
EBUCR	19	rw	**EBU Clock Control** 0_B Disable 1_B Enable
CCUCR	20	rw	**CCU Clock Control** 0_B Disable 1_B Enable
WDTCR	21	rw	**WDT Clock Control** 0_B Disable 1_B Enable
0	1, [15:2], [31:22]	r	**Reserved** Read as 0.

DSLEEPCR

Configuration register that defines some system behavior aspects while in deep-sleep mode. The original system state gets restored upon wake-up from sleep mode except for PLL re-start if enabled before entering deep-sleep mode and configured to go into power down while in deep-sleep mode.

DSLEEPCR
Deep Sleep Control Register **(0634_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0										WDTCR	CCUCR	EBUCR	ETH0CR	MMCCR	USBCR
r										rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		VCOPDN	PLLPDN	FPDN	0									SYSSEL	
r		rw	rw	rw	r									rwh	

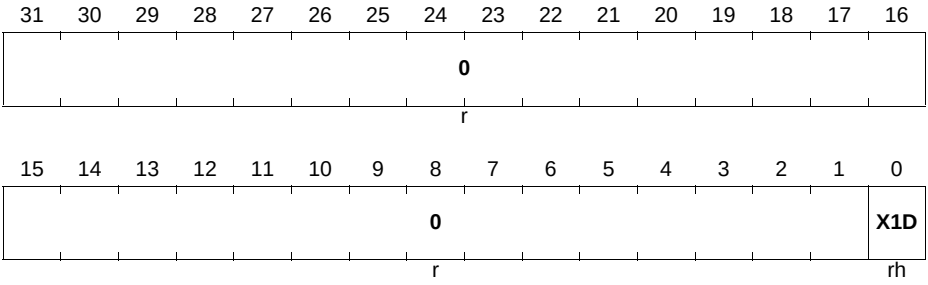
Field	Bits	Type	Description
SYSSEL	[1:0]	rwh	**System Clock Selection Value** 0_B f_{OFI} clock 1_B f_{PLL} clock
FPDN	11	rw	**Flash Power Down** 1_B Flash power down module 0_B No effect
PLLPDN	12	rw	**PLL Power Down** 1_B switch off main PLL 0_B No effect
VCOPDN	13	rw	**VCO Power Down** 1_B switch off VCO of main PLL 0_B No effect
USBCR	16	rw	**USB Clock Control** 0_B Disable 1_B Enable
MMCCR	17	rw	**MMC Clock Control** 0_B Disable 1_B Enable

Field	Bits	Type	Description
ETH0CR	18	rw	**Ethernet Clock Control** 0_B Disable 1_B Enable
EBUCR	19	rw	**EBU Clock Control** 0_B Disable 1_B Enable
CCUCR	20	rw	**CCU Clock Control** 0_B Disable 1_B Enable
WDTCR	21	rw	**WDT Clock Control** 0_B Disable 1_B Enable
0	[10:2], [15:14], [31:22]	r	**Reserved** Read as 0.

OSCHPSTAT

Status register of the OSCHP oscillator.

OSCHPSTAT
OSCHP Status Register **(0700_H)** **Reset Value: $0000\ 0000_H$**



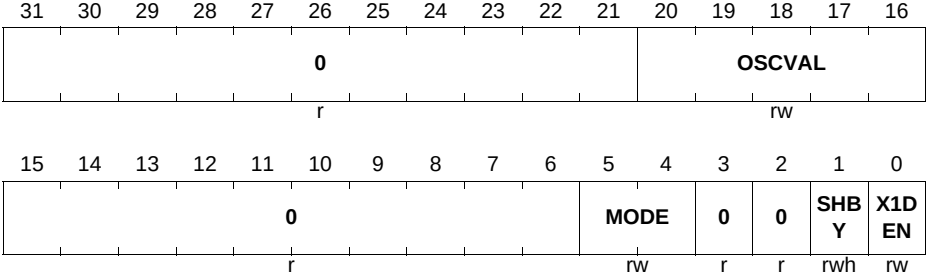
Field	Bits	Type	Description
X1D	0	rh	**XTAL1 Data Value** This bit monitors the value (level) of pin XTAL1. If XTAL1 is not used as clock input it can be used as GPI pin. This bit is only updated if X1DEN is set.

Field	Bits	Type	Description
0	[31:1]	r	**Reserved** Read as 0; should be written with 0.

OSCHPCTRL

Control register of the OSCHP oscillator.

OSCHPCTRL
OSCHP Control Register **(0704_H)** **Reset Value: 0000 003C$_H$**



Field	Bits	Type	Description
X1DEN	0	rw	**XTAL1 Data Enable** 0_B Bit X1D is not updated 1_B Bit X1D can be updated
SHBY	1	rwh	**Shaper Bypass** 0_B The shaper is not bypassed 1_B The shaper is bypassed
MODE	[5:4]	rw	**Oscillator Mode** 00_B External Crystal Mode and External Input Clock Mode. The oscillator Power-Saving Mode is not entered. 01_B OSC is disabled. The oscillator Power-Saving Mode is not entered. 10_B External Input Clock Mode and the oscillator Power-Saving Mode is entered 11_B OSC is disabled. The oscillator Power-Saving Mode is entered.

Field	Bits	Type	Description
OSCVAL	[20:16]	rw	**OSC Frequency Value** This bit field defines the divider value that generates the reference clock that is supervised by the oscillator watchdog. f_{OSC} is divided by OSCVAL + 1 in order to generate f_{OSCREF}.
0	2,3, [15:6], [31:21]	r	**Reserved** Read as 0; should be written with 0.

PLLSTAT

System PLL Status register.

PLLSTAT
PLL Status Register **(0710_H)** **Reset Value: $0000\ 0002_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						**PLLSP**	**PLLHV**	**PLLLV**	**BY**	**K2RDY**	**K1RDY**	**0**	**VCOLOCK**	**PWDSTAT**	**VCOBYST**
r						rh	rh	rh	rh	rh	rh	r	rh	rh	rh

Field	Bits	Type	Description
VCOBYST	0	rh	**VCO Bypass Status** 0_B Free-running / Normal Mode is entered 1_B Prescaler Mode is entered
PWDSTAT	1	rh	**PLL Power-saving Mode Status** 0_B PLL Power-saving Mode was not entered 1_B PLL Power-saving Mode was entered
VCOLOCK	2	rh	**PLL LOCK Status** 0_B PLL not locked 1_B PLL locked

Field	Bits	Type	Description
K1RDY	4	rh	**K1 Divider Ready Status** This bit indicates if the K1-divider operates on the configured value or not. This is of interest if the value is changed. 0_B K1-Divider does not operate with the new value 1_B K1-Divider operate with the new value
K2RDY	5	rh	**K2 Divider Ready Status** This bit indicates if the K2-divider operates on the configured value or not. This is of interest if the value is changed. 0_B K2-Divider does not operate with the new value 1_B K2-Divider operate with the new value
BY	6	rh	**Bypass Mode Status** 0_B Bypass Mode is not entered 1_B Bypass Mode is entered. Input f_{OSC} is selected as output f_{PLL}.
PLLLV	7	rh	**Oscillator for PLL Valid Low Status Bit** This bit indicates if the frequency output of OSC is usable for the VCO part of the PLL. This is checked by the Oscillator Watchdog of the PLL. 0_B The OSC frequency is not usable. Frequency f_{REF} is too low. 1_B The OSC frequency is usable
PLLHV	8	rh	**Oscillator for PLL Valid High Status Bit** This bit indicates if the frequency output of OSC is usable for the VCO part of the PLL. This is checked by the Oscillator Watchdog of the PLL. 0_B The OSC frequency is not usable. Frequency f_{OSC} is too high. 1_B The OSC frequency is usable

Field	Bits	Type	Description
PLLSP	9	rh	**Oscillator for PLL Valid Spike Status Bit** This bit indicates if the frequency output of OSC is usable for the VCO part of the PLL. This is checked by the Oscillator Watchdog of the PLL. 0_B The OSC frequency is not usable. Spikes are detected that disturb a locked operation 1_B The OSC frequency is usable
0	3, [31:10]	r	**Reserved** Read as 0.

PLLCON0

System PLL configuration register 0.

PLLCON0
PLL Configuration 0 Register (0714$_H$) Reset Value: 0003 0003$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0													RESLD	OSCRES	PLLPWD
r													w	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0									OSCDISCDIS	0	FINDIS	0	VCOTR	VCOPWD	VCOBYP
r									rw	r	rwh	r	rw	rw	rw

Field	Bits	Type	Description
VCOBYP	0	rw	**VCO Bypass** 0_B Normal operation, VCO is not bypassed 1_B Prescaler Mode, VCO is bypassed
VCOPWD	1	rw	**VCO Power Saving Mode** 0_B Normal behavior 1_B The VCO is put into a Power Saving Mode and can no longer be used. Only the Bypass and Prescaler Mode are active if previously selected.

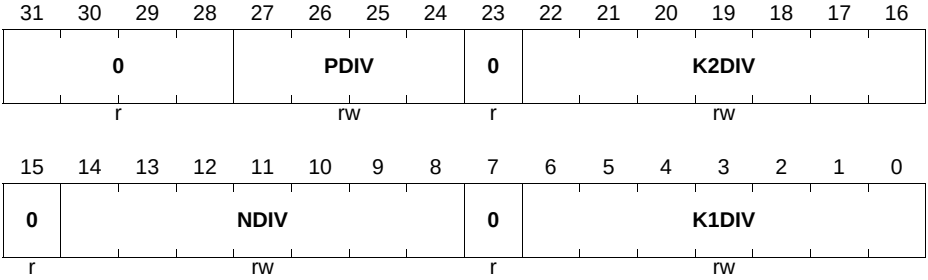
Field	Bits	Type	Description
VCOTR	2	rw	**VCO Trim Control** 0_B VCO bandwidth is operation in the normal range. VCO output frequency is between 260 and 520 MHz for a input frequency between 8 and 16 MHz. 1_B VCO bandwidth is operation in the test range. VCO output frequency is between 260 and 520 MHz for a input frequency between 8 and 16 MHz. Selecting a VCO trim value of one can result in a high jitter but the PLL is still operable.
FINDIS	4	rwh	**Disconnect Oscillator from VCO** 0_B connect oscillator to the VCO part 1_B disconnect oscillator from the VCO part.
OSCDISCDIS	6	rw	**Oscillator Disconnect Disable** This bit is used to disable the control FINDIS in a PLL loss-of-lock case. 0_B In case of a PLL loss-of-lock bit FINDIS is set 1_B In case of a PLL loss-of-lock bit FINDIS is cleared
PLLPWD	16	rw	**PLL Power Saving Mode** 0_B Normal behavior 1_B The complete PLL block is put into a Power Saving Mode and can no longer be used. Only the Bypass Mode is active if previously selected.
OSCRES	17	rw	**Oscillator Watchdog Reset** This bit controls signal *osc_fail_res_i* at the PLL module. 0_B The Oscillator Watchdog of the PLL is not cleared and remains active 1_B The Oscillator Watchdog of the PLL is cleared and restarted
RESLD	18	w	**Restart VCO Lock Detection** Setting this bit will clear bit PLLSTAT.VCOLOCK and restart the VCO lock detection. Reading this bit returns always a zero.

Field	Bits	Type	Description
0	3, 5, [15:7], [31:19]	r	**Reserved** Read as 0; should be written with 0.

PLLCON1

System PLL configuration register 1.

PLLCON1
PLL Configuration 1 Register **(0718_H)** **Reset Value: $0000\ 0000_H$**

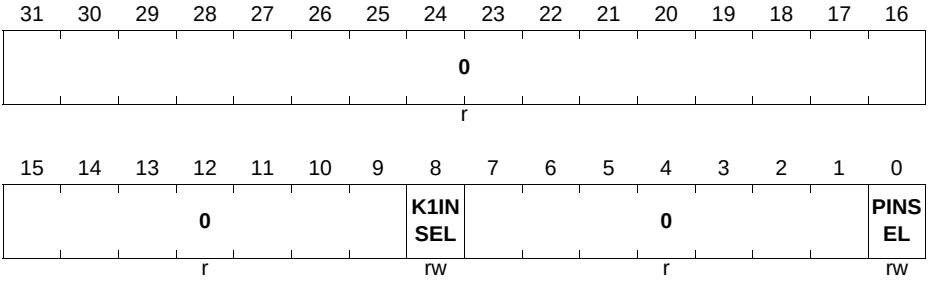


Field	Bits	Type	Description
K1DIV	[6:0]	rw	**K1-Divider Value** The value the K1-Divider operates is K1DIV+1.
NDIV	[14:8]	rw	**N-Divider Value** The value the N-Divider operates is NDIV+1.
K2DIV	[22:16]	rw	**K2-Divider Value** The value the K2-Divider operates is K2DIV+1.
PDIV	[27:24]	rw	**P-Divider Value** The value the P-Divider operates is PDIV+1.
0	7,15, 23, [31:28]	r	**Reserved** Read as 0; should be written with 0.

PLLCON2

System PLL configuration register 2.

PLLCON2
PLL Configuration 2 Register **($071C_H$)** **Reset Value: 0000 0001$_H$**

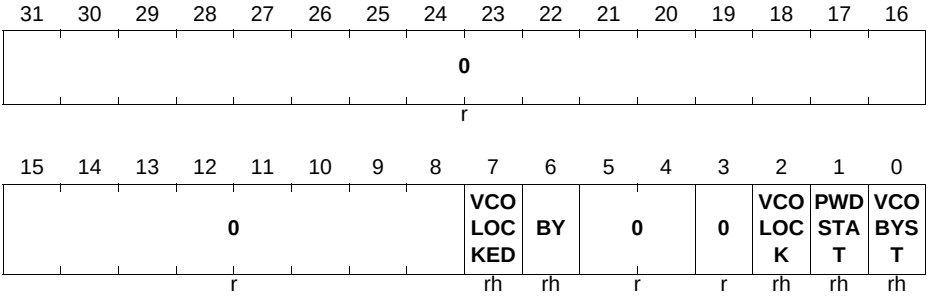


Field	Bits	Type	Description
PINSEL	0	rw	**P-Divider Input Selection** 0_B PLL external oscillator selected 1B Backup clock f_{ofi} selected
K1INSEL	8	rw	**K1-Divider Input Selection** 0_B PLL external oscillator selected 1_B Backup clock f_{ofi} selected
0	[7:1], [31:9]	r	**Reserved** Read as 0; should be written with 0.

USBPLLSTAT

USB PLL Status register.

USBPLLSTAT
USB PLL Status Register **(0720_H)** **Reset Value: 0000 0002$_H$**



Field	Bits	Type	Description
VCOBYST	0	rh	**VCO Bypass Status** 0_B Normal Mode is entered 1_B Prescaler Mode is entered
PWDSTAT	1	rh	**PLL Power-saving Mode Status** 0_B PLL Power-saving Mode was not entered 1_B PLL Power-saving Mode was entered
VCOLOCK	2	rh	**PLL VCO Lock Status** 0_B The frequency difference of f_{REF} and f_{DIV} is greater than allowed. The VCO part of the PLL can not lock on a target frequency. 1_B The frequency difference of f_{REF} and f_{DIV} is small enough to enable a stable VCO operation *Note: In case of a loss of VCO lock the f_{VCO} goes to the upper boundary of the VCO frequency if the reference clock input is greater than expected.* *Note: In case of a loss of VCO lock the f_{VCO} goes to the lower boundary of the VCO frequency if the reference clock input is lower than expected.*
BY	6	rh	**Bypass Mode Status** 0_B Bypass Mode is not entered 1_B Bypass Mode is entered. Input f_{OSC} is selected as output f_{PLL}.
VCOLOCKED	7	rh	**PLL LOCK Status** 0_B PLL not locked 1_B PLL locked
0	3, [5:4], [31:8]	r	**Reserved** Read as 0.

USBPLLCON

USB PLL configuration register 0.

USBPLLCON
USB PLL Configuration Register (0724$_H$) **Reset Value: 0001 0003$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0				PDIV				0					RES LD	0	PLL PWD
r				rw				r					w	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	NDIV							0	OSC DISC DIS	0	FIND IS	0	VCO TR	VCO PWD	VCO BYP
r	rw							r	rw	r	rwh	r	rw	rw	rw

Field	Bits	Type	Description
VCOBYP	0	rw	**VCO Bypass** 0_B Normal operation, VCO is not bypassed 1_B Prescaler Mode, VCO is bypassed
VCOPWD	1	rw	**VCO Power Saving Mode** 0_B Normal behavior 1_B The VCO is put into a Power Saving Mode
VCOTR	2	rw	**VCO Trim Control** 0_B VCO bandwidth is operating in the normal range. VCO output frequency is between 260 and 520 MHz for a input frequency between 8 and 16 MHz. 1_B VCO bandwidth is operating in the test range. VCO output frequency is between 260 and 520 MHz for a input frequency between 8 and 16 MHz. Selecting a VCO trim value of one can result in a high jitter but the PLL is still operable.
FINDIS	4	rwh	**Disconnect Oscillator from VCO** 0_B Connect oscillator to the VCO part 1_B Disconnect oscillator from the VCO part.

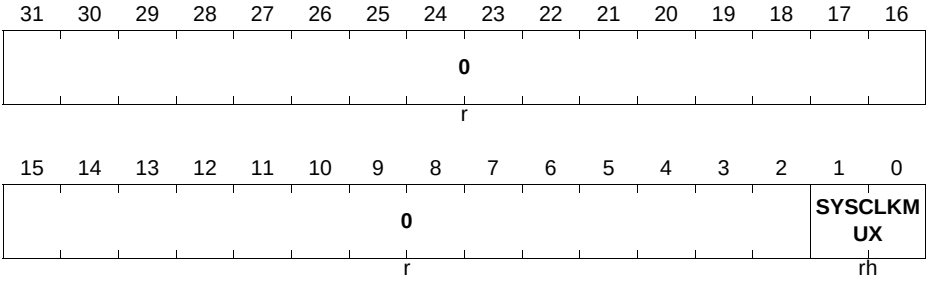
Field	Bits	Type	Description
OSCDISCDIS	6	rw	**Oscillator Disconnect Disable** This bit is used to disable the control FINDIS in a PLL loss-of-lock case. 0_B In case of a PLL loss-of-lock bit FINDIS is set 1_B In case of a PLL loss-of-lock bit FINDIS is cleared
NDIV	[14:8]	rw	**N-Divider Value** The value the N-Divider operates is NDIV+1.
PLLPWD	16	rw	**PLL Power Saving Mode** 0_B Normal behavior 1_B The complete PLL block is put into a Power Saving Mode. Only the Bypass Mode is active if previously selected.
RESLD	18	w	**Restart VCO Lock Detection** Setting this bit will clear bit PLLSTAT.VCOLOCK and restart the VCO lock detection. Reading this bit returns always a zero.
PDIV	[27:24]	rw	**P-Divider Value** The value the P-Divider operates is PDIV+1.
0	3, 5, 7, 15, 17, [23:19], [31:28]	r	**Reserved** Should be written with 0.

CLKMXSTAT

Clock Multiplexer Switching Status

This register shows status of clock multiplexing upon switching from one clock source to another. This register should be checked before disabling any of the multiplexer input clock sources after switching. Bits of this registers indicate which of the corresponding input clocks must not be switched off under any circumstances until indicated as inactive. The clocks sources that are indicated as active are still contributing in driving the output clock from respective multiplexer. This is a side effect of glitch-free clock switching mechanism.

CLKMXSTAT
Clock Multiplexing Status Register (0738$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
SYSCLKMUX	[1:0]	rh	**Status of System clock multiplexing upon source switching** Clock sources that are indicated active are still contributing in glitch-free switching x1$_B$ f_{OFI} clock active 1x$_B$ f_{PLL} clock active
0	[31:2]	r	**Reserved** Read as 0; should be written with 0.

11.11 Interconnects

The system control unit SCU is connected towards the system via bus interface. The bus interface provides register access for monitoring and controlling operation of all sub-modules of the SCU.

A number of signals connected to the submodules os SCU allow monitoring and control of various aspects of the system. The following tables provide logical grouping of different signals vital for the SCU module operation.

Table 11-11 GCU Signals

Input/Output	I/O	Connected To	Description
PARITY_ERROR	I	SRAMs	Parity error from on-chip SRAMs
NMI	O	CPU	NMI output
IRQ	O	CPU	Interrupt output

Table 11-11 GCU Signals (cont'd)

Input/Output	I/O	Connected To	Description
AHB0_BUS_ERR	I	PBA0	Bus error indication for PBA 0
AHB1_BUS_ERR	I	PBA1	Bus error indication for PBA 1
SCU_ETH_PHY_ RMII_O	O	ETH	Used to select the PHY interface from outside Ethernet MAC during reset of MAC.
G0ORCEN6	O	PORTS	Enable Out-Of-Range comparator of VADC kernel 0, channel 6
G0ORCEN6	O	PORTS	Enable Out-Of-Range comparator of VADC kernel 0, channel 7
G1ORCEN6	O	PORTS	Enable Out-Of-Range comparator of VADC kernel 1, channel 6
G1ORCEN7	O	PORTS	Enable Out-Of-Range comparator of VADC kernel 1, channel 7
DBGEN	O	Debug	Debug enable, driven by DBGCON0.DAPSA
alarm	I	RTC	RTC alarm
periodic_event	I	RTC	RTC periodic event

Table 11-12 PCU Signals

Input/Output	I/O	Connected To	Description
POWER_STATUS	O	PORTS	Indication of power status to PORTS

Table 11-13 HCU Signals

Input/Output	I/O	Connected To	Description
HIB_SR0	O	Hibernate Domain	GPI

Table 11-13 HCU Signals (cont'd)

Input/Output	I/O	Connected To	Description
HIB_SR1	O	Hibernate Domain	GPI
WWDT_SERVICE_OUT	I	Hibernate Domain	WDT service signal to external Timeout Watchdog

Table 11-14 RCU Signals

Input/Output	I/O	Connected To	Description
$\overline{\text{PORST}}$	I	Core Modules	Power-on reset
$\overline{\text{SYSRESET}}$	O	Core Modules	Reset signals to core modules
$\overline{\text{STDBYRESET}}$	O	Hibernate Domain	Reset signals to hibernate domain logic
$\overline{\text{DBGRESET}}$	O	DAP	Reset signals to debug logic
$\overline{\text{SYSRSTREQ}}$	I	CPU	Reset request from CPU
wdt_rst_req_n	I	WDT	Reset request from watchdog module
LOCKUP	I	CPU	Lockup signal from CPU
DBG_PRESENT	I	Debug	Debugger present. Connect to CDBGPWRUPREQ

Table 11-15 CCU Signals

Input/Output	I/O	Connected To	Description
External Signals			
XTAL1/CLKIN	I		Crystal pin or external clock input
XTAL2	O		Crystal pin
EXTCLK	O		External clock output
Other Signals			
SLEEPING	I	Debug	Indicates that processor is in sleep mode
SLEEPDEEP	I	Debug	Indicates that processor is in deep sleep mode
System Clock Signals			
f_{SYS}	O	SCU.CCU	System master clock

Table 11-15 CCU Signals (cont'd)

Input/Output	I/O	Connected To	Description
f_{CPU}	O	CPU	CPU and NVIC clock
f_{DMA}	O	DMA0, DMA1	DMA clock
f_{PERIPH}	O	PBA0 PBA1	Peripheral clock for modules connected to peripheral bridges PBA0 and PBA1
f_{CCU}	O	CCU4 CCU8 POSIF	Clock for CCU4, CCU8 and POSIF modules
f_{USB}	O	USB	UTMI clock of USB
f_{ETH}	O	ETH0	ETH clock
f_{EBU}	O	EBU	EBU clock
f_{WDT}	O	WDT	Clock for independent watchdog timer
External Clock Signals			
f_{OSCHP}	I	XTAL1/CLKIN	External crystal input, optionally also direct input clock to system PLL
f_{OSCULP}	I	RTC_XTAL1	External crystal input, optionally also direct input clock to Hibernate domain and RTC module
f_{EXT}	O	Clock output to pin EXTCLK	Chip output clock

Communication Peripherals

12 LED and Touch-Sense (LEDTS)

The LED and Touch-Sense (LEDTS) drives LEDs and controls touch pads used as human-machine interface (HMI) in an application.

Table 12-1 Abbreviations

LEDTS	LED and Touch-sense
TSD	time slice duration
TFD	time frame duration
TPD	time period duration

12.1 Overview

The LEDTS can measure the capacitance of up to 8 touch pads using the relaxation oscillator (RO) topology. The pad capacitance is measured by generating oscillations on the pad for a fixed time period and counting them. The module can also drive up to 64 LEDs in an LED matrix. Touch pads and LEDs can share pins to minimize the number of pins needed for such applications. This configuration is realized by the module controlling the touch pads and driving the LEDs in a time-division multiplexed manner.

The LEDs in the LED matrix are organized into columns and lines. Every line can be shared between up to 8 LEDs and one touch pad. Certain functions such as column enabling, function selection and control are controlled by hardware. Application software is required to update the LED lines and evaluate the touch pad measurement results.

12.1.1 Features

For the LED driving function, LEDTS provides features:

- Selection of up to 8 LED columns; Up to 7 LED columns if touch-sense function is also enabled
- Configurable active time in LED columns to control LED brightness
- Possibility to drive up to 8 LEDs per column, common-anode or common-cathode
- Shadow activation of line pattern for LED column time slice; LED line patterns are updated synchronously to column activation
- Configurable interrupt enable on selected event
- Line and column pins controlled by PORTS SFR setting

For the touch-sensing function, LEDTS provides features:

- Up to 8 touch-sense input turns
- Only one pad can be measured at any time; selection of active pad controllable by software or hardware round-robin
- Flexible measurement time on touch pads
- Pin oscillation control circuit with adjustments for oscillation

- 16-bit counter: For counting oscillations at pin
- Configurable interrupt enable on selected event
- Pin over-rule control for active touch input line (pin)

*Note: This chapter refers to the LED or touch-sense pins, e.g. 'pin COL[x]', 'pin TSIN[x]'. In all instances, it refers to the user-configured pin(s) which selects the LED/touch-sense function. Refer to **Section 12.9.5** for more elaboration.*

Table 12-2 LEDTS Applications

Use Case	Application
Non-mechanical switch	HMI
LED feedback	HMI
Simple PWM	PWM

12.1.2 Block Diagram

The LEDTS block diagram is shown in **Figure 12-1**.

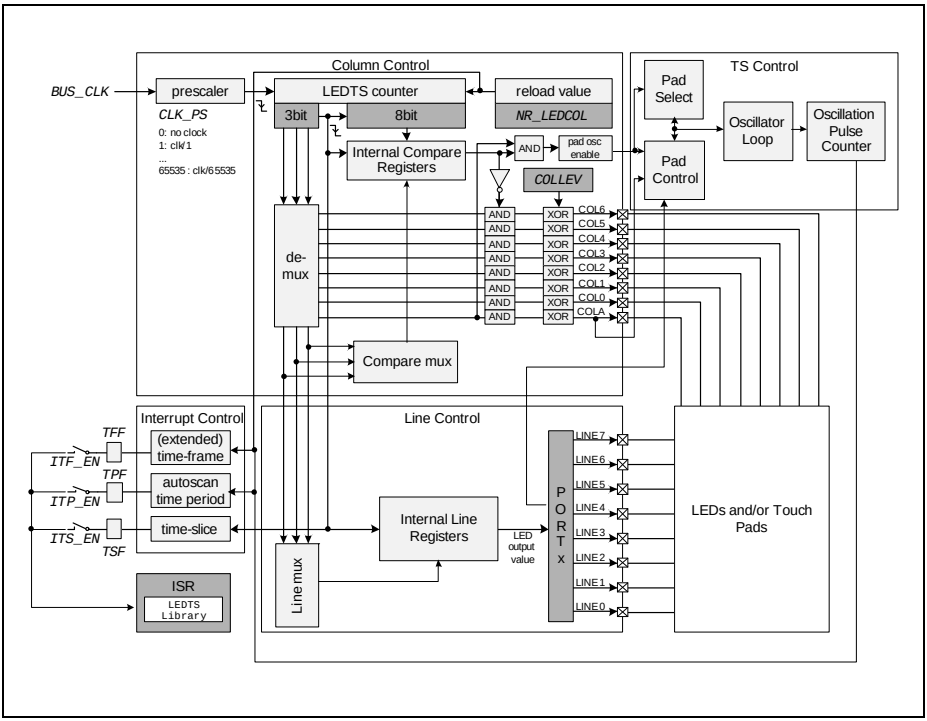


Figure 12-1 LEDTS Block Diagram

12.2 Functional Overview

The same pin can support LED & touch-sense functions in a time-multiplexed manner. LED mode or touch-sense mode can be enabled by hardware for respective function controls.

Time-division multiplexing is done by dividing the time domain into time slots. This basic time slot is called a **time slice**. In one time slice, one LED column is activated or the capacitance of one touch pad is measured.

A **time frame** is composed of 1 or more time slices, up to a maximum of eight. There is one time slice for every LED column enabled. If only LED function is enabled, a time frame can compose up to 8 LED time slices. However, if touch sense function is enabled, the last time slice in every time frame is reserved for touch-sense function. This reduces the maximum number of time slices that can be used for LED function in each time frame to 7. Only one time slice is used for touch sense function in every time frame. This is regardless of the number of touch pads enabled.

In each time slice used for LED function, only one LED column is enabled at a time. In the time slice reserved for touch-sense function, oscillations are enabled and measured on the pin which is activated. No LED column is active during this time slice. A touch pad input line (TSIN[x] pin) is active when its pad turn is enabled. If more than one touch pad input lines are enabled, the enabling and measurement on the touch pads in performed in a round robin manner. Only one touch pad is measured in every time frame.

The resolution of oscillation measurement can be increased by accumulating oscillation counts on each touch input line. When enabled by configuration of “Accumulate Count” (ACCCNT), the pad turn can be extended on consecutive time frames by up to 16 times. This also means that the same touch pad will be measured in consecutive time frames. This control will be handled by hardware. Otherwise it is also possible to enable for software control where the active pad turn is fully under user control.

The number of consecutive time frames, for which a pad turn has been extended, forms an **extended time frame**. When touch-sense function is enabled for automatic hardware pad turn control, several (extended) time frames make up one **autoscan time period** where all pad turns are completed. The time slice duration is configured centrally for the LED and/or touch-sense functions, using the LEDTS-counter. Refer to the description in **Section 12.3**, **Section 12.9.3** and **Figure 12-4**.

If enabled, a time slice interrupt is triggered on overflow of the 8LSBs of the LEDTS-counter for each new time slice started. The (extended) time frame interrupt may also be enabled. It is triggered on (the configured counts of) overflow of the whole LEDTS-counter. The autoscan time period interrupt may also be enabled. However, this interrupt will require that the hardware pad turn control is enabled. It is triggered when hardware completes the last pad turn on the highest enabled touch input line TSIN[NR_TSIN].

The column activation and pin oscillation duty cycles can be configured for each time slice. This allows the duration of activation of LED columns and/or touch-sense

oscillation counting to be flexible. This is also how the relative brightness of the LEDs can be controlled. In case of touch pads, the activation time is called the oscillation window.

Figure 12-1 shows an example for a LED matrix configuration with touch pads. The configuration in this example is 8 X 4 LED matrix with 4 touch input lines (here: 6 touchpads with two being “dual-pad”) enabled in sequence by hardware. Here no pad turn is extended by ACCCNT, so four time frames complete an autoscan time period.

In the time slice interrupt, software can:

- set up line pattern for next time slice
- set up compare value for next time slice
- evaluate current function in time slice (especially for analysis/debugging)

Refer to **Section 12.9.1** for **Interpretation of Bit Field FNCOL** to determine the currently active time slice.

The (extended) time frame interrupt indicates one touch input line TSIN[x] has been sensed (once or number of times in consecutive frames), application-level software can, for example:

- start touch-sense processing (e.g. filtering) routines and update status
- update LED display data to SFR

In the autoscan time period interrupt which indicates all touch-sense input TSIN[x] have been scanned one round, application-level software can:

- evaluate touch detection result & action
- update LED display data to SFR

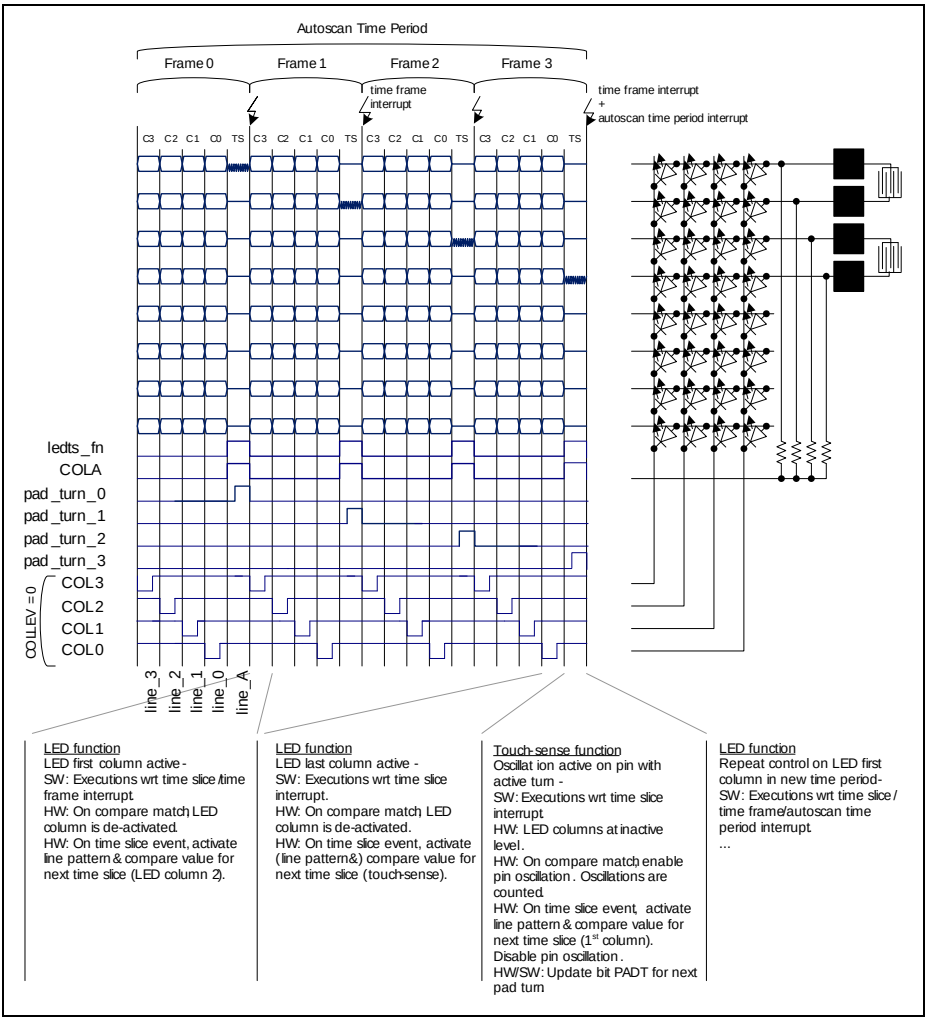


Figure 12-2 Time-Multiplexed LEDTS Functions on Pin (Example)

12.3 LED Drive Mode

LED driving is supported mainly for LED column selection and line control. At one time, only one column is active. The corresponding line level at high or low determines if the associated LED on column is lit or not. Up to eight columns are supported, and up to eight LEDs can be controlled per column.

With direct LED drive, adjustment of luminence for different types of LEDs with different forward voltages is supported. A compare register for LEDTS-counter is provided so that the duty cycle for column enabling per time slice can be adjusted. The LED column is enabled from the beginning of the time slice until compare match event. For 100% duty cycle for LED column enable in time slice, the compare value should be set to FFH. If the compare value is set to 00H, the LED column will stay at passive level during the time slice. Update of the internal compare register for each time slice is by shadow transfer from the corresponding compare SFR which takes place automatically at the beginning of each time slice, refer **Figure 12-3**.

A similar shadow transfer mechanism to update the LED line pattern (LED enabling) per column (time slice) is also provided, as illustrated in **Figure 12-3**. This shadow transfer of the corresponding line pattern to the internal line SFR takes place automatically at the beginning of each new time slice.

Note: Any write to any compare or line SFR within the time slice does not affect the internal latched configuration of current time slice.

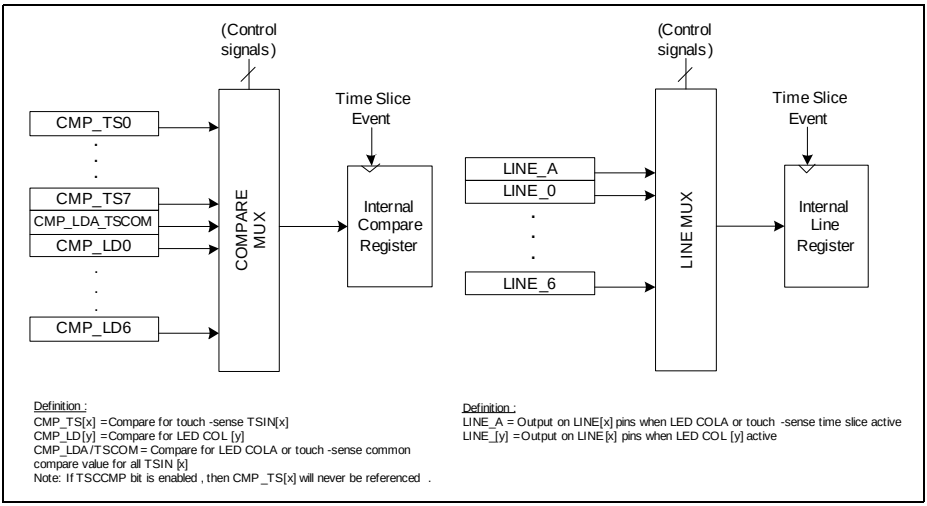


Figure 12-3 Activate Internal Compare/Line Register for New Time Slice

When the LEDTS-counter is first started (enable input clock by CLK_PS), a shadow transfer of line pattern and compare value is activated for the first time slice (column).

A time slice interrupt can be enabled. A new time slice starts on the overflow of the 8LSBs of the LEDTS-counter.

Figure 12-4 shows the LED function control circuit. This circuit also provides the control for enabling the pad oscillator. A 16-bit divider provides pre-scale possibilities to flexibly configure the internal LEDTS-counter count rate, which overflows in one time frame at the end. During a time frame comprising a configurable number of time slices, the configured number of LED columns are activated in sequence. In the last time slice of the time frame, touch-sense function is activated if enabled.

The LEDTS-counter is started when bit CLK_PS is set to any value other than 0 and either the LED or touch-sense function is enabled. It does not run when both functions are disabled. To avoid over-write of function enable which disturbs the hardware control during LEDTS-counter running, the TS_EN and LD_EN bits can only be modified when bit CLK_PS = 0. It is nonetheless possible to set the bits TS_EN and LD_EN in one single write to SFR **GLOBCTL** when setting CLK_PS from 0 to 1, or from 1 to 0.

When started, the counter starts running from a reset/reload value based on enabled function(s): 1) the number of columns (bit-field NR_LEDCOL) when LED function is enabled, 2) add one time slice at end of time frame when touch-sense function is enabled. The counter always counts up and overflows from $7FF_H$ to the reload value which is the same as the reset value. Within each time frame, the sequence of LED column enabling always starts from the most-significant enabled column (column with highest numbering).

To illustrate this point, in the case of four LED columns enabled, the column enabling sequence will be as follows:

- Start with COL3,
- followed by COL2,
- followed by COL1,
- followed by COL0,
- then COLA for touch sense function.

If touch-sense function is not enabled, COLA will be available for LED function as the last LED column time slice of a time frame. The column enabling sequence will then be as follows:

- Start with COL2,
- followed by COL1,
- followed by COL0,
- then COLA.

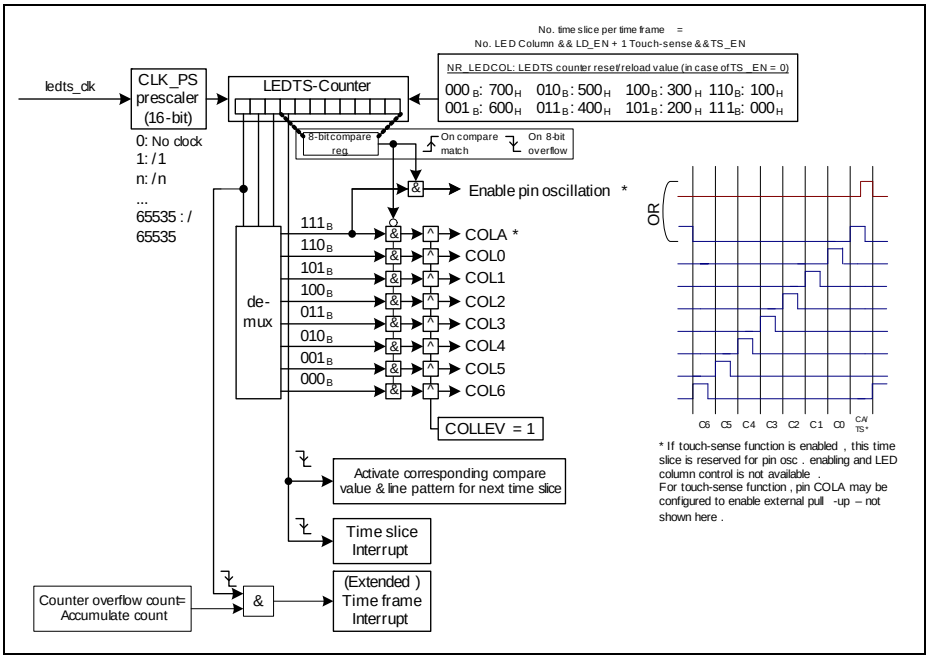


Figure 12-4 LED Function Control Circuit (also provides pad oscillator enable)

In **Section 12.9.3**, the time slice duration and formulations for LEDTS related timings are provided.

12.3.1 LED Pin Assignment and Current Capability

One LED column pin is enabled within each configured time slice duration to control up to eight LEDs at a time. The assignment of COL[x] to pins is configurable to provide options for application pin usage. The current capability of device pins is also a consideration factor for deciding pin assignment to LED function.

The product data-sheet provides data for all I/O parameters relevant to LED drive.

12.4 Touchpad Sensing

Figure 12-5 shows the pin oscillation control unit, which is integrated with the standard PORTS pad. An active pad turn (pad_turn_x) is defined for the touch-sense input pin TSIN[x] as the duration within the touch-sense time slice where the TS-counter is counting oscillations on the pin. In case of hardware pad turn control enabled (default), the same TS-counter is connected sequentially on enabled touch-sense inputs to execute a round-robin touch-sensing of TSIN[x] pins.

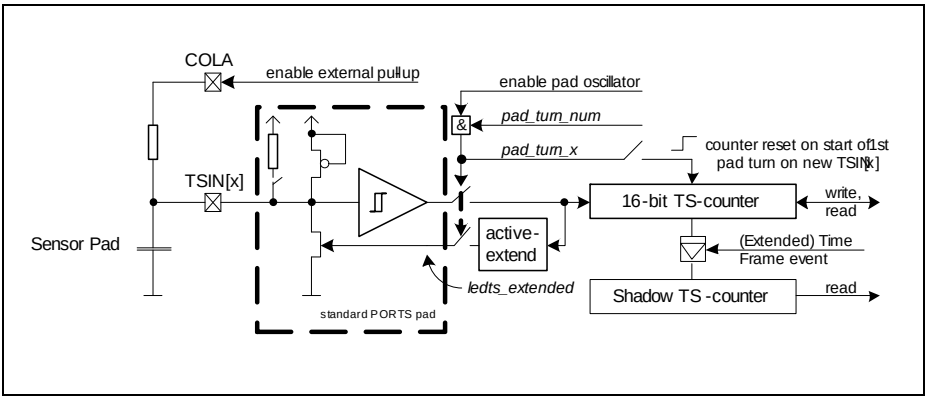


Figure 12-5 Touch-Sense Oscillator Control Circuit

Example in case of four touch-sense inputs, in ordered sequence of touch-sense time slice in sequential frames:

- Always start with TSIN0,
- followed by TSIN1 in touch-sense time slice of next frame,
- followed by TSIN2 in next touch-sense time slice,
- then TSIN3, and repeat.

It is possible to enable the touch-sense time slice on the same touch-sense input for consecutive frames (up to 16 times), to accumulate the oscillation count in TS-counter (see **Figure 12-6**). To illustrate this point, in the same case of four touch input lines enabled, and 2 accumulation counts configured, is as follows:

- Always starts with TSIN0,
- followed by TSIN0 again in touch-sense time slice of next frame,
- followed by TSIN1 in touch-sense time slice of next frame,
- followed by TSIN1 again in touch-sense time slice of next frame,
- followed by TSIN2 in touch-sense time slice of next frame,
- followed by TSIN2 again in touch-sense time slice of next frame,
- followed by TSIN3 in touch-sense time slice of next frame,
- then TSIN3 again, and repeat cycle.

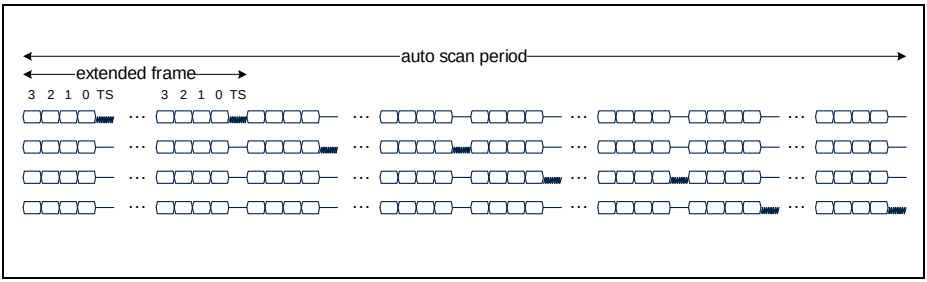


Figure 12-6 Hardware-Controlled Pad Turns for Autoscan of Four TSIN[x] with Extended Frames

There is a 16-bit TS-counter register and there is a 16-bit shadow TS-counter register. The former is both write- and read-accesible, while the latter is only read-accessible. The actual TS-counter counts the latched number of oscillations and can only be written when there is no active pad turn. The content of the TS-counter is latched to the shadow register on every (extended) time frame event. Reading from the shadow register therefore shows the latest valid oscillation count on one TSIN[x] input, ensuring for the application SW there is at least one time slice duration to get the valid oscillation count and meanwhile the actual TS-counter could continually update due to enabled pin oscillations in current time slice.

The TS-counter and shadow TS-counter have another user-enabled function on (extended) time frame event, which is to validate the counter value differences. When this function is enabled by the user and in case the counter values do not differ by 2^n LSB bits ('n' is configurable), the (extended) time frame interrupt request is gated (no interrupt) and the time frame event flag TFF is not set. This gating is on top of the time frame interrupt enable/disable control.

The TS-counter may be enabled for automatic reset (to 00_H) on the start of a new pad turn on the next TSIN[x], i.e. resets in the first touch-sense time slice of each (extended) time frame. Bit TSCTROVF indicates that the counter has overflowed. Alternatively, it can be configured such that the TS-counter stops counting in touch-sense time slice(s) of the same extended frame when the count value saturates, i.e. does not overflow & stops at $FFFF_H$. In this case, the TS-counter starts running again only in a new (extended) frame on the start of a new pad turn on the next TSIN[x].

A configurable pin-low-level active extension is provided for adjustment of oscillation per user system. The extension is active during the discharge phase of oscillation, and can be configured to be extended by a number of peripheral clocks. This function is very useful if there is a series resistor between the pin and the touch pad which makes the discharge slower. **Figure 12-7** illustrates this function.

The configuration of the active touch-sense pin TSIN[x] is over-ruled by hardware in the active duration to enable oscillations, reference **Section 12.9.5**. In particular, the weak

internal pull-up enable over-rule can be optionally de-activated (correspondingly internal pull-down disable over-rule is also de-activated; PORTS pin SFR setting for pull applies instead), such as when the user system utilize external resistor for pull-up instead. In the whole duration of the touch-sense time slice, COLA is activated high. This activates a pull-up via an external resistor connected to pin COLA. This configuration provides some flexibility to adjust the pin oscillation rate for adaptation to user system.

The touch-sense function is time-multiplexed with the LED function on enabled LINE[x]/TSIN[x] pins. During the touch-sense time slice for the other TSIN pins which are not on active pad turn, the corresponding LINE[x] output remains active. Software should take care to set the line bits to 1 to avoid current sink from pin COLA.

The touch-sense function is active in the last time slice of a time frame. Refer to **Section 12.2**, and **Section 12.3** for more details on time slice allocation and configuration.

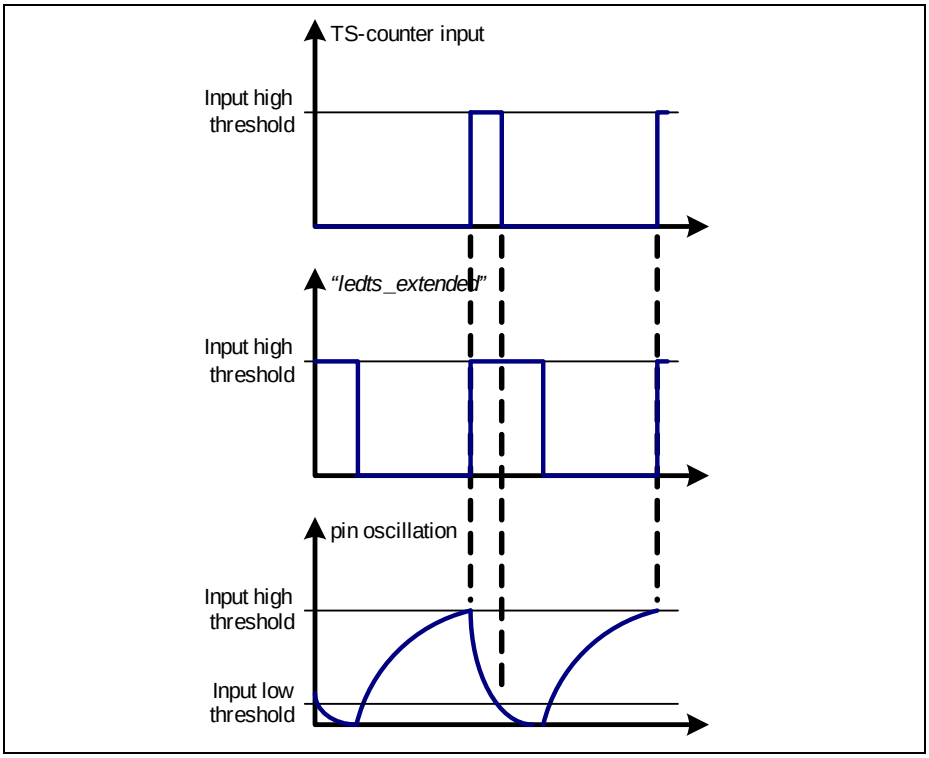


Figure 12-7 Pin-Low-Level Extension Function

The oscillation is enabled on the pin with valid turn for a configurable duration. A compare value provides the means to adjust the duty cycle within the time slice. The pin oscillation is enabled (TS-counter is counting) only on compare match until the end of the time slice. The time interval, in which the TS-counter is counting, is called the oscillation window. For a 100% duty cycle, the compare value has to be set to 00_H. In this case, the oscillation window fills in the entire time slice. Setting the compare value to FF_H results in no pin oscillation in time slice.

The time slice interrupt, (extended) time frame interrupt and/or autoscan time period interrupt may be enabled as required for touch-sense control.

12.4.1 Finger Sensing

When a finger is placed on the sensor pad, it increases the pin capacitance and frequency of oscillation on pin is reduced. Various factors affect the oscillation frequency including the size of touch pad, ground planes around and below the pad, the material and thickness of the overlay cover, the trace length and the individual pin itself (every pin has a different pull-up resistance).

In a real-world application, the printed circuit board (PCB) will not be touched directly. Instead, there is usually some sort of a transparent cover material, like a piece of plexiglass sheet, glued onto the PCB. In most of these applications, the oscillation frequency will change by about 2-10% when touched. This change in oscillation frequency can be considered to be very small, and therefore, further signal processing is necessary for reliable detection. Typically, this processing takes the form of a moving average calculation. It is never recommended to try to detect touches based on the raw oscillation count value.

As described in above section, some flexibility is provided to adjust the oscillation frequency in the user system: 1) Configurable pin low-level active extension, 2) Alternative enabling of external pull-up with resistance selectable by user. With a configurable time slice duration, the software can configure the duration of the active pad turn (adjustable within time slice using compare function) and set a count threshold for oscillations to detect if there is a finger touch or not.

To increase touch-sensing oscillation count accuracy, the input clock to LEDTS kernel should be set as high as possible.

12.5 Operating both LED Drive and Touch-Sense Modes

It is possible to enable both LED driving and touch-sense functions in a single time frame. If both functions are enabled, up to 7 time slices are configurable for the LED function, and the last time slice is reserved for touch-sensing function.

The touch-sense function is time-multiplexed with the LED function on enabled LINE[x]/TSIN[x] pins. During the touch-sense time slice (COLA), the corresponding LINE[s] output remains active for the other TSIN pins which are not on active pad turn.

In a typical application, COLA is not used and the oscillation is generated by the internal pad structure only. The bits in LINE_A will determine whether the pads, that are not being measured in the given COLA time slice, have a floating or 0V value. This setting usually has a serious effect on the sensitivity and noise robustness of the touch pads.

Refer to **Section 12.2** and **Section 12.3** for more details on time slice allocation and configuration.

12.6 Service Request Processing

There are three interrupts triggered by LEDTS kernel, all assigned on same node: 1) time slice event, 2) (extended) time frame event, 3) autoscan time period event. The flags are set on event or when CLK_PS is set from 0 regardless of whether the corresponding interrupt is enabled or not. When enabled, the event (including setting of CLK_PS from 0) activates the SR0 interrupt request from the kernel.

Table 12-3 lists the interrupt event sources from the LEDTS, and the corresponding event interrupt enable bit and flag bit.

Table 12-3 LEDTS Interrupt Events

Event	Event Interrupt Enable Bit	Event Flag Bit
Start of Time Slice	GLOBCTL.ITS_EN	EVFR.TSF
Start of (Extended) Time Frame[1)]	GLOBCTL.ITF_EN	EVFR.TFF
Start of Autoscan Time Period	GLOBCTL.ITP_EN	EVFR.TPF

1) In case of consecutive pad turns enabled on same TSIN[x] pin by ACCCNT bit-field, interrupt is not triggered on a time frame – but on the extended time frame.

Table 12-4 shows the interrupt node assignment for each LEDTS interrupt source.

Table 12-4 LEDTS Events' Interrupt Node Control

Event	Interrupt Node Enable Bit	Interrupt Node Flag Bit	Node ID
Start of Time Slice	LEDTS0.SR0	LEDTS0.SR0	102
Start of (Extended) Time Frame			
Start of Autoscan Time Period			

12.7 Debug Behavior

The LEDTS timers/counters LEDTS-counter and TS-counter can be enabled (together) for suspend operation when debug mode becomes active (indicated by HALTED signal from CPU).

At the onset of debug suspend, these counters stop counting (retains the last value) for the duration of the device in debug mode. The function that was active in current time slice on the onset of debug suspend, continues to be active. When debug suspend is revoked, the kernel would resume operation according to latest SFR settings.

12.8 Power, Reset and Clock

The LEDTS kernel is clocked and accessible on the peripheral bus frequency. User should set up consistent time-slice durations for correct function by ensuring a constant frequency input clock when the kernel is in operation. It is recommended to set the input clock ledts_clk to highest frequency where possible for optimal touch-sensing accuracy.

Kernel is in operation in active mode except power-down modes where touch-sensing and LED functions are not available.

12.9 Initialisation and System Dependencies

This section provides hints for enabling the LEDTS functions and using them.

12.9.1 Function Enabling

It is recommended to set up all configuration for the LEDTS in all SFRs before write **GLOBCTL** SFR to enable and start LED and/or touch-sense function(s).

*Note: SFR bits especially affecting the LEDTS-counter configuration for LED/touch-sense function can only be written when the counter is not running i.e. CLK_PS = 0. Refer to SFR bit description **Section 12.10**.*

Enable LED Function Only

To enable LED function only: set LD_EN, clear TS_EN.

Initialization after reset:

```
MOV GLOBCTL, #0bXXXXXXXX XXXXXXXX XXX00000 0000XX10
     ;set LD_EN and start LEDTS-counter on prescaled clock
     ;(CLK_PS != 0)
```

Re-configuration during run-time:

```
MOV GLOBCTL, #0x0000X00X;stop LEDTS-counter by clearing prescaler
MOV GLOBCTL, #0bXXXXXXXX XXXXXXXX XXX00000 0000XX10
```

Enable Touch-Sense Function Only

To enable touch-sense function only: clear LD_EN, set TS_EN.

Initialization after reset:

```
MOV GLOBCTL, #0bXXXXXXXX XXXXXXXX XXX00000 0000XX01
     ;set TS_EN and start LEDTS-counter on prescaled clock
     ;(CLK_PS != 0)
```

Re-configuration during run-time:

```
MOV GLOBCTL, #0x0000X00X;stop LEDTS-counter by clearing prescaler
MOV GLOBCTL, #0bXXXXXXXX XXXXXXXX XXX00000 0000XX01
```

Enable Both LED and Touch-Sense Function

To enable both functions: set LD_EN, set TS_EN.

Initialization after reset:

```
MOV GLOBCTL, #XXXXXXXX XXXXXXXX XXX00000 0000XX11
     ;set TS_EN and start LEDTS-counter on prescaled clock
     ;(CLK_PS != 0)
```

Re-configuration during run-time:

```
MOV GLOBCTL, #0x0000X00X;stop LEDTS-counter by clearing prescaler
MOV GLOBCTL, #0bXXXXXXXX XXXXXXXX XXX00000 0000XX11
```

12.9.2 Interpretation of Bit Field FNCOL

The interpretation of the FNCOL bit field can be handled by software. The following example where six time slices are enabled (per time frame), with five LED columns and touch-sensing enabled, illustrates this (**Table 12-5**).

The FNCOL bit field provides information on the function/column active in the previous time slice. With this information, software can determine the active function/column in current time slice and prepare the necessary values (to be shadow-transferred) valid for the next time slice.

Referring to the example below, when the FNCOL bit field is 111_B, it can be derived that the touch-sensing function/column was active in the previous time slice and therefore the current active column is LED COL[4]. Hence, the software can update the shadow line and compare registers for LED COL[3] so that these changes will be reflected when LED COL[3] gets activated in the next time slice.

Table 12-5 Interpretation of FNCOL Bit Field

FNCOL	Active Function / Column in Current Time Slice	SW Prepare via “Shadow” Registers for Function / Column of Next Time Slice
111_H	LED COL[4]	LED COL[3]
010_H	LED COL[3]	LED COL[2]
011_H	LED COL[2]	LED COL[1]
100_H	LED COL[1]	LED COL[0]
101_H	LED COL[0]	Touch Input Line TSIN[PADT]
110_H	Touch Input Line TSIN[PADT]	LED COL[4]

12.9.3 LEDTS Timing Calculations

LEDTS main timing or duration formulation are provided in following.

Count-Rate (CR):

$$CR = (fCLK) \div (PREscaler) \quad (12.1)$$

where fCLK = LEDTS module input clock; PREscaler = **GLOBCTL**.CLK_PS

Time slice duration (TSD):

$$TSD = 2^8 \div (CR) \quad (12.2)$$

Time frame duration (TFD):

$$TFD = (\text{Number of time slices}) \times TSD \quad (12.3)$$

Extended TFD:

$$\text{ExtendedTFD} = ACCCNT \times TFD \quad (12.4)$$

where ACCCNT = **FNCTL**.ACCCNT

Autoscan time period duration (TPD):

$$TPD = (\text{Number of touch-sense inputs TSIN[x]}) \times TFD \quad (12.5)$$

LED drive active duration:

$$\text{LED Drive Active Duration} = TSD \times \text{Compare\_VALUE} \div 2^8 \quad (12.6)$$

Touch-sense drive active duration:

$$\text{Touch-sense Drive Active Duration} = TSD \times (2^8 - \text{Compare\_VALUE}) \div 2^8 \quad (12.7)$$

12.9.4 Time-Multiplexed LED and Touch-Sense Functions on Pin

Some hints are provided regarding the time-multiplexed usage of a pin for LED and touch-sense function:

- The maximum number of LED columns = 7 when touch-sense function is also enabled.
- If enabled by pin, COLA outputs HIGH to enable external R (resistor) as pull-up for touch-sense function.
- During touch-sense time slice, it is recommended to set LED lines to output LOW.
- During LED time slice, COLA outputs LOW and will sink current if connected lines output HIGH.
- The effective capacitance for each TSIN[x] depends largely on what is connected to the pin and the application board layout. All touch-pads for the application should be calibrated for robust touch-detection.

12.9.5 LEDTS Pin Control

The user may flexibly assign pins as provided by PORTS SFRs, for the LEDTS functions:

- COL[x] (for LED column control)
- LINE[x] (for LED line control)
- TSIN[x] (for touch-sensing)

Refer also to **Section 12.3** for more considerations with regards to which COL[x] and/or LINE[x]/TSIN[x] will be active based on user configuration.

User code must configure the assigned LED pin PORTS SFR alternate output selection for the LED function, see **Table 12-6** and **Figure 12-8**.

For the touch-sense function, it is also required to configure the PORTS SFRs to enable the hardware function on TSIN[x] pin (similarly alternate output COLA). However, the LEDTS will provide some pin over-rule controls to the assigned touch-sense pin with active pad turn, see **Table 12-6** and **Figure 12-8**.

Table 12-6 LEDTS Pin Control Signals

Function	ld/ts_en	ledts_fn	Pin	Control of Assigned Pin
LED column	LD_EN = 1	0 = LED	Enable COL[x]; Passive level on COL[the rest]. If TS_EN = 1, COLA = 0.	PORTS SFR setting

Table 12-6 LEDTS Pin Control Signals (cont'd)

Function	ld/ts_en	ledts_fn	Pin	Control of Assigned Pin
LED line	LD_EN = 1	0 = LED	LINE[x] = internal line register value latched from bit field LINE_[x]	PORTS SFR setting
Touch-sense	TS_EN = 1	1 = Touch-sense	Enable TSIN[x] for oscillation. All other TSIN pins output line value. Passive level on COL[the rest] except COLA = 1.	Hardware over-rule on pad_turn_x[1] for active duration: - Enable pull-up, disable pull-down (pull over-rule can be disabled by bit EPULL) - Set to output mode (both input, output stages enabled) - Enable open-drain

1) For the other pad inputs not on turn, there is no HW over-rule which means the PORTS SFR setting is active.

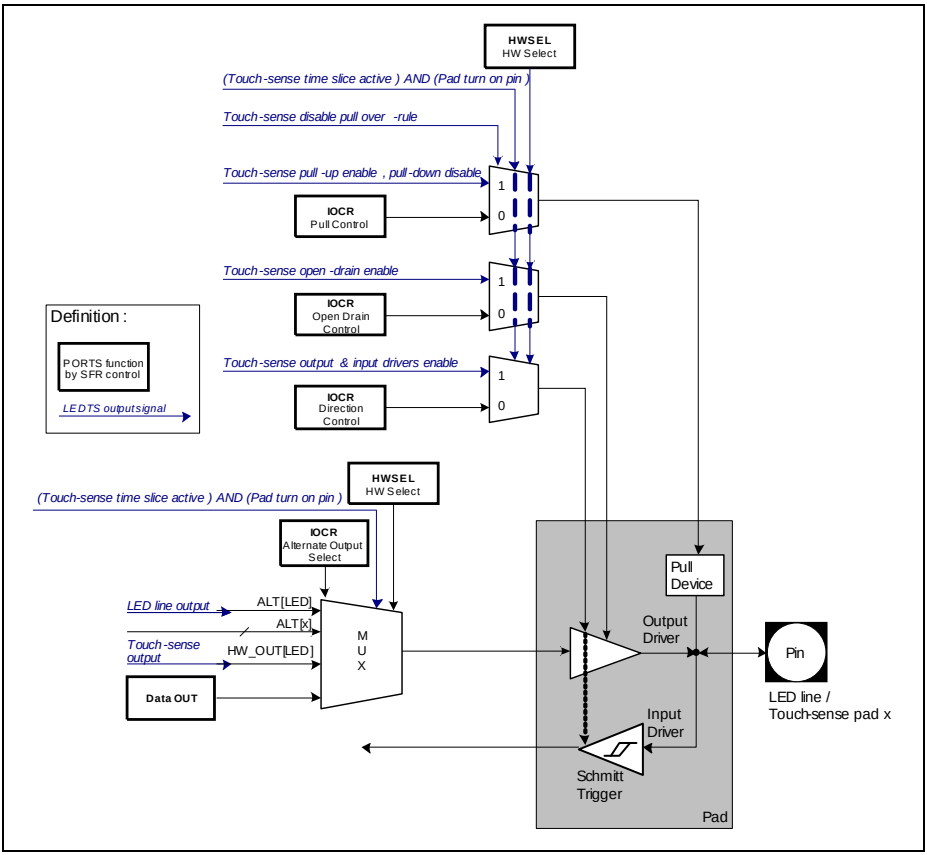


Figure 12-8 Over-rule Control on Pin for Touch-Sense Function

12.9.6 Software Hints

This section provides some useful software hints:

- Compare value 00_H enables oscillation for the full duration of the time slice, whereas FF_H disables oscillation.
- In order to maximize the resolution of the oscillation window, compare value should be selected to maximize the oscillation count without overflowing the TS-counter.
- Valid pad detection period (the time required to detect a valid touch on a pad) can be extended by:
  – enabling dummy LED columns (without assigning/setting the LED column pins)
  – selecting bigger pre-scale factor (**GLOBCTL**.CLK_PS)

  - accumulating the number of pad oscillations (**FNCTL**.ACCCNT)
- Valid pad detection period can be reduced by:
  - selecting smaller pre-scale factor (**GLOBCTL**.CLK_PS)
  - reducing the number of accumulations for pad oscillations (**FNCTL**.ACCCNT)

12.9.7 Hardware Design Hints

This section provides some hardware design hints:

- Touch button oscillation frequency changes when the value of the external pull-up resistor (connected to COLA pin) changes. This results in different sensitivity of the touch button as well as the crosstalk between the adjacent touch buttons.
  - A suitable pull-up resistor should be selected to balance the sensitivity of the touch button and the accuracy of the detection.
- The presence of LEDs modifies the equivalent capacitance for a touch pad. A larger number of LEDs connected to a touch pad will increase the self-capacitance of the pad. This makes the pad less sensitive.
- If possible, LEDs should be located near to the touch pads, to reduce the additional parasitic capacitance introduced by the traces.

12.10 Registers

Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 12-7 Registers Address Space

Module	Base Address	End Address	Note
LEDTS0	4801 0000$_H$	4801 00FF$_H$	

The prefix "**LEDTSx**_" is added to the register names in this table to indicate they belong to the LEDTS kernel.

Access rights within the address range of an LEDTS kernel:

- Read or write access to defined register addresses: U, PV
- Accesses to empty addresses: nBE

Table 12-8 Register Overview of LEDTS

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
ID	Module Identification Register	0000$_H$	U, PV	U, PV	Page 12-23
GLOBCTL	Global Control Register	0004$_H$	U, PV	U, PV	Page 12-24
FNCTL	Function Control Register	0008$_H$	U, PV	U, PV	Page 12-26
EVFR	Event Flag Register	000C$_H$	U, PV	U, PV	Page 12-29
TSVAL	Touch-Sense TS-Counter Value	0010$_H$	U, PV	U, PV	Page 12-31
LINE0	Line Pattern Register 0	0014$_H$	U, PV	U, PV	Page 12-32
LINE1	Line Pattern Register 1	0018$_H$	U, PV	U, PV	Page 12-32
LDCMP0	LED Compare Register 0	001C$_H$	U, PV	U, PV	Page 12-33
LDCMP1	LED Compare Register 1	0020$_H$	U, PV	U, PV	Page 12-34
TSCMP0	Touch-Sense Compare Register 0	0024$_H$	U, PV	U, PV	Page 12-35

Table 12-8 Register Overview of LEDTS (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
TSCMP1	Touch-Sense Compare Register 1	0028_H	U, PV	U, PV	Page 12-35
Reserved	Reserved	$002C_H$ - $1FFC_H$	nBE	nBE	

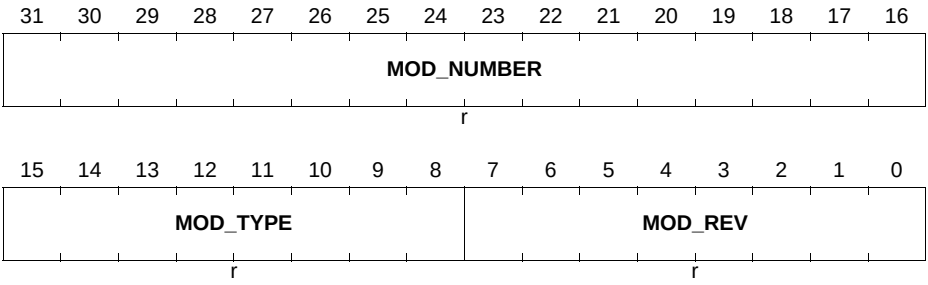
12.10.1 Registers Description

The LEDTS SFRs are organized into registers for global initialization control, functional control comprising TS-counter value, line pattern and compare value registers.

ID

The module identification register indicate the function and the design step of the peripheral.

ID
Module Identification Register **(0000_H)** **Reset Value: 00AB C0XX$_H$**



Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** MOD_REV defines the revision number. The value of a module revision starts with 01_H (first revision).
MOD_TYPE	[15:8]	r	**Module Type** This bit field is $C0_H$. It defines the module as a 32-bit module.

Field	Bits	Type	Description
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines the module identification number.

GLOBCTL

The GLOBCTL register is used to initialize the LEDTS global controls.

GLOBCTL
Global Control Register **(04_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CLK_PS															
rw															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ITP_EN	ITF_EN	ITS_EN	FENVAL	MASKVAL			SUSCFG	0				ENSYNC1)0	CMTR1)0	LD_EN1)0	TS_EN
rw	rw	rw	rw	rw			rw	r				rw	rw	rw	rw

Field	Bits	Type	Description
TS_EN[1)]	0	rw	**Touch-Sense Function Enable** Set to enable the kernel for touch-sense function control when CLK_PS is set from 0.
LD_EN[1)]	1	rw	**LED Function Enable** Set to enable the kernel for LED function control when CLK_PS is set from 0.
CMTR[1)]	2	rw	**Clock Master Disable** 0_B Kernel generates its own clock for LEDTS-counter based on SFR setting 1_B LEDTS-counter takes its clock from another master kernel
ENSYNC[1)]	3	rw	**Enable Autoscan Time Period Synchronization** 0_B No synchronization 1_B Synchronization enabled on Kernel0 autoscan time period

Field	Bits	Type	Description
SUSCFG	8	rw	**Suspend Request Configuration** 0_B Ignore suspend request 1_B Enable suspend according to request This bit is restored to default with Debug Reset.
MASKVAL	[11:9]	rw	**Mask Number of LSB Bits for Event Validation** This defines the number of LSB bits to mask for TS-counter and shadow TS-counter comparison when Time Frame validation is enabled. 0_D Mask LSB bit 1_D Mask 2 LSB bits ... 7_D Mask 8 LSB bits
FENVAL	12	rw	**Enable (Extended) Time Frame Validation** When enabled, TS-counter and shadow TS-counter values are compared to validate a Time Frame event for set flag and interrupt request. When validation fail, TFF flag does not get set and interrupt is not requested. 0_B Disable 1_B Enable
ITS_EN	13	rw	**Enable Time Slice Interrupt** 0_B Disable 1_B Enable
ITF_EN	14	rw	**Enable (Extended) Time Frame Interrupt** 0_B Disable 1_B Enable
ITP_EN	15	rw	**Enable Autoscan Time Period Interrupt** 0_B Disable 1_B Enable (valid only for case of hardware-enabled pad turn control)

Field	Bits	Type	Description
CLK_PS	[31:16]	rw	**LEDTS-Counter Clock Pre-Scale Factor** The constant clock input is prescaled according to setting. 0_D No clock 1_D Divide by 1 ... 65535_DDivide by 65535 This bit can only be set to any other value (from 0) provided at least one of touch-sense or LED function is enabled. The LEDTS-counter starts running on the input clock from reset/reload value based on enabled function(s) (and NR_LEDCOL). Refer **Section 12.3** for details. When this bit is clear to 0 from other value, the LEDTS-counter stops running and resets.
0	[7:4]	r	**Reserved** Read as 0; should be written with 0.

1) This bit can only be modified when bit CLK_PS = 0.

FNCTL

The FNCTL control register provides control bits for the LED and Touch Sense functions.

FNCTL
Function Control Register **(08_H)** **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NR_LEDCOL			COL LEV	NR_TSIN			TSC TRS AT	TSC TRR	TSOEXT		TSC CMP	ACCCNT			
rw			rw	rw			rw	rw	rw		rw	rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								FNCOL			EPU LL	PAD TSW	PADT		
r								rh			rw	rw	rwh		

Field	Bits	Type	Description
PADT	[2:0]	rwh	**Touch-Sense TSIN Pad Turn** This is the TSIN[x] pin that is next or currently active in pad turn. When PADTSW = 0, the value is updated by hardware at the end of touch-sense time slice. Software write is always possible. 0_D TSIN0 ... 7_D TSIN7
PADTSW[1)]	3	rw	**Software Control for Touch-Sense Pad Turn** 0_B The hardware automatically enables the touch-sense inputs in sequence round-robin, starting from TSIN0. 1_B Disable hardware control for software control only. The touch-sense input is configured in bit PADT.
EPULL	4	rw	**Enable External Pull-up Configuration on Pin COLA** When set, the internal pull-up over-rule on active touch-sense input pin is disabled. 0_B HW over-rule to enable internal pull-up is active on TSIN[x] for set duration in touch-sense time slice. With this setting, it is not specified to assign the COLA to any pin. 1_B Enable external pull-up: Output 1 on pin COLA for whole duration of touch-sense time slice. *Note: Independent of this setting, COLA always outputs 1 for whole duration of touch-sense time slice.*
FNCOL	[7:5]	rh	**Previous Active Function/LED Column Status** Shows the active function / LED column in the previous time slice. Updated on start of new time-slice when LEDTS-counter 8LSB over-flows. Controlled by latched value of the internal DE-MUX, see **Figure 12-4**.

Field	Bits	Type	Description
ACCCNT[1)]	[19:16]	rw	**Accumulate Count on Touch-Sense Input** Defines the number of times a touch-sense input/pin is enabled in touch-sense time slice of consecutive frames. This provides to accumulate oscillation count on the TSIN[x]. 0_D 1 time 1_D 2 times ... 15_D 16 times
TSCCMP	20	rw	**Common Compare Enable for Touch-Sense** 0_B Disable common compare for touch-sense 1_B Enable common compare for touch-sense
TSOEXT	[22:21]	rw	**Extension for Touch-Sense Output for Pin-Low-Level** 00_B Extend by 1 ledts_clk 01_B Extend by 4 ledts_clk 10_B Extend by 8 ledts_clk 11_B Extend by 16 ledts_clk
TSCTRR	23	rw	**TS-Counter Auto Reset** 0_B Disable TS-counter automatic reset 1_B Enable TS-counter automatic reset to 00H on the first pad turn of a new TSIN[x]. Triggered on compare match in time slice.
TSCTRSAT	24	rw	**Saturation of TS-Counter** 0_B Disable 1_B Enable. TS-counter stops counting in the touch-sense time slice(s) of the same (extended) frame when it reaches FFH. Counter starts to count again on the first pad turn of a new TSIN[x], triggered on compare match.
NR_TSIN[1)]	[27:25]	rw	**Number of Touch-Sense Input** Defines the number of touch-sense inputs TSIN[x]. Used for the hardware control of pad turn enabling. 0_D 1 ... 7_D 8

Field	Bits	Type	Description
COLLEV	28	rw	**Active Level of LED Column** 0_B Active low 1_B Active high
NR_LEDCOL[1)]	[31:29]	rw	**Number of LED Columns** Defines the number of LED columns. 000_B 1 LED column 001_B 2 LED columns 010_B 3 LED columns 011_B 4 LED columns 100_B 5 LED columns 101_B 6 LED columns 110_B 7 LED columns 111_B 8 LED columns (max. LED columns = 7 if bit TS_EN = 1) *Note: LED column is enabled in sequence starting from highest column number. If touch-sense function is not enabled, COLA is activated in last time slice.*
0	[15:8]	r	**Reserved** Read as 0; should be written with 0.

1) This bit can only be modified when bit CLK_PS = 0.

EVFR

The EVFR register contains the status flags for events and control bits for requesting clearance of event flags.

EVFR
Event Flag Register **($0C_H$)** **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0													CTPF	CTFF	CTSF
r													w	w	w

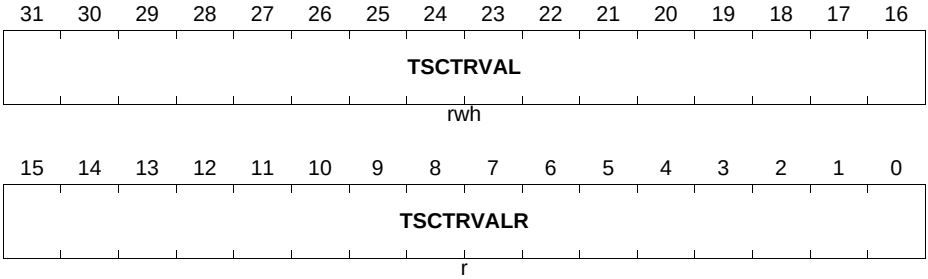
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0												TSCTROVF	TPF	TFF	TSF
r												rh	rh	rh	rh

Field	Bits	Type	Description
TSF	0	rh	**Time Slice Interrupt Flag** Set on activation of each new time slice, including when bit CLK_PS is set from 0. To be cleared by software.
TFF	1	rh	**(Extended) Time Frame Interrupt Flag** Set on activation of each new (extended) time frame, including when bit CLK_PS is set from 0.
TPF	2	rh	**Autoscan Time Period Interrupt Flag** Set on activation of each new time period, including when bit CLK_PS is set from 0. This bit will never be set in case of hardware pad turn control is disabled (bit PADTSW = 1).
TSCTROVF	3	rh	**TS-Counter Overflow Indication** This bit indicates whether a TS-counter overflow has occurred. This bit is cleared on new pad turn, triggered on compare match. 0_B No overflow has occurred. 1_B The TS-counter has overflowed at least once.
CTSF	16	w	**Clear Time Slice Interrupt Flag** 0_B No action. 1_B Bit TSF is cleared. Read always as 0.
CTFF	17	w	**Clear (Extended) Time Frame Interrupt Flag** 0_B No action. 1_B Bit TFF is cleared. Read always as 0.
CTPF	18	w	**Clear Autoscan Time Period Interrupt Flag** 0_B No action. 1_B Bit TPF is cleared. Read always as 0.
0	[31:19], [15:4]	r	**Reserved** Read as 0; should be written with 0.

TSVAL

The TSVAL register holds the current and shadow touch sense counter values.

TSVAL
Touch-sense TS-Counter Value (10_H) Reset Value: 0000 0000$_H$

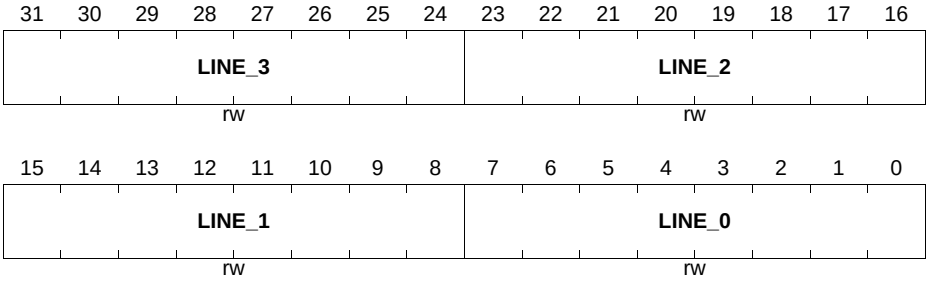


Field	Bits	Type	Description
TSCTRVALR	[15:0]	r	**Shadow TS-Counter (Read)** This is the latched value of the TS-counter (on every extended time frame event). It shows the latest valid oscillation count from the last completed time slice.
TSCTRVAL	[31:16]	rwh	**TS-Counter Value** This is the actual TS-counter value. It can only be written when no pad turn is active. The counter may be enabled for automatic reset once per (extended) frame on the start of a new pad turn on the next TSIN[x] pin.

LINEx (x = 0-1)

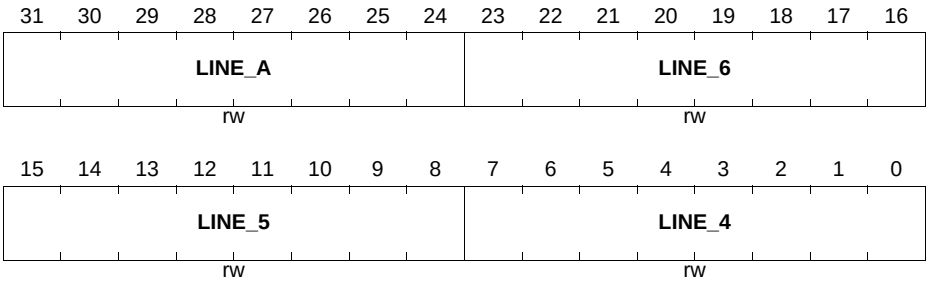
The LINEx registers hold the values that are output to the respective line pins during their active column period.

LINE0
Line Pattern Register 0 **(14_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
LINE_0, **LINE_1,** **LINE_2,** **LINE_3**	[7:0], [15:8], [23:16], [31:24]	rw	**Output on LINE[x]** This value is output on LINE[x] to pin when LED COL[x] is active.

LINE1
Line Pattern Register 1 **(18_H)** **Reset Value: 0000 0000$_H$**



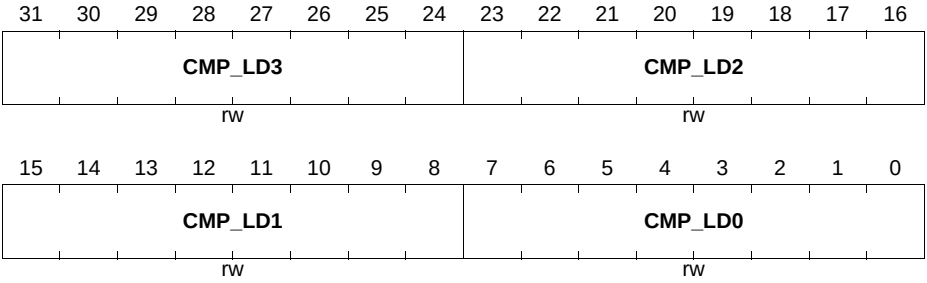
Field	Bits	Type	Description
LINE_4, **LINE_5,** **LINE_6**	[7:0], [15:8], [23:16]	rw	**Output on LINE[x]** This value is output on LINE[x] to pin when LED COL[x] is active.

Field	Bits	Type	Description
LINE_A	[31:24]	rw	**Output on LINE[x]** This value is output on LINE[x] to pin when LED COLA or touch-sense time slice is active.

LDCMPx (x = 0-1)

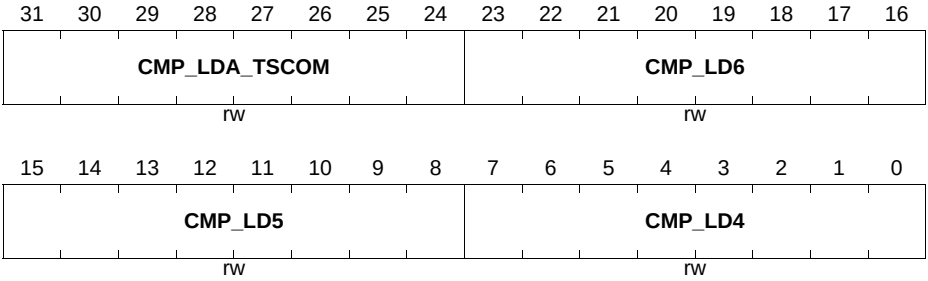
The LDCMPx registers hold the COMPARE values for their respective LED columns. These values are used for LED brightness control.

LDCMP0
LED Compare Register 0 **(1C$_H$)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
CMP_LD0, CMP_LD1, CMP_LD2, CMP_LD3	[7:0], [15:8], [23:16], [31:24]	rw	**Compare Value for LED COL[x]**

LDCMP1
LED Compare Register 1 **(20_H)** **Reset Value: 0000 0000$_H$**

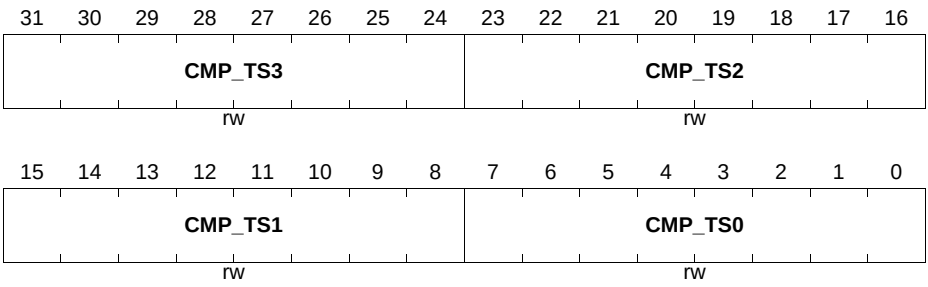


Field	Bits	Type	Description
CMP_LD4, CMP_LD5, CMP_LD6	[7:0], [15:8], [23:16]	rw	**Compare Value for LED COL[x]**
CMP_LDA_TSCOM	[31:24]	rw	**Compare Value for LED COLA / Common Compare Value for Touch-sense Pad Turns** LED function The compare value for LED COLA is only valid when touch-sense function is not enabled. Touch-sense function The common compare value for touch-sense pad turns is enabled by set TSCCMP bit. When enabled for common compare, settings in SFRs LEDTS_TSCMP0,1 are not referenced.

TSCMPx (x = 0-1)

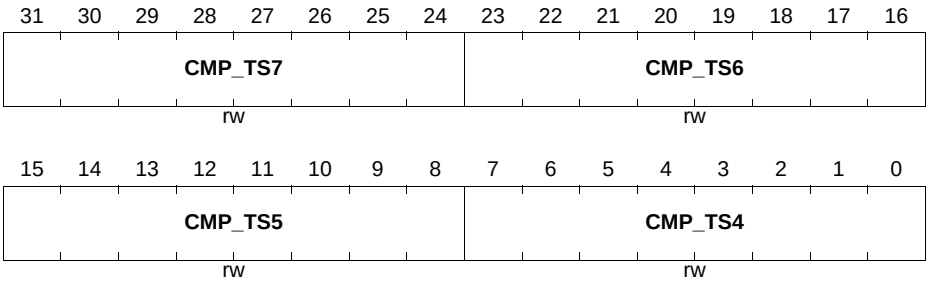
The TSCMPx registers hold the COMPARE values for their respective touch pad input lines. These values determine the size of the pad oscillation window for each pad input lines during their pad turn.

TSCMP0
Touch-sense Compare Register 0 (24_H) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
CMP_TS0, CMP_TS1, CMP_TS2, CMP_TS3	[7:0], [15:8], [23:16], [31:24]	rw	**Compare Value for Touch-Sense TSIN[x]**

TSCMP1
Touch-sense Compare Register 1 (28_H) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
CMP_TS4, CMP_TS5, CMP_TS6, CMP_TS7	[7:0], [15:8], [23:16], [31:24]	rw	**Compare Value for Touch-Sense TSIN[x]**

12.11 Interconnects

The LEDTS has interconnection to other peripherals enabling higher level of automation without requiring software. **Table 12-9** provides a list of the pin connections.

LEDTS.FN is an output signal denoting LEDTS active function. This signal can be used as a source for VADC request gating and background gating.

Table 12-9 Pin Connections

Input/Output	I/O	Connected To	Description
LEDTS.FN	O	VADC.GxREQGTJ	VADC request gating
		VADC.BGREQGTJ	VADC background gating input J

13 SD/MMC Interface (SDMMC)

This chapter describes the SD/MMC module. The XMC4500 uses the following SD and MMC card standard specification. For more detailed information on how to operate the SDMMC interface, please refer to the SD and MMC specification referenced below.

References

[10] SD Specifications Part A2, SD Host Controller Standard Specification, Version 2.00, February 2007
https://www.sdcard.org/developers/overview/host_controller/simple_spec

[11] SD Specifications Part 1, Physical Layer Specification, Version 2.00, May 2006
https://www.sdcard.org/downloads/pls

[12] SD Specifications Part E1, SDIO Specification, Version 2.00, January 2007
https://www.sdcard.org/developers/overview/sdio/sdio_spec

[13] SD Memory Card Security Specification, Version 1.01

[14] The MultiMediaCard System Specification, Version 3.31, 4.2 and 4.4

13.1 Overview

The **S**ecure **D**igital/ **M**ulti**M**edia**C**ard interface (SDMMC) of the XMC4500 provides an interface between SD/SDIO/MMC cards and the AHB bus. The CPU is programmed to support SD, SDIO, SDHC and MMC cards, and can operate up to 48 MHz. The SDMMC module is able to transfer a maximum of 24 MB/sec for SD cards and 48 MB/sec for MMC cards.

13.1.1 Features

The SDMMC Host Controller handles SDIO/SD protocol at transmission level, packing data, adding cyclic redundancy check (CRC), start/end bit, and checking for transaction format correctness. Some useful applications of the SDMMC includes memory extension, data logging, and firmware update.

The SDMMC is compliant with the following specifications:

- SD Card Host Controller Version 2.0
- SD Physical Layer Specification Version 2.0
- SDIO Card Specification Version 2.0
- SD Memory Card Security Specification Version 1.01
- MMC Specification version 3.31, 4.2 and 4.4
- Fully compatible with earlier versions of MMC

The following functionalities are supported by the SDMMC module:

- **System Interface**
  - Data transfer using Programmed IO mode on AHB Slave interface
- **SD/SDIO/MMC Card Interface**
  - Transfers data in 1 bit and 4 bit SD modes and SPI mode
  - Cyclic Redundancy Check CRC7 for command and CRC16 for data integrity
  - Variable-length data transfers for SD/SDIO cards
  - Designed to work with SD I/O cards, Read-only cards and Read/Write cards
  - Supports Read wait Control, and Suspend/Resume operation for SD/SDIO cards
  - Supports MMC Plus and MMC Mobile
  - MMC Card detection for insertion/removal
  - Error Correction Codes (ECC) for eMMC 4.4 cards
  - Password protection for MMC cards
  - Two 512 byte buffer for data transfers between core and cards
  - Handles FIFO overrun and underrun conditions

Table 13-1 SDMMC Applications

Use Case SDMMC	Application
Memory Extension	All
Data Logging	All
Firmware Update	All
Data Transfer (PC and Application)	All

13.1.2 Block Diagram

The SDMMC block diagram is shown in **Figure 13-1**.

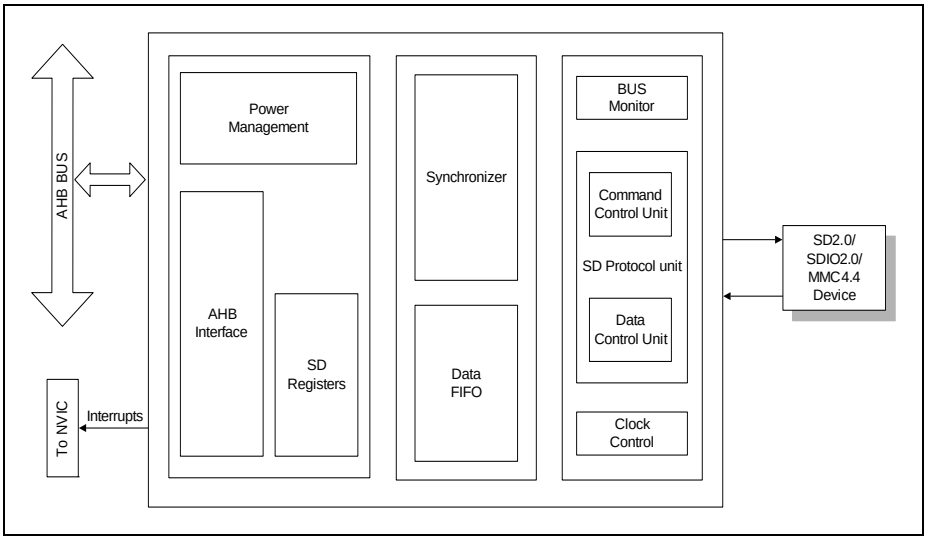


Figure 13-1 SDMMC Block Diagram

13.2 Functional Description

This section describes the functional blocks of the SDMMC.

AHB Interface

Host AHB interface acts as a bridge between AHB and the host controller. The SDMMC host controller provides Programmed IO method in which the ARM Host Driver transfers data using the Buffer Data Port Register **SDMMC_DATA_BUFFER**. The AHB target is having the Host control register **SDMMC_HOST_CTRL** and these registers are programmed by the CPU through the AHB target interface. The data transaction is performed through the AHB target interface in case of Programmed IO method of data transfer.

Interrupt controller

The SDMMC host controller generates interrupt to the Nested Vectored Interrupt Controller (NVIC) if any of the interrupt bits are set in the interrupt status register **SDMMC_INT_STATUS_NORM**.

DATA FIFO

The SDMMC host controller uses two 512 bytes dual port fifo for performing both read and write transactions. During a write transaction (data transferred from CPU to SD/SDIO/MMC card), the data will be filled in to the first and second fifo alternatively. When data from first fifo is transferring to the SD/SDIO/MMC card, the second fifo will be filled and vice versa. The two fifo's are alternatively used to store data which will give maximum throughput. During a read transaction (data transferred from SD/SDIO/MMC card to CPU), the data from SD/SDIO/MMC card will be written in to the two fifo's alternatively. When data from one fifo is transferring to the CPU, the second fifo will be filled and vice versa and thereby the throughput will be maximum. If the host controller cannot accept any data from SD/SDIO/MMC card, it will issue read wait (if card supports read wait mechanism) to stop the data coming from card or through stopping clock.

DAT[0-7] Control Logic

The DAT[0-7] control logic block transmits data in the data lines during write transaction and receives data in the data lines during read transaction.

BUS Monitor

Bus monitor will check for any violations occurring in the SD bus and time-out conditions.

Command Control Logic

The Command control logic block sends the command in the cmd line and receives the response coming from the SD/SDIO/MMC card.

Power Control

The SDMMC host controller controls the SD Bus Power depending on the value programmed in the Power Control Register **SDMMC_POWER_CTRL** by the CPU. The system has the responsibility to supply SD Bus Voltage according to card OCR and supply voltage capabilities depending on the host controller. If SD Bus power **SDMMC_POWER_CTRL**.SD_BUS_POWER = 1, the system shall supply voltage to the Card. If an unsupported voltage is selected in the SD Bus Voltage Select **SDMMC_POWER_CTRL**.SD_BUS_VOLTAGE_SEL field, the system may ignore write to SD Bus Power and keep its value at 0.

Clock Control

The Clock generation block will generate the SD clock depending on the value programmed by the CPU in the Clock Control Register **SDMMC_CLOCK_CTRL**.

Stream write and read transaction

SDMMC host controller will switch to second fifo after writing/reading a block of data to the first fifo, but in stream transaction, block size will not be programmed by the driver.

For both stream write and read transaction, it is recommended to write the maximum fifo size value to Block Size Register **SDMMC_BLOCK_SIZE**. For example if fifo size is 512 bytes, host driver needs to write 512 bytes to the **SDMMC_BLOCK_SIZE**. Fifo switching will occur after writing/ reading 512 bytes of data (fifo size).

13.3 Card Detection

When card insertion or removal in the slot is detected, the status will be sent to the CPU via interrupt methodology. The active low card signal SDCD_n is set to 0 during card detection. The **SDMMC_PRESENT_STATE**.CARD_INSERTED bit indicates whether a card has been inserted. A change from 0 to 1 generates a Card Insertion interrupt in the Normal Interrupt Status register **SDMMC_INT_STATUS_NORM**.CARD_INS = 1, and a change from 1 to 0 generates a Card Removal Interrupt in the Normal Interrupt Status register **SDMMC_INT_STATUS_NORM**.CARD_REMOVAL = 1.

Note: LED light indicates that card is being accessed. Do not remove card when LED light is ON.

13.4 Data Transfer Modes

SDMMC transfers are classified into following three modes, according to how the number of blocks is specified:

Single Block Transfer

Single block transfer mode can be selected by setting **SDMMC_TRANSFER_MODE**.MULTI_BLOCK_SELECT = 0. The number of blocks is specified to the host controller before the transfer via Block Count Register, and it is always set to 1. **SDMMC_BLOCK_COUNT**.BLOCK_COUNT = 0001_H.

Multiple Block Transfer

Multiple block transfer mode can be selected by setting **SDMMC_TRANSFER_MODE**.MULTI_BLOCK_SELECT = 1. The number of blocks is specified to the host controller before the transfer via Block Count Register **SDMMC_BLOCK_COUNT**.BLOCK_COUNT, and it can be set to 1 or more.

Infinite Block Transfer

The number of blocks is not specified to the host controller before the transfer. This transfer is continued until an abort transaction is executed. Refer to **Section 13.5.3** for details on Abort Transaction.

13.5 Read/ Write Operation

The controller will be configued to work with buffer data port registers **SDMMC_DATA_BUFFER** without internal DMA. The CPU will act as a master and start writing / reading data via **SDMMC_DATA_BUFFER**.

13.5.1 Write Operation

On receiving the Buffer Write Ready interrupt the CPU will act as a master and start transferring the data via Buffer data port register **SDMMC_DATA_BUFFER** (fifo_1). Transmitter starts sending the data in SD bus when a block of data is ready in fifo_1. While transmitting the data in sd bus, the buffer write ready interrupt is sent to the interrupt controller for the second block of data. The CPU will act as a master and start sending the second block of data via **SDMMC_DATA_BUFFER** to fifo_2. Buffer write ready interrupt will be asserted only when a fifo is empty to receive a block of data.

During write transaction the host controller will transmit data to card only when a block of data is ready to transmit and also the card is not driving busy. So underrun condition will not occur in SD side. The controller will assert buffer write ready interrupt **SDMMC_INT_STATUS_NORM**.BUFF_WRITE_READY = 1 only if space is available to accept a block of data.

13.5.2 Read Operation

Buffer Read Ready interrupt is asserted whenever a block of data is ready in one of the fifo's. On receiving the Buffer Read Ready interrupt, the CPU will act as a master and start reading the data via Buffer data port register **SDMMC_DATA_BUFFER** (fifo_1). Receiver start reading the data from SD bus only when a fifo is empty to receive a block of data. When both the fifo's are full the host controller will stop the data coming from the card through read wait mechanism (if card supports read wait) or through clock stopping.

During read transaction when fifo is full, that is, no space to accept one block of data from card, the host controller will stop the clock to card so overrun condition will not occur in SD side. The controller will assert buffer read ready interrupt **SDMMC_INT_STATUS_NORM**.BUFF_READ_READY = 1 only on reception of block of data from card.

13.5.3 Abort Transaction

There are two cases where the host driver needs to perform an Abort Transaction:

1. When the host driver stops infinite block transfers.
2. When host driver stops transfers while a multiple block transfer is exacting.

There are two ways to issue an Abort command; Asynchronous Abort and Synchronous Abort.

Asynchronous Abort

In an Asynchronous Abort sequence, the host driver can issue an Abort Command at anytime unless Command Inhibit (CMD) in the Present State Register is set to 1. **SDMMC_PRESENT_STATE**.COMMAND_INHIBIT_CMD = 1.

Synchronous Abort

In a Synchronous Abort, the host driver shall issue an Abort command after the data transfer stopped by using Stop At Block Gap Request in the Block Gap Control register. **SDMMC_BLOCK_GAP_CTRL**.STOP_AT_BLOCK_GAP = 1.

13.6 Special Command Types

There are three types of special commands. Suspend, Resume and Abort. These bits shall be set to 00_B for all other commands.

Suspend Command

Suspend command can be selected by setting **SDMMC_COMMAND**.CMD_TYPE = 01_B.

If the Suspend command succeeds, the host controller shall assume the SD Bus has been released and that it is possible to issue the next command which uses the DAT line. The controller shall de-assert Read Wait for read transactions and stop checking busy for write transactions. The Interrupt cycle shall start, in 4-bit mode. If the Suspend command fails, the controller shall maintain its current state, and the host driver shall restart the transfer by setting Continue Request in the Block Gap Control Register **SDMMC_BLOCK_GAP_CTRL**.CONTINUE_REQ = 1 to restart the transfer.

*Note: Suspend / Resume cannot be supported if Read Wait Control is disabled. Set **SDMMC_BLOCK_GAP_CTRL**.READ_WAIT_CTRL = 1 to enable Read Wait Control if the SD/SDIO card supports read wait function.*

Resume Command

Resume command can be selected by setting **SDMMC_COMMAND**.CMD_TYPE = 10_B.

The host driver re-starts the data transfer by restoring the registers in the range of 000_H - $00D_H$. The host controller shall check for busy before starting write transfers.

*Note: Suspend / Resume cannot be supported if Read Wait Control is disabled. Set **SDMMC_BLOCK_GAP_CTRL**.READ_WAIT_CTRL = 1 to enable Read Wait Control if the SD/SDIO card supports the read wait function.*

Abort Command

Abort command can be selected by setting **SDMMC_COMMAND**.CMD_TYPE = 11_B.

If this command is set when executing a read transfer, the host controller shall stop reads to the buffer. If this command is set when executing a write transfer, the host controller shall stop driving the DAT line. After issuing the Abort command, the controller should issue a software reset

13.7 Error Detection

This section describes data errors or defects detection methods.

Cyclic Redundancy Check (CRC)

The CRC7 and CRC16 generators calculate the CRC for Command and Data respectively to send the CRC to the SD/SDIO/MMC card. The CRC7 and CRC16 checker checks for any CRC error in the Response and Data sent by the SD/SDIO/MMC card. When a CRC error is generated, an interrupt will be triggered if the Error Interrupt Signal Enable is enabled. **SDMMC_EN_INT_SIGNAL_ERR**.DATA_CRC_ERR_EN = 1 for data CRC error, and **SDMMC_EN_INT_STATUS_ERR**.CMD_CRC_ERR_EN = 1 for command CRC error.

Error Correction Code (ECC)

Error correction codes (ECC) may be included in the payload data to detect data defects on the cards. An ECC code is used to store data on the MMC card. This ECC code is used by the SDMMC or application to decode the user data.

13.8 Service Request Generation

The SDMMC module provides one service request output. The service request output MMCI.SR0 is connected to interrupt node in the Nested Vectored Interrupt Controller (NVIC). For details on the service request and interrupt node, please refer to “Service Request Processing” and “Nested Vectored Interrupt Controller” chapters in the reference manual.

13.9 Debug Behavior

Suspend mode can be activated by issuing the suspend command. For details on the suspend command, please refer to **Section 13.6**.

13.10 Power, Reset and Clocks

This section describes the behaviour of power, reset and clocks.

Power

The SD/MMC card power supply can be controlled by the signal bus_pow. The SD bus voltage supported by the SDMMC is 3.3V. If the SD Bus power is set to 1 in the Power Control Register **SDMMC_POWER_CTRL**.SD_BUS_POWER = 1, the system shall supply voltage to the card.

Reset

The SDMMC host controller is reset asynchronously when one of the following occurs:

- A hardware reset to the card triggered by the $\overline{\text{MMC.RST}}$ pin.
- A software reset occurs. A reset pulse is generated when writing 1 to each bit of the Software Reset Register **SDMMC_SW_RESET**.

Clocks

The clocks connected to SDMMC include:

- clk_xin
  - Input clock to the SDMMC controller.
  - This is used to generate clk_sdcard_out and clk_sleep_out.
  - Frequency of clock is 48 MHz, generated from the System Control Unit (SCU) module.
- clk_sdcard_out
  - Clock supplied to the SD/MMC card.
- clk_sdcard_in
  - Feedback clock of clk_sdcard_out from the pad.
  - Feedback clock is used to reduce the pad delay in the clock line.

13.11 Initialisation and System Dependencies

This section provides information on how to initialise and use the SDMMC.

13.11.1 Setup SDMMC Data Transfer

Figure 13-2 shows the flowchart of SDMMC read/ write data transfer.

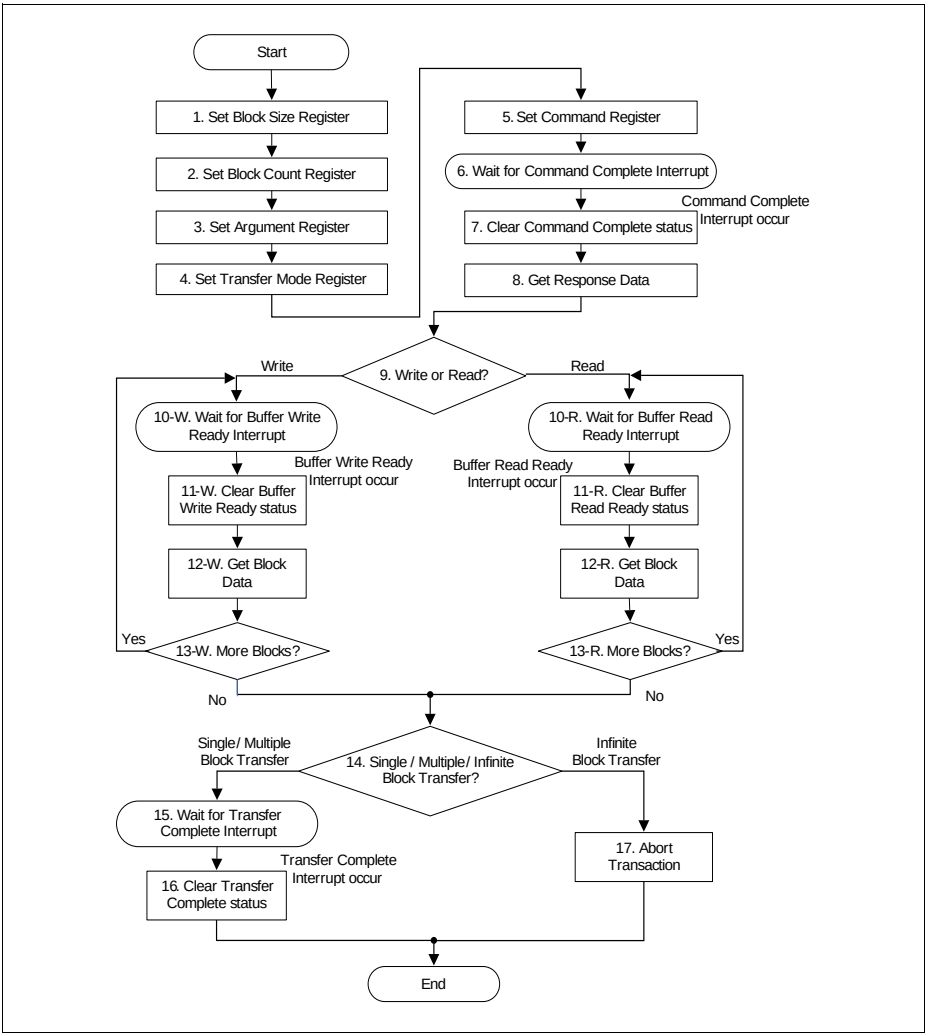


Figure 13-2 Data Transfer sequence

The following describes how to setup read/ write data transfer:

1. Set Block Size Register. Set executed data byte length of one block.

SDMMC_BLOCK_SIZE.TX_BLOCK_SIZE

2. Set Block Count Register. Set executed data block count.

SDMMC_BLOCK_COUNT.BLOCK_COUNT

3. Set Argument Register. Set value correspoinding to issued command.

SDMMC_ARGUMENT1.ARGUMENT1

4. Set Transfer Mode Register. Set Multi / single block, block count enable, data transfer direction, Auto CMD12 enable.

SDMMC_TRANSFER_MODE.MULTI_BLOCK_SELECT.

SDMMC_TRANSFER_MODE.BLOCK_COUNT_EN

SDMMC_TRANSFER_MODE.TX_DIR_SELECT

SDMMC_TRANSFER_MODE.ACMD_EN

5. Set Command Register. Set value corresponding to the issued command.

Note: When writing the upper byte of Command register, SD command is issued.

SDMMC_COMMAND

6. Wait for Command Complete Interrupt.

SDMMC_INT_STATUS_NORM.CMD_COMPLETE

7. Clear Command Complete status

Write **SDMMC_INT_STATUS_NORM**.CMD_COMPLETE = 1 to clear bit

8. Read Response Register and get the necessary information in accordance with the issued command.

SDMMC_RESPONSE

9. For Read Operation (read from card), go to step (10-R). See **Section 13.11.2**.

For Write Operation (write to card), go to step (10-W). See **Section 13.11.3**.

13.11.2 Read Operation

The following shows the configurations for SDMMC read operation:

10-R. Wait for Buffer Read Ready Interrupt

SDMMC_INT_STATUS_NORM.BUFF_READ_READY

11-R. Clear Buffer Read Ready status

Write **SDMMC_INT_STATUS_NORM**.BUFF_READ_READY = 1 to clear bit

12-R. Read Block Data (in accordance with the number of bytes specified in step (1))

SDMMC_DATA_BUFFER

13-R. Repeat until all blocks are received and then go to step (14)

14-R. For Single or Multiple Block Transfer, go to step (15). For Infinite Block Transfer, go to step (17)

15-R. Wait for Transfer Complete Interrupt

SDMMC_INT_STATUS_NORM.TX_COMPLETE

16-R. Get Transfer Complete status and end data transfer

Write **SDMMC_INT_STATUS_NORM**.TX_COMPLETE = 1 to clear bit

17-R. Perform Abort Transaction. See **Section 13.11.4**.

13.11.3 Write Operation

The following shows the configurations for SDMMC write operation:

10-W. Wait for Buffer Write Ready Interrupt

SDMMC_INT_STATUS_NORM.BUFF_WRITE_READY

11-W. Clear Buffer Write Ready status

Write **SDMMC_INT_STATUS_NORM**.BUFF_WRITE_READY = 1 to clear bit

12-W. Write Block Data (in accordance with the number of bytes specified in step (1))

SDMMC_DATA_BUFFER

13-W. Repeat until all blocks are received and then go to step (14)

14-W. For Single or Multiple Block Transfer, go to step (15). For Infinite Block Transfer, go to step (17)

15-W. Wait for Transfer Complete Interrupt

SDMMC_INT_STATUS_NORM.TX_COMPLETE

16-W. Get Transfer Complete status and end data transfer

Write **SDMMC_INT_STATUS_NORM**.TX_COMPLETE = 1 to clear bit

17-W. Perform Abort Transaction. See **Section 13.11.4**.

13.11.4 Abort Transaction

This section describes the sequence for the abort transaction.

Asynchronous Abort

The following shows the asynchronous abort sequence:

1. Check **SDMMC_PRESENT_STATE**.COMMAND_INHIBIT_CMD is not set to 1.
2. Issue Abort Command. **SDMMC_COMMAND**.CMD_TYPE = 11_B.

Synchronous Abort

The following shows the synchronous abort sequence:

1. Set **SDMMC_BLOCK_GAP_CTRL**.STOP_AT_BLOCK_GAP = 1 to stop SD transactions.
2. Wait for Transfer Complete Interrupt. **SDMMC_INT_STATUS_NORM**.TX_COMPLETE.
3. Set **SDMMC_INT_STATUS_NORM**.TX_COMPLETE = 1 to clear this bit.
4. Issue Abort Command. **SDMMC_COMMAND**.CMD_TYPE = 11_B.
5. Set **SDMMC_SW_RESET**.SW_RST_DAT_LINE = 1 and **SDMMC_SW_RESET**.SW_RST_CMD_LINE = 1 to do software reset.
6. Check SW_RST_DAT_LINE and SW_RST_CMD_LINE. If both are 0, end data transfer. If either SW_RST_DAT_LINE or SW_RST_CMD_LINE is 1, repeat step (6).

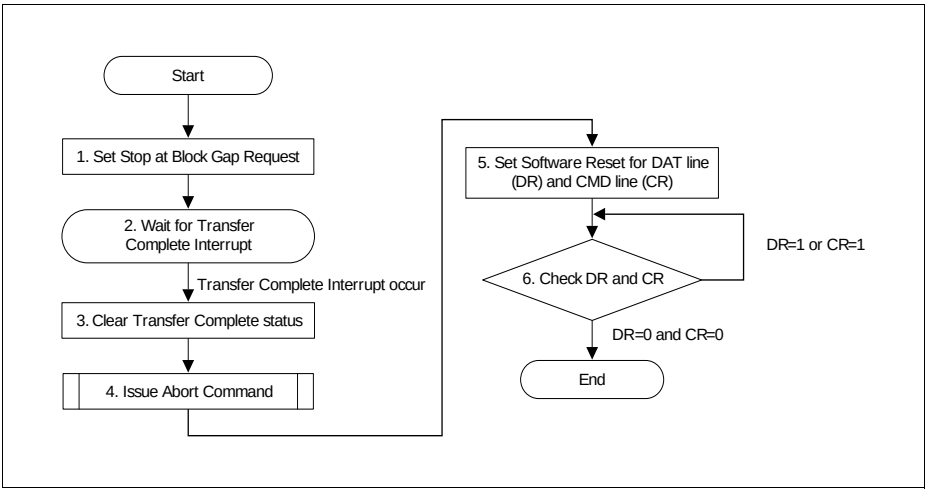


Figure 13-3 Synchronous Abort sequence

13.12 Registers

Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 13-2 Registers Address Space

Module	Base Address	End Address	Note
SDMMC	4801 C000$_H$	4801 FFFF$_H$	

Table 13-3 Register Overview

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
Reserved	Reserved	0000$_H$-0002$_H$	nBE	nBE	
Block Registers					
SDMMC_BLOCK_SIZE	Block Size Register	0004$_H$	U, PV	U, PV	Page 13-20
BLOCK_COUNT	Block Count Register	0006$_H$	U, PV	U, PV	Page 13-21
Argument1 Register					
SDMMC_ARGUMENT1	Argument1 Register	0008$_H$	U, PV	U, PV	Page 13-22
Transfer Mode Register					
SDMMC_TRANSFER_MODE	Transfer Mode Register	000C$_H$	U, PV	U, PV	Page 13-23
Command Register					
SDMMC_COMMAND	Command Register	000E$_H$	U, PV	U, PV	Page 13-26
Response Register					
SDMMC_RESPONSE0	Response 0 Register	0010$_H$	U, PV	U, PV	Page 13-28
SDMMC_RESPONSE2	Response 2 Register	0014$_H$	U, PV	U, PV	Page 13-28

Table 13-3 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
SDMMC_RESPONSE4	Response 4 Register	0018_H	U, PV	U, PV	Page 13-28
SDMMC_RESPONSE6	Response 6 Register	$001C_H$	U, PV	U, PV	Page 13-28
Buffer Data Port Register					
SDMMC_DATA_BUFFER	Data Buffer Register	0020_H	U, PV	U, PV	Page 13-32
Present State Register					
SDMMC_PRESENT_STATE	Present State Register	0024_H	U, PV	U, PV	Page 13-33
Control Registers					
SDMMC_HOST_CTRL	Host Control Register	0028_H	U, PV	U, PV	Page 13-41
SDMMC_POWER_CTRL	Power Control Register	0029_H	U, PV	U, PV	Page 13-43
SDMMC_BLOCK_GAP_CTRL	Block Gap Control Register	$002A_H$	U, PV	U, PV	Page 13-44
SDMMC_WAKEUP_CTRL	Wake-up Control Register	$002B_H$	U, PV	U, PV	Page 13-47
SDMMC_CLOCK_CTRL	Clock Control Register	$002C_H$	U, PV	U, PV	Page 13-48
SDMMC_TIMEOUT_CTRL	Timeout Control Register	$002E_H$	U, PV	U, PV	Page 13-51
SDMMC_SW_RESET	Software Reset Register	$002F_H$	U, PV	U, PV	Page 13-52
Interrupt Status Registers					
SDMMC_INT_STATUS_NORM	Normal Interrupt Status Register	0030_H	U, PV	U, PV	Page 13-54
SDMMC_INT_STATUS_ERR	Error Interrupt Status Register	0032_H	U, PV	U, PV	Page 13-60
SDMMC_EN_INT_STATUS_NORM	Normal Interrupt Status Enable Register	0034_H	U, PV	U, PV	Page 13-64

Table 13-3 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
SDMMC_EN_INT_STATUS_ERR	Error Interrupt Status Enable Register	0036_H	U, PV	U, PV	Page 13-66
SDMMC_EN_INT_SIGNAL_NORM	Normal Interrupt Signal Enable Register	0038_H	U, PV	U, PV	Page 13-68
SDMMC_EN_INT_SIGNAL_ERR	Error Interrupt Signal Enable Register	$003A_H$	U, PV	U, PV	Page 13-70
SDMMC_ACMD_ERR_STATUS	Auto CMD12 Error Status Register	$003C_H$	U, PV	U, PV	Page 13-72
Reserved	Reserved	$003E_H$ - $004E_H$	nBE	nBE	
Error Status Registers					
SDMMC_FORCE_EVENT_ACMD_ERR_STATUS	Force Event Register for Auto CMD Error Status	050_H	U, PV	U, PV	Page 13-75
SDMMC_FORCE_EVENT_ERR_STATUS	Force Event Register for Error Interrupt Status	052_H	U, PV	U, PV	Page 13-77
Reserved	Reserved	0054_H - 0072_H	nBE	nBE	
Debug Selection Register					
SDMMC_DEBUG_SEL	Debug Selection Register	0074_H	U, PV	U, PV	Page 13-79
Reserved	Reserved	0075_H - $00EE_H$	nBE	nBE	
SPI Interrupt Support Register					
SDMMC_SPI	SPI Interrupt Support Register	$00F0_H$			Page 13-80
Reserved	Reserved	$00F2_H$ - $00FA_H$	nBE	nBE	

Table 13-3 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
Slot Interrupt Status Register					
SDMMC_SLOT_INT_STATUS	Slot Interrupt Status Register	$00FC_H$	U, PV	U, PV	Page 13-81
Reserved	Reserved	$00FE_H$	nBE	nBE	

Access Restrictions

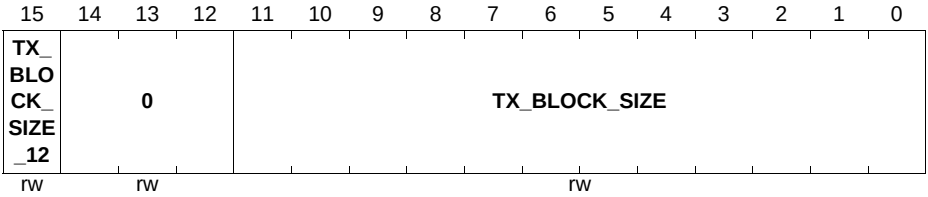
Note: The SDMMC registers are accessible only through word accesses. Half-word and byte accesses on SDMMC registers will not generate a bus error. Writes to unused address space will not cause an error but will be ignored.

13.12.1 Registers Description

SDMMC_BLOCK_SIZE

This register is used to configure the block size for data transfer.

SDMMC_BLOCK_SIZE
Block Size Register **(0004_H)** **Reset Value: 0000_H**

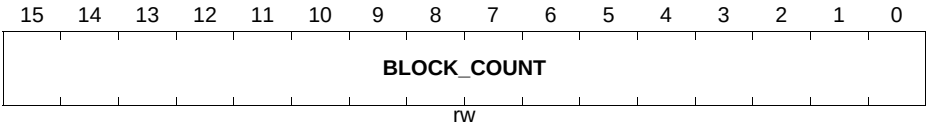


Field	Bits	Type	Description
TX_BLOCK_SIZE_12	15	rw	**Transfer Block Size 12th bit.** This bit is added to support 4Kb Data block transfer. This bit can be set and cleared only by software.
0	[14:12]	rw	**Reserved** Read as 0; must be written with 0.
TX_BLOCK_SIZE	[11:0]	rw	**Transfer Block Size** This register specifies the block size for block data transfers for CMD17, CMD18, CMD24, CMD25, and CMD53. It can be accessed only if no transaction is executing (i.e after a transaction has stopped). Read operations during transfer return an invalid value and write operations shall be ignored. 0000_H No Data Transfer 0001_H 1 Byte 0002_H 2 Bytes 0003_H 3 Bytes 0004_H 4 Bytes ... $01FF_H$ 511 Bytes 0200_H 512 Bytes (Maximum Block Size) *Note: Other values are reserved* This bit can be set and cleared only by software.

SDMMC_BLOCK_COUNT

This register is used to configure the block count for current transfer.

SDMMC_BLOCK_COUNT
Block Count Register **(0006_H)** **Reset Value: 0000_H**

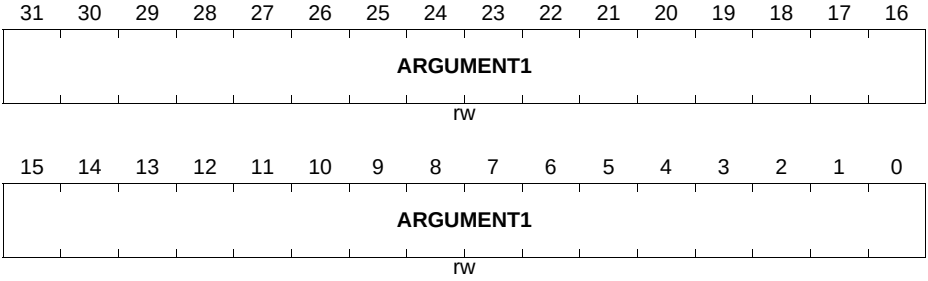


Field	Bits	Type	Description
BLOCK_COUNT	[15:0]	rw	**Blocks Count for Current Transfer** This register is enabled when Block Count Enable in the Transfer Mode register is set to 1 and is valid only for multiple block transfers. The host controller decrements the block count after each block transfer and stops when the count reaches zero. It can be accessed only if no transaction is executing (i.e after a transaction has stopped). Read operations during transfer return an invalid value and write operations shall be ignored. When saving transfer context as a result of Suspend command, the number of blocks yet to be transferred can be determined by reading this register. When restoring transfer context prior to issuing a Resume command, the host driver shall restore the previously save block count. 0000_H Stop Count 0001_H 1 block 0002_H 2 blocks ... $FFFF_H$ 65535 blocks This bit can be set and cleared only by software.

SDMMC_ARGUMENT1

This register is used to configure the SD command argument.

SDMMC_ARGUMENT1
Argument1 Register **(0008_H)** **Reset Value: 00000000_H**

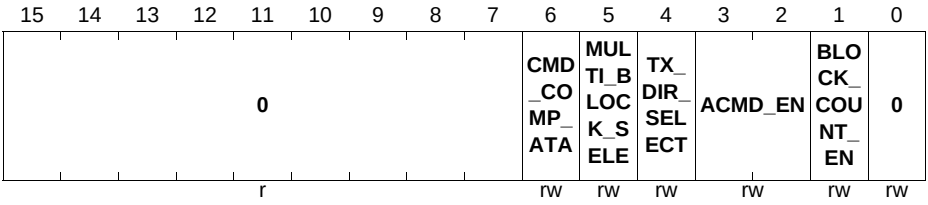


Field	Bits	Type	Description
ARGUMENT1	[31:0]	rw	**Command Argument** The SD Command Argument is specified as bit 39-8 of Command-Format. This bit can be set and cleared only by software.

SDMMC_TRANSFER_MODE

This register is used to configure the data transfer mode.

SDMMC_TRANSFER_MODE
Transfer Mode Register **($000C_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
0	[15:7]	r	**Reserved** Read as 0; should be written with 0.
CMD_COMP_ATA	6	rw	**Command Completion Signal Enable for CE-ATA Device** 1_B Device will send command completion Signal 0_B Device will not send command completion Signal This bit can be set and cleared only by software.
MULTI_BLOCK_SELECT	5	rw	**Multi / Single Block Select** This bit enables multiple block DAT line data transfers. 0_B Single Block 1_B Multiple Block This bit can be set and cleared only by software.
TX_DIR_SELECT	4	rw	**Data Transfer Direction Select** This bit defines the direction of DAT line data transfers. 0_B Write (Host to Card) 1_B Read (Card to Host) This bit can be set and cleared only by software.

Field	Bits	Type	Description
ACMD_EN	[3:2]	rw	**Auto CMD Enable** This field determines use of auto command functions 00_B Auto Command Disabled 01_B Auto CMD12 Enable *Note: Other values are reserved* To stop Multiple-block read and write operation: • Auto CMD12 Enable Multiple-block read and write commands for memory require CMD12 to stop the operation. When this field is set to 01_B, the host controller issues CMD12 automatically when last block transfer is completed. Auto CMD12 error is indicated to the Auto CMD Error Status register. The Host Driver shall not set this bit if the command does not require CMD12. This bit can be set and cleared only by software.
BLOCK_COUNT_EN	1	rw	**Block Count Enable** This bit is used to enable the Block count register, which is only relevant for multiple block transfers. When this bit is 0, the Block Count register is disabled, which is useful in executing an infinite transfer. 0_B Disable 1_B Enable This bit can be set and cleared only by software.
0	0	rw	**Reserved** Read as 0; must be written with 0.

Determination of transfer type

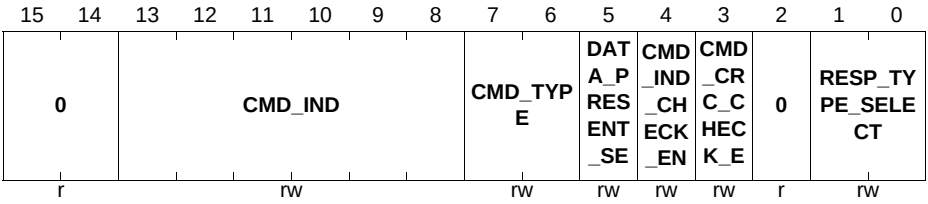
Table 13-4 Determination of transfer type

Multi / Single Block Select	Block Count Enable	Block Count	Function
0	Don’t Care	Don’t Care	Single Transfer
1	0	Don’t Care	Infinite Transfer
1	1	Not Zero	Multiple Transfer
1	1	Zero	Stop Multiple Transfer

SDMMC_COMMAND

This register is used to configure the SDMMC command.

SDMMC_COMMAND
Command Register **($000E_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
0	[15:14]	r	**Reserved** Read as 0; should be written with 0.
CMD_IND	[13:8]	rw	**Command Index** This bit shall be set to the command number (CMD0-63, ACMD0-63). This bit can be set and cleared only by software.
CMD_TYPE	[7:6]	rw	**Command Type** There are three types of special commands. Suspend, Resume and Abort. These bits shall bet set to 00b for all other commands. 00_B Normal 01_B Suspend 10_B Resume 11_B Abort This bit can be set and cleared only by software.
DATA_PRESENT_SELECT	5	rw	**Data Present Select** This bit is set to 1 to indicate that data is present and shall be transferred using the DAT line. If is set to 0 for the following: 1. Commands using only CMD line (ex. CMD52) 2. Commands with no data transfer but using busy signal on DAT[0] line (R1b or R5b ex. CMD38) 3. Resume Command 0_B No Data Present 1_B Data Present This bit can be set and cleared only by software.

Field	Bits	Type	Description
CMD_IND_CHECK_EN	4	rw	**Command Index Check Enable** If this bit is set to 1, the host controller shall check the index field in the response to see if it has the same value as the command index. If it is not, it is reported as a Command Index Error. If this bit is set to 0, the Index field is not checked. 0_B Disable 1_B Enable This bit can be set and cleared only by software.
CMD_CRC_CHECK_EN	3	rw	**Command CRC Check Enable** If this bit is set to 1, the host controller shall check the CRC field in the response. If an error is detected, it is reported as a Command CRC Error. If this bit is set to 0, the CRC field is not checked. 0_B Disable 1_B Enable This bit can be set and cleared only by software.
0	2	r	**Reserved** Read as 0; should be written with 0.
RESP_TYPE_SELECT	[1:0]	rw	**Response Type Select** 00_B No Response 01_B Response length 136 10_B Response length 48 11_B Response length 48 check Busy after response This bit can be set and cleared only by software.

SDMMC_RESPONSE

This register is used to configure the command response. **Table 13-5** shows the relation between parameters and the name of response type.

Table 13-5 Relation between parameters and the name of response type

Response Type	Index Check Enable	CRC Check Enable	Name of Response Type
00	0	0	No Response
01	0	1	R2
10	0	0	R3, R4
10	1	1	R1, R6, R5, R7
11	1	1	R1b, R5b

Table 13-6 describes the mapping of command responses from the SD Bus to this register for each response type. In the table, R[] refers to a bit range within the response data as transmitted on the SD Bus, RESPONSE[] refers to a bit range within the Response register.

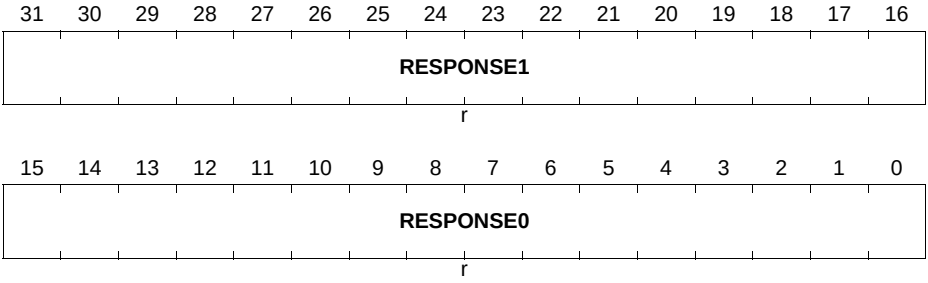
Table 13-6 Response bit definition for each response type

Kind of Response	Meaning of Response	Response Field	Response Register
R1, R1b (normal response)	Card Status	R[39:8]	RESPONSE 0[31:0]
R1b (Auto CMD12 response)	Card Status for Auto CMD12	R[39:8]	RESPONSE 6[31:0]
R2 (CID, CSD Register)	CID or CSD reg. incl.	R[127:8]	RESPONSE 6[23:0], RESPONSE 4[31:0] RESPONSE 2[31:0], RESPONSE 0[31:0]
R3 (OCR Register)	OCR Register for memory	R[39:8]	RESPONSE 0[31:0]
R4 (OCR Register)	OCR Register for I/O etc.	R[39:8]	RESPONSE 0[31:0]

Table 13-6 Response bit definition for each response type (cont'd)

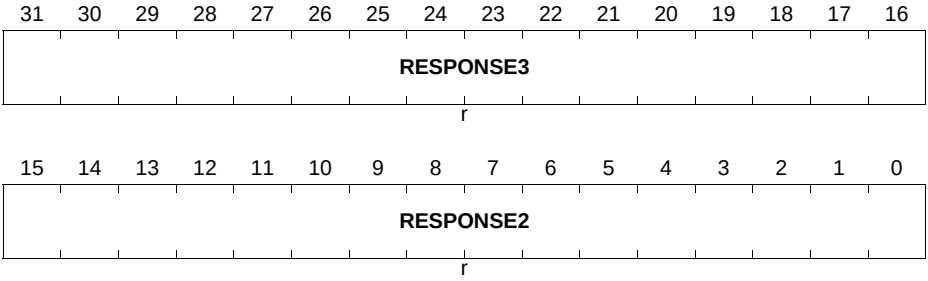
Kind of Response	Meaning of Response	Response Field	Response Register
R5, R5b	SDIO Response	R[39:8]	RESPONSE 0[31:0]
R6 (Published RCA response)	New published RCA[31:16] etc.	R[39:8]	RESPONSE 0[31:0]

SDMMC_RESPONSE0
Response 0 Register **(0010$_H$)** **Reset Value: 00000000$_H$**



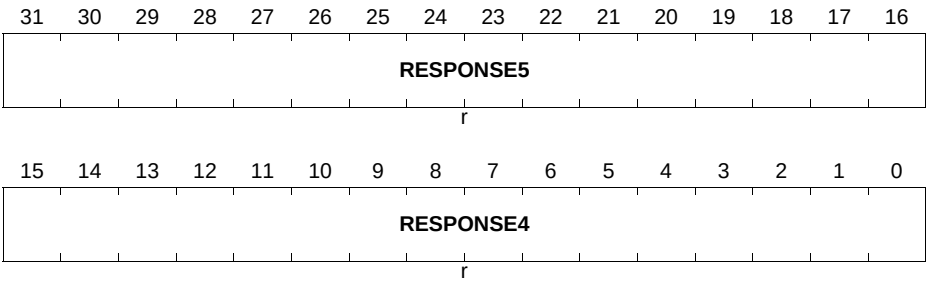
Field	Bits	Type	Description
RESPONSE1	[31:16]	r	**Response1** This bit is initialized to 0 at reset.
RESPONSE0	[15:0]	r	**Response0** This bit is initialized to 0 at reset.

SDMMC_RESPONSE2
Response 2 Register **(0014_H)** **Reset Value: 00000000_H**



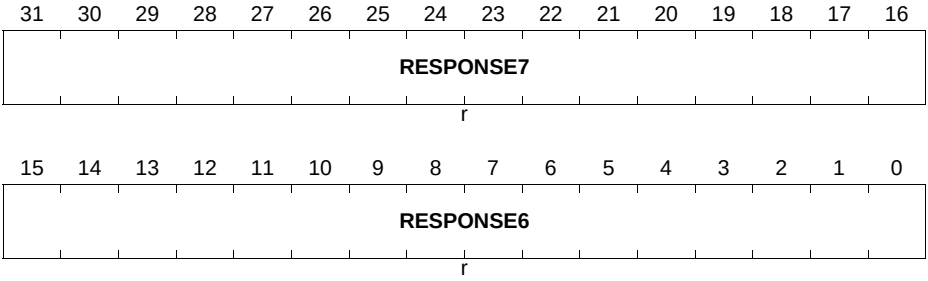
Field	Bits	Type	Description
RESPONSE3	[31:16]	r	**Response3** This bit is initialized to 0 at reset.
RESPONSE2	[15:0]	r	**Response2** This bit is initialized to 0 at reset.

SDMMC_RESPONSE4
Response 4 Register **(0018_H)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
RESPONSE5	[31:16]	r	**Response5** This bit is initialized to 0 at reset.
RESPONSE4	[15:0]	r	**Response4** This bit is initialized to 0 at reset.

SDMMC_RESPONSE6
Response 6 Register **($001C_H$)** **Reset Value: 00000000_H**

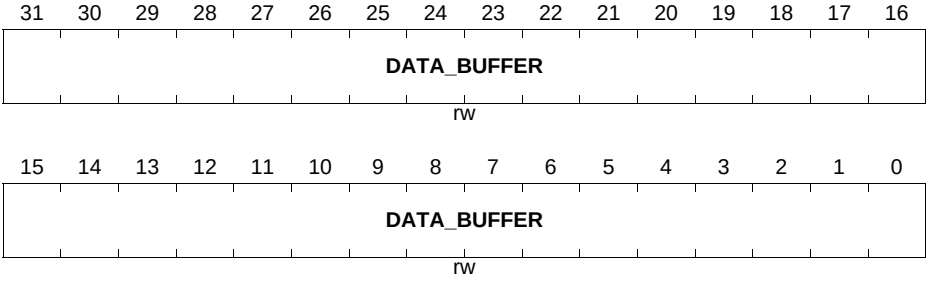


Field	Bits	Type	Description
RESPONSE7	[31:16]	r	**Response7** This bit is initialized to 0 at reset.
RESPONSE6	[15:0]	r	**Response6** This bit is initialized to 0 at reset.

SDMMC_DATA_BUFFER

This register is used to configure the SDMMC host controller data buffer.

SDMMC_DATA_BUFFER
Data Buffer Register **(0020_H)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
DATA_BUFFER	[31:0]	rw	**Data Buffer** The host controller buffer can be accessed through this 32-bit Data Port Register. Reset: X This bit can be set and cleared only by software.

SDMMC_PRESENT_STATE

This register is used to check the present state of the SDMMC host controller.

SDMMC_PRESENT_STATE
Present State Register **(0024_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0			DAT_7_4_PIN_LEVEL				CMD_LINE_LEVEL	DAT_3_0_PIN_LEVEL				WRITE_PROTECT_PI	CARD_DETECT_PIN_	CARD_STATE_STAB	CARD_INSERTED
r			r				r	r				r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				BUFFER_READ_ENA	BUFFER_WRITE_ENA	READ_TRANSFER_A	WRITE_TRANSFER_A	0					DAT_LINE_ACTIVE	COMMAND_INHIBIT_D	COMMAND_INHIBIT_C
r				r	r	r	r	r					r	r	r

Field	Bits	Type	Description
0	[31:29]	r	**Reserved** Read as 0; should be written with 0.
DAT_7_4_PIN_LEVEL	[28:25]	r	**Line Signal Level** This status is used to check DAT line level to recover from errors, and for debugging. D28 - DAT[7] D27 - DAT[6] D26 - DAT[5] D25 - DAT[4] Reset: F_H
CMD_LINE_LEVEL	24	r	**CMD Line Signal Level** This status is used to check CMD line level to recover from errors, and for debugging. Reset: 1_B

Field	Bits	Type	Description
DAT_3_0_PIN_LEVEL	[23:20]	r	**Line Signal Level** This status is used to check DAT line level to recover from errors, and for debugging. This is especially useful in detecting the busy signal level from DAT[0]. D23 - DAT[3] D22 - DAT[2] D21 - DAT[1] D20 - DAT[0] Reset: F_H
WRITE_PROTECT_PIN_LEVEL	19	r	**Write Protect Switch Pin Level** The Write Protect Switch is supported for memory and combo cards. This bit reflects the SDWP pin. 0_B Write protected (SDWP = 1) 1_B Write enabled (SDWP = 0)
CARD_DETECT_PIN_LEVEL	18	r	**Card Detect Pin Level** This bit reflects the inverse value of the SDCD pin. 0_B No Card present (SDCD = 1) 1_B Card present (SDCD = 0)
CARD_STATE_STABLE	17	r	**Card State Stable** This bit is used for testing. If it is 0, the Card Detect Pin Level is not stable. If this bit is set to 1, it means the Card Detect Pin Level is stable. The Software Reset For All in the Software Reset Register shall not affect this bit. 0_B Reset of Debouncing 1_B No Card or Inserted Reset: 1_B

Field	Bits	Type	Description
CARD_INSERTED	16	r	**Card Inserted** This bit indicates whether a card has been inserted. Changing from 0 to 1 generates a Card Insertion interrupt in the Normal Interrupt Status register and changing from 1 to 0 generates a Card Removal Interrupt in the Normal Interrupt Status register. The Software Reset For All in the Software Reset register shall not affect this bit. If a Card is removed while its power is on and its clock is oscillating, the host controller shall clear SD Bus Power in the Power Control register and SD Clock Enable in the Clock control register. In addition the host driver should clear the host controller by the Software Reset For All in Software register. The card detect is active regardless of the SD Bus Power. 0_B Reset or Debouncing or No Card 1_B Card Inserted
0	[15:12]	r	**Reserved** Read as 0; should be written with 0.
BUFFER_READ_ENABLE	11	r	**Buffer Read Enable** This status is used for non-DMA read transfers. This read only flag indicates that valid data exists in the host side buffer status. If this bit is 1, readable data exists in the buffer. A change of this bit from 1 to 0 occurs when all the block data is read from the buffer. A change of this bit from 0 to 1 occurs when all the block data is ready in the buffer and generates the Buffer Read Ready Interrupt. 0_B Read Disable 1_B Read Enable. This bit is initialized to 0 at reset.

Field	Bits	Type	Description
BUFFER_WRITE_ENABLE	10	r	**Buffer Write Enable** This status is used for non-DMA write transfers. This read only flag indicates if space is available for write data. If this bit is 1, data can be written to the buffer. A change of this bit from 1 to 0 occurs when all the block data is written to the buffer. A change of this bit from 0 to 1 occurs when top of block data can be written to the buffer and generates the Buffer Write Ready Interrupt. 0_B Write Disable 1_B Write Enable. This bit is initialized to 0 at reset.
READ_TRANSFER_ACTIVE	9	r	**Read Transfer Active** This status is used for detecting completion of a read transfer. This bit is set to 1 for either of the following conditions: After the end bit of the read command When writing a 1 to continue Request in the Block Gap Control register to restart a read transfer This bit is cleared to 0 for either of the following conditions: When the last data block as specified by block length is transferred to the system. When all valid data blocks have been transferred to the system and no current block transfers are being sent as a result of the Stop At Block Gap Request set to 1. A transfer complete interrupt is generated when this bit changes to 0. 0_B No valid data 1_B Transferring data This bit is initialized to 0 at reset.

Field	Bits	Type	Description
WRITE_TRANSFER_ACTIVE	8	r	**Write Transfer Active** This status indicates a write transfer is active. If this bit is 0, it means no valid write data exists in the host controller. This bit is set in either of the following cases: After the end bit of the write command. When writing a 1 to Continue Request in the Block Gap Control register to restart a write transfer. This bit is cleared in either of the following cases: After getting the CRC status of the last data block as specified by the transfer count (Single or Multiple) After getting a CRC status of any block where data transmission is about to be stopped by a Stop At Block Gap Request. During a write transaction, a Block Gap Event interrupt is generated when this bit is changed to 0, as a result of the Stop At Block Gap Request being set. This status is useful for the host driver in determining when to issue commands during write busy. 0_B No valid data 1_B Transferring data This bit is initialized to 0 at reset.
0	[7:3]	r	**Reserved** Read as 0; should be written with 0.
DAT_LINE_ACTIVE	2	r	**DAT Line Active** This bit indicates whether one of the DAT line on SD bus is in use [1)]. 0_B DAT line inactive 1_B DAT line active This bit is initialized to 0 at reset.

Field	Bits	Type	Description
COMMAND_INHIBIT_DAT	1	r	**Command Inhibit (DAT)** This status bit is generated if either the DAT Line Active or the Read transfer Active is set to 1. If this bit is 0, it indicates the host controller can issue the next SD command. Commands with busy signal belong to Command Inhibit (DAT) (ex. R1b, R5b type). Changing from 1 to 0 generates a Transfer Complete interrupt in the Normal interrupt status register. *Note: The SD Host Driver can save registers in the range of 000_H - $00D_H$ for a suspend transaction after this bit has changed from 1 to 0.* 0_B Can issue command which uses the DAT line 1_B Cannot issue command which uses the DAT line This bit is initialized to 0 at reset.
COMMAND_INHIBIT_CMD	0	r	**Command Inhibit (CMD)** If this bit is 0, it indicates the CMD line is not in use and the host controller can issue a SD command using the CMD line. This bit is set immediately after the Command register (00Fh) is written. This bit is cleared when the command response is received. Even if the Command Inhibit (DAT) is set to 1, Commands using only the CMD line can be issued if this bit is 0. Changing from 1 to 0 generates a Command complete interrupt in the Normal Interrupt Status register. If the host controller cannot issue the command because of a command conflict error or because of Command Not Issued By Auto CMD12 Error, this bit shall remain 1 and the Command Complete is not set. Status issuing Auto CMD12 is not read from this bit. Auto CMD12 consists of two responses. In this case, this bit is not cleared by the response of CMD12 but cleared by the response of a read/write command. Status issuing Auto CMD12 is not read from this bit. So if a command is issued during Auto CMD12 operation, host controller shall manage to issue two commands: CMD12 and a command set by Command register. This bit is initialized to 0 at reset.

1) DAT line active indicates whether one of the DAT line is on SD bus is in use.

(a) In the case of read transactions

This status indicates whether a read transfer is executing on the SD Bus. Changing this value from 1 to 0 generates a Block Gap Event interrupt in the Normal Interrupt Status register, as the result of the Stop At Block Gap Request being set.

This bit shall be set in either of the following cases:

1. After the end bit of the read command.
2. When writing a 1 to Continue Request in the Block Gap Control register to restart a read transfer.

This bit shall be cleared in either of the following cases:

1. When the end bit of the last data block is sent from the SD Bus to the host controller.
2. When a read transfer is stopped at the block gap initiated by a Stop At BlockGap Request.

The host controller shall stop read operation at the start of the interrupt cycle of the next block gap by driving Read Wait or stopping SD clock. If the Read Wait signal is already driven (due to data buffer cannot receive data), the host controller can continue to stop read operation by driving the Read Wait signal. It is necessary to support Read Wait in order to use suspend / resume function.

(b) In the case of write transactions

This status indicates that a write transfer is executing on the SD Bus. Changing this value from 1 to 0 generate a Transfer Complete interrupt in the Normal Interrupt Status register.

This bit shall be set in either of the following cases:

1. After the end bit of the write command.
2. When writing to 1 to Continue Request in the Block Gap Control register to continue a write transfer.

This bit shall be cleared in either of the following cases:

1. When the SD card releases write busy of the last data block. If SD card does not drive busy signal for 8 SD Clocks, the host controller shall consider the card drive "Not Busy".
2. When the SD card releases write busy prior to waiting for write transfer as a result of a Stop At Block Gap Request.

(c) Command with busy

This status indicates whether a command indicates busy (ex. erase command for memory) is executing on the SD Bus. This bit is set after the end bit of the command with busy and cleared when busy is de-asserted. Changing this bit from 1 to 0 generate a Transfer Complete interrupt in the Normal Interrupt Status register.

Note: The host driver can issue cmd0, cmd12, cmd13 (for memory) and cmd52 (for SDIO) when the DAT lines are busy during data transfer. These commands can be issued when Command Inhibit (CMD) is set to zero. Other commands shall be issued when Command Inhibit (DAT) is set to zero.

SDMMC_HOST_CTRL

This register is used to configure the modes of the SDMMC host controller.

SDMMC_HOST_CTRL
Host Control Register **(0028_H)** **Reset Value: 00_H**

7	6	5	4	3	2	1	0
CARD_DET_SIGNAL_DETECT	CARD_DETECT_TEST_LEVEL	0			HIGH_SPEED_EN	DATA_TX_WIDTH	LED_CTRL
rw	rw	rw			rw	rw	rw

Field	Bits	Type	Description
CARD_DET_SIGNAL_DETECT	7	rw	**Card detect signal detetction** This bit selects source for card detection. 0_B SDCD is selected (for normal use) 1_B The card detect test level is selected This bit can be set and cleared only by software.
CARD_DETECT_TEST_LEVEL	6	rw	**Card Detect Test Level** This bit is enabled while the Card Detect Signal Selection is set to 1 and it indicates card inserted or not. Generates (card ins or card removal) Interrupt when the normal int sts enable bit is set. 0_B No Card 1_B Card Inserted This bit can be set and cleared only by software.
0	[5:3]	rw	**Reserved** Read as 0; must be written with 0.
HIGH_SPEED_EN	2	rw	**High Speed Enable** This bit is optional. If this bit is set to 0 (default), the host controller outputs CMD line and DAT lines at the falling edge of the SD clock (up to 25 MHz / 20 MHz for MMC). If this bit is set to 1, the host controller outputs CMD line and DAT lines at the rising edge of the SD clock (up to 50 MHz for SD / 52 MHz for MMC) 0_B Normal Speed Mode 1_B High Speed Mode This bit can be set and cleared only by software.

Field	Bits	Type	Description
DATA_TX_ WIDTH	1	rw	**Data Transfer Width (SD1 or SD4)** This bit selects the data width of the host controller. The host driver shall select it to match the data width of the SD card. 0_B 1 bit mode 1_B 4-bit mode This bit can be set and cleared only by software.
LED_CTRL	0	rw	**LED Control** This bit is used to caution the user not to remove the card while the SD card is being accessed. If the software is going to issue multiple SD commands, this bit can be set during all transactions. It is not necessary to change for each transaction. 0_B LED off 1_B LED on This bit can be set and cleared only by software.

SDMMC_POWER_CTRL

This register is used to configure the SD bus power.

SDMMC_POWER_CTRL
Power Control Register **(0029_H)** **Reset Value: 00_H**

7	6	5	4	3	2	1	0
0			**HARDWARE_RESET**	**SD_BUS_VOLTAGE_SEL**			**SD_BUS_POWER**
r			rw	rw			rw

Field	Bits	Type	Description
0	[7:5]	r	**Reserved** Read as 0; should be written with 0.
HARDWARE_RESET	4	rw	**Hardware reset** Hardware reset signal is generated for eMMC4.4 card when this bit is set This bit can be set and cleared only by software.
SD_BUS_VOLTAGE_SEL	[3:1]	rw	**SD Bus Voltage Select** By setting these bits, the host driver selects the voltage level for the SD card. If an unsupported voltage is selected, the Host System shall not supply SD bus voltage 111_B 3.3V (Flattop.) *Note: Other values are reserved* This bit can be set and cleared only by software.
SD_BUS_POWER	0	rw	**SD Bus Power** Before setting this bit, the SD host driver shall set SD Bus Voltage Select. If the host controller detects the No Card State, this bit shall be cleared. 0_B Power off 1_B Power on This bit can be set and cleared only by software.

SDMMC_BLOCK_GAP_CTRL

This register is used to configure the block gap request.

SDMMC_BLOCK_GAP_CTRL
Block Gap Control Register (002A$_H$) Reset Value: 00$_H$

7	6	5	4	3	2	1	0
0	**0**		**SPI_MODE**	**INT_AT_BLOCK_GAP**	**READ_WAIT_CTRL**	**CONTINUE_REQ**	**STOP_AT_BLOCK_GAP**
r	rw		rw	rw	rw	rw	rw

Field	Bits	Type	Description
0	7	r	**Reserved** Read as 0; should be written with 0.
0	[6:5]	rw	**Reserved** Read as 0; must be written with 0.
SPI_MODE	4	rw	**SPI_MODE** SPI mode enable bit. 0_B SD mode 1_B SPI mode This bit can be set and cleared only by software.
INT_AT_BLOCK_GAP	3	rw	**Interrupt At Block Gap** This bit is valid only in 4-bit mode of the SDIO card and selects a sample point in the interrupt cycle. Setting to 1 enables interrupt detection at the block gap for a multiple block transfer. If the SD card cannot signal an interrupt during a multiple block transfer, this bit should be set to 0. When the host driver detects an SD card insertion, it shall set this bit according to the CCCR of the SDIO card. This bit can be set and cleared only by software.

Field	Bits	Type	Description
READ_WAIT_CTRL	2	rw	**Read Wait Control** The read wait function is optional for SDIO cards. If the card supports read wait, set this bit to enable use of the read wait protocol to stop read data using DAT[2] line. Otherwise the host controller has to stop the SD clock to hold read data, which restricts commands generation. When the host driver detects an SD card insertion, it shall set this bit according to the CCCR of the SDIO card. If the card does not support read wait, this bit shall never be set to 1 otherwise DAT line conflict may occur. If this bit is set to 0, Suspend / Resume cannot be supported 0_B Disable Read Wait Control 1_B Enable Read Wait Control This bit can be set and cleared only by software.
CONTINUE_REQ	1	rw	**Continue Request** This bit is used to restart a transaction which was stopped using the Stop At Block Gap Request. To cancel stop at the block gap, set Stop At block Gap Request to 0 and set this bit to restart the transfer. The host controller automatically clears this bit in either of the following cases: 1. In the case of a read transaction, the DAT Line Active changes from 0 to 1 as a read transaction restarts. 2. In the case of a write transaction, the Write transfer active changes from 0 to 1 as the write transaction restarts. Therefore it is not necessary for Host driver to set this bit to 0. If Stop At Block Gap Request is set to 1, any write to this bit is ignored. 0_B Ignored 1_B Restart

Field	Bits	Type	Description
STOP_AT_BLOCK_GAP	0	rw	**Stop At Block Gap Request** This bit is used to stop executing a transaction at the next block gap for non- DMA transfers. Until the transfer complete is set to 1, indicating a transfer completion the host driver shall leave this bit set to 1. Clearing both the Stop At Block Gap Request and Continue Request shall not cause the transaction to restart. Read Wait is used to stop the read transaction at the block gap. The host controller shall honour Stop At Block Gap Request for write transfers, but for read transfers it requires that the SD card support Read Wait. Therefore the host driver shall not set this bit during read transfers unless the SD card supports Read Wait and has set Read Wait Control to 1. In case of write transfers in which the host driver writes data to the Buffer Data Port register, the host driver shall set this bit after all block data is written. If this bit is set to 1, the host driver shall not write data to Buffer data port register. This bit affects Read Transfer Active, Write Transfer Active, DAT line active and Command Inhibit (DAT) in the Present State register. 0_B Transfer 1_B Stop This bit can be set and cleared only by software.

There are three cases to restart the transfer after stop at the block gap. Which case is appropriate depends on whether the host controller issues a Suspend command or the SD card accepts the Suspend command.

1. If the host driver does not issue Suspend command, the Continue Request shall be used to restart the transfer.
2. If the host driver issues a Suspend command and the SD card accepts it, a Resume Command shall be used to restart the transfer.
3. If the host driver issues a Suspend command and the SD card does not accept it, the Continue Request shall be used to restart the transfer.

Any time Stop At Block Gap Request stops the data transfer, the host driver shall wait for Transfer Complete (in the Normal Interrupt Status register) before attempting to restart the transfer. When restarting the data transfer by Continue Request, the host driver shall clear Stop At Block Gap Request before or simultaneously.

SDMMC_WAKEUP_CTRL

Wakeup functionality depends on the host controller system hardware and software. The host driver shall maintain voltage on the SD Bus, by setting SD Bus power to 1 in the Power Control register, when wakeup event via card interrupt is desired.

SDMMC_WAKEUP_CTRL
Wake-up Control Register **($002B_H$)** **Reset Value: 00_H**

7	6	5	4	3	2	1	0
0					**WAKEUP_EVENT_EN_REM**	**WAKEUP_EVENT_EN_INS**	**WAKEUP_EVENT_EN_INT**
r					rw	rw	rw

Field	Bits	Type	Description
0	[7:3]	r	**Reserved** Read as 0; should be written with 0.
WAKEUP_EVENT_EN_REM	2	rw	**Wakeup Event Enable On SD Card Removal** This bit enables wakeup event via Card Removal assertion in the Normal Interrupt Status register. FN_WUS (Wake up Support) in CIS does not affect this bit. 0_B Disable 1_B Enable This bit can be set and cleared only by software.
WAKEUP_EVENT_EN_INS	1	rw	**Wakeup Event Enable On SD Card Insertion** This bit enables wakeup event via Card Insertion assertion in the Normal Interrupt Status register. FN_WUS (Wake up Support) in CIS does not affect this bit. 0_B Disable 1_B Enable This bit can be set and cleared only by software.
WAKEUP_EVENT_EN_INT	0	rw	**Wakeup Event Enable On Card Interrupt** This bit enables wakeup event via Card Interrupt assertion in the Normal Interrupt Status register. This bit can be set to 1 if FN_WUS (Wake Up Support) in CIS is set to 1. 0_B Disable 1_B Enable This bit can be set and cleared only by software.

SDMMC_CLOCK_CTRL

This register is used to configure the SD Clock.

SDMMC_CLOCK_CTRL
Clock Control Register **($002C_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SDCLK_FREQ_SEL								0		0			SDCLOCK_EN	INTERNAL_CLOCKS	INTERNAL_CLOCKE
rw								rw		r			rw	r	rw

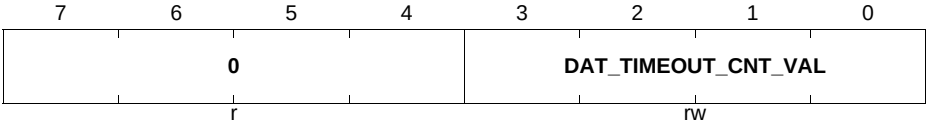
Field	Bits	Type	Description
SDCLK_FREQ_SEL	[15:8]	rw	**SDCLK Frequency Select** This register is used to select the frequency of the SDCLK pin. The frequency is not programmed directly; this register holds the divisor of the Base Clock Frequency for SD clock. Only the following settings are allowed. **8-bit Divided Clock Mode** 00_H base clock(10MHz-63MHz) 01_H base clock divided by 2 10_H base clock divided by 32 02_H base clock divided by 4 04_H base clock divided by 8 08_H base clock divided by 16 20_H base clock divided by 64 40_H base clock divided by 128 80_H base clock divided by 256 Setting 00_H specifies the highest frequency of the SD Clock. When setting multiple bits, the most significant bit is used as the divisor. But multiple bits should not be set. The two default divider values can be calculated by the Base Clock Frequency for SD Clock (48MHz). 1. 25 MHz divider value 2. 400 kHz divider value The frequency of the SDCLK is set by the following formula: Clock Frequency = (Base clock) / divisor. Thus choose the smallest possible divisor which results in a clock frequency that is less than or equal to the target frequency. Maximum Frequency for SD = 48 MHz (base clock) Maximum Frequency for MMC = 48 MHz (base clock) Minimum Frequency = 187.5 kHz (48 MHz / 256), same calculation for MMC This bit can be set and cleared only by software.
0	[7:6]	rw	**Reserved** Read as 0; must be written with 0.
0	[5:3]	r	**Reserved** Read as 0; should be written with 0.

Field	Bits	Type	Description
SDCLOCK_EN	2	rw	**SD Clock Enable** The host controller shall stop SDCLK when writing this bit to 0. SDCLK frequency Select can be changed when this bit is 0. Then, the host controller shall maintain the same clock frequency until SDCLK is stopped (Stop at SDCLK = 0). If the host controller detects the No Card state, this bit shall be cleared. 0_B Disable 1_B Enable This bit can be set and cleared only by software.
INTERNAL_CLOCK_STABLE	1	r	**Internal Clock Stable** This bit is set to 1 when SD clock is stable after writing to Internal Clock Enable in this register to 1. The SD Host Driver shall wait to set SD Clock Enable until this bit is set to 1. *Note: This is useful when using PLL for a clock oscillator that requires setup time.* 0_B Not Ready 1_B Ready This bit is initialized to 0 at reset.
INTERNAL_CLOCK_EN	0	rw	**Internal Clock Enable** This bit is set to 0 when the host driver is not using the host controller or the host controller awaits a wakeup event. The host controller should stop its internal clock to go very low power state. Still, registers shall be able to be read and written. Clock starts to oscillate when this bit is set to 1. When clock oscillation is stable, the host controller shall set Internal Clock Stable in this register to 1. This bit shall not affect card detection. 0_B Stop 1_B Oscillate This bit can be set and cleared only by software.

SDMMC_TIMEOUT_CTRL

This register is used to configure the interval for data timeout.

SDMMC_TIMEOUT_CTRL
Timeout Control Register **($002E_H$)** **Reset Value: 00_H**



Field	Bits	Type	Description
0	[7:4]	r	**Reserved** Read as 0; should be written with 0.
DAT_TIMEOUT_CNT_VAL	[3:0]	rw	**Data Timeout Counter Value** This value determines the interval by which DAT line time-outs are detected. Refer to the Data Time-out Error in the Error Interrupt Status register for information on factors that dictate time-out generation. Time-out clock frequency will be generated by dividing the sdclock TMCLK by this value. When setting this register, prevent inadvertent time-out events by clearing the Data Time-out Error Status Enable (in the Error Interrupt Status Enable register) 0000_B TMCLK * 2^13 0001_B TMCLK * 2^14 ... 1110_B TMCLK * 2^27 1111_B Reserved This bit can be set and cleared only by software.

SDMMC_SW_RESET

A reset pulse is generated when writing 1 to each bit of this register. After completing the reset, the host controller shall clear each bit. Because it takes some time to complete software reset, the SD Host Driver shall confirm that these bits are 0.

SDMMC_SW_RESET
Software Reset Register **($002F_H$)** **Reset Value: 00_H**

7	6	5	4	3	2	1	0
0					**SW_RST_DAT_LINE**	**SW_RST_CMD_LINE**	**SW_RST_ALL**
r					rw	rw	rw

Field	Bits	Type	Description
0	[7:3]	r	**Reserved** Read as 0; should be written with 0.
SW_RST_DAT_LINE	2	rw	**Software Reset for DAT Line** Only part of data circuit is reset. The following registers and bits are cleared by this bit: **Buffer Data Port Register** Buffer is cleared and Initialized. **Present State register** Buffer read Enable Buffer write Enable Read Transfer Active Write Transfer Active DAT Line Active Command Inhibit (DAT) **Block Gap Control register** Continue Request Stop At Block Gap Request **Normal Interrupt Status register** Buffer Read Ready Buffer Write Ready Block Gap Event Transfer Complete 0_B Work 1_B Reset

Field	Bits	Type	Description
SW_RST_CMD_LINE	1	rw	**Software Reset for CMD Line** Only part of command circuit is reset. The following registers and bits are cleared by this bit: **Present State Register** Command Inhibit (CMD) **Normal Interrupt Status Register** Command Complete 0_B Work 1_B Reset
SW_RST_ALL	0	rw	**Software Reset for All**

SDMMC_INT_STATUS_NORM

The Normal Interrupt Status Enable affects read of this register, but Normal Interrupt Signal does not affect these reads. An Interrupt is generated when the Normal Interrupt Signal Enable is enabled and at least one of the status bits is set to 1. For all bits except Card Interrupt and Error Interrupt, writing 1 to a bit clears it. The Card Interrupt is cleared when the card stops asserting the interrupt: that is when the Card Driver services the Interrupt condition.

SDMMC_INT_STATUS_NORM
Normal Interrupt Status Register (0030$_H$) **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERR_INT	0		0				CARD_INT	CARD_REMOVAL	CARD_INS	BUF_READ_READY	BUF_WRITE_READY	0	BLOCK_GAP_EVENT	TX_COMPLETE	CMD_COMPLETE
r	rw		r				r	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
ERR_INT	15	r	**Error Interrupt** If any of the bits in the Error Interrupt Status Register are set, then this bit is set. Therefore the host driver can test for an error by checking this bit first. 0_B No Error. 1_B Error. This bit is initialized to 0 at reset.
0	[14:13]	rw	**Reserved** Read as 0; must be written with 0.
0	[12:9]	r	**Reserved** Read as 0; should be written with 0.

Field	Bits	Type	Description
CARD_INT	8	r	**Card Interrupt** Writing this bit to 1 does not clear this bit. It is cleared by resetting the SD card interrupt factor. In 1-bit mode, the host controller shall detect the Card Interrupt without SD Clock to support wakeup. In 4-bit mode, the card interrupt signal is sampled during the interrupt cycle, so there are some sample delays between the interrupt signal from the card and the interrupt to the Host system. When this status has been set and the host driver needs to start this interrupt service, Card Interrupt Status Enable in the Normal Interrupt Status register shall be set to 0 in order to clear the card interrupt statuses latched in the host controller and stop driving the Host System. After completion of the card interrupt service (the reset factor in the SD card and the interrupt signal may not be asserted), set Card Interrupt Status Enable to 1 and start sampling the interrupt signal again. Interrupt detected by DAT[1] is supported when there is a card in slot. 0_B No Card Interrupt 1_B Generate Card Interrupt
CARD_REMOVAL	7	rw	**Card Removal** This status is set if the Card Inserted in the Present State register changes from 1 to 0. When the host driver writes this bit to 1 to clear this status the status of the Card Inserted in the Present State register should be confirmed. Because the card detect may possibly be changed when the host driver clear this bit an Interrupt event may not be generated. 0_B Card State Stable or Debouncing 1_B Card Removed This bit can be cleared by a software write of 1 to the bit. A software write of 0 to the bit has no effect.

Field	Bits	Type	Description
CARD_INS	6	rw	**Card Insertion** This status is set if the Card Inserted in the Present State register changes from 0 to 1. When the host driver writes this bit to 1 to clear this status the status of the Card Inserted in the Present State register should be confirmed. Because the card detect may possibly be changed when the host driver clear this bit an Interrupt event may not be generated. 0_B Card State Stable or Debouncing 1_B Card Inserted This bit can be cleared by a software write of 1 to the bit. A software write of 0 to the bit has no effect.
BUFF_READ_READY	5	rw	**Buffer Read Ready** This status is set if the Buffer Read Enable changes from 0 to 1. 0_B Not Ready to read Buffer. 1_B Ready to read Buffer. This bit can be cleared by a software write of 1 to the bit. A software write of 0 to the bit has no effect.
BUFF_WRITE_READY	4	rw	**Buffer Write Ready** This status is set if the Buffer Write Enable changes from 0 to 1. 0_B Not Ready to Write Buffer. 1_B Ready to Write Buffer. This bit can be cleared by a software write of 1 to the bit. A software write of 0 to the bit has no effect.
0	3	rw	**Reserved** Read as 0; must be written with 0.

Field	Bits	Type	Description
BLOCK_GAP_EVENT	2	rw	**Block Gap Event** If the Stop At Block Gap Request in the Block Gap Control Register is set, this bit is set. Read Transaction: This bit is set at the falling edge of the DAT Line Active Status (When the transaction is stopped at SD Bus timing. The Read Wait must be supported inorder to use this function). Write Transaction: This bit is set at the falling edge of Write Transfer Active Status (After getting CRC status at SD Bus timing). 0_B No Block Gap Event 1_B Transaction stopped at Block Gap This bit can be cleared by a software write of 1 to the bit. A software write of 0 to the bit has no effect.

Field	Bits	Type	Description
TX_COMPLETE	1	rw	**Transfer Complete** This bit is set when a read / write transaction is completed. Read Transaction: This bit is set at the falling edge of Read Transfer Active Status. There are two cases in which the Interrupt is generated. The first is when a data transfer is completed as specified by data length (After the last data has been read to the Host System). The second is when data has stopped at the block gap and completed the data transfer by setting the Stop At Block Gap Request in the Block Gap Control Register (After valid data has been read to the Host System). Write Transaction: This bit is set at the falling edge of the DAT Line Active Status. There are two cases in which the Interrupt is generated. The first is when the last data is written to the card as specified by data length and Busy signal is released. The second is when data transfers are stopped at the block gap by setting Stop At Block Gap Request in the Block Gap Control Register and data transfers completed. (After valid data is written to the SD card and the busy signal is released). Transfer Complete has higher priority than Data Time-out Error. If both bits are set to 1, the data transfer can be considered complete 0_B No Data Transfer Complete 1_B Data Transfer Complete This bit can be cleared by a software write of 1 to the bit. A software write of 0 to the bit has no effect.

Field	Bits	Type	Description
CMD_COMPLETE	0	rw	**Command Complete** This bit is set when get the end bit of the command response (Except Auto CMD12). Command Time-out Error has higher priority than Command Complete. If both are set to 1, it can be considered that the response was not received correctly. 0_B No Command Complete 1_B Command Complete This bit can be cleared by a software write of 1 to the bit. A software write of 0 to the bit has no effect.

Table 13-7 Relation between transfer complete and data timeout error

Transfer Complete	Data Timeout Error	Meaning of the Status
0	0	Interrupted by Another Factor.
0	1	Timeout occur during transfer.
1	Don’t Care	Data Transfer Complete

Table 13-8 Relation between command complete and command timeout error

Command Complete	Command Timeout Error	Meaning of the Status
0	0	Interrupted by Another Factor.
Don’t Care	1	Response not received within 64 SDCLK cycles.
1	0	Response Received

SDMMC_INT_STATUS_ERR

Status defined in this register can be enabled by the Error Interrupt Status Enable Register, but not by the Error Interrupt Signal Enable Register. The Interrupt is generated when the Error Interrupt Signal Enable is enabled and at least one of the statuses is set to 1. Writing to 1 clears the bit and writing to 0 keeps the bit unchanged. More than one status can be cleared at a single register write.

SDMMC_INT_STATUS_ERR
Error Interrupt Status Register **(0032_H)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		CEATA_ERR	0	0	0		ACMD_ERR	CURRENT_LIMIT_ERR	DATA_END_BIT_ERR	DATA_CRC_ERR	DATA_TIMEOUT_ERR	CMD_IND_ERR	CMD_END_BIT_ERR	CMD_CRC_ERR	CMD_TIMEOUT_ERR
r		rw	rw	r	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
0	[15:14]	r	**Reserved** Read as 0; should be written with 0.
CEATA_ERR	13	rw	**Ceata Error Status** Occurs when ATA command termination has occured due to an error condition the device has encountered. 0_B no error 1_B error This bit can be cleared by a software write of 1 to the bit. A software write of 0 to the bit has no effect.
0	12	rw	**Reserved** Read as 0; must be written with 0.
0	11	r	**Reserved** Read as 0; should be written with 0.
0	[10:9]	rw	**Reserved** Read as 0; must be written with 0.

Field	Bits	Type	Description
ACMD_ERR	8	rw	**Auto CMD Error** Auto CMD12 uses this error status. This bit is set when detecting that one of the bits D00-D04 in Auto CMD Error Status register has changed from 0 to 1. In case of Auto CMD12, this bit is set to 1, not only when the errors in Auto CMD12 occur but also when Auto CMD12 is not executed due to the previous command error. 0_B No Error 1_B Error This bit can be cleared by a software write of 1 to the bit. A software write of 0 to the bit has no effect.
CURRENT_LIMIT_ERR	7	rw	**Current Limit Error** By setting the SD Bus Power bit in the Power Control Register, the host controller is requested to supply power for the SD Bus. If the host controller supports the Current Limit Function, it can be protected from an Illegal card by stopping power supply to the card in which case this bit indicates a failure status. Reading 1 means the host controller is not supplying power to SD card due to some failure. Reading 0 means that the host controller is supplying power and no error has occurred. This bit shall always set to be 0, if the host controller does not support this function. 0_B No Error 1_B Power Fail This bit can be cleared by a software write of 1 to the bit. A software write of 0 to the bit has no effect.
DATA_END_BIT_ERR	6	rw	**Data End Bit Error** Occurs when detecting 0 at the end bit position of read data which uses the DAT line or the end bit position of the CRC status. 0_B No Error 1_B Error This bit can be cleared by a software write of 1 to the bit. A software write of 0 to the bit has no effect.

Field	Bits	Type	Description
DATA_CRC_ERR	5	rw	**Data CRC Error** Occurs when detecting CRC error when transferring read data which uses the DAT line or when detecting the Write CRC Status having a value of other than “010”. 0_B No Error 1_B Error This bit can be cleared by a software write of 1 to the bit. A software write of 0 to the bit has no effect.
DATA_TIMEOUT_ERR	4	rw	**Data Timeout Error** Occurs when detecting one of following timeout conditions. Busy Timeout for R1b, R5b type. Busy Timeout after Write CRC status Write CRC status Timeout Read Data Timeout 0_B No Error 1_B Timeout This bit can be cleared by a software write of 1 to the bit. A software write of 0 to the bit has no effect.
CMD_IND_ERR	3	rw	**Command Index Error** Occurs if a Command Index error occurs in the Command Response. 0_B No Error 1_B Error This bit can be cleared by a software write of 1 to the bit. A software write of 0 to the bit has no effect.
CMD_END_BIT_ERR	2	rw	**Command End Bit Error** Occurs when detecting that the end bit of a command response is 0. 0_B No Error 1_B End Bit Error Generated This bit can be cleared by a software write of 1 to the bit. A software write of 0 to the bit has no effect.

Field	Bits	Type	Description
CMD_CRC_ERR	1	rw	**Command CRC Error** Command CRC Error is generated in two cases. If a response is returned and the Command Time-out Error is set to 0, this bit is set to 1 when detecting a CRT error in the command response The host controller detects a CMD line conflict by monitoring the CMD line when a command is issued. If the host controller drives the CMD line to 1 level, but detects 0 level on the CMD line at the next SDCLK edge, then the host controller shall abort the command (Stop driving CMD line) and set this bit to 1. The Command Timeout Error shall also be set to 1 to distinguish CMD line conflict. 0_B No Error 1_B CRC Error Generated This bit can be cleared by a software write of 1 to the bit. A software write of 0 to the bit has no effect.
CMD_TIMEOUT_ERR	0	rw	**Command Timeout Error** Occurs only if the no response is returned within 64 SDCLK cycles from the end bit of the command. If the host controller detects a CMD line conflict, in which case Command CRC Error shall also be set. This bit shall be set without waiting for 64 SDCLK cycles because the command will be aborted by the host controller. 0_B No Error 1_B Timeout This bit can be cleared by a software write of 1 to the bit. A software write of 0 to the bit has no effect.

Table 13-9 Relation between command CRC error and command time-out error

Command CRC Error	Command Time-out Error	Kinds of Error
0	0	No Error
0	1	Response Timeout Error
1	0	Response CRC Error
1	1	CMD Line Conflict

SDMMC_EN_INT_STATUS_NORM

Interrupt status can be enabled by writing 1 to the bit in this register. The host controller may sample the card Interrupt signal during interrupt period and may hold its value in the flip-flop. If the Card Interrupt Status Enable is set to 0, the host controller shall clear all internal signals regarding Card Interrupt.

SDMMC_EN_INT_STATUS_NORM

Normal Interrupt Status Enable Register(0034_H) **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FIXED_TO_0	0						CARD_INT_EN	CARD_REMOVAL_EN	CARD_INS_EN	BUF_READ_READY	BUF_WRITE_READY	0	BLOCK_GAP_EVENT	TX_COMPLETE_EN	CMD_COMPLETE_EN
r	rw						rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
FIXED_TO_0	15	r	**Fixed to 0** The host controller shall control error Interrupts using the Error Interrupt Status Enable register.
0	[14:9]	rw	**Reserved** Read as 0; must be written with 0.
CARD_INT_EN	8	rw	**Card Interrupt Status Enable** If this bit is set to 0, the host controller shall clear Interrupt request to the System. The Card Interrupt detection is stopped when this bit is cleared and restarted when this bit is set to 1. The host driver may clear the Card Interrupt Status Enable before servicing the Card Interrupt and may set this bit again after all Interrupt requests from the card are cleared to prevent inadvertent Interrupts. 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
CARD_REMOVAL_EN	7	rw	**Card Removal Status Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.

Field	Bits	Type	Description
CARD_INS_EN	6	rw	**Card Insertion Status Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
BUFF_READ_READY_EN	5	rw	**Buffer Read Ready Status Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
BUFF_WRITE_READY_EN	4	rw	**Buffer Write Ready Status Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
0	3	rw	**Reserved** Read as 0; must be written with 0.
BLOCK_GAP_EVENT_EN	2	rw	**Block Gap Event Status Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
TX_COMPLETE_EN	1	rw	**Transfer Complete Status Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
CMD_COMPLETE_EN	0	rw	**Command Complete Status Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.

SDMMC_EN_INT_STATUS_ERR

Interrupt status can be enabled by writing 1 to the bit in this register. To Detect CMD Line conflict, the host driver must set both Command Time-out Error Status Enable and Command CRC Error Status Enable to 1.

SDMMC_EN_INT_STATUS_ERR
Error Interrupt Status Enable Register(0036_H) **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VSES1514_EN		CEATA_ERR_EN	TARGET_RESP_ERR	0	0		ACMD_ERR_EN	CURRENT_LIMIT_ERR	DATA_END_BIT_ERR	DATA_CRC_ERR_EN	DATA_TIMEOUT_ERR	CMD_IND_ERR_EN	CMD_END_BIT_ERR_	CMD_CRC_ERR_EN	CMD_TIMEOUT_ERR_
r		rw	rw	r	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
0	[15:14]	r	**Reserved** Read as 0; should be written with 0.
CEATA_ERR_EN	13	rw	**Ceata Error Status Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
TARGET_RESP_ERR_EN	12	rw	**Target Response Error Status Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
0	11	r	**Reserved** Read as 0; should be written with 0.
0	[10:9]	rw	**Reserved** Read as 0; must be written with 0.
ACMD_ERR_EN	8	rw	**Auto CMD12 Error Status Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
CURRENT_LIMIT_ERR_EN	7	rw	**Current Limit Error Status Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.

Field	Bits	Type	Description
DATA_END_BIT_ERR_EN	6	rw	**Data End Bit Error Status Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
DATA_CRC_ERR_EN	5	rw	**Data CRC Error Status Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
DATA_TIMEOUT_ERR_EN	4	rw	**Data Timeout Error Status Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
CMD_IND_ERR_EN	3	rw	**Command Index Error Status Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
CMD_END_BIT_ERR_EN	2	rw	**Command End Bit Error Status Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
CMD_CRC_ERR_EN	1	rw	**Command CRC Error Status Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
CMD_TIMEOUT_ERR_EN	0	rw	**Command Timeout Error Status Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.

SDMMC_EN_INT_SIGNAL_NORM

This register is used to select which interrupt status is indicated to the Host System as the Interrupt. The interrupt line is shared by all the status bits. Interrupt generation can be enabled by writing 1 to any of these bits.

SDMMC_EN_INT_SIGNAL_NORM
Normal Interrupt Signal Enable Register(0038_H) **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FIXED_TO_0	0						CARD_INT_EN	CARD_REMOVAL_EN	CARD_INS_EN	BUFF_READ_READY	BUFF_WRITE_READY	0	BLOCK_GAP_EVENT	TX_COMPLETE_EN	CMD_COMPLETE_EN
r	rw						rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
FIXED_TO_0	15	r	**Fixed to 0** The host driver shall control error Interrupts using the Error Interrupt Signal Enable register.
0	[14:9]	rw	**Reserved** Read as 0; must be written with 0.
CARD_INT_EN	8	rw	**Card Interrupt Signal Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
CARD_REMOVAL_EN	7	rw	**Card Removal Signal Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
CARD_INS_EN	6	rw	**Card Insertion Signal Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
BUFF_READ_READY_EN	5	rw	**Buffer Read Ready Signal Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.

Field	Bits	Type	Description
BUFF_WRITE_READY_EN	4	rw	**Buffer Write Ready Signal Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
0	3	rw	**Reserved** Read as 0; must be written with 0.
BLOCK_GAP_EVENT_EN	2	rw	**Block Gap Event Signal Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
TX_COMPLETE_EN	1	rw	**Transfer Complete Signal Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
CMD_COMPLETE_EN	0	rw	**Command Complete Signal Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.

SDMMC_EN_INT_SIGNAL_ERR

This register is used to select which interrupt status is notified to the Host System as the Interrupt. The interrupt line is shared by all the status bits. Interrupt generation can be enabled by writing 1 to any of these bits.

SDMMC_EN_INT_SIGNAL_ERR
Error Interrupt Signal Enable Register(003A$_H$) **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		CEATA_ERR_EN	TARGET_RESP_ERR	0	0		ACMD_ERR_EN	CURRENT_LIMIT_ERR	DATA_END_BIT_ERR	DATA_CRC_ERR_EN	DATA_TIMEOUT_ERR	CMD_IND_ERR_EN	CMD_END_BIT_ERR_	CMD_CRC_ERR_EN	CMD_TIMEOUT_ERR_
r		rw	rw	r	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
0	[15:14]	r	**Reserved** Read as 0; should be written with 0.
CEATA_ERR_EN	13	rw	**Ceata Error Signal Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
TARGET_RESP_ERR_EN	12	rw	**Target Response Error Signal Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
0	11	r	**Reserved** Read as 0; should be written with 0.
0	[10:9]	rw	**Reserved** Read as 0; must be written with 0.
ACMD_ERR_EN	8	rw	**Auto CMD12 Error Signal Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
CURRENT_LIMIT_ERR_EN	7	rw	**Current Limit Error Signal Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.

Field	Bits	Type	Description
DATA_END_BIT_ERR_EN	6	rw	**Data End Bit Error Signal Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
DATA_CRC_ERR_EN	5	rw	**Data CRC Error Signal Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
DATA_TIMEOUT_ERR_EN	4	rw	**Data Timeout Error Signal Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
CMD_IND_ERR_EN	3	rw	**Command Index Error Signal Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
CMD_END_BIT_ERR_EN	2	rw	**Command End Bit Error Signal Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
CMD_CRC_ERR_EN	1	rw	**Command CRC Error Signal Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.
CMD_TIMEOUT_ERR_EN	0	rw	**Command Timeout Error Signal Enable** 0_B Masked 1_B Enabled This bit can be set and cleared only by software.

SDMMC_ACMD_ERR_STATUS

This register is used to indicate CMD12 response error of Auto CMD12. The Host driver can determine what kind of Auto CMD12 errors occur by this register. This register is valid only when the Auto CMD Error is set.

SDMMC_ACMD_ERR_STATUS
Auto CMD Error Status Register (003C$_H$) **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								CMD_NOT_ISSUED_B	0		ACMD_IND_ERR	ACMD_END_BIT_ERR	ACMD_CRC_ERR	ACMD_TIMEOUT_ERR	ACMD12_NOT_EXEC_
r								r	r		r	r	r	r	r

Field	Bits	Type	Description
0	15:8	r	**Reserved** Read as 0; should be written with 0.
CMD_NOT_ISSUED_BY_ACMD12_ERR	7	r	**Command Not Issued By Auto CMD12 Error** Setting this bit to 1 means CMD_wo_DAT is not executed due to an Auto CMD12 error (D04 - D01) in this register. 0_B No Error 1_B Not Issued This bit is initialized to 0 at reset.
0	[6:5]	r	**Reserved** Read as 0; should be written with 0.
ACMD_IND_ERR	4	r	**Auto CMD Index Error** Occurs if the Command Index error occurs in response to a command. 0_B No Error 1_B Error This bit is initialized to 0 at reset.
ACMD_END_BIT_ERR	3	r	**Auto CMD End Bit Error** Occurs when detecting that the end bit of command response is 0. 0_B No Error 1_B End Bit Error Generated This bit is initialized to 0 at reset.

Field	Bits	Type	Description
ACMD_CRC_ERR	2	r	**Auto CMD CRC Error** Occurs when detecting a CRC error in the command response. 0_B No Error 1_B CRC Error Generated This bit is initialized to 0 at reset.
ACMD_TIMEOUT_ERR	1	r	**Auto CMD Timeout Error** Occurs if the no response is returned within 64 SDCLK cycles from the end bit of the command. If this bit is set to 1, the other error status bits (D04 - D02) are meaningless. 0_B No Error 1_B Timeout This bit is initialized to 0 at reset.
ACMD12_NOT_EXEC_ERR	0	r	**Auto CMD12 Not Executed** If memory multiple block data transfer is not started due to command error, this bit is not set because it is not necessary to issue Auto CMD12. Setting this bit to 1 means the host controller cannot issue Auto CMD12 to stop memory multiple block transfer due to some error. If this bit is set to 1, other error status bits (D04 - D01) are meaningless. 0_B Executed 1_B Not Executed This bit is initialized to 0 at reset.

Table 13-10 Relation between Auto CMD12 CRC error and Auto CMD12 timeout error

Auto Cmd12 CRC Error	Auto CMD12 Timeout Error	Kinds of Error
0	0	No Error
0	1	Response Timeout Error
1	0	Response CRC Error
1	1	CMD Line Conflict

The timing of changing Auto CMD12 Error Status can be classified in three scenarios:

1. When the host controller is going to issue Auto CMD12.
   a) Set D00 to 1 if Auto CMD12 cannot be issued due to an error in the previous command.
   b) Set D00 to 0 if Auto CMD12 is issued.
2. At the end bit of Auto CMD12 response.
   a) Check received responses by checking the error bits D01, D02, D03, D04.
   b) Set to 1 if Error is Detected.
   c) Set to 0 if Error is Not Detected.
3. Before reading the Auto CMD12 Error Status bit D07
   a) Set D07 to 1 if there is a command cannot be issued.
   b) Set D07 to 0 if there is no command to issue.

Timing of generating the Auto CMD12 Error and writing to the Command register are Asynchronous. Then D07 shall be sampled when driver never writing to the Command register. So just before reading the Auto CMD12 Error Status register is good timing to set the D07 status bit.

SDMMC_FORCE_EVENT_ACMD_ERR_STATUS

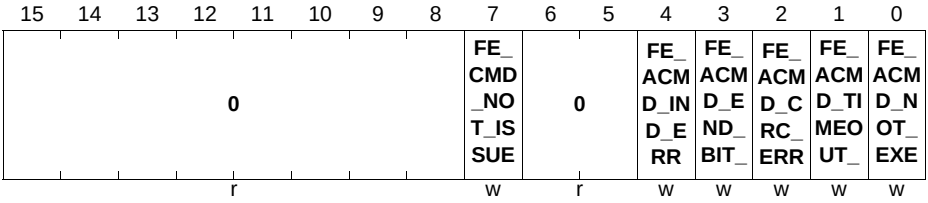
The Force Event Register is an address at which the Auto CMD12 Error Status Register can be written.

Writing 1 : set each bit of the Auto CMD12 Error Status Register

Writing 0 : no effect.

SDMMC_FORCE_EVENT_ACMD_ERR_STATUS

Force Event Register for Auto CMD Error Status(0050_H) **Reset Value: 0000_H**



Field	Bits	Type	Description
0	[15:8]	r	**Reserved** Read as 0; should be written with 0.
FE_CMD_NOT_ISSUED_ACMD12_ERR	7	w	**Force Event for CMD not issued by Auto CMD12 Error** 0_B No interrupt 1_B Interrupt is generated
0	[6:5]	r	**Reserved** Read as 0; should be written with 0.
FE_ACMD_IND_ERR	4	w	**Force Event for Auto CMD Index Error** 0_B No interrupt 1_B Interrupt is generated
FE_ACMD_END_BIT_ERR	3	w	**Force Event for Auto CMD End bit Error** 0_B No interrupt 1_B Interrupt is generated
FE_ACMD_CRC_ERR	2	w	**Force Event for Auto CMD CRC Error** 0_B No interrupt 1_B Interrupt is generated
FE_ACMD_TIMEOUT_ERR	1	w	**Force Event for Auto CMD timeout Error** 0_B No interrupt 1_B Interrupt is generated

Field	Bits	Type	Description
FE_ACMD_NOT_EXEC	0	w	**Force Event for Auto CMD12 NOT Executed** 0_B No interrupt 1_B Interrupt is generated

SDMMC_FORCE_EVENT_ERR_STATUS

The Force Event Register is an address at which the Error Interrupt Status register can be written. The effect of a write to this address will be reflected in the Error Interrupt Status Register if the corresponding bit of the Error Interrupt Status Enable Register is set.

Writing 1 : set each bit of the Error Interrupt Status Register

Writing 0 : no effect

SDMMC_FORCE_EVENT_ERR_STATUS

Force Event Register for Error Interrupt Status(0052_H) **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		FE_CEATA_ERR	FE_TARGET_RESPON	0		0	FE_ACMD12_ERR	FE_CURREN T_LIMIT_	FE_DATA_END_BIT_	FE_DATA_CRC_ERR	FE_DATA_TIMEOUT_	FE_CMD_IND_ERR	FE_CMD_END_BIT_E	FE_CMD_CRC_ERR	FE_CMD_TIMEOUT_E
w		w	w	r		w	w	w	w	w	w	w	w	w	w

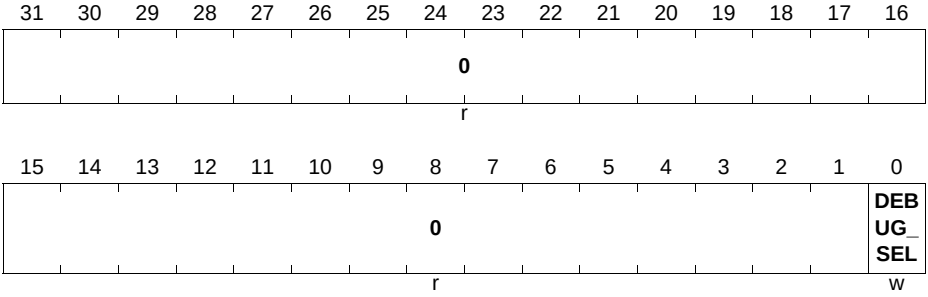
Field	Bits	Type	Description
0	[15:14]	w	**Reserved** Must be written with 0.
FE_CEATA_ERR	13	w	**Force Event for Ceata Error** 0_B No interrupt 1_B Interrupt is generated
FE_TARGET_RESPONSE_ERR	12	w	**Force event for Target Response Error** 0_B No interrupt 1_B Interrupt is generated
0	[11:10]	r	**Reserved** Read as 0; should be written with 0.
0	9	w	**Reserved** Must be written with 0.
FE_ACMD12_ERR	8	w	**Force Event for Auto CMD Error** 0_B No interrupt 1_B Interrupt is generated
FE_CURRENT_LIMIT_ERR	7	w	**Force Event for Current Limit Error** 0_B No interrupt 1_B Interrupt is generated

Field	Bits	Type	Description
FE_DATA_END_BIT_ERR	6	w	**Force Event for Data End Bit Error** 0_B No interrupt 1_B Interrupt is generated
FE_DATA_CRC_ERR	5	w	**Force Event for Data CRC Error** 0_B No interrupt 1_B Interrupt is generated
FE_DATA_TIMEOUT_ERR	4	w	**Force Event for Data Timeout Error** 0_B No interrupt 1_B Interrupt is generated
FE_CMD_IND_ERR	3	w	**Force Event for Command Index Error** 0_B No interrupt 1_B Interrupt is generated
FE_CMD_END_BIT_ERR	2	w	**Force Event for Command End Bit Error** 0_B No interrupt 1_B Interrupt is generated
FE_CMD_CRC_ERR	1	w	**Force Event for Command CRC Error** 0_B No interrupt 1_B Interrupt is generated
FE_CMD_TIMEOUT_ERR	0	w	**Force Event for Command Timeout Error** 0_B No interrupt 1_B Interrupt is generated

SDMMC_DEBUG_SEL

This register is used to select the debug mode.

SDMMC_DEBUG_SEL
Debug Selection Register **(0074_H)** **Reset Value: 00000000_H**

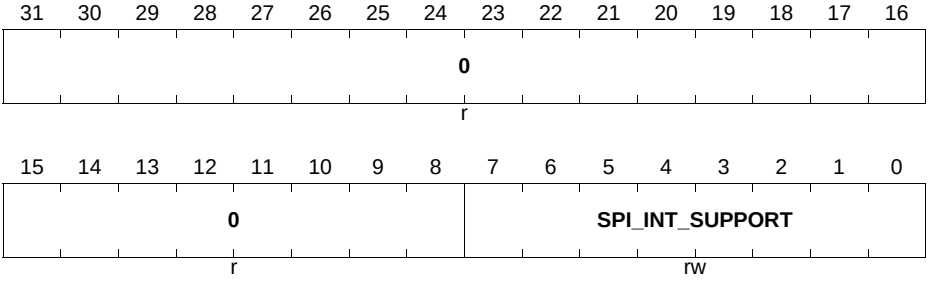


Field	Bits	Type	Description
0	[31:1]	r	**Reserved** Read as 0; should be written with 0.
DEBUG_SEL	0	w	**Debug_sel** 0_B receiver module and fifo_ctrl module signals are probed out 1_B cmd register, Interrupt status, transmitter module and clk sdcard signals are probed out.

SDMMC_SPI

This register is used to configure the SPI interrupt support.

SDMMC_SPI
SPI Interrupt Support Register (00F0$_H$) Reset Value: 00000000$_H$

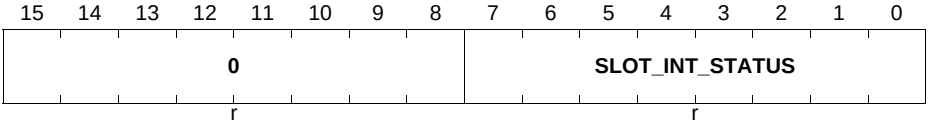


Field	Bits	Type	Description
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.
SPI_INT_SUPPORT	[7:0]	rw	**SPI INT SUPPORT** This bit is set to indicate the assertion of interrupts in the SPI mode at any time, irrespective of the status of the card select (CS) line. If this bit is zero, then SDIO card can only assert the interrupt line in the SPI mode when the CS line is asserted. This bit can be set and cleared only by software.

SDMMC_SLOT_INT_STATUS

This register is used to configure the interrupt signal for card slot.

SDMMC_SLOT_INT_STATUS
Slot Interrupt Status Register (00FC$_H$) Reset Value: 0000$_H$



Field	Bits	Type	Description
0	[15:8]	r	**Reserved** Read as 0; should be written with 0.
SLOT_INT_STATUS	[7:0]	r	**Interrupt Signal for Card Slot** These status bit indicate the Interrupt signal and Wakeup signal for the card slot. By a power on reset or by Software Reset For All, the Interrupt signal shall be de asserted and this status shall read 00$_H$. 00$_H$ Slot 1 *Note: Other values are reserved* This bit is initialized to 0 at reset.

13.13 Interconnects

The interface signals of the SDMMC Host Controller are described below.

Table 13-11 SDMMC Pin Connections

Input/ Output	I/O	Connected to	Description
clk_xin	I	SCU.EXTCLK	Input clock to SDMMC controller
SDMMC.CLK_IN	I	P3.6	Feedback clock of clk_sdcard_out from the pads
SDMMC.DATA7_IN	I	P1.13	MMC8 mode: Data7 Input
SDMMC.DATA6_IN	I	P1.12	MMC8 mode: Data6 Input
SDMMC.DATA5_IN	I	P1.9	MMC8 mode: Data5 Input
SDMMC.DATA4_IN	I	P1.8	MMC8 mode: Data4 Input
SDMMC.DATA3_IN	I	P4.1	SD4/MMC8 mode: Data3 Input
SDMMC.DATA2_IN	I	P1.7	SD4/MMC8 : Data2 input
SDMMC.DATA1_IN	I	P1.6	SD1 mode: Interrupt SD4 mode: Data1 Input or Interrupt (optional) MMC8 mode: Data1 Input
SDMMC.DATA0_IN	I	P4.0	SD1/SD4/MMC8 : Data0 Input SPI mode : Command response input, read data and crc status for write data
$\overline{\text{SDMMC.SDCD}}$	I	P1.10	Active low. Card Detection
SDMMC.SDWC	I	P1.1	Active High. SD Card Write Protect
SDMMC.CMD_IN	I	P3.5	SD1/SD4/MMC8 : Command Input
SDMMC.CLK_OUT	O	P3.6	Clock supplied to SD/MMC card
SDMMC.DATA7_OUT	O	P1.13	MMC8 mode: Data7 Output
SDMMC.DATA6_OUT	O	P1.12	MMC8 mode: Data6 Output
SDMMC.DATA5_OUT	O	P1.9	MMC8 mode: Data5 Output
SDMMC.DATA4_OUT	O	P1.8	MMC8 mode: Data4 Output
SDMMC.DATA3_OUT	O	P4.1	SD4/MMC8 mode: Data3 Output SPI mode : chip select

Table 13-11 SDMMC Pin Connections (cont'd)

Input/ Output	I/O	Connected to	Description
SDMMC.DATA2_OUT	O	P1.7	SD1 mode: Read Wait(optional) SD4 mode: Data2 Output or Read Wait (optional) MMC8 mode: Data2 Output
SDMMC.DATA1_OUT	O	P1.6	SD4/MMC8 : Data1 Output
SDMMC.DATA0_OUT	O	P4.0	SD1/SD4/MMC8 : Data0 Output
SDMMC.CMD_OUT	O	P3.5	SD1/SD4/MMC8 : Command Output
SDMMC.BUS_POWER	O	P3.4	Control Card Power Supply
SDMMC.LED	O	P3.3	LED indication
$\overline{\text{SDMMC.RST}}$	O	P0.11	Hardware reset to card
SDMMC.SR0	O	NVIC	Service request line

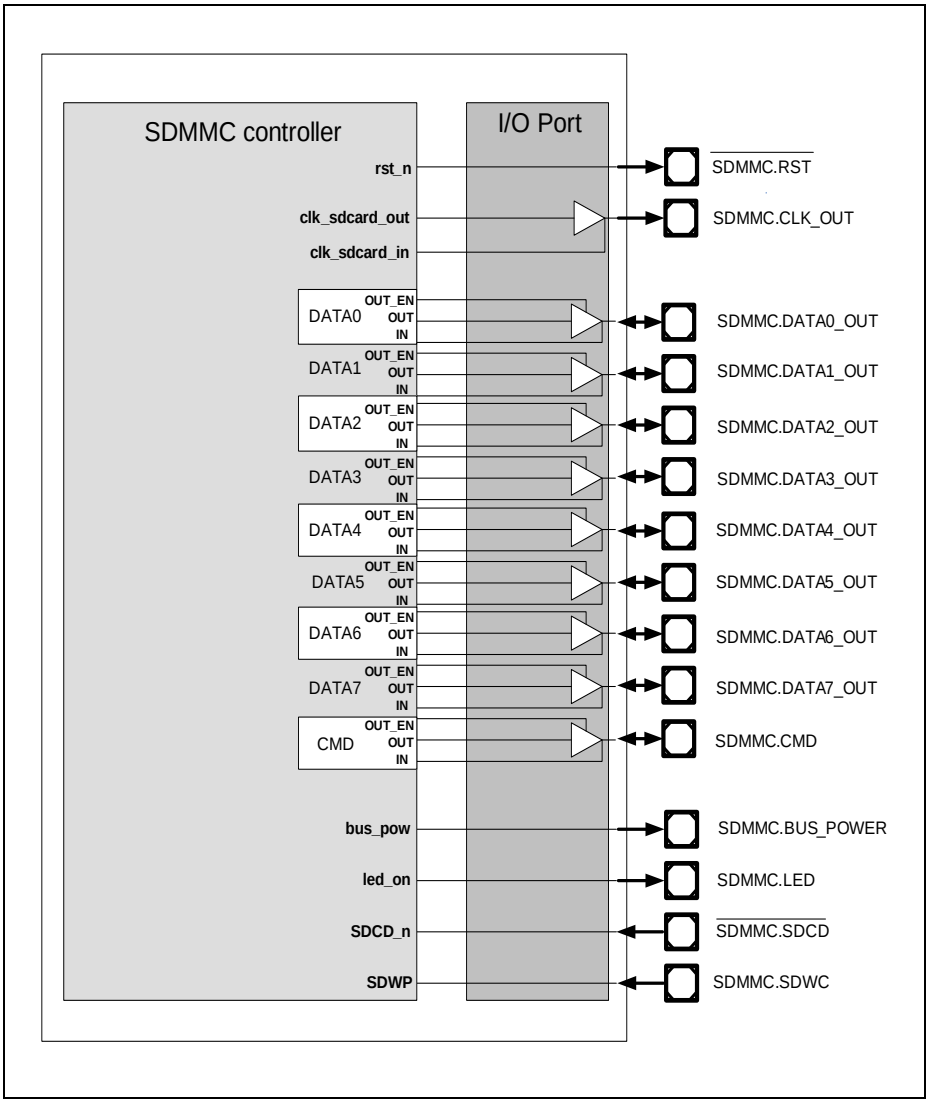


Figure 13-4 External Pin Connections of SDMMC

14 External Bus Unit (EBU)

The EBU controls the transactions between external memories or peripheral units, and the internal memories and peripheral units. Several external device configurations are supported with little or no additional circuitry, making the EBU a very powerful peripheral for expansion and support of several applications.

14.1 Overview

The Memory Controller module for ARM-based systems connects on-chip controller cores (e.g. ARM9EJ CPU, DMA Controller) to external resources such as memories and peripherals. **Figure 14-1** shows Memory Controller within a typical system.

Several type of external memories are supported, such as: Burst FLASH, Cellular RAM, SDRAM or NAND.

Any AHB master can (in conjunction with an AHB Bus Matrix) access external memories through the Memory Controller.

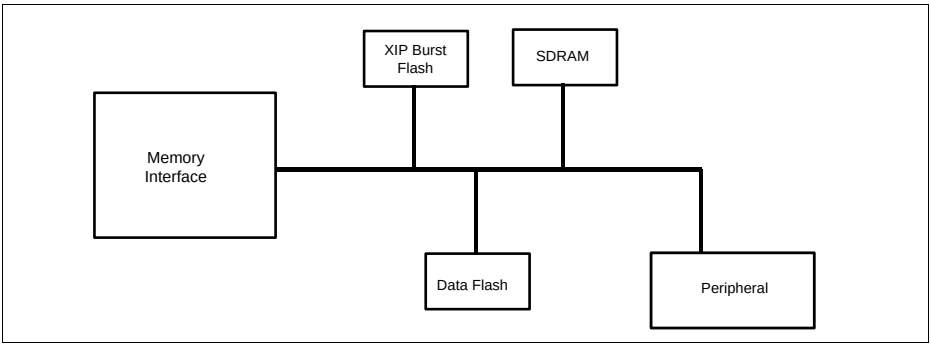


Figure 14-1 Typical External Memory System

14.1.1 Features

- External bus frequency: Module frequency: flash clock = 1:1, 1:2, 1:3 or 1:4.
- External bus frequency: Module frequency: SDRAM clock = 1:1, 1:2, or 1:4.
- Highly programmable access parameters.
- Intel-style peripheral/device support.
- Burst FLASH support (see **Section 14.12** for specific device types).
- Cellular RAM support (see **Section 14.12** for specific device types).
- SDRAM support (see **Section 14.13** for specific device types).
- NAND flash
- Asynchronous static memory device e.g. ROM, RAM, NOR Flash

- Multiplexed access (address & data on the same bus)
- Data Buffering: Two read buffers. One write buffer.
- Multiple (four) programmable address regions.
- Little-endian support.

14.1.2 Block Diagram

Figure 14-2 shows the block diagram of the EBU. The AHB2IF bridge translates the transactions into requests that can be handled by the arbiter block. The arbiter after receiving the transaction requests, it forwards them to the appropriate state machine.

The dedicated state machines are used to sequence the control signals and to coordinate accesses, to each of the different external memory/device types.

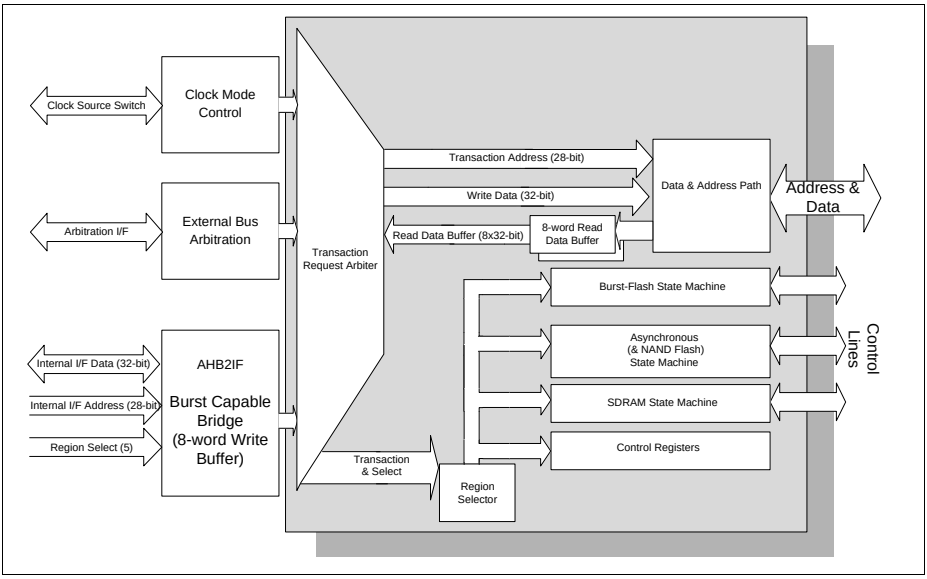


Figure 14-2 EBU Block Diagram

14.2 Interface Signals

The external EBU interface signals are listed in **Table 14-1** below.

Table 14-1 EBU Interface Signals

Signal/Pin	Type	Function
AD[31:0]	I/O	Multiplexed Address/Data bus lines 0-31
A[24:16]	O	Address bus lines 16-24
$\overline{\text{CS}}$[3:0]	O	Chip select 0-3
$\overline{\text{RD}}$	O	Read control line
RD/$\overline{\text{WR}}$	O	Write control line
$\overline{\text{ADV}}$	O	Address valid output
$\overline{\text{BC}}$[3:0]	O	Byte control lines 0-3
BFCLKO	O	Clock Output for Synchronous Accesses
BFCLKI	I	Feedback clock input for Synchronous Accesses
SDCLKO	O	Clock Output for SDRAM Accesses
SDCLKI	I	Feedback Clock for SDRAM Clock Output
CKE	O	Clock Enable Output for SDRAM
$\overline{\text{RAS}}$	O	Row Address Strobe for SDRAM
$\overline{\text{CAS}}$	O	Column Address Strobe for SDRAM
$\overline{\text{WAIT}}$	I	Wait input
$\overline{\text{HOLD}}$	I	Hold request input
$\overline{\text{HLDA}}$	I/O	Hold acknowledge
$\overline{\text{BREQ}}$	O	Bus request output

14.2.1 Address/Data Bus, AD[31:0]

This bus transfers address and data information. The width of this bus is 32 bits. External devices with 8, 16 or 32 bits of data width can be connected to the data bus. Burst Mode instruction fetches can be performed with only 32 or 16 bit data bus width. 8 bit devices have to be treated as 16 bit devices and data alignment accomplished using software.

The EBU adjusts the data on the data bus to the width of the external device, according to the programmed parameters in its control registers. The byte control signals $\overline{\text{BC}}$[3:0] specify which parts of the data bus carry valid data.

14.2.2 Address Bus, A[24:16]

The total address bus of the EBU consists of 25 address output lines, giving a directly addressable range of up to 128 Mbyte (2^{25} * 32-bit). A[15:0] will be output on AD[31:16] restricting support for devices using separate address and data buses to those with a 16 bit data bus. An external device can be selected via one of the chip select lines. Since there are four chip select lines, four such devices with up to 512 Mbyte of address range can be used in the external system.

14.2.3 Chip Selects, $\overline{\text{CS}}$[3:0]

The EBU provides up to four chip select outputs, $\overline{\text{CS0}}$, $\overline{\text{CS1}}$, $\overline{\text{CS2}}$ and $\overline{\text{CS3}}$. The address range for each chip select is fixed. More details are described on **Section 14.7.2**.

14.2.4 Read/Write Control Lines, RD, RD/$\overline{\text{WR}}$

Two lines are provided to trigger the read ($\overline{\text{RD}}$) and write (RD/$\overline{\text{WR}}$) operations of external devices. While some read/write devices require both signals, there are devices with only one control input. The RD/$\overline{\text{WR}}$ line is then used for these devices. This line will go to an active-low level on a write, and will stay inactive high on a read. The external device should only evaluate this signal in conjunction with an active chip select. Thus, an active chip select in combination with a high level on the RD/$\overline{\text{WR}}$ line indicates a read access to this device.

14.2.5 Address Valid, $\overline{\text{ADV}}$

The address valid signal, $\overline{\text{ADV}}$, validates the address lines A[23:0] (and also the address placed on the data bus AD[31:0] when attaching multiplexed address/data devices). It can be used to latch these addresses externally.

14.2.6 Byte Controls, $\overline{\text{BC}}$[3:0]

The byte control signals $\overline{\text{BC}}$[3:0] select the appropriate byte lanes of the data bus for both read and write accesses. **Table 14-2** shows the activation on access to a 16-bit or 8-bit external device. Please note that this scheme supports little-endian devices.

Table 14-2 Byte Control Pin Usage

Width of External Device	$\overline{\text{BC3}}$	$\overline{\text{BC2}}$	$\overline{\text{BC1}}$	$\overline{\text{BC0}}$
32-bit device with byte write capability	D[31:24]	D[23:16]	D[15:8]	D[7:0]

Table 14-2 Byte Control Pin Usage (cont'd)

Width of External Device	BC3	BC2	BC1	BC0
16-bit device with byte write capability	inactive (high)	inactive (high)	D[15:8]	D[7:0]
8-bit device	inactive (high)	inactive (high)	inactive (high)	D[7:0]

Signals BCx can be programmed for different timing. The available modes cover a wide range of external devices, such as RAM with separate byte write-enable signals, and RAM with separate byte chip select signals. This allows external devices to connect without any external "glue" logic.

Table 14-3 Byte Control Signal Timing Options

Mode	BUSCONx.BCGEN	Description
Chip Select Mode	00_B	BCx signals have the same timing as the generated chip select CS.
Control Mode	01_B	BCx signals have the same timing as the generated control signals RD or RD/WR.
Write Enable Mode	10_B	BCx signals have the same timing as the generated control signal RD/WR.

14.2.7 Burst Flash Clock Output/Input, BFCLKO/BFCLKI

The flash clock output signal of the EBU is provided at pin BFCLKO. It is used for timing purposes (timing reference) during Burst Mode accesses. BFCLKO is, by default, only generated during synchronous accesses.

The clock input BFCLKI of the EBU is used to latch read data into the EBU. Normally BFCLKI is directly fed back and connected to BFCLKO. This feedback path can be configured externally to maximize the operating frequency for a given Flash device or to compensate the BFCLKO clock pad delay. More details are given on **Section 14.12.4**.

14.2.8 Wait Input, WAIT

This is an input signal to the EBU that is used to dynamically insert wait states into read or write data cycles controlled by the device on the external bus.

14.2.9 SDRAM Clock Output/Input SDCLKO/SDCLKI

The EBU provides a clock output for SDRAM devices on the SDCLKO pin. SDCLKO is, by default, a continuously running signal but can also be configured to switch off between accesses to conserve power.

The feedback clock input, SDCLKI, is used as a timing reference for the capture of read data on SDRAM accesses. It should be connected via a PCB trace to the clock pin of the SDRAM device.

14.2.10 SDRAM Control Signals, CKE, $\overline{\text{CAS}}$ and $\overline{\text{RAS}}$

These signals implement, along with the RD/$\overline{\text{WR}}$ signal, the command interface for an attached SDRAM memory device.

14.2.11 Bus Arbitration Signals, $\overline{\text{HOLD}}$, $\overline{\text{HLDA}}$, and $\overline{\text{BREQ}}$

The $\overline{\text{HOLD}}$ input signal is the external bus arbitration signal that indicates to the EBU when an external bus master requests to obtain ownership of the external bus.

The $\overline{\text{HLDA}}$ is used as input or output depending on the arbitration mode. With this signal the bus master informs the participant that ownership of the external bus has been obtained.

The $\overline{\text{BREQ}}$ output signal is the external bus arbitration signal that is asserted by the EBU when it requests to obtain back the ownership of the external bus.

14.2.12 EBU Reset

The EBU is asynchronously initialized by the system reset.

14.2.12.1 Allocation of Unused Signals as GPIO

The EBU will allow pins not required for its programmed configuration to be allocated for use as GPIO. The signals required are as defined below:

Table 14-4 EBU Interface Signals Required by Operating Mode

Signal/Pin	When Needed by EBU
AD[15:0]	Always needed when the EBU is enabled. MODCON.ARBMODE != 00_B [1)]
$\overline{\text{RD}}$	
RD/$\overline{\text{WR}}$	
$\overline{\text{BC}}$[1:0]	

Table 14-4 EBU Interface Signals Required by Operating Mode (cont'd)

Signal/Pin	When Needed by EBU
$\overline{\text{WAIT}}$	This signal is required by the EBU when any enable region is configured to use WAIT by setting the BUSCONx.WAIT field to a value other than 00_B (MODCON.ARBMODE != 00_B) AND (ADDRSELx.REGENAB=1_B OR ADDRSELx.ALTENAB=1_B) AND (BUSRCONx.WAIT != 00_B)
A[24:16]	These address bits can be individually enabled for use as GPIO by setting the relevant enable bit in the USERCON register. Setting MODCON.ARBMODE = 00_B will also enable for GPIO.
$\overline{\text{ADV}}$	The ADV output can be made available for GPIO by setting the relevant bit in the USERCON register. Setting MODCON.ARBMODE = 00_B will also enable for GPIO.
BFCLKO BFCLKI	These signals are required by the EBU when the EBU is enabled and an enabled region is configured for burst device support BFCLKI can be driven from BFCLKO when configuring the chip internal feedback in the PORTS (MODCON.ARBMODE != 00_B) AND (ADDRSELx.REGENAB=1_B OR ADDRSELx.ALTENAB=1_B) AND (BUSRCONx.AGEN = 1_H OR 3_H OR 5_H OR 7_H)
$\overline{\text{RAS}}$ $\overline{\text{CAS}}$ CKE SDCLKO SDCLKI	These signals are required by the EBU when the EBU is enabled and an enabled region is configured for SDRAM support (MODCON.ARBMODE != 00_B) AND ((ADDRSELx.REGENAB=1_B OR ADDRSELx.ALTENAB=1_B) AND (BUSRCONx.AGEN = 1000_B)
$\overline{\text{HOLD}}$ $\overline{\text{BREQ}}$ $\overline{\text{HLDA}}$	These signals are required by the EBU when the EBU is configured to arbitrate for the external bus. e.g. MODCON.ARBMODE = 01_B or 10_B

Table 14-4 EBU Interface Signals Required by Operating Mode (cont'd)

Signal/Pin	When Needed by EBU
AD[31:16]	These signals are required by the EBU when the EBU is enabled and an enabled region is configured for accessing 32 bit memory or a non-muxed memory type (MODCON.ARBMODE != 00_B) AND (((ADDRSELx.REGENAB=1_B OR ADDRSELx.ALTENAB=1_B) AND (BUSRCONx.PORTW = 10_B or 11_B) OR (BUSCONx.AGEN = non muxed device type)) Availability of pins AD[31:30] as GPIO is controlled directly by the PORTS module. These pins can be used as GPIO.
$\overline{BC}$[3:2]	
$\overline{CS}$[3:0]	The Chip select lines are individually controlled and will be required for EBU operation when the associated memory region is enabled. (MODCON.ARBMODE != 00_B) AND (ADDRSELx.REGENAB=1_B OR ADDRSELx.ALTENAB=1_B)

1) If the EBU is disabled by writing 00 to the EBUCON.ARBMODE field, there will be a delay before the signals become available for GPIO usage as the EBU will wait for all pending external memory accesses to be completed and the arbitration logic to return to the "nobus" state before releasing the signals.

14.3 External Bus States when EBU inactive

The state of the various bus signals is controlled by the EBU during inactive states as listed below. Note that in the XMC4500 the PORTS unit disables the EBU port control lines during reset. The lines must be explicitly enabled by the user software.

Table 14-5 Memory Controller External Bus pin states during reset

Pin Name	State during Reset and "no bus" mode[1)]	State during Idle[2)]
AD(31:0)	GPIO	High Impedance - pull ups enabled to pull to '1'.
A(24:16)	GPIO	Driven to '0' after reset, otherwise last used address
$\overline{CS}$(3:0)	GPIO	Driven to '1' (High).
RD	GPIO	Driven to '1' (High).
WR	GPIO	Driven to '1' (High).
CAS	GPIO	Driven to '1' (High).
RAS	GPIO	Driven to '1' (High).

Table 14-5 Memory Controller External Bus pin states during reset (cont'd)

Pin Name	State during Reset and "no bus" mode[1)]	State during Idle[2)]
CKE	GPIO	Dependant on SDRAM clocking/power save mode
ADV	GPIO	Driven to '1' (High).
SDCLKO	GPIO	Dependant on SDRAM clocking/power save mode
SDCLKI	GPIO	High Impedance
BFCLKI	GPIO	High Impedance
BFCLKO	GPIO	Driven to '0' (Low).
$\overline{BC}$(3:0)	GPIO	Driven to '1' (High).
$\overline{WAIT}$	GPIO	Always an input (must have a pull-resistor to inactive state).

1) GPIO controlled pins should be high impedance with pull up during reset, except for CKE which should be high impedance with pull down.

2) Assuming that the pins have not been made available as GPIO.

Applicable reset is cgu_con_clk_rst_n_i.

14.4 Memory Controller Structure

The AHBIF bridges translate AHB transactions into appropriate transaction requests which can be transferred to the arbiter (see **Section 14.5**).

The arbiter looks at the transaction requests and schedules corresponding requests to the relevant state machine. Only one state machine can be active at any time. The dedicated state machines are used to sequence control signals and to co-ordinate accesses to each of the different external memory/device types and also the internal registers.

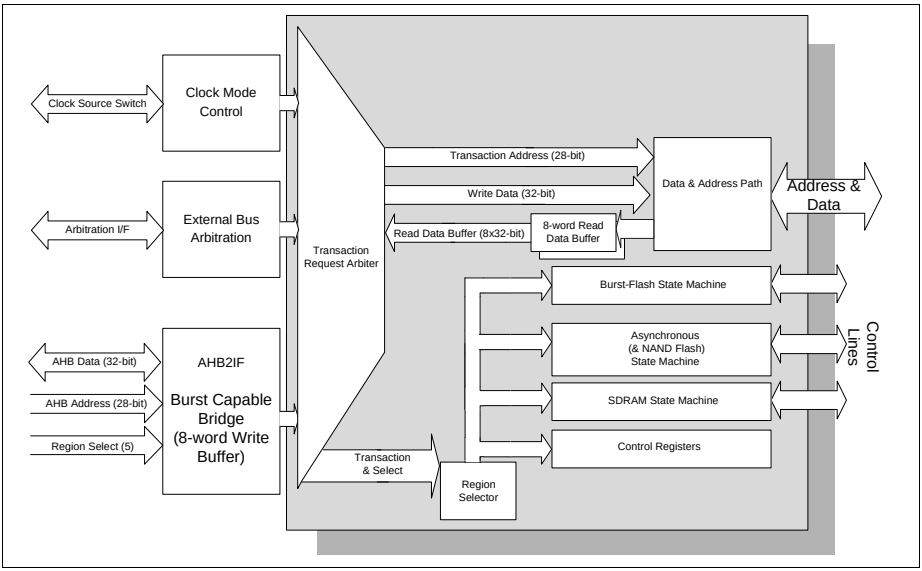


Figure 14-3 Memory Controller Block Diagram

14.5 Memory Controller AHBIF Bridge

As shown in the Memory Controller Block Diagram (**Figure 14-3**) Memory Controller contains an EBU specific AHBIF bridge. This bridge supports the AHB-lite protocol which means (among other things) that masters are not allowed to perform early burst termination (except after an error response) and that retries are not generated by slaves.

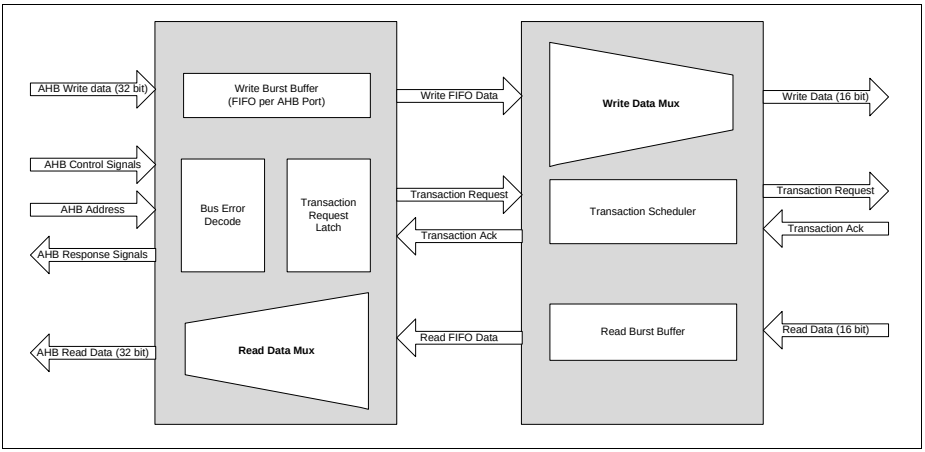


Figure 14-4 AHB Bridge Block Diagram

The bridges map the AHB accesses into appropriate external memory/device transaction requests. Only a limited subset of AHB transactions are supported optimally in order to simplify and reduce the area of the design. All other AHB transactions are split into single data phases and passed to the core logic as multiple accesses.

Table 14-6 Supported AHB Transactions

AHB Transfer Type			Support[1)]
Transfer Size	**Comment**	**Type**	
Byte (8 bits)	Aligned to any byte address	single	Yes
		others	No (split)
Half-Word (16-bits)	Aligned to any half-word address	single	Yes
		others	No (split)
Word (32-bits)	Aligned to any word address	single	Yes
		incr4	Yes (if address aligned, otherwise split))
		incr8	Yes (if address aligned, otherwise split))
		wrap8	Yes
		others	No (split)

1) Unsupported transactions are split into multiple, 32 bit data transfers

The bridge contains a “Write Burst Buffer” which allows the bridge to accept a complete AHB Write Access prior to generation of the matching Write Transaction Request.

The bridge can be operated either Synchronously or Asynchronously. Support is provided for dynamic switching of clocking modes (see **Section 14.6.1**).

The AHB port is configured with a 8 x 32-bit word write buffer and supports all AHB transactions.

Byte and half-word transfers are supported for all transactions from external memory shown in the **Table 14-6**. Byte accesses may be aligned to any byte boundary. Half-word accesses may be aligned to any half-word boundary.

The Memory Controller will ensure that the limitations of external memories are transparent to the AHB interfaces. If an AHB request cannot be directly mapped to an access supported by the external memory, the Memory Controller will split and realign the access into transfers supportable by the external memory. The most noticeable effects of this are:

1. All burst accesses to an external memory are realigned so that the lowest address is fetched first (unless specifically disabled using BUSRCON[3:0].dba)
   This prevents unexpected interaction between AHB access wrapping and any wrapping built into external memories such as SDRAM or burst flash.
2. An AHB, burst access to an asynchronous memory such as SRAM will result in multiple accesses on the external bus as the Memory Controller fetches enough single words from the memory to complete the burst

14.5.1 AHB Error Generation

The bridge generates AHB Error Events as appropriate. There are several types of AHB Error Events which can be generated by Memory Controller. These errors are:-

1. **Access to disabled region:** If an access is attempted to a memory region which is disabled then an AHB Error Response is returned (i.e. the AHB access is terminated with an error). Note that the region can be disabled for reads and/or writes separately.
2. **Illegal Register Access:** Register accesses must ONLY be 32-bit accesses. Write accesses must be performed with HPROT in privilege mode. Any write accesses attempted in user mode, or any type of access attempted as either an 8-bit or 16-bit transfer, or a burst will be terminated with an AHB Error Response.

Once a phase of a transaction has been errored, the initiating master is allowed to terminate the burst without completing the remaining phases. This is the only case in which the memory controller supports early burst termination.

14.5.2 Read Data Buffering

The memory controller contains two, identical read buffers with a capacity of eight 32-bit words. A read access will be allowed to proceed if one of the buffers is flagged as available. The data read from the external memory will be stored in the read buffer and

the outputs from the read buffer will be multiplexed to the AHB port. Once the AHB port signals that all data has been returned to the requesting AHB master, the read buffer will again be flagged as available.

This architecture allows reads to be in progress simultaneously, as a second read can be running while the first read is still waiting for data to be returned to the AHB master.

14.5.3 Write Data Buffering

The data for all AHB writes are “posted” into a buffer in the AHB interface before the access is passed to the state machine blocks for execution.

14.6 Clocking Strategy and Local Clock Generation

The Memory Controller can be configured to operate from several possible clock sources. The clock generation logic is used to select between these clock sources and generate the internal clock used for the memory controller, EBU_CLK.

14.6.1 Clocking Modes

The bridges can be operated in one of three modes:-

- Asynchronous: The AHB clock and Memory Controller internal clock (EBU_CLK) are assumed to be asynchronous. EBU_CLK is derived from a dedicated clock source
- Synchronous: The EBU_CLK is derived from the AHB bus clock . The CLK and AHB interface clocks have aligned edges (although pulse swallowing can be used on the AHB interface clock, so that the AHB interfaces run at the same speed as the rest of the bus matrix).
- Divide by 2: The EBU clock is running at half the frequency of the AHB bus clock . The EBU clock is edge aligned with the processor and AHB interface clock.

The clock for the AHB interface of the memory controller must always be derived from the same synchronous source (the AHB bus clock).

Operation of the bridge in Asynchronous mode provides maximum flexibility in clocking different domains at different frequencies. This is, however, at the cost of additional latency (for signal resynchronisation) through the bridge.

Operation of the bridge in synchronous mode minimizes the latency through the bridge at the cost of forcing the AHB and EBU_CLK domains to run from the same source clock (the AHB bus clock).

The mode of operation of the bridge is controlled by the **EBUCLC.SYNC** register field. The **EBUCLC.SYNCACK** field will be updated to report the current operating mode.

The **CLC.SYNC** register field can be used (dynamically) to switch between these two modes. The Memory Controller contains internal control logic to sequence the switching between modes to ensure that external bus accesses are not corrupted as a result of switching clocking modes. The Memory Controller updates the **CLC.SYNCACK** to signal

the status of the bridge (‘1’ = operating in Synchronous Mode, ‘0’ = operating in Asynchronous Mode).

If **CLC.DIV2** is set to 0_B, then EBU.CLC.SYNC also controls the clock input and switches it between the AHB bus clock and the dedicated EBU clock source from the Clock Generation Unit.

However, if **EBUCLC.DIV2** is set to 1_B, then EBU clock source is forced to be half the frequency of the AHB bus clock and **EBUCLC.SYNC** only enables and disables the resynchronisation stages necessary for asynchronous operation. Setting **EBUCLC.SYNC** to 0_B will only activate the resynchronisation stages, the EBU clock will remain fixed at half the AHB bus clock frequency. The value of **DIV2** in use by the memory controller is stored in the **EBUCLC.DIV2ACK** field.

Local Clock Divider

A local divider can be used to reduce the frequency of EBU_CLK. The divider can be programmed to for divide ratios of 1:1, 2:1, 3:1 or 4:1 using the **EBUCLC.DIV** register field. The ratio currently in use is provided in the **EBUCLC.DIVACK** register field.

The purposes of the divider is to allow the memory controller core to operate synchronously at an integer divide ratio of the AHB bus clock.

If a divide ratio other than 1:1 is selected using the local clock divider and a memory device is being used at a 1:1 external clock ratio, then the device clock outputs BFCLKO and SDCLKO will not have a 50% duty cycle. This is because the local divider operates by pulse swallowing the module input clock to generate EBU_CLK.

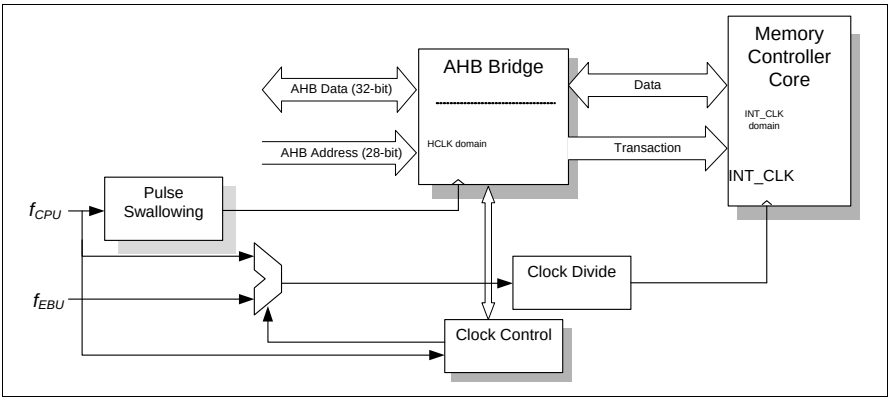


Figure 14-5 AHB/Memory Controller Clocking Domains (Simplified Block Diagram)

14.6.1.1 Clock Requirements

The Memory Controller has the ability to clock SDRAM and other synchronous memory devices at the same frequency as the Memory Controller core logic. In this case, using pulse swallowing on the clock inputs can cause the asymmetric clock waveform generated by pulse swallowing to violate clock pulse width parameters of the connected devices.

This is particularly important when using SDRAM at the maximum supported frequency of the device.

14.6.2 Standby Mode

The Memory Controller can be configured to disable its internal clock and enter standby mode by writing a logic ‘1’ to the EBUCLC.DISR register field.

Once the register field is written, the Memory Controller will wait for any running accesses to finish and will then disable the clock to the core logic of the Memory Controller (EBU_CLK). As this will also disable the refresh counters, it is necessary to set the SDRMREF.AUTOSELFR register field if this mode is to be used with SDRAM. This will instruct the EBU to place any attached SDRAM into self refresh mode before allowing clock mode switching or standby.

Note: For the purposes of standby mode, AHB accesses which are split by the AHB interfaces into multiple accesses to the memory controller count as multiple accesses. This means that the memory controller will attempt to enter standby mode at the end of the current data transfer, not at the end of the final data transfer of the AHB access.

An access arriving on any of the Memory Controller, AHB interfaces will trigger an automatic exit from standby mode to service the access request. This condition may also prevent standby mode being entered at all depending upon when the new access arrives at the AHB interface.

Note: Once a pending AHB access has triggered an exit from standby mode, if all pending AHB accesses have been serviced and the ARM is still in “standby wait for interrupt” mode, the EBU will not return to standby mode.

An automatic exit from standby mode will not clear the EBUCLC.DISR field. This has to be explicitly written with ‘0’ by software before writing another ‘1’ will trigger a further entry to standby mode.

14.7 External Bus Operation

The EBU supports interconnection to a wide variety of memory/peripheral devices with flexible programming of the access parameters. In the following sections, the basic features for these access modes are described. The types of external access cycles provided by the EBU are:

- Asynchronous and synchronous devices with demultiplexed access
  - ROMs, EPROMs
  - NOR flash devices
  - Static RAMs and PSRAMs
- Asynchronous and synchronous devices with multiplexed access
  - NOR flash devices
  - PSRAMs
- SDRAM Memories

Note: Not all memory types supported by the memory controller are known to be available in all quality grades.

Each internal AHB master can access external devices via the EBU. The EBU provides four user-programmable external memory regions. Each of these regions is provided with a set of registers that determine the parameters of the external bus transaction and one chip select signal. An AHB transaction that matches one of these external memory regions is translated by the EBU to the appropriate external access(es).

The address space of each of the four regions and the registers is defined at the system level by the address decoder in the AHB bus matrix.

14.7.1 External Memory Regions

The memory controller of the XMC4500 supports four memory regions, which have its own associated chip select outputs $\overline{CS}$[3:0]. Each of these regions has a set of control registers to specify the type of memory/peripheral device and the access parameters.

Each of the four user-programmable regions (x = 0-3 is the numbering index of these regions) is can be configured to respond to a particular address space through registers ADDRSELx,

The access parameters for each of the regions can be programmed individually to accommodate different types of external devices. Separate control registers are available to control read and write accesses. This allows optimal access types, speeds and parameters to be chosen. Access type is configured via BUSRCONx and BUSWCONx. Access parameters are configured via BUSRAPx and BUWAPx.

Throughout this document the generic term BUSCONx is used when either of BUSRCONx or BUSWCONx is applicable and BUSAPx is used when either of BUSRAPx or BUSWAPx is applicable.

Table 14-7 EBU Address Regions, Registers and Chip Selects

Region	Associated Chip Select	Address Select Registers	Bus Configuration Registers	Bus Access Parameters Registers
Region 0	$\overline{CS0}$	ADDRSEL0	BUSRCON0 BUSWCON0	BUSRAP0 BUSWAP0
Region 1	$\overline{CS1}$	ADDRSEL1	BUSRCON1 BUSWCON1	BUSRAP1 BUSWAP1
Region 2	$\overline{CS2}$	ADDRSEL2	BUSRCON2 BUSWCON2	BUSRAP2 BUSWAP2
Region 3	$\overline{CS3}$	ADDRSEL3	BUSRCON3 BUSWCON3	BUSRAP3 BUSWAP3

Table 14-8 lists the programmable parameters that are available for the four external regions (regions 0 to 3).

Table 14-8 Programmable Parameters of Regions

Register	Parameter (Bit/Bit field)	Function
ADDRSELx	WPROT	Write Protect bit for each region.
	ALTENAB	Alternate segment enable of a region.
	REGENAB	Enable bit for each region.
BUSCONx	AGEN	Region access type: See **Section 14.7.3**

14.7.2 Chip Select Control

The EBU generates four chip select signals, $\overline{CSx}$, which are all available at dedicated chip select outputs.

14.7.3 Programmable Device Types

Each CS region (0 to 3) can be individually configured using the BUSCONx.AGEN register field, to be connected to one of the following external memory/device types:

Table 14-9 AGEN description

AGEN value	Device Type
0	Muxed Asynchronous Type External Memory (default after reset)
1	Muxed Burst Type External Memory

Table 14-9 AGEN description (cont'd)

AGEN value	Device Type
2	NAND flash (page optimized)
3	Muxed Cellular RAM External Memory
4	Demuxed Asynchronous Type External Memory
5	Demuxed Burst Type External Memory
6	Demuxed Page Mode External Memory
7	Demuxed Cellular RAM External Memory
8	SDRAM External Memory
9	reserved
10	reserved
11	reserved
12	reserved
13	reserved
14	reserved
15	reserved

14.7.4 Support for Multiplexed Device Configurations

Memory Controller supports a number of configurations of Multiplexed memory/peripheral devices using different values of the BUSRCONx.PORTW bit-field. The BUSWCONx registers also contain the PORTW field but in this case the field is read only and reflects the value set in the related one of the BUSRCONx registers. The values set in the BUSRCONx registers are used for both read and write accesses.

Note: When using multiplexed devices a non-zero recovery phase is mandatory for all devices to prevent read data from one access conflicting with the address for the multiplexed memory.

Table 14-10 Pins used to connect Multiplexed Devices to Memory Controller

Memory Device Configuration	Memory Controller Pins			Section
	A(24:16)[1)]	AD(31:16)[2)]	AD(15:0)	
16-bit MUX	A(24:16)	-	A(15:0)/ D(15:0)	16-bit Multiplexed Memory/Peripheral Configuration

Table 14-10 Pins used to connect Multiplexed Devices to Memory Controller

Memory Device Configuration	Memory Controller Pins			Section
	A(24:16)[1]	AD(31:16)[2]	AD(15:0)	
Twin 16-bit MUX	A(24:16)	A(15:0)/ D(31:16)	A(15:0)/ D(15:0)	Twin 16-bit Multiplexed Device Configuration
32-bit MUX	-	A(31:16)/ D(31:16)	A(15:0)/ D(15:0)	32-bit Multiplexed Memory/Peripheral Configuration

1) These pins are always outputs which are connected to address pins on the Multiplexed device(s)

2) These pins are dual function and act as AD(31:16) when required for 32-bit, multiplexed devices

Table 14-11 Selection of Multiplexed Device Configuration

PORTW value	
00_B	reserved
01_B	16-bit multiplexed[1]
10_B	Twin, 16-bit Multiplexed[2]
11_B	32 bit multiplexed[3]

1) Address will only be driven onto AD(15:0) during the address and address hold phases. A(15:0) will be driven with address for duration of access

2) Lower 16 bits of address will be driven onto both A(15:0) and AD(15:0) during the address and address hold phases

3) Full address will be driven onto A(15:0) and AD(15:0) during the address and address hold phases

16-bit Multiplexed Memory/Peripheral Configuration

Throughout the complete external bus cycle the address[1] is driven onto Memory Controller pins A(24:16). During the address phase the low 16 bits of the address are driven to Memory Controller pins AD(15:0). Data (16-bit) is driven to/read back from the AD(15:0) pins during the data phase. The interconnect between Memory Controller and a 16-bit Multiplexed device in this mode is shown below (note: for clarity only the address/data signals are shown):-

1) This address is pre-aligned according to the bus width as detailed in **Section 14.7.7**.

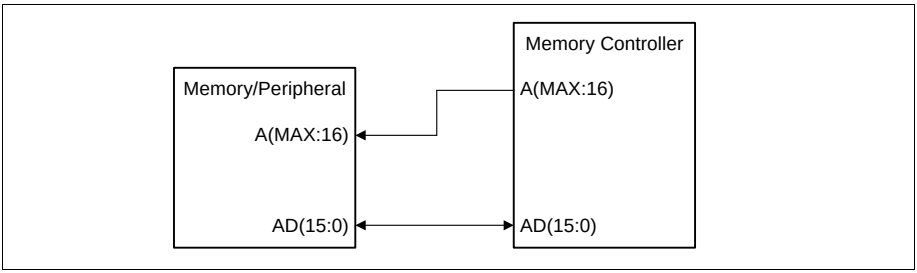


Figure 14-6 Connection of a 16-bit Multiplexed Device to Memory Controller

Twin 16-bit Multiplexed Device Configuration

This mode allows the use of two 16-bit multiplexed devices to create a 32-bit wide bus. Throughout the complete external bus cycle the address[1)] is driven onto Memory Controller pins A(24:0). During the address phase the low 16 bits of the address are driven (in parallel) to Memory Controller pins AD(15:0) and AD(31:16). This ensures that both multiplexed devices are issued with the same address during the address phase. Data (32-bit) is written to/read from the AD(31:16) pins for MSW and the AD(15:0) pins for LSW during the data phase. The interconnect between Memory Controller and two 16-bit Multiplexed devices in this mode is shown below (note: for clarity only the address/data signals are shown):-

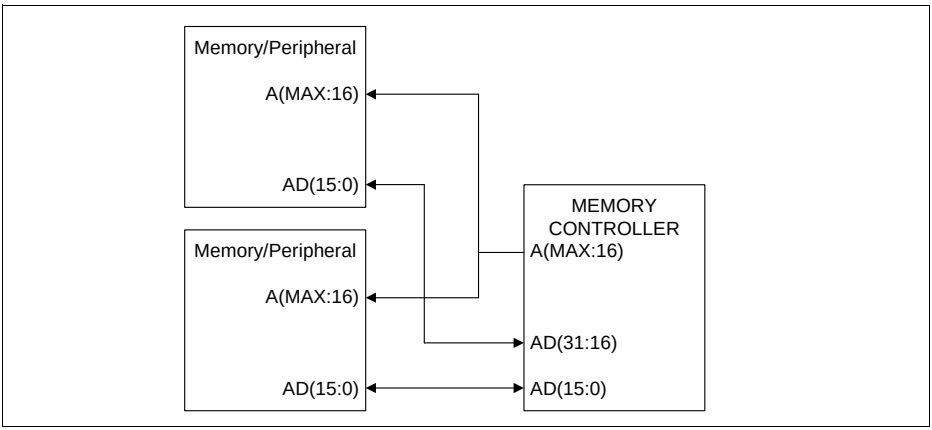


Figure 14-7 Connection of twin 16-bit Multiplexed Device's to Memory Controller

1) This address is pre-aligned according to the bus width as detailed in **Section 14.7.7**.

32-bit Multiplexed Memory/Peripheral Configuration

During the address phase the lower 16 bits of the 25 bit address are driven to Memory Controller pins AD(15:0), the most significant 9 bits of the address are driven to pins AD(24:16) and pins AD(31:17) are driven with 0 (zero). Data (32-bit) is driven to/read from the AD(31:0) pins during the data phase. The interconnect between Memory Controller and a 32-bit Multiplexed device in this mode is shown below (note: for clarity only the address/data signals are shown):-

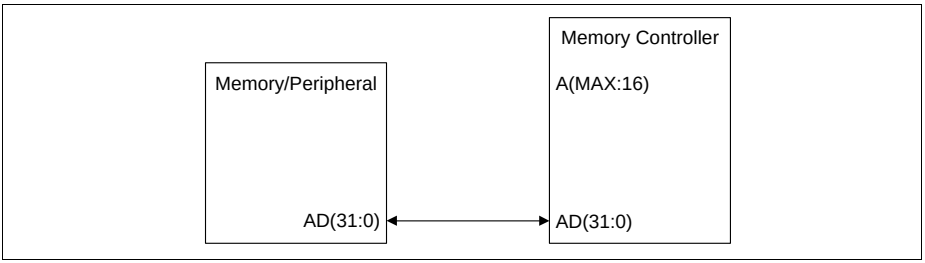


Figure 14-8 Connection of a 32-bit Multiplexed Device to Memory Controller

14.7.5 Support for Non-Multiplexed Device Configurations

The Memory Controller supports 16-bit non-multiplexed memory devices.

Table 14-12 Pins used to connect non-multiplexed Devices to Memory Controller

Memory Device Configuration	Memory Controller Pins			Section
	A(24:16)[1]	AD(31:16)	AD(15:0)	
16-bit non-MUX	A(24:16)	A(15:0)	D(15:0)	16-bit non-Multiplexed Memory/Peripheral Configuration

1) These pins are always outputs which are connected to address pins on the device(s)

Table 14-13 Selection of non-Multiplexed Device Configuration

PORTW value	
00_B	reserved
01_B	16-bit[1]

Table 14-13 Selection of non-Multiplexed Device Configuration (cont'd)

PORTW value	
10_B	reserved
11_B	32-bit - not supported

1) Address will only be driven onto AD(15:0) during the address and address hold phases. AD(31:16) will be driven with address for duration of access

16-bit non-Multiplexed Memory/Peripheral Configuration

Throughout the complete external bus cycle the address[1)] is driven onto Memory Controller pins A(24:16) and AD(15:0). Data (16-bit) is driven to/read back from the AD(15:0) pins during the data phase. The interconnect between Memory Controller and a 16-bit non-Multiplexed device in this mode is shown below (note: for clarity only the address/data signals are shown):-

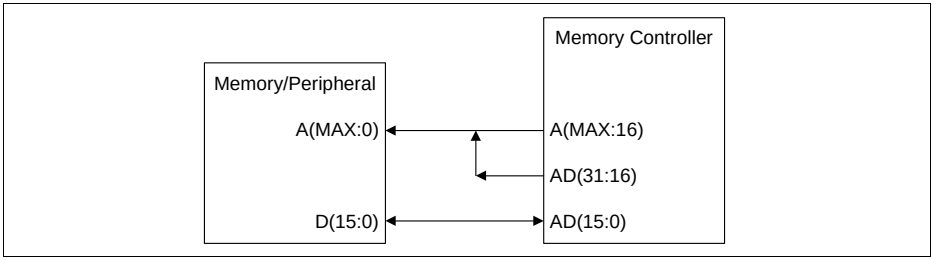


Figure 14-9 Connection of a 16-bit non-Multiplexed Device to Memory Controller

14.7.6 AHB Bus Width Translation

If the internal access width is wider than the external bus width specified for the selected external region, the internal access is split in the EBU into several external accesses. For example, if the AHB requests to read a 64-bit word and the external device is only 16-bit wide, the EBU will automatically perform four external 16-bit accesses. When multiple accesses are generated in this way, external bus arbitration is blocked until the multiple access is complete. This means that the EBU remains the owner of the external bus for the duration of the access sequence. The external accesses are performed in ascending AHB address order.

To allow proper bus width translation, the EBU has the capability to re-align data between the external bus and the AHB as shown in **Figure 14-10**.

1) This address is pre-aligned according to the bus width as detailed in **Section 14.7.7**.

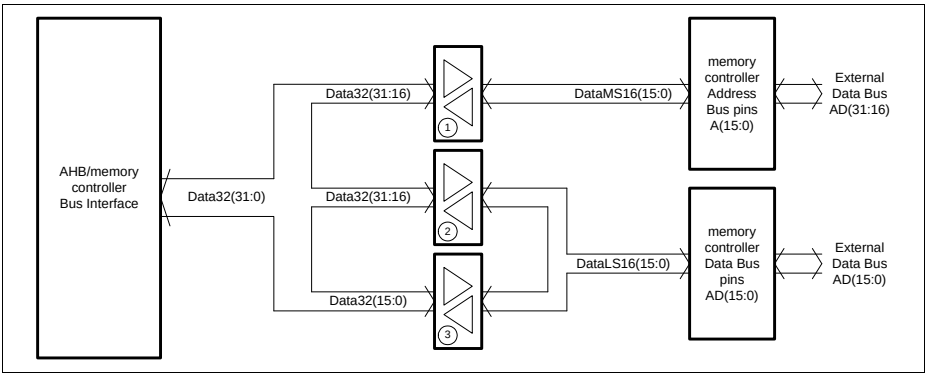


Figure 14-10 AHB to External Bus Data Re-Alignment

- During an access to a 32-bit wide external region, Buffer 1 and Buffer 2 are enabled.
- During an access to a 16-bit wide external region, either Buffer 1 or Buffer 3 is enabled (according to bit 1 of the AHB address being accessed). This allows either AHB channel byte pair (i.e. any properly aligned 16-bit data) to be re-aligned to the lower 16 bits of the external data bus D[15:0].

14.7.7 Address Alignment During Bus Accesses

During an external bus access, the EBU will align the internal byte address to generate the appropriate external word or half-word address aligned to the external address pins. The address alignment will be done as follows:

- For 16 bit memory accesses
  - AD[15:0] will be driven with the value from AHBA[16:1] (multiplexed accesses only)
  - AD[31:16] will be driven with the value from AHBA[15:1] (non-multiplexed accesses only)
  - A[24:16] will be driven with the value from AHBA[24:17]
- For 32 bit memory accesses
  - AD[15:0] and AD[31:16] will be driven with the value from AHBA[17:2] (accesses to paired 16-bit multiplexed devices only)
  - AD[31:0] will be driven with the right-justified and zero-padded value from AHBA[31:2] (accesses to paired 32-bit multiplexed devices only)
  - A[24:16] will be driven with the value from AHBA[25:18]

14.8 External Bus Arbitration

External bus arbitration is provided to allow the EBU to share its external bus with other master devices. This capability allows other external master devices to obtain ownership

of the external bus, and to use the bus to access external devices connected to this bus. The scheme provided by the EBU is compatible with other Infineon microcontroller devices and therefore allows the use of such devices as (external bus) masters together with the XMC4500.

*Note: In this section, the term "external master" is used to denote a device which is located on the **external** bus and is capable of generating accesses across the external bus (i.e. is capable of driving the external bus). An external master is not able to access units that are located inside the XMC4500.*

14.8.1 External Bus Modes

The EBU can operate in two bus modes on the external bus:

- Owner Mode
- Hold Mode

When in Owner Mode, the EBU operates as the master of the external bus. In other words, the EBU drives the external bus as required in order to access devices located on the external bus. While the EBU is in Owner Mode it is not possible for any other master to perform any accesses on the external bus.

During Hold Mode, the EBU tri-states the appropriate signal on the external bus in order to allow another external bus master to perform accesses on the external bus (i.e. to allow another master to drive the various external bus signals without contention with EBU).

14.8.2 Arbitration Signals and Parameters

The arbitration scheme consists of an external bus master that is responsible for controlling the allocation of the external bus. This master is referred to as the "Arbiter" within this document. The other external bus master (termed Participant within this document) requests ownership of the bus, and when necessary, from the Arbiter. The EBU can be programmed to operate either as an Arbiter or as a Participant (see **Section 14.8.3**). The following three lines are used by the EBU to arbitrate the external bus.

Table 14-14 EBU External Bus Arbitration Signals

Signal	Direction	Function
$\overline{\text{HOLD}}$	In	$\overline{\text{HOLD}}$ is asserted (low) by an external bus master when the external bus master requests to obtain ownership of the external bus from the EBU.
$\overline{\text{HLDA}}$	In/Out[1)]	$\overline{\text{HLDA}}$ is asserted (low) by the Arbiter to signal that the external bus is available for use by the Participant (i.e. the bus is not being used by the Arbiter). $\overline{\text{HLDA}}$ is sampled by the Participant to detect when it may use the external bus.
$\overline{\text{BREQ}}$	Out	$\overline{\text{BREQ}}$ is asserted (low) by the EBU when the EBU requests to obtain ownership of the external bus.

1) The direction of this signal depends upon the mode in which the EBU is operating (see **Section 14.8.3**).

Two components that are equipped with the EBU arbitration protocol can be directly connected together (without additional external logic) as shown below:

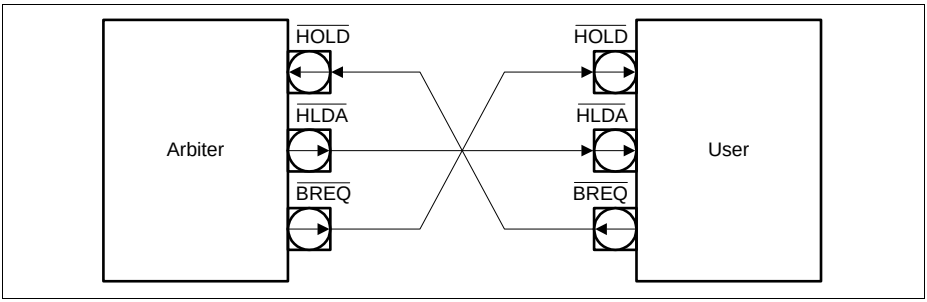


Figure 14-11 Connection of Bus Arbitration Signals

*Note: In the example of **Figure 14-11**, it is possible for the EBU to perform the function or either Arbiter or Participant (or indeed both the Arbiter and Participant may be the EBU).*

Table 14-15 lists the programmable parameters for the external bus arbitration.

Table 14-15 External Bus Arbitration Programmable Parameters

Parameter	Function	Description see
MODCON.ARBMODE	Arbitration mode selection	Section 14.8.3
MODCON.ARBSYNC	Arbitration input signal sampling control	
MODCON.EXTLOCK	External bus ownership locking control	
MODCON.TIMEOUTC	External bus time-out control	

14.8.3 Arbitration Modes

The arbitration mode of the EBU can be selected through configuration pins during reset or by programming the MODCON.ARBMODE bit field (see **Section 14.8.3**) after reset. Four different modes are available:

- No Bus Mode
- Sole Master Mode
- Arbiter Mode
- Participant Mode

14.8.3.1 No Bus Arbitration Mode

All accesses of the EBU to devices on the external bus are prohibited and will generate an AHB bus error. The EBU operates in Hold Mode all the time.

No Bus Mode is selected by MODCON.ARBMODE = 00_B.

14.8.3.2 Sole Master Arbitration Mode

In this mode, the EBU must be the only master on the external bus. Therefore no arbitration is necessary and the EBU has access to the external bus at any time. The EBU operates in Owner Mode all the time.

Sole Master Mode is selected by MODCON.ARBMODE = 11_B.

14.8.3.3 Arbiter Mode Arbitration Mode

The EBU is the default owner of the external bus (e.g. applicable when operating from external memory). Arbitration is performed if an external master (e.g. second CPU) needs to access the external bus.

The EBU is cooperative in relinquishing ownership of the external bus while operating in Arbiter Mode. When the $\overline{\text{HOLD}}$ input is active, the EBU will generate a “retry” in response to any attempt to access the external bus from the internal AHB. However, the EBU is aggressive in regaining ownership of the external bus while operating in Arbiter Mode.

The EBU, having yielded ownership of the bus, will always request return of ownership even if there is no EBU external bus access pending.

Arbiter Mode is selected by MODCON.ARBMODE = 01_B.

Table 14-16 and **Figure 14-12** show the functionality of the arbitration signals in Arbiter Mode.

Table 14-16 Function of Arbitration Pins in Arbiter Mode

Pin	Type	Pin Function in Arbiter Mode
$\overline{\text{HOLD}}$	In	In Owner Mode (EBU is the owner of the external bus), a low level at $\overline{\text{HOLD}}$ indicates a request for bus ownership from the external master. In Hold Mode (EBU is not the owner of the external bus), a high level at $\overline{\text{HOLD}}$ indicates that the external master has relinquished bus ownership, which causes the EBU to exit Hold Mode.
$\overline{\text{HLDA}}$	Out	While $\overline{\text{HLDA}}$ is high, the EBU is operating in Owner Mode. A high-to-low transition indicates that the EBU has entered Hold Mode and that the external bus is available to the external master. While $\overline{\text{HLDA}}$ is low, the EBU is operating in Hold Mode. A low-to-high transition indicates that the EBU has exited Hold Mode, and has retaken ownership of the external bus.
$\overline{\text{BREQ}}$	Out	$\overline{\text{BREQ}}$ is high during normal operation. The EBU drives $\overline{\text{BREQ}}$ low for two EBU clock cycles after entering Hold Mode (after asserting $\overline{\text{HLDA}}$ low). $\overline{\text{BREQ}}$ returns high one clock cycle after the EBU has exited Hold Mode (after driving $\overline{\text{HLDA}}$ high).

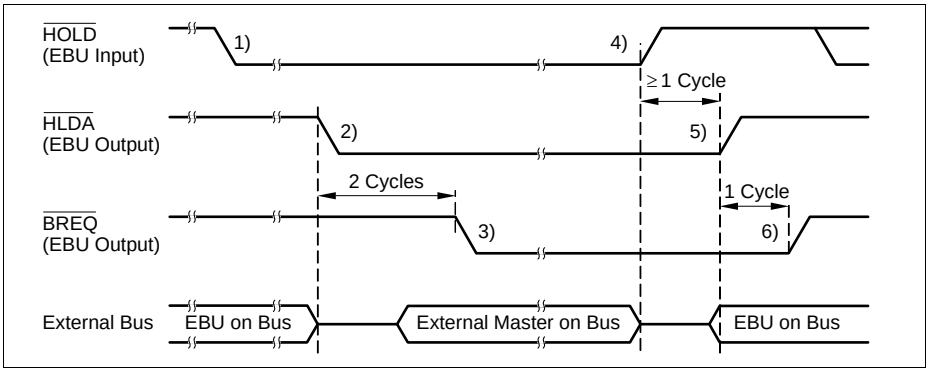


Figure 14-12 Arbitration Sequence with the EBU in Arbiter Mode

In Arbiter Mode, the arbitration sequence starts with the EBU as owner of the external bus.

1. The external master wants to perform an external bus access by asserting a low signal on the HOLD input.
2. When the EBU is able to release bus ownership, it enters Hold Mode by tri-stating its bus interface lines and drives HLDA = 0 to indicate that it has released the bus. At this point, the external master ia allowed to drive the bus.
3. Two clock (EBU_CLK) cycles minimum after issuing HLDA low, the EBU drives BREQ low in order to regain bus ownership. This bus request is issued whether or not the EBU has a pending external bus access. However, the external master will ignore this signal until it has finished its bus access. This scheme assures that the external master can perform at least one complete external bus access.
4. When the external master has completed its access, it tri-states its bus interface and sets HOLD to inactive (high) level to signal that it has released the bus back to the EBU.
5. When the EBU detects that the bus has been released, it returns HLDA to high level and returns to Owner Mode by actively driving the bus interface lines. There is always at least one clock (EBU_CLK) cycle delay from the release of the HOLD input to the EBU driving the bus.
6. Finally, the EBU deactivates the BREQ signal a minimum of one clock (EBU_CLK) cycle after deactivation of HLDA. Now (and not earlier) the external master can generate a new hold request to the EBU.

This sequence assures that the EBU can perform at least one complete bus cycle before it re-enters Hold Mode as a result of a request from the external master.

The conditions that cause change of bus ownership when the EBU is operating in Arbiter Mode are shown in **Figure 14-13**.

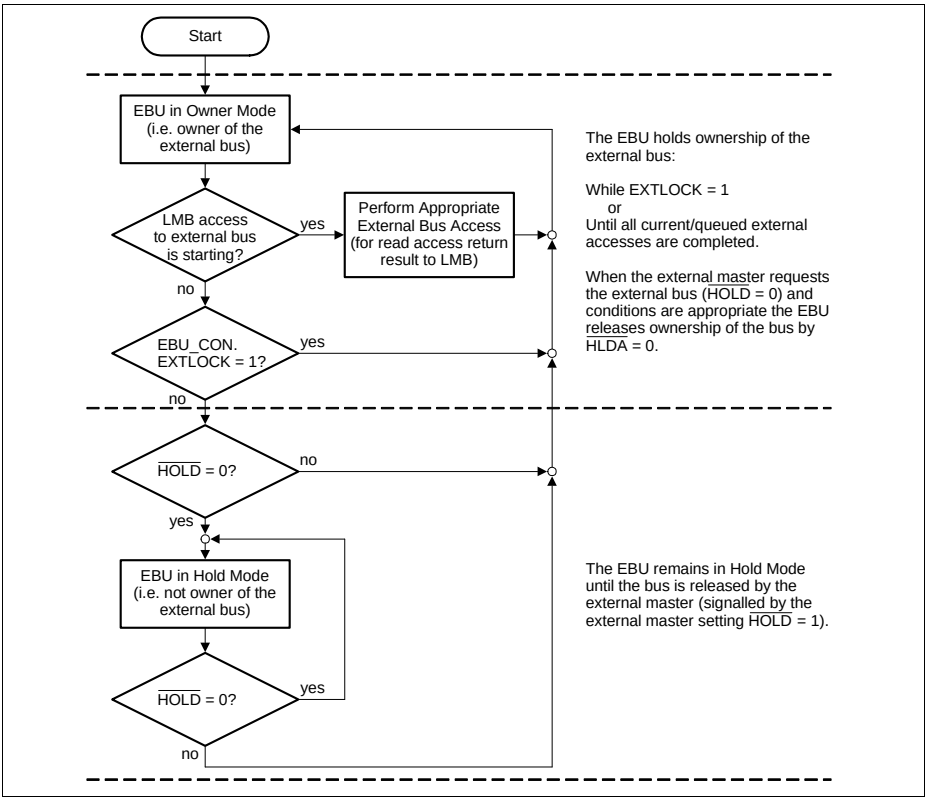


Figure 14-13 Bus Ownership Control in Arbiter Mode

14.8.3.4 “Participant Mode” Arbitration Mode

The EBU tries to gain bus ownership only in case of pending transfers (e.g. when operating from internal memory and performing stores to external memory). While the EBU is not the owner of the external bus (default state), any AHB access to the external bus will be issued with a retry by the EBU. Any such access will, however, cause the EBU to arbitrate for ownership of the external bus.

Once the EBU has gained ownership of the external bus, it will wait until either the occurrence of an external bus access (e.g. the repeat of the request that originally caused the arbitration to occur) or for a programmable time-out (see **Section 14.8.7**). Once the first access has been completed, the EBU will continue to accept requests from the AHB bus until the external master asserts $\overline{HOLD}$ = 0. After the external master has asserted $\overline{HOLD}$ = 0, the EBU will respond to subsequent AHB accesses to external memory with a retry, and will return ownership of the bus to the external master once any ongoing transaction is complete.

Note: Regardless of the state of the $\overline{HOLD}$ input, the EBU will always perform at least one external bus access (assumed that there is not a time-out) before returning ownership of the bus to the external master.

The use of the arbitration signals in Participant Mode is:

Table 14-17 Function of Arbitration Pins in Participant Mode

Pin	Type	Pin Function in Participant Mode
$\overline{HOLD}$	In	When the EBU is not in Hold Mode ($\overline{HLDA}$ = 0) and has completely taken over control of the external bus, a low level at $\overline{HOLD}$ requests the EBU to return to Hold Mode.
$\overline{HLDA}$	In	When the $\overline{HLDA}$ signal is high, the EBU is in Hold Mode. When the EBU has requested ownership of the bus by a high-to-low transition at $\overline{HLDA}$, the EBU is released from Hold Mode.
$\overline{BREQ}$	Out	$\overline{BREQ}$ remains high as long as the EBU does not need to access the external bus. When the EBU detects that an external access is required, it sets $\overline{BREQ}$ = 0 and waits for signal $\overline{HLDA}$ to become low. When the EBU has completed the external bus access (and has re-entered Hold Mode), it will set $\overline{BREQ}$ = 1 to signal that it has relinquished ownership of the external bus.

Participant Mode arbitration mode is selected by CON.ARBMODE = 10_B.

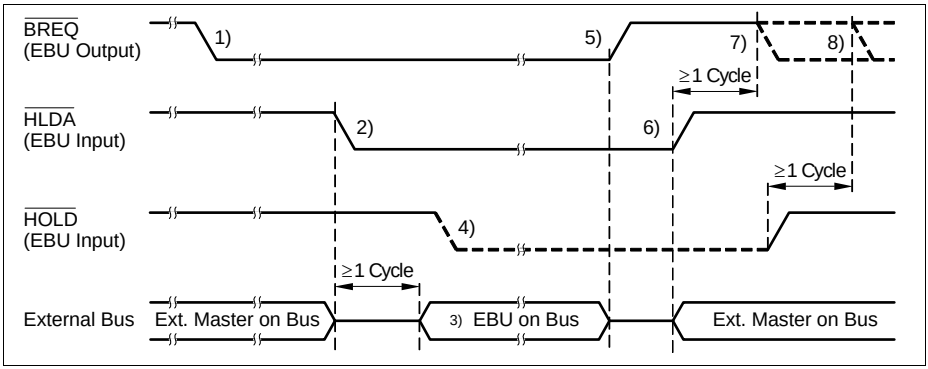


Figure 14-14 Arbitration Sequence with the EBU in Participant Mode

In Participant Mode, the arbitration sequence starts with the EBU in Hold Mode.

1. The EBU detects that it has to perform an external bus access by asserting a low signal on the $\overline{\text{BREQ}}$ output.
2. When the external master is able to release bus ownership, the external master releases the external bus by tri-stating its bus interface lines and drives the $\overline{\text{HLDA}}$ = 0.
3. At least one clock (EBU_CLK) cycle after detecting $\overline{\text{HLDA}}$ = 0, the EBU will start to drive the external bus.
4. When the EBU is in Owner Mode, the external master may optionally drive $\overline{\text{HOLD}}$ = 0 to signal that it wants to regain ownership of the external bus.
5. When the criteria are met for the EBU to release the bus ownership, the EBU enters Hold Mode and drives $\overline{\text{BREQ}}$ = 1 output high to signal that it has released the bus.
6. When the external master detects that the EBU has released the bus ($\overline{\text{BREQ}}$ = 1), it returns $\overline{\text{HLDA}}$ to high level and takes ownership of the external bus.
7. The EBU will not request ownership of the external bus again ($\overline{\text{BREQ}}$ = 0) at least one clock (EBU_CLK) cycle after $\overline{\text{HLDA}}$ has been driven high).
8. In Owner Mode, the EBU will not request ownership of the external bus $\overline{\text{BREQ}}$ = 0) for at least one clock (EBU_CLK) cycle after its $\overline{\text{HOLD}}$ input has been driven high.

The conditions that cause change of bus ownership when the EBU is operating in Participant Mode is shown in **Figure 14-13**.

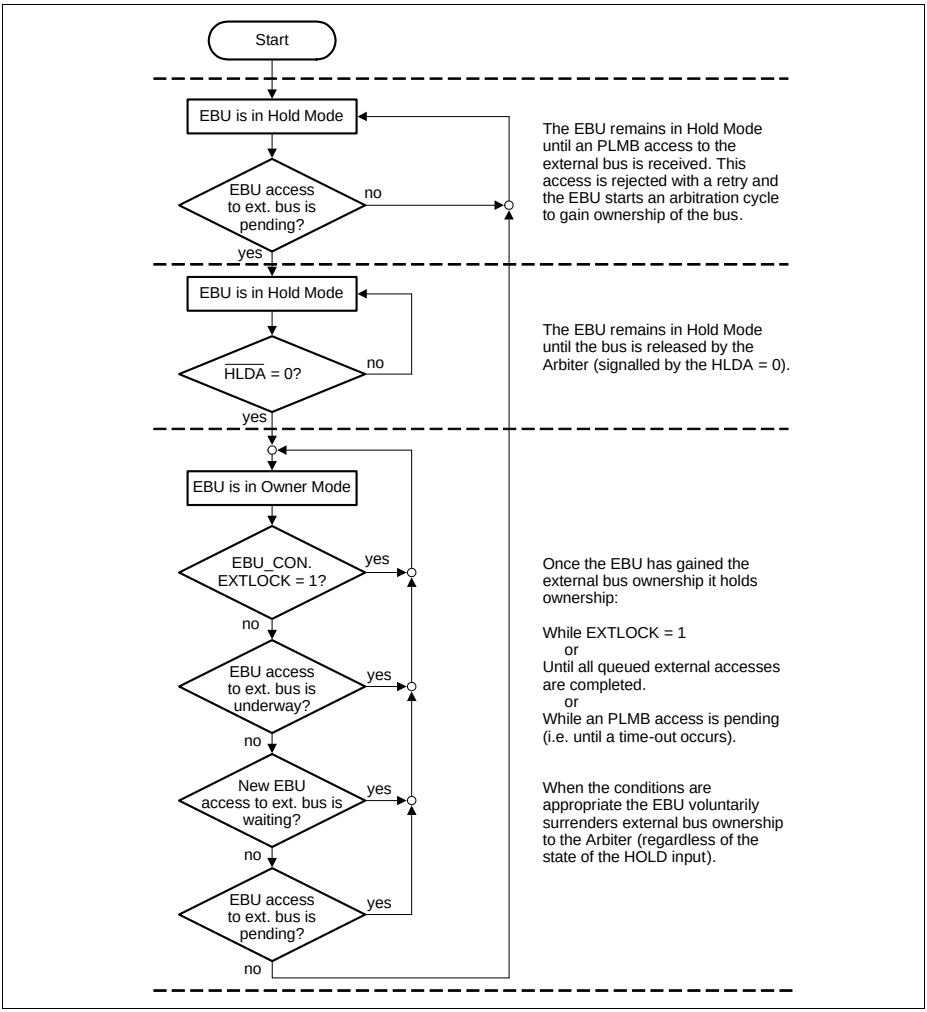


Figure 14-15 Bus Ownership Control with the EBU in Participant Mode

14.8.4 Arbitration Input Signal Sampling

The sampling of the arbitration inputs can be programmed for two modes:

- Synchronous Arbitration
- Asynchronous Arbitration

When synchronous arbitration signal sampling is selected (ARBSYNC = 0), the arbitration input signals are sampled and evaluated in the same clock cycle. This mode provides the least overhead during arbitration (i.e. when changing bus ownership). The disadvantage is that the input signals must adhere to setup and hold times with respect to EBU_CLK to prevent the propagation of meta-stable signals in the EBU.

When asynchronous arbitration signal sampling is selected (ARBSYNC = 1), the arbitration signals are sampled and then fed to an additional latch to be evaluated in the cycle following that in which they were sampled. This provides the EBU with good immunity to signals changing state at or around the time at which they are sampled. The disadvantage is the introduction of additional latency during arbitration (i.e. when changing bus ownership).

14.8.5 Locking the External Bus

The external bus can be locked to allow the EBU to perform uninterrupted sequences of external bus accesses. The EBU allows two methods of locking the external bus:

- Locked AHB accesses
- Lock bit EXTLOCK

When the EBU has ownership of the external bus and is performing external bus accesses in response to a locked AHB access sequence, the ownership of the external bus will not be relinquished until the locked AHB access sequence has been completed.

When lock bit EXTLOCK = 1, the EBU will hold the ownership of the external bus until EXTLOCK is subsequently cleared. If EXTLOCK is written to 1 while the EBU is the owner of the external bus, the EBU is immediately prevented from responding to requests for the external bus until EXTLOCK is cleared[1]. If EXTLOCK is written to 1 while the EBU is not the owner of the external bus, the EBU will immediately attempt to gain ownership. When the EBU gains the ownership of the external bus the next time, the external master is prevented from regaining ownership of the external bus until EXTLOCK is again cleared.

Note: There is no time-out mechanism available for the EXTLOCK bit. When the EBU is owner of the external bus with EXTLOCK bit set, the external master will remain locked off the bus until the EXTLOCK bit is cleared by software.

1) Requests for the external bus already pending when EXTLOCK is set will not be cancelled so the EBU can give up control of the external bus after EXTLOCK is set provided that the request occurs before the EXTLOCK bit is set.

14.8.6 Reaction to an AHB Access to the External Bus

The reaction of the memory controller to an external bus request from an AHB master is controlled as shown in **Figure 14-16**.

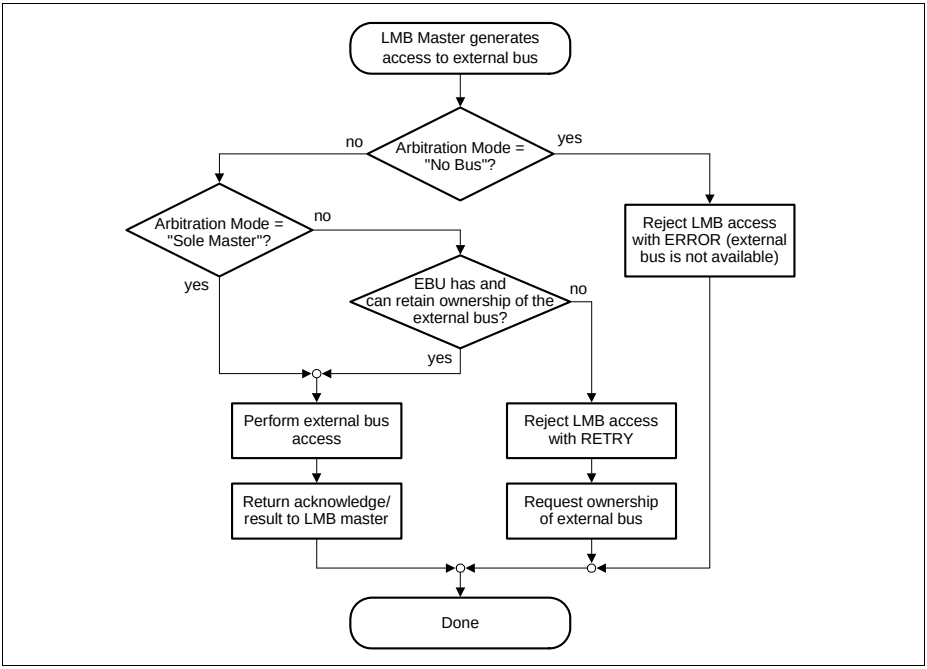


Figure 14-16 EBU Reaction to AHB to External Bus Access

If the EBU is operating in No Bus Mode, it is not possible for an AHB master to access the external bus. For this reason, the EBU generates an AHB error whenever an attempt is made to access the external bus while in No Bus Mode.

If the EBU is operating in Sole Master Mode, it has access to the external bus at all times and as a result it is possible for the EBU to immediately perform the required external bus access.

If the EBU is operating in Arbiter or Participant Modes and receives a request for an external access from an AHB when it is not the owner of the external bus (or is not able to retain ownership of the bus), the request is stalled by taking HRDY low. As shown in **Figure 14-16**, this event also triggers the EBU to arbitrate with the external master in order to attempt to gain ownership of the external bus so that the request can be serviced.

14.8.7 Pending Access Time-Out

The strategy of stalling an AHB access (when the EBU is not the owner of the external bus) as described in the previous section may result in the occurrence of a time-out condition.

To avoid a bus-locking condition, the EBU contains a time-out mechanism. When the EBU has gained ownership of the external bus, it will retain ownership only until a AHB-to-external bus access occurs or a programmable number of EBU_CLK clock cycles has elapsed. If one of these conditions has occurred, the pending access is cancelled and the EBU will continue to arbitrate the external bus in the normal fashion. The desired time-out time (number of EBU_CLK cycles) is programmed using bit field TIMEOUTC. The time-out value can be in the range 1×8 up to 255×8 EBU_CLK clock cycles.

14.8.8 Arbitrating SDRAM control signals

Normally, the memory controller will not surrender control of a connected SDRAM device when arbitrating the external bus. This is because the memory controller needs to keep track of which pages are open in the SDRAM and also because of restrictions on the SDRAM clock.

However, the memory controller can be programmed to tri-state the SDRAM control signals SDCLKO, CKE, $\overline{RAS}$ and $\overline{CAS}$ by setting the MODCON.SDTRI bit to 1_B. When this bit is set, SDRAM can be shared with another controller provided certain conditions are met by both memory controllers.

- The SDRAM must be in self refresh mode with the clock safely stopped before ownership of the external bus is transferred. This ensures that all pages in the SDRAM are closed and that the CKE signal is at logic zero. This can be achieved for the memory controller by setting the SDRMREF.AUTOSELFR to 1_B.
- The SDRAM CKE input must have a pull down sufficient to ensure that there is a guaranteed logic zero on the input while bus ownership is being transferred.

14.9 Start-Up/Boot Process

The EBU pins will be held in a tri-state condition while the memory controller is in reset.

After reset is removed, the EBU will configure itself for ”no bus mode” and wait for configuration by software. The PORTS logic also needs to be configured by software to allow the EBU to control its related pins.

14.10 Standard Access Phases

Accesses to asynchronous devices are composed of a number of standard access phases (according to the type of device and the type of access).

There are six access phases defined:

- Address Phase AP (mandatory for read and write cycles of both device types)

- Command Delay Phase CD (optional)
- Command Phase CP (mandatory for read and write cycles of asynchronous device types)
- Data Hold Phase DH (optional, only applies to write cycles)
- Recovery Phase RP (optional)

Throughout the remainder of this document, a short-hand notation is adopted to represent any clock cycle in any phase. This notation consists of two or three letters followed by a number. The letters identify the access phase within which the clock cycle is located (e.g. AP for Address Phase). The number denotes the number of EBU_CLK clock cycles within the phase (i.e. 1 = first, etc.). In the case of delays that can be extended by external control inputs the lower case letters “e” and “i” are inserted following the two letter phase identifier to differentiate between internally (“i”) and externally (“e”) generated delays. For example, AP2 identifies the second clock in the Address Phase. CPe3 identifies the third clock in the Command Phase which is being extended by external wait-states.

14.10.1 Address Phase (AP)

The Address Phase is mandatory. It always consists of at least one or more EBU_CLK cycles. The phase can be optionally extended to accommodate slower devices.

At the start of the Address Phase, the EBU:

- Selects the device to be accessed by asserting the appropriate $\overline{\text{CSx}}$ signal,
- Issues the address which is to be accessed on the address bus,
- Asserts the $\overline{\text{ADV}}$ signal low,[1)]
- Asserts the appropriate $\overline{\text{BCx}}$ signals if these are programmed to be asserted with the $\overline{\text{CSx}}$ signal,
- At the end of the Address Phase the EBU returns the $\overline{\text{ADV}}$ signal to high.

The length (number of EBU_CLK cycles) of the Address Phase is programmed via the BUSAPx.ADDRC bit field parameter.

14.10.2 Address Hold Phase (AH)

The Address Hold Phase is optional. It consists of zero or more EBU_CLK cycles. It is intended to provide hold time for the multiplexed address bits after the $\overline{\text{ADV}}$ signal has returned to the inactive state.

At the end of the address hold phase, the multiplexed address can be removed from the bus:

- During a read access, the multiplexed address/data bus can return to the high impedance condition to allow the read data to be driven by the external memory

1) If an active high, ALE, signal is required, the polarity of the $\overline{\text{ADV}}$ output can be inverted by setting the ALE field of the MODCON register.

- During a write access, the write data can be driven onto the multiplexed address/data bus

14.10.3 Command Delay Phase (CD)

The Command Delay phase is optional. This means that it can also be programmed for a length of zero EBU_CLK clock cycles. The CD phase allows for the insertion of a delay between Address Phase (or optional Address Hold phase) and Command Phase(s). This phase accommodates devices that are not fast enough to receive commands immediately after getting the address or multiplexed devices which require a bus turnaround delay on reads.

The length (number of EBU_CLK cycles) of the Command Delay phase is programmed via the BUSAPx.CMDDELAY bit field. This parameter makes it possible to select between zero to seven Command Delay phases.

14.10.4 Command Phase (CP)

The Command Phase is mandatory for asynchronous devices. It always consists of at least one or more EBU_CLK cycles. The phase can optionally be extended to accommodate slower devices.

The length (number of EBU_CLK cycles) of the Command Phase is separately programmable for read and write accesses. Bit field BUSAPx.WAITRDC determines the basic length of Command Phases during read cycles and bit field BUSAPx.WAITWRC determines the basic length of Command Phases during write cycles.

Additionally, when accessing asynchronous devices, a Command Phase can also be extended externally using the $\overline{\text{WAIT}}$ signal when the region being accessed is programmed for external command delay control via bit BUSCONx.WAIT or EMUBC.WAIT.

The Command Phase is further subdivided into:

- CPi (= internally-programmed Command Phase)
- CPe (= externally-prolonged Command Phase, i.e. prolonged by the assertion of the $\overline{\text{WAIT}}$ signal).

At the start of the Command Phase, the EBU:

- Asserts the appropriate control signal $\overline{\text{RD}}$ or RD/$\overline{\text{WR}}$ low according to the access type (read or write),
- Issues the data to be written on the data bus AD[15:0] (in the case of a write cycle),
- Asserts the appropriate $\overline{\text{BCx}}$ low (in the case where $\overline{\text{BCx}}$ is programmed to be asserted with the $\overline{\text{RD}}$ or RD/$\overline{\text{WR}}$ signals).

At the end of the Command Phase during an asynchronous access, the EBU:

- Returns the appropriate control signal $\overline{\text{RD}}$ or RD/$\overline{\text{WR}}$ high according to the type of access type (read or write),

- Latches the data from the data bus AD[15:0] (in the case of a read cycle),
- Returns the appropriate $\overline{BCx}$ high (in the case where $\overline{BCx}$ is programmed to be asserted with the $\overline{RD}$ or RD/$\overline{WR}$ signals).

14.10.5 Data Hold Phase (DH)

The Data Hold phase is optional. This means that it can also be programmed for a length of zero EBU_CLK clock cycles. Furthermore, it is only available for asynchronous write accesses. The Data Hold phase extends the amount of time for which data is still held on the bus after the rising edge of the RD/$\overline{WR}$ signal occurred. The Data Hold phase is used to accommodate external devices that require a data hold time after the rising edge of the RD/$\overline{WR}$ signal. The length (number of EBU_CLK cycles) of the Data Hold phase is programmed via the BUSAPx.DATAC bit field.

14.10.6 Burst Phase (BP)

The Burst Phase is mandatory during burst accesses. At the end of the Burst Phase the EBU reads data from the data bus or updates the write data. During a burst access, Burst Phases are repeated as many times as required in order to read or write the required amount of data from or to the external memory device.

The first burst phase of an access will always start on arising edge of BFCLKO. If necessary, the length of the previous phase will be extended to ensure that this happens.

At the end of the last Burst Phase during a burst read access, the EBU:

- Returns the $\overline{CSx}$ signal high,
- Returns the $\overline{RD}$ signal high.

During accesses to Burst Flash devices the length of the Burst Phase must be programmed such that the end of the Burst Phase always coincides with a positive edge of the appropriate BFCLKO (Burst Flash Clock) signal.

A Burst Phase is always at least one clock cycle in length. When BUSRCONx.AGEN is not equal to 1101_B, Demuxed Burst Type External Memory (DDR flash protocol), then the length of each Burst Phase (i.e. the number of EBU_CLK cycles) is derived from the value of the EXTCLOCK and EXTDATA fields in the BUSAPx register. The length of the burst phase will be either be:

- one period of BFCLKO if EXTDATA is 00_B,
- two periods of BFCLKO if EXTDATA is 01_B.
- four periods of BFCLKO if EXTDATA is 10_B.
- eight periods of BFCLKO if EXTDATA is 11_B.

If BUSCON.AGEN is equal to 1101_B, then the length of the burst phase will be controlled by the EXTDATA field only and the burst phase length will be equal to EXTDATA+1. This allows the external clock period to be an integer multiple of the burst phase length. This will allow support for devices which return data on both edges of the device clock. (e.g. setting EXTDATA to 00_B and EXTCLOCK to 01_B will support a DDR style flash device)

14.10.7 Recovery Phase (RP)

The Recovery Phase is optional (although for access types which would cause a bus contention a single cycle of recovery is normally forced by the memory controller logic). This means that it can also be programmed for a length of zero EBU_CLK clock cycles. This phase allows the insertion of a delay following an external bus access that delays the start of the Address Phase for the next external bus access. This permits flexible adjustment of the delay between accesses to the various external devices. The following individually programmable delays are provided on a region by region basis for the following conditions:

- Bit fields BUSAPx.RDRECOVC determine the basic length of the Recovery Phase after a read access.
- Bit fields BUSAPx.WRRECOVC determine the basic length of the Recovery Phase after a write access.
- Bit fields BUSAPx.DTACS determine the length (basic number of EBU_CLK clock cycles) of the Recovery Phase after a read/write access of one region that is followed by a read/write access of another region or a read to one region is followed by a write to the same region (BUSRAPx.DTACS) or a write to one region is followed by a read to the same region (BUSWAPx.DTACS).

The EBU implements a “highest wins” algorithm to ensure that the longest applicable recovery delay is always used between consecutive accesses to the external bus. **Table 14-18** shows the scheme for determining this delay for all possible circumstances. For example, if a read access to a region associated with $\overline{CS1}$ is followed by a write to a region associated with $\overline{CS2}$, the delay will be the highest of BUSRAP1.DTACS and BUSRAP1.RDRECOVC. In this case, if BUSRAP1.DTACS is greater than BUSRAP1.RDRECOVC, then the number of recovery cycles between the two accesses is BUSRAP1.DTACS clock cycles (minimum).

Table 14-18 Parameters for Recovery Phase

Case			Parameter(s) used to calculate "Highest Wins" Recovery Phase
Region	**Current Access**	**Next Access**	
Same $\overline{CSn}$	Read	Read	RDRECOVC
	Write	Write	WRRECOVC
	Read	Write	BUSRAPx.DTACS
	Write	Read	BUSWAPx.DTACS
Different $\overline{CSn}$	Read	Read	BUSRAPx.DTACS, RDRECOVC
	Write	Write	BUSWAPx.DTACS, WRRECOVC
	Read	Write	BUSRAPx.DTACS, RDRECOVC
	Write	Read	BUSWAPx.DTACS, WRRECOVC

14.11 Asynchronous Read/Write Accesses

Asynchronous read/write access of the EBU support the following features:

- EBU_CLK clock-synchronous signal generation
- Support for 16-bit and 32-bit bus width
  Performing an AHB access with a data width greater than that of the external device automatically triggers a sequence of the appropriate number of external accesses to match the AHB access width.
- Demultiplexed address/data lines
- Programmable access parameters
  – Internal control of command delay cycles
  – External and/or internal control of wait states
  – Variable data hold cycles for write operation (to allow flexible hold time adjustment)
  – Variable inactive/recovery cycles when:
  Switching between different memory regions (CS),
  Switching between read and write operations,
  After each read cycle,
  After each write cycle.

Software driver routines are required in order to support Nand Flash devices using asynchronous device accesses. A single Nand Flash access sequence is performed by generating the appropriate sequence of discrete asynchronous device accesses in software.

The EBU does not provide support 8-bit bus width. When 8-bit SRAM devices are used, they must be used in pairs to implement a 16-bit wide memory region.

14.11.1 Signal List

The following signals of the EBU are used for asynchronous accesses:

Table 14-19 Asynchronous Mode Signal List

Signal/Pin	Type	Function
AD[31:0]	I/O	Address/Data bus lines 0-31
A[24:16]	O	Address bus lines 16-24
$\overline{\text{CS}}$[3:0]	O	Chip select 0-3
$\overline{\text{RD}}$	O	Read control line
RD/$\overline{\text{WR}}$	O	Write control line
$\overline{\text{BC}}$[3:0]	O	Byte control lines 0-3
$\overline{\text{WAIT}}$	I	Wait input

14.11.2 Standard Asynchronous Access Phases

Accesses to asynchronous devices are composed of a subset of the standard access phases which are detailed in **Section 14.10**. The standard access phases for asynchronous devices are:

- AP: Address Phase (compulsory - see **Section 14.10.1**)
- CD: Command Delay Phase (optional - see **Section 14.10.3**)
- CP: Command Phase (compulsory - see **Section 14.10.4**)
- DH: Data Hold Phase (optional - see **Section 14.10.5**)
- RP: Recovery Phase (optional - see **Section 14.10.7**)

14.11.3 Control of $\overline{\text{ADV}}$ & $\overline{\text{CS}}$ Delays During Asynchronous Accesses

For asynchronous accesses, the Memory Controller output signals: $\overline{\text{ADV}}$, $\overline{\text{CS}}$, $\overline{\text{RD}}$, RD/$\overline{\text{WR}}$, $\overline{\text{BC}}$ and AD signals can be delayed with respect to the start of the access phases they are asserted in. The amount by which the signal is delayed depends on the setting of the EXTCLOCK field of BUSAPx:-

- When EXTCLOCK is set to 00_B, signals are asserted on the negative edge of EBU_CLK (i.e. it is in effect delayed by 1/2 an Memory Controller clock cycle with respect to the other signals).
- When EXTCLOCK is not set to 00_B, control signals are asserted on the next positive edge of EBU_CLK (i.e. it is in effect delayed by an EBU_CLK cycle with respect to the other signals).

Memory Controller allows these delays to be removed independently via user programmable bits. The default setting after reset has the delay enabled

Table 14-20 $\overline{\text{ADV}}$ and $\overline{\text{Chip Select}}$ Signal Timing

EXTCLOCK is set to	Delay Disabled[1)]	Delay Enabled
00_B	Start of AP1	Middle of AP1
01_B, 10_B, 11_B	Start of AP1	End of AP1

1) See **Figure 14-24** for details of this signal positioning.

This function is controlled by the register bits BUSCONx.EBSE for $\overline{\text{ADV}}$ and BUSCONCx.ECSE for the other control signals.

Note: If $\overline{CS}$ is delayed a recovery phase must be used to prevent conflicts between chip selects as the rising edge of chip select will be delayed past the end of the burst phases. Also, for muxed devices, the write data will be delayed into the address phase of the next access, resulting in the valid address being driven one clock after $\overline{ADV}$ is asserted.

14.11.4 Programmable Parameters

Table 14-21 lists the programmable parameters for asynchronous accesses. These parameters only apply to asynchronous devices when BUSCONx.AGEN = 000_B. Note that emulation registers “EMU…” include parameters that control the emulator chip select region ($\overline{\text{CSEMU}}$ output), while “BUS…x” registers include parameters that control the four $\overline{\text{CS}}$[3:0] chip select regions x. The equivalent registers contain identical bits and bit fields.

Table 14-21 Asynchronous Access Programmable Parameters

Register	Parameter (Bit/Bit field)	Function
BUSAPx	ADDRC	Number of cycles in address phase
	CMDDELAY	Number of programmed command delay cycles.
	WAITRDC	Number of programmed wait states for read accesses.
	WAITWRC	Number of programmed wait states for write accesses.
	DATAC	Number of Data Hold cycles.

Table 14-21 Asynchronous Access Programmable Parameters (cont'd)

Register	Parameter (Bit/Bit field)	Function
BUSAPx	RDRECOVC	Number of minimum recovery cycles after a read access.
	WRRECOVC	Number of minimum recovery cycles after a write access.
	DTARDWR	Number of minimum recovery cycles between a read access and a write access.
	DTACS	Number of minimum recovery cycles when the next access going to a different memory region.
BUSCONx	WAIT	External Wait State control (OFF, asynchronous, synchronous)
	WAITINV	Reversed polarity at $\overline{\text{WAIT}}$: active low or active high

14.11.5 Accesses to Multiplexed Devices

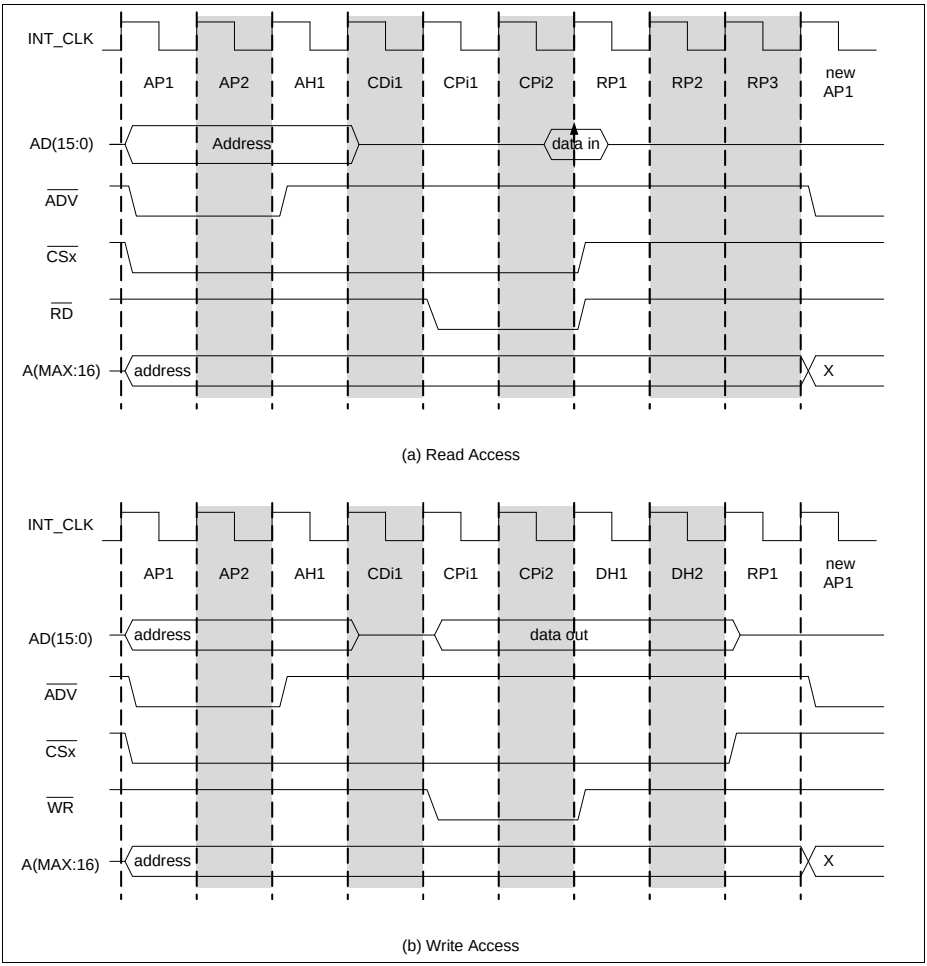


Figure 14-17 Multiplexed External Bus Access Cycles

Figure 14-17 shows an example of a read access to a multiplexed device. This type of access cycle consists of two to six phases as follows:

- Address Phase (compulsory)
- Address Hold Phase (optional)
- Command Delay Phase (optional)

- Command Phase (compulsory)
- Data Hold Phase (optional)
- Recovery Phase (optional)

14.11.6 Dynamic Command Delay and Wait State Insertion

In general, there are two critical phases during asynchronous device accesses. These phases are:

- **Command Delay Phase** (see **Section 14.10.3**).
- **Command Phase** (see **Section 14.10.4**).

In the EBU, internal length programming for the Command Delay Phase is available via bit fields BUSAPx.CMDDELAY.

The equivalent control capability for the Command Phase is available for bit fields BUSRAPx.WAITRDC and BUSWAPx.WAITWRC.

14.11.6.1 External Extension of the Command Phase by WAIT

The $\overline{\text{WAIT}}$ input can be used to cause the EBU to extend the Command Phase by inserting additional cycles prior to deactivation of the $\overline{\text{RD}}$ and RD/$\overline{\text{WR}}$ lines. This signal can be programmed separately for each region to be ignored or sampled either synchronously or asynchronously (selected via the BUSCONx.WAIT bit field). Additionally, the polarity of $\overline{\text{WAIT}}$ can be programmed for active low (default after reset) or active high function via bit BUSCONx.WAITINV. The signal will only take effect after the programmed number of Command Phase cycles has passed. This means that the signal can only be used to extend the phase, not to shorten it.

When programmed for synchronous operation, $\overline{\text{WAIT}}$ is sampled on every rising edge of EBU_CLK during the Command Phase. The sampled value is then used on the next rising edge of EBU_CLK to decide whether to prolong the Command Phase or to start the next phase. **Figure 14-18** shows an example of $\overline{\text{WAIT}}$ used in Synchronous Mode.

Note: Due to the one-cycle delay in Synchronous Mode between the sampling of the $\overline{\text{WAIT}}$ input and its evaluation by the EBU, the Command Phase must always be programmed to be at least one EBU_CLK cycle (via BUSAPx.WAITRDC or BUSAPx.WAITWRC) in this mode.

When programmed for asynchronous operation, $\overline{\text{WAIT}}$ is also sampled at each rising edge of EBU_CLK during the Command Phase. However, an extra synchronization cycle is inserted prior to the use of the sampled value. This means that the sampled value is not used until the second following rising edge of EBU_CLK. **Figure 14-19** shows an example of $\overline{\text{WAIT}}$ used in Asynchronous Mode.

Note: Due to the two-cycle delay in Asynchronous Mode between the sampling of the $\overline{\text{WAIT}}$ input and its evaluation by the EBU, the Command Phase must always be

programmed to be at least two EBU_CLK cycles (via BUSAPx.WAITRDC or BUSAP.WAITWRC) in this mode.

Figure 14-18 shows an example of the extension of the Command Phase through the $\overline{\text{WAIT}}$ input in synchronous mode:

- At EBU_CLK edge 1 (at the end of the Address Phase), the EBU samples the $\overline{\text{WAIT}}$ input as low and starts the first cycle of the Command Phase (CPi1 - internally programmed).
- At EBU_CLK edge 2, the EBU samples the $\overline{\text{WAIT}}$ input as low and starts an additional Command Phase cycle (CPe2 - externally generated) as a result of the $\overline{\text{WAIT}}$ input sampled as low at EBU_CLK edge 1.
- At EBU_CLK edge 3, the EBU samples the $\overline{\text{WAIT}}$ input as high and starts an additional Command Phase cycle (CPe3 - externally generated) as a result of the $\overline{\text{WAIT}}$ input sampled as low at EBU_CLK edge 2.
- Finally at EBU_CLK edge 4, as a result of the $\overline{\text{WAIT}}$ input sampled as high at point 3, the EBU terminates the Command Phase, reads the input data from D[31:0] and starts the Recovery Phase.

Note: Synchronous operation means that even though access to the device may be asynchronous, the control logic generating the control signals must meet setup and hold time requirements with respect to EBU_CLK.

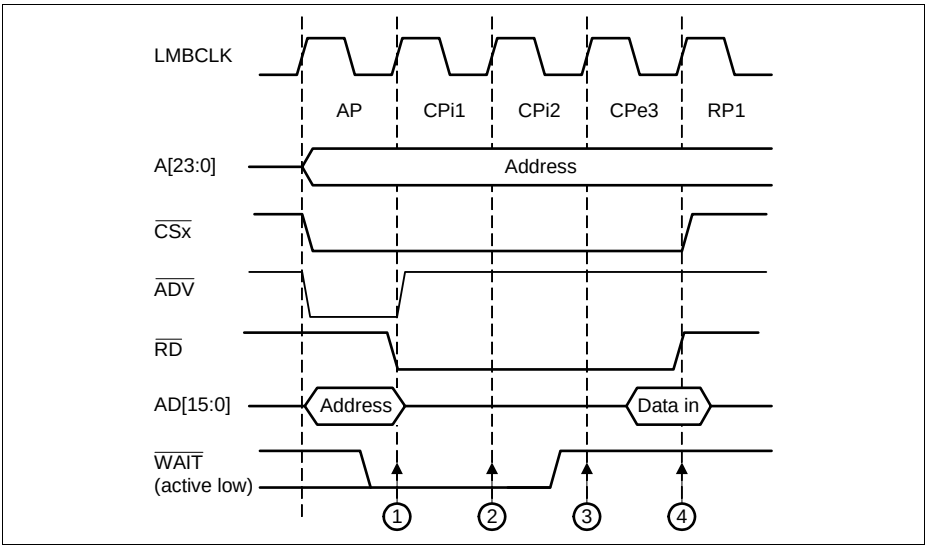


Figure 14-18 External Wait Insertion (Synchronous Mode)

Figure 14-19 shows an example of the extension of the Command Phase through the $\overline{\text{WAIT}}$ input in asynchronous mode:

- At EBU_CLK edge 1 (at the end of the Address Phase), the EBU samples the $\overline{\text{WAIT}}$ input as low and starts the first cycle of the Command Phase (CPi1 - internally programmed).
- At EBU_CLK edge 2, the EBU samples the $\overline{\text{WAIT}}$ input as low and starts the second cycle of the Command Phase (CPi2 - internally programmed).
- At EBU_CLK edge 3, the EBU samples the $\overline{\text{WAIT}}$ input as high and starts an additional Command Phase cycle (CPe3 - externally generated) as a result of the $\overline{\text{WAIT}}$ input sampled as low at EBU_CLK edge 1.
- At EBU_CLK edge 4, the EBU starts an additional Command Phase cycle (CPe4 - externally generated) as a result of the $\overline{\text{WAIT}}$ input sampled as low at EBU_CLK edge 2.
- Finally at EBU_CLK edge 5, as a result of the $\overline{\text{WAIT}}$ input sampled as high at EBU_CLK edge 3, the EBU terminates the Command Phase, reads the input data from AD[15:0],and starts the Recovery Phase.

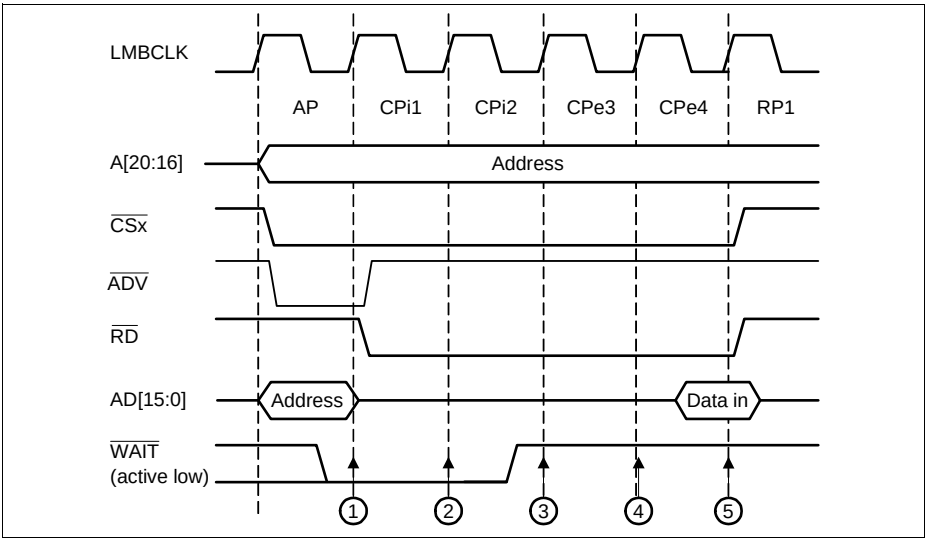


Figure 14-19 External Wait Insertion (Asynchronous Mode)

14.11.7 Interfacing to Nand Flash Devices

The memory controller provides limited support for specific Nand Flash devices. The required access sequences (read or write) are generated by connecting the Nand Flash device as an Asynchronous Device and using appropriate processor generated access sequences to emulate the NAND flash commands. **Figure 14-20** Shows an example of Memory Controller connected to a Nand Flash device:-

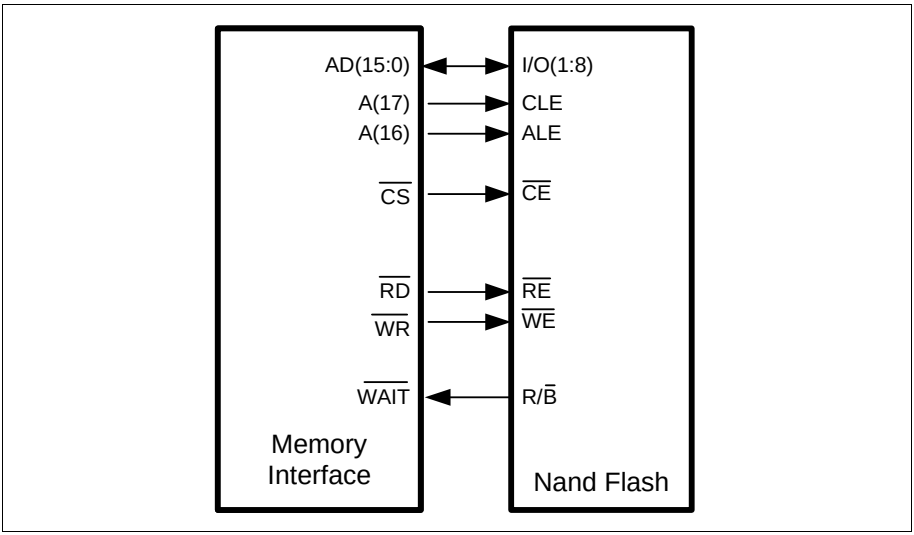


Figure 14-20 Example of interfacing a Nand Flash device to the Memory Controller

The R/$\overline{B}$ input from the NAND flash is connected to the memory controller $\overline{WAIT}$ input and is available as the MODCON.STS. This enables a NAND flash to be driven by software from the processor.

As shown above only two address lines are connected to the Nand Flash, and rather than being connected to address inputs, they are connected to control inputs. This allows access to three “registers” in the Nand Flash as follows:-

Table 14-22 Nand Flash “Registers”

AHB Address	“Register”	Comment
Base + 00000_H	Data Register	Read/Write: Used to read data from and write data to the device.
Base + 20000_H	Address Register	Write only: Used to write the required access address to the device.
Base + 40000_H	Command Register	Write only: Used to write the required command to the device.

Note: AHB addresses are byte addresses and addresses on the external bus are 16 bit word addresses. Therefore [AHB address(18)]->[external address(17)] and [AHB address(17)]->[external address(16)].

Note that the Memory Controller does not directly support byte wide devices. Writes to 8 bit, NAND Flash devices must therefore be done as 16-bit word writes with the valid byte in the lower part and the upper-byte padded.

14.11.7.1 NAND flash page mode

NAND flash memories are page oriented devices capable of extended read operations with a single setup phase for command signals at the beginning of the access. The asynchronous controller of the Memory Controller will split a large transfer into multiple accesses to external memory but each of these accesses will have the overhead of the initial setup phase. Enabling page mode, using the AGEN field in BUSCONx will cause the standard flow of the controller to be modified as follows:

- For a read, if data remains to be fetched at the end of a command phase, the controller will start a new command delay phase, instead of a new address phase or recovery phase and the address will not be incremented. If BUSRAPx.cmddelay is set to zero, the command delay phase will have a duration of one clock cycle but in this case the command delay phase is mandatory to ensure that the RD and RD/WR signals return to the high state.
- For a write, if data remains to be written at the end of a data hold phase (or command phase if the length of data hold is zero), the controller will start a new command phase, instead of a new address phase or recovery phase and the address will not be incremented. If BUSWAPx.datac is set to zero, the data hold phase will have a duration of one clock cycle as in this case the data hold phase is mandatory to ensure that the RD and RD/WR signals return to the high state. The command phase will be forced to have a minimum length of two clocks.

See **Figure 14-21** for example waveforms.

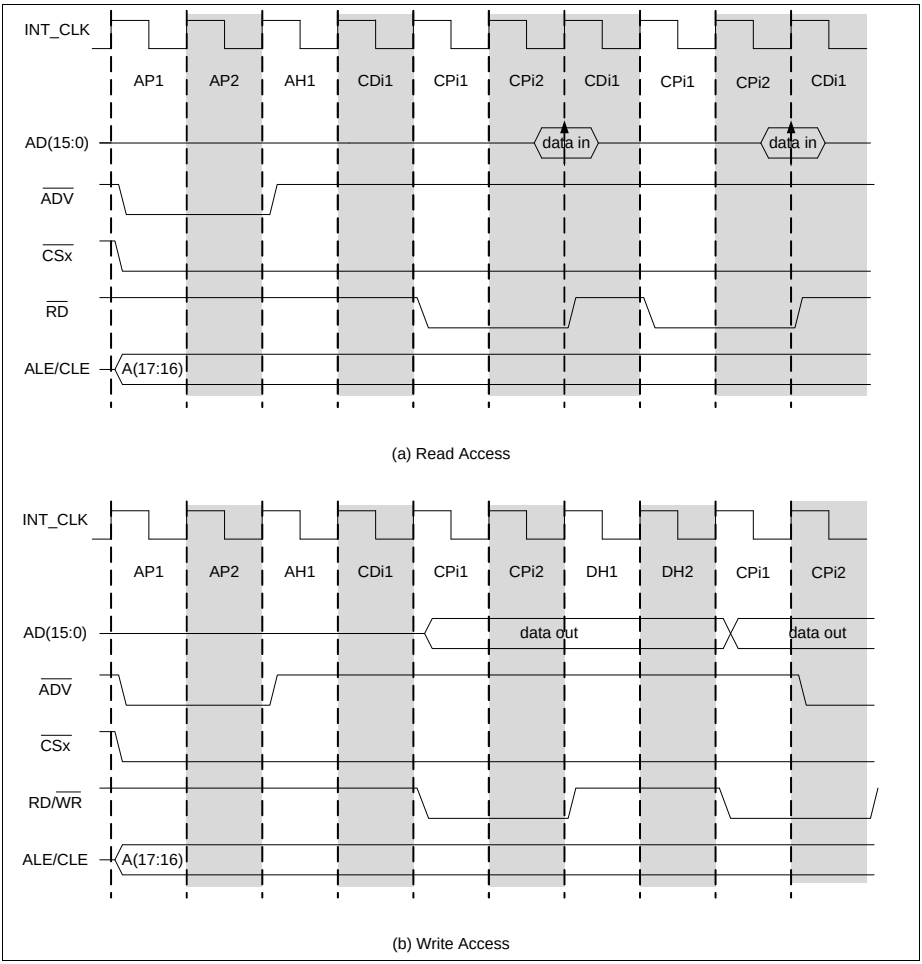


Figure 14-21 NAND Flash Page Mode Accesses

Example Nand Flash Read Sequence

Figure 14-22 shows an example of how the processor can generate a Nand Flash read access sequence given this configuration:-

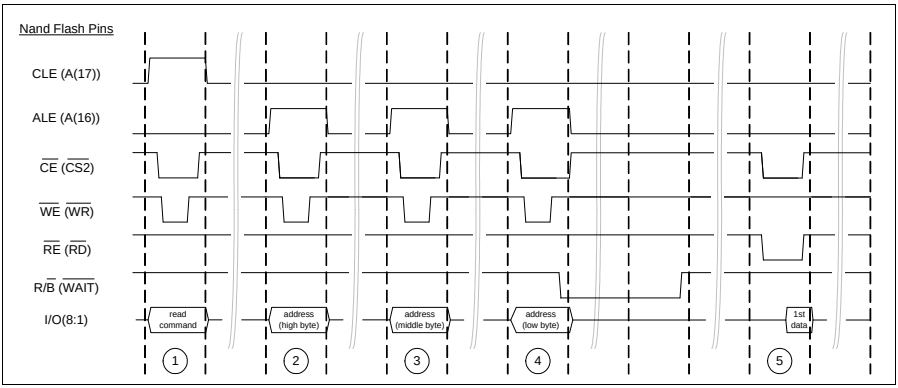


Figure 14-22 Example of an Memory Controller Nand Flash access sequence (read)

1. In the cycle marked ‘1’ in **Figure 14-22** the processor initiates a read sequence by writing the “Read Command” value to address “NAND_FLASH_BASE + 0x40000”. This generates a write sequence with CLE (A(17)) driven high and ALE (A(16)) driven low.
2. In the cycle marked ‘2’ the processor loads the most significant byte of the read address by writing to address “NAND_FLASH_BASE + 0x20000”. This generates a write sequence with CLE (A(17)) driven low and ALE (A(16)) driven high.
3. In the cycle marked ‘3’ the processor loads the middle significant byte of the read address by repeating the access specified in ‘2’ above.
4. In the cycle marked ‘4’ the processor loads the least significant byte of the read address by repeating the access specified in ‘2’ above. The Nand Flash responds to this final address byte by driving it’s R/$\overline{B}$ output low. The processor monitors this pin (using the MODCON.sts bit) until the Nand Flash has completed it’s internal data fetch.
5. In the cycle marked ‘5’ the processor reads the first byte of data by reading address “NAND_FLASH_BASE + 0x00000”. The processor can subsequently read any additionally required (sequential) data bytes by repeating cycle ‘5’.

Note: A similar scheme can be used to generate write access sequences.

14.12 Synchronous Read/Write Accesses

The Memory Controller is designed to generate waveforms compatible with the burst modes of:

1. INTEL and compatible burst flash devices
2. SPANSION and compatible burst flash devices
3. INFINEON and MICRON cellular RAM

4. Fujitsu and Compatible FCRAM/uTRAM/CosmoRAM
5. Samsung OneNAND burst capable NAND flash and compatible devices
6. M-Systems DiskOnchipG3 and compatible devices
7. GSI SSRAM

Note: Not all of the supported synchronous memory types are known to be available in automotive grade

Features

The Synchronous Access Controller is primarily designed to perform burst mode read cycles for an external instruction memory and read and write cycles for an external Cellular RAM or FCRAM data memory. In general, the features are:-

- Fully synchronous timing with flexible programmable timing parameters (address cycles, read wait cycles, data cycles).
- Programmable $\overline{\text{WAIT}}$ function.
- Programmable burst (mode and length)
- 16-bit device width.
- 32-bit device width
- Page mode read accesses.
- Resynchronisation of read data to a feedback clock to maximize the frequency of operation (optional).

14.12.1 Signals

The following signals are used for the Burst FLASH interface:-

Table 14-23 Burst Flash Signal List

Signal	Type	Function
AD(31:0)	I/O	Multiplexed Address/Data bus
$\overline{\text{RD}}$	O	Read control
$\overline{\text{WR}}$	O	Write control
A(24:16)	O	Most Significant Part of Address bus
$\overline{\text{ADV}}$	O	Address valid strobe
$\overline{\text{WAIT}}$	I	Wait/terminate burst control
$\overline{\text{CS}}$(3:0)	O	Chip select
BFCLKO	O	Burst FLASH Clock, running equal to, 1/2, 1/3 or 1/4 of the frequency of EBU_CLK.

Table 14-23 Burst Flash Signal List (cont'd)

Signal	Type	Function
BFCLKI	I	Burst FLASH Clock Feedback.
STS	I	Burst FLASH Status Input (Optional, value of $\overline{\text{WAIT}}$ pin available through status register)

14.12.2 Support for four Burst FLASH device types

Support is provided for a maximum of four different Burst FLASH configurations on the external bus - i.e. one on each external chip select.

Bit-fields BUSCONx.EBSE, BUSCONx.ECSE, BUSCONx.wait, BUSCONx.FBBMSEL, BUSCONx.BFCMSEL and BUSCONx.FETBLEN are used to configure specific characteristics for burst access cycles.

14.12.3 Typical Burst Flash Connection

The figure below shows a typical burst flash connection.

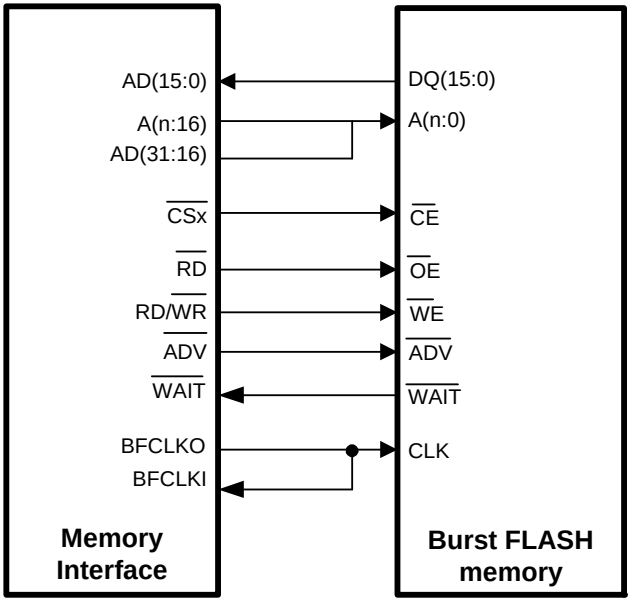


Figure 14-23 Typical Burst Flash Connection

14.12.4 Burst Flash Clock

Since the EBU_CLK can run too fast for clocking Burst FLASH devices, the Memory Controller provides an additional clock source (BFCLKO). This signal is generated by a programmable clock divider driven by EBU_CLK and allows EBU_CLK to BFCLKO ratios of 1:1, 2:1, 3:1 and 4:1 to be selected. The frequency of the signal is determined by bit-field BUSRP.EXTCLOCK. Note that it is possible to set a different clock rate for synchronous writes to the same device by programming BUSWP.EXTCLOCK to a different value.

If a continuously running BFCLKO is required, then the BUSRCONx.BFCMSEL field can be used to enable an ungated flash clock. This bit is normally set to 1_B in all the BUSRCONx registers after reset. If cleared, the related BUSRAPx.EXTCLOCK field will be used to generate a stable BFCLKO. If multiple BUSRCONx.BFCMSEL fields are set to 0_B, then the highest priority (lowest index) BUSRAPx.EXTCLOCK field will be used.

During a burst access to a synchronous device, BFCLKO will generate correctly aligned clock edges as shown in **Figure 14-24**. The BFCLKO signal is gated to ensure that it is low (zero) at all other times (including asynchronous read/writes of/to synchronous devices). This provides power savings and ensures correct asynchronous accesses to Burst FLASH device(s).

The start of the address and burst phases are synchronized, by hardware, to the rising edge of BFCLKO. Exiting from a phase extended by the WAIT input will also be synchronized to the rising edge of BFCLKO.

Note: The length of the standard accesses phases during Burst FLASH accesses are programmed as a multiple of EBU_CLK independent of the BFCLKO frequency. It is the users responsibility to program the access phases to ensure that the sampling of data by Memory Controller guarantees valid sampling of the data from the Burst FLASH device.

The EBU uses the EBU_CLK clock to generate all external bus access sequences.

Table 14-24 EXTCLOCK to clock ratio mapping

EXTCLOCK value	BFCLKO divide ratio
00	1:1
01	1:2
10	1:3
11	1:4

Unless documented elsewhere, all outputs to the external bus are generated of the rising edge of EBU_CLK.

The BFCLKO phase is controlled so that control signal changes will normally occur at the rising edge of BFCLKO unless configured otherwise by register settings.

14.12.5 Standard Access Phases

Accesses to burst FLASH devices are composed of a number of “Standard Access Phases” (which are detailed in **Section 14.10**). The Standard Access Phases for Burst FLASH devices are:-

- AP: Address Phase (compulsory - see **Section 14.10.1**).
- AH: Address Hold Phase (optional see **Section 14.10.2**).
- CD: Command Delay Phase (optional - see **Section 14.10.3**).
- CP: Command Phase (compulsory - see **Section 14.10.4**).
- BP: Burst Phase (compulsory - see **Section 14.10.6**).
- RP: Recovery Phase (optional - see **Section 14.10.7**).

Note: During a burst access the Burst Phase (BP) is repeated the required number of times to complete the burst length.

14.12.6 Burst Length Control

The maximum number of valid data samples that can be generated by a flash device in a single read access is set by the BUSCONx.FBBMSEL bit and the BUSCONx.FETBLEN bit field.

The BUSCONx.FBBMSEL bit is used to select Continuous Burst Mode where there is no limit to the number of data samples in a burst read access.

The BUSCONx.FBBMSEL and BUSCONx.FETBLEN bit-fields are used to select the maximum number of data samples in a single access. Where an AHB request exceeds the amount of data that can be fetched or stored by the programmed number of data samples, the EBU will automatically generate the appropriate number of burst accesses to transfer the required amount of data.

Note: Selection of Continuous Burst Mode (by use of the ‘FBBMSEL’ bit) overrides the maximum burst setting (specified by the FETBLEN bit-field).

14.12.7 Control of $\overline{\text{ADV}}$ & $\overline{\text{CS}}$ Delays During Burst FLASH Access

By default the Memory Controller output signals: $\overline{\text{ADV}}$, $\overline{\text{CS}}$, $\overline{\text{RD}}$, RD/$\overline{\text{WR}}$, $\overline{\text{BC}}$ and AD signals are delayed with respect to the other clock. The amount by which the signal is delayed depends on the ratio of EBU_CLK to the Burst FLASH clock as follows:-

- When the ratio of EBU_CLK to BFCLKO is 1:1, signals are asserted on the negative edge of EBU_CLK (i.e. it is in effect delayed by 1/2 an Memory Controller clock cycle with respect to the other signals).
- When the ratio of EBU_CLK to BFCLKO is 1:2 or 1:3, control signals are asserted on the next positive edge of EBU_CLK (i.e. it is in effect delayed by an EBU_CLK cycle with respect to the other signals).

- When the ratio of EBU_CLK to BFCLKO is 1:4, control signals are asserted on the negative edge of BFCLK (i.e. it is in effect delayed by two EBU_CLK cycles with respect to the other signals).

The default setting after reset has the delay enabled

If the delay is disabled, then the signals will not be delayed in 1:1 mode (except for ADV which will be guaranteed to be after the edge of BFCLKO). In 2:1, 3:1 and 4:1 mode, the signals will be delayed by half of an EBU_CLK cycle from the start of the cycle in which they are asserted.

Table 14-25 $\overline{\text{ADV}}$ and $\overline{\text{Chip Select}}$ Signal Timing

EBU_CLK:BFCLKO Ratio	Delay Disabled[1)]	Delay Enabled
1:1	Start of AP1	Middle of AP1
2:1, 3:1	half clock cycle after start of AP1	End of AP1
4:1	half clock cycle after start of AP1	End of AP2

1) See **Figure 14-24** for details of this signal positioning.

This function is controlled by the register bits BUSCONx.EBSE for the $\overline{\text{ADV}}$signal and by BUSCON.ECSE for the $\overline{\text{CS}}$, RD/$\overline{\text{WR}}$ and write data signals

Note: If CS is delayed a recovery phase must be used to prevent conflicts between chip selects as the rising edge of chip select will be delayed past the end of the burst phases. Also, for muxed devices, the write data will be delayed into the address phase of the next access, resulting in the valid address being driven one clock after ADV is asserted.

14.12.8 Burst Flash Clock Feedback

The Memory Controller can be configured to use clock feedback to optimize the operating frequency for a given flash device. This is enabled by setting the BUSCONx.FDBKEN bit to one. With this bit enabled the first sampling stage for read data has its own clock (PD_BFCLKFEEDBK_I). This will be derived from the BFCLKO output by using a second pad (BFCLKI) to monitor the BFCLKO signal after the output pad delay.

Clock feedback should be used whenever possible as it allows the best possible performance

In addition the number of synchronization stages (one or two) used to transfer the read-data into the EBU_CLK domain can also be selected via the EBUCON_BUSAPx.BFSSS bit.

When using two synchronization stages (default) the data is initially resynchronized to PD_BFCLKFEEDBK_I and then is additionally internally resynchronized to BFCLKO before being passed to the normal logic. In this mode PD_BFCLKFEEDBK_I can therefore be skewed by almost an entire BFCLK cycle relative to the EBU_CLK clock without losing data integrity. A side effect of using this mode is an increase in data latency of two cycles of BFCLK (compared to not using clock feedback).

When using a single synchronization stage the data is resynchronized to PD_BFCLKFEEDBK_I before being passed to the normal logic. This provides a compromise setting for operating frequency and latency for 1:1 clocking mode where the second resynchronisation stage offers no advantage.

Note: If EBU_CLK:BFCLKO = 1:1, then the second and third resynchronisation stages have identical clock signals. There is therefore no advantage to having the second resynchronisation stage and it can be bypassed without loss of performance.

As above, a side effect of using this mode is an increase in data latency. In this case addition of one BFCLK cycle (compared to not using clock feedback).

Note: Clock feedback will be automatically disabled for burst writes as the additional latency on the WAIT input cannot be tolerated

14.12.9 Asynchronous Address Phase

As operating frequency increases, it becomes increasingly hard to avoid violating some timing parameters. The asynchronous address phase allows the address to be latched into the flash memory using the ADV signal before the clock is enabled. This is only possible if explicitly allowed by the flash data sheet

If the BUSCONx.AAP is set, then the clock will not start until the end of the address hold phase of the access. The rising edge of the clock will always be co-incident with the transition from the address hold phase. If the address hold phase has zero length then the first rising edge of the clock will coincide with the transition from the address phase. If this mode is enabled a recovery phase of one BFCLK period will be enforced at the end of the previous transaction to ensure that the clock has time to turn off before the start of the next access.

Setting this mode for any region will force the clocks on all synchronous accesses to be disabled in the recovery phase. Only one clock pulse will occur during the recovery phase.

AAP mode is incompatible with the continuous clock mode and will be disabled automatically is continuous clocking is enabled by setting any BUSRCONx.BFCMSEL bit to 0_B.

14.12.10 Page Mode Support

If page mode support is enabled using the correct setting of the AGEN field in **BUSRCONx** or **BUSWCONx** then the address on the dedicated address bus pins, A(24:0), will be incremented at the end of every burst phase and the clock will not run for the duration of the access.

The page size of the attached device should be programmed using the **FETBLEN** and **FBBMSEL** fields of the **BUSCONx** register to ensure that the address is not incremented past the page end of the memory device. 16 bit devices with a page size of 16 words should be configured for continuous burst.

Note: The cache line fill will use an AHB, BTR4 transfer. This translates to a 16 word burst for a 16 bit device and is the largest AHB transfer supported by the memory controller as a single transfer on the external bus. The memory controller supports a 16 word burst using the continuous burst setting for FBBMSEL.

Page mode devices do not support WAIT and the relevant **BUSRCONx**.**WAIT** and **BUSWCONx**.**WAIT** should be set to b00.

Note: This mode can only be used with devices that have a separate address and data bus. Use with devices with muxed address/data connections will not return correct data

14.12.11 Critical Word First Read Accesses

In the default case, the memory controller will always start a burst at the lowest address possible and wrapping of the burst data is handled internally to the memory controller. However, some burst devices implement a wrapping feature which is compatible with the wrapped bursts used by the processor cache fill requests. If this is the case, there is an advantage to using the wrapping mode in the device as the instruction required by the processor (the critical word) can be fetched first from the external memory.

The mode is enabled by setting the BUSCONx.dba bit for the appropriate region. Once enabled, the memory controller will not align the start address for the burst and the device will be relied on to return data in the correct order. The memory controller must fetch all the data in a single burst. If the transaction is split into multiple accesses on the external bus by use of the FETBLEN field, the issued addresses will be incorrect.

Note: The cache line fill will use an AHB, BTR4 transfer. This translates to a 16 word burst for a 16 bit device. The device must therefore support a 16 word wrap setting. A 32 bit memory must support a 8 word wrap setting. The memory controller supports a 16 word burst using the continuous burst setting for FBBMSEL. Other BTR opcodes must not be generated for accesses to the external memory by the system if this mode is enabled, otherwise data corruption will occur.

14.12.12 Example Burst Flash Access Cycle

The figure below shows an example burst flash access.

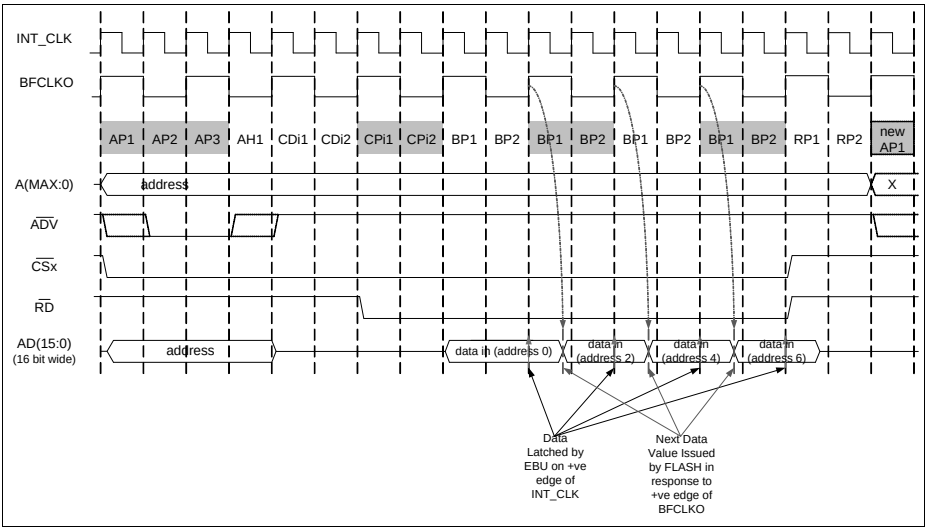


Figure 14-24 Burst FLASH Read without Clock Feedback (burst length of 4)

Note:

4. *The start of the cycle is synchronised to a +ve edge of the BFCLKO signal*
5. *The BFCLKO signal is used to clock the Burst FLASH devices*
6. *BFCLKO to Internal Clock frequency ratio can be programmed to 1:1, 1:2, 1:3 or 1:4. Each BFCLKO +ve edge is generated from a +ve edge of internal clock*
7. *Addresses show are "byte addresses"*
8. *ADV signal positioning is programmable via the EBSE bitfield in the BUSCON registers*

Figure 14-24 shows an example of a burst read access (burst length of four) to a Burst FLASH device with $\overline{\text{WAIT}}$ and clock feedback functions disabled.

Programmability of the length of the Address, Command Delay and Command phases allows flexible configuration to meet the initial read access time of a Burst FLASH device.

Data is sampled at the end of each Burst Phase cycle. The Burst Phase is repeated the appropriate number of times for the programmed burst length (programmable for lengths of 1, 2, 4 or 8 via the BUSCONx.FETBLEN bit-field).

Figure 14-24 shows an access cycle with the following settings:-

- Clock Feedback disabled.
- Address Phase length = 3 EBU_CLK cycles (see ADDRC and **Section 14.10.1**).

- Command Delay Phase length = 3 EBU_CLK cycles (see CMDDELAY and **Section 14.10.3**).
- Command Phase length = 2 EBU_CLK cycles (see WAITRDC and **Section 14.10.4**).
- Burst Phase length = 2 EBU_CLK cycles (see EXTCLOCK, EXTDATA and **Section 14.10.6**).
- Recovery Phase length = 2 EBU_CLK cycles (see **Section 14.10.7**).
- Burst Length = 4 (see FETBLEN).
- BFCLKO frequency = 1/2 of EBU_CLK frequency (see EXTCLOCK).

14.12.13 External Cycle Control via the $\overline{\text{WAIT}}$ Input

Memory Controller provides control of the Burst FLASH device via the $\overline{\text{WAIT}}$ input. This allows Memory Controller to support operation of Burst FLASH while crossing Burst FLASH page boundaries. During a Burst FLASH access the $\overline{\text{WAIT}}$ input operates in one of four modes:-

- Disabled
- Early Wait for Page Load.
- Wait for Page Load.
- Abort and Retry Access.

Selection of the mode in which the $\overline{\text{WAIT}}$ input operates during Burst FLASH reads is selected via the BUSCONx.wait bits.

Note: Selection of "Disabled" via the wait bit-field prevents the $\overline{\text{WAIT}}$ input having any effect on a Burst FLASH access cycle

Wait for Page Load Mode

This mode supports devices which assert a $\overline{\text{WAIT}}$ output for the duration of clock cycles in which the data output by the device is invalid or, alternatively, one clock cycle earlier than the data output is invalid. This includes Intel and AMD Burst FLASH devices (and compatibles) configured for Early Wait Generation Mode (BUSCONX.wait=01_B) and standard wait generation (BUSCONx.wait=10_B).

In operation, the burst flash controller loads a counter with the required number of samples at the start of each burst. At the end of each burst phase, the burst flash controller samples the $\overline{\text{WAIT}}$ input and the data bus at the end of each Burst phase. If $\overline{\text{WAIT}}$ is inactive, the sample is valid, the sample counter is decremented and the sampled data is passed to the datapath of the Memory Controller. This synchronous sampling means that the validity of the sample can not be determined until the clock cycle after the end of the burst phase. The Burst Flash controller will therefore overrun and generate extra burst phases until the sample counter is decremented to zero. Extra data samples returned after the sample counter is zero will be discarded.

The only difference if early wait is used is that the validity of data in burst phase "n" is determined by the value of $\overline{\text{WAIT}}$ in burst phase "n-1".

This mode of operation is compatible with the use of clock feedback as, with feedback enabled, WAIT is fed through the same resynchronisation signals as the data bus. The only effect on operation is that the number of overrun cycles will increase as the decrementing of the sample counter will be lagged by the resynchronisation stages.

During the initial phases of an access, WAIT is sampled on every edge of EBU_CLK. This is so the first burst phase is working with an accurate value for the WAIT signal. To ensure this is the case, the command phase should be of sufficient length to allow the device to drive WAIT and for the signal to propagate to the controller.

14.12.14 Flash Non-Array Access Support

Several types of flash memories will assert WAIT permanently during an access which is not directed to the memory array. An example of this would be polling the status register to check if a programming operation has completed. If the BUSRCON[3:0].NAA field is set, then an access to the region with AHB A(26) set will proceed as if the appropriate wait field in BUSRCON[3:0] or BUSWCON[3:0] was set to 00_B and WAIT was disabled. When set, this field affects both read and write accesses.

14.12.15 Termination of a Burst Access

A burst read operation is terminated by de-asserting CSx signal followed by the appropriate length Recovery Phase. **Figure 14-25** shows an example of termination of a burst access following the read of two locations (i.e. two Burst Phases) from a 16-bit non-multiplexed Burst FLASH device.

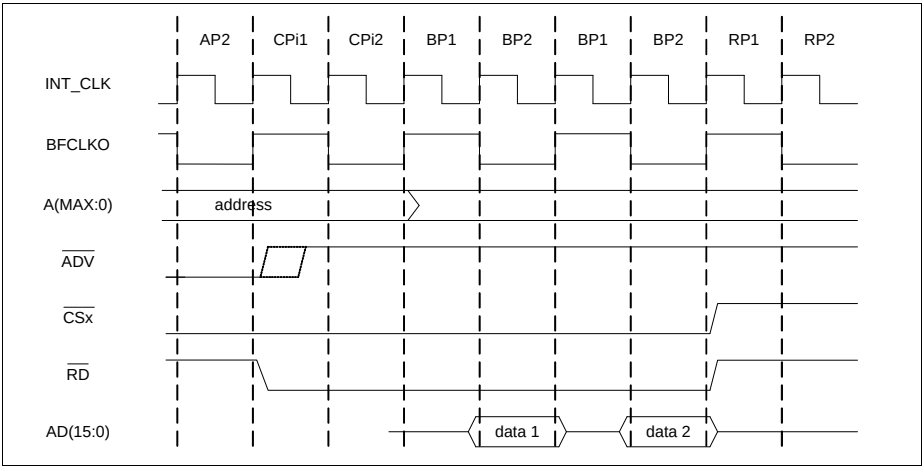


Figure 14-25 Terminating a Burst by de-asserting CS

14.12.16 Burst Flash Device Programming Sequences

Command sequences for some Burst Flash devices must not be interrupted by other read/write operations to the same device. If this applies to an attached device, the AHB bus master initiating the command sequence must ensure that no accesses are allowed from another AHB bus master until the command sequence has completed.

14.12.17 Cellular RAM

Cellular RAM devices have been designed to meet the growing memory and bandwidth demands of modern cellular phone designs. The devices have been designed with a “multi-protocol” interface to allow use of the devices with existing memory interfaces (i.e. by re-use of existing memory protocols). The supported interface protocols supported by Cellular RAM devices are:-

1. SRAM (Asynchronous Read and Write).
2. NOR Flash (Synchronous Burst Read, Asynchronous Write).
3. Synchronous (Synchronous Burst Read and Write).

In principle, when using previous versions of Memory Controller, the first two of the above modes (1 and 2 above) provided Cellular RAM support. For maximum performance, the Memory Controller now supports Synchronous Mode (3 above) for Cellular RAM (Synchronous Burst Read and Write).

As Cellular RAM Synchronous Mode consists of a Burst FLASH compatible Burst Read access, Cellular RAM support has been provided by enhancing the Burst FLASH interface by the inclusion of a Burst Write capability. For this reason Cellular RAM is treated as a special type of Burst FLASH device.

Cellular RAM support is selected by programming the desired region as cellular RAM via the BUSCONx.AGEN bit-field.

Synchronous Read Access

A Synchronous Cellular RAM Burst Read Access is compatible with a Burst FLASH Burst Read Access. As a result preceding sections applying to Burst FLASH devices apply and should be consulted for details of Cellular RAM Burst Read Accesses.

Synchronous Write Access

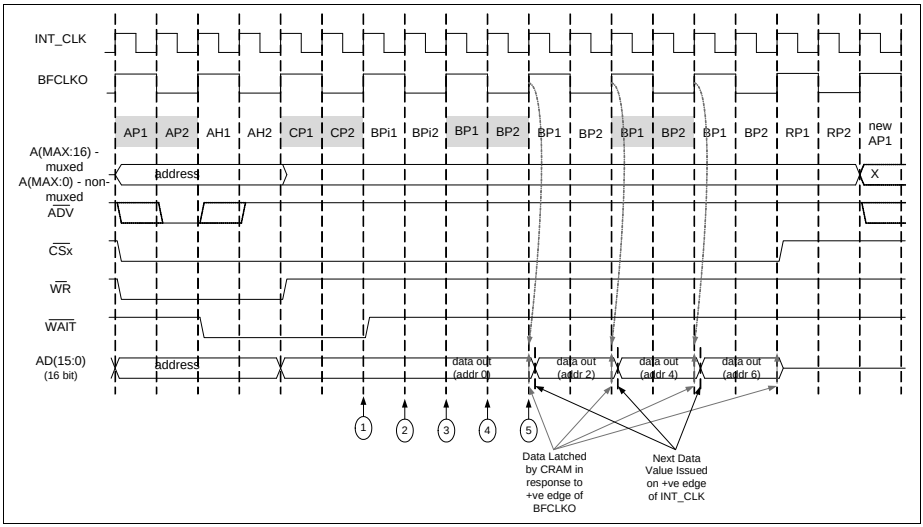


Figure 14-26 Burst Cellular RAM Burst Write Access (burst length of 4)

Note:

9. *The start of the cycle is synchronised to a +ve edge of the BFCLKO signal*
10. *The BFCLKO signal is used to clock the Cellular RAM devices*
11. *BFCLKO to Internal Clock frequency ratio can be programmed to 1:1, 1:2, 1:3 or 1:4. Each BFCLKO +ve edge is generated from a +ve edge of internal clock*
12. *Addresses show are "byte addresses"*
13. *ADV signal positioning is programmable via the EBSE bitfield in the BUSCON registers*

Figure 14-26 shows an example of a Cellular RAM burst write access.

*Note: **Figure 14-26** shows operation with a BFCLKO to EBU_CLK ratio of 1:2.*

The Start of the access cycle is the same as for a Synchronous Read access (see **Figure 14-24**) except that the $\overline{WR}$ signal is treated as an address phase signal (i.e. it is asserted active during the Address Phase and Address Hold Phase and is then deasserted). See **"Fujitsu FCRAM Support (burst write with WR active during data phase)" on Page 14-64** for alternative $\overline{WR}$ timing during burst write.

The remaining sequence is as follows (with reference to the figure above):-

1. At the positive edge of EBU_CLK labelled as '1' above the first Burst Phase starts. As the state machine is currently in the command phase, the interface samples the $\overline{WAIT}$ input. This is sampled as "active". By coincidence, in this example, the Cellular

RAM also deasserts it's WAIT output as a response to this clock edge to signal that it will start to take the data from the data bus on the BFCLKO rising clock edge after the next (i.e. the rising edge of BFCLKO labelled as '5' above) - this need not be the case.

2. At the positive edge of EBU_CLK labelled as '2' above the second programmed EBU_CLK period of the Burst Phase begins.
3. At the positive edge of EBU_CLK labelled as '3' above the Burst FLASH evaluates the WAIT sample from '1' above. As this sample was "active" the write data is not updated. As this clock edge is coincident with the end of a burst phase the WAIT input is resampled. The value of this new WAIT sample is "inactive".
4. At the positive edge of EBU_CLK labelled as '5' above the Burst FLASH again evaluates the WAIT sample from '3' above. As this sample was "in-active", and the edge is coincident with the end of a burst phase, the next data value is issued to the AD(15:0) pins and the next Burst Phase is started.

This process continues until all the data is written.

Fujitsu FCRAM Support (burst write with WR active during data phase)

The FCRAM device type can be supported in two ways. Later FCRAMs have a compatibility bit in the device configuration register which programmes the device to expect the WR signal to be active with the address and to be latched with the ADV signal. In this mode, FCRAM can be treated as an Infineon/Micron cellular RAM.

Alternatively, if a write is attempted to a region configured as a burst flash, the memory controller will generate a burst write with the WR signal asserted with the write data. This should be directly compatible with an FCRAM operating in its native mode.

14.12.18 Programmable Parameters

The following table lists the programmable parameters for burst flash accesses. These parameters only apply when the BUSCONx.AGEN parameter for a particular memory region is set for access to synchronous burst devices (page mode or otherwise).

Table 14-26 Burst Flash Access Programmable Parameters

Parameter	Function	Register
ADDRC	Number of cycles in Address Phase.	BUSAPx
AHOLDC	Number of cycles in Address Hold.	BUSAPx
CMDDELAY	Number of programmed Command Delay cycles.	BUSAPx
WAITRDC	Number of programmed wait states for read accesses.	BUSAPx

Table 14-26 Burst Flash Access Programmable Parameters (cont'd)

Parameter	Function	Register
WAITWRC	Number of programmed wait states for write accesses.	BUSWAPx
EXTDATA	Extended data	BUSAPx
RDRECOVC	Number of minimum recovery cycles after a read access when the next access is to the same region.	BUSRAPx
WRRECOVC	Number of minimum recovery cycles after a write access when the next access is to the same region.	BUSWAPx
RDDTACS	Number of minimum recovery cycles after a read access when the next access is to a different region.	BUSRAPx
WRDTACS	Number of minimum recovery cycles after a write access when the next access is to a different region.	BUSWAPx
WAIT	Sampling of $\overline{\text{WAIT}}$ input: OFF, SYNCHRONOUS, ASYNCHRONOUS or WAIT_CELLULAR_RAM	BUSCONx
FBBMSEL	Flash synchronous burst mode: CONTINUOUS or DEFINED (as in FETBLEN)	BUSCONX
FETBLEN	Synchronous burst length: SINGLE, BURST2, BURST4 or BURST8	BUSCONx
BFCMSEL	Flash Clock Mode, continuous or gated	BUSRCONx
EXTCLOCK	Frequency of external clock at pin BFCLKO: equal, 1/2, 1/3 or 1/4 of EBU_CLK	BUSCONx
EBSE	delay $\overline{\text{ADV}}$ output to improve hold margin	BUSCONx
ECSE	delay $\overline{\text{CS}}$, $\overline{\text{WR}}$ and write data outputs to improve hold margin	BUSCONx
LOCKCS	enable locked write sequences for this region	BUSWCONx

Table 14-26 Burst Flash Access Programmable Parameters (cont'd)

Parameter	Function	Register
FDBKEN	enable clock feedback to improve read data margins	BUSRCONx
BFSSS	disable second pipeline stage for clock feedback	BUSRCONx
DBA	disable alignment of read bursts on external bus	BUSRCONx
AAP	enable the "asynchronous address phase" mode.	BUSCONx
PORTW	memory port width	BUSRCONx

Note: datac is not used for burst write accesses

14.13 SDRAM Interface

The SDRAM interface supports:-

- 64 MBit (organized as 4 banks x 1M x 16)
- 128 MBit (as 4 banks x 2M x16)
- 256 MBit (as 4 banks x 4M x16) SDRAMs.
- 512 MBit (as 4 banks x 16M x16) SDRAMs.

The Memory Controller can support a single SDRAM region. To enable SDRAM support the **AGEN** fields of a single register pair of BUSRCON and BUSWCON must be set to "1000_b".

*Note: Programming the **AGEN** fields of multiple regions for SDRAM and connecting multiple SDRAMs will result in data corruption as the "page open" tags in the SDRAM controller will be applied indiscriminately to all connected devices.*

14.13.1 Features

- Compatible with mobile PC133/PC100 memories at 100 MHz (if maximum bus load is not exceeded).
- Mobile SDRAM support.
- Multibank support.
- Interleaved access support.
- Support for 64, 128, 256 and 512 MBit SDRAM devices.
- Auto-refresh mode support for power-down mode.
- Data types (16-bit bus): byte and half-word for single reads/writes and half-word for burst reads/writes.

- Power-on/mode-set sequence triggered by AHB write to SDRAM configuration register.
- Programmable refresh rate.
- Programmable timing parameters (row-to-column delay, row-precharge time, mode-register setup time, initialization refresh cycles, refresh periods).

14.13.2 Signal List

The following signals are used for the SDRAM interface:-

Table 14-27 SDRAM Signal List (16 bit support)

Signal	Type	Function
AD(15:0)	I/O	Data bus
AD(31:16)	O	Address bus
RD/$\overline{WR}$	O	Read and write control
CKE	O	Clock enable
CS(3:0)	O	Chip select
SDCLK0	O/I	External SDRAM Clock.
SDCLKI	I	External SDRAM Clock Feedback
RAS	O	Row Address Strobe for SDRAM accesses.
CAS	O	Column Address Strobe for SDRAM accesses.
DQM(1:0)	O	Data Qualifiers (output on $\overline{BC}$(1:0))

14.13.3 External Interface

The external interface can be directly connected to DRAM chips without any glue-logic. Special board layout and timing constraints may apply when additional memory/peripherals (in addition to SDRAM devices) are directly connected to the bus.

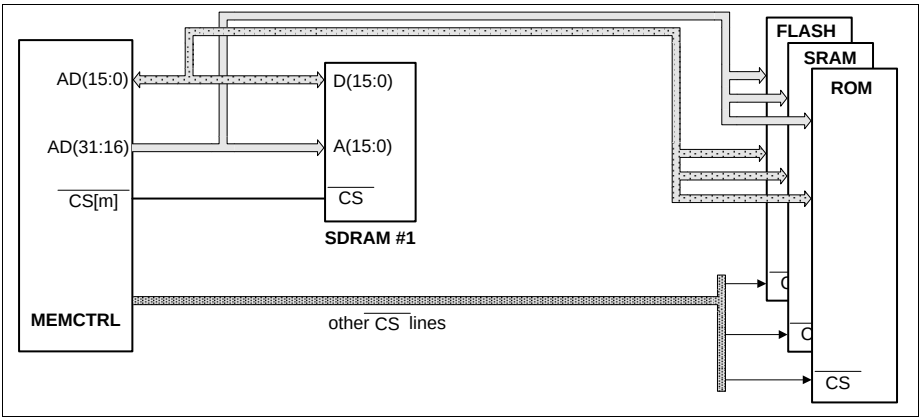


Figure 14-27 Connectivity for 16 bit SDRAM

14.13.4 External Bus Clock Generation

The Memory Controller uses the EBU_CLK clock to generate all external bus access sequences.

SDCLKO is required by SDRAM memories and the frequency of this output is controlled by the BUSRAP.EXTCLOCK field of the highest priority (lowest region number) region which has BUSRCON.AGEN set to "0b1000" which is the value used to select SDRAM. BUWAP.EXTCLOCK has no effect for SDRAM.

Unless documented elsewhere, all outputs to the external bus are generated of the rising edge of EBU_CLK. The SDCLKO signal (in 1:1 mode) is antiphase to EBU_CLK. This means that the SDRAM memory device sees control signal changes occur on the negative clock edge. The clock generation logic is constructed so that this relationship is maintained for the other clock ratios.

Table 14-28 EXTCLOCK to clock ratio mapping

EXTCLOCK value	SDCLKO divide ratio
00	1:1
01	1:2
10	1:4
11	1:4

14.13.5 SDRAM Characteristics

SDRAMs are synchronous DRAMs with burst read/write capability which are controlled by a set of commands at the pins $\overline{CS}$, $\overline{RAS}$, $\overline{CAS}$, $\overline{WE}$, DQM and A10. As for standard DRAMs, a periodic refresh must be performed.

SDRAM devices are subdivided into “banks”. Each bank is subdivided into a number of “rows[1)]”. Each row is, in turn, subdivided into a number of “columns”. The number of banks and the size of a row varies from one SDRAM device to another. A specific location (half-word) within a device is specified by supplying a bank, row and column address. Devices supported by Memory Controller must conform to the following criteria:-

- **Number of Banks:** 2 or 4 only.
- **Row Size:** 256, 512 or 1024 only.

SDRAM devices produce high speed data transfer rates by use of the bank and row architecture. When an initial access is made to a specific row within a specific bank then Memory Controller must issue a “row” address to specify which row in which bank is to accessed. In response to this the SDRAM device loads the entire row to a local (high speed) buffer area. At this point (i.e. when the local buffer associated with a bank contains data from the main SDRAM array) the bank is said to be “open”. Memory Controller then issues a “column” address to specify which location(s) within the row are to be accessed. Subsequent accesses to locations within the same row can then be performed at high speed (with Memory Controller supplying only a column address) since the appropriate data is already contained within the local buffer and there is no requirement for the SDRAM to fetch data from the main SDRAM array. Prior to accessing a location in a different row Memory Controller must issue a “precharge” command so that the local buffer is written back to the main SDRAM array.

An SDRAM device provides a local buffer for each bank within the device, thus it is simultaneously possible for each of the banks to be “open”, this is termed “Multibanking”. Multibanking is supported in order to allow interleaved bank accesses. Comparison of banks is done prior to initiating external memory accesses (see **Section 14.13.13**).

14.13.6 Supported SDRAM commands

Table 14-29 lists the supported SDRAM commands, how they are triggered and which signals are activated:-

1) Previous Memory Controller documentation uses the term “page” to refer to a “row”. Where possible this has been changed to reflect the more commonly used term “row”.

Table 14-29 Supported SDRAM commands

Command	Event	CKE (n-1)	CKE (n)	CS	RAS	CAS	RD/ WR	See Table 14-41 for Memory Controller pins			
								A12[1)] A11	A10	A (9:0)	BA (1:0)
Device deselect	region not sel'ted	H	-	H	-	-	-	-	-	-	-
Nop	idle	H	-	L	H	H	H	-	-	-	-
Bank activate	open a closed bank	H	-	L	L	H	H	valid address			
Read	read access	H	-	L	H	L	H	valid addr	L	valid address	
Write	write access	H	-	L	H	L	L	valid addr	L	valid address	
Read with autoprecharge	read access	H	-	L	H	L	H	valid addr	H	valid address	
Write with autoprecharge	write access	H	-	L	H	L	L	valid addr	H	valid address	
Precharge selective	bank or row miss	H	-	L	L	H	L	-	L	-	bank
Precharge all	refresh is due or going into power down	H	-	L	L	H	L	-	H	-	-
Autorefresh	refresh is due, after precharge all is done	H	H	L	L	L	H	-	-	-	-

Table 14-29 Supported SDRAM commands (cont'd)

Command	Event	CKE (n-1)	CKE (n)	CS	RAS	CAS	RD/ WR	See Table 14-41 for Memory Controller pins			
								A12[1] A11	A10	A (9:0)	BA (1:0)
Self refresh entry	going into power down after precharge all is done	H	L	L	L	L	H	-	-	-	-
Self refresh exit	coming out of power down	L	H	H	-	-	-	-	-	-	-
Mode register set	during initialization	H	-	L	L	L	L	valid mode (see register SDRMOD)			00_B
Extended Mode register set	during initialization	H	-	L	L	L	L	valid mode (see register SDRMOD)			10_B[2]

1) A12 is required by larger memories

2) 10_B is default value for SDRAM. This can be changed using the SDRMCON.**XBA** field

14.13.7 SDRAM device size

Memory Controller supports SDRAM's with the following sizes:-

- **Size**[1]**:** 64MBit, 128MBit, 256MBit and 512MBit.

14.13.8 Power Up Sequence

During power-up the SDRAM should be initialized with the proper sequence. This includes the requirement of bringing up the VDD, VDDQ and the stable clock (minimum 200 µs before any accesses to SDRAM) and $\overline{CS}$ remains inactive.

1) In addition verified support is limited to specific SDRAM device geometries (number of banks and row size). Support for other sizes/geometries may be possible but this has not been verified.

14.13.9 Initialization sequence

SDRAMs must be initialized before being used. Application of power must be followed by 200 µs pause (timed by software) with a stable clock. Then a Precharge All Banks command must be issued. Following this, the device must go through Auto Refresh Cycles (the number of refresh commands is programmable through **Crfsh** in SDRMCON registers and the number of NOP cycles in between is programmable through **Crc**). At the end of it, the Mode Register must be programmed through the address lines. Following that some number of NOP cycles programmable through **Crsc** in SDRMCON.

Note: This sequence will be referred to as a "cold start",and is necessary when both the memory and the memory controller have just had power applied. Conversely a "warm start" will be required when the memory controller has just been powered up but data has been retained in the external memory by the use of self refresh mode.

The SDRAM controller will power up with the SDRAM clock set to gated mode and CKE high. However until an EBU region is configured for SDRAM by setting BUSRCON.AGEN the SDRAM clock and CKE pins will be allocated for GPIO.

Care must be taken during software configuration of Memory Controller to ensure the correct SDRAM initialization sequence is generated for both cold start and warm start.

The recommended sequence for Memory Controller register initialization after a cold start when using SDRAM devices is as follows:-

1. Write to SDRMREF to set CKE high. (**SELFREX** = '1') but leave all refresh fields at 0 to disable auto refresh. This will maintain CKE high when BUSRCON is written in the next step
2. Write to BUSRCON to define which region has SDRAM connected and the required divide ratio for the SDRAM clock.
3. Write to SDRMCON to configure the controller for the attached SDRAM device(s) and to enable the SDRAM clock. (**SDCMSEL**=0.)
4. All other Memory Controller registers except SDRAM specific registers (i.e. other than those listed below).
5. Wait for 200µs (or the appropriate initialization delay required by the attached device)
6. Write to SDRMOD with the "**COLDSTART**" bit set to write the mode register values to the SDRAM mode register.
7. Write to SDRMREF to configure refresh rate.

The recommended sequence for Memory Controller register initialization after a warm start when using SDRAM devices is as follows:-

1. Write to BUSCON to define which region has SDRAM connected and the required divide ratio for the SDRAM clock.
2. Write to SDRMCON to configure the controller for the attached SDRAM device(s) and to enable the SDRAM clock. (**SDCMSEL**=0.)

3. Write to SDRMREF to set CKE high. (**SELFREX**='1') but leave all refresh fields at 0 to disable
4. All other Memory Controller registers except SDRAM specific registers (i.e. other than those listed below).
5. Write to SDRMOD with the "**COLDSTART**" bit cleared to update the mode register values.
6. Write to SDRMREF to configure refresh rate.

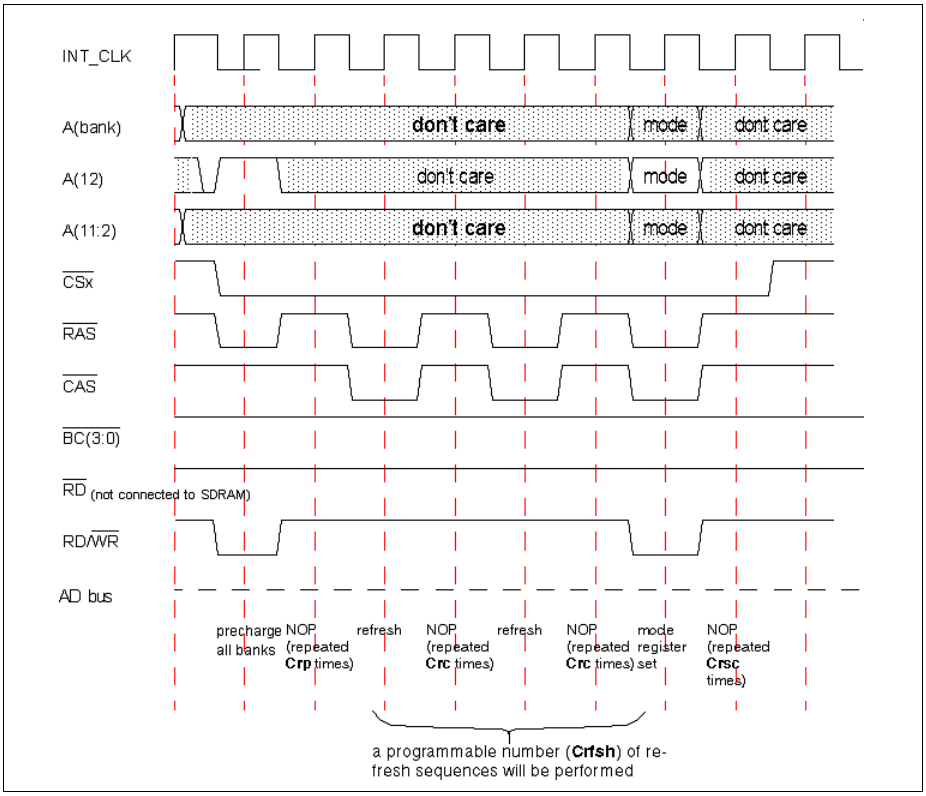


Figure 14-28 SDRAM Initialization

The sequence is triggered by a write to the SDRAM mode register SDRMOD. A region having **AGEN** in BUSCONx set to '1000_B' will be configured with the mode from SDRMOD. While this sequence is being executed, **sdrmbusy** flag in the SDRMSTAT status register will be set accordingly.

Note: As no other accesses are permitted in the current implementation while the SDRAM initialization sequence is running, it will not be possible to poll the sdrmbusy bit at '1' unless there has been a failure in the controller logic.

The user has to make sure that the SDRAM is programmed in the following way:

Table 14-30 SDRAM Mode Register Setting

Field	Value	Meaning	SDRMOD Position	Corresponding Address Pins
Burst length	"100" “011” “010” “001” "000"	bursts of length 16 bursts of length 8 bursts of length 4 bursts of length 2 bursts of length 1	**burstl**[2:0]	A[2:0]
Burst type	‘0’	sequential bursts	**btyp**[3]	A[3]
$\overline{\text{CAS}}$ latency	“001” “010” “011” “1xx”	reserved latency 2 latency 3 reserved	**caslat**[6:4]	A[6:4]
Operation Mode	all ‘0’	burst read and burst write	**opmode** [13:7]	A[12:7]

The Memory Controller uses the $\overline{\text{CAS}}$ latency value and burst length to adjust the burst read timing. All other fields have no influence on the Memory Controller, which means only single value is accepted for those fields.

The complete initialization sequence described will only be issued on the first write (since reset) to the SDRMOD register with the **COLDSTART** field set to logic '1'. On subsequent writes with the **COLDSTART** field set to logic '1', the SDRAM device does not need to be initialized, so a simple mode register set command can be issued to refresh the contents of the registers in the SDRAM. A precharge-all command needs to be issued to the SDRAM before this can happen.

An initialization sequence will write to both the mode register and the extended mode register (if the extended mode register has been enabled).

A write to the SDRMOD register with the **COLDSTART** cleared will update the EBU register and will also write to the configuration registers of the SDRAM but will not execute the refresh cycles which are part of the full initialization required at cold start.

14.13.10 Mobile SDRAM Support

Mobile SDRAMs include an “Extended Mode Register”. This is accessed using a similar mechanism to the existing PC-133 Mode Register but with an additional select code on the SDRAM device BA pins.

The SDRMOD.**XBA** bits are used to select “Mobile” SDRAM support for each of the SDRAM devices. If this field is non-zero, then the Extended Mode Register will be automatically written during the Initialization phase (immediately after the “standard” Mode Register write). In addition writes to the Extended Mode Register(s) will be triggered by writes to the SDRMOD register (i.e. whenever the “standard” Mode Register is written). The SDRMOD.**XOPM** bit-field is used to program the value that is to be written to the Extended Mode Register. The SDRMOD.**XBA** bit-field is used to program the logic levels asserted on the device BA(1:0) pins (i.e. to program the specific command used to access the extended mode register).

The **XBA** field contains four bits even though there are only two bank address connections to SDRAM devices. This is because, for normal accesses, the Memory Controller assumes that the SDRAM bank address lines are connected to the least-significant, available address lines (see **Table 14-41** for details). This means that the bank address inputs to the SDRAM can be connected to either A[13:12], A[14:13] or A[15:14] depending on the number of rows in the connected SDRAM. The **XBA** field value will be output on A[15:12] and the **XOPM** field will be output on A[11:0] allowing all possible SDRAM connections to be handled correctly.

Note: In order to cater for possible future device variations the Memory Controller allows the user to select the logic levels issued on the BA(1:0) pins during an Extended Mode Register. Care should be taken in programming this bit field since it is possible to generate an unwanted “standard” Mode Register write by use of this bit field.

14.13.11 Burst Accesses

SDR Operation

The Memory Controller supports burst lengths of 1, 2, 4, 8 and 16. Bursts of other lengths are supported but are implemented using data-masking. Burst length 16 is currently not supported by available SDR memories.

14.13.12 Short Burst Accesses

SDR Operation

The Memory Controller can be configured to generate SDRAM bursts lengths of either one, four or eight via the SDRMOD.**BURSTL** bit fields. When configured for burst lengths of four or eight the interface will use data masking to support shorter write accesses. However, when configured for a burst length of one data masking is not used. **Figure 14-29** shows how short burst write accesses are handled. During the write access data masking is activated (with zero clock latency) to prevent unwanted write operation. Data masking is activated through the $\overline{BCx}$ outputs (connected to $\overline{DQM}$) during a write cycle.

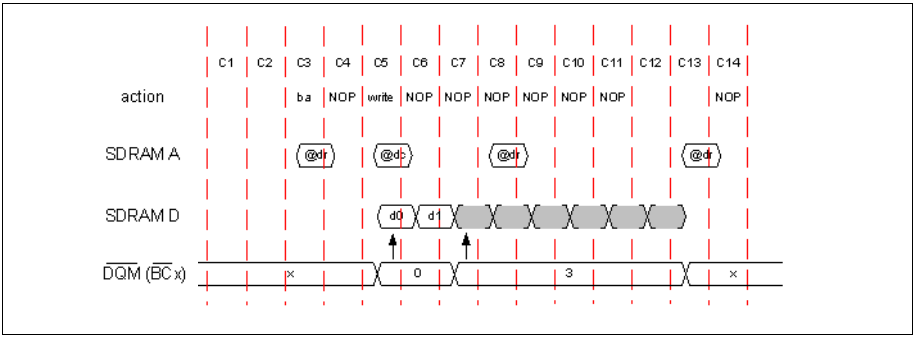


Figure 14-29 Short Burst Write Access through Data Masking

The figure shows how a two beat burst write is translated to an eight beat burst write with data masking. During the first two data cycles (C5 and C6) the $\overline{BC0}$ and $\overline{BC1}$ outputs are driven low to cause the SDRAM device to write the required data. In cycle C7 the $\overline{BC0}$ and $\overline{BC1}$ outputs are driven high to mask subsequent data writes.

14.13.13 Multibanking Operation

The design supports up to 4 banks being simultaneously open for an SDRAM region. This means that for each bank Memory Controller must track the status of the bank (“open” or “closed”) and the last row address issued to that bank. This allows the Memory Controller to determine whether each access is a “row hit” or a “row miss”.

- A “row hit” means that the access can be serviced by data which is already in the SDRAM local data buffer of the specified bank (i.e. the bank is open and the last row address that was issued to the bank matches the row address for the current access - see **Section 14.13.5**). In this case Memory Controller can then proceed with one of the access commands without having to close the specified bank.

- A “row miss” means that the access cannot be serviced from the SDRAM local data buffer (i.e. the specified bank is closed, or the last row address that was issued to the bank does not match the row address of the current access). In this case Memory Controller must close the specified bank and then re-open it with the new row address.

The Memory Controller must be configured so that it can detect “row hits” and “row misses” for different SDRAM configurations (number of banks/row size). This allows the Memory Controller to properly issue the appropriate SDRAM commands to allow up to four SDRAM rows to be kept open simultaneously (i.e. one row open in each bank assuming a four bank device). To perform this Memory Controller maintains two items of data for each bank:-

1. Bank status (1 bit): “open” or “closed” (upon reset all of these bits are preloaded with “closed”).
2. Last row address (up to 18 bits).

These two items are referred to as a “bank tag”. In order to maintain these bank tags The Memory Controller must be made aware of:-

- The AHB address range that defines which bank is being accessed.
- The AHB address range that defines which row is being accessed.

These AHB address ranges change according to the geometry of an SDRAM device. **Table 14-41** shows some examples of how banks and rows are determined from the address bits. This configuration is performed by means of the “Bank Mask” and “Row Mask”. For example, if the SDRAM is configured as:-

- 16-bit wide
- 4 banks in the device
- 8192 rows
- row size of 512

The bank to be accessed is determined by bits 24 and 23 of the address (Address[24:23]). Each open bank has an associated open row, and for our example the row tag is Address[22:10].

Each time there is a new AHB request, the address is compared against the appropriate bank tag. After one clock cycle there will be two decisions to make. If the current access is targeted to an SDRAM region(s) then Memory Controller must determine whether the requested address is a row hit or row miss.

14.13.14 Bank Mask

The "SDRMCON.**BANKM**" (bank mask) bit-field must be set to the appropriate value to set the AHB address bit range used to detect which bank is being accessed. The value to be written to this bit-field is determined by the device size setting (see **“SDRAM device size” on Page 14-71**) and the number of banks in the device.

- When the device has 2 banks then the "**BANKM**" value must be set to include the most significant address bit of the AHB address range occupied by the SDRAM device (i.e. region) - see **Table 14-41**.
- When the device has 4 banks then the "bankm" value must be set to include the most significant two address bits of the AHB address range occupied by the SDRAM device (i.e. region) - see **Table 14-41**.

The following settings should be used:-

Table 14-31 "BANKM" Selection

"BANKM" setting	AHB Address Bits used to determine bank hit/miss	Comment
0	none	Reserved - do not use (default after reset).
1	A_{AHB}[21 to 20]	Bank Size = 8MBit
2	A_{AHB}[22 to 21]	Bank Size = 16MBit
3	A_{AHB}[23 to 22]	Bank Size = 32MBit
4	A_{AHB}[24 to 23]	Bank Size = 64MBit
5	A_{AHB}[25 to 24]	Bank Size = 128MBit
6	A_{AHB}[26 to 25]	Bank Size = 256MBit.
7	A_{AHB}[27 to 26]	not supported for SDRAM/DDRAM

14.13.15 Row Mask

The "SDRMCON.**ROWM**" bit-field must be set to the appropriate value to set the AHB address range used to detect row hit/miss. The value to be written to this bit-field is determined by the device size setting (see above), the number of banks in the device and the number of rows in the device.

The following "**ROWM**" settings should be used:-

Table 14-32 “ROWM” Selection

“ROWM” setting	AHB Address Bits used to determine row hit/miss	Comment
0	none	Reserved - do not use (default after reset) (Always generate row miss, may cause invalid SDRAM command sequences).
1	A_{AHB}[n to 9]	Row size = 256 x 16-bit.
2	A_{AHB}[n to 10]	Row size = 512 x 16-bit, 256 x 32 bit.
3	A_{AHB}[n to 11]	Row size = 1024 x 16-bit, 512 x 32 bit.
4	A_{AHB}[n to 12]	Row size = 1024 x 32 bit.
5	A_{AHB}[n to 13]	Not appropriate.
6	reserved;	
7	reserved;	

It can be seen that the **ROWM** bit-field only has an effect on the low-end of the address range used to determine the row address (i.e. for use in comparison of the AHB address with the address stored in the bank tag). The upper end of the comparison is determined by the **BANKM** setting. "n" is therefore (**BANKM**+18)

Decisions over “row hit”

When a row hit occurs, Memory Controller can continue the access operation without updating the stored bank tag.

A row miss unfortunately can result in several other activities.

- If the row miss is due to the bank being closed then Memory Controller does not have to issue a precharge operation but can activate the bank, update the appropriate bank tag to reflect the new bank status (i.e. to “open” with the specified row address) and continue the access operation.
- If the row miss occurs on an open bank then Memory Controller has to close the current bank, (i.e. do a precharge). This is then followed by (re-)activating the bank, updating the appropriate bank tag to reflect the new bank status (i.e. remaining “open” with the new row address) and continuing the access operation.

Table 14-33 lists the activities on a cycle by cycle basis.

Table 14-33 Cycle by cycle activities of multibanking operation

Cycle n	Cycle n+1	Cycle n+2	Cycle n+3
`Comparing` the AHB address with the stored bank tags.	**row_hit:**		
	Continue with `read` or `write` command.	not relevant	not relevant
	row_miss and bank_open:		
	Issue `Precharge` to the specific bank and change the bank tag to reflect the new row address.	Insert `idle` cycles (repeatable) to satisfy $t_{RP.}$	Continue with `bank activate` command, etc.
	row_miss and bank_closed:		
	Issue `Bank Activate` to the specific bank and change the bank tag to reflect the new status (“open”) and row address.	not relevant	not relevant

14.13.16 Banks Precharge

The system is required to precharge a bank under one of the following conditions:

1. When the next access to a bank is to a different row to the previous access within the bank.
2. When an AHB request cannot be completed before the row active time $t_{RAS\,max}$ is due, then the bank must explicitly be closed and opened again for the current request. Since $t_{RAS\,max}$ (of the order of 100 µs) is usually much greater compared to the refresh period (distributed refresh is in order of 15 µs for 4096 rows) this is generally fulfilled by systematically carrying out refresh to the SDRAMs (see ‘d’ below).
3. Accompanying a row miss is also naturally a selective bank precharge operation, as mentioned previously.
4. All banks must also be pre-charged, when a refresh cycle is due as explained next. See **Section 14.13.17**.
5. All banks must also be pre-charged, prior to issuing Self Refresh Entry command. See **Section 14.13.18**.

Activities in (1) and (3) are carried out as a result of the address comparison explained in **Section 14.13.13**. Activity (2) and (4) are covered by refresh timer.

14.13.17 Refresh Cycles

It is required within certain time limit that the devices must be refreshed. Prior to being refreshed the devices must be pre-charged as mentioned above.

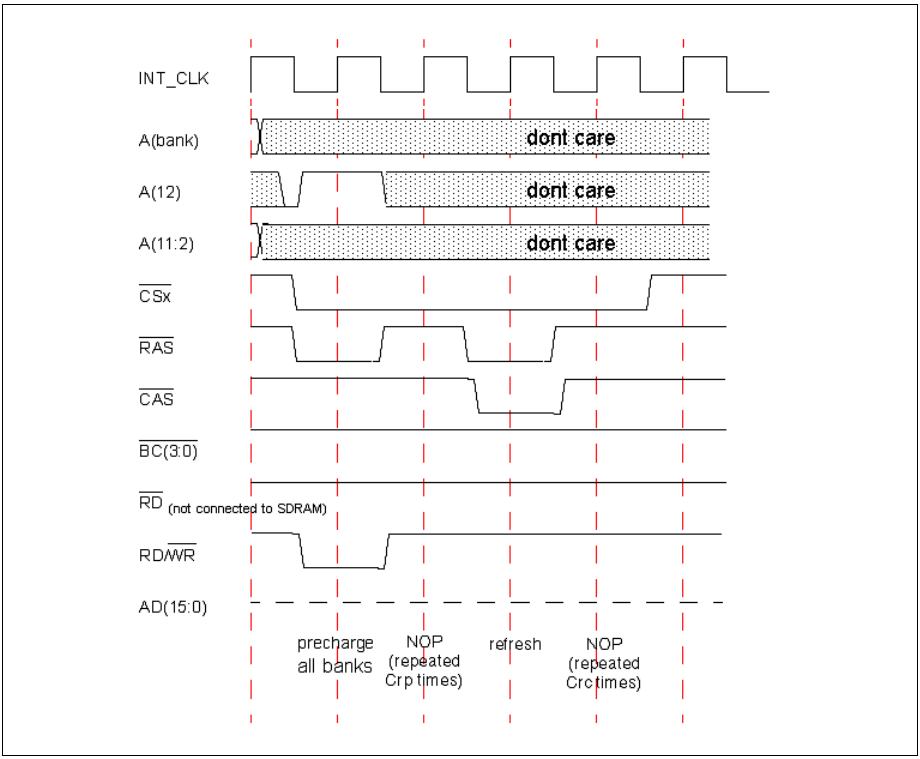


Figure 14-30 SDRAM Refresh

This sequence is periodically triggered by an internal refresh counter with programmable rate set using the **ERFSHC** and **refreshc** fields in the SDRMREF register. These fields are combined to create an eight bit value (**ERFSHC** as MSBs). This value is then multiplied by 64 and used as the number of EBU_CLK cycles between refresh operations being requested.

All SDRAM banks will be pre-charged before the refresh sequence can be started. The specific refresh command being issued is Auto Refresh (CBR) command, in which the device keeps track of the row addresses to be refreshed. The number of this command being issued for each refresh operation is programmable through **refreshr** in SDRMREF.

A refresh request has precedence over a AHB access to SDRAM, i.e. if both occur at the same time the refresh sequence is entered and the AHB access is delayed. A refresh error occurs when a previous refresh request has not been satisfied yet and another refresh request occurs and an error flag (**referr**) in the SDRSTAT status register will be

set accordingly. This error flag can be cleared by writing to SDRMCON respective to the appropriate address region.

14.13.18 Self-Refresh Mode

SDRAM devices provide a Self-Refresh Mode. In this mode the SDRAM automatically performs internal refresh sequences in response to an on-chip timer. Self Refresh Mode is entry command is asserted with RAS, CAS, and CKE low and WE high. In Self-Refresh Mode all external control signals except CKE (but including the clock) are disabled). Returning CKE to high enables the clock and initiates the Self-Refresh Mode exit operation. After the exit command, at least one tRC delay is required prior to any access command.

Low Power SDRAMs provide additional power saving features such as:-

Programmable refresh period of the on-chip timer such that the refresh period can be optimized (maximized) by taking the device operating temperature in to account.

Partial array self-refresh mode such that only selected banks will be refreshed. Data written to the non-selected banks will be lost (due to lack of refresh to the bank) after a period defined by t_{REF}.

These additional features are programmed by issuing an Extended Mode Register write (see **Section 14.13.10**).

To activate Self-Refresh Mode, software must write ’1’ to bit **selfren** in SDRMREF register. Memory Controller will then:

1. precharge all the banks, and
2. issue a self refresh command (see **Table 14-29**) to all SDRAM devices (regardless whether the device belongs to access type0 or type1).

In completion of this command all SDRAM devices will ignore all inputs but CKE signal. The read-only bit **selfrenst** reflects the status of issuing this command. When the command is completed, power-down can be safely entered. The devices would perform low-current self refresh during the power down. When exiting from power-down and before doing any accesses to SDRAM, software must write ’1’ to bit **selfrex** in SDRMREF registers. Memory Controller will then assert the CKE signal for all the SDRAM devices to get out of the self-refresh mode. The read-only bit **selfrexst** reflects the completion of this command, upon which an access to SDRAMs can be performed. As the SDRAM controller has CKE set low after reset a ’1’ must be written to **SELFREX** as part of the initialization sequence to enable CKE. See **Section 14.13.9**.

Two additional fields affect the method the memory controller uses to exit self refresh.

1. After CKE is taken high (self refresh exit command), a single NOP cycle is generated. The SDRMREF.**ARFSH** field is checked. If set to one a single auto refresh command is output to the memory.
2. After step 1, the SDRMREF.**SELFREX_DLY** field is checked. If the field is non-zero, the value in the field is used to generate a sequence of NOP instructions to the

memory. This allows between 1 and 255 NOPs to be inserted before the device sees a non-null command.

For predictable operation of the device during warm start, both the SDRMREF.**ARFSH** and SDRMREF.**SELFREX_DLY** fields should be set to 0.

14.13.19 SDRAM Addressing Scheme

SDRAM devices use a multiplexed address issued as “bank”, “row” and “column” addresses. The column address determines which location is being accessed within a row. The row address determines which row is being accessed within a bank. Since row sizes can differ from one SDRAM device to another it is necessary to provide a programmable address multiplexing scheme. Selection of the multiplexing scheme is via the "SDRMCON.**AWIDTH**" bit-field.

The "SDRMCOM.**AWIDTH**" bit-field must be set to the appropriate value to set the address multiplexing (i.e. row/column) to be used to issue the address (and command) to the SDRAM. The value to be written to this bit-field is determined by the device row size.

Table 14-34 Selection of address multiplexing

AWIDTH	Row size	16 bit DRAM
00_B	Reserved; do not use	-
01_B	256 words	$A_{AHB}(8:0)$
10_B	512 words	$A_{AHB}(9:0)$
11_B	1024 words	$A_{AHB}(10:0)$

When performing byte writes, byte selection is handled via the $\overline{BC}(3:0)$ signals which used to generate DQM signals during an SDRAM access.

Row Address Multiplexing

When a row address is issued (with the above specified settings):-

- The most significant Memory Controller address outputs not required by the SDRAM (A[24:16]) are driven with '0' (zero).
- The least significant sixteen address outputs (AD[31:16]) are driven with the (correctly aligned) row address. This address alignment is performed according to the device row size specified by the "awidth" bit-field (see **Table 14-35**).

During the issue of a row address the following address multiplexing is used:-

Table 14-35 Row address generation for 16 bit SDRAM

AWIDTH	16 bit, PORTW="01" Address Generation (at Memory Controller pins)	Comment
01_B	A[24:16] = '0' AD[31:16] = A_{AHB}[24:9]	Row size is 256 words.
10_B	A[24:16] = '0' AD[31:16] = A_{AHB}[25:10]	Row size is 512 words.
11_B	A[24:16] = '0' AD[31:16] = A_{AHB}[26:11]	Row size is 1024 words.

Column Address Multiplexing

When a column address is issued (with the above specified settings):-

- The most significant Memory Controller address outputs (A[24:16]) are driven with '0' (zero).
- The least significant ten address outputs (A[9:0]) are driven with the (correctly aligned) column address. This address alignment is a one bit right shift of the AHB address if the SDRAM is 16 bit or a two bit shift if the SDRAM is 32 bit.
- Address output ten (A[10]) is driven with a "command" value (used by the SDRAM in conjunction with the other control signals to determine which command is to be executed.
- The remaining address outputs (A[15:11]) are driven with the appropriate AHB addresses (matching the multiplexing scheme for these pins during a row address shown in **Table 14-35** above) to ensure that consistent row selection is achieved (i.e. the row information must be the same regardless of whether a row or column address is being issued).

During the issue of a column address the following address multiplexing is used:-

Table 14-36 Column Address Generation for 16 bit SDRAM

AWIDTH	16 bit, PORTW="01"Address Generation (at Memory Controller pins)
01_B	A[24:16] = '0' A[15:11] = A_{AHB}[24:20] A[10] = Command A[9:0] = A_{AHB}[10:1]
10_B	A[24:16] = '0' A[15:11] = A_{AHB}[25:21] A[10] = Command A[9:0] = A_{AHB}[10:1]
11_B	A[24:16] = '0' A[15:11] = A_{AHB}[26:22] A[10] = Command A[9:0] = A_{AHB}[10:1]

Bank Address Multiplexing

A bank address is always issued whenever either a row or column address is issued. As a result the Bank Address multiplexing must be the same regardless of whether a row or column address is being issued. From **Table 14-35** and **Table 14-36** it can be see that, with the same "awidth" value, the address multiplexing for the address output pins A[15:11] is the same regardless of the type of address being issued.

Note: Memory Controller uses it's address output pins to select the bank being accessed (rather than having dedicated Bank Select outputs).

The SDRAM Bank Select pin(s) (BA0 and BA1) must be connected to the appropriate Memory Controller A[15:11] address pins to ensure that they are driven by the appropriate AHB Address (according to the SDRAM geometry). In practice this means that BA0 and BA1 (if present) must be wired to the sequential Memory Controller address outputs above those which are connected to the highest SDRAM address input. As a result, since Memory Controller only supports devices with more rows than columns, this can be determined directly from the number of rows in the device as follows:-

Table 14-37 Bank Address to Memory Controller Address Pin Connection

Number of Rows	BA0	BA1[1)]
2048	A[11]	A[12]
4096	A[12]	A[13]

Table 14-37 Bank Address to Memory Controller Address Pin Connection (cont'd)

Number of Rows	BA0	BA1[1)]
8192	A[13]	A[14]
16384	A[14]	A[15]

1) For devices with four banks only.

The following table shows all the multiplexing schemes discussed in the previous sections:

Table 14-38 SDRAM Address Multiplexing Scheme

Port Width	Address Type	Pin Usage	Mode
16-bit (**PORTW** = 01_B)	column address	Memory Controller Pins A(9:0) := A_{AHB}(10:1) Memory Controller Pins A(10) := CMD	all modes
		Memory Controller Pins A(15:11) := A_{AHB} (26:22)	awidth = "11"
		Memory Controller Pins A(15:11) := A_{AHB} (25:21)	awidth = "10"
		Memory Controller Pins A(15:11) := A_{AHB} (24:20)	awidth = "01"
	row address	Memory Controller Pins A(15:0) := A_{AHB}(26:11)	awidth = "11"
		Memory Controller Pins A(15:0) := A_{AHB}(25:10)	awidth = "10"
		Memory Controller Pins A(15:0) := A_{AHB}(24:9)	awidth = "01"

The Memory Controller requires the SDRAM to be configured to read / write bursts of length 1, 4 or 8. A burst shorter than the programmed burst (e.g. a single access) can be generated by masking the excess data phases with DQM. Due to the wrap around feature of the SDRAMs a burst has to start at certain addresses to prevent the wrap around (a burst must not cross an address modulo 8*4). This guarantees also that the internal row boundaries of the SDRAMs will not be crossed by any burst access. For bursts that are 16-bit wide transfers, AHB address A[0] of any burst address must be 0. For bursts that are 16-bit wide transfers, AHB address A[1:0] of any burst address must be 0. This restriction is currently enforced by the AHB interface logic which will split any unsupported bursts into 32 bit transfers

Table 14-39 16-bit Burst Address Restrictions, A[0] = "0"

Burst Length	AHB Address A[4:1]	SDRAM Burst Address Generation
1	any	single access
2	"--0"	0 -> 1
4	"--00"	0 -> 1 -> 2 -> 3
8	“-000” (0)	0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7
16	"0000"	0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 - 13 -> 14 -> 15

Table 14-40 32-bit Burst Address Restrictions, A(1:0) = "00"

Burst Length	AHB Address A(4:2)	SDRAM Burst Address Generation
1	any	single access
2	"--0"	0 -> 1
4	"-00"	0 -> 1 -> 2 -> 3
8	“000” (0)	0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7

The following SDRAM types can be connected to Memory Controller:

Table 14-41 Supported Configurations for 16-bit wide data bus (Part 1)

SDRAM portw = 01_B (16-bit)				Memory Controller Pins					
			A(15)	A(14)	A(13)	A(12)	A(11)	A(10)	A(9:0)
1GBit	SDRAM Pins		BA(1)	BA(0)	A(13)	A(12)	A(11)	A(10)	A(9:0)
	64Mx 16	row	RA(15) /BA(1)	RA(14) / BA(0)	RA(13)	RA(12)	RA(11)	RA(10)	RA(9:0)
		col						CMD	CA(9:0)
512MBit	SDRAM Pins			BA(1)	BA(0)	A(12)	A(11)	A(10)	A(9:0)
	32Mx 16	row		RA(14) / BA(1)	RA(13) / BA(0)	RA(12)	RA(11)	RA(10)	RA(9:0)
		col						CMD	CA(9:0)
256MBit	SDRAM Pins			BA(1)	BA(0)	A(12)	A(11)	A(10)	A(9:0)
	16Mx 16	row		RA(14) / BA(1)	RA(13) / BA(0)	RA(12)	RA(11)	RA(10)	RA(9:0)
		col						CMD	CA(8:0)

Table 14-41 Supported Configurations for 16-bit wide data bus (Part 1) (cont'd)

SDRAM portw = 01_B (16-bit)				Memory Controller Pins						
			A(15)	A(14)	A(13)	A(12)	A(11)	A(10)	A(9:0)
128MBit	SDRAM Pins				BA(1)	BA(0)	A(11)	A(10)	A(9:0)
	8Mx 16	row			RA(13) / BA(1)	RA(12) / BA(0)	RA(11)	RA(10)	RA(9:0)
		col						CMD	CA(8:0)
64MBit	SDRAM Pins				BA(1)	BA(0)	A(11)	A(10)	A(9:0)
	16Mx 4	row			RA(13) / BA(1)	RA(12) / BA(0)	RA(11)	RA(10)	RA(9:0)
		col						CMD	CA(9:0)
	8Mx8	row			RA(13) / BA(1)	RA(12) / BA(0)	RA(11)	RA(10)	RA(9:0)
		col						CMD	CA(8:0)
	4Mx 16	row			RA(13) / BA(1)	RA(12) / BA(0)	RA(11)	RA(10)	RA(9:0)
		col						CMD	CA(7:0)
16MBit	SDRAM Pins						BS	A(10)	A(9:0)
	4Mx4	row					RA(11) / BA(0)	RA(10)	RA(9:0)
		col						CMD	CA(9:0)
	2Mx8	row					RA(11) / BA(0)	RA(10)	RA(9:0)
		col						CMD	CA(8:0)
	1Mx 16	row					RA(11) / BA(0)	RA(10)	RA(9:0)
		col						CMD	CA(7:0)

Table 14-42 Supported Configurations for 16-bit wide data bus (Part 2)

SDRAM portw = 01_B (16-bit)			Multiplexed AHB Address	AWIDTH setting
1GBit	SDRAM Pins			
	64Mx 16	row	A(26:11)	11
		col	A(26:25), A(10:1)	
512MBit	SDRAM Pins			
	32Mx 16	row	A(25:11)	11
		col	A(25:23), A(10:1)	

Table 14-42 Supported Configurations for 16-bit wide data bus (Part 2) (cont'd)

SDRAM portw = 01_B (16-bit)			Multiplexed AHB Address	AWIDTH setting
256MBit	SDRAM Pins			
	16Mx 16	row	A(24:10)	10
		col	A(24:23), A(9:1)	
128MBit	SDRAM Pins			
	8Mx 16	row	A(23:10)	10
		col	A(23:22), A(9:1)	
64MBit	SDRAM Pins			
	16Mx 4	row	A(24:11)	11
		col	A(24:23), A(10:1)	
	8Mx8	row	A(23:10)	10
		col	A(23:22), A(9:1)	
	4Mx 16	row	A(22:9)	01
		col	A(22:21), A(8:1)	
16MBit	SDRAM Pins			
	4Mx4	row	A(22:11)	11
		col	A(22), A(10:1)	
	2Mx8	row	A(21:10)	10
		col	A(21), A(9:1)	
	1Mx 16	row	A(20:9)	01
		col	A(20), A(8:1)	

Notes:

- RA: row address
- BA: bank select (MSB of row address)
- CA: column address
- CMD: auto pre-charge command is currently not supported
- Areas in shades are not recommended for SDRAM configurations, in order to minimize loads on the pads.

14.13.20 Power Down Mode

In order to reduce standby power consumption SDRAM devices provide a Power Down Mode. All banks can optionally be precharged before the device enters Power Down

mode. Once Power Down mode is initiated by holding CKE low, all receiver circuits except for CLK and CKE are gated off. Power Down mode does not perform any refresh operations, therefore to prevent loss of data, the device must not remain in Power Down mode longer than the Refresh period (tREF) of the device. Exit from this mode is performed by taking CKE “high”. One clock delay is required for power down mode entry and exit.

The Memory Controller provides automatic support for Power Down Mode via the SDRMCON.SDCMSEL (SDRAM clock mode select) bit. When this bit is ‘0’ Power Down mode will not be used and the SDRAM clock will always be present at the SDCLKO pin. When the bit is ‘1’ the device will automatically be placed into Power Down Mode when there are no SDRAM accesses pending. In this case the SDRAM clock will only be present during an Memory Controller-generated SDRAM access (data, refresh, bank/row open etc.) and will be gated off at all other times.

When a refresh is required (at the programmed rate) Memory Controller will automatically take the device out of Power Down Mode, issue the required refresh and will then return the device to Power Down Mode (providing no other SDRAM accesses are pending following the refresh).

By default, the Memory Controller automatically issues the “Pre-Charge All” command sequence and closes all pages prior to entry into Power Down Mode (SDRMCON.PWR_MODE set to 00_B).

The memory controller can also be configured to use the auto-precharge option when running a command (SDRMCON.PWR_MODE set to 01_B) or not to precharge banks at all (active power down mode) with SDRMCON.PWR_MODE set to 10_B.

As a final option, "clock stop" power down mode is also supported. In this case, the clock is disabled between accesses with no preparatory command cycles (SDRMCON.PWR_MODE set to 11_B).

The default reset state of Memory Controller is Power Down Mode enabled (SDRMCON.SDCMSEL = ‘1’).

Note: The programmer should be very careful about the use of this feature as external devices may require this clock to be running in some modes. There are restrictions within the PC-133 specification about when the clock can be disabled, especially if the SDRAMs are operated in self-refresh mode.

A separate field SDRMCON.RES_DLY is provided to allow a delay to be programmed after exiting the power down mode. This field is the delay, in external clock cycles (NOPs), after CKE is taken high on exiting power down mode before another command is permitted.

An additional bit SDRMCON.CLKDIS is provided to allow the clock output to be completely disabled. The projected use for this bit is for DDR cold start where CKE should be high before the clock is enabled. Setting this bit will allow a self refresh exit to be performed to enable CKE without starting the clock.

14.13.21 SDRAM Recovery Phases

Recovery after SDRAM Command

A recovery phase can be programmed to increase the minimum gap between an SDR access and an access to another connected memory.

For write cycles, the minimum value for this gap is two periods of the internal clock between last command/data driven and the next access starting (for DDR the gap for data is one cycle). This can be increased by setting the BUSWAP.WRDTACS field to the required number of internal clock cycles.

For read accesses, the gap can also be increased using the BUSRAP.RDDTACS field in the same way. However, the counter is started when the last read command is issued by the controller and the controller does not permit another device to be accessed until two clock cycles after the last data has been read into the read buffers. As the read data is significantly delayed by the latency through the read synchronization logic (minimum latency is 2 clock cycles CAS latency plus 2 clock cycles internal latency), setting this for read accesses is unlikely to make a significant difference unless very large values are used.

14.13.22 Programmable Parameters

The following table lists programmable parameters for SDRAM accesses. These parameters only matter when parameter **AGEN** (in BUSCONx registers) for a particular memory region is set to "1000_b".

Table 14-43 SDRAM Access Programmable Parameters

Parameter	Function	Register
refreshr	Number of refresh commands issued during each refresh operation.	SDRMREF
ERFSHC & refreshc	Number of cycles (multiplied by 64) between refresh operations: 0 : no refresh needed 1 - 255 : refresh period defined	SDRMREF
ARFSH	execute auto refresh on exit from self refresh when set to one	SDRMREF
SELFREX_DLY	delay after exiting self refresh before permitting any command other than NOP	SDRMREF

Table 14-43 SDRAM Access Programmable Parameters (cont'd)

Parameter	Function	Register
bankm	To select one pattern for bank mask.	SDRMCON
rowm	To select one pattern for row mask	SDRMCON
Crc	Number of NOP cycles between refresh commands.	SDRMCON
Crcd	Number of NOP cycles between a row and column address	SDRMCON
awidth	Number of address bits to be used for column address	SDRMCON
Crp	Number of NOP cycles after a precharge command	SDRMCON
Crsc	Number of NOP cycles after a mode register set command	SDRMCON
Crfsh	Number of refresh commands during initialization	SDRMCON
Cras	Number of cycles between row activate and a precharge command	SDRMCON
opmode	To specify write operation mode: only BURST_WRITE is recognized	SDRMOD
caslat	To specify $\overline{\text{CAS}}$ latency: 2 or 3 clocks	SDRMOD
btyp	To specify burst operation mode: SEQUENTIAL	SDRMOD
burstl	To specify burst length: 1,2,4, 8 or 16	SDRMOD
XOPM	Value to be written to the extended mode register	SDRMOD
XBA	Bank Address value to be used for extended mode register write	SDRMOD
sdrmbusy	Indicate the busy status of SDRAM	SDRSTAT
referr	Indicate a refresh error	SDRSTAT
SDERR	Indicates an error has occurred on and SDRAM read	SDRSTAT
selfren	To kick-off a self refresh entry command	SDRMREF
selfrenst	Status of self refresh entry command	SDRMREF

Table 14-43 SDRAM Access Programmable Parameters (cont'd)

Parameter	Function	Register
selfrex	To kick-off a self refresh exit command	SDRMREF
selfrexst	Status of self refresh exit command	SDRMREF
autoselfr	To activate automatic self refresh entry/exit	SDRMREF
RDDTACS	recovery time after read command before accessing another region	BUSRAP
WRDTACS	recovery time after write command before accessing another region	BUSWAP
EXTCLOCK	ratio between internal clock and external memory clock for SDRAM accesses (no effect for DDR. External clock always runs at internal clock frequency)	BUSRAP

14.14 Debug Behavior

The EBU will lock the external bus arbitration with the EBU owning the external bus to allow the debug system unrestricted access to external memories if the debug suspend input becomes active.

The entry into debug mode is controlled via the halted signal triggered by the CPU.

14.15 Power, Reset and Clock

14.15.1 Clocks

The EBU receives two clocks from the system:

- AHB bus clock
- dedicated EBU clock

The dedicated EBU clock is allowed to be asynchronous to AHB clock. Therefore it is also possible to run the EBU at a higher clock rate than the AHB. This mode is suitable for higher performance applications.

If higher EBU performance is not required it is recommended to operate the EBU on the AHB bus clock because this will save resynchronisation cycles.

The dedicated EBU clock is described on the SCU (System Control Unit) chapter as f_{EBU}. The AHB bus clock for the EBU block is described on the SCU chapter as f_{CPU}.

It is possible to program the f_{EBU} frequency via the EBUCLKR register on the SCU.

The EBU clock, f_{EBU}, can also be enabled or disabled via the CLKSET.EBUCEN and CLKR.EBUCDI bitfields, respectively (see the SCU chapter for a complete description).

14.15.2 Module Reset

The EBU is configured so that its port control lines are going to be in the “nobus” state during and after reset. In this state the EBU will still take control over the ports. It is not wanted that this “nobus” state control is applied to the external pins until the EBU is explicitly programmed by the user. Therefore the default state of the PORTS logic must be so that EBU hardware control is ignored.

The assertion or deassertion of the EBU reset is controlled via the PRSET3.EBU and PRCLR3.EBU bitfields, respectively (this fields are described on the SCU chapter).

14.15.3 Power

The EBU is inside the power core domain, therefore no special considerations about power up or power down sequences need to be taken. For an explanation about the different power domains, please address the SCU (System Control Unit) chapter.

An internal power down mode for the EBU, can be achieved by disabling the clock provided to it. For this one should disable the clock via the specific SCU bitfield (CLKR.EBUCDI).

14.16 System Dependencies

Following features are made available in the different packages.

Table 14-44 Supported operating modes per package

Mode	100 pins	144 pins
16-bit MUX	Yes	Yes
Twin 16-bit MUX	No	Yes
32-bit MUX	No	Yes
16-bit DEMUX, Burst Flash	No	Yes
16-bit SDRAM	No	Yes

14.17 Registers

This section describes the registers and programmable parameters of the EBU. All these registers can be read in User Mode, but can only be written in Supervisor Mode.

All registers are reset by the module reset.

Table 14-45 Registers Address Space

Module	Base Address	End Address	Note
EBU	5800 8000_H	5800 $BFFF_H$	-

Table 14-46 Registers Overview EBU Control Registers

Register Short Name	Register Long Name	Offset Address	Access Mode		Description see
			Read	Write	
CLC	EBU Clock Control Register	0000_H	U, PV, 32	PV, 32	Page 14-97
MODCON	EBU Configuration Register	0004_H	U, PV, 32	PV, 32	Page 14-99
ID	EBU Module ID Register	0008_H	U, PV, 32	PV, 32	Page 14-123
USERCON	EBU Test/Control Configuration Register	$000C_H$	U, PV, 32	PV, 32	Page 14-122
Reserved	Reserved	0010_H	nBE	nBE	
Reserved	Reserved	0014_H	nBE	nBE	
ADDRSEL0	EBU Address Select Register 0	0018_H	U, PV, 32	PV, 32	Page 14-101
ADDRSEL1	EBU Address Select Register 1	$001C_H$	U, PV, 32	PV, 32	Page 14-101
ADDRSEL2	EBU Address Select Register 2	0020_H	U, PV, 32	PV, 32	Page 14-101
ADDRSEL3	EBU Address Select Register 3	0024_H	U, PV, 32	PV, 32	Page 14-101
BUSRCON0	EBU Bus Configuration Register 0	0028_H	U, PV, 32	PV, 32	Page 14-102
BUSRAP0	EBU Bus Access Parameter Register 0	$002C_H$	U, PV, 32	PV, 32	Page 14-109

Table 14-46 Registers Overview EBU Control Registers (cont'd)

Register Short Name	Register Long Name	Offset Address	Access Mode		Description see
			Read	Write	
BUSWCON0	EBU Bus Configuration Register 0	0030_H	U, PV, 32	PV, 32	Page 14-106
BUSWAP0	EBU Bus Access Parameter Register 0	0034_H	U, PV, 32	PV, 32	Page 14-111
BUSRCON1	EBU Bus Configuration Register 1	0038_H	U, PV, 32, 46	PV, 32	Page 14-102
BUSRAP1	EBU Bus Access Parameter Register 1	$003C_H$	U, PV, 32	PV, 32	Page 14-109
BUSWCON1	EBU Bus Configuration Register 1	0040_H	U, PV, 32	PV, 32	Page 14-106
BUSWAP1	EBU Bus Access Parameter Register 1	0044_H	U, PV, 32	PV, 32	Page 14-111
BUSRCON2	EBU Bus Configuration Register 2	0048_H	U, PV, 32	PV, 32	Page 14-102
BUSRAP2	EBU Bus Access Parameter Register 2	$004C_H$	U, PV, 32	PV, 32	Page 14-109
BUSWCON2	EBU Bus Configuration Register 2	0050_H	U, PV, 32	PV, 32	Page 14-106
BUSWAP2	EBU Bus Access Parameter Register 2	0054_H	U, PV, 32	PV, 32	Page 14-111
BUSRCON3	EBU Bus Configuration Register 3	0058_H	U, PV, 32	PV, 32	Page 14-102
BUSRAP3	EBU Bus Access Parameter Register 3	$005C_H$	U, PV, 32	PV, 32	Page 14-109
BUSWCON3	EBU Bus Configuration Register 3	0060_H	U, PV, 32	PV, 32	Page 14-106
BUSWAP3	EBU Bus Access Parameter Register 3	0064_H	U, PV, 32	PV, 32	Page 14-111
SDRMCON	EBU, SDRAM Control Register	0068_H	U, PV, 32	PV, 32	Page 14-114
SDRMOD	EBU, SDRAM Mode Register	$006C_H$	U, PV, 32	PV, 32	Page 14-117

Table 14-46 Registers Overview EBU Control Registers (cont'd)

Register Short Name	Register Long Name	Offset Address	Access Mode		Description see
			Read	Write	
SDRMREF	EBU, SDRAM Refresh Control Register	0070_H	U, PV, 32	PV, 32	Page 14-119
SDRSTAT	EBU, SDRAM Status Register	0074_H	U, PV, 32	PV, 32	Page 14-121
Reserved	Reserved	0078_H - $BFFC_H$	nBE	nBE	

Access Restrictions

Note: The EBU registers are accessible only through word accesses. Half-word and byte accesses on EBU registers will generate a bus error. Writes to unused address space will not cause an error but be ignored.

14.17.1 Clock Control Register, CLC

CLC
EBU Clock Control Register **(000_H)** **Reset Value: 0011 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								EBUDIVACK		DIV2ACK	SYNCACK	EBUDIV		DIV2	SYNC
r								r		r	r	rw		rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0														DISS	DISR
r														r	rw

Field	Bits	Type	Description
DISR	0	rw	**EBU Disable Request Bit** This bit is used for enable/disable control of the EBU. 0_B EBU disable is not requested 1_B EBU disable is requested

Field	Bits	Type	Description
DISS	1	r	**EBU Disable Status Bit** DISS is always read as 0, as accessing the register in the EBU will cause the EBU to be automatically enabled. 0_B EBU is enabled (default after reset) 1_B EBU is disabled
SYNC	16	rw	**EBU Clocking Mode** 0_B request EBU to run asynchronously to AHB bus clock and use separate clock source 1_B request EBU to run synchronously to ARM processor (default after reset)
DIV2	17	rw	**DIV2 Clocking Mode** 0_B standard clocking mode. clock input selected by SYNC bitfield (default after reset). 1_B request EBU to run off AHB bus clock divided by 2.
EBUDIV	[19:18]	rw	**EBU Clock Divide Ratio** 00_B request EBU to run off input clock (default after reset) 01_B request EBU to run off input clock divided by 2 10_B request EBU to run off input clock divided by 3 11_B request EBU to run off input clock divided by 4
SYNCACK	20	r	**EBU Clocking Mode Status** 0_B the EBU is asynchronous to the AHB bus clock and is using a separate clock source 1_B EBU is synchronous to the AHB bus clock (default after reset)
DIV2ACK	21	r	**DIV2 Clocking Mode Status** 0_B EBU is using standard clocking mode. clock input selected by SYNC bitfield (default after reset). 1_B EBU is running off AHB bus clock divided by 2.
EBUDIVACK	[23:22]	r	**EBU Clock Divide Ratio Status** 00_B EBU is running off input clock (default after reset) 01_B EBU is running off input clock divided by 2 10_B EBU is running off input clock divided by 3 11_B EBU is running off input clock divided by 4

Field	Bits	Type	Description
0	[15:2], [31:24]	r	**Reserved** Read as 0; should be written with 0.

Note: While the DISR bit is implemented in the EBU, it connects to the standby logic which will disable the clock tree. Standby mode will be exited automatically when an attempt is made to access the EBU. This register can be Endinit-protected after initialization. Writing to this register in this state will cause the EBU to generate an AHB Error.

14.17.2 Configuration Register, MODCON

MODCON
EBU Configuration Register **(004_H)**

Reset Value: $0000\ 0020_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ALE	**0**	**ACC SINH ACK**	**ACC SINH**	**GLOBALCS**				**LOCKTIMEOUT**							
rw	r	r	rw	rw				rw							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TIMEOUTC								**ARBMODE**		**ARB SYNC**	**EXT LOCK**	**0**	**SDT RI**	**LCK ABRT**	**STS**
rw								rw		rw	rw	r	rw	r	r

Field	Bits	Type	Description
STS	0	r	**Memory Status Bit** Software access to the $\overline{\text{WAIT}}$ input pin to the EBU.
LCKABRT	1	r	**Lock Abort** Reserved, will read as 0

Field	Bits	Type	Description
SDTRI	2	rw	**SDRAM Tristate** The signals affected by this setting are CKE, SDCLKO, CAS and RAS 0_B SDRAM control signals are driven by the EBU when the EBU does not own the external bus. SDRAM cannot be shared. 1_B SDRAM control signals are tri-stated by the EBU when the EBU does not own the external bus. The SDRAM can be shared.
EXTLOCK	4	rw	**External Bus Lock Control** 0_B External bus is not locked after the EBU gains ownership 1_B External bus is locked after the EBU gains ownership
ARBSYNC	5	rw	**Arbitration Signal Synchronization Control** 0_B Arbitration inputs are synchronous 1_B Arbitration inputs are asynchronous
ARBMODE	[7:6]	rw	**Arbitration Mode Selection** 00_B No Bus arbitration mode selected 01_B Arbiter Mode arbitration mode selected 10_B Participant arbitration mode selected 11_B Sole Master arbitration mode selected
TIMEOUTC	[15:8]	rw	**Bus Time-out Control** This bit field determines the number of inactive cycles leading to a bus time-out after the EBU gains ownership. 00_H Time-out is disabled. 01_H Time-out is generated after 1×8 clock cycles. … FF_H Time-out is generated after 255×8 clock cycles.
LOCKTIMEOUT	[23:16]	rw	**Lock Timeout Counter Preload** Reserved, must be written with 0
GLOBALCS	[27:24]	rw	**Global Chip Select Enable** No effect, should be written with 0
ACCSINH	28	rw	**Access Inhibit request** Reserved, must be written with 0

Field	Bits	Type	Description
ACCSINHACK	29	r	**Access inhibit acknowledge** Reserved, will always read 0
ALE	31	rw	**ALE Mode** Switch the $\overline{ADV}$ output to be an active high ALE signal instead of active low $\overline{ADV}$. 0_B Output is $\overline{ADV}$ 1_B Output is ALE
0	3, 30	r	**Reserved** Read as 0; should be written with 0.

14.17.3 Address Select Register, ADDRSELx

ADDRSELx (x = 0-3)
EBU Address Select Register x **(018_H+x*4_H)** **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0													WPROT	ALTENAB	REGENAB
r													rw	rw	rw

Field	Bits	Type	Description
REGENAB	0	rw	**Memory Region Enable** 0_B Memory region is disabled (default after reset). 1_B Memory region is enabled.
ALTENAB	1	rw	**Alternate Region Enable** 0_B Memory region is disabled (default after reset). 1_B Memory region is enabled.
WPROT	2	rw	**Memory Region Write Protect** 0_B Region is enabled for write accesses 1_B Region is write protected.
0	[31:3]	r	**Reserved** Read as 0; should be written with 0.

14.17.4 Bus Configuration Register, BUSRCONx

BUSRCONx (x = 0-3)
EBU Bus Configuration Register (028_H+x*10_H)

Reset Value: 00D3 0040_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AGEN				0	AAP	WAIT		PORTW		BCGEN		WAITINV	DBA	EBSE	ECSE
rw				r	rw	rw		rw		rw		rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								NAA	BFCMSEL	FDBKEN	BFSSS	FBBMSEL	FETBLEN		
r								rw	rw	rw	rw	rw	rw		

Field	Bits	Type	Description
FETBLEN	[2:0]	rw	**Burst Length for Synchronous Burst** Defines maximum number of burst data cycles which are executed by Memory Controller during a burst access to a Synchronous Burst device. 000_B 1 data access (default after reset). 001_B 2 data accesses. 010_B 4 data accesses. 011_B 8 data accesses. $1xx_B$ reserved.
FBBMSEL	3	rw	**Synchronous burst buffer mode select** 0_B Burst buffer length defined by value in FETBLEN (default after reset). 1_B Continuous mode. All data required for transaction is transferred in a single burst.
BFSSS	4	rw	**Read Single Stage Synchronization:** The second read-data synchronization stage in the pad-logic can be bypassed. Reduces access latency at the expense of the maximum achievable operating frequency. 0_B Two stages of synchronization used. (maximum margin) 1_B One stage of synchronization used. (minimum latency)

Field	Bits	Type	Description
FDBKEN	5	rw	**Burst FLASH Clock Feedback Enable** 0_B BFCLK feedback not used. 1_B Incoming data and control signals (from the Burst FLASH device) are re-synchronized to the BFCLKI input.
BFCMSEL	6	rw	**Burst Flash Clock Mode Select** 0_B Burst Flash Clock runs continuously with values selected by this register 1_B Burst Flash Clock is disabled between accesses
NAA	7	rw	**Enable flash non-array access workaround** set to logic one to enable workaround when region is accessed with address bit 28 set. See **Section 14.12.14**
ECSE	16	rw	**Early Chip Select for Synchronous Burst** 0_B CS is delayed. 1_B CS is not delayed. *Note: (see **Section 14.11.3** and **Section 14.12.7**)*
EBSE	17	rw	**Early Burst Signal Enable for Synchronous Burst** 0_B ADV is delayed. 1_B ADV is not delayed. *Note: (see **Section 14.11.3** and **Section 14.12.7**)*

Field	Bits	Type	Description
DBA	18	rw	**Disable Burst Address Wrapping** 0_B Memory Controller automatically re-aligns any non-aligned synchronous burst access so that data can be fetched from the device in a single burst transaction. 1_B Memory Controller always starts any burst access to a synchronous burst device at the address specified by the AHB request. Any required address wrapping must be automatically provided by the Burst FLASH device. *Note: Care must be taken with the use of this feature. The Burst capable device must be programmed to wrap at the appropriate address boundary prior to selection of this mode.* ***Section 14.12.11***
WAITINV	19	rw	**Reversed polarity at WAIT** 0_B **OFF**, input at WAIT pin is active low (default after reset). 1_B **Polarity reversed**, input at WAIT pin is active high. *Note: This bit has no effect when using Burst FLASH Data Handshake Mode*
BCGEN	[21:20]	rw	**Byte Control Signal Control** This bit field selects the timing mode of the byte control signals. 00_B Byte control signals follow chip select timing. 01_B Byte control signals follow control signal timing ($\overline{RD}$, RD/$\overline{WR}$) (default after reset). 10_B Byte control signals follow write enable signal timing (RD/$\overline{WR}$ only). 11_B Reserved.
PORTW	[23:22]	rw	**Device Addressing Mode** See **Table 14-11** and **Table 14-13**

Field	Bits	Type	Description
WAIT	[25:24]	rw	**External Wait Control** Function of the WAIT input. This is specific to the device type (i.e. the AGEN field). **For Asynchronous Devices:** 0_D OFF (default after reset). 1_D Asynchronous input at WAIT. 2_D Synchronous input at WAIT. 3_D reserved. *Note: See **Section 14.11.6.1*** **For Synchronous Burst Devices:** 0_D OFF (default after reset). 1_D Wait for page load (Early WAIT). 2_D Wait for page load (WAIT with data). 3_D Abort and retry access. *Note: See **Section 14.12.13***
AAP	26	rw	**Asynchronous Address phase:** Enables an access mode for synchronous memories where the clock is not started until after the address hold phase. 0_B Clock is enabled at beginning of access. 1_B Clock is enabled at after address phase.
AGEN	[31:28]	rw	**Device Type for Region** See **Section 14.7.3**
0	[15:8], 27	r	**Reserved** Read as 0; should be written with 0.

14.17.5 Bus Write Configuration Register, BUSWCONx

BUSWCONx (x = 0-3)
EBU Bus Write Configuration Register(030_H+x*10_H)

Reset Value: 00D3 0000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AGEN				LOCKCS	AAP	WAIT		PORTW		BCGEN		WAITINV	0	EBSE	ECSE
rw				rw	rw	rw		r		rw		rw	r	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								NAA	0			FBBMSEL	FETBLEN		
r								r	r			rw	rw		

Field	Bits	Type	Description
FETBLEN	[2:0]	rw	**Burst Length for Synchronous Burst** Defines maximum number of burst data cycles which are executed by Memory Controller during a burst access to a Synchronous Burst device. 000_B 1 data access (default after reset). 001_B 2 data accesses. 010_B 4 data accesses. 011_B 8 data accesses. $1xx_B$ reserved.
FBBMSEL	3	rw	**Synchronous burst buffer mode select** 0_B Burst buffer length defined by value in FETBLEN (default after reset). 1_B Continuous mode. All data required for transaction transferred in single burst
NAA	7	r	**Enable flash non-array access workaround** When set to logic one workaround for non-array is access when region is accessed with address bit 28 set is enabled. See **Section 14.12.14**. Mirror of equivalent field in BUSRCON register. To set write to equivalent field in BUSRCON register
0	[15:8]	r	**Reserved** 00_H Reserved Value

Field	Bits	Type	Description
ECSE	16	rw	**Early Chip Select for Synchronous Burst** 0_B CS is delayed. 1_B CS is not delayed. *Note: (see **Section 14.11.3** and **Section 14.12.7**)*
EBSE	17	rw	**Early Burst Signal Enable for Synchronous Burst** 0_B ADV is delayed. 1_B ADV is not delayed. *Note: (see **Section 14.11.3** and **Section 14.12.7**)*
WAITINV	19	rw	**Reversed polarity at WAIT** 0_B **OFF**, input at WAIT pin is active low (default after reset). 1_B **Polarity reversed**, input at WAIT pin is active high. *Note: This bit has no effect when using Burst FLASH Data Handshake Mode*
BCGEN	[21:20]	rw	**Byte Control Signal Control** This bit field selects the timing mode of the byte control signals. 00_B Byte control signals follow chip select timing. 01_B Byte control signals follow control signal timing ($\overline{RD}$, RD/$\overline{WR}$) (default after reset). 10_B Byte control signals follow write enable signal timing (RD/$\overline{WR}$ only). 11_B Reserved.
PORTW	[23:22]	r	**Device Addressing Mode** See **Table 14-11** and **Table 14-12**

Field	Bits	Type	Description
WAIT	[25:24]	rw	**External Wait Control** Function of the WAIT input. This is specific to the device type (i.e. the AGEN field). **For Asynchronous Devices:** 0_D OFF (default after reset). 1_D Asynchronous input at WAIT. 2_D Synchronous input at WAIT. 3_D reserved. *Note: See **Section 14.11.6.1*** **For Synchronous Burst Devices:** 0_D OFF (default after reset). 1_D Wait for page load (Early WAIT). 2_D Wait for page load (WAIT with data). 3_D Abort and retry access. *Note: See **Section 14.12.13***
AAP	26	rw	**Asynchronous Address phase:** Enables an access mode for synchronous memories where the clock is not started until after the address hold phase. 0_B Clock is enabled at beginning of access. 1_B Clock is enabled at after address phase.
LOCKCS	27	rw	**Lock Chip Select** Enable Chip Select for Automatic Locking in the event of a write access 0_B Chip Select cannot be locked (default after reset). 1_B Chip Select will be automatically locked when written to from the processor data port.
AGEN	[31:28]	rw	**Device Type for Region** See **Section 14.7.3**
0	[6:4], [15:8], 18	r	**Reserved** Read as 0; should be written with 0.

14.17.6 Bus Read Access Parameter Register, BUSRAPx

BUSRAPx (x = 0-3)
EBU Bus Read Access Parameter Register

($02C_H$+x*10_H) **Reset Value: FFFF $FFFF_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADDRC				AHOLDC				CMDDELAY				EXTDATA		EXTCLOCK	
rw				rw				rw				rw		rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATAC				WAITRDC					RDRECOVC			RDDTACS			
rw				rw					rw			rw			

Field	Bits	Type	Description
RDDTACS	[3:0]	rw	**Recovery Cycles between Different Regions** This bit field determines the number of clock cycles of the Recovery Phase between consecutive accesses directed to different regions or different types of access. See **Section 14.10.7**. 0000_B No Recovery Phase clock cycles available. 0001_B 1 clock cycle selected. … 1110_B 14 clock cycles selected. 1111_B 15 clock cycles selected.
RDRECOVC	[6:4]	rw	**Recovery Cycles after Read Accesses** This bit field determines the basic number of clock cycles of the Recovery Phase at the end of read accesses. 000_B No Recovery Phase clock cycles available. 001_B 1 clock cycle selected. … 110_B 6 clock cycles selected. 111_B 7 clock cycles selected.

Field	Bits	Type	Description
WAITRDC	[11:7]	rw	**Programmed Wait States for read accesses** Number of programmed wait states for read accesses. For synchronous accesses, this will always be adjusted so that the phase exits on a rising edge of the external clock. 00000_B1 wait state. 00001_B1 wait states. 00010_B2 wait state. ... 11110_B30 wait states. 11111_B31 wait states.
DATAC	[15:12]	rw	**Data Hold Cycles for Read Accesses** This bit field determines the basic number of Data Hold phase clock cycles during read accesses. It has no effect in the current implementation
EXTCLOCK	[17:16]	rw	**Frequency of external clock at pin BFCLKO** 00_B Equal to INT_CLK frequency. 01_B 1/2 of INT_CLK frequency. 10_B 1/3 of INT_CLK frequency. 11_B 1/4 of INT_CLK frequency (default after reset). *Note: See **Section 14.12.4**.*
EXTDATA	[19:18]	rw	**Extended data** See **Section 14.10.6** 00_B external memory outputs data every BFCLK cycle 01_B external memory outputs data every two BFCLK cycles 10_B external memory outputs data every four BFCLK cycles 11_B external memory outputs data every eight BFCLK cycles
CMDDELAY	[23:20]	rw	**Command Delay Cycles** This bit field determines the basic number of Command Delay phase clock cycles. 0000_B0 clock cycle selected. 0001_B1 clock cycle selected. … 1110_B14 clock cycles selected. 1111_B15 clock cycles selected.

Field	Bits	Type	Description
AHOLDC	[27:24]	rw	**Address Hold Cycles** This bit field determines the number of clock cycles of the address hold phase. 0000_B0 clock cycle selected 0001_B1 clock cycle selected ... 1110_B14 clock cycles selected 1111_B15 clock cycles selected
ADDRC	[31:28]	rw	**Address Cycles** This bit field determines the number of clock cycles of the address phase. 0000_B1 clock cycle selected 0001_B1 clock cycle selected ... 1110_B14 clock cycles selected 1111_B15 clock cycles selected

14.17.7 Bus Write Access Parameter Register, BUSWAPx

BUSWAPx (x = 0-3)
EBU Bus Write Access Parameter Register
(034_H+x*10_H) **Reset Value: FFFF $FFFF_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADDRC				AHOLDC				CMDDELAY				EXTDATA		EXTCLOCK	
rw				rw				rw				rw		rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATAC				WAITWRC					WRRECOVC			WRDTACS			
rw				rw					rw			rw			

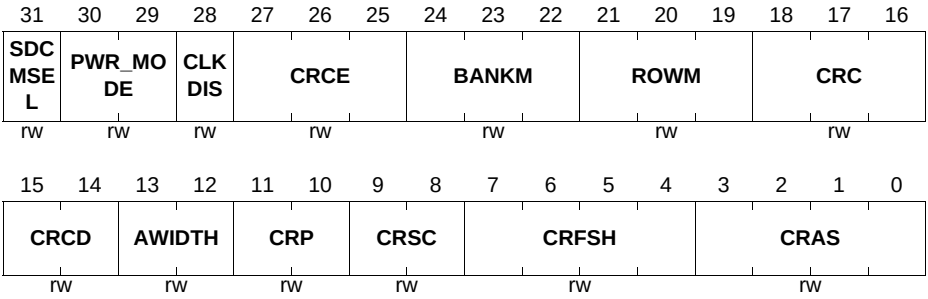
Field	Bits	Type	Description
WRDTACS	[3:0]	rw	**Recovery Cycles between Different Regions** This bit field determines the number of clock cycles of the Recovery Phase between consecutive accesses directed to different regions or different types of access. See **Section 14.10.7** 0000_B No Recovery Phase clock cycles available. 0001_B 1 clock cycle selected. … 1110_B 14 clock cycles selected. 1111_B 15 clock cycles selected.
WRRECOVC	[6:4]	rw	**Recovery Cycles after Write Accesses** This bit field determines the basic number of clock cycles of the Recovery Phase at the end of write accesses. 000_B No Recovery Phase clock cycles available. 001_B 1 clock cycle selected. … 110_B 6 clock cycles selected. 111_B 7 clock cycles selected.
WAITWRC	[11:7]	rw	**Programmed Wait States for write accesses** Number of programmed wait states for write accesses. For synchronous accesses, this will always be adjusted so that the phase exits on a rising edge of the external clock. 00000_B1 wait state. 00001_B1 wait states. 00010_B2 wait state. ... 11110_B30 wait states. 11111_B31 wait states.
DATAC	[15:12]	rw	**Data Hold Cycles for Write Accesses** This bit field determines the basic number of Data Hold phase clock cycles during write accesses. 0000_B No Recovery Phase clock cycles available. 0001_B 1 clock cycle selected. … 1110_B 14 clock cycles selected. 1111_B 15 clock cycles selected.

Field	Bits	Type	Description
EXTCLOCK	[17:16]	rw	**Frequency of external clock at pin BFCLKO** 00_B Equal to INT_CLK frequency. 01_B 1/2 of INT_CLK frequency. 10_B 1/3 of INT_CLK frequency. 11_B 1/4 of INT_CLK frequency (default after reset). *Note: See **Section 14.12.4**.*
EXTDATA	[19:18]	rw	**Extended data** See **Section 14.10.6**. 00_B external memory outputs data every BFCLK cycle 01_B external memory outputs data every two BFCLK cycles 10_B external memory outputs data every four BFCLK cycles 11_B external memory outputs data every eight BFCLK cycles
CMDDELAY	[23:20]	rw	**Command Delay Cycles** This bit field determines the basic number of Command Delay phase clock cycles. 0000_B0 clock cycle selected. 0001_B1 clock cycle selected. … 1110_B14 clock cycles selected. 1111_B15 clock cycles selected.
AHOLDC	[27:24]	rw	**Address Hold Cycles** This bit field determines the number of clock cycles of the address hold phase. 0000_B0 clock cycle selected 0001_B1 clock cycle selected ... 1110_B14 clock cycles selected 1111_B15 clock cycles selected
ADDRC	[31:28]	rw	**Address Cycles** This bit field determines the number of clock cycles of the address phase. 0000_B1 clock cycle selected 0001_B1 clock cycle selected ... 1110_B14 clock cycles selected 1111_B15 clock cycles selected

14.17.8 SDRAM Control Register, SDRMCON

SDRAM Control Register

SDRMCON
EBU SDRAM Control Register **(068_H)** **Reset Value: 8000 0000$_H$**



Field	Bits	Type	Description
SDCMSEL	31	rw	**SDRAM clock mode select** 1_B clock disabled between accesses 0_B clock continuously runs *Note: (see **Section 14.13.20**)*
PWR_MODE	[30:29]	rw	**Power Save Mode used for gated clock mode** 0_H precharge before clock stop (default after reset) 1_H auto-precharge before clock stop 2_H active power down (stop clock without precharge) 3_H clock stop power down *Note: (see **Section 14.13.20**)*
CLKDIS	28	rw	**Disable SDRAM clock output** 0_B clock enabled 1_B clock disabled *Note: (see **Section 14.13.20**)*
CRCE	[27:25]	rw	**Row cycle time counter extension** Extends the range of the Crc bit field (see below).

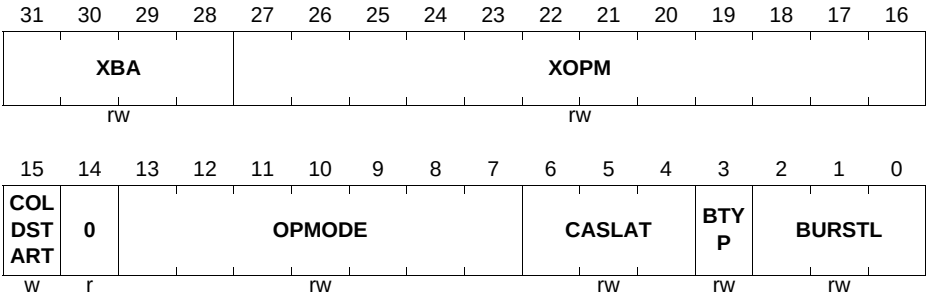
Field	Bits	Type	Description
BANKM	[24:22]	rw	**Mask for bank tag** AHB address bits to be used for determining bank number. 0_H Reserved; (default after reset) 1_H Address bit 21 to 20 2_H Address bit 22 to 21 3_H Address bit 23 to 22 4_H Address bit 24 to 23 5_H Address bit 25 to 24 6_H Address bit 26 to 25 7_H Address bit 26 *Note: See **Section 14.13.14**.*
ROWM	[21:19]	rw	**Mask for row tag** Number of address bits from bit 26 to be used for comparing row tags. 0_H reserved; (default after reset) 1_H Address bit 26 to 9 2_H Address bit 26 to 10 3_H Address bit 26 to 11 4_H Address bit 26 to 12 5_H Address bit 26 to 13 6_H reserved 7_H reserved *Note: See **Section 14.13.15***
CRC	[18:16]	rw	**Row cycle time counter** Number of NOP cycles following a refresh command before another command (other than a NOP) can be issued to the SDRAM. Combined with the CRCE bit as follows: Insert (CRCE * 8) + CRC + 1 NOP cycles.
CRCD	[15:14]	rw	**Row to column delay counter** Number of NOP cycles between a row address and a column address: Insert CRCD + 1 NOP cycles (default after reset CRCD is 0).

Field	Bits	Type	Description
AWIDTH	[13:12]	rw	**Width of column address** Number of address bits from bit 0 to be used for column address. See also **Section 14.13.19**. e.g. for 16 bit DRAMs 0_H **reserved**, do not use 1_H Address(8:0) 2_H Address(9:0) 3_H Address(10:0)
CRP	[11:10]	rw	**Row precharge time counter** Number of NOP cycles inserted after a precharge command. The actual number performed can be greater due to CAS latency and burst length. Insert CRP + 1 NOP cycles (default after reset CRP is 0)
CRSC	[9:8]	rw	**Mode register set-up time** Number of NOP cycles after a mode register set command. Insert CRSC + 1 NOP cycles (default after reset CRSC is 0)
CRFSH	[7:4]	rw	**Initialization refresh commands counter** Number of refresh commands issued during power-up initialization sequence. Perform CRFSH + 1 refresh cycles (default after reset CRFSH is 0)
CRAS	[3:0]	rw	**Row to precharge delay counter** Number of clock cycles between row activate command and a precharge command. Minimum CRAS + 1 clock cycles (default after reset CRAS is 0)

14.17.9 SDRAM Mode Register, SDRMOD

SDRAM Mode Register

SDRMOD
EBU SDRAM Mode Register **($6C_H$)** **Reset Value: 0000 0020$_H$**



Field	Bits	Type	Description
XBA	[31:28]	rw	**Extended Operation Bank Select** Value to be written to the bank select pins of a “Mobile” SDRAM device during an extended mode register write operation. Control of these bits is provided to allow support of future enhanced “Mobile” SDRAM devices. See **Section 14.13.10** *Note: Care must be taken when programming these bits to ensure that a valid extended mode register access occurs (e.g. it is possible to generate an extra unwanted standard mode register write by incorrect programming of these bits).* *14. Consult the appropriate SDRAM documentation for the function of these bits.*

Field	Bits	Type	Description
XOPM	[27:16]	rw	**Extended Operation Mode** Value to be written to the extended mode register of a “Mobile” SDRAM device. This value is issued to the SDRAM via it’s address inputs during an extended mode register write. This field is wider than current extended mode registers to allow support of future enhanced “Mobile” SDRAM devices. *Note: Consult the appropriate SDRAM documentation for the function of these bits.* *15. The Memory Controller provides a 13-bit wide bit-field for the extended mode register to cater for devices that could theoretically use an additional control bit (in comparison to currently available devices).*
COLDSTART	15	**w**	**SDRAM coldstart** This bit will always read 0. If a write to the **SDRMOD** register takes place with this bit set, the SDRAM device mode register will be updated to match the data written to the register. See **Section 14.13.9**
OPMODE	[13:7]	rw	**Operation Mode** Memory Controller only supports burst write standard operation. 000000_B Only this value must be written (default after reset) *Note: Other values reserved.*
CASLAT	[6:4]	rw	**CAS latency** Number of clocks between a READ command and the availability of data. 2_D Two clocks (default after reset) 3_D Three clocks *Note: Other values reserved.*
BTYP	3	rw	**Burst type** Memory Controller only supports sequential burst. 0_B Only this value should be written (default after reset) 1_B Reserved

Field	Bits	Type	Description
BURSTL	[2:0]	rw	**Burst length** Number of locations can be accessed with a single command. 0_D 1 (default after reset) 1_D 2 2_D 4 3_D 8 4_D 16 *Note: Other values reserved.*
0	14	r	**Reserved** Read as 0; should be written with 0.

14.17.10 SDRAM Refresh Control Register, SDRMREF

SDRAM Refresh Control Register

SDRMREF
EBU SDRAM Refresh Control Register(070_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0				RES_DLY			ARFSH	SELFREX_DLY							
r				rw			rw	rw							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERFSHC		AUTOSELFRN	SELFREN	SELFRENST	SELFREX	SELFREXST	REFRESHR			REFRESHC					
rw		rw	rw	rw	rw	rw	rw			rw					

Field	Bits	Type	Description
RES_DLY	[27:25]	rw	**Delay on Power Down Exit** Number of NOPs after the SDRAM controller exits power down before an active command is permitted. *Note: See **Section 14.13.20***

Field	Bits	Type	Description
ARFSH	24	rw	**Auto Refresh on Self refresh Exit** If set to one, an auto refresh cycle will be performed on exiting self refresh before the self refresh exit delay. If set to zero, no refresh will be performed. *Note: See **Section 14.13.18***
SELFREX_DLY	[23:16]	rw	**Self Refresh Exit Delay** Number of NOP cycles inserted after a self refresh exit before a command is permitted to the SDRAM/DDRAM. *Note: See **Section 14.13.18***
ERFSHC	[15:14]	rw	**Extended Refresh Counter Period** This field is used to increase the range of the refreshc field from 6 bits to 8 bits with ERFSHC being used as bits 7 and 6 of the extended field and refreshc as bit 5 to 0.
AUTOSELFR	13	rw	**Automatic Self Refresh** When this bit is set to ’1’, Memory Controller will automatically issue the Self Refresh Entry command to all SDRAM devices when it gives up control of the external bus, and will automatically issue Self Refresh Exit when it regains control of the bus.
SELFREN	12	rw	**Self Refresh Entry** When this bit is written with ’1’ the Self Refresh Entry command is issued to all SDRAM devices, regardless whether they are attached to type 0 or type 1.
SELFRENST	11	r	**Self Refresh Entry Status.** If this bit is set to ’1’, it means the Self Refresh Entry command has been successfully issued. This bit is reset when bit SELFREX is set to ’1’ or a reset takes place.
SELFREX	10	rw	**Self Refresh Exit (Power Up).** When this bit is written with ’1’ the Self Refresh Exit command is issued to all SDRAM devices, regardless whether they are attached to type 0 or type 1.

Field	Bits	Type	Description
SELFREXST	9	r	**Self Refresh Exit Status.** If this bit is set to '1', it means the Self Refresh Exit command has been successfully issued. This bit is reset when bit SELFREN is set to '1' or a reset takes place.
REFRESHR	[8:6]	rw	**Number of refresh commands** The number of additional refresh commands issued to SDRAM each time a refresh is due. Number of refresh commands to use is REFRESHR + 1
REFRESHC	[5:0]	rw	**Refresh counter period** Number of clock cycles between refresh operations. Refresh period is REFRESHC x 64 clock cycles.
0	[30:28]	r	**Reserved** Read as 0; should be written with 0.

14.17.11 SDRAM Status Register, SDRSTAT

SDRAM Status Register

SDRSTAT
EBU SDRAM Status Register **(074_H)** **Reset Value: 0001 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0													SDERR	SDRMBUSY	REFERR
r						r							r	r	r

Field	Bits	Type	Description
SDERR	2	r	**SDRAM read error** SDRAM controller has detected an error when returning read data 0_B Reads running successfully 1_B Read error condition has been detected *Note: This bit is reset by a write access to SDRMCON.*
SDRMBUSY	1	r	**SDRAM Busy** The status of power-up initialization sequence. 0_B Power-up initialization sequence is not running 1_B Power-up initialization sequence is running
REFERR	0	r	**SDRAM Refresh Error** Unsuccessful previous refresh request collides with a new request. This bit is reset by a write access to SDRMCON. 0_B No refresh error. 1_B Refresh error occurred.
0	[31:3]	r	**Reserved** Read as 0; should be written with 0.

14.17.12 Test/Control Configuration Register, USERCON

USERCON
EBU Test/Control Configuration Register

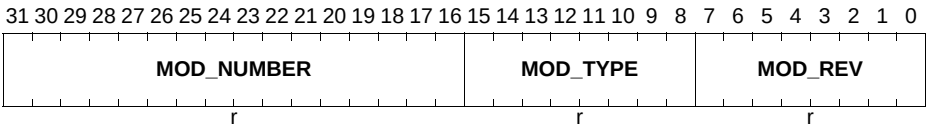
($00C_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0						ADVIO	ADDIO								
r						rw	rw								

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0															DIP
r															rw

Field	Bits	Type	Description
DIP	0	rw	**Disable Internal Pipelining** Reserved, must be set to 0_B
ADDIO	[24:16]	rw	**Address Pins to GPIO Mode** Individual Control Bits for Address Bus Bits 24 down to 16 respectively. 0_B Address Bit is required for addressing memory 1_B Address Bit is available for GPIO function
ADVIO	25	rw	**ADV Pin to GPIO Mode** Control Bit for the ADV/ALE output 0_B $\overline{ADV}$ pin is required for controlling memory 1_B $\overline{ADV}$ pin is available for GPIO function
0	[15:1], [31:26]	r	**Reserved** Read as 0; should be written with 0.

ID
EBU Module Identification Register (08_H) **Reset Value: 0014 C0XX$_H$**



Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision** Indicates the revision number of the implementation. This information depends on the design step.
MOD_TYPE	[15:8]	r	**Module Type** This internal marker is fixed to $C0_H$.
MOD_NUMBER	[31:16]	r	**Module Number** Indicates the module identification number

15 Ethernet MAC (ETH)

The Ethernet MAC (ETH) is a major communication peripheral that supports 10/100 MBit/s data transfer rates in compliance with the IEEE 802.3-2002 standard. The ETH may be used to impliment Internet connected applications using IPv4 and IPv6. The ETH also includes support for IEEE1588 time synchronisation to allow implimentation of Real Time Ethernet protocols.

Table 15-1 Abbreviations

ETH	Ethernet MAC Peripheral
MTL	MAC Transaction Layer
PHY	Physical Layer Interface
MMC	MAC Management Counters
SMI	Station Management Interface
COE	Checksum Offload Engine
PMT	Power Management
MII	Media Independent Interface
RMII	Reduced media Independent interface

The following document is reprinted with permission of Synopsys Inc.
No disclosure of Databook to Synopsys' Competitors without Synopsys consent. List of Competitors is available from Infineon Technologies AG or Synopsys Inc.

15.1 Overview

The ETH peripheral is comprised of five major functional units. The ETH-Core takes user provided data frames and formats them for transmission to an external PHY via an MII or RMII interface. The ETH MAC Transaction Layer (MTL) acts as a bridge between the application and the ETH Core. The MTL provides two 2K byte FIFO’s to buffer the transmit and receive frames. The application may write data frames directly to the MTL (cut through mode) or more normally will use the dedicated ETH DMA unit. The ETH DMA allows the application to define a region of RAM to be used as transmit and receive buffers. DMA transfers are initiated by DMA descriptors which are also held in RAM. The ETH also includes a system time module which allows timestamping of transmit and receive frames. The ETH also includes an extensive set of MAC Mangement counters which provide detailed bus statistics.

The ETH includes the following features, listed by category.

15.1.1 ETH Core Features

- Supports 10/100-Mbit/s data transfer rates with the following PHY interfaces
  - IEEE 802.3-compliant RMII/MII (default) interface to communicate with an external Fast Ethernet PHY
- Supports both full-duplex and half-duplex operation
  - Supports CSMA/CD Protocol for half-duplex operation
  - Supports IEEE 802.3x flow control for full-duplex operation
  - Optional forwarding of received pause control frames to the user application in full-duplex operation
  - Back-pressure support for half-duplex operation
  - Automatic transmission of zero-quanta pause frame on deassertion of flow control input in full-duplex operation
- Preamble and start-of-frame data (SFD) insertion in Transmit, and deletion in Receive paths
- Automatic CRC and pad generation controllable on a per-frame basis
- Options for Automatic Pad/CRC Stripping on receive frames
- Programmable frame length to support Standard or Jumbo Ethernet frames with sizes up to 16 KB
- Programmable InterFrameGap (40-96 bit times in steps of 8)
- Supports a variety of flexible address filtering modes:
  - Up to 3 additional 48-bit perfect (DA) address filters with masks for each byte
  - Up to 3 48-bit SA address comparison check with masks for each byte
  - 64-bit Hash filter for multicast and uni-cast (DA) addresses
  - Option to pass all multicast addressed frames
  - Promiscuous mode support to pass all frames without any filtering for network monitoring
  - Passes all incoming packets (as per filter) with a status report
- Separate 32-bit status returned for transmission and reception packets
- Supports IEEE 802.1Q VLAN tag detection for reception frames
- Separate transmission, reception, and control interfaces to the Application
- Supports 32-bit data transfer interface on the system-side
- Complete network statistics with RMON/MIB Counters (RFC1757/RFC2819 / RFC2665). It is completely under control of higher protocol level (SW) to make use of these counters.
- MDIO Master interface for PHY device configuration and management, e.g. for switching the PHY in external loopback mode.
- Detection of LAN wake-up frames and AMD Magic Packet frames
- Enhanced Receive module for checking IPv4 header checksum and TCP, UDP, or ICMP checksum encapsulated in IPv4 or IPv6 datagrams.
- Module to support Ethernet frame time stamping as described in IEEE 1588-2008. Sixty-four-bit time stamps are given in each frame’s transmit or receive status.

15.1.2 DMA Block Features

The DMA block exchanges data between the MTL block and the XMC4500 memory. A set of registers (DMA CSR) to control DMA operation is accessible by the XMC4500.

DMA features include:

- 32-bit data transfers
- Single-channel Transmit and Receive engines
- Fully synchronous design operating on a single system clock (except for CSR module, when a separate CSR clock is configured)
- Optimization for packet-oriented DMA transfers with frame delimiters
- Byte-aligned addressing for data buffer support
- Dual-buffer (ring) or linked-list (chained) descriptor chaining
- Descriptor architecture, allowing large blocks of data transfer with minimum CPU intervention; each descriptor can transfer up to 8 KB of data
- Comprehensive status reporting for normal operation and transfers with errors
- Individual programmable burst size for Transmit and Receive DMA Engines for optimal bus utilization
- Programmable interrupt options for different operational conditions
- Per-frame Transmit/Receive complete interrupt control
- Round-robin or fixed-priority arbitration between Receive and Transmit engines
- Start/Stop modes
- Separate ports for CPU CSR access and data interface

15.1.3 Transaction Layer (MTL) Features

The MTL block consists of two sets of FIFOs: a Transmit FIFO with programmable threshold capability, and a Receive FIFO with a configurable threshold (default of 64 bytes).

MTL features include:

- 32--bit Transaction Layer block providing a bridge between the application and the -CORE
- Single-channel Transmit and Receive engines
- Data transfers executed using simple FIFO-protocol
- Synchronization for all clocks in the design (Transmit, Receive and system clocks)
- Optimization for packet-oriented transfers with frame delimiters
- Four Separate ports for system-side and -CORE-side transmission and reception
- Two RAM-based asynchronous FIFOs of 2K Bytes depth with synchronous/asynchronous Read and Write operation with respect to the Read and Write clocks (one for transmission and one for reception)
- Receive Status vectors inserted into the Receive FIFO after the EOF transfer enables multiple-frame storage in the Receive FIFO without requiring another FIFO to store those frames' Receive Status.

- Configurable Receive FIFO threshold (default fixed at 64 bytes) in Cut-Through mode
- Option to filter all error frames on reception and not forward them to the application in Store-and-Forward mode
- Option to forward under-sized good frames
- Supports statistics by generating pulses for frames dropped or corrupted (due to overflow) in the Receive FIFO
- Supports Store and Forward mechanism for transmission to the core
- Supports threshold control for transmit buffer management
- Supports configurable number of frames to be stored in FIFO at any time. The default is up to 8 frames in -MTL configuration.
- Automatic generation of PAUSE frame control or backpressure signal to the -core based on Receive FIFO-fill (threshold configurable) level.
- Handles automatic retransmission of Collision frames for transmission
- Discards frames on late collision, excessive collisions, excessive deferral and underrun conditions
- Software control to flush Tx FIFO
- Data FIFO RAM chip-select disabled when inactive, to reduce power consumption
- module to calculate and insert IPv4 header checksum and TCP, UDP, or ICMP checksum in frames transmitted in Store-and-Forward mode.

15.1.4 Monitoring, Test, and Debugging Support Features

- Supports internal loopback on the MII for debugging
- External loopback is supported via the integrated MDIO controlling the PHY
- DMA states (Tx and Rx) given as status bits
- Debug status register that gives status of FSMs in Transmit and Receive data-paths and FIFO fill-levels.
- Application Abort status bits
- MMC (RMON) module in the core
- Current Tx/Rx Buffer pointer as status registers
- Current Tx/Rx Descriptor pointer as status registers

15.1.5 Block Diagram

A block diagram of the ETH's major system configurations is provided in **Figure 15-1**.

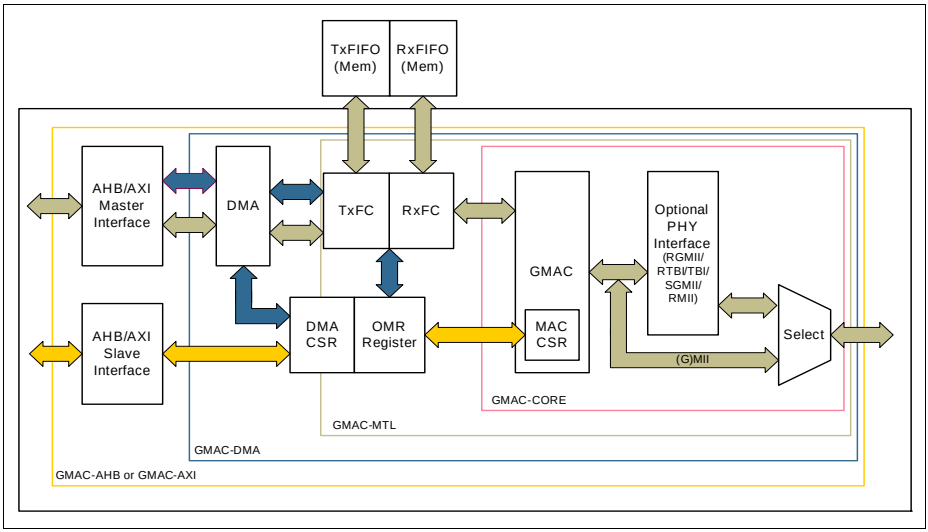


Figure 15-1 ETH Block Diagram

15.2 Functional Description

This chapter describes the structure and programming requirements of the features within the ETH subsystem. Each significant programming feature is discussed in a seperate section.

15.2.1 ETH Core

The ETH core supports two interfaces towards the PHY chip, MII and RMII. The PHY interface can be selected only once after reset. The ETH core communicates with the application side with the MAC Transmit Interface (MTI), MAC Receive Interface (MRI) and the MAC Control Interface (MCI).

15.2.1.1 Transmission

Transmission is initiated when the MTL Application pushes in data with the SOF . When the SOF signal is detected, the ETH accepts the data and begins transmitting to the MII. The time required to transmit the frame data to the RMII/MII after the Application initiates transmission is variable, depending on delay factors like IFG delay, time to transmit preamble/SFD, and any back-off delays for Half-Duplex mode. Until then, the ETH does not accept the data received from MTL.

After the EOF is transferred to the ETH Core, the core complete normal transmission and then gives the Status of Transmission back to the MTL. If a normal collision (in Half-duplex mode) occurs during transmission, the ETH core makes valid the Transmit Status to the MTL. It then accepts and drops all further data until the next SOF is received. The MTL block should retransmit the same frame from SOF on observing a Retry request (in the Status) from the ETH.

The ETH issues an underflow status if the MTL is not able to provide the data continuously during the transmission. During the normal transfer of a frame from MTL, if the ETH receives a SOF without getting an EOF for the previous frame, then it (the SOF) is ignored and the new frame is considered as continuation of the previous frame.

The following six modules constitute the transmission function of the ETH:

- Transmit Bus Interface Module (TBU)
- Transmit Frame Controller Module (TFC)
- Transmit Protocol Engine Module (TPE)
- Transmit Scheduler Module (STX)
- Transmit CRC Generator Module (CTX)
- Transmit Flow Control Module (FTX)

Transmit Bus Interface Module

This module interfaces the transmit path of the ETH core with the external frame with a FIFO interface.

This module also outputs the (32-bit) Transmit Status to the application at the end of normal transmission or collision.

Additionally, this module outputs the Transmit Snapshot register value.

Transmit Frame Controller Module

The Transmit Frame Controller (TFC) consists of two registers to hold data, byte enables, and the last data control received from the TBU. The register provides a buffer between the Application and the TPE to regulate data flow as well as converts the input data into an 8-bit bus towards the TPE.

When the number of bytes received from the Application falls below 60 (DA+SA+LT+DATA), the state machine that interfaces with the TBU automatically appends zeros to the transmitting frame to make the data length exactly 46 bytes to meet the minimum data field requirement of IEEE 802.3. The ETH can be programmed not to append any padding.

The cyclic redundancy check (CRC) for the Frame Check Sequence (FCS) field is calculated before transmission to the TPE module. This value is computed by CTX module. The TFC module receives the computed CRC and appends it to the data being transmitted to the TPE module. When the ETH is programmed to not append the CRC value to the end of Ethernet frames, the TFC module ignores the computed CRC and transmits only the data received from the TBU module to the TPE module. An exception to this rule is that when the ETH is programmed to append pads for frames (DA+SA+LT+DATA) less than 60 bytes sent by the TBU module, the TFC module will append the CRC at the end of padded frame.

The TFC converts the data received from the TBU into 8-bit data for the TPE module.

Transmit Protocol Engine Module

The Transmit Protocol Engine (TPE) module consists of a transmit state machine that controls the operation of Ethernet frame transmission. The module's transmit state machine performs the following functions to meet the IEEE 802.3 specifications.

- Generates preamble and SFD
- Generates jam pattern in Half-Duplex mode
- Jabber timeout
- Flow control for Half-Duplex mode (back pressure)
- Generates transmit frame status
- Contains time stamp snapshot logic for IEEE 1588 support

When a new frame transmission from the TFC is requested, the transmit state machine sends out the preamble and SFD, followed by the data received. The preamble is defined as 7 bytes of 10101010_B pattern, and the SFD is defined as 1 byte of 10101011_B pattern.

The collision window is defined as 1 slot time (512 bit times for 10/100 Mbit/s Ethernet). The jam pattern generation is applicable only to Half-Duplex mode, not to Full-Duplex mode. In Full-Duplex mode, the transmit state machine ignores the collision signal from the PHY.

In MII mode, if a collision occurs any time from the beginning of the frame to the end of the CRC field, the transmit state machine sends a 32-bit jam pattern of 55555555_H on the MII to inform all other stations that a collision has occurred. If the collision is seen during the preamble transmission phase, the transmit state machine completes the transmission of preamble and SFD and then sends the jam pattern.

If the collision occurs after the collision window and before the end of the FCS field (or the end of Burst if the Frame Burst mode is enabled), the transmit state machine sends a 32-bit jam pattern and sets the late collision bit in the transmit frame status.

The TPE module maintains a jabber timer to cut off the transmission of Ethernet frames if the TFC module transfers more than 2048 (default) bytes. The time-out is changed to 10240 bytes when the Jumbo frame is enabled.

The Transmit state machine uses the deferral mechanism for the flow control (Back Pressure) in Half-Duplex mode. When the Application requests to stop receiving frames, the Transmit state machine sends a JAM pattern of 32 bytes whenever it senses a reception of a frame, provided the transmit flow control is enabled. This will result in a collision and the remote station will back off. The Application requests the flow control by setting **ETH0_FLOW_CONTROL**.FCA_BPA bit. If the application requests a frame to be transmitted, then it will be scheduled and transmitted even when the backpressure is activated. Note that if the backpressure is kept activated for a long time (and more than 16 consecutive collision events occur) then the remote stations will abort their transmissions due to excessive collisions.

If IEEE 1588 time stamping is enabled for the transmit frame, this block takes a snapshot of the system time when the SFD is put onto the transmit MII bus. The system time source is either an external input or internally generated, according to the configuration selected.

Transmit Scheduler Module

The Transmit Scheduler (STX) module is responsible for scheduling the frame transmission on the MII. The two major functions of this module are to maintain the inter-frame gap between two transmitted frames and to follow the Truncated Binary Exponential Back-off algorithm for Half-Duplex mode. This module provides an enable signal to the TPE module after satisfying the IFG and Back-off delays.

The STX module maintains an idle period of the configured inter-frame gap (**ETH0_MAC_CONFIGURATION**.IFG bits) between any two transmitted frames. If frames from the TFC arrive at the TPE module sooner than the configured IFG time, the TPE module waits for the enable signal from the STX module before starting the transmission on the MII. The STX module starts its IFG counter as soon as the carrier signal of the MII goes inactive. At the end of programmed IFG value, the module issues an enable signal to the TPE module in Full-Duplex mode. In Half-Duplex mode and when IFG is configured for 96 bit times, the STX module follows the rule of deference specified in Section 4.2.3.2.1 of the IEEE 802.3 specification. The module resets its IFG counter

if a carrier is detected during the first two-thirds (64-bit times for all IFG values) of the IFG interval. If the carrier is detected during the final one third of the IFG interval, the STX module continues the IFG count and enables the transmitter after the IFG interval.

The STX module implements the Truncated Binary Exponential Back-off algorithm when it operates in Half-Duplex mode.

Transmit CRC Generator Module

The Transmit CRC Generator (CTX) module interfaces with the TFC module to generate CRC for the FCS field of the Ethernet frame. The TFC module sends the frame data and any necessary padding to the CTX module through an 8-bit interface.

This module calculates the 32-bit CRC for the FCS field of the Ethernet frame. The encoding is defined by the following generating polynomial.

$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

The module gets the Ethernet frame's byte data from the TFC module (DA + SA + LT + DATA + PAD) qualified with a Data Valid signal. The TFC also indicates to the CTX when to reset the previously calculated CRC and to start the new CRC calculation for the coming frame. The TFC module issues the start command before sending the new frame data for calculation. The calculated CRC is valid on the next clock after the data is received.

Transmit Flow Control Module

The Transmit Flow Control (FTX) module generates Pause frames and transmit them to the TFC module as necessary, in Full-Duplex mode. The TFC module receives the Pause frame from the FTX module, appends the calculated CRC, and sends the frame to the TPE module. Pause frame generation can be initiated in two ways. The Application can request the FTX module to send a Pause frame either by setting the **ETH0_FLOW_CONTROL**.FCB bit or in response to the receive FIFO full conditions (packet buffer).

If the Application has requested the flow control by setting the FLOW_CONTROL.FCB bit of, the FTX module will generate and transmit a single Pause frame to the TFC module. The value of the Pause Time in the generated frame contains the programmed Pause Time value in the FLOW_CONTROL Register. To extend the pause or end the pause prior to the time specified in the previously transmitted Pause frame, the application must request another Pause frame transmission after programming the Pause Time register with appropriate value.

If the Application has requested the flow control by asserting the mti_flowctrl_i signal, the FTX module will generate and transmit a Pause frame to the TFC module. The value of the Pause Time in the generated frame contains the programmed Pause Time value in the FLOW_CONTROL Register. The FTX module monitors the MTI Flow Control Signal . If it remains asserted at a configurable number of slot-times (FLOW_CONTROL.PLT

bits) before this Pause-time runs-out, a second Pause frame will be transmitted to the TFC module. The process will be repeated as long as the MTI flow control signal remains asserted.

If the MTI flow control signal goes inactive prior to the sampling time, the FTX module will transmit a Pause frame with zero Pause Time to indicate to the remote end that the receive buffer is ready to receive new data frames.

15.2.1.2 MAC Transmit Interface Protocol

The MAC Transmit Interface (MTI) connects the application with the MTL in the ETH to provide the Ethernet data for transmission.

The application initiates the Ethernet frame transmission by writing the first data of the frame to the ETH, provided the ETH is ready to accept data . The Application can push-in data as long as the ETH core is ready to accept it.

If the frame transmission is not successful (due to underflow, collision, jabber timeout, excessive deferral events), the ETH core will assert the transmit status even before the EOF is received. The Application will have to take the appropriate action as per the status. The ETH will drop all further data input to it until the next SOF.

15.2.1.3 Reception

A receive operation is initiated when the ETH detects an SFD on the MII. The core strips the preamble and SFD before proceeding to process the frame. The header fields are checked for the filtering and the FCS field used to verify the CRC for the frame. The received frame is stored in a shallow buffer until the address filtering is performed. The frame is dropped in the core if it fails the address filter.

The following are the functional blocks in the Receive path of the ETH core.

- Receive Protocol Engine Module (RPE)
- Receive CRC Module (CRX)
- Receive Frame Controller Module (RFC)
- Receive Flow Control Module (FRX)
- Receive IP Checksum checker (IPC)
- Receive Bus Interface Unit Module (RBU)
- Address Filtering Module (AFM)

Receive Protocol Engine Module

The RPE consists of the receive state machine which strips the preamble SFD. Once the external PHY detects ethernet traffic, the RPE’s receive state machine begins hunting for the SFD field from the receive modifier logic. Until then, the state machine drops the receiving preambles. Once the SFD is detected, the state machine begins sending the data of the Ethernet frame to the RFC module, beginning with the first byte following the SFD (destination address).

If IEEE 1588 time stamping is enabled, the RPE takes a snapshot of the system time when any frame's SFD is detected on the MII. Unless the MAC filters out and drops the frame, this time stamp is passed on to the application.

In MII mode, the RPE converts the received nibble data into bytes, then forwards the valid frame data to the RFC module

The receive state machine of the RPE module decodes the Length/Type field of the receiving Ethernet frame. If the Length/Type field is less than 600 (hex) and if the MAC is programmed for the auto crc/pad stripping option, the state machine sends the data of the frame up to the count specified in the Length/Type field, then starts dropping bytes (including the FCS field). The state machine of the RPE module decodes the Length/Type field and checks for the Length interpretation.

If the Length/Type field is greater than or equal to 600 (hex), the RPE module will send all received Ethernet frame data to the RFC module, irrespective of the value on the programmed auto-CRC strip option.

As a default, the ETH is programmed for watchdog timer to be enabled, that is, frames above 2.048 (10.240 if Jumbo Frame is enabled) bytes (DA + SA + LT + DATA + PAD + FCS) are cut off at the RPE module. This feature can be disabled by programming the **ETH0_MAC_CONFIGURATION**.WD bit. However even if the watchdog timer is disabled, frames greater than 16 KB in size are cut off and a watchdog time-out status is given.

The ETH supports loopback of transmitted frames onto its receiver. As a default, the ETH loopback function is disabled, but this feature can be enabled by programming the ETH Configuration register, Loopback bit. The transmit and receive clocks can have an asynchronous timing relationship, so an asynchronous FIFO is used to make the loopback path of the PHY transmit path connected onto the receive path. The asynchronous FIFO is 6 bits wide to accommodate the PHY transmit,receive and enable signals. The FIFO is nine words deep and free-running to write on the write clock and read on every read clock.

The write and read pointers gets re-initialized to have an offset of 4 at the start of each frame read out of the FIFO. This helps to avoid overflow/underflow during the transfer of a frame, and ensures that the overflow/underflow occurs only during the IFG period between the frames. Please note that the FIFO depth of nine is sufficient to prevent data corruption for frame sizes up to 9.022 bytes with a difference of 200 ppm between the MII Transmit and Receive clock frequencies. Hence, bigger frames should not be looped back, as they may get corrupted in this loopback FIFO.

At the end of every received frame, the RPE module generates received frame status and sends it to the RFC module. Control, missed frame, and filter fail status are added to the receive status in the RFC module.

Receive CRC Module

The Receive CRC (CRX) interfaces to the RPE module to check for any CRC error in the receiving frame.

This module calculates the 32-bit CRC for the received frame that includes the Destination address field through the FCS field. The encoding is defined by the following generating polynomial.

$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

The module gets the data from the RPE module (DA+SA+LT+DATA+PAD+FCS). The RPE module also sends a control signal that indicates the validity of the data. Irrespective of the auto pad/CRC strip, the CRX module receives the entire frame to compute the CRC check for received frame. As a note on the auto pad/CRC strip settings, the entire frame is not transferred between the RPE and RFC 8-bit interface.

Receive Checksum Offload Engine

The Receive Checksum Offload engine can detect both IPv4 and IPv6 frames in the received ethernet packets. Once an IP frame is detected it is processed for data integrity.

The Receive Checksum Offload engine is enabled by setting the **ETH0_MAC_CONFIGURATION**.IPC bit. The ETH receiver identifies IPv4 or IPv6 frames by checking for value 0800_H or $86DD_H$, respectively, in the received Ethernet frames' Type field. This identification applies to VLAN-tagged frames as well.

The Receive Checksum Offload engine calculates IPv4 header checksums and checks that they match the received IPv4 header checksums. The result of this operation (pass or fail) is given to the RFC module for insertion into the receive status word. The IP Header Error bit is set for any mismatch between the indicated payload type (Ethernet Type field) and the IP header version, or when the received frame does not have enough bytes, as indicated by the IPv4 header's Length field (or when fewer than 20 bytes are available in an IPv4 or IPv6 header).

This engine also identifies a TCP, UDP or ICMP payload in the received IP datagrams (IPv4 or IPv6) and calculates the checksum of such payloads properly, as defined in the TCP, UDP, or ICMP specifications. This engine includes the TCP/UDP/ICMPv6 pseudo-header bytes for checksum calculation and checks whether the received checksum field matches the calculated value. The result of this operation is given as a Payload Checksum Error bit in the receive status word. This status bit is also set if the length of the TCP, UDP, or ICMP payload does not tally to the expected payload length given in the IP header.

As mentioned in **TCP/UDP/ICMP Checksum Engine**, this engine bypasses the payload of fragmented IP datagrams, IP datagrams with security features, IPv6 routing headers, and payloads other than TCP, UDP or ICMP.

In this configuration, the core does not append any payload checksum bytes to the received Ethernet frames.

Receive Frame Controller Module

The Receive Frame Controller (RFC) receives the Ethernet frame data and status from the RPE module. The RFC module consists of a FIFO of parameterized depth (default set to 4 deep and 37 bits wide) and two state machines for writing and reading the FIFO. The FIFO holds the received Ethernet frame data and byte enables, along with a control bit to indicate the last data. The state machines manage the FIFO and provide a frame buffering for the receiving Ethernet frame from the RPE module. The main functions of the RFC module are:

- Data path conversion, which converts the 8-bit data to 32-bit data to the RBU module.
- Frame filtering
- Attaching the calculated IP Checksum input from IPC.
- Update the Receive Status and forward to RBU.

If the **ETH0_MAC_FRAME_FILTER**.RA bit is set, the RFC module initiates the data transfer to the RBU module as soon as 4 bytes of Ethernet data are received from the RPE module. At the end of the data transfer, the RFC module sends out the received frame status that includes the frame filter bits (MAC_FRAME_FILTER.SA Filterfail and MAC_FRAME_FILTER.DA Filterfail) and status from the RFC module. These bits are generated based on the filter-fail signals from the AFM module. This status bit indicates to the Application whether the received frame has passed the filter controls (both address filter and Frame Filter controls from CSR). The RFC module will not drop any frame on its own in this mode.

If the MAC_FRAME_FILTER.RA bit is reset, the RFC module performs frame filtering based on the destination/source address (the Application still needs to perform another level of filtering if it decides not to receive any bad frames like runt, CRC error frames, etc. The RFC module waits to receive the first 14 bytes of received data (type field) from the RPE module. Until then, the module will not initiate any transfers to the RBU module. After receiving the destination/source address bytes, the RFC checks the filter-fail signal from the AFM module for an address match. On detecting a filter-fail from AFB, the frame is dropped at the RFC module and not transferred to the Application.

On a delayed filter response from the AFM (this can only occur if you change the AFM logic), the RFC module waits until the FIFO is full, and then proceeds with the frame transfer to the RBU module. However, it will still take the delayed response from the AFM module and if it is a (DA/SA) filter failure, then it will drop the rest of the frame and send the Rx Status Word (with zero frame-length, CRC Error and Runt Error bits set) immediately indicating the filter-fail. If there is no response from the AFM until the end of frame is transmitted, the filter fail status in the Rx Status Word is updated accordingly.

When the PMT module is configured for power-down mode, all received frames are dropped by this block, and are not forwarded to the application.

Receive Flow Control Module

The Receive Flow Controller (FRX) detects the receiving Pause frame and pauses the frame transmission for the delay specified within the received Pause frame. The FRX module is enabled only in Full-Duplex mode. The Pause frame detection function can be enabled or disabled with the **ETH0_FLOW_CONTROL**.RFE bit.

Once the receive flow control is enabled, the FRX module begins monitoring the received frame destination address for any match with the multicast address of the control frame ($0180C2000001_H$). If a match is detected, the FRX module indicates to the RFC module, that the destination address of the received frame matches the reserved control frame destination address. The RFC module then decides whether or not to transfer the received control frame to the Application, based on the **ETH0_MAC_FRAME_FILTER**.PCF bit .

The FRX module also decodes the Type, Op-code, and Pause Timer field of the receiving control frame. At the end of received frame, the FRX module gets the received frame status from RPE. If the byte count of the status indicates 64 bytes, and if there is no CRC error, the FRX module requests the MAC transmitter to pause the transmission of any data frame for the duration of the decoded Pause Time value, multiplied by the slot time (64 byte times). Meanwhile, if another Pause frame is detected with a zero Pause Time value, the FRX module resets the Pause Time and gives another pause request to the Transmitter. If the received control frame matches neither the Type field (8808_H), Opcode (00001_H), nor byte length (64 bytes), or if there is a CRC error, the FRX module does not generate a Pause request to Transmitter.

In the case of a pause frame with a multicast destination address, the RFC filters the frame based on the address match from the FRX module. For a pause frame with a unicast destination address, the filtering in the FRX module depends on whether the DA matched the contents of the MAC Address Register 0 and the **ETH0_FLOW_CONTROL**.UP bit is set (detecting a pause frame even with a unicast destination address). The MAC_FRAME_FILTER.PCF register bits control the filtering for control frames in addition to the Address filter module.

Receive Bus Interface Unit Module

The Receive Bus Interface Unit (RBU) converts the 32-bit data received from the RFC module into a 32-bit FIFO protocol on the Application side. The RBU module interfaces with the Application through the MAC receive interface (MRI).

If IEEE 1588 time stamping is enabled, the RBU also outputs the time stamp captured from the received frame.

Address Filtering Module

The Address Filtering (AFM) module performs the destination and source address checking function on all received frames and reports the address filtering status to the

RFC module. The address checking is based on different parameters (Frame Filter register) chosen by the Application. These parameters are inputs to the AFM module as control signals, and the AFM module reports the status of the address filtering based on the combination of these inputs. The AFM module does not filter the receive frames by itself, but reports the status of the address filtering (whether to drop the frame or not) to the RFC module. The AFM module also reports whether the receiving frame is a multicast frame or a broadcast frame, as well as the address filter status.

The AFM module probes the 8-bit receive data path between the RPE module and the RFC module and checks the destination and source address field of each incoming packet. In MII mode the module takes 14/26 clocks (from the start of frame) to compare the destination/ source address of the receiving frame. The AFM module gets the station's physical (MAC) address and the Multicast Hash table from CSR module for address checking. The CSR module provides the Frame Filter register parameters to AFM.

Unicast Destination Address Filter

The AFM supports up to 4 MAC addresses for unicast perfect filtering. If perfect filtering is selected (HUC bit of Frame Filter register is reset), the AFM compares all 48 bits of the received unicast address with the programmed MAC address for any match. Default MacAddr0 is always enabled, other addresses MacAddr1–MacAddr3 are selected with an individual enable bit. Each byte of these other addresses (MacAddr1–MacAddr3) can be masked during comparison with the corresponding received DA byte by setting the corresponding Mask Byte Control bit in the register. This helps group address filtering for the DA.

In Hash filtering mode (When HUC bit is set), the AFM performs imperfect filtering for unicast addresses using a 64-bit Hash table. For hash filtering, the AFM uses the upper 6 bits CRC of the received destination address to index the content of the Hash table. A value of 000000 selects Bit 0 of the selected register, and a value of 111111 selects Bit 63 of the Hash Table register. If the corresponding bit (indicated by the 6-bit CRC) is set to 1, the unicast frame is said to have passed the Hash filter; otherwise, the frame has failed the Hash filter.

Multicast Destination Address Filter

The ETH can be programmed to pass all multicast frames by setting the **ETH0_MAC_FRAME_FILTER**.PM bit. If the MAC_FRAME_FILTER.PM bit is reset, the AFM performs the filtering for multicast addresses based on the MAC_FRAME_FILTER.HMC bit. In Perfect Filtering mode, the multicast address is compared with the programmed MAC Destination Address registers (1–31). Group address filtering is also supported.

In Hash filtering mode, the AFM performs imperfect filtering using a 64-bit Hash table. For hash filtering, the AFM uses the upper 6 bits CRC of the received multicast address

to index the content of the Hash table. A value of 000000_B selects Bit 0 of the selected register and a value of 111111_B selects Bit 63 of the Hash Table register.

If the corresponding bit is set to 1, then the multicast frame is said to have passed the Hash filter; otherwise, the frame has failed the Hash filter.

Hash or Perfect Address Filter

The DA filter can be configured to pass a frame when its DA matches either the Hash filter or the Perfect filter by setting the MAC_FRAME_FILTER.HPF bit and setting the corresponding **ETH0_MAC_FRAME_FILTER**.HUC or MAC_FRAME_FILTER.HMC bits. This configuration applies to both unicast and multicast frames. If the HPF bit is reset, only one of the filters (Hash or Perfect) is applied to the received frame.

Broadcast Address Filter

The AFM doesn't filter any broadcast frames in the default mode. However, if the ETH is programmed to reject all broadcast frames by setting the MAC_FRAME_FILTER.DBF bit, the DAF module asserts the Filter fail signal to RFC, whenever a broadcast frame is received. This will tell the RFC module to drop the frame.

Unicast Source Address Filter

The ETH can also perform a perfect filtering based on the source address field of the received frames. By default, the AFM compares the SA field with the values programmed in the SA registers. The MAC Address registers [1:3] can be configured to contain SA instead of DA for comparison, by setting Bit 30 of the corresponding Register. Group filtering with SA is also supported. The frames that fail the SA Filter are dropped by the ETH if the MAC_FRAME_FILTER.SAF bit of Frame Filter register is set.

When MAC_FRAME_FILTER.SAF bit is set, the result of SA Filter and DA filter is AND'ed to decide whether the frame needs to be forwarded. This means that either of the filter fail result will drop the frame and both filters have to pass in-order to forward the frame to the application.

Inverse Filtering Operation

For both Destination and Source address filtering, there is an option to invert the filter-match result at the final output. These are controlled by the DAIF and SAIF bits of the Frame Filter register respectively. The MAC_FRAME_FILTER.DAIF bit is applicable for both Unicast and Multicast DA frames. The result of the unicast/multicast destination address filter is inverted in this mode. Similarly, when the MAC_FRAME_FILTER.SAIF bit is set, the result of unicast SA filter is reversed.

Table 15-2 and **Table 15-3** summarize the Destination and Source Address filtering based on the type of frames received.

Table 15-2 Destination Address Filtering Table

Frame Type	PR	HPF	HUC	DAIF	HMC	PM	DB	DA Filter Operation
Broadcast	1	X	X	X	X	X	X	Pass
	0	X	X	X	X	X	0	Pass
	0	X	X	X	X	X	1	Fail
Unicast	1	X	X	X	X	X	X	Pass all frames.
	0	X	0	0	X	X	X	Pass on Perfect/Group filter match.
	0	X	0	1	X	X	X	Fail on Perfect/Group filter match.
	0	0	1	0	X	X	X	Pass on Hash filter match.
	0	0	1	1	X	X	X	Fail on Hash filter match.
	0	1	1	0	X	X	X	Pass on Hash or Perfect/Group filter match.
	0	1	1	1	X	X	X	Fail on Hash or Perfect/Group filter match.
Multicast	1	X	X	X	X	X	X	Pass all frames.
	X	X	X	X	X	1	X	Pass all frames.
	0	X	X	0	0	0	X	Pass on Perfect/Group filter match and drop PAUSE control frames if PCF = 0x.
	0	0	X	0	1	0	X	Pass on Hash filter match and drop PAUSE control frames if PCF = 0x.
	0	1	X	0	1	0	X	Pass on Hash or Perfect/Group filter match and drop PAUSE control frames if PCF = 0x.
	0	X	X	1	0	0	X	Fail on Perfect/Group filter match and drop PAUSE control frames if PCF = 0x.

Table 15-2 Destination Address Filtering Table (cont'd)

Frame Type	PR	HPF	HUC	DAIF	HMC	PM	DB	DA Filter Operation
	0	0	X	1	1	0	X	Fail on Hash filter match and drop PAUSE control frames if PCF = 0x.
	0	1	X	1	1	0	X	Fail on Hash or Perfect/Group filter match and drop PAUSE control frames if PCF = 0x.

Table 15-3 Source Address Filtering Table

Frame Type	PR	SAIF	SAF	SA Filter Operation
Unicast	1	X	X	Pass all frames.
	0	0	0	Pass status on Perfect/Group filter match but do not drop frames that fail.
	0	1	0	Fail status on Perfect/Group filter match but do not drop frame.
	0	0	1	Pass on Perfect/Group filter match and drop frames that fail.
	0	1	1	Fail on Perfect/Group filter match and drop frames that fail.

15.2.2 MAC Transaction Layer (MTL)

The MAC Transaction Layer provides FIFO memory to buffer and regulate the frames between the application system memory and the ETH core. It also enables the data to be transferred between the application clock domain and the ETH clock domains. The MTL layer has 2 data paths, namely the Transmit path and the Receive Path. The data path for both directions is 32-bit wide and operates with a simple FIFO protocol.

The ETH-MTL communicates with the application side with the Application Transmit Interface (ATI), Application Receive Interface (ARI), and the MAC Control Interface (MCI).

15.2.2.1 Transmit Path

DMA controls all transactions for the transmit path through the ATI. Ethernet frames read from the system memory is pushed into the FIFO by the DMA. The frame is then popped

out and transferred to the ETH core when triggered. When the end-of-frame is transferred, the status of the transmission is taken from the ETH core and transferred back to the DMA.

The Transmit FIFO has a depth of 22K bytes. A 2 FIFO-fill level is indicated to the DMA so that it can initiate a data fetch in required bursts from the system memory, using the Bus interface. The data from the Bus Master interface is pushed into the FIFO with the appropriate byte lanes qualified by the DMA. The DMA also indicates the start-of-frame (SOF) and end-of-frame (EOF) transfers along with a few signals controlling the pad-insertion/CRC generation for that frame in the ETH core.

Per-frame control bits, such as Automatic Pad/CRC Stripping disable, time stamp capture, and so forth are taken as control inputs on the ATI, stored in a separate register FIFO, and passed on to the core transmitter when the corresponding frame data is read from the Transmit FIFO.

There are two modes of operation for popping data towards the ETH core. In Threshold mode, as soon as the number of bytes in the FIFO crosses the configured threshold level (or when the end-of-frame is written before the threshold is crossed), the data is ready to be popped out and forwarded to the ETH core. The threshold level is configured using the TTC bits of DMA **ETH0_BUS_MODE** Register. In store-and-forward mode, the MTL pops the frame towards the ETH core only when one or more of the following conditions are true:

- When a complete frame is stored in the FIFO
- When the TX FIFO becomes almost full
- When the ATI watermark becomes low. The watermark becomes low when the requested FIFO does not have space to accommodate the requested burst-length on the ATI.

Therefore, the MTL never stops in the store-and-forward mode even if the Ethernet frame length is bigger than the Tx FIFO depth.

The application can flush the Transmit FIFO of all contents by setting the **ETH0_OPERATION_MODE**.FTF bit. This bit is self-clearing and initializes the FIFO pointers to the default state. If the FTF bit is set during a frame transfer from the MTL to the ETH core, then the MTL stops further transfer as the FIFO is considered to be empty. Hence an underflow event occurs at the ETH transmitter and the corresponding Status word is forwarded to the DMA.

Initialization through **Transmit Status Word** detail initialization and transmit operations for the MTL Layer.

Initialization

Upon reset, the MTL is ready to manage the flow of data to and from the DMA and the ETH .

There are no requirements for enabling the MTL. However, the ETH block and the DMA controller must be enabled individually through their respective CSRs.

Single-Packet Transmit Operation

During a transmit operation, the MTL block is slaved to the DMA controller. The general sequence of events for a transmit operation is as follows.

1. If the system has data to be transferred, the DMA controller, if enabled, fetches data from the XMC4500 RAM through the Bus Master interface and starts forwarding it to the MTL. The MTL pushes the data received from the DMA into the FIFO. It continues to receive the data until the end-of frame of the frame is transferred.
2. The data is taken out of the FIFO and sent to the MAC by the FIFO controller engine. When the threshold level is crossed or a full packet of data is received into the FIFO, the MTL pops out the frame data and drives them to the ETH core. The engine continues to transfer data from the FIFO until a complete packet has been transferred to the MAC. Upon completion of the frame, the MTL receives the Status from the ETH and then notifies the DMA controller

Transmit Operation—Two Packets in the Buffer

1. Because the DMA must update the descriptor status before releasing it to the CPU, there can be at the most two frames inside a transmit FIFO. The second frame will be fetched by the DMA and put into the FIFO only if the OSF (Operate on Second Frame bit is set). If this bit is not set, the next frame will be fetched from the memory only after the MAC has completely processed the frame and the DMA has released the descriptors.
2. If the OSF bit is set, the DMA starts fetching the second frame immediately after completing the transfer of the first frame to the FIFO. It does not wait for the status to be updated. The MTL, in the meantime, receives the second frame into the FIFO while transmitting the first frame. As soon as the first frame has been transferred and the status is received from the MAC, the MTL pushes it to the DMA. If the DMA has already completed sending the second packet to the MTL, it must wait for the status of the first packet before proceeding to the next frame.

Transmit Operation—Multiple Packets in Buffer

In ETH -MTL configuration, the transmit FIFO can be configured to accept more than 2 packets at a time. This option limits the number of status words that can be stored in the MTL before it is transferred to the DMA/CPU. By default, this number is limited to 2 but can be configured for 4 or 8 as well. Once the MTL FIFO accepts the number of frames equal to the status FIFO depth, it will stop accepting further frames unless the transmit Status that is given out and accepted by the CPU/DMA thus freeing up the space in this small FIFO.

Retransmission During Collision

While a frame is being transferred from the MTL to the ETH , a collision event occurs on the ETH line interface in Half-Duplex mode. The ETH then indicates a retry attempt to the MTL by giving the status even before the end-of-frame is transferred from MTL. Then the MTL will enable the retransmission by popping out the frame again from the FIFO.

After more than 96 bytes are popped towards the ETH core, the FIFO controller frees up that space and makes it available to the DMA to push in more data. This means that the retransmission is not possible after this threshold is crossed or when the ETH core indicates a late-collision event.

Transmit FIFO Flush Operation

The ETH provides a control signal to the software to flush the Transmit FIFO in the MTL layer through the use of the **ETH0_OPERATION_MODE**.FTF bit. The Flush operation is immediate and the MTL clears the Tx FIFO and the corresponding pointers to the initial state even if it is in the middle of transferring a frame to the ETH Core. The data which is already accepted by the MAC transmitter will not be flushed. It will be scheduled for transmission and will result in underflow as TxFIFO does not complete the transfer of rest of the frame. As in all underflow conditions, a runt frame will be transmitted and observed on the line. The status of such a frame will be marked with both Underflow and Frame Flush events ($TDES0_{RAM}$ bits 13 and 1).

The MTL layer also stops accepting any data from the application (DMA) during the Flush operation. It will generate and transfer Transmit Status Words to the application for the number of frames that is flushed inside the MTL (including partial frames). Frames that are completely flushed in the MTL will have the Frame Flush Status bit (TDES0 13_{RAM}) set. The MTL completes the Flush operation when the application (DMA) accepts all of the Status Words for the frames that were flushed, and then clears the Transmit FIFO Flush control register bit. At this point, the MTL starts accepting new frames from the application (DMA).

Transmit Status Word

At the end of transfer of the Ethernet frame to the ETH core and after the core completes the transmission of the frame, the MTL outputs the transmit status to the application.The detailed description of the Transmit Status is the same as for bits [23:0] of $TDES0_{RAM}$, given in **Table 15-9**.

If IEEE 1588 time stamping is enabled, the MTL returns specific frame's 64-bit time stamp, along with the ATI's transmit status.

Transmit Checksum Offload Engine

Communication protocols such as TCP and UDP implement checksum fields, which help determine the integrity of data transmitted over a network. Because the most widespread

use of Ethernet is to encapsulate TCP and UDP over IP datagrams, the ETH has an Checksum Offload Engine (COE) to support checksum calculation and insertion in the transmit path, and error detection in the receive path. This section explains the operation of the Checksum Offload Engine for transmitted frames.

*Note: The checksum for TCP, UDP, or ICMP is calculated over a complete frame, then inserted into its corresponding header field. Due to this requirement, this function is enabled only when the Transmit FIFO is configured for Store-and-Forward mode (that is, when the **ETH0_OPERATION_MODE**.TSF bit is set .). If the core is configured for Threshold (cut-through) mode, the Transmit COE is bypassed.*

Note: You must make sure that the Transmit FIFO is deep enough to store a complete frame before that frame is transferred to the ETH *Core transmitter. The reason being that when space is not available to accept the programmed burst length of the data, then the MTL TxFIFO starts reading to avoid dead-lock. Once reading starts, then checksum insertion engine fails and consequently all succeeding frames may get corrupted due to improper recovery. Therefore, you must enable the checksum insertion only in the frames that are less than the following number of bytes in size (even in the store-and-forward mode):*
FIFO Depth – PBL – 3 FIFO Locations
*The **ETH0_BUS_MODE**.PBL is the programmed burst-length.*

This checksum engine can be controlled for each frame by setting the CIC bits (Bits 28:27 of $TDES1_{RAM}$, described in **Transmit Descriptor 1**).

Note: See IETF specifications RFC 791, RFC 793, RFC 768, RFC 792, RFC 2460, and RFC 4443 for IPv4, TCP, UDP, ICMP, IPv6, and ICMPv6 packet header specifications, respectively.

IP Header Checksum Engine

In IPv4 datagrams, the integrity of the header fields is indicated by the 16-bit Header Checksum field (the eleventh and twelfth bytes of the IPv4 datagram). The COE detects an IPv4 datagram when the Ethernet frame's Type field has the value 0800_H and the IP datagram's Version field has the value 4_H. The input frame's checksum field is ignored during calculation and replaced with the calculated value.

IPv6 headers do not have a checksum field; thus, the COE does not modify IPv6 header fields.

The result of this IP header checksum calculation is indicated by the IP Header Error status bit in the Transmit status (Bit 16 in **Table 15-9**). This status bit is set whenever the values of the Ethernet Type field and the IP header's Version field are not consistent, or when the Ethernet frame does not have enough data, as indicated by the IP header Length field.

In other words, this bit is set when an IP header error is asserted under the following circumstances:

For IPv4 datagrams

- The received Ethernet type is 0800_H, but the IP header's Version field does not equal 4_H
- The IPv4 Header Length field indicates a value less than 5_H (20 bytes)
- The total frame length is less than the value given in the IPv4 Header Length field

For IPv6 datagrams

- The Ethernet type is $86DD_H$ but the IP header Version field does not equal 6_H
- The frame ends before the IPv6 header (40 bytes) or extension header (as given in the corresponding Header Length field in an extension header) is completely received.

Even when the COE detects such an IP header error, it inserts an IPv4 header checksum if the Ethernet Type field indicates an IPv4 payload.

TCP/UDP/ICMP Checksum Engine

The TCP/UDP/ICMP Checksum Engine processes the IPv4 or IPv6 header (including extension headers) and determines whether the encapsulated payload is TCP, UDP, or ICMP.

Note: For non-TCP, -UDP, or -ICMP/ICMPv6 payloads, this checksum engine is bypassed and nothing further is modified in the frame.

Note: Fragmented IP frames (IPv4 or IPv6), IP frames with security features (such as an authentication header or encapsulated security payload), and IPv6 frames with routing headers are not processed by this engine, and therefore must be bypassed. In other words, payload checksum insertion must not be enabled for such frames.

The checksum is calculated for the TCP, UDP, or ICMP payload and inserted into its corresponding field in the header. This engine can work in the following two modes:

- In the first mode, the TCP, UDP, or ICMPv6 pseudo-header is not included in the checksum calculation and is assumed to be present in the input frame's Checksum field. This engine includes the Checksum field in the checksum calculation, then replaces the Checksum field with the final calculated checksum.
- In the second mode, the engine ignores the Checksum field, includes the TCP, UDP, or ICMPv6 pseudo-header data into the checksum calculation, and overwrites the checksum field with the final calculated value.

Note: For ICMP-over-IPv4 packets, the Checksum field in the ICMP packet must always be 16'h0000 in both modes, because pseudo-headers are not defined for such packets. If it does not equal 16'h0000, an incorrect checksum may be inserted into the packet.

The result of this operation is indicated by the Payload Checksum Error status bit in the Transmit Status vector (Bit 12 in **Table 15-9**). This engine sets the Payload Checksum

Error status bit when it detects that the frame has been forwarded to the MAC Transmitter engine in Store-and-Forward mode without the end-of-frame being written to the FIFO, or when the packet ends before the number of bytes indicated by the Payload Length field in the IP Header is received. When the packet is longer than the indicated payload length, the COE ignores them as stuff bytes, and no error is reported. When this engine detects the first type of error, it does not modify the TCP, UDP, or ICMP header. For the second error type, it still inserts the calculated checksum into the corresponding header field.

15.2.2.2 Receive Path

This module receives the frames given out by the ETH core and pushes them into the Rx FIFO. The status (fill level) of this FIFO is indicated to the DMA once it crosses the configured Receive threshold (**ETH0_OPERATION_MODE**.RTC bits). The MTL also indicates the FIFO fill level so that the DMA can initiate pre-configured burst transfers towards the Bus interface.

Receive Operation through **Receive Status Word** detail receive operations for the MTL Layer.

Receive Operation

During an Rx operation, the MTL is slaved to the ETH. The general sequence of Receive operation events is as follows:

1. When the ETH receives a frame, it pushes in data along with byte enables. The ETH also indicates the SOF and EOF. The MTL accepts the data and pushes it into the Rx FIFO. After the EOF is transferred, the ETH drives the status word, which is also pushed into the same Rx FIFO by the MTL.
2. When IEEE 1588 time stamping is enabled and the 64-bit time stamp is available along with the receive status, it is appended to the frame received from the ETH and is pushed into the RxFIFO before the corresponding receive status word is written. Thus, two additional locations per frame are taken for storing the time stamp in the RxFIFO.
3. The MTL_RX engine takes the data out of the FIFO and sends it to the DMA. In the default Cut-Through mode, when 64 bytes (configured with the **ETH0_OPERATION_MODE**.RTC bits or a full packet of data are received into the FIFO, the MTL_RX engine pops out the data and indicates its availability to the DMA. Once the DMA initiates the transfer to the Bus interface, the MTL_RX engine continues to transfer data from the FIFO until a complete packet has been transferred. Upon completion of the EOF frame transfer, the MTL pops out the status word and sends it to the DMA controller.
4. In Rx FIFO Store-and-Forward mode (configured by the Operation Mode.RSF bit), a frame is read out only after being written completely into the Receive FIFO. In this mode, all error frames are dropped (if the core is configured to do so) such that only

valid frames are read out and forwarded to the application. In Cut-Through mode, some error frames are not dropped, because the error status is received at the end-of-frame, by which time the start of that frame has already been read out of the FIFO.

Note: The time-stamp transfer takes two clock cycles and the lower 32-bit of the time-stamp is given out first. The status also may be extended to two cycles when Advanced Time-stamp feature is enabled.

Receive Operation Multiframe Handling

Since the status is available immediately following the data, the MTL is capable of storing any number of frames into the FIFO, as long as it is not full.

Error Handling

If the MTL Rx FIFO is full before it receives the EOF data from the ETH, an overflow is declared, the whole frame (including the status word) is dropped, and the overflow counter in the DMA (**ETH0_MISSED_FRAME_AND_BUFFER_OVERFLOW_COUNTER** Register) is incremented. This is true even if the Forward Error Frame (**ETH0_OPERATION_MODE**.FEF bit) is set. If the start address of such a frame has already been transferred to the Read Controller, the rest of the frame is dropped and a dummy EOF is written to the FIFO along with the status word. The status will indicate a partial frame due to overflow. In such frames, the Frame Length field is invalid.

The MTL Rx Control logic can filter error and undersized frames, if enabled (using the Operation Mode.FEF and Operation Mode.FUF bits). If the start address of such a frame has already been transferred to the Rx FIFO Read Controller, that frame is not filtered. The start address of the frame is transferred to the Read Controller after the frame crosses the receive threshold (set by the Operation Mode.RTC bits).

If the MTL Receive FIFO is configured to operate in Store-and-Forward mode, all error frames can be filtered and dropped.

Receive Status Word

At the end of the transfer of the Ethernet frame to the XMC4500 RAM, the MTL outputs the receive status to the Application. The detailed description of the receive status is the same as for Bits[31:0] of $RDES0_{RAM}$, given in **Table 15-4**, except that Bits 31, 14, 9, and 8 are reserved and have a reset of 0 by default. When the status of a partial frame due to overflow is given out, the Frame Length field in the status word is not valid.

Note: When Advanced Time Stamp feature is enabled, the status is composed of two parts - normal (default [31:0]), and extended. The extended status[63:32] gives the information about the received ethernet payload when it is carrying PTP packets or TCP/UDP/ICMP over IP packets. These are transferred over two clock cycles. The detailed description of the receive status is the same as described in RDES0

*and RDES4 in **Receive Descriptor**, except that bits 31, 14, 9, and 8 of normal status is reserved and have a reset value of 0_B. When the status of a partial frame due to overflow is given out, the Frame Length field in the status word is not valid.*

15.2.3 DMA Controller

The DMA has independent Transmit and Receive engines, and a CSR space. The Transmit Engine transfers data from system memory to the device port (MTL), while the Receive Engine transfers data from the device port to system memory. The controller utilizes descriptors to efficiently move data from source to destination with minimal CPU intervention. The DMA is designed for packet-oriented data transfers such as frames in Ethernet. The controller can be programmed to interrupt the CPU for situations such as Frame Transmit and Receive transfer completion, and other normal/error conditions.

The DMA and the CPU driver communicate through two data structures:

- Control and Status registers (CSR)
- Descriptor lists and data buffers

Control and Status registers are described in detail in **Chapter 15.6**. Descriptors are described in detail in **DMA Descriptors**.

The DMA descriptors are held in ram. To avoid confusion with the ETH registers the DMA descriptors use the subscript RAM for example $RDES0[0]_{RAM}$.

The DMA transfers data frames received by the core to the Receive Buffer in the XMC4500 memory, and Transmit data frames from the Transmit Buffer in the XMC4500 memory. Descriptors that reside in the XMC4500 memory act as pointers to these buffers.

There are two descriptor lists; one for reception, and one for transmission. The base address of each list is written into DMA RECEIVE_DESCRIPTOR_LIST_ADDRESS Register and TRANSMIT_DESCRIPTOR_LIST_ADDRESS Register, respectively. A descriptor list is forward linked (either implicitly or explicitly). The last descriptor may point back to the first entry to create a ring structure. Explicit chaining of descriptors is accomplished by setting the second address chained in both Receive and Transmit descriptors ($RDES1[24]_{RAM}$ and $TDES1[24]_{RAM}$). The descriptor lists resides in the XMC4500 physical memory address space. Each descriptor can point to a maximum of two buffers. This enables two buffers to be used, physically addressed, rather than contiguous buffers in memory.

A data buffer resides in the XMC4500 physical memory space, and consists of an entire frame or part of a frame, but cannot exceed a single frame. Buffers contain only data, buffer status is maintained in the descriptor. Data chaining refers to frames that span multiple data buffers. However, a single descriptor cannot span multiple frames. The DMA will skip to the next frame buffer when end-of-frame is detected. Data chaining can be enabled or disabled.

The descriptor ring and chain structure is shown in **Figure 15-2**.

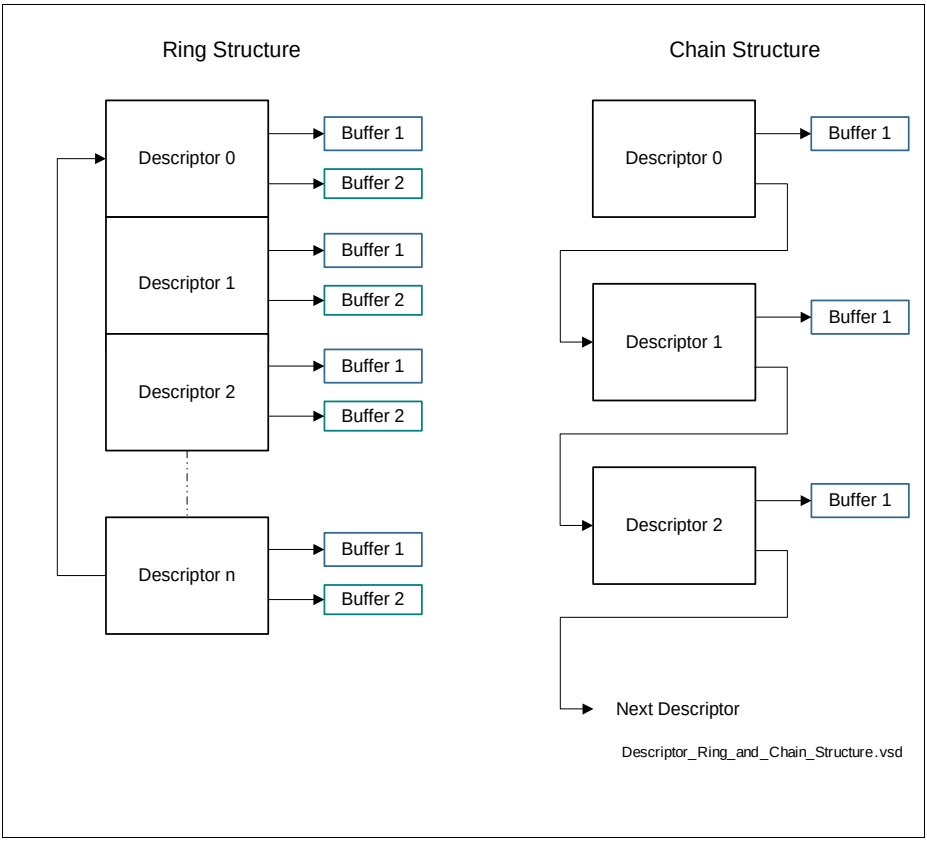


Figure 15-2 Descriptor Ring and Chain Structure

15.2.3.1 Initialization

Initialization for the ETH is as follows.

1. Write to **ETH0_BUS_MODE** Register to set XMC4500 bus access parameters.
2. Write to **ETH0_INTERRUPT_ENABLE** Register to mask unnecessary interrupt causes.
3. The software driver creates the Transmit and Receive descriptor lists. Then it writes to both DMA **ETH0_RECEIVE_DESCRIPTOR_LIST_ADDRESS** Register and DMA **ETH0_TRANSMIT_DESCRIPTOR_LIST_ADDRESS** Register, providing the DMA with the starting address of each list.

4. Write to ETH Registers **ETH0_TRANSMIT_POLL_DEMAND**, **ETH0_RECEIVE_POLL_DEMAND**, and Receive Descriptor List Aaddress for desired filtering options.
5. Write to **ETH0_MAC_CONFIGURATION** Register to configure and enable the Transmit and Receive operating modes. The **ETH0_MAC_CONFIGURATION**.DM bit is set based on the auto-negotiation result (read from the PHY).
6. Write to **ETH0_OPERATION_MODE**.ST and **ETH0_OPERATION_MODE**.SR bits start transmission and reception.
7. The Transmit and Receive engines enter the Running state and attempt to acquire descriptors from the respective descriptor lists. The Receive and Transmit engines then begin processing Receive and Transmit operations. The Transmit and Receive processes are independent of each other and can be started or stopped separately.

XMC4500 Bus Burst Access

The DMA will attempt to execute fixed-length Burst transfers on the Bus Master interface if configured to do so (**ETH0_BUS_MODE**.FB Register). The maximum Burst length is indicated and limited by the PBL field (Bus Mode.PBL Register). The Receive and Transmit descriptors are always accessed in the maximum possible (limited by PBL or (16 * 8)/32) burst-size for the 16-bytes to be read.

The Transmit DMA will initiate a data transfer only when sufficient space to accommodate the configured burst is available in MTL Transmit FIFO or the number of bytes till the end of frame (when it is less than the configured burst-length). The DMA will indicate the start address and the number of transfers required to the Bus Master Interface. When the Bus Interface is configured for fixed-length burst, then it will transfer data using the best combination of INCR4/8 and SINGLE transactions.

The Receive DMA will initiate a data transfer only when sufficient data to accommodate the configured burst is available in MTL Receive FIFO or when the end of frame (when it is less than the configured burst-length) is detected in the Receive FIFO. The DMA will indicate the start address and the number of transfers required to the Bus Master Interface. When the Bus Interface is configured for fixed-length burst, then it will transfer data using the best combination of INCR4/8 and SINGLE transactions. If the end-of frame is reached before the fixed-burst ends on the Bus interface, then dummy transfers are performed in-order to complete the fixed-burst.

When the Bus interface is configured for address-aligned beats, both DMA engines ensure that the first burst transfer the Bus initiates is less than or equal to the size of the configured PBL. Thus, all subsequent beats start at an address that is aligned to the configured PBL. The DMA can only align the address for beats up to size (for PBL >), because the Bus interface does not support more than INCR8/16.

XMC4500 Data Buffer Alignment

The Transmit and Receive data buffers do not have any restrictions on start address alignment. For example, the start address for the buffers can be aligned to any of the four bytes. However, the DMA always initiates transfers with address aligned to the bus width with dummy data for the byte lanes not required. This typically happens during the transfer of the beginning or end of an Ethernet frame.

Example - Buffer Read

If the Transmit buffer address is $00000FF2_H$, and 15 bytes need to be transferred, then the DMA will read five full words from address $00000FF0_H$, but when transferring data to the MTL Transmit FIFO, the extra bytes (the first two bytes) will be dropped or ignored. Similarly, the last 3 bytes of the last transfer will also be ignored. The DMA always ensures it transfers a full 32-bit data to the MTL Transmit FIFO, unless it is the end-of-frame.

Buffer Size Calculations

The DMA does not update the size fields in the Transmit and Receive descriptors. The DMA updates only the status fields ($RDES_{RAM}$ and $TDES_{RAM}$) of the descriptors. The driver has to perform the size calculations.

The transmit DMA transfers to the ETH the exact number of bytes (indicated by buffer size field of $TDES1_{RAM}$) towards the ETH core. If a descriptor is marked as first (FS bit of $TDES1_{RAM}$ is set), then the DMA marks the first transfer from the buffer as the start of frame. If a descriptor is marked as last (LS bit of $TDES1_{RAM}$), then the DMA marks the last transfer from that data buffer as the end-of frame to the MTL.

The Receive DMA transfers data to a buffer until the buffer is full or the end-of frame is received from the MTL. If a descriptor is not marked as last (LS bit of $RDES0_{RAM}$), then the descriptor's corresponding buffer(s) are full and the amount of valid data in a buffer is accurately indicated by its buffer size field minus the data buffer pointer offset when the FS bit of that descriptor is set. The offset is zero when the data buffer pointer is aligned to the data bus width. If a descriptor is marked as last, then the buffer may not be full (as indicated by the buffer size in $RDES1_{RAM}$). To compute the amount of valid data in this final buffer, the driver must read the frame length (FL bits of $RDES0[29:16]_{RAM}$) and subtract the sum of the buffer sizes of the preceding buffers in this frame. The Receive DMA always transfers the start of next frame with a new descriptor.

Note: Even when the start address of a receive buffer is not aligned to a word boundary, the system should allocate a receive buffer aligned to a word boundary. For example, if the system allocates a 1024-byte (1 KB) receive buffer starting from address 1000_H, the software can program the buffer start address in the Receive descriptor to have a 1002_H offset. The Receive DMA writes the frame to this buffer with dummy data in the first two locations (1000_H and 1001_H). The actual frame is written from location 1002_H. Thus, the actual useful space in this buffer is 1022

bytes, even though the buffer size is programmed as 1024 bytes, due to the start address offset.

DMA Arbiter

The arbiter inside the DMA module performs the arbitration between the Transmit and Receive channel accesses to the Bus Master interface. Two types of arbitrations are possible: round-robin, and fixed-priority.

When round-robin arbitration is selected (**ETH0_BUS_MODE**.DA bit is reset), the arbiter allocates the data bus in the ratio set by the Bus Mode.PR Bits , when both Transmit and Receive DMAs are requesting for access simultaneously. When the DA bit is set, the Receive DMA always gets priority over the Transmit DMA for data access.

15.2.3.2 Transmission

The Transmit DMA engine has two operating modes, default and Opertate Second Frame (OSF). Both these modes are described below.

TxDMA Operation: Default (Non-OSF) Mode

The Transmit DMA engine in default mode proceeds as follows:

1. The CPU sets up the transmit descriptor ($TDES0_{RAM}$ -$TDES3_{RAM}$) and sets the Own bit ($TDES0[31]_{RAM}$) after setting up the corresponding data buffer(s) with Ethernet Frame data.
2. Once the **ETH0_OPERATION_MODE**.ST bit is set, the DMA enters the Run state.
3. While in the Run state, the DMA polls the Transmit Descriptor list for frames requiring transmission. After polling starts, it continues in either sequential descriptor ring order or chained order. If the DMA detects a descriptor flagged as owned by the CPU, or if an error condition occurs, transmission is suspended and both the Transmit Buffer Unavailable (**ETH0_STATUS**.TU) and Normal Interrupt Summary (STATUS.NIS Register) bits are set. The Transmit Engine proceeds to Step 8.
4. If the acquired descriptor is flagged as owned by DMA ($TDES0[31]_{RAM} = 1_B$), the DMA decodes the Transmit Data Buffer address from the acquired descriptor.
5. The DMA fetches the Transmit data from the XMC4500 memory and transfers the data to the MTL for transmission.
6. If an Ethernet frame is stored over data buffers in multiple descriptors, the DMA closes the intermediate descriptor and fetches the next descriptor. Steps Step 2, Step 3 and Step 4 are repeated until the end-of-Ethernet-frame data is transferred to the MTL.
7. When frame transmission is complete, if IEEE 1588 time stamping was enabled for the frame (as indicated in the transmit status) the time-stamp value obtained from MTL is written to the transmit descriptor ($TDES2_{RAM}$ and $TDES3_{RAM}$) that contains the end-of-frame buffer. The status information is then written to this transmit descriptor ($TDES0_{RAM}$). Because the Own bit is cleared during this step, the CPU now owns this

descriptor. If time stamping was not enabled for this frame, the DMA does not alter the contents of $TDES2_{RAM}$ and $TDES3_{RAM}$.

8. Transmit Interrupt (**ETH0_STATUS**.TI) is set after completing transmission of a frame that has Interrupt on Completion ($TDES1[31]_{RAM}$) set in its Last Descriptor. The DMA engine then returns to Step 2.
9. In the Suspend state, the DMA tries to re-acquire the descriptor (and thereby return to Step 2) when it receives a Transmit Poll demand and the Underflow Interrupt Status bit is cleared.

The TxDMA transmission flow in default mode is shown in **Figure 15-3**.

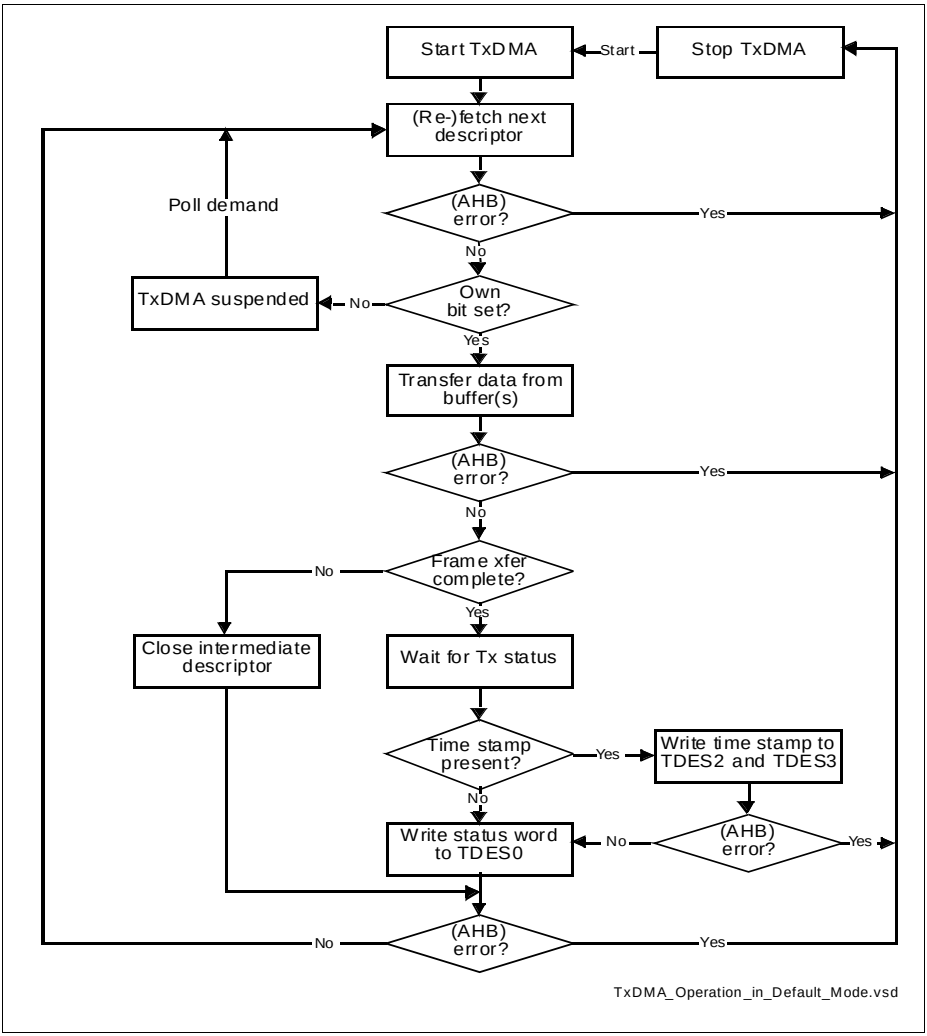


Figure 15-3 TxDMA Operation in Default Mode

TxDMA Operation: OSF Mode

While in the Run state, the transmit process can simultaneously acquire two frames without closing the Status descriptor of the first if **ETH0_OPERATION_MODE**.OSF is set. As the transmit process finishes transferring the first frame, it immediately polls the

Transmit Descriptor list for the second frame. If the second frame is valid, the transmit process transfers this frame before writing the first frame's status information.

In OSF mode, the Run state Transmit DMA operates in the following sequence:

1. The DMA operates as described in Step 1–Step 5 of the TxDMA (default mode).
2. Without closing the previous frame's last descriptor, the DMA fetches the next descriptor.
3. If the DMA owns the acquired descriptor, the DMA decodes the transmit buffer address in this descriptor. If the DMA does not own the descriptor, the DMA goes into Suspend mode and skips to Step 6.
4. The DMA fetches the Transmit frame from the XMC4500 memory and transfers the frame to the MTL until the End-of-Frame data is transferred, closing the intermediate descriptors if this frame is split across multiple descriptors.
5. The DMA waits for the previous frame's frame transmission status and time stamp. Once the status is available, the DMA writes the time stamp to $TDES2_{RAM}$ and $TDES3_{RAM}$, if such time stamp was captured (as indicated by a status bit). The DMA then writes the status, with a cleared Own bit, to the corresponding $TDES0_{RAM}$, thus closing the descriptor. If time stamping was not enabled for the previous frame, the DMA does not alter the contents of $TDES2_{RAM}$ and $TDES3_{RAM}$.
6. If enabled, the Transmit interrupt is set, the DMA fetches the next descriptor, then proceeds to Step 2 (when Status is normal). If the previous transmission status shows an underflow error, the DMA goes into Suspend mode (Step 6).
7. In Suspend mode, if a pending status and time stamp are received from the MTL, the DMA writes the time stamp (if enabled for the current frame) to $TDES2_{RAM}$ and $TDES3_{RAM}$, then writes the status to the corresponding $TDES0_{RAM}$. It then sets relevant interrupts and returns to Suspend mode.
8. The DMA can exit Suspend mode and enter the Run state (go to Step 1 or Step 2 depending on pending status) only after receiving a Transmit Poll demand (**ETH0_TRANSMIT_POLL_DEMAND** Register).

Note: As the DMA fetches the next descriptor in advance before closing the current descriptor, the descriptor chain should have more than 2 different descriptors for correct and proper operation.

The basic flow is charted in **Figure 15-4**.

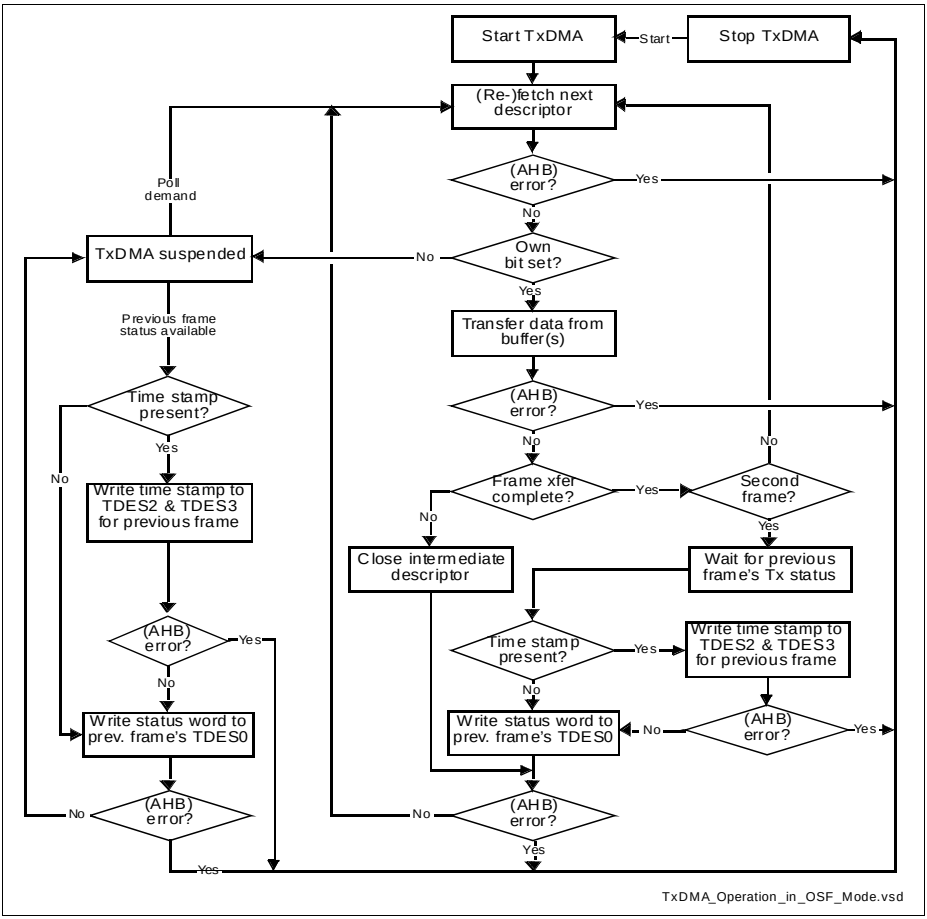


Figure 15-4 TxDMA Operation in OSF Mode

Transmit Frame Processing

The Transmit DMA expects that the data buffers contain complete Ethernet frames, excluding preamble, pad bytes, and FCS fields. The DA, SA, and Type/Len fields must contain valid data. If the Transmit Descriptor indicates that the MAC core must disable CRC or PAD insertion, the buffer must have complete Ethernet frames (excluding preamble), including the CRC bytes.

Frames can be data-chained and can span several buffers. Frames must be delimited by the First Descriptor ($TDES1[29]_{RAM}$) and the Last Descriptor ($TDES1[30]_{RAM}$), respectively.

As transmission starts, the First Descriptor must have ($TDES1[29]_{RAM}$) set. When this occurs, frame data transfers from the XMC4500 RAM buffer to the MTL Transmit FIFO. Concurrently, if the current frame has the Last Descriptor ($TDES1[30]_{RAM}$) clear, the Transmit Process attempts to acquire the Next Descriptor. The Transmit Process expects this descriptor to have $TDES1[29]_{RAM}$ clear. If $TDES1[30]_{RAM}$ is clear, it indicates an intermediary buffer. If $TDES1[30]_{RAM}$ is set, it indicates the last buffer of the frame.

After the last buffer of the frame has been transmitted, the DMA writes back the final status information to the Transmit Descriptor 0 ($TDES0_{RAM}$) word of the descriptor that has the last segment set in Transmit Descriptor 1 ($TDES1[30]_{RAM}$). At this time, if Interrupt on Completion ($TDES1[31]_{RAM}$) was set, Transmit Interrupt (**ETH0_STATUS**.TI) is set, the Next Descriptor is fetched, and the process repeats.

Actual frame transmission begins after the MTL Transmit FIFO has reached either a programmable transmit threshold (**ETH0_OPERATION_MODE**.TTC), or a full frame is contained in the FIFO. There is also an option for Store and Forward Mode (Operation Mode.TSF). Descriptors are released (Own bit $TDES0[31]_{RAM}$ clears) when the DMA finishes transferring the frame.

Transmit Polling Suspended

Transmit polling can be suspended by either of the following conditions:

- The DMA detects a descriptor owned by the CPU ($TDES0[31]_{RAM}$ =0). To resume, the driver must give descriptor ownership to the DMA and then issue a Poll Demand command.
- A frame transmission is aborted when a transmit error due to underflow is detected. The appropriate Transmit Descriptor 0 ($TDES0_{RAM}$) bit is set.

If the second condition occurs, both Abnormal Interrupt Summary (STATUS.AIS) and Transmit Underflow bits (STATUS.TU) are set, and the information is written to Transmit Descriptor 0, causing the suspension. If the DMA goes into SUSPEND state due to the first condition, then both Normal Interrupt Summary (STATUS.NIS) and Transmit Buffer Unavailable (STATUS.TU) are set.

In both cases, the position in the Transmit List is retained. The retained position is that of the descriptor following the Last Descriptor closed by the DMA.

The driver must explicitly issue a Transmit Poll Demand command after rectifying the suspension cause.

15.2.3.3 Reception

The Receive DMA engine's reception sequence is depicted in **Figure 15-5** and proceeds as follows:

1. The CPU sets up Receive descriptors ($RDES0_{RAM}$ -$RDES3_{RAM}$) and sets the Own bit ($RDES0[31_{RAM}$).
2. Once the **ETH0_OPERATION_MODE**.SR bit is set, the DMA enters the Run state. While in the Run state, the DMA polls the Receive Descriptor list, attempting to acquire free descriptors. If the fetched descriptor is not free (is owned by the CPU), the DMA enters the Suspend state and jumps to Step 8.
3. The DMA decodes the receive data buffer address from the acquired descriptors.
4. Incoming frames are processed and placed in the acquired descriptor's data buffers.
5. When the buffer is full or the frame transfer is complete, the Receive engine fetches the next descriptor.
6. If the current frame transfer is complete, the DMA proceeds to Step 6. If the DMA does not own the next fetched descriptor and the frame transfer is not complete (EOF is not yet transferred), the DMA sets the Descriptor Error bit in the RDES0 (unless flushing is disabled). The DMA closes the current descriptor (clears the Own bit) and marks it as intermediate by clearing the Last Segment (LS) bit in the RDES0 value (marks it as Last Descriptor if flushing is not disabled), then proceeds to Step 7. If the DMA does own the next descriptor but the current frame transfer is not complete, the DMA closes the current descriptor as intermediate and reverts to Step 3.
7. If IEEE 1588 time stamping is enabled, the DMA writes the time stamp (if available) to the current descriptor's $RDES2_{RAM}$ and $RDES3_{RAM}$. It then takes the receive frame's status from the MTL and writes the status word to the current descriptor's $RDES0_{RAM}$, with the Own bit cleared and the Last Segment bit set.
8. The Receive engine checks the latest descriptor's Own bit. If the CPU owns the descriptor (Own bit is 1'b0) the Receive Buffer Unavailable bit (**ETH0_STATUS**.RU) is set and the DMA Receive engine enters the Suspended state (Step 8). If the DMA owns the descriptor, the engine returns to Step 3 and awaits the next frame.
9. Before the Receive engine enters the Suspend state, partial frames are flushed from the Receive FIFO (You can control flushing using Operation Mode.DFF).
10. The Receive DMA exits the Suspend state when a Receive Poll demand is given or the start of next frame is available from the MTL's Receive FIFO. The engine proceeds to Step 1 and refetches the next descriptor.

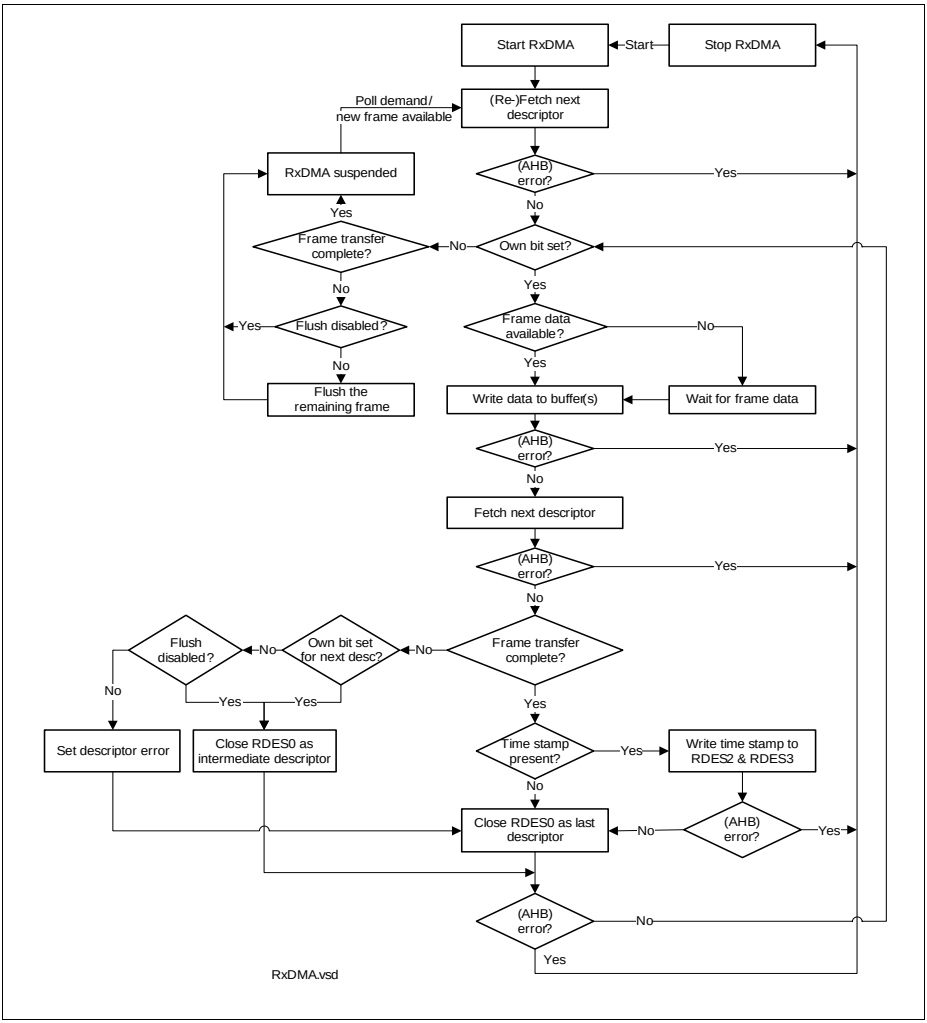


Figure 15-5 Receive DMA Operation

The DMA does not acknowledge accepting the status from the MTL until it has completed the time stamp write-back and is ready to perform status write-back to the descriptor.

If software has enabled time stamping through CSR, when a valid time stamp value is not available for the frame (for example, because the receive FIFO was full before the time stamp could be written to it), the DMA writes all-ones to $RDES2_{RAM}$ and $RDES3_{RAM}$.

Otherwise (that is, if time stamping is not enabled), the $RDES2_{RAM}$ and $RDES3_{RAM}$ remain unchanged.

Receive Descriptor Acquisition

The Receive Engine always attempts to acquire an extra descriptor in anticipation of an incoming frame. Descriptor acquisition is attempted if any of the following conditions is satisfied:

- The receive Start/Stop bit (**ETH0_OPERATION_MODE**.SR) has been set immediately after being placed in the Run state.
- The data buffer of current descriptor is full before the frame ends for the current transfer.
- The controller has completed frame reception, but the current Receive Descriptor is not yet closed.
- The receive process has been suspended because of a CPU-owned buffer ($RDES0[31]_{RAM}$ = 0) and a new frame is received.
- A Receive poll demand has been issued.

Receive Frame Processing

The ETH transfers the received frames to the XMC4500 memory only when the frame passes the address filter and frame size is greater than or equal to configurable threshold bytes set for the Receive FIFO of the MTL, or when the complete frame is written to the FIFO in Store-and-Forward mode.

If the frame fails the address filtering, it is dropped in the ETH block itself (unless Receive All **ETH0_MAC_FRAME_FILTER**.RA is set). Frames that are shorter than 64 bytes, because of collision or premature termination, can be purged from the MTL Receive FIFO.

After 64 (configurable threshold) bytes have been received, the MTL block requests the DMA block to begin transferring the frame data to the Receive Buffer pointed to by the current descriptor. The DMA sets the First Descriptor bit ($RDES0[9]_{RAM}$) after the DMA CPU Interface becomes ready to receive a data transfer (if DMA is not fetching transmit data from the XMC4500 RAM), to delimit the frame. The descriptors are released when the Own ($RDES[31]_{RAM}$) bit is reset to 1'b0, either as the Data buffer fills up or as the last segment of the frame is transferred to the Receive buffer. If the frame is contained in a single descriptor, both Last Descriptor bit($RDES[8]_{RAM}$) and First Descriptor bit ($RDES[9]_{RAM}$) are set.

The DMA fetches the next descriptor, sets the Last Descriptor ($RDES[8]_{RAM}$) bit, and releases the $RDES0_{RAM}$ status bits in the previous frame descriptor. Then the DMA sets Receive Interrupt flag (**ETH0_STATUS**.RI). The same process repeats unless the DMA encounters a descriptor flagged as being owned by the CPU. If this occurs, the Receive Process sets Receive Buffer Unavailable (STATUS.RU) and then enters the Suspend state. The position in the receive list is retained.

Receive Process Suspended

If a new Receive frame arrives while the Receive Process is in Suspend state, the DMA refetches the current descriptor in the XMC4500 memory. If the descriptor is now owned by the DMA, the Receive Process re-enters the Run state and starts frame reception. If the descriptor is still owned by the CPU, by default, the DMA discards the current frame at the top of the MTL Rx FIFO and increments the missed frame counter. If more than one frame is stored in the MTL Rx FIFO, the process repeats.

The discarding or flushing of the frame at the top of the MTL Rx FIFO can be avoided by setting **ETH0_OPERATION_MODE**.DFF bit. In such conditions, the receive process sets the Receive Buffer Unavailable status and returns to the Suspend state.

15.2.3.4 Interrupts

Interrupts can be generated as a result of various events. The **ETH0_STATUS** Register contains all the bits that might cause an interrupt. The **ETH0_INTERRUPT_ENABLE** Register contains an enable bit for each of the events that can cause an interrupt.

There are two groups of interrupts, Normal and Abnormal, as described in the STATUS Register. Interrupts are cleared by writing 1_B to the corresponding bit position. When all the enabled interrupts within a group are cleared, the corresponding summary bit is cleared. When both the summary bits are cleared, the interrupt signal to the NVIC is deasserted. If the ETH core is the cause for assertion of the interrupt, then any of the ELI, EMI, or EPI bits of DMA STATUS Register will be set high.

Note: The interrupt signal to the NVIC will be asserted due to any event in the DMA STATUS register only if the corresponding interrupt enable bit is set in DMA Interrupt Enable Register.

Interrupts are not queued and if the interrupt event occurs before the driver has responded to it, no additional interrupts are generated. For example, Receive Interrupt (STATUS.RI) indicates that one or more frames was transferred to the XMC4500 RAM buffer. The driver must scan all descriptors, from the last recorded position to the first one owned by the DMA.

An interrupt is generated only once for simultaneous, multiple events. The driver must scan the STATUS Register for the cause of the interrupt. The interrupt is not generated again unless a new interrupting event occurs, after the driver has cleared the appropriate bit in the STATUS Register . For example, the controller generates a Receive interrupt (DMA STATUS.RI) and the driver begins reading the STATUS Register. Next, Receive Buffer Unavailable (STATUS Register) occurs. The driver clears the Receive interrupt. Even then, the DMA interupt signal to the NVIC is not deasserted, due to the active or pending Receive Buffer Unavailable interrupt.

An interrupt timer (**ETH0_RECEIVE_INTERRUPT_WATCHDOG_TIMER**) is given for flexible control of Receive Interrupt (STATUS.RI). When this Interrupt timer is programmed with a non-zero value, it will get activated as soon as the RxDMA

completes a transfer of a received frame to system memory without asserting the Receive Interrupt because it is not enabled in the corresponding Receive Descriptor ($RDES1[31]_{RAM}$ in Table 7-3). When this timer runs out as per the programmed value, RI bit is set and the interrupt is asserted if the corresponding **ETH0_INTERRUPT_ENABLE**.RI bit is enabled. This timer gets disabled before it runs out, when a frame is transferred to memory and the **ETH0_STATUS**.RI is set because it is enabled for that descriptor.

15.2.4 DMA Descriptors

This chapter describes the descriptor format used by the ETH DMA.The ETH DMA descriptors are held in RAM. To avoid confusion with the ETH registers the DMA descriptors use the subscript $]_{RAM}$ for example RDES0[0]$_{RAM}$.

15.2.4.1 Descriptor Formats

The DMA in the Ethernet subsystem transfers data based on a linked list of descriptors, as explained in **DMA Controller**. The default descriptor formats (common for both Receive and Transmit Descriptors) are shown in **Figure 15-6**, and field descriptions are provided in **Chapter** to **Chapter** .

Note:

16. *Changes to the default descriptor format when IEEE1588 time stamping is enabled are described in **Chapter "Descriptor Format With IEEE 1588 Time Stamping Enabled" on Page 15-53**".*

Each descriptor contains two buffers, two byte-count buffers, and two address pointers, which enable the adapter port to be compatible with various types of memory management schemes.

The descriptor addresses must be aligned to 32 bit word boundaries .

Figure 15-6 show the descriptor format .

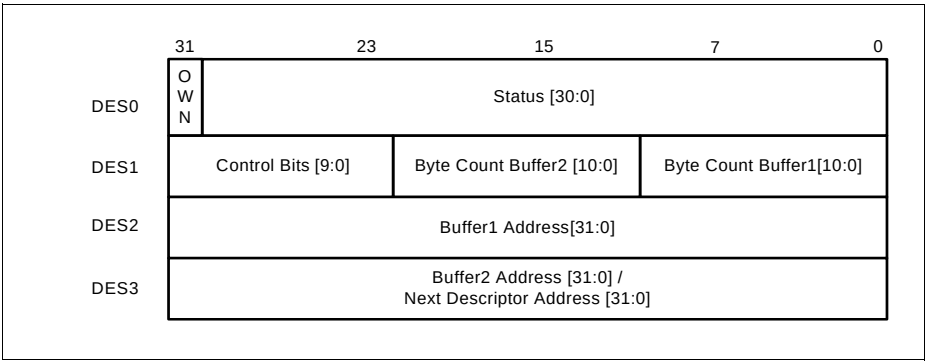


Figure 15-6 Rx/Tx Descriptors

Receive Descriptor

The ETH Subsystem requires at least two descriptors when receiving a frame. The Receive state machine of the DMA (in the ETH Subsystem) always attempts to acquire an extra descriptor in anticipation of an incoming frame. (The size of the incoming frame

is unknown). Before the RxDMA closes a descriptor, it will attempt to acquire the next descriptor even if no frames are received.

In a single descriptor (receive) system, the subsystem will generate a descriptor error if the receive buffer is unable to accommodate the incoming frame and the next descriptor is not owned by the DMA. Thus, the CPU is forced to increase either its descriptor pool or the buffer size. Otherwise, the subsystem starts dropping all incoming frames.

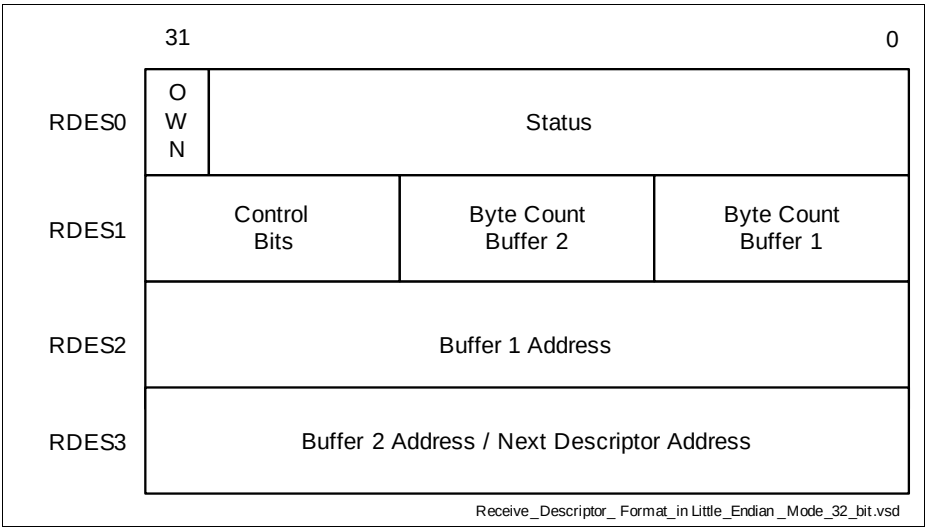


Figure 15-7 Receive Descriptor Format

Receive Descriptor 0 ($RDES0_{RAM}$)

$RDES0_{RAM}$ contains the received frame status, the frame length, and the descriptor ownership information.The format of the descriptor is given in tables **Table 15-4** through **Table 15-12**.

Table 15-4 Receive Descriptor 0

Bit	Description
31	**OWN: Own Bit** When set, this bit indicates that the descriptor is owned by the DMA of the ETH Subsystem. When this bit is reset, this bit indicates that the descriptor is owned by the CPU. The DMA clears this bit either when it completes the frame reception or when the buffers that are associated with this descriptor are full.
30	**AFM: Destination Address Filter Fail** When set, this bit indicates a frame that failed in the DA Filter in the ETH Core.

Table 15-4 Receive Descriptor 0 (cont'd)

Bit	Description
29:16	**FL: Frame Length** These bits indicate the byte length of the received frame that was transferred to XMC4500 memory (including CRC). This field is valid when Last Descriptor ($RDES0[8]_{RAM}$ is set and either the Descriptor Error ($RDES0[14]_{RAM}$) or Overflow Error bits are are reset. The frame length also includes the two bytes appended to the Ethernet frame when IP checksum calculation (Type 1) is enabled and the received frame is not a MAC control frame. This field is valid when Last Descriptor ($RDES0[8]_{RAM}$) is set. When the Last Descriptor and Error Summary bits are not set, this field indicates the accumulated number of bytes that have been transferred for the current frame.
15	**ES: Error Summary** Indicates the logical OR of the following bits: • $RDES0[0]_{RAM}$: Payload Checksum Error • $RDES0[1]_{RAM}$: CRC Error • $RDES0[3]_{RAM}$: Receive Error • $RDES0[4]_{RAM}$: Watchdog Timeout • $RDES0[6]_{RAM}$: Late Collision • $RDES0[7]_{RAM}$: IPC Checksum (Type 2) / Giant Frame • $RDES0[11]_{RAM}$: Overflow Error • $RDES0[14]_{RAM}$: Descriptor Error This field is valid only when the Last Descriptor ($RDES0[8]_{RAM}$) is set.
14	**DE: Descriptor Error** When set, this bit indicates a frame truncation caused by a frame that does not fit within the current descriptor buffers, and that the DMA does not own the Next Descriptor. The frame is truncated. This field is valid only when the Last Descriptor ($RDES0[8]_{RAM}$) is set.
13	**SAF: Source Address Filter Fail** When set, this bit indicates that the SA field of frame failed the SA Filter in the ETH Core.
12	**LE: Length Error** When set, this bit indicates that the actual length of the frame received and that the Length/ Type field does not match. This bit is valid only when the Frame Type ($RDES0[5]_{RAM}$) bit is reset. Length error status is not valid when CRC error is present.
11	**OE: Overflow Error** When set, this bit indicates that the received frame was damaged due to buffer overflow in MTL.

Table 15-4 Receive Descriptor 0 (cont'd)

Bit	Description
10	**VLAN: VLAN Tag** When set, this bit indicates that the frame pointed to by this descriptor is a VLAN frame tagged by the ETH Core.
9	**FS: First Descriptor** When set, this bit indicates that this descriptor contains the first buffer of the frame. If the size of the first buffer is 0, the second buffer contains the beginning of the frame. If the size of the second buffer is also 0, the next Descriptor contains the beginning of the frame.
8	**LS: Last Descriptor** When set, this bit indicates that the buffers pointed to by this descriptor are the last buffers of the frame
7	**IPC Checksum Error/Giant Frame** This bit indicates an error in the IPv4 or IPv6 header. This error can be due to inconsistent Ethernet Type field and IP header Version field values, a header checksum mismatch in IPv4, or an Ethernet frame lacking the expected number of IP header bytes. Refer to **Table 15-5** for more details.
6	**LC: Late Collision** When set, this bit indicates that a late collision has occurred while receiving the frame in Half-Duplex mode.
5	**FT: Frame Type** When set, this bit indicates that the Receive Frame is an Ethernet-type frame (the LT field is greater than or equal to 0600_H). When this bit is reset, it indicates that the received frame is an IEEE802.3 frame. This bit is not valid for Runt frames less than 14 bytes. Refer to **Table 15-5** for more details.
4	**RWT: Receive Watchdog Timeout** When set, this bit indicates that the Receive Watchdog Timer has expired while receiving the current frame and the current frame is truncated after the Watchdog Timeout.
3	**RE: Receive Error** When set, this bit indicates that the MII Receive Error signal is asserted while Carrier Sense signal is asserted during frame reception. This error also includes carrier extension error in MII and Half-duplex mode. Error can be of less/no extension, or error (rxd ≠ 0f) during extension.
2	**DE: Dribble Bit Error** When set, this bit indicates that the received frame has a non-integer multiple of bytes (odd nibbles). This bit is valid only in MII Mode.

Table 15-4 Receive Descriptor 0 (cont'd)

Bit	Description
1	**CE: CRC Error** When set, this bit indicates that a Cyclic Redundancy Check (CRC) Error occurred on the received frame. This field is valid only when the Last Descriptor ($RDES0[8]_{RAM}$) is set.
0	**Payload Checksum Error** When set, indicates the TCP, UDP, or ICMP checksum the core calculated does not match the received encapsulated TCP, UDP, or ICMP segment's Checksum field. This bit is also set when the received number of payload bytes does not match the value indicated in the Length field of the encapsulated IPv4 or IPv6 datagram in the received Ethernet frame. Refer to **Table 15-5** for more details.

The permutations of bits 5, 7, and 0 reflect the conditions discussed in **Table 15-5**.

Table 15-5 Receive Descriptor 0 When COE (Type 2) Is Enabled

Bit 5: Frame Type	Bit 7: IPC Checksum Error	Bit 0: Payload Checksum Error	Frame Status
0	0	0	IEEE 802.3 Type frame (Length field value is less than 0600_H)
1	0	0	IPv4/IPv6 Type frame, no checksum error detected
1	0	1	IPv4/IPv6 Type frame with a payload checksum error (as described for PCE) detected
1	1	0	IPv4/IPv6 Type frame with an IP header checksum error (as described for IPC CE) detected
1	1	1	IPv4/IPv6 Type frame with both IP header and payload checksum errors detected
0	0	1	IPv4/IPv6 Type frame with no IP header checksum error and the payload check bypassed, due to an unsupported payload
0	1	1	A Type frame that is neither IPv4 or IPv6 (the Checksum Offload engine bypasses checksum completely.)
0	1	0	Reserved

Receive Descriptor 1 ($RDES1_{RAM}$)

$RDES1_{RAM}$ contains the buffer sizes and other bits that control the descriptor chain/ring.

*Note: See **Buffer Size Calculations** for further detail on calculating buffer sizes.*

Table 15-6 Receive Descriptor 1

Bit	Description
31	**Disable Interrupt on Completion** When set, this bit will prevent the setting of the **ETH0_STATUS**.RI bit of the Status Register for the received frame that ends in the buffer pointed to by this descriptor. This, in turn, will disable the assertion of the interrupt to the CPU due to RI for that frame.
30:26	**Reserved**
25	**RER: Receive End of Ring** When set, this bit indicates that the descriptor list reached its final descriptor. The DMA returns to the base address of the list, creating a Descriptor Ring.
24	**RCH: Second Address Chained** When set, this bit indicates that the second address in the descriptor is the Next Descriptor address rather than the second buffer address. When $RDES1[24]_{RAM}$ is set, $RBS2_{RAM}$ ($RDES1[21\text{-}11]_{RAM}$) is a “don’t care” value. $RDES1[25]_{RAM}$ takes precedence over $RDES1[24]_{RAM}$.
23:22	**Reserved**
21:11	**RBS2: Receive Buffer 2 Size** These bits indicate the second data buffer size in bytes. The buffer size must be a multiple of 4 , even if the value of $RDES3_{RAM}$ (buffer2 address pointer) is not aligned to bus width. In the case where the buffer size is not a multiple of 4, the resulting behavior is undefined. This field is not valid if $RDES1[24]_{RAM}$ is set.
10:0	**RBS1: Receive Buffer 1 Size** Indicates the first data buffer size in bytes. The buffer size must be a multiple of 4, even if the value of $RDES2_{RAM}$ (buffer1 address pointer) is not aligned. In the case where the buffer size is not a multiple of 4/8/16, the resulting behavior is undefined. If this field is 0, the DMA ignores this buffer and uses Buffer 2 or next descriptor depending on the value of RCH (Bit 24).

Receive Descriptor 2 ($RDES2_{RAM}$)

$RDES2_{RAM}$ contains the address pointer to the first data buffer in the descriptor.

*Note: See **XMC4500 Data Buffer Alignment** for further detail on buffer address alignment.*

Table 15-7 Receive Descriptor 2 (Default Operation)

Bit	Description
31:0	**Buffer 1 Address Pointer** These bits indicate the physical address of Buffer 1. There are no limitations on the buffer address alignment except for the following condition: The DMA uses the configured value for its address generation when the RDES2 value is used to store the start of frame. Note that the DMA performs a write operation with the $RDES2[3/2/1:0]_{RAM}$ bits as 0 during the transfer of the start of frame but the frame data is shifted as per the actual Buffer address pointer. The DMA ignores $RDES2[3/2/1:0]_{RAM}$ if the address pointer is to a buffer where the middle or last part of the frame is stored.

Receive Descriptor 3 ($RDES3_{RAM}$)

$RDES3_{RAM}$ contains the address pointer either to the second data buffer in the descriptor or to the next descriptor.

Table 15-8 Receive Descriptor 3

Bit	Description
31:0	**Buffer 2 Address Pointer (Next Descriptor Address)** These bits indicate the physical address of Buffer 2 when a descriptor ring structure is used. If the Second Address Chained ($RDES1[24]_{RAM}$) bit is set, this address contains the pointer to the physical memory where the Next Descriptor is present. If $RDES1[24]_{RAM}$ is set, the buffer (Next Descriptor) address pointer must be bus width-aligned ($RDES3[3, 2, or 1:0]_{RAM}$ = 0, corresponding to a bus width of 128, 64, or 32. LSBs are ignored internally.) However, when $RDES1[24]_{RAM}$ is reset, there are no limitations on the $RDES3_{RAM}$ value, except for the following condition: The DMA uses the configured value for its buffer address generation when the RDES3 value is used to store the start of frame. The DMA ignores $RDES3[3, 2, or 1:0]_{RAM}$ if the address pointer is to a buffer where the middle or last part of the frame is stored.

Transmit Descriptor

The descriptor addresses must be aligned to the 32 bit word boundary . **Figure 15-8** shows the transmit descriptor format.

Each descriptor is provided with two buffers, two byte-count buffers, and two address pointers, which enable the adapter port to be compatible with various types of memory-management schemes.

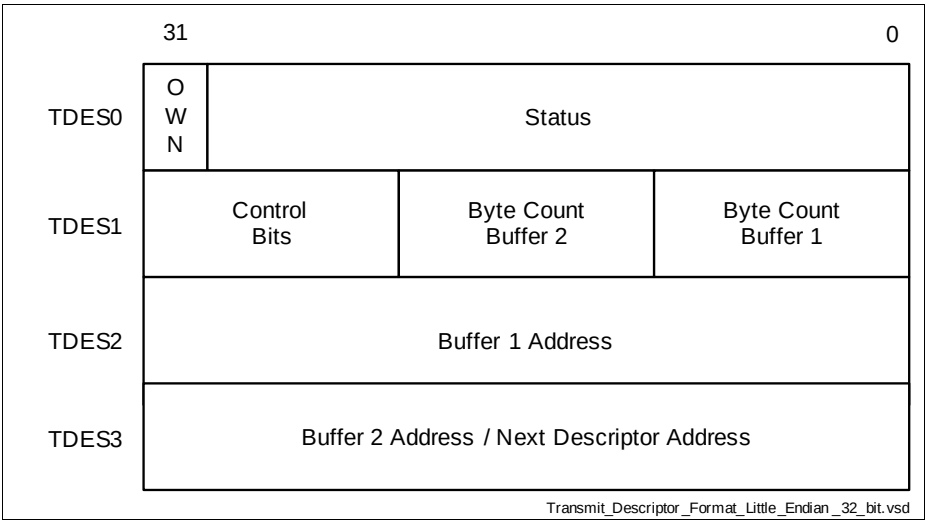


Figure 15-8 Transmit Descriptor Format

Transmit Descriptor 0 ($TDES0_{RAM}$)

$TDES0_{RAM}$ contains the transmitted frame status and the descriptor ownership information.

Table 15-9 Transmit Descriptor 0

Bit	Description
31	**OWN: Own Bit** When set, this bit indicates that the descriptor is owned by the DMA. When this bit is reset, this bit indicates that the descriptor is owned by the CPU. The DMA clears this bit either when it completes the frame transmission or when the buffers allocated in the descriptor are empty. The ownership bit of the First Descriptor of the frame should be set after all subsequent descriptors belonging to the same frame have been set. This avoids a possible race condition between fetching a descriptor and the driver setting an ownership bit.
30:18	**Reserved**

Table 15-9 Transmit Descriptor 0 (cont'd)

Bit	Description
17	**TTSS: Tx Time Stamp Status** This status bit indicates that a time stamp has been captured for the corresponding transmit frame. When this bit is set, $TDES2_{RAM}$ and $TDES3_{RAM}$ have time stamp values that were captured for the transmit frame. This field is valid only when the Last Segment control bit ($TDES1[30]_{RAM}$) in a descriptor is set. This bit is valid only when IEEE1588 time stamping feature is enabled; otherwise, it is reserved.
16	**IHE: IP Header Error** When set, this bit indicates that the Checksum Offload engine detected an IP header error and consequently did not modify the transmitted frame for any checksum insertion.
15	**ES: Error Summary** Indicates the logical OR of the following bits: • $TDES0[14]_{RAM}$: Jabber Timeout • $TDES0[13]_{RAM}$: Frame Flush • $TDES0[11]_{RAM}$: Loss of Carrier • $TDES0[10]_{RAM}$: No Carrier • $TDES0[9]_{RAM}$: Late Collision • $TDES0[8]_{RAM}$: Excessive Collision • $TDES0[2]_{RAM}$: Excessive Deferral • $TDES0[1]_{RAM}$: Underflow Error
14	**JT: Jabber Timeout** When set, this bit indicates the ETH transmitter has experienced a jabber time-out. This bit is only set when the ETH configuration register's JD bit is not set.
13	**FF: Frame Flushed** When set, this bit indicates that the DMA/MTL flushed the frame due to a SW flush command given by the CPU.
12	**PCE: Payload Checksum Error** This bit, when set, indicates that the Checksum Offload engine had a failure and did not insert any checksum into the encapsulated TCP, UDP, or ICMP payload. This failure can be either due to insufficient bytes, as indicated by the IP Header's Payload Length field, or the MTL starting to forward the frame to the MAC transmitter in Store-and-Forward mode without the checksum having been calculated yet. This second error condition only occurs when the Transmit FIFO depth is less than the length of the Ethernet frame being transmitted: to avoid deadlock, the MTL starts forwarding the frame when the FIFO is full, even in Store-and-Forward mode.

Table 15-9 Transmit Descriptor 0 (cont'd)

Bit	Description
11	**LC: Loss of Carrier** When set, this bit indicates that Loss of Carrier occurred during frame transmission . This is valid only for the frames transmitted without collision and when the ETH operates in Half-Duplex Mode.
10	**NC: No Carrier** When set, this bit indicates that the carrier sense signal form the PHY was not asserted during transmission.
9	**LC: Late Collision** When set, this bit indicates that frame transmission was aborted due to a collision occurring after the collision window (64 byte times including Preamble in MII Mode). Not valid if Underflow Error is set.
8	**EC: Excessive Collision** When set, this bit indicates that the transmission was aborted after 16 successive collisions while attempting to transmit the current frame. If the DR (Disable Retry) bit in the ETH Configuration Register is set, this bit is set after the first collision and the transmission of the frame is aborted.
7	**VF: VLAN Frame** When set, this bit indicates that the transmitted frame was a VLAN-type frame.
6:3	**CC: Collision Count** This 4-bit counter value indicates the number of collisions occurring before the frame was transmitted. The count is not valid when the Excessive Collisions bit ($TDES0[8]_{RAM}$) is set.
2	**ED: Excessive Deferral** When set, this bit indicates that the transmission has ended because of excessive deferral of over 24,288 bit times (155,680 bits times in 1000 Mbit/s mode, or in Jumbo Frame enabled mode) if the Deferral Check (DC) bit is set high in the ETH Control Register.
1	**UF: Underflow Error** When set, this bit indicates that the ETH aborted the frame because data arrived late from the XMC4500 memory. Underflow Error indicates that the DMA encountered an empty Transmit Buffer while transmitting the frame. The transmission process enters the suspended state and sets both STATUS.TU and STATUS.TI.
0	**DB: Deferred Bit** When set, this bit indicates that the ETH defers before transmission because of the presence of carrier. This bit is valid only in Half-Duplex mode.

Transmit Descriptor 1 ($TDES1_{RAM}$)

$TDES1_{RAM}$ contains the buffer sizes and other bits which control the descriptor chain/ring and the frame being transferred.

*Note: See **Buffer Size Calculations** for further detail on calculating buffer sizes.*

Table 15-10 Transmit Descriptor 1

Bit	Description
31	**IC: Interrupt on Completion** When set, this bit sets Transmit Interrupt, STATUS.TI bit after the present frame has been transmitted.
30	**LS: Last Segment** When set, this bit indicates that the buffer contains the last segment of the frame.
29	**FS: First Segment** When set, this bit indicates that the buffer contains the first segment of a frame.
28:27	**CIC: Checksum Insertion Control** These bits control the insertion of checksums in Ethernet frames that encapsulate TCP, UDP, or ICMP over IPv4 or IPv6 as described below. • 00: Do nothing. Checksum Engine is bypassed • 01: Insert IPv4 header checksum. Use this value to insert IPv4 header checksum when the frame encapsulates an IPv4 datagram. • 10: Insert TCP/UDP/ICMP checksum. The checksum is calculated over the TCP, UDP, or ICMP segment only and the TCP, UDP, or ICMP pseudo-header checksum is assumed to be present in the corresponding input frame's Checksum field. An IPv4 header checksum is also inserted if the encapsulated datagram conforms to IPv4. • 11: Insert a TCP/UDP/ICMP checksum that is fully calculated in this engine. In other words, the TCP, UDP, or ICMP pseudo-header is included in the checksum calculation, and the input frame's corresponding Checksum field has an all-zero value. An IPv4 Header checksum is also inserted if the encapsulated datagram conforms to IPv4. The Checksum engine detects whether the TCP, UDP, or ICMP segment is encapsulated in IPv4 or IPv6 and processes its data accordingly.
26	**DC: Disable CRC** When set, the ETH does not append the Cyclic Redundancy Check (CRC) to the end of the transmitted frame. This is valid only when the first segment ($TDES1[29]_{RAM}$) bit is set.
25	**TER: Transmit End of Ring** When set, this bit indicates that the descriptor list reached its final descriptor. The returns to the base address of the list, creating a descriptor ring.

Table 15-10 Transmit Descriptor 1 (cont'd)

Bit	Description
24	**TCH: Second Address Chained** When set, this bit indicates that the second address in the descriptor is the Next Descriptor address rather than the second buffer address. When $TDES1[24]_{RAM}$ is set, TBS2 ($TDES1[21–11]_{RAM}$) are "don't care" values. $TDES1[25]_{RAM}$ takes precedence over $TDES1[24]_{RAM}$.
23	**DP: Disable Padding** When set, the ETH does not automatically add padding to a frame shorter than 64 bytes. When this bit is reset, the DMA automatically adds padding and CRC to a frame shorter than 64 bytes and the CRC field is added despite the state of the DC ($TDES1[26]_{RAM}$) bit. This is valid only when the first segment ($TDES1[29]_{RAM}$) is set.
22	**TTSE: Transmit Time Stamp Enable** When set, this bit enables IEEE1588 hardware time stamping for the transmit frame referenced by the descriptor. This field is valid only when the First Segment control bit ($TDES1[29]_{RAM}$) is set.
21:11	**TBS2: Transmit Buffer 2 Size** These bits indicate the Second Data Buffer in bytes. This field is not valid if $TDES1[24]_{RAM}$ is set.
10:0	**TBS1: Transmit Buffer 1 Size** These bits indicate the First Data Buffer byte size. If this field is 0, the DMA ignores this buffer and uses Buffer 2 or next descriptor depending on the value of TCH (Bit 24).

Transmit Descriptor 2 (TDES2)

TDES2 contains the address pointer to the first buffer of the descriptor.

Table 15-11 Transmit Descriptor 2

Bit	Description
31:0	**Buffer 1 Address Pointer** These bits indicate the physical address of Buffer 1. There is no limitation on the buffer address alignment. See **XMC4500 Data Buffer Alignment** for further detail on buffer address alignment.

Transmit Descriptor 3 ($TDES3_{RAM}$)

$TDES3_{RAM}$ contains the address pointer either to the second buffer of the descriptor or the next descriptor.

Table 15-12 Transmit Descriptor 3

Bit	Description
31:0	**Buffer 2 Address Pointer (Next Descriptor Address)** Indicates the physical address of Buffer 2 when a descriptor ring structure is used. If the Second Address Chained (TDES1[24]$_{RAM}$) bit is set, this address contains the pointer to the physical memory where the Next Descriptor is present. The buffer address pointer must be aligned to the bus width only when TDES1[24]$_{RAM}$ is set. (LSBs are ignored internally.)

Descriptor Format With IEEE 1588 Time Stamping Enabled

The default descriptor format (as described in **"Receive Descriptor" on Page 15-55** and **"Transmit Descriptor" on Page 15-47**), and field descriptions remain unchanged when created by software (Own bit is set in DES0$_{RAM}$). However, if the software has enabled IEEE 1588 functionality, the DES2$_{RAM}$ and DES3$_{RAM}$ descriptor fields (see **Figure 15-9**) take on a different meaning when the DMA closes the descriptor (own bit in DES0$_{RAM}$ is cleared).

The DMA updates the DES2$_{RAM}$ and DES3$_{RAM}$ with the time stamp value before clearing the Own bit in DES0$_{RAM}$.

DES2$_{RAM}$ is updated with the lower 32 time stamp bits (the Sub-Second field, called TSL in subsequent sections) and DES3$_{RAM}$ is updated with the upper 32 time stamp bits (the Seconds field, called TSH in subsequent sections).

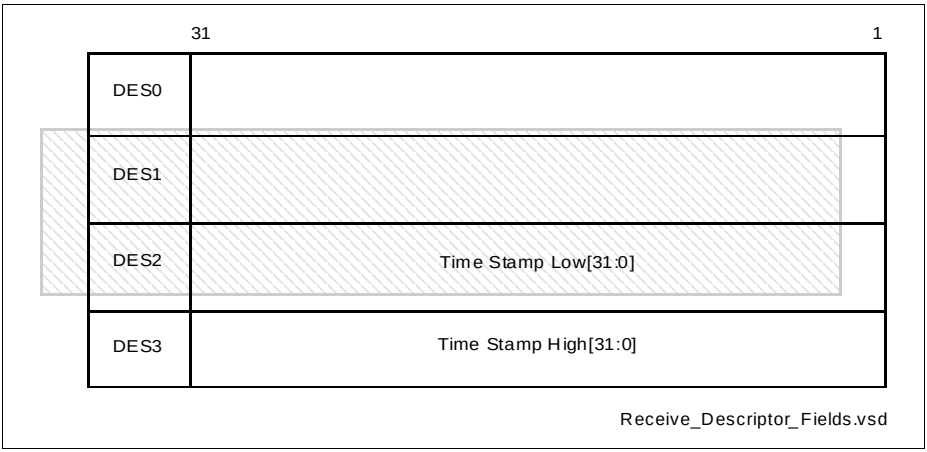


Figure 15-9 Receive Descriptor Fields When DMA Clears the Own Bit

The following sections describe the details specific to receive and transmit descriptors in this mode.

Receive Descriptor

Receive Time Stamp

The tables below describe the fields that have different meaning for $RDES2_{RAM}$ and $RDES3_{RAM}$ when the receive descriptor is closed and time stamping is enabled.

*Note: When software disables the time stamping feature (the **ETH0_TIMESTAMP_CONTROL**.TSENA bit is low), the DMA does not update the descriptor's $RDES2_{RAM}$/$RDES3_{RAM}$ fields before closing the $RDES0_{RAM}$.*

Table 15-13 Receive Descriptor Fields (RDES2)

Bit	Description
31:0	**RTSL: Receive Frame Time Stamp Low** The DMA updates this field with the least significant 32 bits of the time stamp captured for the corresponding receive frame. The DMA updates this field only for the last descriptor of the receive frame indicated by Last Descriptor status bit ($RDES0[8]_{RAM}$). When this field and the RTSH field in $RDES3_{RAM}$ show an all-ones value, the time stamp must be treated as corrupt.

Table 15-14 Receive Descriptor Fields (RDES3)

Bit	Description
31:0	**RTSH: Receive Frame Time Stamp High** The DMA updates this field with the most significant 32 bits of the time stamp captured for the corresponding receive frame. The DMA updates this field only for the last descriptor of the receive frame indicated by Last Descriptor status bit (RDES0[8]RAM). When this field and RDES2RAM's RTSL field show all-ones values, the time stamp must be treated as corrupt.

Transmit Descriptor

In addition to the changes described in **"Descriptor Format With IEEE 1588 Time Stamping Enabled" on Page 15-53**, the Transmit descriptor has additional control and status bits (TTSE and TTSS, respectively) for time stamping, as shown in **Figure 15-10**. Software sets the TTSE bit (when the Own bit is set), instructing the core to generate a time stamp for the corresponding Ethernet frame being transmitted. The DMA sets the TTSS bit if the time stamp has been updated in the $TDES2_{RAM}$ and $TDES3_{RAM}$ fields when the descriptor is closed (Own bit is cleared).

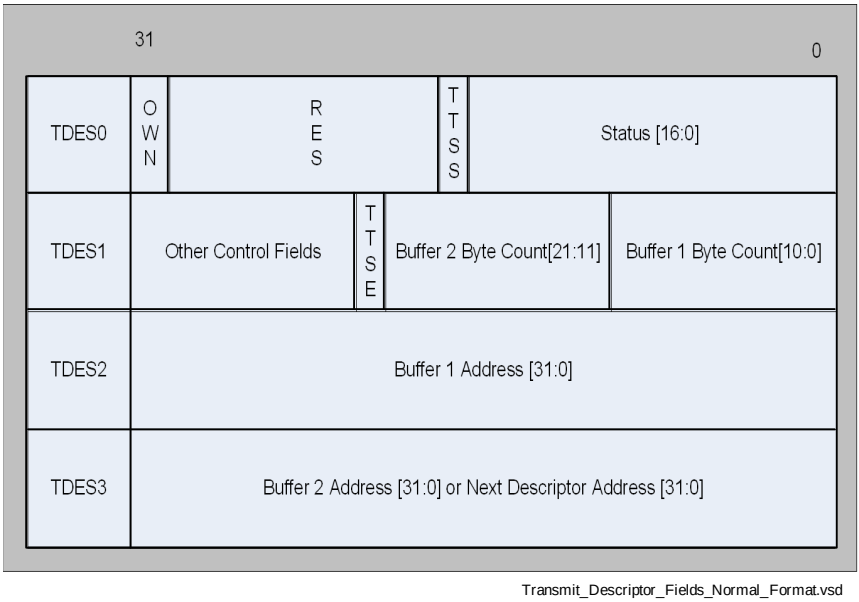


Figure 15-10 Transmit Descriptor Fields

Transmit Time Stamp Control and Status Fields

The value of this field ishall be preserved by the DMA at the time of closing the descriptor.

Updates to **Table 15-8** and **Table 15-9** are described below.

Table 15-15 Transmit Time Stamp Status – Normal Descriptor Format Case (TDES0RAM)

Bit	Description
17	**TTSS: Transmit Time Stamp Status** This field is a status bit indicating that a time stamp was captured for the corresponding transmit frame. When this bit is set, both TDES2RAM and TDES3RAM have a time stamp value that was captured for the transmit frame. This field is valid only when the Last Segment control bit (TDES1[30]RAM in the descriptor) is set.

Table 15-16 Transmit Time Stamp Control – Normal Descriptor Format Case ($TDES1_{RAM}$)

Bit	Description
22	**TTSE: Transmit Time Stamp Enable** When set, this field enables IEEE1588 hardware time stamping for the transmit frame described by the descriptor. This field is valid only when the First Segment control bit ($TDES1[29]_{RAM}$ in the descriptor) is set.

Transmit Time Stamp Field

The transmit descriptor format and field descriptions remain unchanged when they are created by software (when the Own bit is set). However, when the DMA closes the last descriptor and IEEE 1588 functionality is enabled (the Own bit is cleared), the $TDES2_{RAM}$ and TDES3 descriptor fields are updated with the time stamp, if taken, for that frame.

Table 15-17 and **Table 15-18** describe the fields that have a different meaning when the descriptor is closed.

Table 15-17 Transmit Descriptor Fields (TDES2RAM)

Bit	Description
31:0	**TTSL: Transmit Frame Time Stamp Low** This field is updated by DMA with the least significant 32 bits of the time stamp captured for the corresponding transmit frame. This field has the time stamp only if the Last Segment control bit (LS) in the descriptor is set.

Table 15-18 Transmit Descriptor Fields (TDES3)

Bit	Description
31:0	**TTSH: Transmit Frame Time Stamp High** This field is updated by DMA with the most significant 32 bits of the time stamp captured for the corresponding transmit frame. This field has the time stamp only if the Last Segment control bit (LS) in the descriptor is set.

15.2.5 MAC Management Counters

The MMC module maintains a set of registers for gathering statistics on the received and transmitted frames. These include a control register for controlling the behavior of the registers, two 32-bit registers containing interrupts generated (receive and transmit), and two 32-bit registers containing masks for the Interrupt register (receive and transmit). These registers are accessible from the Application through the MAC Control Interface

(MCI). Each register is 32 bits wide. Non-32-bit accesses are allowed as long as the address is word-aligned.

The organization of these registers is shown in **Table 15-27**. The MMCs are accessed using transactions, in the same way the CSR address space is accessed. The following sections in the chapter describe the various counters and list the address for each of the statistics counters. This address will be used for Read/Write accesses to the desired transmit/receive counter.

The Receive MMC counters are updated for frames that are passed by the Address Filter (AFM) block. Statistics of frames that are dropped by the AFM module are not updated unless they are runt frames of less than 6 bytes (DA bytes are not received fully).

The MMC module gathers statistics on encapsulated IPv4, IPv6, TCP, UDP, or ICMP payloads in received Ethernet frames. The address map of the corresponding registers, 0200_H–$02FC_H$, is given in **Table 15-27**.

15.2.6 Power Management Block

This section describes the power management (PMT) mechanisms supported by the ETH. PMT supports the reception of network (remote) wake-up frames and Magic Packet frames. PMT does not perform the clock gate function, but generates interrupts for wake-up frames and Magic Packets received by the ETH. The PMT block sits on the receiver path of the ETH and is enabled with remote wake-up frame enable and Magic Packet enable. These enables are in the PMT_CONTROL_STATUS register and are programmed by the Application.

PMT registers are accessed in the same manner as with ETH-CSR registers. Refer to **Figure 15-27** for mapping information.

When the power down mode is enabled in the PMT, then all received frames are dropped by the core and they are not forwarded to the application. The core comes out of the power down mode only when either a Magic Packet or a Remote Wake-up frame is received and the corresponding detection is enabled.

15.2.6.1 PMT Block Description

PMT Control and Status Register

The PMT CSR program the request wake-up events and monitor the wake-up events.See **ETH0_PMT_CONTROL_STATUS** register for a full description

Remote Wake-Up Frame Filter Register

The Remote Wake up Frame Filter consists of eight words which are programed via the **ETH0_REMOTE_WAKE_UP_FRAME_FILTER** Register. The eight words of the

Remote Wake Up Frame Filter must be written sequentially to the REMOTE_WAKE_UP_FRAME_FILTER Register. The structure of the Remote Wake Up Frame Filter is described below. The Remote Wake Up Frame Filter values must be loaded sequentially starting with wkupfmfilter0 through to wkupfmfilter7. The REMOTE_WAKE_UP_FRAME_FILTER Register is read in the same way.

Note: The internal counter to access the appropriate wkupfmfilter_reg is incremented when lane3 is accessed by the CPU. This should be kept in mind if you are accessing these registers in byte or half-word mode.

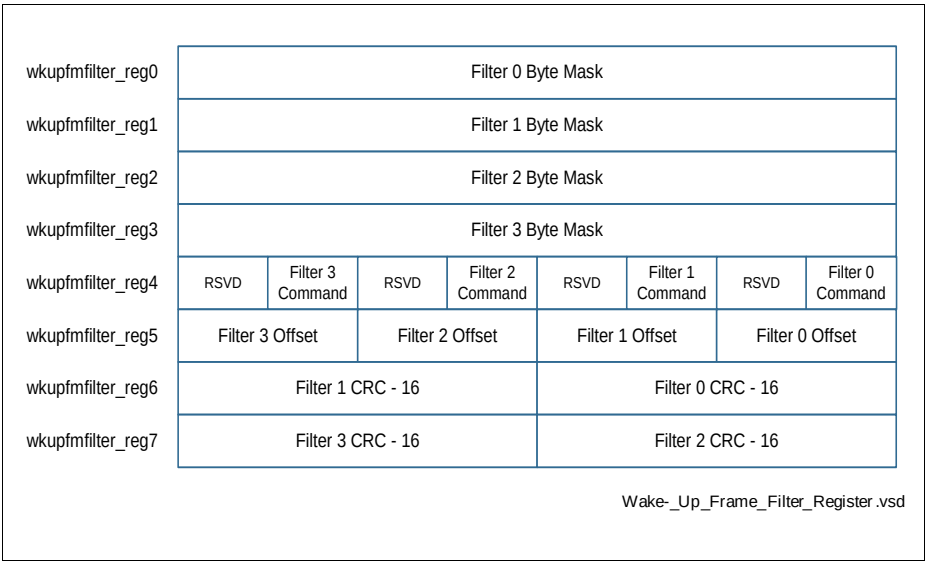


Figure 15-11 Wake-Up Frame Filter Register

Filter i Byte Mask

This register defines which bytes of the frame are examined by filter i (0, 1, 2, and 3) in order to determine whether or not the frame is a wake-up frame. The Most Significant Bit (thirty-first bit) must be zero. Bit j [30:0] is the Byte Mask. If bit j (byte number) of the Byte Mask is set, then Filter i Offset + j of the incoming frame is processed by the CRC block; otherwise Filter i Offset + j is ignored.

Filter i Command

This 4-bit command controls the filter i operation. Bit 3 specifies the address type, defining the pattern's destination address type. When the bit is set, the pattern applies to only multicast frames; when the bit is reset, the pattern applies only to unicast frame.

Bit 2 and Bit 1 are reserved. Bit 0 is the enable for filter i; if Bit 0 is not set, filter i is disabled.

Filter i Offset

This register defines the offset (within the frame) from which the frames are examined by filter i. This 8-bit pattern-offset is the offset for the filter i first byte to examined. The minimum allowed is 12, which refers to the 13th byte of the frame (offset value 0 refers to the first byte of the frame).

Filter i CRC-16

This register contains the CRC_16 value calculated from the pattern, as well as the byte mask programmed to the wake-up filter register block.

15.2.6.2 Remote Wake-Up Frame Detection

When the ETH is in sleep mode and the remote wake-up bit is enabled in PMT Control and Status register , normal operation is resumed after receiving a remote wake-up frame. The Application writes all eight wake-up filter registers, by performing a sequential Write to **ETH0_REMOTE_WAKE_UP_FRAME_FILTER** Register. The Application enables remote wake-up by setting **ETH0_PMT_CONTROL_STATUS**.PWRDWN.

PMT supports four programmable filters that allow support of different receive frame patterns. If the incoming frame passes the address filtering of Filter Command, and if Filter CRC-16 matches the incoming examined pattern, then the wake-up frame is received.

Filter_offset (minimum value 12, which refers to the 13th byte of the frame) determines the offset from which the frame is to be examined. Filter Byte Mask determines which bytes of the frame must be examined. The thirty-first bit of Byte Mask must be set to zero.

The remote wake-up CRC block determines the CRC value that is compared with Filter CRC-16. The wake-up frame is checked only for length error, FCS error, dribble bit error, MII error, collision, and to ensure that it is not a runt frame. Even if the wake-up frame is more than 512 bytes long, if the frame has a valid CRC value, it is considered valid. Wake-up frame detection is updated in the PMT_Control _Status register for every remote Wake-up frame received. A PMT interrupt to the Application triggers a Read to the PMT_CONTROL_STATUS register to determine reception of a wake-up frame.

15.2.6.3 Magic Packet Detection

The Magic Packet frame is based on a method that uses Advanced Micro Device's Magic Packet technology to power up the sleeping device on the network. The ETH receives a specific packet of information, called a Magic Packet, addressed to the node on the network.

Only Magic Packets that are addressed to the device or a broadcast address will be checked to determine whether they meet the wake-up requirements. Magic Packets that pass the address filtering (unicast or broadcast) will be checked to determine whether they meet the remote Wake-on-LAN data format of 6 bytes of all ones followed by a ETH Address appearing 16 times.

The application enables Magic Packet wake-up by writing a 1 to Bit 1 of the **ETH0_PMT_CONTROL_STATUS** register. The PMT block constantly monitors each frame addressed to the node for a specific Magic Packet pattern. Each frame received is checked for a FFFF FFFF $FFFF_H$ pattern following the destination and source address field. The PMT block then checks the frame for 16 repetitions of the ETH address without any breaks or interruptions. In case of a break in the 16 repetitions of the address, the FFFF FFFF $FFFF_H$ pattern is scanned for again in the incoming frame. The 16 repetitions can be anywhere in the frame, but must be preceded by the synchronization stream (FFFF FFFF $FFFF_H$). The device will also accept a multicast frame, as long as the 16 duplications of the ETH address are detected.

If the MAC address of a node is 0011 2233 4455_H, then the ETH scans for the data sequence:

```
Destination Address Source Address …….. FF FF FF FF FF FF
00 11 22 33 44 55 00 11 22 33 44 55 00 11 22 33 44 55
00 11 22 33 44 55 00 11 22 33 44 55 00 11 22 33 44 55
00 11 22 33 44 55 00 11 22 33 44 55 00 11 22 33 44 55
00 11 22 33 44 55 00 11 22 33 44 55 00 11 22 33 44 55
…CRC
```

Magic Packet detection is updated in the PMT_CONTROL_STATUS register for Magic Packet received. A PMT interrupt to the Application triggers a read to the PMT CSR to determine whether a Magic Packet frame has been received.

15.2.6.4 System Considerations During Power-Down

The ETH neither gates nor stops clocks when Power-Down mode is enabled. Power saving by clock gating must be done outside the core by the application. The receive data path must is clocked during Power-Down mode, because it is involved in magic packet/wake-on-LAN frame detection. However, the transmit path and the application path clocks can be gated off during Power-Down mode.

The power management interrupt signal is asserted when a valid wake-up frame is received. This signal is generated in the receive clock domain.

The recommended power-down and wake-up sequence is as follows.

1. Disable the Transmit DMA (if applicable) and wait for any previous frame transmissions to complete. These transmissions can be detected when Transmit Interrupt, **ETH0_STATUS**.TI is received.
2. Disable the MAC transmitter and MAC receiver by clearing the appropriate bits in the **ETH0_MAC_CONFIGURATION** register.
3. Wait until the Receive DMA empties all the frames from the Rx FIFO (a software timer may be required).
4. Enable Power-Down mode by appropriately configuring the PMT registers.
5. Enable the MAC Receiver and enter Power-Down mode.
6. Gate the application and transmit clock inputs to the core (and other relevant clocks in the system) to reduce power and enter Sleep mode.
7. On receiving a valid wake-up frame, the ETH asserts the power management interrupt signal and exits Power-Down mode.
8. On receiving the interrupt, the system must enable the application and transmit clock inputs to the core.
9. Read the **ETH0_PMT_CONTROL_STATUS** register to clear the interrupt, then enable the other modules in the system and resume normal operation.

15.2.7 PHY Interconnect

The ETH supports two external interconnects to external PHY devices. The ETH peripheral may be connected to the external PHY by either a Media Independent Interface (MII) or by a Reduced Media Independent Interface (RMII). Additionally a Station Management Interface (SMI) provides a two wire serial interface between the external PHY and the ETH. The SMI allows the ETH to program the internal PHY configuration registers. The SMI supports connection of up to 32 external PHY devices.

15.2.7.1 PHY Interconnect selection

Selection of the external interconnect configuration between MII or RMII is made by the ETH0_CON.INFSEL register. This must be done while the ETH peripheral is held in reset. Once the PHY interconnect has been configured it may not be changed without placing the ETH back into reset.

15.2.8 Station Management Interface

The Station Management Agent (SMA) module allows the Application to access any PHY registers through a 2-wire Station Management interface (SMI). The PHY interconnect supports accessing up to 32 PHYs.

The application can select one of the 32 PHYs and one of the 32 registers within any PHY and send control data or receive status information. Only one register in one PHY can be addressed at any given time. For more details on the communication from the Application to the PHYs, refer to the Reconciliation Sublayer and Media Independent Interface Specifications section of the IEEE 802.3z specification, 1000BASE Ethernet.

The application sends the control data to the PHY and receives status information from the PHY through the SMA module, as shown in **Figure 15-12**.

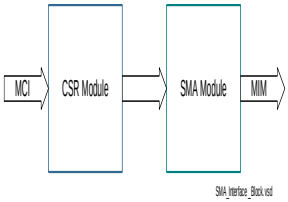


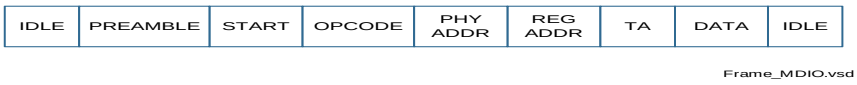
Figure 15-12 SMA Interface Block

15.2.8.1 Station Management Functions

The ETH initiates the Management Write/Read operation. The MDC clock is a divided clock from the ETH MAC clock . The divide factor depends on the clock range setting in the MII Address register. Clock range is set as follows:

Selection	ETH MAC Clock	MDC Clock
0000	60-100 MHz	ETH Clock/42
0001	100-150 MHz	ETH Clock/62
0010	20-35 MHz	ETH Clock/16
0011	35-60 MHz	ETH Clock/26
0100	150-250 MHz	ETH Clock/102
0101	250-300 MHz	ETH Clock/124
0110, 0111	Reserved	

The frame structure on the MDIO line is shown below.



IDLE The mdio line is Tri-state; there is no clock on mdc
PREAMBLE 32 continuous bits of value 1
START Start-of-frame is 01_B
OPCODE 10_B for Read and $b01_B$ for Write
PHY ADDR 5-bit address select for one of 32 PHYs
REG ADDR Register address in the selected PHY

TA Turnaround is $Z0_B$ for Read and 10_B for Write
DATA Any 16-bit value. In a Write operation, the ETH drives mdio; in a Read
operation, PHY drives it.

15.2.8.2 Station Management Write Operation

When the user sets the MII Write and Busy bits (**ETH0_GMII_ADDRESS**.MW and GMII_ADDRESS.MB) the ETH CSR module transfers the PHY address, the register address in PHY, and the write data to the SMA to initiate a Write operation into the PHY registers. At this point, the SMA module starts a Write operation on the MII Management Interface using the Management Frame Format specified in the MII specifications (Section 22.2.4.5 of IEEE Standard). The application should not change the GMII_ADDRESS register contents or the GMII_DATA register while the transaction is ongoing. Write operations to the GMII_ADDRESS register or the **ETH0_GMII_DATA** Register during this period are ignored (the Busy bit is high), and the transaction is completed without any error on the MCI interface.

After the Write operation has completed, the SMA indicates this to the CSR which then resets the Busy bit. The SMA module divides the CSR (Application) clock with the clock divider programmed (CR bits of MII Address Register) to generate the MDC clock for this interface. The ETH drives the MDIO line for the complete duration of the frame. The frame format for the Write operation is as follows:

IDLE	PREAMBLE	START	OPCODE	PHY ADDR	REG ADDR	TA	DATA	IDLE
Z	1111...11	01	01	AAAAA	RRRRR	10	DDD . . . .DDD	Z

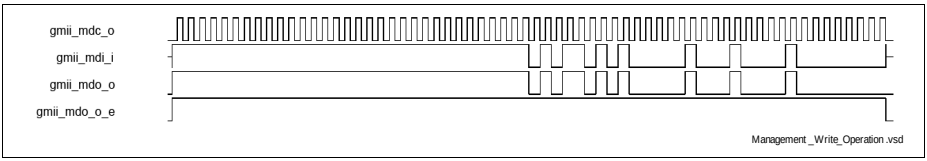


Figure 15-13 Management Write Operation

Figure 15-13 is a reference for the Write operation.

15.2.8.3 Station Management Read Operation

When the user sets the MII Busy bit (**ETH0_GMII_ADDRESS**.MB) with the MII Write bit (GMII_ADDRESS.WB) as 0, the ETH CSR module transfers the PHY address and the register address in PHY to the SMA to initiate a Read operation in the PHY registers. At

this point, the SMA module starts a Read operation on the MII Management Interface using the Management Frame Format specified in the MII specifications (Section 22.2.4.5 of IEEE Standard). The application should not change the GMII_ADDRESS register contents or the GMII_DATA register while the transaction is ongoing. Write operations to the GMII_ADDRESS register or **ETH0_GMII_DATA** Register during this period are ignored (the Busy bit is high) and the transaction completed without any error on the MCI interface.

After the Read operation has completed, the SMA indicates this to the CSR, which then resets the Busy bit and updates the GMII_DATA register with the data read from the PHY. The SMA module divides the CSR (Application) clock with the clock divider programmed (GMII_ADDRESS.CR bits) to generate the MDC clock for this interface. The ETH drives the MDIO line for the complete duration of the frame except during the Data fields when the PHY is driving the MDIO line. The frame format for the Read operation is as follows:

IDLE	PREAMBLE	START	OPCODE	PHY ADDR	REG ADDR	TA	DATA	IDLE
Z	1111...11	01	10	AAAAA	RRRRR	Z0	DDD . . . .DDD	Z

Figure 15-14 is a reference for the read operation.

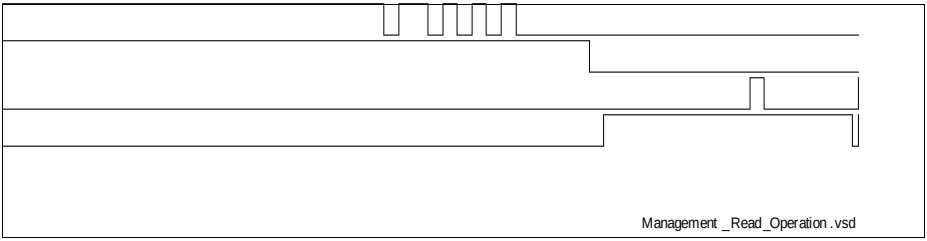


Figure 15-14 Management Read Operation

15.2.9 Media Independent interface

The Media Independent Interface (MII) provides an interconnect to external PHY devices standerdised by IEEE 802.3u. The MII interconnect consists of 16 pins for data and control. The MII interconnect provides two seperate nibble wide busses for transmit and receive each with a dedicated clock running at 2.5Mhz for 10Mbit/s and 25Mhz for 100Mbit/s speeds. Transmit and receive control signals consist of a TX Enable (TX_EN) that allows the ETH to present data to the PHY and a Receive Data Valid (RX_DV) that allows the PHY to present data to the ETH. A Receive Error (RX_ER) signal is also provided that allows the PHY signal the ETH when an error was detected somewhere in

the current received packet. The two remaining control signals are MII collision detect (MII_COL) which is asserted by the PHY when an arbitration collision occurs and MII carrier sense (MII_CRS) which is asserted by the PHY when either Transmit or Recive are not idle.

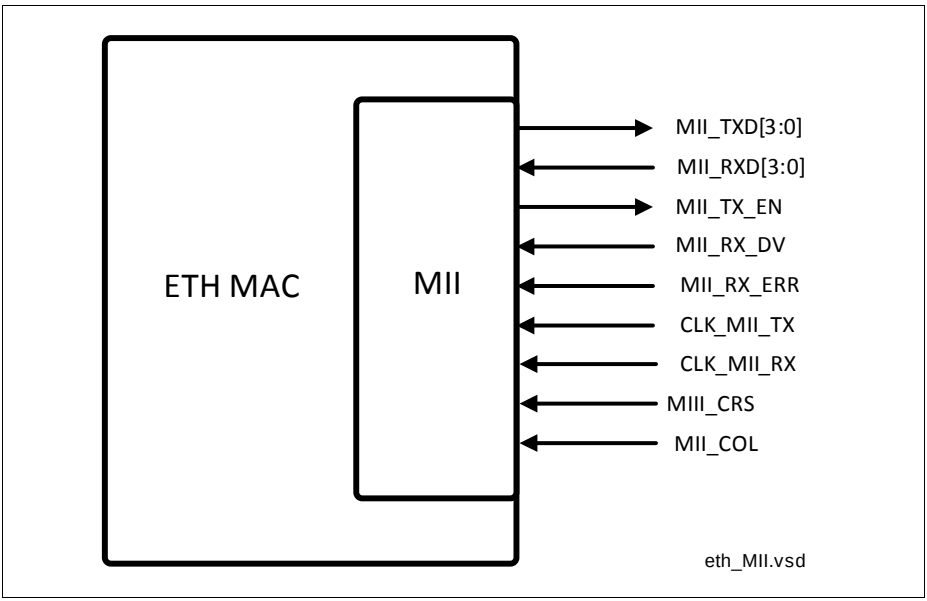


Figure 15-15 Media Independent Interface

15.2.10 Reduced Media Independent Interface

The Reduced Media Independent Interface (RMII) specification reduces the pin count between Ethernet PHYs and Switch ASICs . According to the IEEE 802.3u standard, an MII contains 16 pins for data and control. In devices incorporating multiple MAC or PHY interfaces (such as switches), the number of pins adds significant cost with increase in port count. The RMII specification addresses this problem by reducing the pin count to 7 for each port — a 62.5% decrease in pin count.

- The RMII module is instantiated between the ETH and the PHY. This helps translation of the MAC's MII into the RMII. The RMII block has the following characteristics:
- Supports 10 Mbit/s and 100 Mbit/s operating rates.
- Two clock references are sourced externally, providing independent, 2-bit wide transmit and receive paths.

15.2.10.1 RMII Block Diagram

Figure 15-16 shows the position of the RMII block relative to the ETH and RMII PHY. The RMII block is placed in front of the ETH to translate the MII signals to RMII signals.

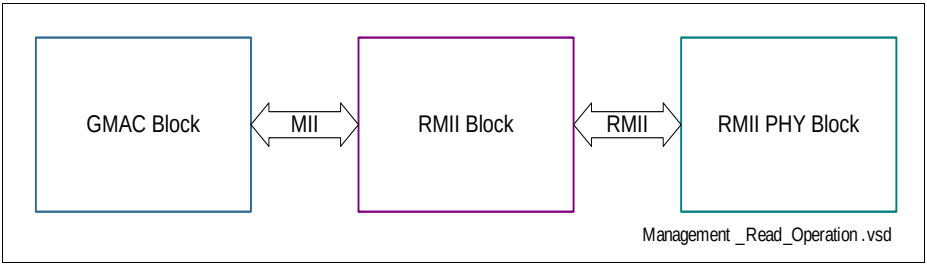


Figure 15-16 RMII Block Diagram

15.2.10.2 RMII Block Overview

The following list describes the RMII's hardware components, which are shown in **Figure 15-17**. Each of these blocks is briefly described in the following sections.

MII-RMII Transmit (MRT) Block: This block translates all MII transmit signals to RMII transmit signals.

MII-RMII Receive (MRR) Block: This block translates all RMII receive signals to MII receive signals.

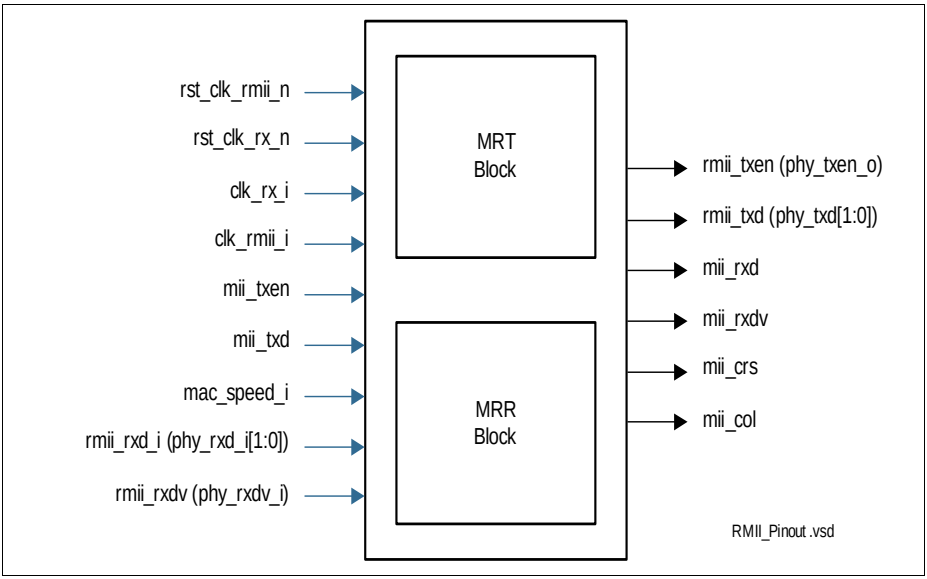


Figure 15-17 RMII Pinout

Note: The MAC Configuration.FES bit configures the RMII to operate at 10 Mbit/s or 100 Mbit/s.

15.2.10.3 Transmit Bit Ordering

Each nibble from the MII must be transmitted on the RMII a di-bit at a time with the order of di-bit transmission shown in **Figure 15-18**. The lower order bits (D1 and D0) are transmitted first followed by higher order bits (D2 and D3).

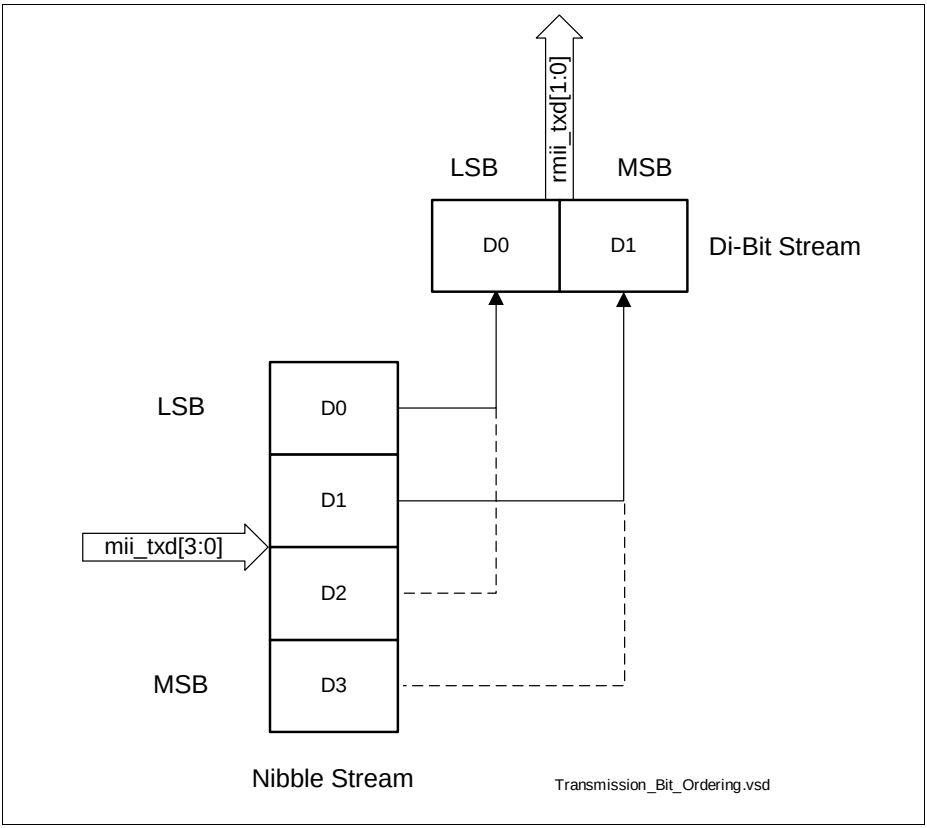


Figure 15-18 Transmission Bit Ordering

15.2.10.4 RMII Transmit Timing Diagrams

Figure 15-19 through **Figure 15-22** show MII-to-RMII transaction timing.

Figure 15-19 shows the start of MII transmission and the following RMII transmission in 100 Mbit/s mode.

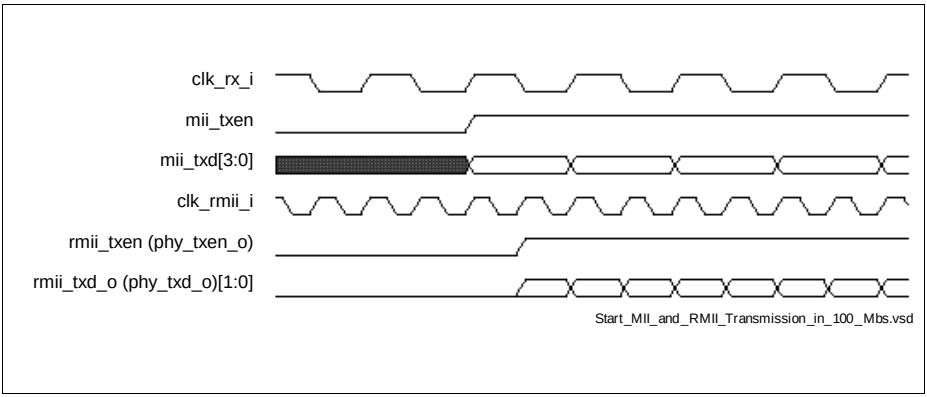


Figure 15-19 Start of MII and RMII Transmission in 100 Mbit/s Mode

Figure 15-20 shows the end of frame transmission for MII and RMII in 100 Mbit/s mode.

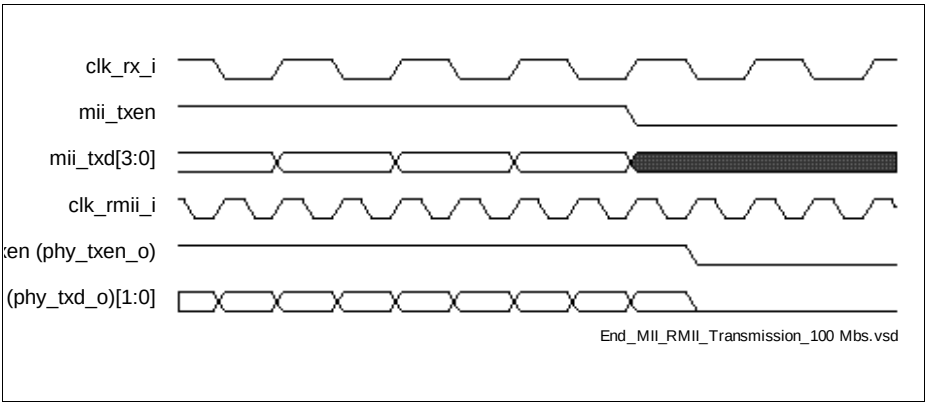


Figure 15-20 End of MII and RMII Transmission in 100 Mbit/s Mode

Figure 15-21 shows the start of MII transmission and the following RMII transmission in 10 Mbit/s mode.

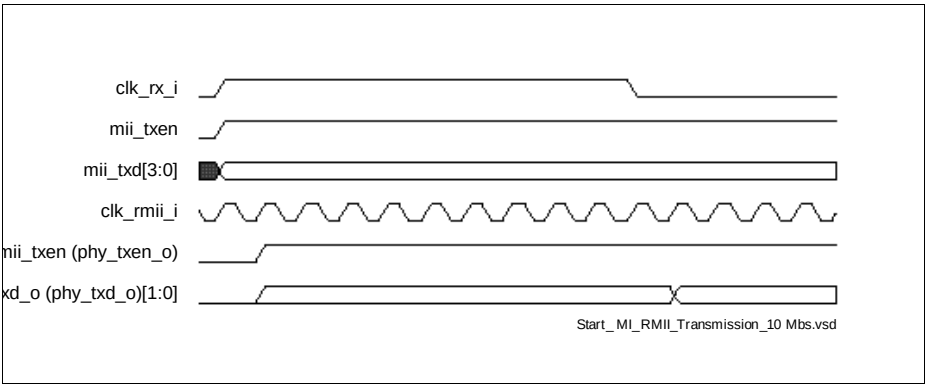


Figure 15-21 Start of MII and RMII Transmission in 10 Mbit/s Mode

Figure 15-22 shows the end of MII transmission and RMII transmission in 10 Mbit/s mode.

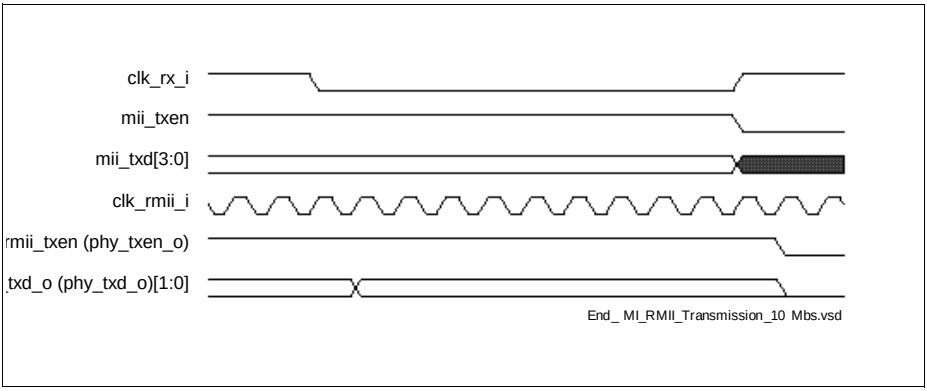


Figure 15-22 End of MII and RMII Transmission in 10 Mbit/s Mode

Receive Bit Ordering

Each nibble is transmitted to the MII from the di-bit received from the RMII in the nibble transmission order shown in **Figure 15-23**. The lower order bits (D0 and D1) are received first, followed by the higher order bits (D2 and D3).

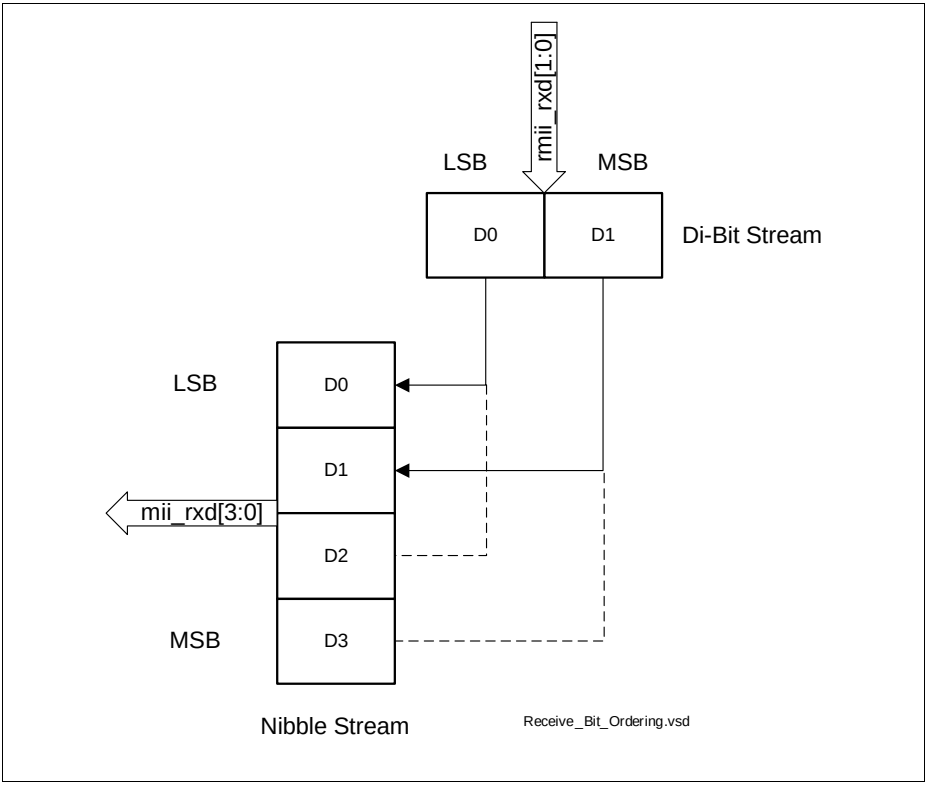


Figure 15-23 Receive Bit Ordering

15.2.11 IEEE 1588-2002 Overview

The IEEE 1588 standard defines a protocol enabling precise synchronization of clocks in measurement and control systems implemented with technologies such as network communication, local computing, and distributed objects. The protocol applies to systems communicating by local area networks supporting multicast messaging, including (but not limited to) Ethernet. This protocol enables heterogeneous systems that include clocks of varying inherent precision, resolution, and stability to synchronize. The protocol supports system-wide synchronization accuracy in the sub-microsecond range with minimal network and local clock computing resources.

The messaged-based protocol, named Precision Time Protocol (PTP), is transported over UDP/IP. The system or network is classified into Master and Slave nodes for distributing the timing/clock information. The protocol's technique for synchronizing a slave node to a master node by exchanging PTP messages is depicted in **Figure 15-24**.

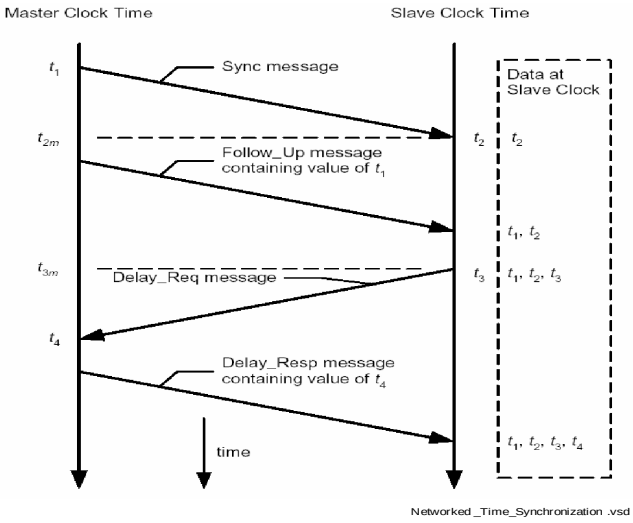


Figure 15-24 Networked Time Synchronization

1. The master broadcasts PTP Sync messages to all its nodes. The Sync message contains the master's reference time information. The time at which this message leaves the master's system is t1 and must, for Ethernet ports, be captured at the MII.
2. A slave receives the Sync message and also captures the exact time, t2, using its timing reference.
3. The master then sends the slave a Follow_up message, which contains t1 information for later use.
4. The slave sends the master a Delay_Req message, noting the exact time, t3, at which this frame leaves the MII.
5. The master receives this message, capturing the exact time, t4, at which it enters its system.
6. The master sends the t4 information to the slave in the Delay_Resp message.
7. The slave uses the four values of t1, t2, t3, and t4 to synchronize its local timing reference to the master's timing reference.

Most of the protocol implementation occurs in the software, above the UDP layer. As described above, however, hardware support is required to capture the exact time when specific PTP packets enter or leave the Ethernet port at the MII. This timing information must be captured and returned to the software for the proper implementation of PTP with high accuracy.

15.2.11.1 Reference Timing Source

To get a snapshot of the time, the core requires a reference time in 64-bit format (split into two 32-bit channels, with the upper 32-bits providing time in seconds, and the lower 32-bits indicating time in nanoseconds) as defined in the IEEE 1588 specification.

Internal Reference Time

This takes only the reference clock input and uses it to generate the Reference time (also called the System Time) internally and use it to capture time stamps. The generation, update, and modification of the System Time are described in **System Time Register Module**.

15.2.11.2 Transmit Path Functions

When a frame's SFD is output on the MII, a time stamp is captured. Frames for which capturing a time stamp is required are controllable on a per-frame basis. In other words, each transmit frame can be marked to indicate whether or not a time stamp must be captured for that frame.

No snooping or processing of the transmitted frames is performed to identify PTP frames. Framewise control is exercised through control bits in the transmit descriptor (as described in **Descriptor Format With IEEE 1588 Time Stamping Enabled**).

Captured time stamps are returned to the application in a manner similar to that in which status is provided for frames. time stamp is returned to software inside the corresponding transmit descriptor, thus connecting the time stamp automatically to the specific PTP frame. The 64-bit time stamp information is written back to the $TDES2_{RAM}$and $TDES3_{RAM}$fields, with TDES2 holding the time stamp's 32 least significant bits, except as described in **Transmit Time Stamp Field**.

Note: When the alternate (enhanced) descriptor is selected, the 64-bit time-stamp is written in $TDES6_{RAM}$and $TDES7_{RAM}$, respectively

15.2.11.3 Receive Path Functions

When the IEEE 1588 time-stamping feature is selected and enabled, the Ethernet MAC captures the time stamp of all frames received on the MII. No snooping or processing of the received frames is performed to identify PTP frames in the default mode (Advanced Time Stamp feature is not selected).

The core returns the time-stamp to the software in the corresponding receive descriptor. The 64-bit time stamp information is written back to the RDES2 and RDES3 fields, with RDES2 holding the time stamp's 32 least significant bits, except as mentioned in **Receive Time Stamp** The time stamp is only written to the receive descriptor for which the Last Descriptor status field has been set to 1 (the EOF marker). When the time stamp is not available (for example, due to an RxFIFO overflow) an all-ones pattern is written

to the descriptors (RDES2 and RDES3), indicating that time stamp is not correct. If the software uses a control register bit to disable time stamping, the DMA does not alter RDES2 or RDES3.

Note: When the alternate (enhanced) descriptor is selected, the 64-bit time-stamp is written in RDES6 and RDES7, respectively. RDES0[7] will indicate whether the time-stamp is updated in RDES6/7 or not.

15.2.11.4 Time Stamp Error Margin

According to the IEEE 1588 specifications, the time stamp must be captured at the SFD of transmitted and received frames at the MII interface. Since the reference timing source is different from the MII clocks, a small error margin is introduced, due to the transfer of information across asynchronous clock domains.

In the transmit path, the captured and reported time stamp has a maximum error margin of 2 PTP clocks. In other words, the captured time stamp has the value of the reference time source given within 2 clocks after the SFD has been transmitted on the MII.

Similarly, on the receive path, the error margin is 3 MII clocks, plus up to 2 PTP clocks. You can ignore the error margin due to the 3 MII clocks by assuming that this constant delay is present in the system (or link) before the SFD data reaches the ETH's MII interface.

15.2.11.5 Frequency Range of Reference Timing Clock

Because asynchronous logic is in place for time stamp information transfers across clock domain, a minimum delay is required between two consecutive time stamp captures. This delay is 3 clock cycles of both the MII and PTP clocks. If the gap is shorter, the ETH does not take a time stamp snapshot for the second frame.

The maximum PTP clock frequency is limited by the maximum resolution of the reference time and the timing constraints achievable for logic operating on the PTP clock. Another factor to consider is that the resolution, or granularity, of the reference time source determines the accuracy of the synchronization. Hence, a higher PTP clock frequency gives better system performance. The minimum PTP clock frequency depends on the time required between two consecutive SFD bytes. Because the MII clock frequency is fixed by IEEE specification, the minimum PTP clock frequency required for proper operation depends on the core's operating mode and operating speed.

For example, in 100 Mbit/s full-duplex operation, the minimum gap between two SFDs is 160 MII clocks (128 clocks for a 64-byte frame + 24 clocks of min IFG + 8 clocks of preamble).

In the example, $(3 \times \text{PTP}) + (3 \times \text{MII}) \leq 160 \times \text{MII}$; thus, the minimum PTP clock frequency is about 0.5 MHz $((160 - 3) \times 40\ \text{ns} \div 3 = 2.093\ \text{ns}$ period)

15.2.11.6 Advanced Time Stamp Feature Support

In addition to the basic features for time stamp mentioned in **Receive Time Stamp**, the advanced time stamp option has the following features.

- Support for the IEEE 1588-2008 (Version 2) timestamp format.
- Option to take snapshot for all frames or for PTP type frames.
- Option for taking snapshot for event messages only.
- Option to take the snapshot based on the clock type (ordinary, boundary, end-to-end and peer-to-peer)
- Option to select the node to be a Master or Slave for ordinary and boundary clock.
- Identification of PTP message type, version, and PTP payload sent directly over Ethernet given as status.
- Option to measure time in digital or binary format.

15.2.11.7 Peer-to-Peer PTP (Pdelay) Transparent Clock (P2P TC) Message Support

The IEEE 1588-2008 version supports Pdelay message in addition to SYNC, Delay Request, Follow-up and Delay Response messages. **Figure 15-25** shows the method to calculate the propagation delay in clocks supporting peer-to-peer path correction.

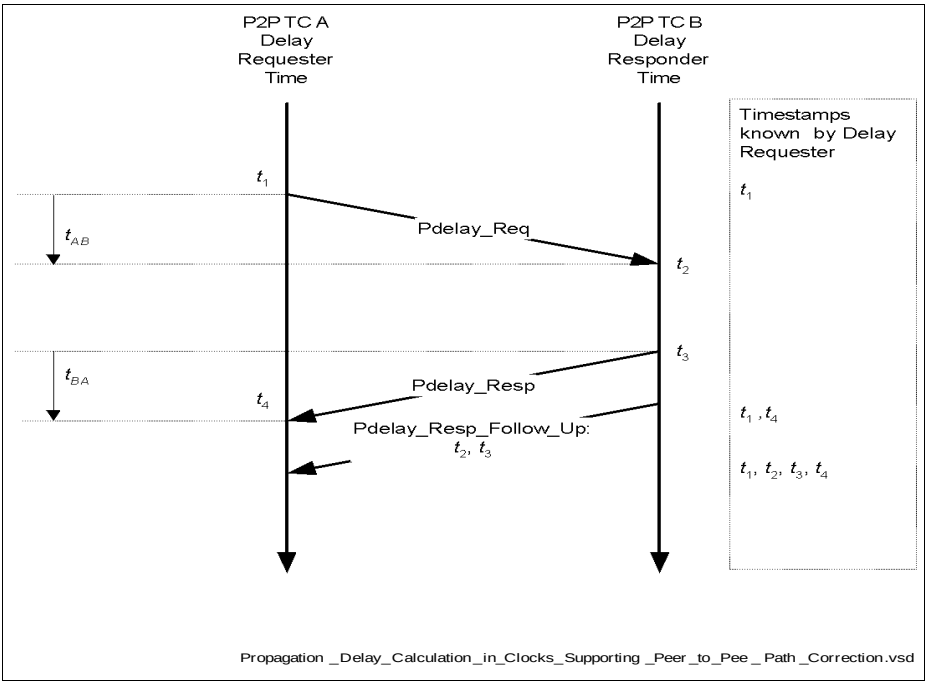


Figure 15-25 Propagation Delay Calculation in Clocks Supporting Peer-to-Peer Path Correction

The link delay measurement starts with port-1 issuing a “Pdelay_Req” message and generating a timestamp, for the Pdelay_Req message. Port-2 receives the “Pdelay_Req” message and generates a timestamp, t2, for this message. Port-2 returns a Pdelay_Resp message and generates a timestamp, t3, for this message. To minimize errors due to any frequency offset between the two ports, Port-2 returns the Pdelay_Resp message as quickly as possible after the receipt of the Pdelay_Req message.

Port-2 either:

- Returns the difference between the timestamps t2 and t3 in the Pdelay_Resp message,
- Returns the difference between the timestamps t2 and t3 in a Pdelay_Resp_Follow_Up message, or
- Returns the timestamps t2 and t3 in the Pdelay_Resp and Pdelay_Resp_Follow_Up messages respectively.

Port-1 generates a timestamp, t4, upon receiving the Pdelay_Resp message. Port-1 then uses these four timestamps to compute the mean link delay.

15.2.11.8 Clock Types

The type of clock nodes supported in IEEE 1588-2008 is described in this section. The corresponding support provided by the advanced time stamp feature for each of the clock type is also mentioned.

1. Ordinary clock support: In this type the clock can be a grandmaster or a slave clock. This clock has a single PTP state.
   Table 15-19 shows the messages for which time-stamp snapshot is taken on the receive side for Master and Slave nodes.
   The ordinary clock in the domain supports a single copy of the protocol and has a single PTP state and will typically be a single physical port. In typical industrial automation applications, an ordinary clock is associated with an application device such as sensors and actuators. In telecom applications, the ordinary clock may be associated with a timing demarcation device.
   Typically for ordinary clock, you will need to take snapshot for only one type of PTP messages. For e.g. you will require supporting either version 1 or 2 PTP messages, not both.
   The following features are supported.
   a) Sends and receives PTP messages. The time stamp snapshot can be controlled as described by the **ETH0_TIMESTAMP_CONTROL** Register.
   b) Maintains the data sets (e.g., time stamp values).
2. Boundary clock support: This type of clock is similar to the ordinary clock except for the following.
   Hence the features of ordinary clock holds good for the boundary clock also.
   The boundary clock typically has several physical ports communicating with the network. The messages related to synchronization, master-slave hierarchy and signaling terminate in the protocol engine of the boundary clock and are not forwarded. The PTP message type status given by the core (refer to **Receive Path Functions**) will help you to quickly identify the type of message and take appropriate action.
   a) The clock data sets are common to all ports of the boundary clock
   b) The local clock is common to all ports of the boundary clock.
3. End to end transparent clock support: The end-to-end transparent clock forwards all messages like normal bridge, router or repeater. The resident time needs to be computed to update the correctionField. Hence snapshot needs to be taken for the messages mentioned in **Table 15-20**.
   In the end-to-end transparent clock, the residence times are accumulated in a special field (correctionField) of the PTP event (SYNC) message or the associated Follow-up (FOLLOW_UP) Message. Hence it is important to take a snapshot for these messages alone. This can be quickly done by setting the control bit (TSEVNTENA), which enables snapshot to be taken for event messages and also selecting the type of clock in the **ETH0_TIMESTAMP_CONTROL** Register.
   The residence time is also corrected for Delay_Req messages (but snapshot of the

timestamp is not required). The message type statuses provided helps you to quickly identity the message and update the correctionField.
The message type status provided will also help in taking appropriate action depending on the type of PTP message received.

4. Peer to peer transparent clock support: In this type of clock the computation of the link delay is based on an exchange of Pdelay_Req, Pdelay_Resp and Pdelay_Resp_Follow_Up messages with the link peer. Hence support for taking snapshot for the event messages related to Pdelay is added. **Table 15-21**.
   The transparent clock corrects only the SYNC and Follow-up message. As discussed earlier this can be achieved using the message status provided.
   The type of clock to be implemented will be configurable through **ETH0_TIMESTAMP_CONTROL** register. To ensure that the snapshot is taken only for the messages indicated in the table for the corresponding clock type, the **ETH0_TIMESTAMP_CONTROL**.TSEVNTENA bit has to be set.

Table 15-19 PTP Messages for which Snapshot is Taken on Receive Side for Ordinary Clock

Master	Slave
Delay_Req	SYNC

Table 15-20 PTP Messages for which Snapshot is Taken for Transparent Clock Implementation

SYNC
FOLLOW_UP

Table 15-21 PTP Messages for which Snapshot is Taken for Peer-to-Peer Transparent Clock Implementation

SYNC
Pdelay_Req
Pdelay_Resp

15.2.11.9 PTP Processing and Control

The common message header for PTP messages is shown below. This format is taken from IEEE standard 1588-2008 (Revision of IEEE Std. 1588-2002).

Table 15-22 Message Format Defined in IEEE 1588-2008

BITS		OCTETS	OFFSET
transportSpecific	messageType	1	0
Reserved	versionPTP	1	1
messageLength		2	2
domainNumber		1	4
Reserved		1	5
flagField		2	6
correctionField		8	8
Reserved		4	16
sourcePortIdentity		10	20
sequenceId		2	30
controlField[1)]		1	32
logMessageInterva		1	33

1) controlField is used in version 1. For version 2, messageType field will be used for detecting different message types.

There are some fields in the PTP frame that are used to detect the type and control the snapshot to be taken. This is different for PTP frames sent directly over Ethernet, PTP frames sent over UDP / IPv4 and PTP frames that are sent over UDP / IPv6. The following sections provide information on the fields that are used to control taking the snapshot.

PTP Frame Over IPv4

Table 15-23 gives the details of the fields that will be matched to control snapshot for PTP packets over UDP over IPv4 for IEEE 1588 version 1 and 2. Note that the octet positions for tagged frames will be offset by 4. This is based on Appendix-D of the IEEE 1588-2008 standard and the message format defined in **Table 15-22**.

Table 15-23 IPv4-UDP PTP Frame Fields Required for Control and Status

Field Matched	Octet Position	Matched Value	Description
MAC Frame type	12, 13	0800_H	IPv4 datagram
IP Version and Header Length	14	45_H	IP version is IPv4
Layer-4 protocol	23	11_H	UDP

Table 15-23 IPv4-UDP PTP Frame Fields Required for Control and Status (cont'd)

Field Matched	Octet Position	Matched Value	Description
IP Multicast address (IEEE 1588 version 1)	30, 31, 32, 33	$E0_H$,00_H, 01_H,81_H (or 82_H or 83_H or 84_H)	Multicast IPv4 addresses allowed. 224.0.1.129 224.0.1.130 224.0.1.131 224.0.1.132
IP Multicast address (IEEE 1588 version 2)	30, 31, 32, 33	$E0_H$, 00_H, 01_H, 81_H $E0_H$, 00_H, 00_H, $6B_H$	PTP-primary multicast address: 224.0.1.129 PTP-Pdelay multicast address: 224.0.0.107
UDP destination port	36, 37	$013F_H$, 0140_H	$013F_H$ – PTP event message[1)] 0140_H – PTP general messages
PTP control field (IEEE version 1)	74	00_H/01_H/02_H/03_H /04_H	00_H – SYNC, 01_H – Delay_Req, 02_H – Follow_Up 03_H – Delay_Resp 04_H – Management
PTP Message Type Field (IEEE version 2)	42 (nibble)	0_H/1_H/2_H/3_H/8_H/9_H/B_H/C_H/D_H	0_H – SYNC 1_H – Delay_Req 2_H – Pdelay_Req 3_H – Pdelay_Resp 8_H – Follow_Up 9_H – Delay_Resp A_H – Pdelay_Resp_Follow_Up B_H – Announce C_H – Signaling D_H - Management
PTP version field	43 (nibble)	1_H or 2_H	1 – Supports PTP version 1 2 – Supports PTP version 2

1) PTP event messages are SYNC, Delay_Req (IEEE 1588 version 1 and 2) or Pdelay_Req, Pdelay_Resp (IEEE 1588 version 2 only).

PTP Frame Over IPv6

Table 15-24 gives the details of the fields that will be matched to control snapshot for PTP packets over UDP over IPv6 for IEEE 1588 version 1 and 2. Note that the octet positions for tagged frames will be offset by 4. This is based on Appendix-E of the IEEE 1588-2008 standard and the message format defined in **Table 15-22**.

Table 15-24 IPv6-UDP PTP Frame Fields Required for Control and Status

Field Matched	Octet Position	Matched Value	Description
MAC Frame type	12, 13	$86DD_H$	IP datagram
IP version	14 (bits [7:4])	6_H	IP version is IPv6
Layer-4 protocol	20[1)]	11_H	UDP
PTP Multicast address	38 – 53	FF0:0:0:0:0:0:0:1 81_H FF02:0:0:0:0:0:0: $6B_H$	PTP – primary multicast address: FF0:0:0:0:0:0:0:0:181$_H$ PTP – Pdelay multicast address: FF02:0:0:0:0:0:0:0:6B$_H$
UDP destination port	56, 57 (*)	$013F_H$, 140_H	$013F_H$ – PTP event message 0140_H – PTP general messages
PTP control field (IEEE 1588 Version 1)	93 (*)	00_H/01_H/02_H/03_H/ 04_H	00_H – SYNC, 01_H – Delay_Req, 02_H – Follow_Up 03_H – Delay_Resp 04_H – Management (version1)
PTP Message Type Field (IEEE version 2)	74 (*) (nibble)	0_H/1_H/2_H/3_H/8_H/9_H/B_H/C_H/D_H	0_H – SYNC 1_H – Delay_Req 2_H – Pdelay_Req 3_H – Pdelay_Resp 8_H – Follow_Up 9_H – Delay_Resp A_H – Pdelay_Resp_Follow_Up B_H – Announce C_H – Signaling D_H - Management
PTP version field	75 (nibble)	1_H or 2_H	1_H – Supports PTP version 1 2_H – Supports PTP version 2

1) The Extension Header is not defined for PTP packets.

PTP Frame Over Ethernet

Table 15-25 gives the details of the fields that will be matched to control snapshot for PTP packets over Ethernet for IEEE 1588 version 1 and 2. Note that the octet positions

for tagged frames will be offset by 4. This is based on Appendix-E of the IEEE 1588-2008 standard and the message format defined in **Table 15-22**.

Table 15-25 Ethernet PTP Frame Fields Required for Control And Status

Field Matched	Octet Position	Matched Value	Description
MAC Frame type	12, 13	$88F7_H$	PTP Ethernet frame.
PTP control field (IEEE Version 1)	45	$00_H/01_H/02_H/03_H/04_H$	00_H – SYNC 01_H – Delay_Req 02_H – Follow_Up 03_H – Delay_Resp 04_H – Management
PTP Message Type Field (IEEE version 2)	14 (nibble)	$0_H/1_H/2_H/3_H/8_H/9_H/B_H/ C_H/D_H$	0_H – SYNC 1_H – Delay_Req 2_H – Pdelay_Req 3_H – Pdelay_Resp 8_H – Follow_Up 9_H – Delay_Resp A_H – Pdelay_Resp_Follow_Up B_H – Announce C_H – Signaling D_H – Management
MAC Destination multicast address[1)]	0-5	01-1B-19-00-00-00_H 01-80-C2-00-00-$0E_H$	All except peer delay messages - 01-1B-19-00-00-00_H Pdelay messages - 01-80-C2-00-00-$0E_H$
PTP version field	15 (nibble)	1_H or 2_H	1_H – Supports PTP version 1 2_H – Supports PTP version 2

1) In addition, the address match of destination addresses (DA) programmed in MAC address 1 to 31 will be used, if the control bit 18 (TSENMACADDR: Enable MAC address for PTP frame filtering) of the Time Stamp Control register is set.

15.2.11.10Reference Timing Source (for Advance Timestamp Feature)

The updated functionality for advanced timestamp support is mentioned in the following points.

1. The IEEE 1588-2008 standard defines the seconds field of the time to be 48 bits wide. The fields to time-stamp will be the following.
   a) UInteger48- seconds field

b) UInteger32-nanoseconds field
The “seconds” field is the integer portion of the timestamp in units of seconds. The “nanoseconds” field is the fractional portion of the timestamp in units of nanoseconds. E.g. 2.000000001 seconds is represented as secondsField = 0000 0000 0002_H and nanoSeconds = 0000 0001_H. Thus the maximum value in nanoseconds field in this format will be 3B9A $C9FF_H$ value (i.e (10e9-1) nano-seconds). This is defined as digital rollover mode of operation. It will also support the older mode in which the nano-seconds field will roll-over and increment the seconds field after the value of 7FFF $FFFF_H$. (Accuracy is ~0.466 ns per bit). This is defined as the binary rollover mode. The modes can be controlled using the “**ETH0_TIMESTAMP_CONTROL**.TSCTRLSSR bit.

2. When the Advanced IEEE 1588 time-stamp feature is selected time maintained in the core will still be 64-bit wide, as the overflow to the upper 16-bits of seconds register happens once in 130 years. The value of the upper 16-bits of the seconds field can only be obtained from the CSR register.
3. There is also a pulse-per-second output given to indicate 1 second interval (default). Option is provided to change the interval in the **ETH0_PPS_CONTROL** Register.
4.

15.2.11.11Transmit Path Functions

There are no changes in the transmit path functions for ETH-CORE and ETH-MTL configuration for the Advanced time stamp option.

structure of the descriptor changes when Advanced IEEE 1588 version support is enabled. The IEEE 1588 timestamp feature is supported using Alternate (Enhanced) descriptors format only. The descriptor is 32-bytes long (8 DWORDS) and the snapshot of the timestamp is written in descriptor 6 and 7.

15.2.11.12Receive Path Functions

When the advanced time stamp feature is selected, processing of the received frames to identify valid PTP frames is done. The snapshot of the time to be sent to the application can be controlled.

The following options are provided in the TIMESTAMP_CONTROL register to control the snapshot.

1. Option to enable snapshot for all frames.
2. Enable snapshot for IEEE 1588 version 2 or version 1 time stamp.
3. Enable snapshot for PTP frames transmitted directly over Ethernet or UDP-IP-Ethernet.
4. Enable time stamp snapshot for the received frame for IPv4 or IPv6.
5. Enable time stamp snapshot for EVENT messages (SYNC, DELAY_REQ, PDELAY_REQ or PDELAY_RESP) only.

6. Enable the node to be a Master or Slave. This will control the type of messages for which snap-shot will be taken (this depends on the type of clock that is selected and is valid for ordinary or boundary clock only).

Note that PTP messages over VLAN frames are also supported.

15.2.12 System Time Register Module

A system time clock is maintained in this module. A 64 bit timer is incremented using the PTP clock as refference. This time is the source for taking snapshots (time stamps) of Ethernet frames being transmitted or received at the MII.

The System Time counter can be initialized or corrected using the coarse correction method. In this method, the initial value or the offset value is written to the Time Stamp Update register. For initialization, the System Time counter(**ETH0_SYSTEM_TIME_SECONDS** and **ETH0_SYSTEM_TIME_NANOSECONDS**) is written with the value in the Time Stamp Update registers (**ETH0_SYSTEM_TIME_SECONDS_UPDATE** and **ETH0_SYSTEM_TIME_NANOSECONDS_UPDATE**) , while for system time correction, the offset value is added to or subtracted from the system time.

In the fine correction method, a slave clock's frequency drift with respect to the master clock (as defined in IEEE 1588) is corrected over a period of time instead of in one clock, as in coarse correction. This helps maintain linear time and does not introduce drastic changes (or a large jitter) in the reference time between PTP Sync message intervals. In this method, an accumulator sums up the contents of the Addend register, as shown in **Figure 15-26**. The arithmetic carry that the accumulator generates is used as a pulse to increment the system time counter. The accumulator and the addend are 32-bit registers. Here, the accumulator acts as a high-precision frequency multiplier or divider.

This algorithm is depicted in **Figure 15-26**:

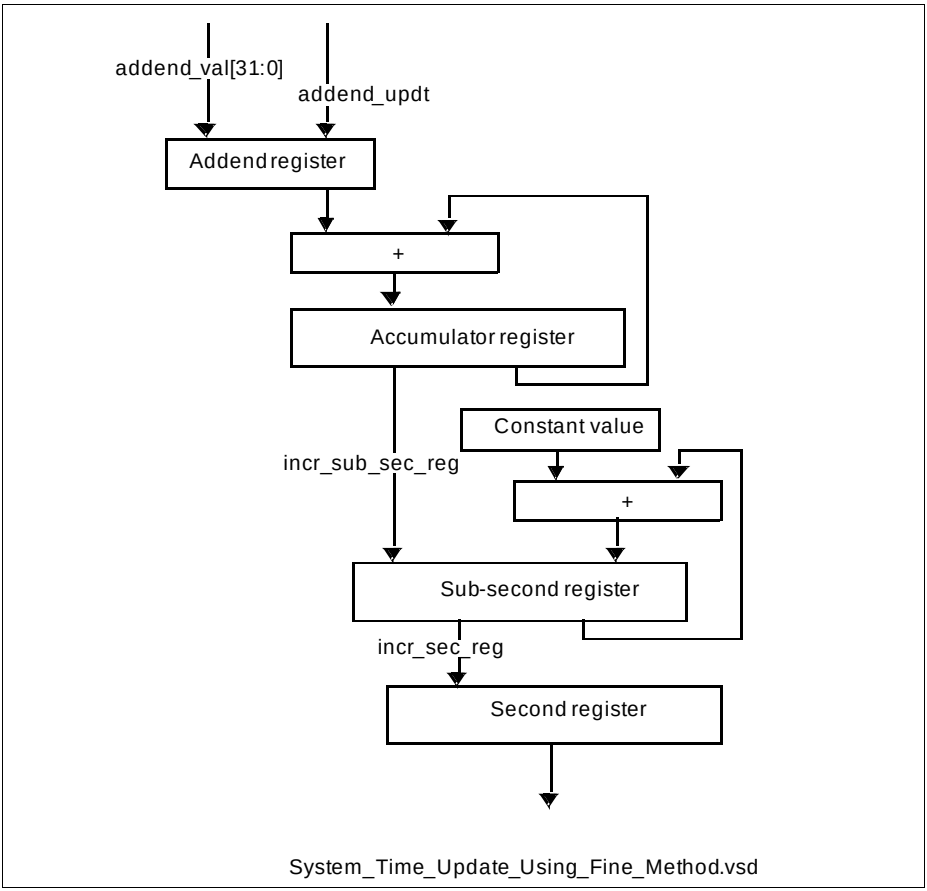


Figure 15-26 System Time Update Using Fine Method

The System Time Update logic requires a 50-MHz clock frequency to achieve 20-ns accuracy. The frequency division is the ratio of the reference clock frequency to the required clock frequency. Hence, if the reference clock is, for example, 66 MHz, this ratio is calculated as 66 MHz / 50 MHz = 1.32. Hence, the default addend value to be set in the register is 2^{32} / 1.32, $0C1F07C1F_H$.

If the reference clock drifts lower, to 65 MHz for example, the ratio is 65 / 50, or 1.3 and the value to set in the addend register is 2^{32} / 1.30, or $0C4EC4EC4_H$. If the clock drifts higher, to 67 MHz for example, the addend register must be set to $0BF0B7672_H$. When the clock drift is nil, the default addend value of $0C1F07C1F_H$ (2^{32} / 1.32) must be programmed.

In **Figure 15-26**, the constant value used to accumulate the sub-second register is decimal 43, which achieves an accuracy of 20 ns in the system time (in other words, it is incremented in 20-ns steps). Two different methods are used to update the System Time register, depending on which configuration you choose (See **Block Diagram**).

The software must calculate the drift in frequency based on the Sync messages and update the Addend register accordingly.

Initially, the slave clock is set with FreqCompensationValue0 in the Addend register. This value is as follows:

$FreqCompensationValue_0 = 2^{32} / FreqDivisionRatio$

If MasterToSlaveDelay is initially assumed to be the same for consecutive Sync messages, the algorithm described below must be applied. After a few Sync cycles, frequency lock occurs. The slave clock can then determine a precise MasterToSlaveDelay value and re-synchronize with the master using the new value.

The algorithm is as follows:

- At time $MasterSyncTime_n$ the master sends the slave clock a Sync message. The slave receives this message when its local clock is $SlaveClockTime_n$ and computes $MasterClockTime_n$ as:
  $MasterClockTime_n = MasterSyncTime_n + MasterToSlaveDelay_n$
- The master clock count for current Sync cycle, $MasterClockCount_n$ is given by:
  $MasterClockCount_n = MasterClockTime_n – MasterClockTime_{n-1}$ (assuming that MasterToSlaveDelay is the same for Sync cycles n and n – 1)
- The slave clock count for current Sync cycle, $SlaveClockCount_n$ is given by:
  $SlaveClockCount_n = SlaveClockTime_n – SlaveClockTime_{n-1}$
- The difference between master and slave clock counts for current Sync cycle, $ClockDiffCount_n$ is given by:
  $ClockDiffCount_n = MasterClockCount_n – SlaveClockCount_n$
- The frequency-scaling factor for slave clock, $FreqScaleFactor_n$ is given by:
  $FreqScaleFactor_n = (MasterClockCount_n + ClockDiffCount_n) / SlaveClockCount_n$
- The frequency compensation value for Addend register, $FreqCompensationValue_n$ is given by:
  $FreqCompensationValue_n = FreqScaleFactor_n * FreqCompensationValue_{n-1}$

In theory, this algorithm achieves lock in one Sync cycle; however, it may take several cycles, due to changing network propagation delays and operating conditions.

This algorithm is self-correcting: if for any reason the slave clock is initially set to a value from the master that is incorrect, the algorithm will correct it at the cost of more Sync cycles.

15.2.13 Application BUS Interface

In the ETH core, the DMA Controller interfaces with the CPU through the Bus Interface. The Bus Master Interface controls data transfers while the Bus Slave interface accesses

CSR space. The DMA can be used in applications where DMA is required to optimize data transfer between the ETH and system memory.

The Bus Master interface converts the internal DMA request cycles into Bus cycles.

Characteristics of this interface include the following:

- You can choose fixed burst length of SINGLE, INCR4, INCR8 by programming the **ETH0_BUS_MODE**.MB bits
  - When transfering fixed burst length data , the Bus master always initiates a burst with SINGLE or INCR4/8type. But when such a burst is responded with SPLIT/RETRY/early burst termination, the Bus master will re-initiate the pending transfers of the burst with INCR or SINGLE burst-length type. It will terminate such INCR bursts when the original requested fixed-burst is transferred. In Fixed Burst-Length mode, if the DMA requests a burst transfer that is not equal to INCR4/8, the Bus interface splits the transfer into multiple burst transactions. For example, if the DMA requests a 15-beat burst transfer, the Bus interface splits it into multiple transfers of INCR8 and INCR4 and 3 SINGLE transactions.
- Takes care of Bus SPLIT, RETRY, and ERROR conditions. Any ERROR response will halt all further transactions for that DMA, and indicate the error as fatal through the CSR and interrupt. The application must give a hard or soft reset to the module to restart the operation.
- Takes care of Bus 1K boundary breaking
- Handles all data transfers, except for Descriptor Status Write accesses (which are always 32-bit). In any burst data transfer, the address bus value is always aligned to the data bus width and need not be aligned to the beat size.
  All Bus burst transfers can be aligned to an address value by enabling the **ETH0_BUS_MODE**.AAL bit. If both the FB and AAL bits are set to 1, the Bus interface and the DMA together ensure that all initiated beats are aligned to the address, completing the frame transfer in the minimum number of required beats.
  For example, if a data buffer transfer's start address is F000 0008_H and the DMA is configured for a maximum beat size of, the Bus transfers occur in the following sequence:
  – 2 SINGLE transfers at addresses F000 0008_H and F000 $000C_H$
  – 1 INCR4 transfer at address F000 0010_H
- The DMA Controller requests an Bus Burst Read transfer only when it can accept the received burst data completely. Data read from the Bus is always pushed into the DMA without any delay or BUSY cycles.
- The DMA requests an Bus Burst Write transfer only when it has the sufficient data to transfer the burst completely. The Bus interface always assumes that it has data available to push into the bus. However, the DMA can prematurely indicate end-of-valid data (due to the transfer of end-of-frame of an Ethernet frame) during the burst. The Bus Master interface continues the burst with dummy data until the specified length is completed.

The Bus 32-bit Slave interface provides access to the DMA and ETH CSR space. Characteristics of this interface include the following:

- Supports single and INCR4/8transfers
- Supports busy and early terminations
- Supports 32-bit, 16-bit, and 8-bit write/read transfers to the CSR; 32-bit access to the CSR are recommended to avoid any SW synchronization problems.
- Generates OKAY only response; does not generate SPLIT, RETRY, or ERROR responses.

15.3 Service Request Generation

Service requests can be generated from the ETH core as a result of various events in the modules within the ETH peripheral. There are four sources of service request, the ETH DMA, The Power Management module, the System timer module and the MAC management counters. Each of the events raised by these modules are ORed together and connected to an ETH Service Request line which is connected to the NVIC. The events are not queued and the application software must check all the status bits t ensure all events are serviced. Before exiting the service request routine the application software must ensure all status bits are de asserted or spurious service requests will be generated

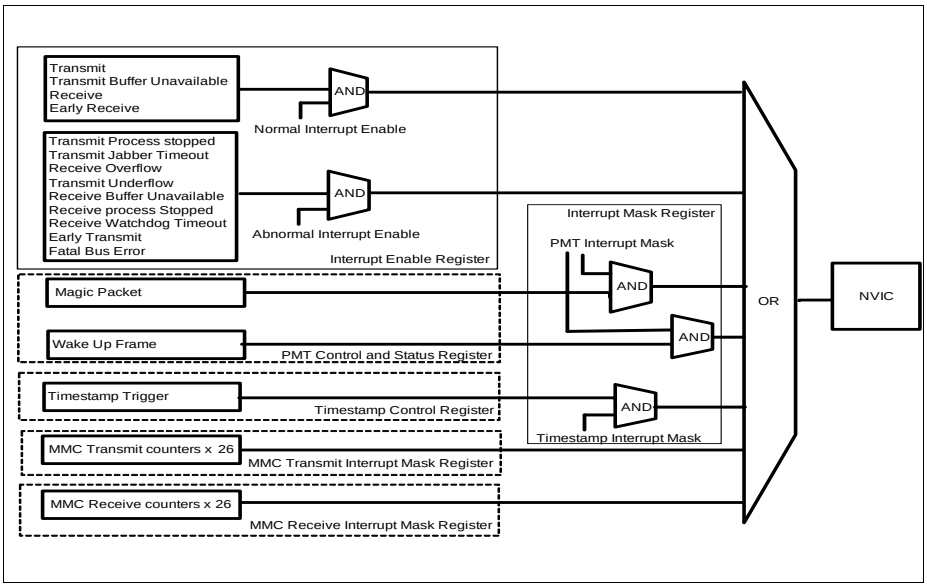


Figure 15-27 ETH Core Service Request Structure

15.3.1 DMA Service Requests

The ETH DMA has two groups of Service Request, Normal and Abnormal requests. Each Service Request source is enabled in the**ETH0_INTERRUPT_ENABLE** register. Each group of Service Requests must also be enabled by setting the Normal Interrupt enable and Abnormal Interrupt enable bits in the same register. When a service request is raised the matching Service request bits will be set in the **ETH0_STATUS** register. Global summary bits for the Power management MAC management counters and system time module are also provided in the status register.

15.3.2 Power Management Service Requests

The power management module provides two Service Requests, Wake up Frame and Magic packet. Both of these service requests may be enabled and monitored in the **PMT_CONTROL_STATUS** Register. To enable any power management service request is it also necessary to clear the **INTERRUPT_MASK**.PMTIM bit. The **INTERRUPT_STATUS**.PMTIS bit provides a global status bit for power management service requests.

15.3.3 System Time Module

The system time module provides a single Timestamp trigger service request which can be enabled in the timestanp control register. The **INTERRUPT_MASK**.TSIM bit must also be cleared to enable the timestamp trigger Service request.The **INTERRUPT_STATUS**.TSIS is set when a system time module service request occurs.

15.3.4 MAC Management Counter Service Requests

Each of the MAC Management Counters can generate a service request. The service requests are split into two groups of transmit and receive counters. Each counter may be individually enabled in either the **MMC_RECEIVE_INTERRUPT_MASK** register or the **MMC_TRANSMIT_INTERRUPT_MASK** register. When a MMC service request is generated status bits for each counter are set in the **MMC_RECEIVE_INTERRUPT** register or the **MMC_TRANSMIT_INTERRUPT** register.Two global status bits for the transmit and receive counters are provided in the **INTERRUPT_STATUS** register

15.4 Debug

Module specific debug behaviour TBD

In addition the ETH has a number of intrinsic features to assist debugging, these are described below.

- The **DEBUG** register provides flags which indicate the operating status of the ETH MAC and MTL.
- The **STATUS** register provides information on the operating status of the DMA.
- The **MAC Management Counters** provide extensive information about the Received and transmitted Ethernet frames.
- The **MAC_CONFIGURATION**.LM bit places the ETH in internal loopback mode for self test and debug
- External loopback is supported via the integrated MDIO controlling the PHY
- The**CURRENT_HOST_TRANSMIT_DESCRIPTOR** and **CURRENT_HOST_RECEIVE_DESCRIPTOR** provide pointers to the current location of the transmit and receive frame buffers held in RAM

15.5 Power Reset and Clock

The module, including all registers, can be reset to its default state by a system reset or a software reset triggered through the setting of corresponding bits in PRSETx registers.

The module has the following input clocks:

- clk_eth_ahb : The module clock
- clk_eth_sram : A seperate clock for the module internal RAM

Important

After the XMC4500 is released from reset the ETH module remains held in reset. While the ETH is held in reset the software driver must select the PHY interconnect see **Section 15.2.7.1** . Once the PHY interconnect has been selected ETH reset line must be deasserted by setting PRCLR2.ETH0RS in the system control unit.

The clk_eth_ahb has a minumum frequency of 50Mhz.

The clk_eth_sram frequency must be 2 x the clk_eth_ahb and has a minumum frequency of 100Mhz.

15.6 ETH Registers

The application controls the ETH by reading from and writing to the Control and Status Registers (CSRs) through the BUS Slave interface. These registers are 32 bits wide and the addresses are 32-bit block aligned

15.6.1 Register Description

ETH Register Map

Table 15-27 provides the address map of the ETH core registers.

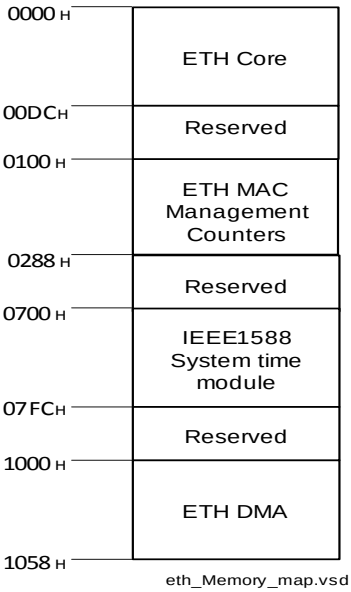


Figure 15-28 ETH Register memory Map

Table 15-26 Registers Address Space - ETH Module

Module	Base Address	End Address	Note
ETH0	5000 C000$_H$	5000 FFFF$_H$	-

15.6.2 Registers Overview

Table 15-27 ETH Registers Overview

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
MAC Configuration Registers					
MAC_CONFIGURATION	MAC Configuration Register	0000_H	U,PV	U,PV	Page 15-110
MAC_FRAME_FILTER	MAC Frame Filter Register	0004_H	U,PV	U,PV	Page 15-117
HASH_TABLE_HIGH	Hash Table High Register	0008_H	U,PV	U,PV	Page 15-122
HASH_TABLE_LOW	Hash Table Low Register	$000C_H$	U,PV	U,PV	Page 15-124
GMII_ADDRESS	MII Address Register	0010_H	U,PV	U,PV	Page 15-125
GMII_DATA	MII Data Register	0014_H	U,PV	U,PV	Page 15-128
FLOW_CONTROL	Flow Control Register	0018_H	U,PV	U,PV	Page 15-129
VLAN_TAG	VLAN Tag Register	$001C_H$	U,PV	U,PV	Page 15-133
VERSION	Version Register	0020_H	U,PV	NC	Page 15-135
DEBUG	Debug Register	0024_H	U,PV	U,PV	Page 15-136
REMOTE_WAKE_UP_FRAME_FILTER	Remote Wake Up Frame Filter Register	0028_H	U,PV	U,PV	Page 15-139
PMT_CONTROL_STATUS	PMT Control Status Register	$002C_H$	U,PV	U,PV	Page 15-140
Do not use	Do not use	0030_H - 0034_H	nBE	nBE	Do not use

Table 15-27 ETH Registers Overview (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
INTERRUPT_STATUS	Interrupt Status Register	0038_H	U,PV	U,PV	Page 15-142
INTERRUPT_MASK	Interrupt Mask Register	$003C_H$	U,PV	U,PV	Page 15-144]
MAC_ADDRESS0_HIGH	MAC Address 0 High Register	0040_H	U,PV	U,PV	Page 15-145
MAC_ADDRESS0_LOW	MAC Address0 Low Register	0044_H	U,PV	U,PV	Page 15-146]
MAC_ADDRESS1_HIGH	MAC Address1 High Register	0048_H	U,PV	U,PV	Page 15-147
MAC_ADDRESS1_LOW	MAC Address1 Low Register	$004C_H$	U,PV	U,PV	Page 15-149
MAC_ADDRESS2_HIGH	MAC Address High Register	0050_H	U,PV	U,PV	Page 15-150
MAC_ADDRESS2_LOW	MAC Address1 Low Register	0054_H	U,PV	U,PV	Page 15-152
MAC_ADDRESS3_HIGH	MAC Address High Register	0058_H	U,PV	U,PV	Page 15-153
MAC_ADDRESS3_LOW	MAC Address1 Low Register	$005C_H$	U,PV	U,PV	Page 15-155
Do not use	Do not use	$00DC_H$ - $00FC_H$	nBE	nBE	Do not use
MAC Mangement Counters					
MMC_CONTROL	MMC Control	0100_H	U,PV	U,PV	Page 15-156
MMC_RECEIVE_INTERRUPT	MMC Receive Interrupt	0104_H	U,PV	U,PV	Page 15-158

Table 15-27 ETH Registers Overview (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
MMC_TRANSMIT_INTERRUPT	MMC Transmit Interrupt	0108_H	U,PV	U,PV	Page 15-163
MMC_RECEIVE_INTERRUPT_MASK	MMC Receive Interrupt mask	$010C_H$	U,PV	U,PV	Page 15-168
MMC_TRANSMIT_INTERRUPT_MASK	MMC Transmit Interrupt Mask	0110_H	U,PV	U,PV	Page 15-173
TX_OCTET_COUNT_GOOD_BAD	Number of bytes transmitted, exclusive of preamble and retried bytes, in good and bad frames.	0114_H	U,PV	U,PV	Page 15-178
TX_FRAME_COUNT_GOOD_BAD	Number of good and bad frames transmitted, exclusive of retried frames.	0118_H	U,PV	U,PV	Page 15-179
TX_BROADCAST_FRAMES_GOOD	Number of good broadcast frames transmitted.	$011C_H$	U,PV	U,PV	Page 15-180
TX_MULTICAST_FRAMES_GOOD	Number of good multicast frames transmitted.	0120_H	U,PV	U,PV	Page 15-181
TX_64OCTETS_FRAMES_GOOD_BAD	Number of good and bad frames transmitted with length 64 bytes, exclusive of preamble and retried frames.	0124_H	U,PV	U,PV	Page 15-182
TX_65TO127OCTETS_FRAMES_GOOD_BAD	Number of good and bad frames transmitted with length between 65 and 127 (inclusive) bytes, exclusive of preamble and retried frames.	0128_H	U,PV	U,PV	Page 15-183

Table 15-27 ETH Registers Overview (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
TX_128TO255OCTETS_FRAMES_GOOD_BAD	Number of good and bad frames transmitted with length between 128 and 255 (inclusive) bytes, exclusive of preamble and retried frames.	$012C_H$	U,PV	U,PV	Page 15-184
TX_256TO511OCTETS_FRAMES_GOOD_BAD	Number of good and bad frames transmitted with length between 256 and 511 (inclusive) bytes, exclusive of preamble and retried frames.	0130_H	U,PV	U,PV	Page 15-185
TX_512TO1023OCTETS_FRAMES_GOOD_BAD	Number of good and bad frames transmitted with length between 512 and 1.023 (inclusive) bytes, exclusive of preamble and retried frames.	0134_H	U,PV	U,PV	Page 15-186
TX_1024TOMAXOCTETS_FRAMES_GOOD_BAD	Number of good and bad frames transmitted with length between 1.024 and maxsize (inclusive) bytes, exclusive of preamble and retried frames.	0138_H	U,PV	U,PV	Page 15-187 .
TX_UNICAST_FRAMES_GOOD_BAD	Number of good and bad unicast frames transmitted.	$013C_H$	U,PV	U,PV	Page 15-188
TX_MULTICAST_FRAMES_GOOD_BAD	Number of good and bad multicast frames transmitted.	0140_H	U,PV	U,PV	Page 15-189
TX_BROADCAST_FRAMES_GOOD_BAD	Number of good and bad broadcast frames transmitted.	0144_H	U,PV	U,PV	Page 15-190

Table 15-27 ETH Registers Overview (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
TX_UNDERFLOW_ERROR_FRAMES	Number of frames aborted due to frame underflow error.	0148_H	U,PV	U,PV	Page 15-191
TX_SINGLE_COLLISION_GOOD_FRAMES	Number of successfully transmitted frames after a single collision in Half-duplex mode.	$014C_H$	U,PV	U,PV	Page 15-192
TX_MULTIPLE_COLLISION_GOOD_FRAMES	Number of successfully transmitted frames after more than a single collision in Half-duplex mode.	0150_H	U,PV	U,PV	Page 15-193
TX_DEFERRED_FRAMES	Number of successfully transmitted frames after a deferral in Half-duplex mode.	0154_H	U,PV	U,PV	Page 15-194
TX_LATE_COLLISION_FRAMES	Number of frames aborted due to late collision error.	0158_H	U,PV	U,PV	Page 15-195
TX_EXCESSIVE_COLLISION_FRAMES	Number of frames aborted due to excessive (16) collision errors.	$015C_H$	U,PV	U,PV	Page 15-196
TX_CARRIER_ERROR_FRAMES	Number of frames aborted due to carrier sense error (no carrier or loss of carrier).	0160_H	U,PV	U,PV	Page 15-197
TX_OCTET_COUNT_GOOD	Number of bytes transmitted, exclusive of preamble, in good frames only.	0164_H	U,PV	U,PV	Page 15-198
TX_FRAME_COUNT_GOOD	Number of good frames transmitted.	0168_H	U,PV	U,PV	Page 15-199

Table 15-27 ETH Registers Overview (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
TX_EXCESSIVE_DEFERRAL_ERROR	Number of frames aborted due to excessive deferral error (deferred for more than two max-sized frame times).	$016C_H$	U,PV	U,PV	Page 15-200
TX_PAUSE_FRAMES	Number of good PAUSE frames transmitted.	0170_H	U,PV	U,PV	Page 15-201
TX_VLAN_FRAMES_GOOD	Number of good VLAN frames transmitted, exclusive of retried frames.	0174_H	U,PV	U,PV	Page 15-202
TX_OSIZE_FRAMES_GOOD	Number of transmitted good Oversize frames, exclusive of retried frames.	0178_H	U,PV	U,PV	Page 15-203
Reserved		$017C_H$	nBE	nBE	
RX_FRAMES_COUNT_GOOD_BAD	Number of good and bad frames received.	0180_H	U,PV	U,PV	Page 15-204
RX_OCTET_COUNT_GOOD_BAD	Number of bytes received, exclusive of preamble, in good and bad frames.	0184H	U,PV	U,PV	Page 15-205
RX_OCTET_COUNT_GOOD	Number of bytes received, exclusive of preamble, only in good frames.	0188H	U,PV	U,PV	Page 15-206
RX_BROADCAST_FRAMES_GOOD	Number of good broadcast frames received.	018CH	U,PV	U,PV	Page 15-207
RX_MULTICAST_FRAMES_GOOD	Number of good multicast frames received.	0190H	U,PV	U,PV	Page 15-208

Table 15-27 ETH Registers Overview (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
RX_CRC_ERROR_FRAMES	Number of frames received with CRC error.	0194H	U,PV	U,PV	Page 15-209
RX_ALIGNMENT_ERROR_FRAMES	Number of frames received with alignment (dribble) error.	0198H	U,PV	U,PV	Page 15-210
RX_RUNT_ERROR_FRAMES	Number of frames received with runt (<64 bytes and CRC error) error.	019CH	U,PV	U,PV	Page 15-211
RX_JABBER_ERROR_FRAMES	Number of giant frames received with length (including CRC) greater than 1.518 bytes (1.522 bytes for VLAN tagged) and with CRC error. If Jumbo Frame mode is enabled, then frames of length greater than 9,018 bytes (9,022 for VLAN tagged) are considered as giant frames.	01A0H	U,PV	U,PV	Page 15-212
RX_UNDERSIZE_FRAMES_GOOD	Number of frames received with length less than 64 bytes, without any errors.	01A4H	U,PV	U,PV	Page 15-213
RX_OVERSIZE_FRAMES_GOOD	Number of frames received with length greater than the maxsize (1.518 or 1.522 for VLAN tagged frames), without errors.	01A8H	U,PV	U,PV	Page 15-214
RX_64OCTETS_FRAMES_GOOD_BAD	Number of good and bad frames received with length 64 bytes, exclusive of preamble.	01ACH	U,PV	U,PV	Page 15-215

Table 15-27 ETH Registers Overview (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
RX_65TO127OCTETS_FRAMES_GOOD_BAD	Number of good and bad frames received with length between 65 and 127 (inclusive) bytes, exclusive of preamble.	01B0H	U,PV	U,PV	Page 15-216
RX_128TO255OCTETS_FRAMES_GOOD_BAD	Number of good and bad frames received with length between 128 and 255 (inclusive) bytes, exclusive of preamble.	01B4H	U,PV	U,PV	Page 15-217
RX_256TO511OCTETS_FRAMES_GOOD_BAD	Number of good and bad frames received with length between 256 and 511 (inclusive) bytes, exclusive of preamble.	01B8H	U,PV	U,PV	Page 15-218
RX_512TO1023OCTETS_FRAMES_GOOD_BAD	Number of good and bad frames received with length between 512 and 1.023 (inclusive) bytes, exclusive of preamble.	01BCH	U,PV	U,PV	Page 15-219
RX_1024TOMAXOCTETS_FRAMES_GOOD_BAD	Number of good and bad frames received with length between 1.024 and maxsize (inclusive) bytes, exclusive of preamble and retried frames.	01C0H	U,PV	U,PV	Page 15-220
RX_UNICAST_FRAMES_GOOD	Number of good unicast frames received.	01C4H	U,PV	U,PV	Page 15-221

Table 15-27 ETH Registers Overview (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
RX_LENGTH_ERROR_FRAMES	Number of frames received with length error (Length type field ¼ frame size), for all frames with valid length field.	01C8H	U,PV	U,PV	Page 15-222
RX_OUT_OF_RANGE_TYPE_FRAMES	Number of frames received with length field not equal to the valid frame size (greater than 1.500 but less than 1.536).	01CCH	U,PV	U,PV	Page 15-223
RX_PAUSE_FRAMES	Number of good and valid PAUSE frames received.	01D0H	U,PV	U,PV	Page 15-224
RX_FIFO_OVERFLOW_FRAMES	Number of missed received frames due to FIFO overflow.	01D4H	U,PV	U,PV	Page 15-225
RX_VLAN_FRAMES_GOOD_BAD	Number of good and bad VLAN frames received.	01D8H	U,PV	U,PV	Page 15-226
RX_WATCHDOG_ERROR_FRAMES	Number of frames received with error due to watchdog timeout error (frames with a data load larger than 2.048 bytes).	01DCH	U,PV	U,PV	Page 15-227
RX_RECEIVE_ERROR_FRAMES	Number of frames received with error because of the MII RXER error.	01E0H	U,PV	U,PV	Page 15-228
RX_CONTROL_FRAMES_GOOD	Number of god control frames received.	01E4H	U,PV	U,PV	Page 15-229
Reserved		01E8H – 01FCH	nBE	nBE	

Table 15-27 ETH Registers Overview (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
MMC_IPC_RECEIVE_INTERRUPT_MASK	MMC IPC Receive Checksum Offload Interrupt Mask maintains the mask for the interrupt generated from the receive IPC statistic counters.	0200H	U,PV	U,PV	Page 15-230
Reserved		0204H	nBE	nBE	
MMC_IPC_RECEIVE_INTERRUPT	MMC Receive Checksum Offload Interrupt maintains the interrupt that the receive IPC statistic counters generate.	0208H	U,PV	U,PV	Page 15-235
Reserved		020CH	nBE	nBE	
RXIPV4_GOOD_FRAMES	Number of good IPv4 datagrams received with the TCP, UDP, or ICMP payload	0210H	U,PV	U,PV	Page 15-240
RXIPV4_HEADER_ERROR_FRAMES	Number of IPv4 datagrams received with header (checksum, length, or version mismatch) errors	0214H	U,PV	U,PV	Page 15-241
RXIPV4_NO_PAYLOAD_FRAMES	Number of IPv4 datagram frames received that did not have a TCP, UDP, or ICMP payload processed by the Checksum engine	0218H	U,PV	U,PV	Page 15-242
RXIPV4_FRAGMENTED_FRAMES	Number of good IPv4 datagrams with fragmentation	021CH	U,PV	U,PV	Page 15-243

Table 15-27 ETH Registers Overview (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
RXIPV4_UDP_CHECKSUM_DISABLED_FRAMES	Number of good IPv4 datagrams received that had a UDP payload with checksum disabled	0220H	U,PV	U,PV	Page 15-244
RXIPV6_GOOD_FRAMES	Number of good IPv6 datagrams received with TCP, UDP, or ICMP payloads	0224H	U,PV	U,PV	Page 15-245
RXIPV6_HEADER_ERROR_FRAMES	Number of IPv6 datagrams received with header errors (length or version mismatch)	0228H	U,PV	U,PV	Page 15-246
RXIPV6_NO_PAYLOAD_FRAMES	Number of IPv6 datagram frames received that did not have a TCP, UDP, or ICMP payload. This includes all IPv6 datagrams with fragmentation or security extension headers	022CH	U,PV	U,PV	Page 15-247
RXUDP_GOOD_FRAMES	Number of good IP datagrams with a good UDP payload. This counter is not updated when the RXIPV4_UDP_CHECKSUM_DISABLED_FRAMES counter is incremented.	0230H	U,PV	U,PV	Page 15-248
RXUDP_ERROR_FRAMES	Number of good IP datagrams whose UDP payload has a checksum error	0234H	U,PV	U,PV	Page 15-249

Table 15-27 ETH Registers Overview (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
RXTCP_GOOD_FRAMES	Number of good IP datagrams with a good TCP payload	0238H	U,PV	U,PV	Page 15-250
RXTCP_ERROR_FRAMES	Number of good IP datagrams whose TCP payload has a checksum error	023CH	U,PV	U,PV	Page 15-251
RXICMP_GOOD_FRAMES	Number of good IP datagrams with a good ICMP payload	0240H	U,PV	U,PV	Page 15-252
RXICMP_ERROR_FRAMES	Number of good IP datagrams whose ICMP payload has a checksum error	0244H	U,PV	U,PV	Page 15-253
Reserved		0248H – 024CH	nBE	nBE	
RXIPV4_GOOD_OCTETS	Number of bytes received in good IPv4 datagrams encapsulating TCP, UDP, or ICMP data. (Ethernet header, FCS, pad, or IP pad bytes are not included in this counter or in the octet counters listed below).	0250H	U,PV	U,PV	Page 15-254
RXIPV4_HEADER_ERROR_OCTETS	Number of bytes received in IPv4 datagrams with header errors (checksum, length, version mismatch). The value in the Length field of IPv4 header is used to update this counter.	0254H	U,PV	U,PV	Page 15-260

Table 15-27 ETH Registers Overview (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
RXIPV4_NO_PAYLOAD_OCTETS	Number of bytes received in IPv4 datagrams that did not have a TCP, UDP, or ICMP payload. The value in the IPv4 header's Length field is used to update this counter.	0258H	U,PV	U,PV	Page 15-256
RXIPV4_FRAGMENTED_OCTETS	Number of bytes received in fragmented IPv4 datagrams. The value in the IPv4 header's Length field is used to update this counter.	025CH	U,PV	U,PV	Page 15-257
RXIPV4_UDP_CHECKSUM_DISABLE_OCTETS	Number of bytes received in a UDP segment that had the UDP checksum disabled. This counter does not count IP Header bytes.	0260H	U,PV	U,PV	Page 15-258
RXIPV6_GOOD_OCTETS	Number of bytes received in good IPv6 datagrams encapsulating TCP, UDP or ICMPv6 data	0264H	U,PV	U,PV	Page 15-259
RXIPV6_HEADER_ERROR_OCTETS	Number of bytes received in IPv6 datagrams with header errors (length, version mismatch). The value in the IPv6 header's Length field is used to update this counter.	0268H	U,PV	U,PV	Page 15-260

Table 15-27 ETH Registers Overview (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
RXIPV6_NO_PAYLOAD_OCTETS	Number of bytes received in IPv6 datagrams that did not have a TCP, UDP, or ICMP payload. The value in the IPv6 header's Length field is used to update this counter.	026CH	U,PV	U,PV	Page 15-261
RXUDP_GOOD_OCTETS	Number of bytes received in a good UDP segment. This counter (and the counters below) does not count IP header bytes.	0270H	U,PV	U,PV	Page 15-262
RXUDP_ERROR_OCTETS	Number of bytes received in a UDP segment that had checksum errors	0274H	U,PV	U,PV	Page 15-263
RXTCP_GOOD_OCTETS	Number of bytes received in a good TCP segment	0278H	U,PV	U,PV	Page 15-264
RXTCP_ERROR_OCTETS	Number of bytes received in a TCP segment with checksum errors	027CH	U,PV	U,PV	Page 15-265
RXICMP_GOOD_OCTETS	Number of bytes received in a good ICMP segment	0280H	U,PV	U,PV	Page 15-266
RXICMP_ERROR_OCTETS	Number of bytes received in an ICMP segment with checksum errors	0284H	U,PV	U,PV	Page 15-267
Reserved		0288H – 02FCH	nBE	nBE	

Table 15-27 ETH Registers Overview (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
Do not use	Do not use	0300_H - $06FC_H$	nBE	nBE	Do not use
System Time Registers					
TIMESTAMP_CONTROL.	Timestamp Controlic Register	0700_H	U,PV	U,PV	Page 15-268
SUB_SECOND_INCREMENT	Sub Second Increment Register	0704_H	U,PV	U,PV	Page 15-272
SYSTEM_TIME_SECONDS	System Time Seconds Register	0708_H	U,PV	U,PV	Page 15-273
SYSTEM_TIME_NANOSECONDS	System Time Nanoseconds Register	$070C_H$	U,PV	U,PV	Page 15-274
SYSTEM_TIME_SECONDS_UPDATE	System Time Seconds Update Register	0710_H	U,PV	U,PV	Page 15-275
SYSTEM_TIME_NANOSECONDS_UPDATE	System Time Nanoseconds Update Register	0714_H	U,PV	U,PV	Page 15-276
TIMESTAMP_ADDEND	Timestamp Addend Register	0718_H	U,PV	U,PV	Page 15-277
TARGET_TIME_SECONDS	Target Time Seconds Register	$071C_H$	U,PV	U,PV	Page 15-278
TARGET_TIME_NANOSECONDS	Target Time Nanoseconds Register	0720_H	U,PV	U,PV	Page 15-279
SYSTEM_TIME_HIGHER_WORD_SECONDS	System Time Higher Word Seconds Register	0724_H	U,PV	U,PV	Page 15-281
TIMESTAMP_STATUS	Timestamp Status Register	0728_H	U,PV	U,PV	Page 15-282
PPS_CONTROL	PPS Control Register	$072C_H$	U,PV	U,PV	Page 15-285

Table 15-27 ETH Registers Overview (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
Do not use	Do not use	0738_H - $07FC_H$	nBE	nBE	Do not use
Do not use	Do not use	0738_H - $07FC_H$	nBE	nBE	Do not use
DMA Registers					
BUS_MODE	BUS Mode Register	1000_H	U,PV	U,PV	Page 15-289
TRANSMIT_POLL_DEMAND	Transmit Poll Demand Register	1004_H	U,PV	U,PV	Page 15-294
RECEIVE_POLL_DEMAND	Receive Poll Demand Register	1008_H	U,PV	U,PV	Page 15-295
RECEIVE_DESCRIPTOR_LIST_ADDRESS	Receive Descriptor List Address Register	$100C_H$	U,PV	U,PV	Page 15-296
TRANSMIT_DESCRIPTOR_LIST_ADDRESS	Transmit Descriptor List Address Register	1010_H	U,PV	U,PV	Page 15-297
STATUS	Status Register	1014_H	U,PV	U,PV	Page 15-298
OPERATION_MODE	Operation Mode Register	1018_H	U,PV	U,PV	Page 15-304
INTERRUPT_ENABLE	Interrupt Enable Register	Register101C H	U,PV	U,PV	Page 15-310
MISSED_FRAME_AND_BUFFER_OVERFLOW_COUNTER	Missed Frame And_Buffer Overflow Counter Register	1020_H	U,PV	U,PV	Page 15-313
RECEIVE_INTERRUPT_WATCHDOG_TIMER	Receive Interrupt Watchdog Timer Register	1024_H	U,PV	U,PV	Page 15-314
		1028_H	U,PV	U,PV	
Do not use	Do not use	$102C_H$	nBE	nBE	Do not use
Do not use	Do not use	1030_H-1044_H	nBE	nBE	Do not use

Table 15-27 ETH Registers Overview (cont'd)

Short Name	Description	Offset Addr.[1]	Access Mode		Description See
			Read	Write	
CURRENT_HOST_TRANSMIT_DESCRIPTOR	Current Host Transmit Descriptor Register	1048_H	U,PV	U,PV	Page 15-316
CURRENT_HOST_RECEIVE_DESCRIPTOR	Current Host Receive Descriptor Register	$104C_H$	U,PV	U,PV	Page 15-317
CURRENT_HOST_TRANSMIT_BUFFER_ADDRESS	Current Host Transmit Buffer Address Register	1050_H	U,PV	U,PV	Page 15-318
CURRENT_HOST_RECEIVE_BUFFER_ADDRESS	Current Host Receive Buffer Address Register	1054_H	U,PV	U,PV	Page 15-319
HW_FEATURE	HW Feature Register	1058_H	U,PV	NC	Page 15-320

1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

15.6.2.1 Registers Description

MAC_CONFIGURATION

The MAC Configuration register establishes receive and transmit operating modes.

ETH0_MAC_CONFIGURATION
MAC Configuration Register **(0_H)** **Reset Value: 0000 8000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved_31	SARC			2KPE	Reserved_26	CST	TC	WD	JD	BE	JE	IFG			DCRS
r	r			rw	r	rw	r	rw	rw	r	rw	rw			rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	FES	DO	LM	DM	IPC	DR	Reserved_8	ACS	BL		DC	TE	RE	PRELEN	
r	rw	rw	rw	rw	rw	rw	r	rw	rw		rw	rw	rw	rw	

Field	Bits	Type	Description
PRELEN	[1:0]	rw	**Preamble Length for Transmit Frames** These bits control the number of preamble bytes that are added to the beginning of every Transmit frame. The preamble reduction occurs only when the MAC is operating in the full-duplex mode. * 00_B: 7 bytes of preamble * 01_B: 5 byte of preamble * 10_B: 3 bytes of preamble * 11_B: reserved
RE	2	rw	**Receiver Enable** When this bit is set, the receiver state machine of the MAC is enabled for receiving frames from the MII. When this bit is reset, the MAC receive state machine is disabled after the completion of the reception of the current frame, and does not receive any further frames from the MII.

Field	Bits	Type	Description
TE	3	rw	**Transmitter Enable** When this bit is set, the transmit state machine of the MAC is enabled for transmission on the MII. When this bit is reset, the MAC transmit state machine is disabled after the completion of the transmission of the current frame, and does not transmit any further frames.
DC	4	rw	**Deferral Check** When this bit is set, the deferral check function is enabled in the MAC. The MAC issues a Frame Abort status, along with the excessive deferral error bit set in the transmit frame status, when the transmit state machine is deferred for more than 24,288 bit times . If the Jumbo frame mode is enabled in the 10 or 100 Mbps mode, the threshold for deferral is 155,680 bits times. Deferral begins when the transmitter is ready to transmit, but is prevented because of an active carrier sense signal (CRS) on the MII. Defer time is not cumulative. When the transmitter defers for 10,000 bit times, it transmits, collides, backs off, and then defers again after completion of back-off. The deferral timer resets to 0 and restarts. When this bit is reset, the deferral check function is disabled and the MAC defers until the CRS signal goes inactive. This bit is applicable only in the half-duplex mode and is reserved (RO) in the full-duplex-only configuration.

Field	Bits	Type	Description
BL	[6:5]	rw	**Back-Off Limit** The Back-Off limit determines the random integer number (r) of slot time delays (512 bit times for 10/100 Mbps) for which the MAC waits before rescheduling a transmission attempt during retries after a collision. This bit is applicable only in the half-duplex mode and is reserved (RO) in the full-duplex-only configuration. * 00_B: k = min (n, 10) * 01_B: k = min (n, 8) * 10_B: k = min (n, 4) * 11_B: k = min (n, 1) where n = retransmission attempt. The random integer r takes the value in the range 0 <= r < kth power of 2
ACS	7	rw	**Automatic Pad or CRC Stripping** When this bit is set, the MAC strips the Pad or FCS field on the incoming frames only if the value of the length field is less than 1,536 bytes. All received frames with length field greater than or equal to 1,536 bytes are passed to the application without stripping the Pad or FCS field. When this bit is reset, the MAC passes all incoming frames, without modifying them, to the XMC4500 Memory.
Reserved_8	8	r	**Reserved**
DR	9	rw	**Disable Retry** When this bit is set, the MAC attempts only one transmission. When a collision occurs on the MII interface, the MAC ignores the current frame transmission and reports a Frame Abort with excessive collision error in the transmit frame status. When this bit is reset, the MAC attempts retries based on the settings of the BL field (Bits [6:5]). This bit is applicable only in the half-duplex mode and is reserved (RO with default value) in the full-duplex-only configuration.

Field	Bits	Type	Description
IPC	10	rw	**Checksum Offload** When this bit is set, the MAC calculates the 16-bit ones complement of the ones complement sum of all received Ethernet frame payloads. It also checks whether the IPv4 Header checksum (assumed to be bytes 2526 or 2930 (VLAN-tagged) of the received Ethernet frame) is correct for the received frame and gives the status in the receive status word. The MAC also appends the 16-bit checksum calculated for the IP header datagram payload (bytes after the IPv4 header) and appends it to the Ethernet frame transferred to the application (when Type 2 COE is deselected). When this bit is reset, this function is disabled. When Type 2 COE is selected, this bit, when set, enables the IPv4 header checksum checking and IPv4 or IPv6 TCP, UDP, or ICMP payload checksum checking. When this bit is reset, the COE function in the receiver is disabled and the corresponding PCE and IP HCE status bits are always cleared.
DM	11	rw	**Duplex Mode** When this bit is set, the MAC operates in the full-duplex mode where it can transmit and receive simultaneously.
LM	12	rw	**Loopback Mode** When this bit is set, the MAC operates in the loopback mode using the MII. The MII Receive clock input is required for the loopback to work properly, because the Transmit clock is not looped-back internally.
DO	13	rw	**Disable Receive Own** When this bit is set, the MAC disables the reception of frames in the half-duplex mode. When this bit is reset, the MAC receives all packets that are given by the PHY while transmitting. This bit is not applicable if the MAC is operating in the full-duplex mode.
FES	14	rw	**Speed** This bit selects the speed in the MII or RMII: * 0_B: 10 Mbps * 1_B: 100 Mbps This bit generates link speed encoding when TC (Bit 24) is set in the RMII mode.

Field	Bits	Type	Description
Reserved	15	r	**Reserved**
DCRS	16	rw	**Disable Carrier Sense During Transmission** When set high, this bit makes the MAC transmitter ignore the MII CRS signal during frame transmission in the half-duplex mode. This request results in no errors generated because of Loss of Carrier or No Carrier during such transmission. When this bit is low, the MAC transmitter generates such errors because of Carrier Sense and can even abort the transmissions.
IFG	[19:17]	rw	**Inter-Frame Gap** These bits control the minimum IFG between frames during transmission. * 000_B: 96 bit times * 001_B: 88 bit times * 010_B: 80 bit times * ... * 111_B: 40 bit times In the half-duplex mode, the minimum IFG can be configured only for 64 bit times (IFG = 100_B). Lower values are not considered.
JE	20	rw	**Jumbo Frame Enable** When this bit is set, the MAC allows Jumbo frames of 9,018 bytes (9,022 bytes for VLAN tagged frames) without reporting a giant frame error in the receive frame status.
BE	21	r	**Frame Burst Enable** When this bit is set, the MAC allows frame bursting during transmission in the MII half-duplex mode. This bit is reserved (and RO) in the 10/100 Mbps only or full-duplex-only configurations.
JD	22	rw	**Jabber Disable** When this bit is set, the MAC disables the jabber timer on the transmitter. The MAC can transfer frames of up to 16,384 bytes. When this bit is reset, the MAC cuts off the transmitter if the application sends out more than 2,048 bytes of data (10,240 if JE is set high) during transmission.

Field	Bits	Type	Description
WD	23	rw	**Watchdog Disable** When this bit is set, the MAC disables the watchdog timer on the receiver. The MAC can receive frames of up to 16,384 bytes. When this bit is reset, the MAC does not allow more than 2,048 bytes (10,240 if JE is set high) of the frame being received. The MAC cuts off any bytes received after 2,048 bytes.
TC	24	r	**Transmit Configuration in RMII** When set, this bit enables the transmission of duplex mode, link speed, and link up or down information to the PHY in RMII. When this bit is reset, no such information is driven to the PHY.
CST	25	rw	**CRC Stripping of Type Frames** When set, the last 4 bytes (FCS) of all frames of Ether type (type field greater than 0600_H) are stripped and dropped before forwarding the frame to the application. This function is not valid when the IP Checksum Engine (Type 1) is enabled in the MAC receiver.
Reserved_26	26	r	**Reserved**
2KPE	27	rw	**IEEE 802.3as support for 2K packets Enable** When set, the MAC considers all frames, with up to 2,000 bytes length, as normal packets. When Bit 20 (Jumbo Enable) is not set, the MAC considers all received frames of size more than 2K bytes as Giant frames. When this bit is reset and Bit 20 (Jumbo Enable) is not set, the MAC considers all received frames of size more than 1,518 bytes (1,522 bytes for tagged) as Giant frames. When Bit 20 (Jumbo Enable) is set, setting this bit has no effect on Giant Frame status.

Field	Bits	Type	Description
SARC	[30:28]	r	**Source Address Insertion or Replacement Control** This field controls the source address insertion or replacement for all transmitted frames. Bit 30 specifies which MAC Address register (0 or 1) is used for source address insertion or replacement based on the values of Bits [29:28]: * 10_B: - If Bit 30 is set to 0, the MAC inserts the content of the MAC Address 0 registers (**ETH0_MAC_ADDRESS0_HIGH** and **ETH0_MAC_ADDRESS0_LOW**) in the SA field of all transmitted frames. - If Bit 30 is set to 1 the MAC inserts the content of the MAC Address 1 registers (**ETH0_MAC_ADDRESS1_HIGH** and **ETH0_MAC_ADDRESS1_LOW**) in the SA field of all transmitted frames. * 11_B: - If Bit 30 is set to 0, the MAC replaces the content of the MAC Address 0 registers (**ETH0_MAC_ADDRESS0_HIGH** and **ETH0_MAC_ADDRESS0_LOW**) in the SA field of all transmitted frames. - If Bit 30 is set to 1 and the Enable MAC Address Register 1 option is selected during core configuration, the MAC replaces the content of the MAC Address 1 registers (**ETH0_MAC_ADDRESS1_HIGH** and **ETH0_MAC_ADDRESS1_LOW**) in the SA field of all transmitted frames. Note: - Changes to this field take effect only on the start of a frame. If you write this register field when a frame is being transmitted, only the subsequent frame can use the updated value, that is, the current frame does not use the updated value.
Reserved_31	31	r	**Reserved**

MAC_FRAME_FILTER

The MAC Frame Filter register contains the filter controls for receiving frames. Some of the controls from this register go to the address check block of the MAC, which performs the first level of address filtering. The second level of filtering is performed on the incoming frame, based on other controls such as Pass Bad Frames and Pass Control Frames.

ETH0_MAC_FRAME_FILTER
MAC Frame Filter **(4_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RA	Reserved_30_22									DNTU	IPFE	Reserved_19_17			VTFE
rw	r									r	r	r			rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved_15_11					HPF	SAF	SAIF	PCF		DBF	PM	DAIF	HMC	HUC	PR
r					rw	rw	rw	rw		rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
PR	0	rw	**Promiscuous Mode** When this bit is set, the Address Filter module passes all incoming frames regardless of its destination or source address. The SA or DA Filter Fails status bits of the Receive Status Word are always cleared when PR is set.
HUC	1	rw	**Hash Unicast** When set, MAC performs destination address filtering of unicast frames according to the hash table. When reset, the MAC performs a perfect destination address filtering for unicast frames, that is, it compares the DA field with the values programmed in DA registers. If Hash Filter is not selected during core configuration, this bit is reserved (and RO).

Field	Bits	Type	Description
HMC	2	rw	**Hash Multicast** When set, MAC performs destination address filtering of received multicast frames according to the hash table. When reset, the MAC performs a perfect destination address filtering for multicast frames, that is, it compares the DA field with the values programmed in DA registers. If Hash Filter is not selected during core configuration, this bit is reserved (and RO).
DAIF	3	rw	**DA Inverse Filtering** When this bit is set, the Address Check block operates in inverse filtering mode for the DA address comparison for both unicast and multicast frames. When reset, normal filtering of frames is performed.
PM	4	rw	**Pass All Multicast** When set, this bit indicates that all received frames with a multicast destination address (first bit in the destination address field is '1') are passed. When reset, filtering of multicast frame depends on HMC bit.
DBF	5	rw	**Disable Broadcast Frames** When this bit is set, the AFM module filters all incoming broadcast frames. In addition, it overrides all other filter settings. When this bit is reset, the AFM module passes all received broadcast frames.

Field	Bits	Type	Description
PCF	[7:6]	rw	**Pass Control Frames** These bits control the forwarding of all control frames (including unicast and multicast PAUSE frames). * 00B: MAC filters all control frames from reaching the application. * 01B: MAC forwards all control frames except PAUSE control frames to application even if they fail the Address filter. * 10B: MAC forwards all control frames to application even if they fail the Address Filter. * 11B: MAC forwards control frames that pass the Address Filter. The following conditions should be true for the PAUSE control frames processing: * Condition 1: The MAC is in the full-duplex mode and flow control is enabled by setting FLOW_CONTROL.RFE. * Condition 2: The destination address (DA) of the received frame matches the special multicast address or the MAC Address 0 when FLOW_CONTROL.UP is set. * Condition 3: The Type field of the received frame is 8808H and the OPCODE field is 0001H. Note: This field should be set to 01 only when the Condition 1 is true, that is, the MAC is programmed to operate in the full-duplex mode and the RFE bit is enabled. Otherwise, the PAUSE frame filtering may be inconsistent. When Condition 1 is false, the PAUSE frames are considered as generic control frames. Therefore, to pass all control frames (including PAUSE control frames) when the full-duplex mode and flow control is not enabled, you should set the PCF field to 10 or 11 (as required by the application).
SAIF	8	rw	**SA Inverse Filtering** When this bit is set, the Address Check block operates in inverse filtering mode for the SA address comparison. The frames whose SA matches the SA registers are marked as failing the SA Address filter. When this bit is reset, frames whose SA does not match the SA registers are marked as failing the SA Address filter.

Field	Bits	Type	Description
SAF	9	rw	**Source Address Filter Enable** When this bit is set, the MAC compares the SA field of the received frames with the values programmed in the enabled SA registers. If the comparison matches, then the SA Match bit of RxStatus Word is set high. When this bit is set high and the SA filter fails, the MAC drops the frame. When this bit is reset, the MAC forwards the received frame to the application and with the updated SA Match bit of the RxStatus depending on the SA address comparison.
HPF	10	rw	**Hash or Perfect Filter** When this bit is set, it configures the address filter to pass a frame if it matches either the perfect filtering or the hash filtering as set by the HMC or HUC bits. When this bit is low and the HUC or HMC bit is set, the frame is passed only if it matches the Hash filter.
Reserved_15_11	[15:11]	r	**Reserved**
VTFE	16	rw	**VLAN Tag Filter Enable** When set, this bit enables the MAC to drop VLAN tagged frames that do not match the VLAN Tag comparison. When reset, the MAC forwards all frames irrespective of the match status of the VLAN Tag.
Reserved_19_17	[19:17]	r	**Reserved**
IPFE	20	r	**Layer 3 and Layer 4 Filter Enable** When set, this bit enables the MAC to drop frames that do not match the enabled Layer 3 and Layer 4 filters. If Layer 3 or Layer 4 filters are not enabled for matching, this bit does not have any effect. When reset, the MAC forwards all frames irrespective of the match status of the Layer 3 and Layer 4 fields. If the Layer 3 and Layer 4 Filtering feature is not selected during core configuration, this bit is reserved (RO with default value).

Field	Bits	Type	Description
DNTU	21	r	**Drop non-TCP/UDP over IP Frames** When set, this bit enables the MAC to drop the non-TCP or UDP over IP frames. The MAC forward only those frames that are processed by the Layer 4 filter. When reset, this bit enables the MAC to forward all non-TCP or UDP over IP frames.
Reserved_30_22	[30:22]	r	**Reserved**
RA	31	rw	**Receive All** When this bit is set, the MAC Receiver module passes all received frames, irrespective of whether they pass the address filter or not, to the Application. The result of the SA or DA filtering is updated (pass or fail) in the corresponding bits in the Receive Status Word. When this bit is reset, the Receiver module passes only those frames to the Application that pass the SA or DA address filter.

HASH_TABLE_HIGH

The 64-bit Hash table is used for group address filtering. For hash filtering, the contents of the destination address in the incoming frame is passed through the CRC logic, and the upper 6 bits of the CRC register are used to index the contents of the Hash table. The most significant bit determines the register to be used (Hash Table High or Hash Table Low), and the other 5 bits determine which bit within the register. A hash value of 00000B selects Bit 0 of the selected register, and a value of 11111B selects Bit 31 of the selected register. The hash value of the destination address is calculated in the following way:

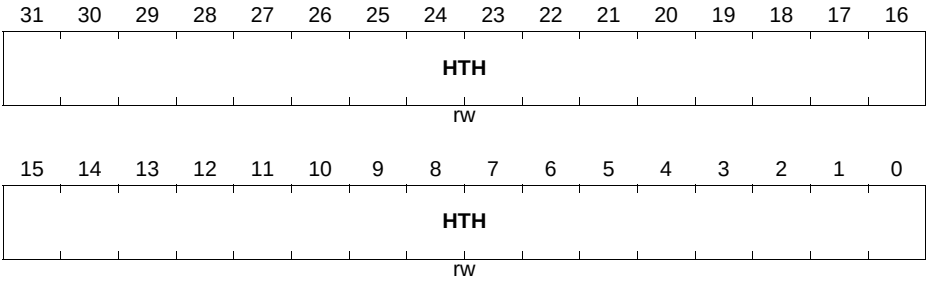
1. Calculate the 32-bit CRC for the DA (See IEEE 802.3, Section 3.2.8 for the steps to calculate CRC32).

2. Perform bitwise reversal for the value obtained in Step 1.

3. Take the upper 6 bits from the value obtained in Step 2.

For example, if the DA of the incoming frame is received as 1F52 419C B6AFH (1FH is the first byte received on MII interface), then the internally calculated 6-bit Hash value is 2CH and Bit 12 of Hash Table High register is checked for filtering. If the DA of the incoming frame is received as A00A 9800 0045H, then the calculated 6-bit Hash value is 07H and Bit 7 of Hash Table Low register is checked for filtering. Note: To help you program the hash table, a sample C routine that generates a DA's 6-bit hash is included in the /sample_codes/ directory of your workspace. If the corresponding bit value of the register is 1, the frame is accepted. Otherwise, it is rejected. If the PM (Pass All Multicast) bit is set in the MAC Frame Filter Register, then all multicast frames are accepted regardless of the multicast hash values. If the Hash Table register is configured to be double-synchronized to the MII clock domain, the synchronization is triggered only when Bits[31:24] (in little-endian mode) of the Hash Table High or Low registers are written. Consecutive writes to these register should be performed only after at least four clock cycles in the destination clock domain when double-synchronization is enabled. The Hash Table High register contains the higher 32 bits of the Hash table.

ETH0_HASH_TABLE_HIGH
Hash Table High Register **(8_H)** **Reset Value: 0000 0000$_H$**

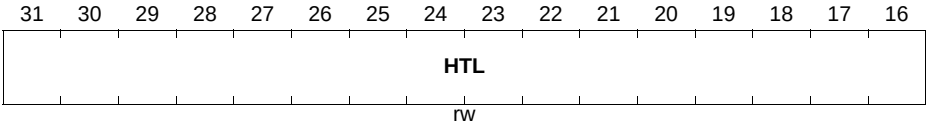


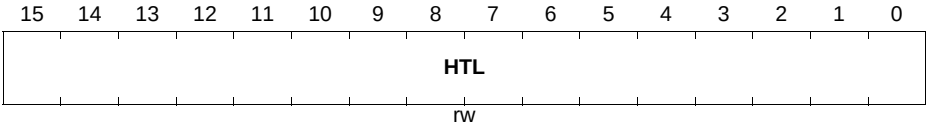
Field	Bits	Type	Description
HTH	[31:0]	rw	**Hash Table High** This field contains the upper 32 bits of the Hash table.

HASH_TABLE_LOW

The Hash Table Low register contains the lower 32 bits of the Hash table. Both Register 2 and Register 3 are reserved if the Hash Filter Function is disabled or the 128-bit or 256-bit Hash Table is selected during core configuration.

ETH0_HASH_TABLE_LOW
Hash Table Low Register **(C_H)** **Reset Value: 0000 0000$_H$**



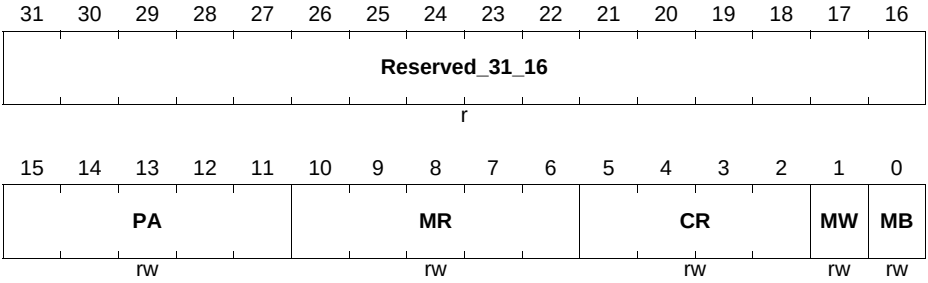


Field	Bits	Type	Description
HTL	[31:0]	rw	**Hash Table Low** This field contains the lower 32 bits of the Hash table.

GMII_ADDRESS

The MII Address register controls the management cycles to the external PHY through the management interface.

ETH0_GMII_ADDRESS
MII Address Register **(10_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
MB	0	rw	**MII Busy** This bit should read logic 0 before writing to the MII Address and Data registers. During a PHY register access, the software sets this bit to 1 to indicate that a Read or Write access is in progress. The MII Data Register is invalid until this bit is cleared by the MAC. Therefore the MII Data Register should be kept valid until the MAC clears this bit during a PHY Write operation. Similarly for a read operation, the contents of the MII Data Register are not valid until this bit is cleared. The subsequent read or write operation should happen only after the previous operation is complete. Because there is no acknowledgment from the PHY to MAC after a read or write operation is completed, there is no change in the functionality of this bit even when the PHY is not present.
MW	1	rw	**MII Write** When set, this bit indicates to the PHY that this is a Write operation using the MII Data register. If this bit is not set, it indicates that this is a Read operation, that is, placing the data in the MII Data register.

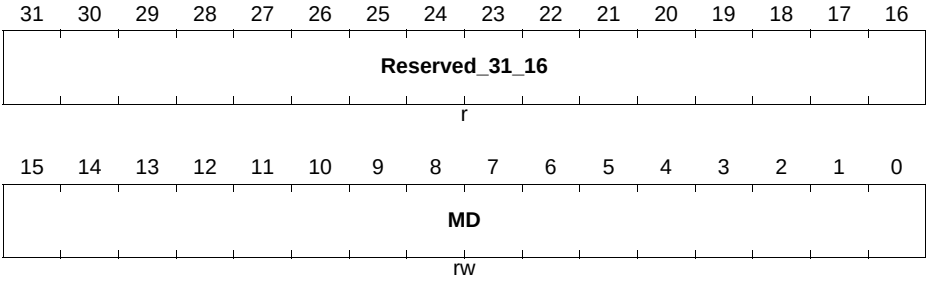
Field	Bits	Type	Description
CR	[5:2]	rw	**CSR Clock Range** The CSR Clock Range selection determines the frequency of the MDC clock according to the ETH Clock frequency used in your design. The suggested range of ETH Clock frequency applicable for each value (when Bit[5] = 0) ensures that the MDC clock is approximately between the frequency range 1.0 MHz - 2.5 MHz. - 0000B: The frequency of the ETH Clock is 60-100 MHz and the MDC clock is ETH Clock /42. - 0001B: The frequency of the ETH Clock is 100-150 MHz and the MDC clock is ETH Clock /62. - 0010B: The frequency of the ETH Clock is 20-35 MHz and the MDC clock is ETH Clock /16. - 0011B: The frequency of the ETH Clock is 35-60 MHz and the MDC clock is ETH Clock /26. - 0100B: The frequency of the ETH Clock is 150-250 MHz and the MDC clock is ETH Clock /102. - 0100B: The frequency of the ETH Clock is 250-300 MHz and the MDC clock is ETH Clock /124. - 0110B and 0111B: Reserved When Bit 5 is set, you can achieve MDC clock of frequency higher than the IEEE 802.3 specified frequency limit of 2.5 MHz and program a clock divider of lower value. For example, when the ETH Clock is of 100 MHz frequency and you program these bits as 1010B, then the resultant MDC clock is of 12.5 MHz which is outside the limit of IEEE 802.3 specified range. Program the following values only if the interfacing chips support faster MDC clocks: - 1000: ETH Clock /4 - 1001: ETH Clock /6 - 1010: ETH Clock /8 - 1011: ETH Clock /10 - 1100: ETH Clock /12 - 1101: ETH Clock /14 - 1110: ETH Clock /16 - 1111:ETH Clock /18
MR	[10:6]	rw	**MII Register** These bits select the desired MII register in the selected PHY device.

Field	Bits	Type	Description
PA	[15:11]	rw	**Physical Layer Address** This field indicates which of the 32 possible PHY devices are being accessed.
Reserved_31_16	[31:16]	r	**Reserved**

GMII_DATA

The MII Data register stores Write data to be written to the PHY register located at the address specified in the MII Address Register. This register also stores the Read data from the PHY register located at the address specified by the MII Address Register.

ETH0_GMII_DATA
MII Data Register **(14_H)** **Reset Value: 0000 0000$_H$**

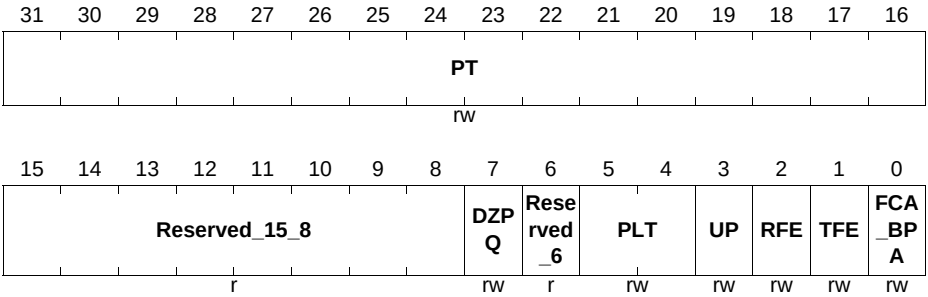


Field	Bits	Type	Description
MD	[15:0]	rw	**MII Data** This field contains the 16-bit data value read from the PHY or RevMII after a Management Read operation or the 16-bit data value to be written to the PHY before a Management Write operation.
Reserved_31_16	[31:16]	r	**Reserved**

FLOW_CONTROL

The Flow Control register controls the generation and reception of the Control (Pause Command) frames by the MAC's Flow control module. A Write to a register with the Busy bit set to '1' triggers the Flow Control block to generate a Pause Control frame. The fields of the control frame are selected as specified in the 802.3x specification, and the Pause Time value from this register is used in the Pause Time field of the control frame. The Busy bit remains set until the control frame is transferred onto the cable. The CPU must make sure that the Busy bit is cleared before writing to the register.

ETH0_FLOW_CONTROL
Flow Control Register **(18_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
FCA_BPA	0	rw	**Flow Control Busy or Backpressure Activate** This bit initiates a Pause Control frame in the full-duplex mode and activates the backpressure function in the half-duplex mode if the TFE bit is set. In the full-duplex mode, this bit should be read as 0 before writing to the Flow Control register. To initiate a Pause control frame, the Application must set this bit to 1. During a transfer of the Control Frame, this bit continues to be set to signify that a frame transmission is in progress. After the completion of Pause control frame transmission, the MAC resets this bit to 1'b0. The Flow Control register should not be written to until this bit is cleared. In the half-duplex mode, when this bit is set (and TFE is set), then backpressure is asserted by the MAC. During backpressure, when the MAC receives a new frame, the transmitter starts sending a JAM pattern resulting in a collision.When the MAC is configured for the full-duplex mode, the BPA is automatically disabled.
TFE	1	rw	**Transmit Flow Control Enable** In the full-duplex mode, when this bit is set, the MAC enables the flow control operation to transmit Pause frames. When this bit is reset, the flow control operation in the MAC is disabled, and the MAC does not transmit any Pause frames. In half-duplex mode, when this bit is set, the MAC enables the back-pressure operation. When this bit is reset, the back-pressure feature is disabled.
RFE	2	rw	**Receive Flow Control Enable** When this bit is set, the MAC decodes the received Pause frame and disables its transmitter for a specified (Pause) time. When this bit is reset, the decode function of the Pause frame is disabled.

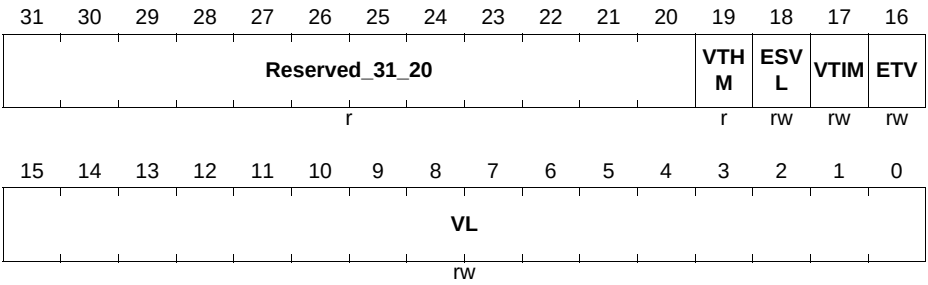
Field	Bits	Type	Description
UP	3	rw	**Unicast Pause Frame Detect** When this bit is set, then in addition to the detecting Pause frames with the unique multicast address, the MAC detects the Pause frames with the station's unicast address specified in the MAC Address0 High Register and MAC Address0 Low Register. When this bit is reset, the MAC detects only a Pause frame with the unique multicast address specified in the 802.3x standard.
PLT	[5:4]	rw	**Pause Low Threshold** This field configures the threshold of the PAUSE timer at which the input flow control signal is checked for automatic retransmission of PAUSE Frame. The threshold values should be always less than the Pause Time configured in Bits[31:16]. For example, if PT = 100H (256 slot-times), and PLT = 01, then a second PAUSE frame is automatically transmitted if the flow control signal is asserted at 228 (256 - 28) slot times after the first PAUSE frame is transmitted. The following list provides the threshold values for different values: - 00B: The threshold is Pause time minus 4 slot times (PT - 4 slot times). - 01B: The threshold is Pause time minus 28 slot times (PT - 28 slot times). - 10B: The threshold is Pause time minus 144 slot times (PT - 144 slot times). - 11B: The threshold is Pause time minus 256 slot times (PT - 256 slot times). The slot time is defined as the time taken to transmit 512 bits (64 bytes) on the MII interface.
Reserved_6	6	r	**Reserved**
DZPQ	7	rw	**Disable Zero-Quanta Pause** When this bit is set, it disables the automatic generation of the Zero-Quanta Pause Control frames on the de-assertion of the flow-control signal from the FIFO layer When this bit is reset, normal operation with automatic Zero-Quanta Pause Control frame generation is enabled.

Field	Bits	Type	Description
Reserved_15_8	[15:8]	r	**Reserved**
PT	[31:16]	rw	**Pause Time** This field holds the value to be used in the Pause Time field in the transmit control frame. If the Pause Time bits is configured to be double-synchronized to the MII clock domain, then consecutive writes to this register should be performed only after at least four clock cycles in the destination clock domain.

VLAN_TAG

The VLAN Tag register contains the IEEE 802.1Q VLAN Tag to identify the VLAN frames. The MAC compares the 13th and 14th bytes of the receiving frame (Length/Type) with 8100H, and the following two bytes are compared with the VLAN tag. If a match occurs, the MAC sets the received VLAN bit in the receive frame status. The legal length of the frame is increased from 1,518 bytes to 1,522 bytes. If the VLAN Tag register is configured to be double-synchronized to the MII clock domain, then consecutive writes to these register should be performed only after at least four clock cycles in the destination clock domain.

ETH0_VLAN_TAG
VLAN Tag Register **($1C_H$)** **Reset Value: $0000\ 0000_H$**



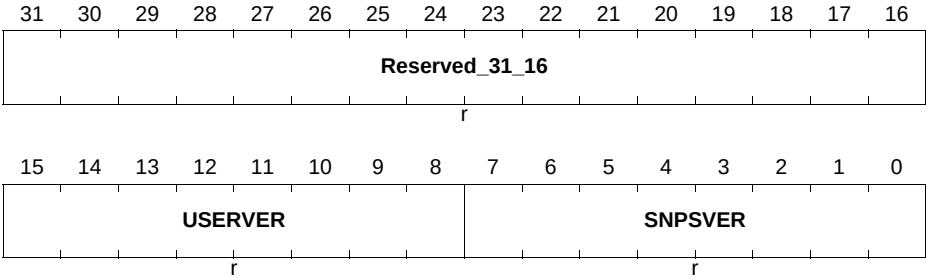
Field	Bits	Type	Description
VL	[15:0]	rw	**VLAN Tag Identifier for Receive Frames** This field contains the 802.1Q VLAN tag to identify the VLAN frames and is compared to the 15th and 16th bytes of the frames being received for VLAN frames. The following list describes the bits of this field: * Bits [15:13]: User Priority * Bit 12: Canonical Format Indicator (CFI) or Drop Eligible Indicator (DEI) * Bits[11:0]: VLAN tag's VLAN Identifier (VID) field When the ETV bit is set, only the VID (Bits[11:0]) is used for comparison. If VL (VL[11:0] if ETV is set) is all zeros, the MAC does not check the fifteenth and 16th bytes for VLAN tag comparison, and declares all frames with a Type field value of 8100H or 88A8H as VLAN frames.

Field	Bits	Type	Description
ETV	16	rw	**Enable 12-Bit VLAN Tag Comparison** When this bit is set, a 12-bit VLAN identifier is used for comparing and filtering instead of the complete 16-bit VLAN tag. Bits 11-0 of VLAN tag are compared with the corresponding field in the received VLAN-tagged frame. Similarly, when enabled, only 12 bits of the VLAN tag in the received frame are used for hash-based VLAN filtering. When this bit is reset, all 16 bits of the 15th and 16th bytes of the received VLAN frame are used for comparison and VLAN hash filtering.
VTIM	17	rw	**VLAN Tag Inverse Match Enable** When set, this bit enables the VLAN Tag inverse matching. The frames that do not have matching VLAN Tag are marked as matched. When reset, this bit enables the VLAN Tag perfect matching. The frames with matched VLAN Tag are marked as matched.
ESVL	18	rw	**Enable S-VLAN** When this bit is set, the MAC transmitter and receiver also consider the S-VLAN (Type = 88A8H) frames as valid VLAN tagged frames.
VTHM	19	r	**VLAN Tag Hash Table Match Enable** When set, the most significant four bits of the VLAN tags CRC are used to index the content of Register 354 [VLAN Hash Table Register]. A value of 1 in the VLAN Hash Table register, corresponding to the index, indicates that the frame matched the VLAN hash table. When Bit 16 (ETV) is set, the CRC of the 12-bit VLAN Identifier (VID) is used for comparison whereas when ETV is reset, the CRC of the 16-bit VLAN tag is used for comparison. When reset, the VLAN Hash Match operation is not performed.
Reserved_31_20	[31:20]	r	**Reserved**

VERSION

The VERSION registers identifies the version of the ETH. This register contains two bytes: one that Synopsys uses to identify the core release number, and the other that you set during core configuration.

ETH0_VERSION
Version Register **(20_H)** **Reset Value: 0000 1037$_H$**

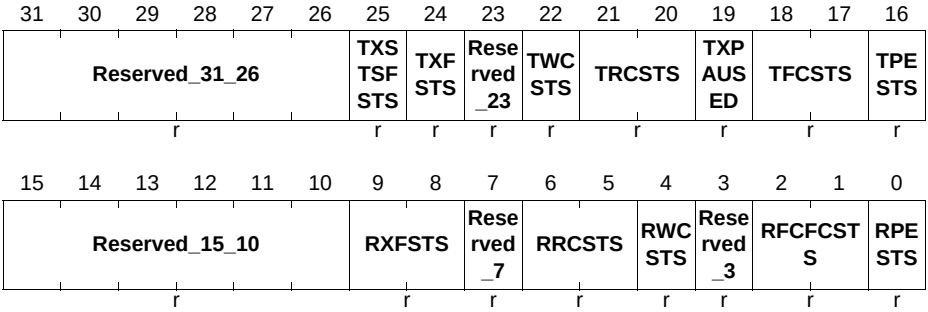


Field	Bits	Type	Description
SNPSVER	[7:0]	r	**Synopsys-defined Version (3.7)**
USERVER	[15:8]	r	**User-defined Version (Configured with the coreConsultant)**
Reserved_31_16	[31:16]	r	**Reserved**

DEBUG

The DEBUG register gives the status of all main modules of the transmit and receive data-paths and the FIFOs. An all-zero status indicates that the MAC is in idle state (and FIFOs are empty) and no activity is going on in the data-paths.

ETH0_DEBUG
Debug Register **(24_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
RPESTS	0	r	**MAC MII Receive Protocol Engine Status** When high, this bit indicates that the MAC MII receive protocol engine is actively receiving data and not in IDLE state.
RFCFCSTS	[2:1]	r	**MAC Receive Frame Controller FIFO Status** When high, this field indicates the active state of the small FIFO Read and Write controllers of the MAC Receive Frame Controller Module.
Reserved_3	3	r	**Reserved**
RWCSTS	4	r	**MTL Rx FIFO Write Controller Active Status** When high, this bit indicates that the MTL Rx FIFO Write Controller is active and is transferring a received frame to the FIFO.

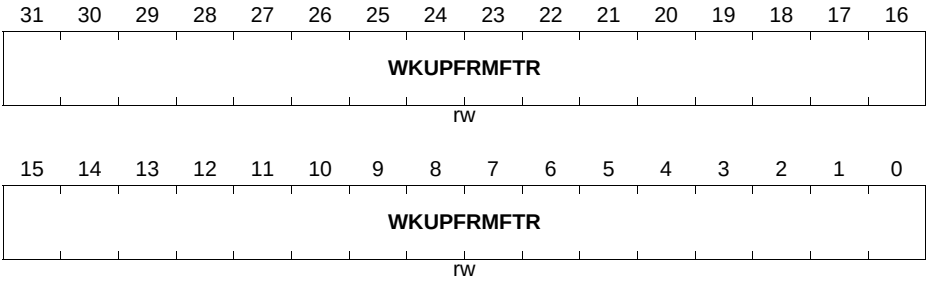
Field	Bits	Type	Description
RRCSTS	[6:5]	r	**MTL Rx FIFO Read Controller State** This field gives the state of the Rx FIFO read Controller: * 00B: IDLE state * 01B: Reading frame data * 10B: Reading frame status (or timestamp) * 11B: Flushing the frame data and status
Reserved_7	7	r	**Reserved**
RXFSTS	[9:8]	r	**MTL Rx FIFO Fill-level Status** This field gives the status of the fill-level of the Rx FIFO: * 00B: Rx FIFO Empty * 01B: Rx FIFO fill level is below the flow-control deactivate threshold * 10B: Rx FIFO fill level is above the flow-control activate threshold * 11B: Rx FIFO Full
Reserved_15_10	[15:10]	r	**Reserved**
TPESTS	16	r	**MAC MII Transmit Protocol Engine Status** When high, this bit indicates that the MAC MII transmit protocol engine is actively transmitting data and is not in the IDLE state.
TFCSTS	[18:17]	r	**MAC Transmit Frame Controller Status** This field indicates the state of the MAC Transmit Frame Controller module: * 00B: IDLE state * 01B: Waiting for Status of previous frame or IFG or backoff period to be over * 10B: Generating and transmitting a PAUSE control frame (in the full-duplex mode) * 11B: Transferring input frame for transmission
TXPAUSED	19	r	**MAC transmitter in PAUSE** When high, this bit indicates that the MAC transmitter is in the PAUSE condition (in the full-duplex only mode) and hence does not schedule any frame for transmission.

Field	Bits	Type	Description
TRCSTS	[21:20]	r	**MTL Tx FIFO Read Controller Status** This field indicates the state of the Tx FIFO Read Controller: * 00B: IDLE state * 01B: READ state (transferring data to MAC transmitter) * 10B: Waiting for TxStatus from MAC transmitter * 11B: Writing the received TxStatus or flushing the Tx FIFO
TWCSTS	22	r	**MTL Tx FIFO Write Controller Active Status** When high, this bit indicates that the MTL Tx FIFO Write Controller is active and transferring data to the Tx FIFO.
Reserved_23	23	r	**Reserved**
TXFSTS	24	r	**MTL Tx FIFO Not Empty Status** When high, this bit indicates that the MTL Tx FIFO is not empty and some data is left for transmission.
TXSTSFSTS	25	r	**MTL TxStatus FIFO Full Status** When high, this bit indicates that the MTL TxStatus FIFO is full. Therefore, the MTL cannot accept any more frames for transmission. This bit is reserved in the ETH-AHB and ETH-DMA configurations.
Reserved_31_26	[31:26]	r	**Reserved**

REMOTE_WAKE_UP_FRAME_FILTER

This is the address through which the application writes or reads the remote wake-up frame filter registers (wkupfmfilter_reg). The wkupfmfilter_reg register is a pointer to eight wkupfmfilter_reg registers. The wkupfmfilter_reg register is loaded by sequentially loading the eight register values. Eight sequential writes to this address (0028H) writes all wkupfmfilter_reg registers. Similarly, eight sequential reads from this address (0028H) read all wkupfmfilter_reg registers. This register contains the higher 16 bits of the seventh MAC address.

ETH0_REMOTE_WAKE_UP_FRAME_FILTER
Remote Wake Up Frame Filter Register (28_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
WKUPFRMFTR	[31:0]	rw	**Remote Wake-Up Frame Filter**

PMT_CONTROL_STATUS

ETH0_PMT_CONTROL_STATUS

PMT Control and Status Register (2C$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RWKFILTRST	Reserved_30_10														
rw	r														

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved_30_10						GLBLUCAST	Reserved_8_7		RWKPRCVD	MGKPRCVD	Reserved_4_3		RWKPKTEN	MGKPKTEN	PWRDWN
r						rw	r		r	r	r		rw	rw	rw

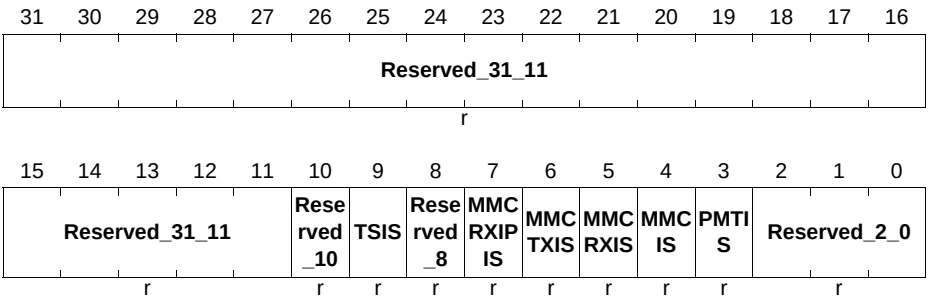
Field	Bits	Type	Description
PWRDWN	0	rw	**Power Down** When set, the MAC receiver drops all received frames until it receives the expected magic packet or wake-up frame. This bit is then self-cleared and the power-down mode is disabled. The Software can also clear this bit before the expected magic packet or wake-up frame is received. The frames, received by the MAC after this bit is cleared, are forwarded to the application. This bit must only be set when the Magic Packet Enable, Global Unicast, or Wake-Up Frame Enable bit is set high. Note: You can gate-off the CSR clock during the power-down mode. However, when the CSR clock is gated-off, you cannot perform any read or write operations on this register. Therefore, the Software cannot clear this bit.
MGKPKTEN	1	rw	**Magic Packet Enable** When set, enables generation of a power management event because of magic packet reception.
RWKPKTEN	2	rw	**Wake-Up Frame Enable** When set, enables generation of a power management event because of wake-up frame reception.

Field	Bits	Type	Description
Reserved_4_3	[4:3]	r	**Reserved**
MGKPRCVD	5	r	**Magic Packet Received** When set, this bit indicates that the power management event is generated because of the reception of a magic packet. This bit is cleared by a Read into this register.
RWKPRCVD	6	r	**Wake-Up Frame Received** When set, this bit indicates the power management event is generated because of the reception of a wake-up frame. This bit is cleared by a Read into this register.
Reserved_8_7	[8:7]	r	**Reserved**
GLBLUCAST	9	rw	**Global Unicast** When set, enables any unicast packet filtered by the MAC (DAF) address recognition to be a wake-up frame.
Reserved_30_10	[30:10]	r	**Reserved**
RWKFILTRST	31	rw	**Wake-Up Frame Filter Register Pointer Reset** When set, resets the remote wake-up frame filter register pointer to 000B. It is automatically cleared after 1 clock cycle.

INTERRUPT_STATUS

The Interrupt Status register identifies the events in the MAC that can generate interrupt.

ETH0_INTERRUPT_STATUS
Interrupt Register **(38_H)** **Reset Value: 0000 0000$_H$**



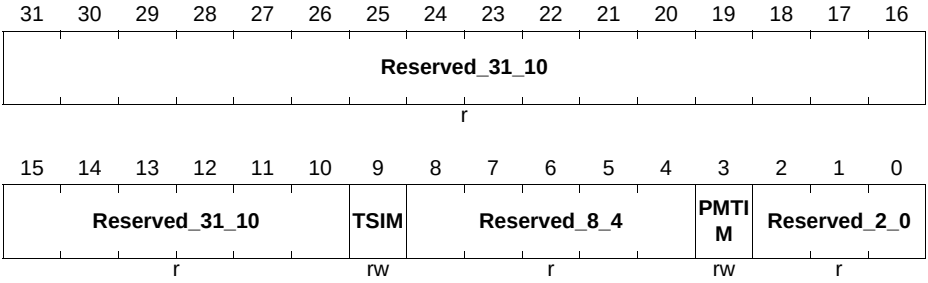
Field	Bits	Type	Description
Reserved_2_0	[2:0]	r	**Reserved**
PMTIS	3	r	**PMT Interrupt Status** This bit is set when a Magic packet or Wake-on-LAN frame is received in the power-down mode. This bit is cleared when both PMT_CONTROL_STATUS.MGKPRC VD and PMT_CONTROL_STATUS.RWKPRCVD are cleared because of a read operation to the PMT Control and Status register.
MMCIS	4	r	**MMC Interrupt Status** This bit is set high when any of the Bits MMCRXIS, MMCTXIS or MMCRXIPIS is set high and cleared only when all of these bits are low.
MMCRXIS	5	r	**MMC Receive Interrupt Status** This bit is set high when an interrupt is generated in the MMC Receive Interrupt Register. This bit is cleared when all the bits in this interrupt register are cleared.

Field	Bits	Type	Description
MMCTXIS	6	r	**MMC Transmit Interrupt Status** This bit is set high when an interrupt is generated in the MMC Transmit Interrupt Register. This bit is cleared when all the bits in this interrupt register are cleared.
MMCRXIPIS	7	r	**MMC Receive Checksum Offload Interrupt Status** This bit is set high when an interrupt is generated in the MMC Receive Checksum Offload Interrupt Register. This bit is cleared when all the bits in this interrupt register are cleared.
Reserved_8	8	r	**Reserved**
TSIS	9	r	**Timestamp Interrupt Status** When the Advanced Timestamp feature is enabled, this bit is set when any of the following conditions is true: * The system time value equals or exceeds the value specified in the Target Time High and Low registers. * There is an overflow in the seconds register. This bit is cleared on reading Timestamp STATUS.TSSOVF Register. If default Timestamping is enabled, when set, this bit indicates that the system time value is equal to or exceeds the value specified in the Target Time registers. In this mode, this bit is cleared after the completion of the read of this bit. In all other modes, this bit is reserved.
Reserved_10	10	r	**Reserved**
Reserved_31_11	[31:11]	r	**Reserved**

INTERRUPT_MASK

The Interrupt Mask Register bits enable you to mask the interrupt signal because of the corresponding event in the Interrupt Status Register.

ETH0_INTERRUPT_MASK
Interrupt Mask Register **($3C_H$)** **Reset Value: 0000 0000$_H$**

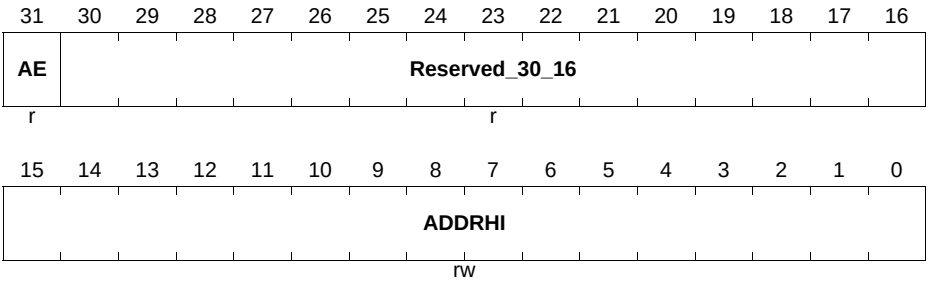


Field	Bits	Type	Description
Reserved_2_0	[2:0]	r	**Reserved**
PMTIM	3	rw	**PMT Interrupt Mask** When set, this bit disables the assertion of the interrupt signal because of the setting of PMT Interrupt Status bit INTERRUPT_STATUS.PMTIS.
Reserved_8_4	[8:4]	r	**Reserved**
TSIM	9	rw	**Timestamp Interrupt Mask** When set, this bit disables the assertion of the interrupt signal because of the setting of Timestamp Interrupt Status bit INTERRUPT_STATUS.TSIS. This bit is valid only when IEEE1588 timestamping is enabled. In all other modes, this bit is reserved.
Reserved_31_10	[31:10]	r	**Reserved**

MAC_ADDRESS0_HIGH

The MAC Address0 High register holds the upper 16 bits of the first 6-byte MAC address of the station. The first DA byte that is received on the MII interface corresponds to the LS byte (Bits [7:0]) of the MAC Address Low register. For example, if 1122 3344 5566H is received (11H in lane 0 of the first column) on the MII as the destination address, then the MacAddress0 Register [47:0] is compared with 6655 4433 2211H. If the MAC address registers are configured to be double-synchronized to the MII clock domains, then the synchronization is triggered only when Bits[31:24] of the MAC Address0 Low Register are written. For proper synchronization updates, the consecutive writes to this Address Low Register should be performed after at least four clock cycles in the destination clock domain.

ETH0_MAC_ADDRESS0_HIGH
MAC Address0 High Register **(40_H)** **Reset Value: 8000 FFFF$_H$**

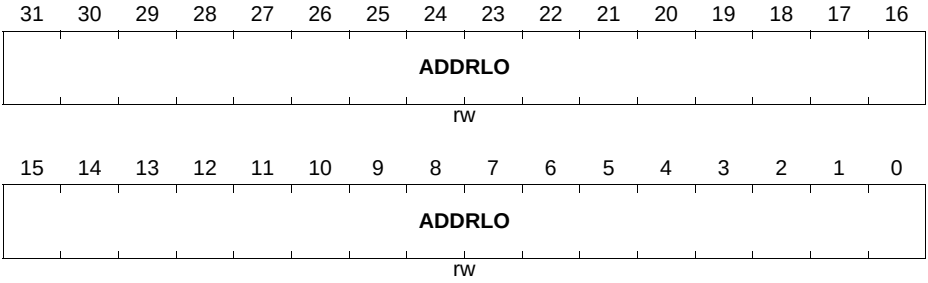


Field	Bits	Type	Description
ADDRHI	[15:0]	rw	**MAC Address0 [47:32]** This field contains the upper 16 bits (47:32) of the first 6-byte MAC address. The MAC uses this field for filtering the received frames and inserting the MAC address in the Transmit Flow Control (PAUSE) Frames.
Reserved_30_16	[30:16]	r	**Reserved**
AE	31	r	**Address Enable** This bit is always set to 1.

MAC_ADDRESS0_LOW

The MAC Address0 Low register holds the lower 32 bits of the first 6-byte MAC address of the station.

ETH0_MAC_ADDRESS0_LOW
MAC Address0 Low Register **(44_H)** **Reset Value: FFFF FFFF$_H$**

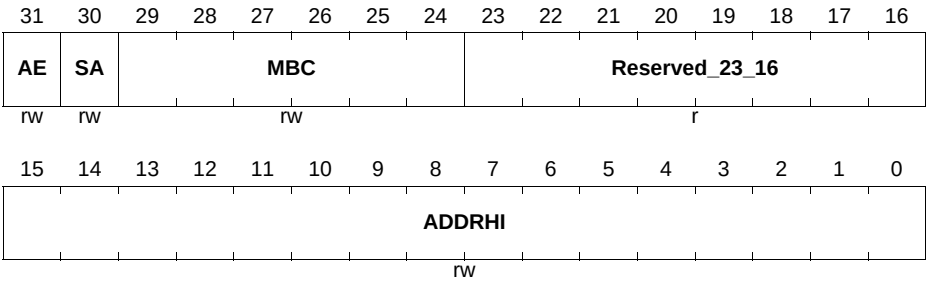


Field	Bits	Type	Description
ADDRLO	[31:0]	rw	**MAC Address0 [31:0]** This field contains the lower 32 bits of the first 6-byte MAC address. This is used by the MAC for filtering the received frames and inserting the MAC address in the Transmit Flow Control (PAUSE) Frames.

32-bit Register - MAC_ADDRESS1_HIGH

The MAC Address1 High register holds the upper 16 bits of the second 6-byte MAC address of the station. If the MAC address registers are configured to be double-synchronized to the MII clock domains, then the synchronization is triggered only when Bits[31:24] of the MAC Address1 Low Register are written. For proper synchronization updates, the consecutive writes to this Address Low Register should be performed after at least four clock cycles in the destination clock domain.

ETH0_MAC_ADDRESS1_HIGH
MAC Address1 High Register **(48_H)** **Reset Value: 0000 FFFF$_H$**



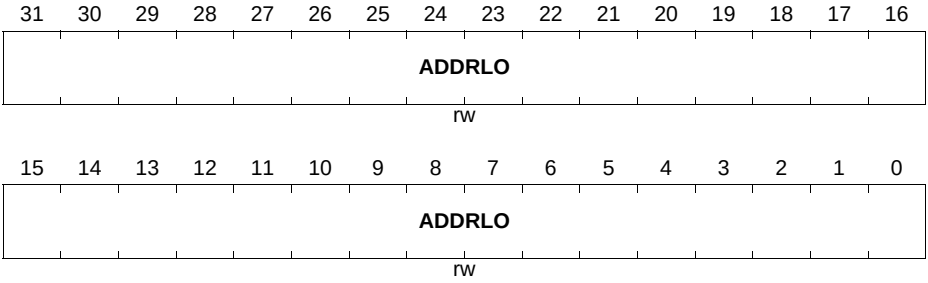
Field	Bits	Type	Description
ADDRHI	[15:0]	rw	**MAC Address1 [47:32]** This field contains the upper 16 bits (47:32) of the second 6-byte MAC address.
Reserved_23_16	[23:16]	r	**Reserved**

Field	Bits	Type	Description
MBC	[29:24]	rw	**Mask Byte Control** These bits are mask control bits for comparison of each of the MAC Address bytes. When set high, the MAC does not compare the corresponding byte of received DA or SA with the contents of MAC Address1 registers. Each bit controls the masking of the bytes as follows: * Bit 29: MAC_ADDRESS1_HIGH [15:8] * Bit 28: MAC_ADDRESS1_HIGH [7:0] * Bit 27: MAC_ADDRESS1_LOW [31:24] * ... * Bit 24: MAC_ADDRESS1_LOW [7:0] You can filter a group of addresses (known as group address filtering) by masking one or more bytes of the address.
SA	30	rw	**Source Address** When this bit is set, the MAC Address1[47:0] is used to compare with the SA fields of the received frame. When this bit is reset, the MAC Address1[47:0] is used to compare with the DA fields of the received frame.
AE	31	rw	**Address Enable** When this bit is set, the address filter module uses the second MAC address for perfect filtering. When this bit is reset, the address filter module ignores the address for filtering.

MAC_ADDRESS1_LOW

The MAC Address1 Low register holds the lower 32 bits of the second 6-byte MAC address of the station.

ETH0_MAC_ADDRESS1_LOW
MAC Address1 Low Register **($4C_H$)** **Reset Value: FFFF FFFF$_H$**

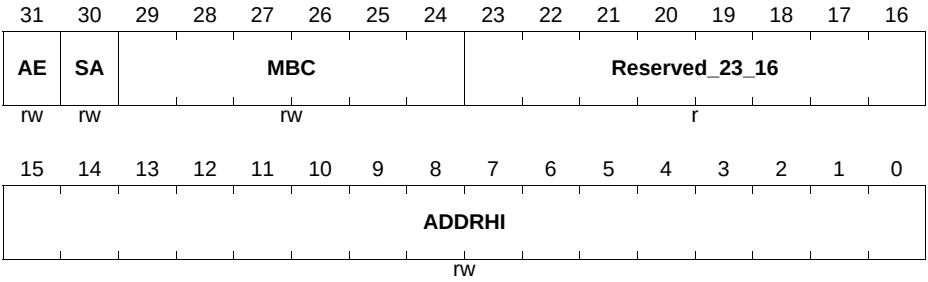


Field	Bits	Type	Description
ADDRLO	[31:0]	rw	**MAC Address1 [31:0]** This field contains the lower 32 bits of the second 6-byte MAC address. The content of this field is undefined until loaded by the Application after the initialization process.

MAC_ADDRESS2_HIGH

The MAC Address2 High register holds the upper 16 bits of the third 6-byte MAC address of the station. If the MAC address registers are configured to be double-synchronized to the MII clock domains, then the synchronization is triggered only when Bits[31:24] (in little-endian mode) or Bits[7:0] (in big-endian mode) of the MAC Address2 Low Register are written. For proper synchronization updates, consecutive writes to this MAC Address2 Low Register must be performed after at least four clock cycles in the destination clock domain.

ETH0_MAC_ADDRESS2_HIGH
MAC Address2 High Register **(50_H)** **Reset Value: 0000 FFFF$_H$**



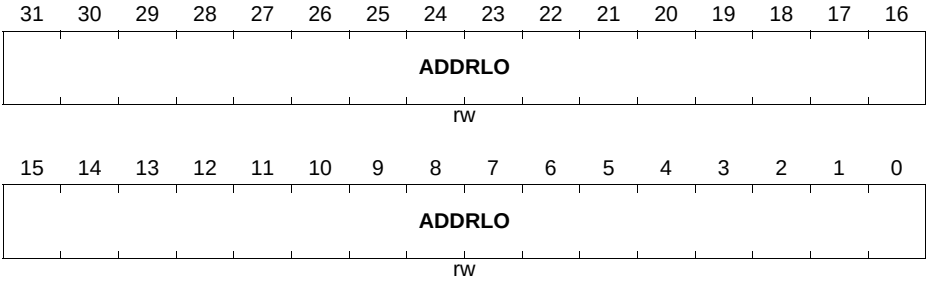
Field	Bits	Type	Description
ADDRHI	[15:0]	rw	**MAC Address2 [47:32]** This field contains the upper 16 bits (47:32) of the third 6-byte MAC address.
Reserved_23_16	[23:16]	r	**Reserved**
MBC	[29:24]	rw	**Mask Byte Control** These bits are mask control bits for comparison of each of the MAC Address bytes. When set high, the MAC does not compare the corresponding byte of received DA or SA with the contents of MAC Address2 registers. Each bit controls the masking of the bytes as follows: * Bit 29: MAC_ADDRESS1_HIGH [15:8] * Bit 28: MAC_ADDRESS1_HIGH [7:0] * Bit 27: MAC_ADDRESS1_LOW [31:24] * ... * Bit 24: MAC_ADDRESS1_LOW [7:0]

Field	Bits	Type	Description
SA	30	rw	**Source Address** When this bit is set, the MAC Address2[47:0] is used to compare with the SA fields of the received frame. When this bit is reset, the MAC Address2[47:0] is used to compare with the DA fields of the received frame.
AE	31	rw	**Address Enable** When this bit is set, the address filter module uses the third MAC address for perfect filtering. When this bit is reset, the address filter module ignores the address for filtering.

MAC_ADDRESS2_LOW

The MAC Address2 Low register holds the lower 32 bits of the third 6-byte MAC address of the station.

ETH0_MAC_ADDRESS2_LOW
MAC Address2 Low Register **(54_H)** **Reset Value: FFFF FFFF$_H$**

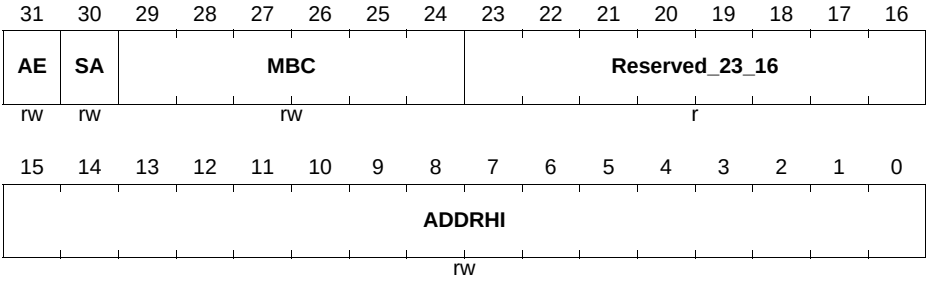


Field	Bits	Type	Description
ADDRLO	[31:0]	rw	**MAC Address2 [31:0]** This field contains the lower 32 bits of the third 6-byte MAC address. The content of this field is undefined until loaded by the Application after the initialization process.

MAC_ADDRESS3_HIGH

The MAC Address3 High register holds the upper 16 bits of the fourth 6-byte MAC address of the station. If the MAC address registers are configured to be double-synchronized to the MII clock domains, then the synchronization is triggered only when Bits[31:24] of the MAC Address3 Low Register are written. For proper synchronization updates, consecutive writes to this MAC Address3 Low Register must be performed after at least four clock cycles in the destination clock domain.

ETH0_MAC_ADDRESS3_HIGH
MAC Address3 High Register **(58_H)** **Reset Value: 0000 FFFF$_H$**



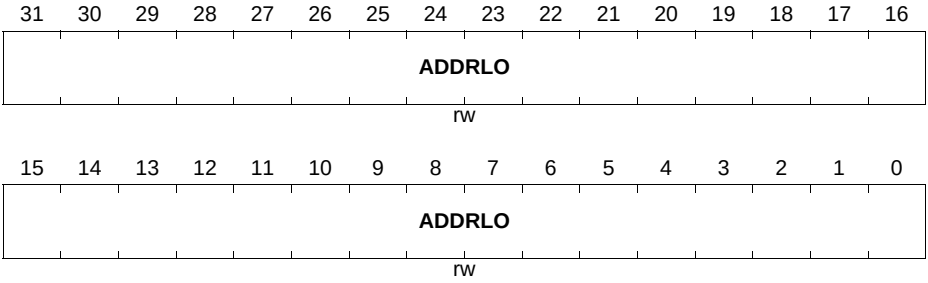
Field	Bits	Type	Description
ADDRHI	[15:0]	rw	**MAC Address3 [47:32]** This field contains the upper 16 bits (47:32) of the fourth 6-byte MAC address.
Reserved_23_16	[23:16]	r	**Reserved**
MBC	[29:24]	rw	**Mask Byte Control** These bits are mask control bits for comparison of each of the MAC Address bytes. When set high, the MAC does not compare the corresponding byte of received DA or SA with the contents of MAC Address3 registers. Each bit controls the masking of the bytes as follows: * Bit 29: MAC_ADDRESS1_HIGH [15:8] * Bit 28: MAC_ADDRESS1_HIGH [7:0] * Bit 27: MAC_ADDRESS1_HIGH [31:24] * ... * Bit 24: MAC_ADDRESS1_HIGH [7:0]

Field	Bits	Type	Description
SA	30	rw	**Source Address** When this bit is set, the MAC Address3[47:0] is used to compare with the SA fields of the received frame. When this bit is reset, the MAC Address3[47:0] is used to compare with the DA fields of the received frame.
AE	31	rw	**Address Enable** When this bit is set, the address filter module uses the fourth MAC address for perfect filtering. When this bit is reset, the address filter module ignores the address for filtering.

MAC_ADDRESS3_LOW

The MAC Address3 Low register holds the lower 32 bits of the fourth 6-byte MAC address of the station.

ETH0_MAC_ADDRESS3_LOW
MAC Address3 Low Register **($5C_H$)** **Reset Value: FFFF FFFF$_H$**

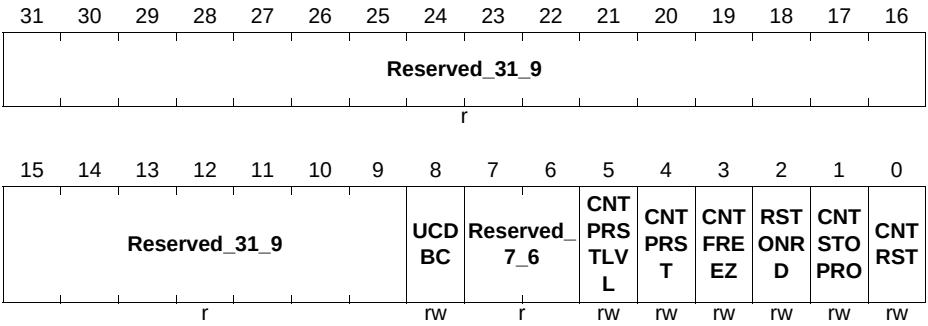


Field	Bits	Type	Description
ADDRLO	[31:0]	rw	**MAC Address3 [31:0]** This field contains the lower 32 bits of the fourth 6-byte MAC address. The content of this field is undefined until loaded by the Application after the initialization process.

MMC_CONTROL

The MMC Control register establishes the operating mode of the management counters. Note: The bit 0 (Counters Reset) has higher priority than bit 4 (Counter Preset). Therefore, when the Software tries to set both bits in the same write cycle, all counters are cleared and the bit 4 is not set.

ETH0_MMC_CONTROL
MMC Control Register **(100_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
CNTRST	0	rw	**Counters Reset** When this bit is set, all counters are reset. This bit is cleared automatically after one clock cycle.
CNTSTOPRO	1	rw	**Counters Stop Rollover** When this bit is set, after reaching maximum value, the counter does not roll over to zero.
RSTONRD	2	rw	**Reset on Read** When this bit is set, the MMC counters are reset to zero after Read (self-clearing after reset). The counters are cleared when the least significant byte lane (bits[7:0]) is read.
CNTFREEZ	3	rw	**MMC Counter Freeze** When this bit is set, it freezes all MMC counters to their current value. Until this bit is reset to 0, no MMC counter is updated because of any transmitted or received frame. If any MMC counter is read with the Reset on Read bit set, then that counter is also cleared in this mode.

Field	Bits	Type	Description
CNTPRST	4	rw	**Counters Preset** When this bit is set, all counters are initialized or preset to almost full or almost half according to bit 5. This bit is cleared automatically after 1 clock cycle. This bit, along with bit 5, is useful for debugging and testing the assertion of interrupts because of MMC counter becoming half-full or full.
CNTPRSTLVL	5	rw	**Full-Half Preset** When low and bit 4 is set, all MMC counters get preset to almost-half value. All octet counters get preset to 7FFF F800H (half - 2KBytes) and all frame-counters gets preset to 7FFF FFF0H (half - 16). When this bit is high and bit 4 is set, all MMC counters get preset to almost-full value. All octet counters get preset to FFFF F800H (full - 2KBytes) and all frame-counters gets preset to FFFF FFF0H (full - 16). For 16-bit counters, the almost-half preset values are 7800H and 7FF0H for the respective octet and frame counters. Similarly, the almost-full preset values for the 16-bit counters are F800H and FFF0H.
Reserved_7_6	[7:6]	r	**Reserved**
UCDBC	8	rw	**Update MMC Counters for Dropped Broadcast Frames** When set, this bit enables MAC to update all the related MMC Counters for Broadcast frames dropped due to setting of MAC_Filter.DBF bit. When reset, MMC Counters are not updated for dropped Broadcast frames.
Reserved_31_9	[31:9]	r	**Reserved**

MMC_RECEIVE_INTERRUPT

The MMC Receive Interrupt register maintains the interrupts that are generated when the following happens: * Receive statistic counters reach half of their maximum values (8000 0000H for 32-bit counter and 8000H for 16-bit counter). * Receive statistic counters cross their maximum values (FFFF FFFFH for 32-bit counter and FFFFH for 16-bit counter). When the Counter Stop Rollover is set, then interrupts are set but the counter remains at all-ones. The MMC Receive Interrupt register is a 32-bit wide register. An interrupt bit is cleared when the respective MMC counter that caused the interrupt is read. The least significant byte lane (Bits[7:0]) of the respective counter must be read in order to clear the interrupt bit.

ETH0_MMC_RECEIVE_INTERRUPT
MMC Receive Interrupt Register (104_H) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved_31_26						RXCTRLFIS	RXRCVERRFIS	RXWDOGFIS	RXVLANGBFIS	RXFOVFIS	RXPAUSFIS	RXORANGEFIS	RXLENERFIS	RXUCGFIS	RX1024TMAXOCTGBFI
r						r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RX512T1023OCTGBFIS	RX256T511OCTGBFIS	RX128T255OCTGBFIS	RX65T127OCTGBFIS	RX64OCTGBFIS	RXOSIZEGFIS	RXUSIZEGFIS	RXJABERFIS	RXRUNTFIS	RXALGNERFIS	RXCRCERFIS	RXMCGFIS	RXBCGFIS	RXGOCTIS	RXGBOCTIS	RXGBFRMIS
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Field	Bits	Type	Description
RXGBFRMIS	0	r	**MMC Receive Good Bad Frame Counter Interrupt Status** This bit is set when the rxframecount_bg counter reaches half of the maximum value or the maximum value.
RXGBOCTIS	1	r	**MMC Receive Good Bad Octet Counter Interrupt Status** This bit is set when the rxoctetcount_bg counter reaches half of the maximum value or the maximum value.

Field	Bits	Type	Description
RXGOCTIS	2	r	**MMC Receive Good Octet Counter Interrupt Status.** This bit is set when the RX_OCTET_COUNT_GOOD counter reaches half of the maximum value or the maximum value.
RXBCGFIS	3	r	**MMC Receive Broadcast Good Frame Counter Interrupt Status.** This bit is set when the RX_BROADCAST_FRAMES_GOOD counter reaches half of the maximum value or the maximum value.
RXMCGFIS	4	r	**MMC Receive Multicast Good Frame Counter Interrupt Status** This bit is set when the RX_MULTICAST_FRAMES_GOOD counter reaches half of the maximum value or the maximum value.
RXCRCERFIS	5	r	**MMC Receive CRC Error Frame Counter Interrupt Status** This bit is set when the RX_CRC_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
RXALGNERFIS	6	r	**MMC Receive Alignment Error Frame Counter Interrupt Status** This bit is set when the RX_ALIGNMENT_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
RXRUNTFIS	7	r	**MMC Receive Runt Frame Counter Interrupt Status** This bit is set when the RX_RUNT_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
RXJABERFIS	8	r	**MMC Receive Jabber Error Frame Counter Interrupt Status** This bit is set when the RX_JABBER_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
RXUSIZEGFIS	9	r	**MMC Receive Undersize Good Frame Counter Interrupt Status** This bit is set when the RX_UNDERSIZE_FRAMES_GOOD counter reaches half of the maximum value or the maximum value.

Field	Bits	Type	Description
RXOSIZEGFIS	10	r	**MMC Receive Oversize Good Frame Counter Interrupt Status** This bit is set when the RX_OVERSIZE_FRAMES_GOOD counter reaches half of the maximum value or the maximum value.
RX64OCTGBFIS	11	r	**MMC Receive 64 Octet Good Bad Frame Counter Interrupt Status** This bit is set when the RX_64OCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
RX65T127OCTGBFIS	12	r	**MMC Receive 65 to 127 Octet Good Bad Frame Counter Interrupt Status** This is set when the RX_65TO127OCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
RX128T255OCTGBFIS	13	r	**MMC Receive 128 to 255 Octet Good Bad Frame Counter Interrupt Status** This bit is set when the RX_128TO255OCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
RX256T511OCTGBFIS	14	r	**MMC Receive 256 to 511 Octet Good Bad Frame Counter Interrupt Status** This bit is set when the RX_256TO511OCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
RX512T1023OCTGBFIS	15	r	**MMC Receive 512 to 1023 Octet Good Bad Frame Counter Interrupt Status** This bit is set when the RX_512TO1023OCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.

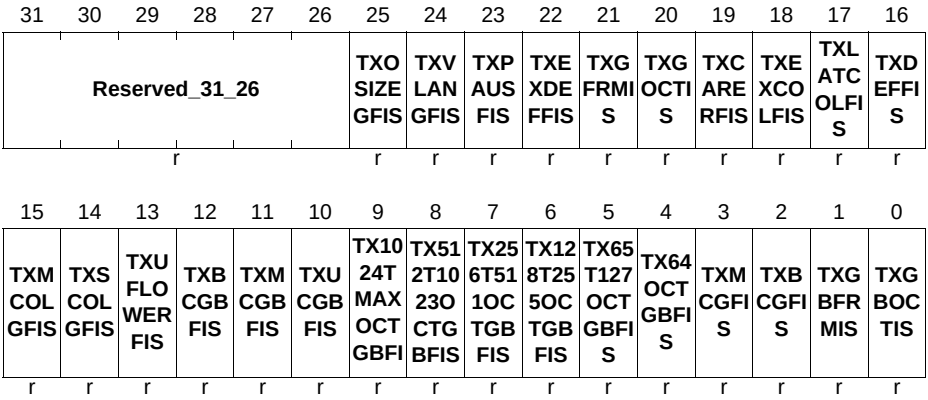
Field	Bits	Type	Description
RX1024TMAXOCTGBFIS	16	r	**MMC Receive 1024 to Maximum Octet Good Bad Frame Counter Interrupt Status** This bit is set when the RX_1024TOMAXOCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
RXUCGFIS	17	r	**MMC Receive Unicast Good Frame Counter Interrupt Status** This bit is set when the rxunicastframes_gb counter reaches half of the maximum value or the maximum value.
RXLENERFIS	18	r	**MMC Receive Length Error Frame Counter Interrupt Status** This bit is set when the RX_LENGTH_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
RXORANGEFIS	19	r	**MMC Receive Out Of Range Error Frame Counter Interrupt Status** This bit is set when the RX_OUT_OF_RANGE_TYPE_FRAMES counter reaches half of the maximum value or the maximum value.
RXPAUSFIS	20	r	**MMC Receive Pause Frame Counter Interrupt Status** This bit is set when the rxpauseframe counter reaches half of the maximum value or the maximum value.
RXFOVFIS	21	r	**MMC Receive FIFO Overflow Frame Counter Interrupt Status** This bit is set when the RX_FIFO_OVERFLOW_FRAMES counter reaches half of the maximum value or the maximum value.
RXVLANGBFIS	22	r	**MMC Receive VLAN Good Bad Frame Counter Interrupt Status** This bit is set when the RX_VLAN_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.

Field	Bits	Type	Description
RXWDOGFIS	23	r	**MMC Receive Watchdog Error Frame Counter Interrupt Status** This bit is set when the RX_WATCHDOG_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
RXRCVERRFIS	24	r	**MMC Receive Error Frame Counter Interrupt Status** This bit is set when the rxrcverror counter reaches half of the maximum value or the maximum value.
RXCTRLFIS	25	r	**MMC Receive Control Frame Counter Interrupt Status** This bit is set when the rxctrlframes_g counter reaches half of the maximum value or the maximum value.
Reserved_31_26	[31:26]	r	**Reserved**

MMC_TRANSMIT_INTERRUPT

The MMC Transmit Interrupt register maintains the interrupts generated when transmit statistic counters reach half of their maximum values (8000 0000H for 32-bit counter and 8000H for 16-bit counter), and the maximum values (FFFF FFFFH for 32-bit counter and FFFFH for 16-bit counter). When Counter Stop Rollover is set, then interrupts are set but the counter remains at all-ones. The MMC Transmit Interrupt register is a 32-bit wide register. An interrupt bit is cleared when the respective MMC counter that caused the interrupt is read. The least significant byte lane (Bits[7:0]) of the respective counter must be read in order to clear the interrupt bit.

ETH0_MMC_TRANSMIT_INTERRUPT
MMC Transmit Interrupt Register (108_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
TXGBOCTIS	0	r	**MMC Transmit Good Bad Octet Counter Interrupt Status** This bit is set when the TX_OCTET_COUNT_GOOD_BAD counter reaches half of the maximum value or the maximum value.
TXGBFRMIS	1	r	**MMC Transmit Good Bad Frame Counter Interrupt Status** This bit is set when the TX_FRAME_COUNT_GOOD_BAD counter reaches half of the maximum value or the maximum value.

Field	Bits	Type	Description
TXBCGFIS	2	r	**MMC Transmit Broadcast Good Frame Counter Interrupt Status** This bit is set when the TX_BROADCAST_FRAMES_GOOD counter reaches half of the maximum value or the maximum value.
TXMCGFIS	3	r	**MMC Transmit Multicast Good Frame Counter Interrupt Status** This bit is set when the TX_MULTICAST_FRAMES_GOOD counter reaches half of the maximum value or the maximum value.
TX64OCTGBFIS	4	r	**MMC Transmit 64 Octet Good Bad Frame Counter Interrupt Status.** This bit is set when the TX_64OCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
TX65T127OCTGBFIS	5	r	**MMC Transmit 65 to 127 Octet Good Bad Frame Counter Interrupt Status** This bit is set when the TX_65TO127OCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
TX128T255OCTGBFIS	6	r	**MMC Transmit 128 to 255 Octet Good Bad Frame Counter Interrupt Status** This bit is set when the TX_128TO255OCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
TX256T511OCTGBFIS	7	r	**MMC Transmit 256 to 511 Octet Good Bad Frame Counter Interrupt Status** This bit is set when the TX_256TO511OCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.

Field	Bits	Type	Description
TX512T1023OCTGBFIS	8	r	**MMC Transmit 512 to 1023 Octet Good Bad Frame Counter Interrupt Status** This bit is set when the TX_512TO1023OCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
TX1024TMAXOCTGBFIS	9	r	**MMC Transmit 1024 to Maximum Octet Good Bad Frame Counter Interrupt Status** This bit is set when the TX_1024TOMAXOCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
TXUCGBFIS	10	r	**MMC Transmit Unicast Good Bad Frame Counter Interrupt Status** This bit is set when the TX_UNICAST_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
TXMCGBFIS	11	r	**MMC Transmit Multicast Good Bad Frame Counter Interrupt Status** This bit is set when the TX_MULTICAST_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
TXBCGBFIS	12	r	**MMC Transmit Broadcast Good Bad Frame Counter Interrupt Status** This bit is set when the TX_BROADCAST_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
TXUFLOWERFIS	13	r	**MMC Transmit Underflow Error Frame Counter Interrupt Status** This bit is set when the TX_UNDERFLOW_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.

Field	Bits	Type	Description
TXSCOLGFIS	14	r	**MMC Transmit Single Collision Good Frame Counter Interrupt Status** This bit is set when the TX_SINGLE_COLLISION_GOOD_FRAMES counter reaches half of the maximum value or the maximum value.
TXMCOLGFIS	15	r	**MMC Transmit Multiple Collision Good Frame Counter Interrupt Status** This bit is set when the TX_MULTIPLE_COLLISION_GOOD_FRAMES counter reaches half of the maximum value or the maximum value.
TXDEFFIS	16	r	**MMC Transmit Deferred Frame Counter Interrupt Status** This bit is set when the TX_DEFERRED_FRAMES counter reaches half of the maximum value or the maximum value.
TXLATCOLFIS	17	r	**MMC Transmit Late Collision Frame Counter Interrupt Status** This bit is set when the TX_LATE_COLLISION_FRAMES counter reaches half of the maximum value or the maximum value.
TXEXCOLFIS	18	r	**MMC Transmit Excessive Collision Frame Counter Interrupt Status** This bit is set when the txexcesscol counter reaches half of the maximum value or the maximum value.
TXCARERFIS	19	r	**MMC Transmit Carrier Error Frame Counter Interrupt Status** This bit is set when the TX_CARRIER_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
TXGOCTIS	20	r	**MMC Transmit Good Octet Counter Interrupt Status** This bit is set when the TX_OCTET_COUNT_GOOD counter reaches half of the maximum value or the maximum value.
TXGFRMIS	21	r	**MMC Transmit Good Frame Counter Interrupt Status** This bit is set when the TX_FRAME_COUNT_GOOD counter reaches half of the maximum value or the maximum value.

Field	Bits	Type	Description
TXEXDEFFIS	22	r	**MMC Transmit Excessive Deferral Frame Counter Interrupt Status** This bit is set when the TX_EXCESSIVE_DEFERRAL_ERROR counter reaches half of the maximum value or the maximum value.
TXPAUSFIS	23	r	**MMC Transmit Pause Frame Counter Interrupt Status** This bit is set when the txpauseframeserror counter reaches half of the maximum value or the maximum value.
TXVLANGFIS	24	r	**MMC Transmit VLAN Good Frame Counter Interrupt Status** This bit is set when the TX_VLAN_FRAMES_GOOD counter reaches half of the maximum value or the maximum value.
TXOSIZEGFIS	25	r	**MMC Transmit Oversize Good Frame Counter Interrupt Status** This bit is set when the txoversize_g counter reaches half of the maximum value or the maximum value.
Reserved_31_26	[31:26]	r	**Reserved**

MMC_RECEIVE_INTERRUPT_MASK

ETH0_MMC_RECEIVE_INTERRUPT_MASK

MMC Reveive Interrupt Mask Register ($10C_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved_31_26						RXCTRLFIM	RXRCVERRFIM	RXWDOGFIM	RXVLANGBFIM	RXFOVFIM	RXPAUSFIM	RXORANGEFIM	RXLENERFIM	RXUCGFIM	RX1024TMAXOCTGBFI
r						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RX512T1023OCTGBFIM	RX256T511OCTGBFIM	RX128T255OCTGBFIM	RX65T127OCTGBFIM	RX64OCTGBFIM	RXOSIZEGFIM	RXUSIZEGFIM	RXJABERFIM	RXRUNTFIM	RXALGNERFIM	RXCRCERFIM	RXMCGFIM	RXBCGFIM	RXGOCTIM	RXGBOCTIM	RXGBFRMIM
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
RXGBFRMIM	0	rw	**MMC Receive Good Bad Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_FRAMES_COUNT_GOOD_BAD counter reaches half of the maximum value or the maximum value.
RXGBOCTIM	1	rw	**MMC Receive Good Bad Octet Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_OCTET_COUNT_GOOD_BAD counter reaches half of the maximum value or the maximum value.
RXGOCTIM	2	rw	**MMC Receive Good Octet Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_OCTET_COUNT_GOOD counter reaches half of the maximum value or the maximum value.

Field	Bits	Type	Description
RXBCGFIM	3	rw	**MMC Receive Broadcast Good Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_BROADCAST_FRAMES_GOOD counter reaches half of the maximum value or the maximum value.
RXMCGFIM	4	rw	**MMC Receive Multicast Good Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_MULTICAST_FRAMES_GOOD counter reaches half of the maximum value or the maximum value.
RXCRCERFIM	5	rw	**MMC Receive CRC Error Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_CRC_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
RXALGNERFIM	6	rw	**MMC Receive Alignment Error Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_ALIGNMENT_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
RXRUNTFIM	7	rw	**MMC Receive Runt Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_RUNT_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
RXJABERFIM	8	rw	**MMC Receive Jabber Error Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_JABBER_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
RXUSIZEGFIM	9	rw	**MMC Receive Undersize Good Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_UNDERSIZE_FRAMES_GOOD counter reaches half of the maximum value or the maximum value.
RXOSIZEGFIM	10	rw	**MMC Receive Oversize Good Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_OVERSIZE_FRAMES_GOOD counter reaches half of the maximum value or the maximum value.

Field	Bits	Type	Description
RX64OCTGBFIM	11	rw	**MMC Receive 64 Octet Good Bad Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_64OCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
RX65T127OCTGBFIM	12	rw	**MMC Receive 65 to 127 Octet Good Bad Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_65TO127OCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
RX128T255OCTGBFIM	13	rw	**MMC Receive 128 to 255 Octet Good Bad Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_128TO255OCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
RX256T511OCTGBFIM	14	rw	**MMC Receive 256 to 511 Octet Good Bad Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_256TO511OCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
RX512T1023OCTGBFIM	15	rw	**MMC Receive 512 to 1023 Octet Good Bad Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_512TO1023OCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
RX1024TMAXOCTGBFIM	16	rw	**MMC Receive 1024 to Maximum Octet Good Bad Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_1024TOMAXOCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.

Field	Bits	Type	Description
RXUCGFIM	17	rw	**MMC Receive Unicast Good Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_UNICAST_FRAMES_GOOD counter reaches half of the maximum value or the maximum value.
RXLENERFIM	18	rw	**MMC Receive Length Error Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_LENGTH_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
RXORANGEFIM	19	rw	**MMC Receive Out Of Range Error Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_OUT_OF_RANGE_TYPE_FRAMES counter reaches half of the maximum value or the maximum value.
RXPAUSFIM	20	rw	**MMC Receive Pause Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_PAUSE_FRAMES counter reaches half of the maximum value or the maximum value.
RXFOVFIM	21	rw	**MMC Receive FIFO Overflow Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_FIFO_OVERFLOW_FRAMES counter reaches half of the maximum value or the maximum value.
RXVLANGBFIM	22	rw	**MMC Receive VLAN Good Bad Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RX_VLAN_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
RXWDOGFIM	23	rw	**MMC Receive Watchdog Error Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the rxwatchdog counter reaches half of the maximum value or the maximum value.
RXRCVERRFIM	24	rw	**MMC Receive Error Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the rxrcverror error counter reaches half the maximum value, and also when it reaches the maximum value.

Field	Bits	Type	Description
RXCTRLFIM	25	rw	**MMC Receive Control Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the rxctrlframes counter reaches half the maximum value, and also when it reaches the maximum value.
Reserved_31_26	[31:26]	r	**Reserved**

MMC_TRANSMIT_INTERRUPT_MASK

The MMC Transmit Interrupt Mask register maintains the masks for the interrupts generated when the transmit statistic counters reach half of their maximum value or maximum value. This register is 32-bits wide.

ETH0_MMC_TRANSMIT_INTERRUPT_MASK
MMC Transmit Interrupt Mask Register (110_H) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved_31_26						TXOSIZEGFIM	TXVLANGFIM	TXPAUSFIM	TXEXDEFFIM	TXGFRMIM	TXGOCTIM	TXCARERFIM	TXEXCOLFIM	TXLATCOLFIM	TXDEFFIM
r						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TXMCOLGFIM	TXSCOLGFIM	TXUFLOWERFIM	TXBCGBFIM	TXMCGBFIM	TXUCGBFIM	TX1024TMAXOCTGBFI	TX512T1023OCTGBFIM	TX256T511OCTGBFIM	TX128T255OCTGBFIM	TX65T127OCTGBFIM	TX64OCTGBFIM	TXMCGFIM	TXBCGFIM	TXGBFRMIM	TXGBOCTIM
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
TXGBOCTIM	0	rw	**MMC Transmit Good Bad Octet Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_OCTET_COUNT_GOOD_BAD counter reaches half of the maximum value or the maximum value.
TXGBFRMIM	1	rw	**MMC Transmit Good Bad Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_FRAME_COUNT_GOOD_BAD counter reaches half of the maximum value or the maximum value.
TXBCGFIM	2	rw	**MMC Transmit Broadcast Good Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_BROADCAST_FRAMES_GOOD counter reaches half of the maximum value or the maximum value.

Field	Bits	Type	Description
TXMCGFIM	3	rw	**MMC Transmit Multicast Good Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_MULTICAST_FRAMES_GOOD counter reaches half of the maximum value or the maximum value.
TX64OCTGBFIM	4	rw	**MMC Transmit 64 Octet Good Bad Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_64OCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
TX65T127OCTGBFIM	5	rw	**MMC Transmit 65 to 127 Octet Good Bad Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_65TO127OCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
TX128T255OCTGBFIM	6	rw	**MMC Transmit 128 to 255 Octet Good Bad Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_128TO255OCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
TX256T511OCTGBFIM	7	rw	**MMC Transmit 256 to 511 Octet Good Bad Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_256TO511OCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
TX512T1023OCTGBFIM	8	rw	**MMC Transmit 512 to 1023 Octet Good Bad Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_512TO1023OCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.

Field	Bits	Type	Description
TX1024TMAXOCTGBFIM	9	rw	**MMC Transmit 1024 to Maximum Octet Good Bad Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_1024TOMAXOCTETS_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
TXUCGBFIM	10	rw	**MMC Transmit Unicast Good Bad Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_UNICAST_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
TXMCGBFIM	11	rw	**MMC Transmit Multicast Good Bad Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_MULTICAST_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
TXBCGBFIM	12	rw	**MMC Transmit Broadcast Good Bad Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_BROADCAST_FRAMES_GOOD_BAD counter reaches half of the maximum value or the maximum value.
TXUFLOWERFIM	13	rw	**MMC Transmit Underflow Error Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_UNDERFLOW_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
TXSCOLGFIM	14	rw	**MMC Transmit Single Collision Good Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_SINGLE_COLLISION_GOOD_FRAMES counter reaches half of the maximum value or the maximum value.

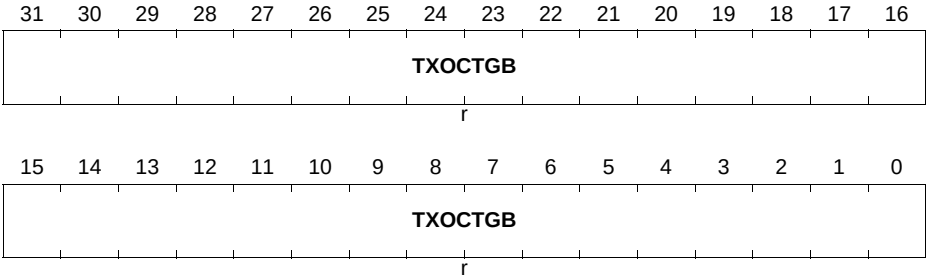
Field	Bits	Type	Description
TXMCOLGFIM	15	rw	**MMC Transmit Multiple Collision Good Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_MULTIPLE_COLLISION_GOOD_FRAMES counter reaches half of the maximum value or the maximum value.
TXDEFFIM	16	rw	**MMC Transmit Deferred Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_DEFERRED_FRAMES counter reaches half of the maximum value or the maximum value.
TXLATCOLFIM	17	rw	**MMC Transmit Late Collision Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_LATE_COLLISION_FRAMES counter reaches half of the maximum value or the maximum value.
TXEXCOLFIM	18	rw	**MMC Transmit Excessive Collision Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the txexcesscol counter reaches half of the maximum value or the maximum value.
TXCARERFIM	19	rw	**MMC Transmit Carrier Error Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_CARRIER_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
TXGOCTIM	20	rw	**MMC Transmit Good Octet Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_OCTET_COUNT_GOOD counter reaches half of the maximum value or the maximum value.
TXGFRMIM	21	rw	**MMC Transmit Good Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_FRAME_COUNT_GOOD counter reaches half of the maximum value or the maximum value.

Field	Bits	Type	Description
TXEXDEFFIM	22	rw	**MMC Transmit Excessive Deferral Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_EXCESSIVE_DEFERRAL_ERROR counter reaches half of the maximum value or the maximum value.
TXPAUSFIM	23	rw	**MMC Transmit Pause Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_PAUSE_FRAMES counter reaches half of the maximum value or the maximum value.
TXVLANGFIM	24	rw	**MMC Transmit VLAN Good Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the TX_VLAN_FRAMES_GOOD counter reaches half of the maximum value or the maximum value.
TXOSIZEGFIM	25	rw	**MMC Transmit Oversize Good Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the txoversize_g counter reaches half of the maximum value or the maximum value.
Reserved_31_26	[31:26]	r	**Reserved**

TX_OCTET_COUNT_GOOD_BAD

This register maintains the number of bytes transmitted in good and bad frames exclusive of preamble and retried bytes.

ETH0_TX_OCTET_COUNT_GOOD_BAD
Transmit Octet Count for Good and Bad Frames Register (114_H) Reset Value: $0000\ 0000_H$

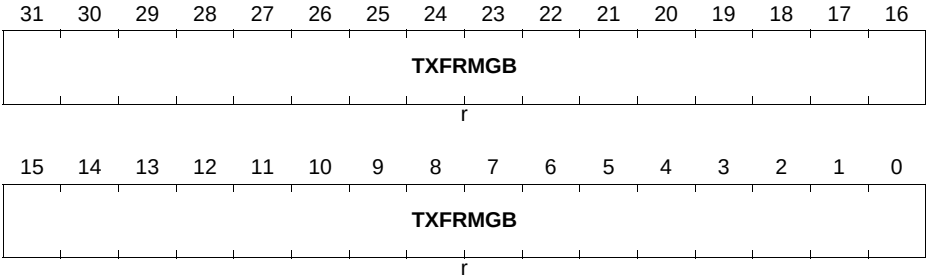


Field	Bits	Type	Description
TXOCTGB	[31:0]	r	**This field indicates the number of bytes transmitted in good and bad frames exclusive of preamble and retried bytes.**

TX_FRAME_COUNT_GOOD_BAD

This register maintains the number of good and bad frames transmitted, exclusive of retried frames.

ETH0_TX_FRAME_COUNT_GOOD_BAD
Transmit Frame Count for Goodand Bad Frames Register (118_H) Reset Value: $0000\ 0000_H$

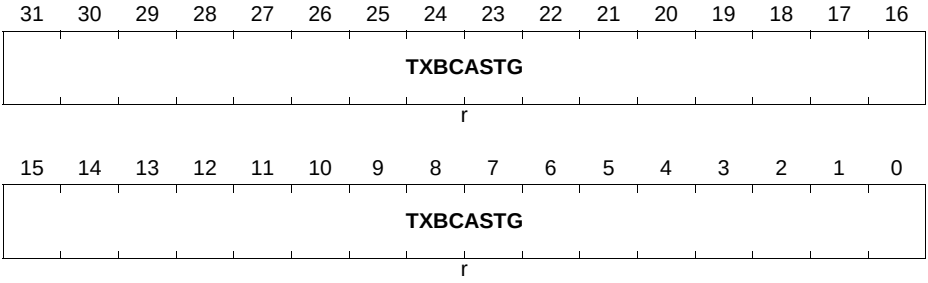


Field	Bits	Type	Description
TXFRMGB	[31:0]	r	**This field indicates the number of good and bad frames transmitted, exclusive of retried frames**

TX_BROADCAST_FRAMES_GOOD

This register maintains the number of transmitted good broadcast frames.

ETH0_TX_BROADCAST_FRAMES_GOOD
Transmit Frame Count for Good Broadcast Frames ($11C_H$) **Reset Value: $0000\ 0000_H$**

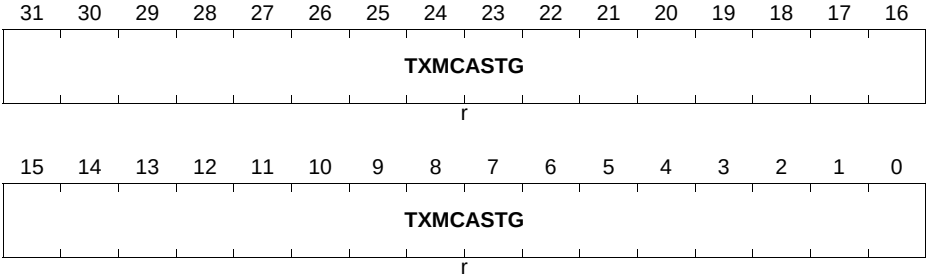


Field	Bits	Type	Description
TXBCASTG	[31:0]	r	**This field indicates the number of transmitted good broadcast frames.**

TX_MULTICAST_FRAMES_GOOD

This register maintains the number of transmitted good multicast frames.

ETH0_TX_MULTICAST_FRAMES_GOOD
Transmit Frame Count for Good Multicast Frames (120_H) Reset Value: $0000\ 0000_H$

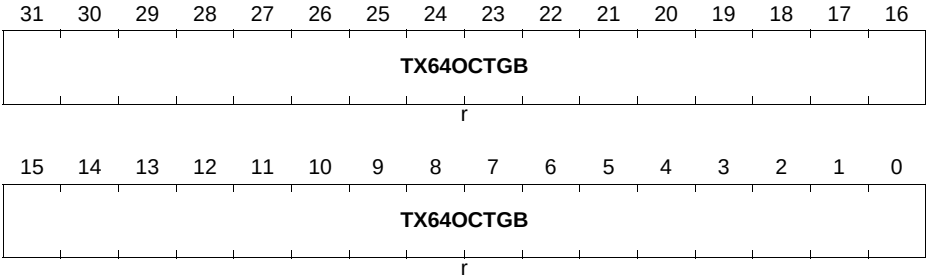


Field	Bits	Type	Description
TXMCASTG	[31:0]	r	**This field indicates the number of transmitted good multicast frames.**

TX_64OCTETS_FRAMES_GOOD_BAD

This register maintains the number of transmitted good and bad frames with length of 64 bytes, exclusive of preamble and retried frames.

ETH0_TX_64OCTETS_FRAMES_GOOD_BAD
Transmit Octet Count for Good and Bad 64 Byte Frames (124_H) Reset Value: $0000\ 0000_H$

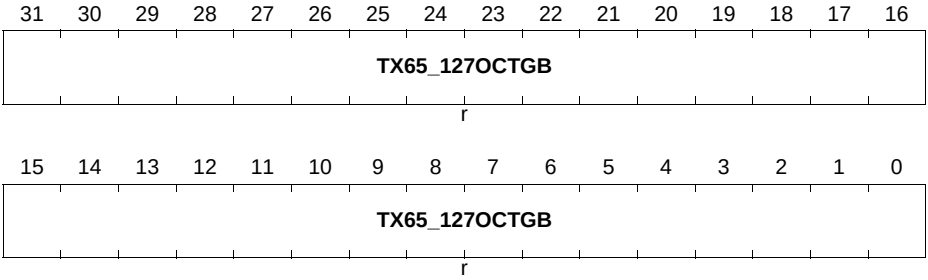


Field	Bits	Type	Description
TX64OCTGB	[31:0]	r	**This field indicates the number of transmitted good and bad frames with length of 64 bytes, exclusive of preamble and retried frames.**

TX_65TO127OCTETS_FRAMES_GOOD_BAD

This register maintains the number of transmitted good and bad frames with length between 65 and 127 (inclusive) bytes, exclusive of preamble and retried frames.

ETH0_TX_65TO127OCTETS_FRAMES_GOOD_BAD
Transmit Octet Count for Good and Bad 65 to 127 Bytes Frames (128_H)Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
TX65_127 OCTGB	[31:0]	r	**This field indicates the number of transmitted good and bad frames with length between 65 and 127 (inclusive) bytes, exclusive of preamble and retried frames.**

TX_128TO255OCTETS_FRAMES_GOOD_BAD

This register maintains the number of transmitted good and bad frames with length between 128 and 255 (inclusive) bytes, exclusive of preamble and retried frames.

ETH0_TX_128TO255OCTETS_FRAMES_GOOD_BAD
Transmit Octet Count for Good and Bad 128 to 255 Bytes Frames ($12C_H$) Reset Value: $0000\ 0000_H$

Field	Bits	Type	Description
TX128_255OCTGB	[31:0]	r	**This field indicates the number of transmitted good and bad frames with length between 128 and 255 (inclusive) bytes, exclusive of preamble and retried frames.**

TX_256TO511OCTETS_FRAMES_GOOD_BAD

This register maintains the number of transmitted good and bad frames with length between 256 and 511 (inclusive) bytes, exclusive of preamble and retried frames.

ETH0_TX_256TO511OCTETS_FRAMES_GOOD_BAD
Transmit Octet Count for Good and Bad 256 to 511 Bytes Frames(130_H) Reset Value: $0000\ 0000_H$

Field	Bits	Type	Description
TX256_511OCTGB	[31:0]	r	**This field indicates the number of transmitted good and bad frames with length between 256 and 511 (inclusive) bytes, exclusive of preamble and retried frames.**

TX_512TO1023OCTETS_FRAMES_GOOD_BAD

This register maintains the number of transmitted good and bad frames with length between 512 and 1,023 (inclusive) bytes, exclusive of preamble and retried frames.

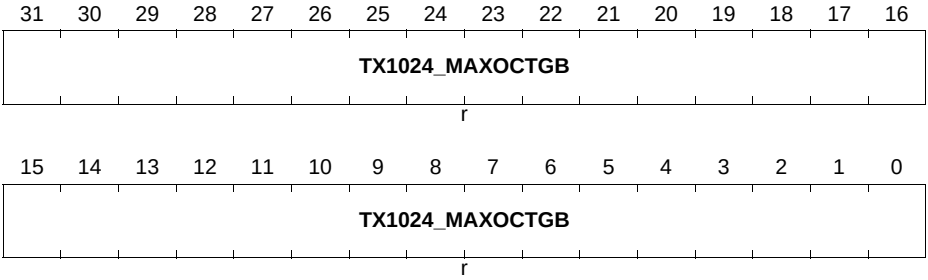
ETH0_TX_512TO1023OCTETS_FRAMES_GOOD_BAD
Transmit Octet Count for Good and Bad 512 to 1023 Bytes Frames(134_H) Reset Value: $0000\ 0000_H$

Field	Bits	Type	Description
TX512_1023OCTGB	[31:0]	r	**This field indicates the number of transmitted good and bad frames with length between 512 and 1,023 (inclusive) bytes, exclusive of preamble and retried frames.**

TX_1024TOMAXOCTETS_FRAMES_GOOD_BAD

This register maintains the number of transmitted good and bad frames with length between 1,024 and maxsize (inclusive) bytes, exclusive of preamble and retried frames.

ETH0_TX_1024TOMAXOCTETS_FRAMES_GOOD_BAD
Transmit Octet Count for Good and Bad 1024 to Maxsize Bytes Frames(138_H)
Reset Value: $0000\ 0000_H$

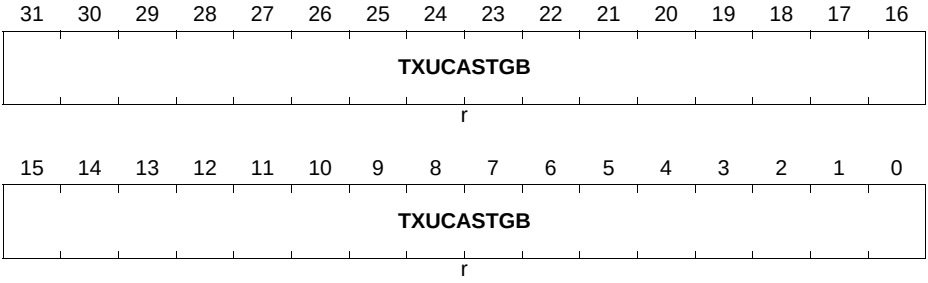


Field	Bits	Type	Description
TX1024_MAXOCTGB	[31:0]	r	**This field indicates the number of good and bad frames transmitted with length between 1,024 and maxsize (inclusive) bytes, exclusive of preamble and retried frames.**

TX_UNICAST_FRAMES_GOOD_BAD

This register maintains the number of transmitted good and bad unicast frames.

ETH0_TX_UNICAST_FRAMES_GOOD_BAD
Transmit Frame Count for Good and Bad Unicast Frames ($13C_H$) Reset Value: $0000\ 0000_H$

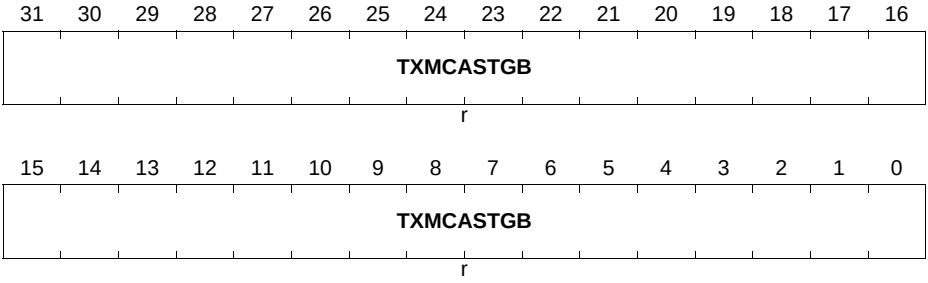


Field	Bits	Type	Description
TXUCASTGB	[31:0]	r	**This field indicates the number of transmitted good and bad unicast frames.**

TX_MULTICAST_FRAMES_GOOD_BAD

This register maintains the number of transmitted good and bad multicast frames.

ETH0_TX_MULTICAST_FRAMES_GOOD_BAD
Transmit Frame Count for Good and Bad Multicast Frames(140_H) Reset Value: $0000\ 0000_H$

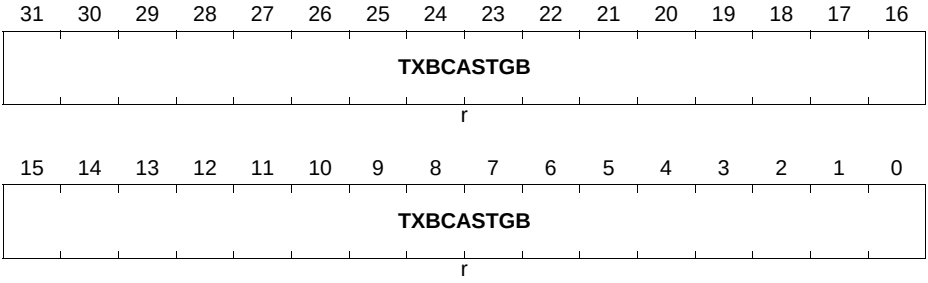


Field	Bits	Type	Description
TXMCASTGB	[31:0]	r	**This field indicates the number of transmitted good and bad multicast frames.**

TX_BROADCAST_FRAMES_GOOD_BAD

This register maintains the number of transmitted good and bad broadcast frames.

ETH0_TX_BROADCAST_FRAMES_GOOD_BAD
Transmit Frame Count for Good and Bad Broadcast Frames(144_H) Reset Value: $0000\ 0000_H$

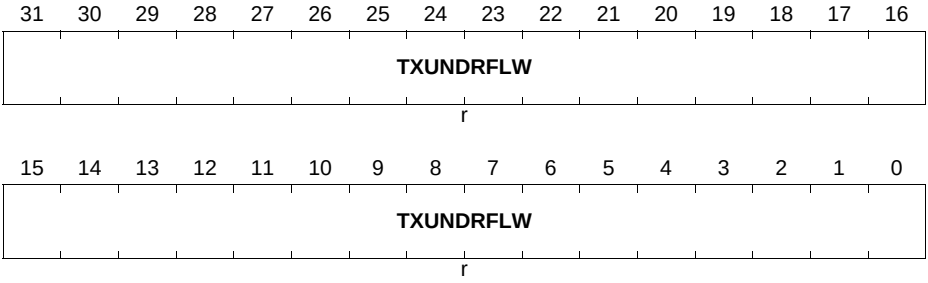


Field	Bits	Type	Description
TXBCASTGB	[31:0]	r	**This field indicates the number of transmitted good and bad broadcast frames.**

TX_UNDERFLOW_ERROR_FRAMES

This register maintains the number of frames aborted because of frame underflow error.

ETH0_TX_UNDERFLOW_ERROR_FRAMES
Transmit Frame Count for Underflow Error Frames (148_H) Reset Value: 0000 0000_H

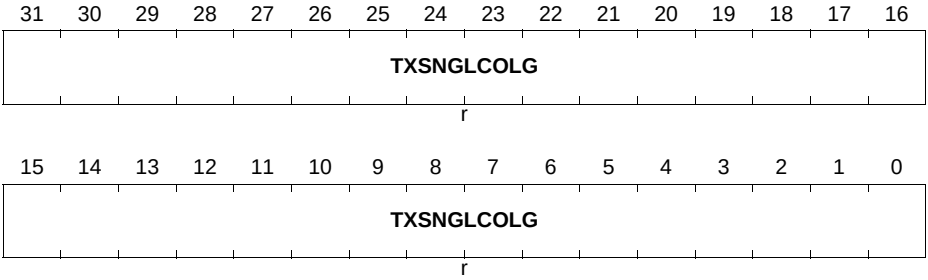


Field	Bits	Type	Description
TXUNDRFLW	[31:0]	r	**This field indicates the number of frames aborted because of frame underflow error.**

TX_SINGLE_COLLISION_GOOD_FRAMES

This register maintains the number of successfully transmitted frames after a single collision in the half-duplex mode.

ETH0_TX_SINGLE_COLLISION_GOOD_FRAMES
Transmit Frame Count for Frames Transmitted after Single Collision($14C_H$) Reset Value: $0000\ 0000_H$

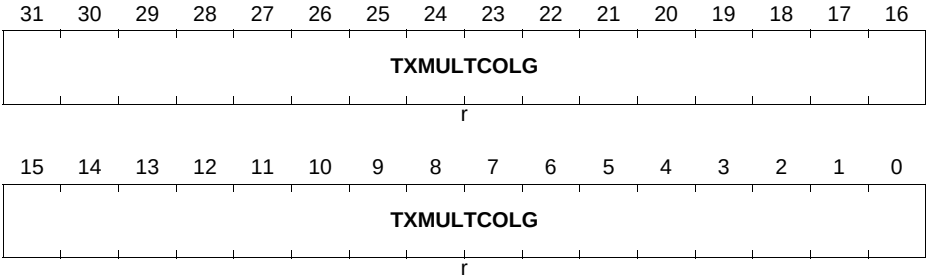


Field	Bits	Type	Description
TXSNGLCOLG	[31:0]	r	**This field indicates the number of successfully transmitted frames after a single collision in the half-duplex mode.**

TX_MULTIPLE_COLLISION_GOOD_FRAMES

This register maintains the number of successfully transmitted frames after multiple collisions in the half-duplex mode.

ETH0_TX_MULTIPLE_COLLISION_GOOD_FRAMES
Transmit Frame Count for Frames Transmitted after Multiple Collision(150_H)
Reset Value: $0000\ 0000_H$

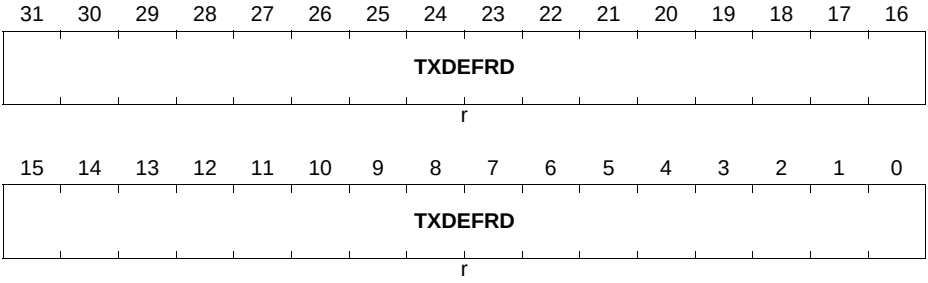


Field	Bits	Type	Description
TXMULTCOLG	[31:0]	r	**This field indicates the number of successfully transmitted frames after multiple collisions in the half-duplex mode.**

TX_DEFERRED_FRAMES

This register maintains the number of successfully transmitted frames after a deferral in the half-duplex mode.

ETH0_TX_DEFERRED_FRAMES
Tx Deferred Frames Register **(154_H)** **Reset Value: $0000\ 0000_H$**

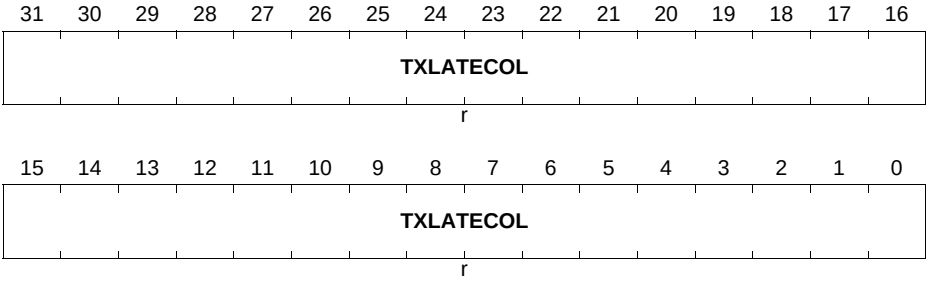


Field	Bits	Type	Description
TXDEFRD	[31:0]	r	**This field indicates the number of successfully transmitted frames after a deferral in the half-duplex mode.**

TX_LATE_COLLISION_FRAMES

This register maintains the number of frames aborted because of late collision error.

ETH0_TX_LATE_COLLISION_FRAMES
Transmit Frame Count for Late Collision Error Frames(158_H) Reset Value: $0000\ 0000_H$

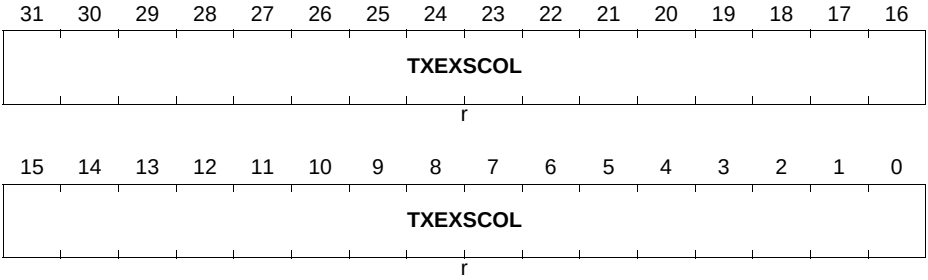


Field	Bits	Type	Description
TXLATECOL	[31:0]	r	**This field indicates the number of frames aborted because of late collision error.**

TX_EXCESSIVE_COLLISION_FRAMES

This register maintains the number of frames aborted because of excessive (16) collision error.

ETH0_TX_EXCESSIVE_COLLISION_FRAMES
Transmit Frame Count for Excessive Collision Error Frames(15C$_H$) Reset Value: 0000 0000$_H$

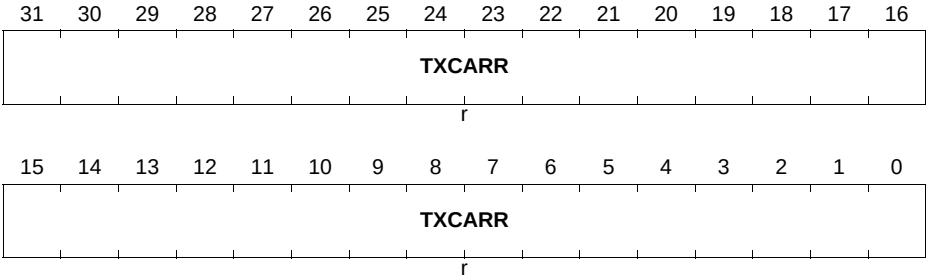


Field	Bits	Type	Description
TXEXSCOL	[31:0]	r	**This field indicates the number of frames aborted because of excessive (16) collision error.**

TX_CARRIER_ERROR_FRAMES

This register maintains the number of frames aborted because of carrier sense error (no carrier or loss of carrier).

ETH0_TX_CARRIER_ERROR_FRAMES
Transmit Frame Count for Carrier Sense Error Frames(160_H) Reset Value: $0000\ 0000_H$

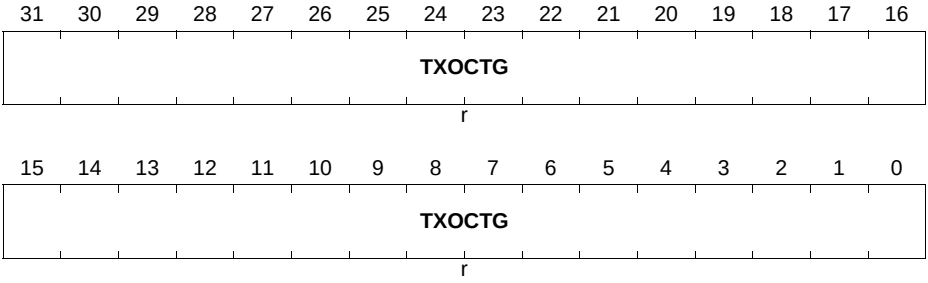


Field	Bits	Type	Description
TXCARR	[31:0]	r	**This field indicates the number of frames aborted because of carrier sense error (no carrier or loss of carrier).**

TX_OCTET_COUNT_GOOD

This register maintains the number of bytes transmitted, exclusive of preamble, in good frames.

ETH0_TX_OCTET_COUNT_GOOD
Tx Octet Count Good Register **(164_H)** **Reset Value: $0000\ 0000_H$**

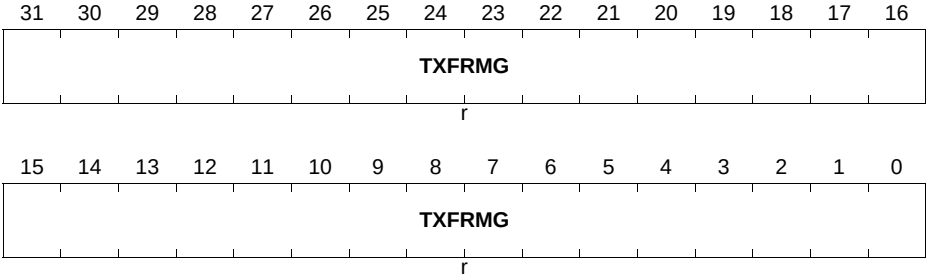


Field	Bits	Type	Description
TXOCTG	[31:0]	r	**This field indicates the number of bytes transmitted, exclusive of preamble, in good frames.**

TX_FRAME_COUNT_GOOD

This register maintains the number of transmitted good frames, exclusive of preamble.

ETH0_TX_FRAME_COUNT_GOOD
Tx Frame Count Good Register (168_H) Reset Value: $0000\ 0000_H$

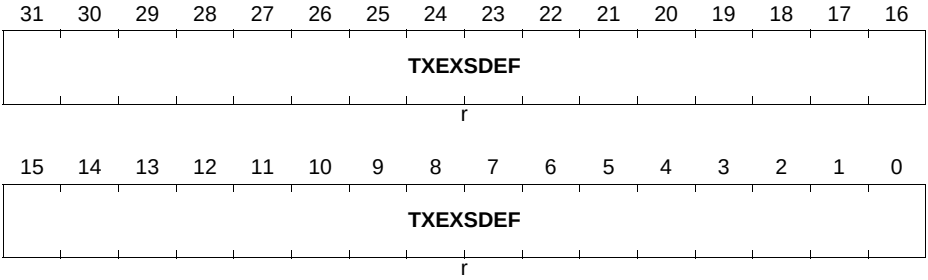


Field	Bits	Type	Description
TXFRMG	[31:0]	r	**This field indicates the number of transmitted good frames, exclusive of preamble.**

TX_EXCESSIVE_DEFERRAL_ERROR

This register maintains the number of frames aborted because of excessive deferral error, that is, frames deferred for more than two max-sized frame times.

ETH0_TX_EXCESSIVE_DEFERRAL_ERROR
Transmit Frame Count for Excessive Deferral Error Frames(16C$_H$) Reset Value: 0000 0000$_H$



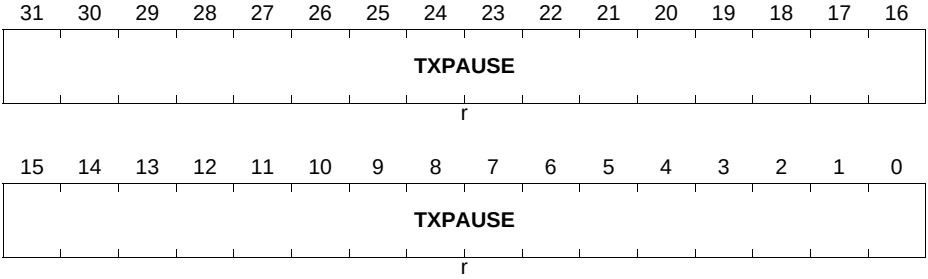
Field	Bits	Type	Description
TXEXSDEF	[31:0]	r	**This field indicates the number of frames aborted because of excessive deferral error, that is, frames deferred for more than two max-sized frame times.**

TX_PAUSE_FRAMES

This register maintains the number of transmitted good PAUSE frames.

ETH0_TX_PAUSE_FRAMES

Transmit Frame Count for Good PAUSE Frames(170_H) **Reset Value: $0000\ 0000_H$**

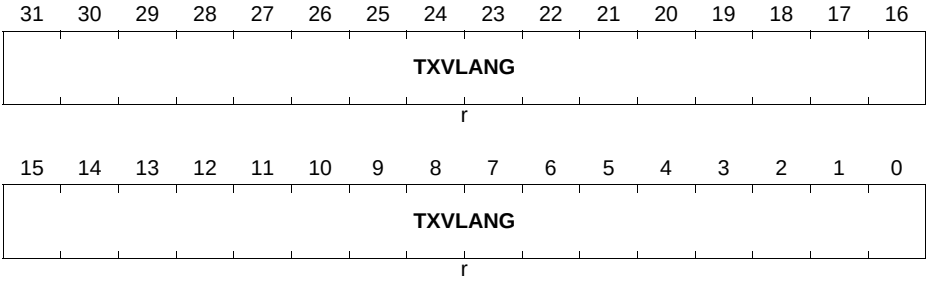


Field	Bits	Type	Description
TXPAUSE	[31:0]	r	**This field indicates the number of transmitted good PAUSE frames.**

TX_VLAN_FRAMES_GOOD

This register maintains the number of transmitted good VLAN frames, exclusive of retried frames.

ETH0_TX_VLAN_FRAMES_GOOD
Transmit Frame Count for Good VLAN Frames(174_H) **Reset Value: $0000\ 0000_H$**

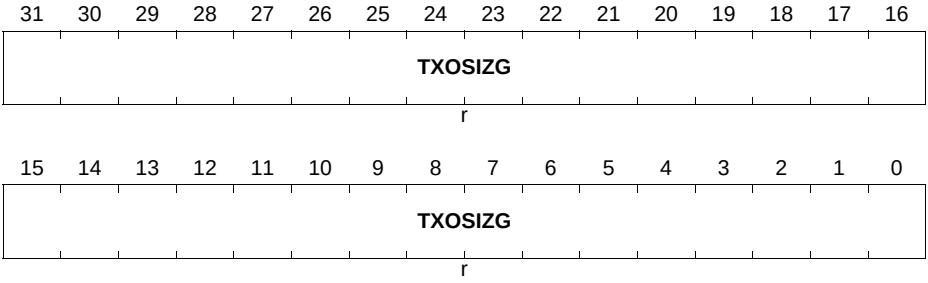


Field	Bits	Type	Description
TXVLANG	[31:0]	r	**This register maintains the number of transmitted good VLAN frames, exclusive of retried frames.**

TX_OSIZE_FRAMES_GOOD

This register maintains the number of transmitted good Oversize frames, exclusive of retried frames.

ETH0_TX_OSIZE_FRAMES_GOOD
Transmit Frame Count for Good Oversize Frames(178_H) Reset Value: $0000\ 0000_H$

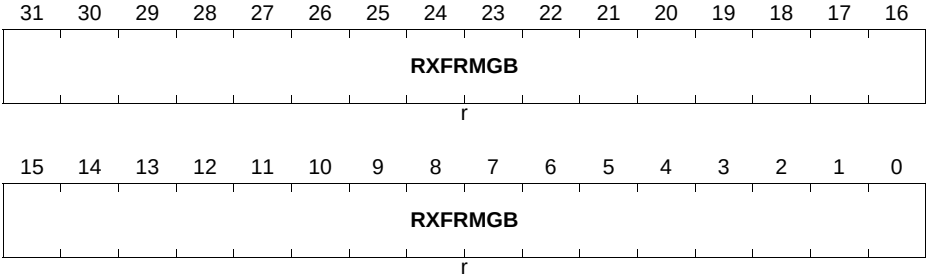


Field	Bits	Type	Description
TXOSIZG	[31:0]	r	**This field indicates the number of frames transmitted without errors and with length greater than the maxsize (1,518 or 1,522 bytes for VLAN tagged frames; 2000 bytes if enabled by setting MAC Configuration.2KPE).**

RX_FRAMES_COUNT_GOOD_BAD

This register maintains the number of received good and bad frames.

ETH0_RX_FRAMES_COUNT_GOOD_BAD
Receive Frame Count for Good and Bad Frames(180_H) Reset Value: $0000\ 0000_H$

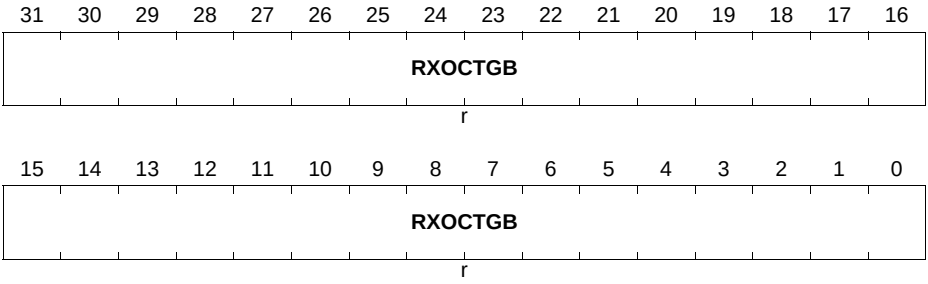


Field	Bits	Type	Description
RXFRMGB	[31:0]	r	**This field indicates the number of received good and bad frames.**

RX_OCTET_COUNT_GOOD_BAD

This register maintains the number of bytes received, exclusive of preamble, in good and bad frames.

ETH0_RX_OCTET_COUNT_GOOD_BAD
Receive Octet Count for Good and Bad Frames(184_H) **Reset Value: $0000\ 0000_H$**

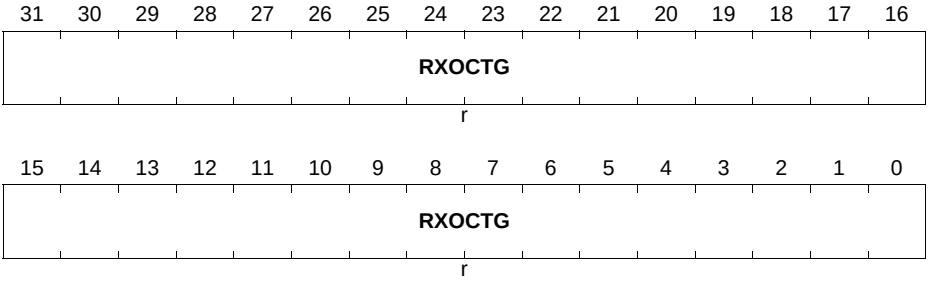


Field	Bits	Type	Description
RXOCTGB	[31:0]	r	**This field indicates the number of bytes received, exclusive of preamble, in good and bad frames.**

RX_OCTET_COUNT_GOOD

This register maintains the number of bytes received, exclusive of preamble, only in good frames.

ETH0_RX_OCTET_COUNT_GOOD
Rx Octet Count Good Register (188$_H$) Reset Value: 0000 0000$_H$

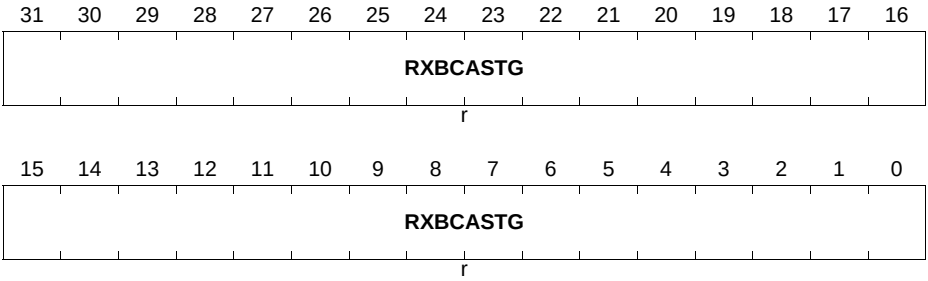


Field	Bits	Type	Description
RXOCTG	[31:0]	r	**This field indicates the number of bytes received, exclusive of preamble, only in good frames.**

RX_BROADCAST_FRAMES_GOOD

This register maintains the number of received good broadcast frames.

ETH0_RX_BROADCAST_FRAMES_GOOD
Receive Frame Count for Good Broadcast Frames(18C$_H$) Reset Value: 0000 0000$_H$

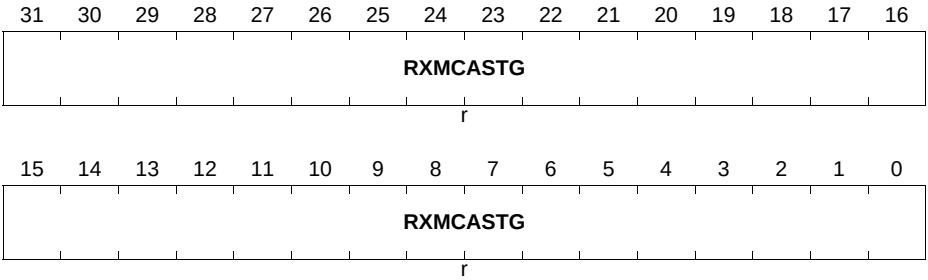


Field	Bits	Type	Description
RXBCASTG	[31:0]	r	**This field indicates the number of received good broadcast frames.**

RX_MULTICAST_FRAMES_GOOD

This register maintains the number of received good multicast frames.

ETH0_RX_MULTICAST_FRAMES_GOOD
Receive Frame Count for Good Multicast Frames(190_H) Reset Value: $0000\ 0000_H$

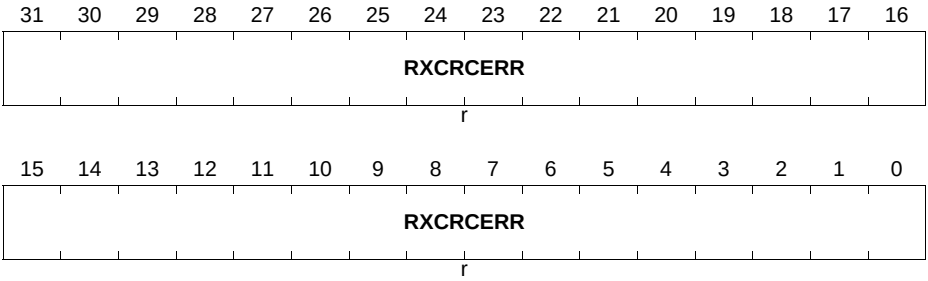


Field	Bits	Type	Description
RXMCASTG	[31:0]	r	**This field indicates the number of received good multicast frames.**

RX_CRC_ERROR_FRAMES

This register maintains the number of frames received with CRC error.

ETH0_RX_CRC_ERROR_FRAMES
Receive Frame Count for CRC Error Frames(194_H) **Reset Value: $0000\ 0000_H$**

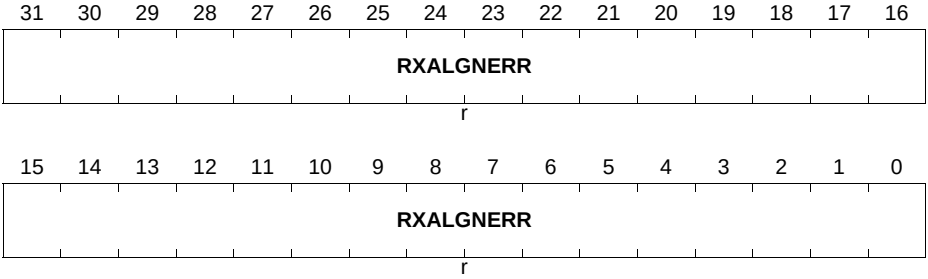


Field	Bits	Type	Description
RXCRCERR	[31:0]	r	**This field indicates the number of frames received with CRC error.**

RX_ALIGNMENT_ERROR_FRAMES

This register maintains the number of frames received with alignment (dribble) error.

ETH0_RX_ALIGNMENT_ERROR_FRAMES
Receive Frame Count for Alignment Error Frames(198_H) Reset Value: $0000\ 0000_H$



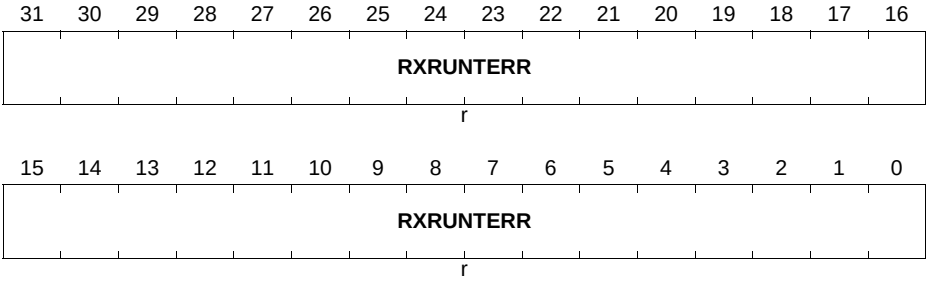
Field	Bits	Type	Description
RXALGNERR	[31:0]	r	**This field indicates the number of frames received with alignment (dribble) error.**

RX_RUNT_ERROR_FRAMES

This register maintains the number of frames received with runt error(<64 bytes and CRC error).

ETH0_RX_RUNT_ERROR_FRAMES

Receive Frame Count for Runt Error Frames(19C$_H$) **Reset Value: 0000 0000$_H$**



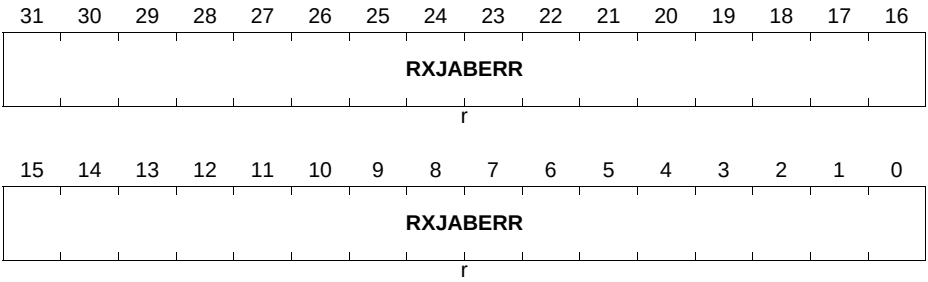
Field	Bits	Type	Description
RXRUNTERR	[31:0]	r	**This field indicates the number of frames received with runt error(<64 bytes and CRC error).**

RX_JABBER_ERROR_FRAMES

This register maintains the number of giant frames received with length (including CRC) greater than 1,518 bytes (1,522 bytes for VLAN tagged) and with CRC error. If Jumbo Frame mode is enabled, then frames of length greater than 9,018 bytes (9,022 for VLAN tagged) are considered as giant frames.

ETH0_RX_JABBER_ERROR_FRAMES

Receive Frame Count for Jabber Error Frames($1A0_H$) **Reset Value: $0000\ 0000_H$**

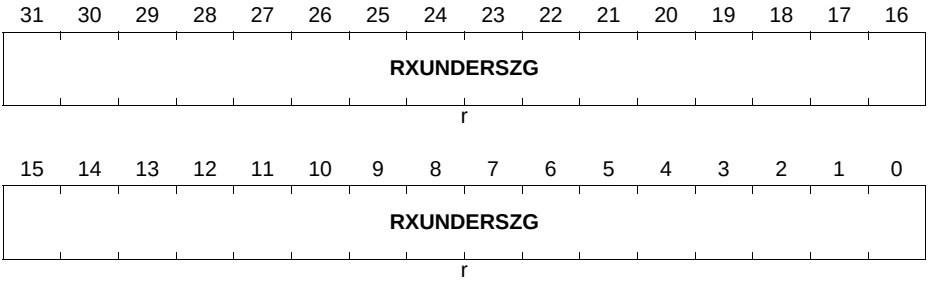


Field	Bits	Type	Description
RXJABERR	[31:0]	r	**This field indicates the number of giant frames received with length (including CRC) greater than 1,518 bytes (1,522 bytes for VLAN tagged) and with CRC error. If Jumbo Frame mode is enabled, then frames of length greater than 9,018 bytes (9,022 for VLAN tagged) are considered as giant frames.**

RX_UNDERSIZE_FRAMES_GOOD

This register maintains the number of frames received with length less than 64 bytes and without errors.

ETH0_RX_UNDERSIZE_FRAMES_GOOD
Receive Frame Count for Undersize Frames(1A4$_H$) **Reset Value: 0000 0000$_H$**

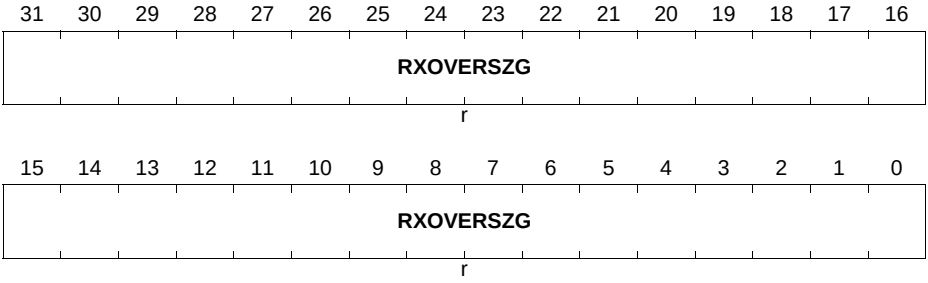


Field	Bits	Type	Description
RXUNDERSZG	[31:0]	r	**This field indicates the number of frames received with length less than 64 bytes and without errors.**

RX_OVERSIZE_FRAMES_GOOD

This register maintains the number of frames received with length greater than the maxsize (1,518 or 1,522 for VLAN tagged frames) and without errors.

ETH0_RX_OVERSIZE_FRAMES_GOOD
Rx Oversize Frames Good Register (1A8$_H$) **Reset Value: 0000 0000$_H$**

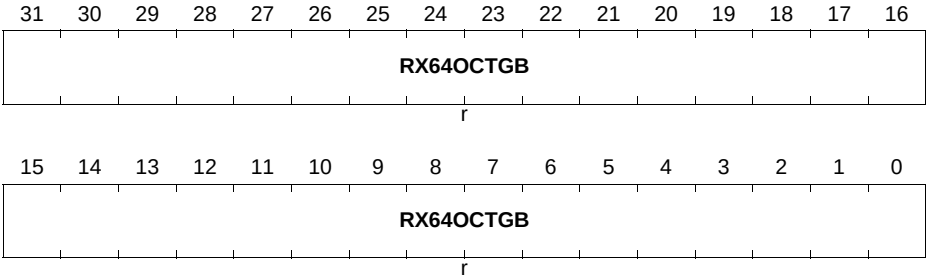


Field	Bits	Type	Description
RXOVERSZG	[31:0]	r	**This field indicates the number of frames received without errors, with length greater than the maxsize (1,518 or 1,522 for VLAN tagged frames; 2,000 bytes if enabled by setting MAC Configuration.2KPE).**

RX_64OCTETS_FRAMES_GOOD_BAD

This register maintains the number of received good and bad frames with length 64 bytes, exclusive of preamble.

ETH0_RX_64OCTETS_FRAMES_GOOD_BAD
Receive Frame Count for Good and Bad 64 Byte Frames(1AC$_H$) Reset Value: 0000 0000$_H$

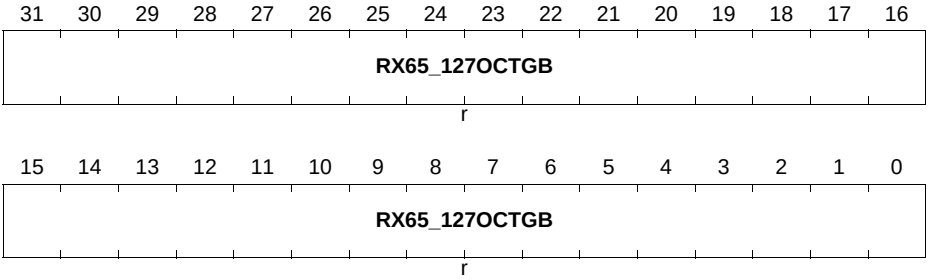


Field	Bits	Type	Description
RX64OCTGB	[31:0]	r	**This field indicates the number of received good and bad frames with length 64 bytes, exclusive of preamble.**

RX_65TO127OCTETS_FRAMES_GOOD_BAD

This register maintains the number of received good and bad frames received with length between 65 and 127 (inclusive) bytes, exclusive of preamble.

ETH0_RX_65TO127OCTETS_FRAMES_GOOD_BAD
Receive Frame Count for Good and Bad 65 to 127 Bytes Frames($1B0_H$) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
RX65_127 OCTGB	[31:0]	r	**This field indicates the number of received good and bad frames received with length between 65 and 127 (inclusive) bytes, exclusive of preamble.**

RX_128TO255OCTETS_FRAMES_GOOD_BAD

This register maintains the number of received good and bad frames with length between 128 and 255 (inclusive) bytes, exclusive of preamble.

ETH0_RX_128TO255OCTETS_FRAMES_GOOD_BAD
Receive Frame Count for Good and Bad 128 to 255 Bytes Frames($1B4_H$) Reset Value: $0000\ 0000_H$

Field	Bits	Type	Description
RX128_255OCTGB	[31:0]	r	**This field indicates the number of received good and bad frames with length between 128 and 255 (inclusive) bytes, exclusive of preamble.**

RX_256TO511OCTETS_FRAMES_GOOD_BAD

This register maintains the number of received good and bad frames with length between 256 and 511 (inclusive) bytes, exclusive of preamble.

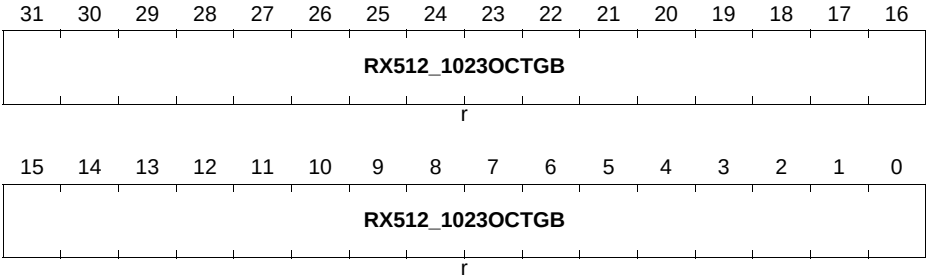
ETH0_RX_256TO511OCTETS_FRAMES_GOOD_BAD
Receive Frame Count for Good and Bad 256 to 511 Bytes Frames(1B8$_H$) Reset Value: 0000 0000$_H$

Field	Bits	Type	Description
RX256_511OCTGB	[31:0]	r	**This field indicates the number of received good and bad frames with length between 256 and 511 (inclusive) bytes, exclusive of preamble.**

RX_512TO1023OCTETS_FRAMES_GOOD_BAD

This register maintains the number of received good and bad frames with length between 512 and 1,023 (inclusive) bytes, exclusive of preamble.

ETH0_RX_512TO1023OCTETS_FRAMES_GOOD_BAD
Receive Frame Count for Good and Bad 512 to 1,023 Bytes Frames(1BC$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
RX512_1023OCTGB	[31:0]	r	**This field indicates the number of received good and bad frames with length between 512 and 1,023 (inclusive) bytes, exclusive of preamble.**

RX_1024TOMAXOCTETS_FRAMES_GOOD_BAD

This register maintains the number of received good and bad frames with length between 1,024 and maxsize (inclusive) bytes, exclusive of preamble.

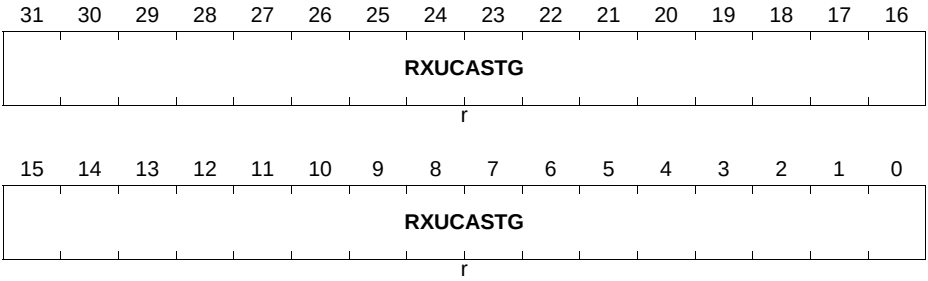
ETH0_RX_1024TOMAXOCTETS_FRAMES_GOOD_BAD
Receive Frame Count for Good and Bad 1,024 to Maxsize Bytes Frames($1C0_H$)
Reset Value: $0000\ 0000_H$

Field	Bits	Type	Description
RX1024_MAXOCTGB	[31:0]	r	**This field indicates the number of received good and bad frames with length between 1,024 and maxsize (inclusive) bytes, exclusive of preamble and retried frames.**

RX_UNICAST_FRAMES_GOOD

This register maintains the number of received good unicast frames.

ETH0_RX_UNICAST_FRAMES_GOOD
Receive Frame Count for Good Unicast Frames($1C4_H$) **Reset Value: $0000\ 0000_H$**

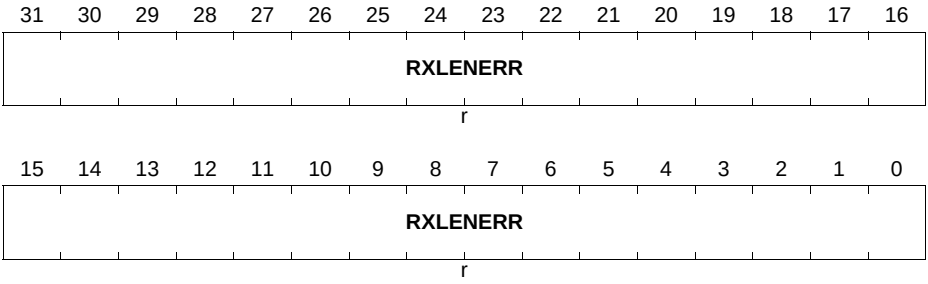


Field	Bits	Type	Description
RXUCASTG	[31:0]	r	**This field indicates the number of received good unicast frames.**

RX_LENGTH_ERROR_FRAMES

This register maintains the number of frames received with length error (Length type field not equal to frame size) for all frames with valid length field.

ETH0_RX_LENGTH_ERROR_FRAMES
Receive Frame Count for Length Error Frames($1C8_H$) **Reset Value: $0000\ 0000_H$**

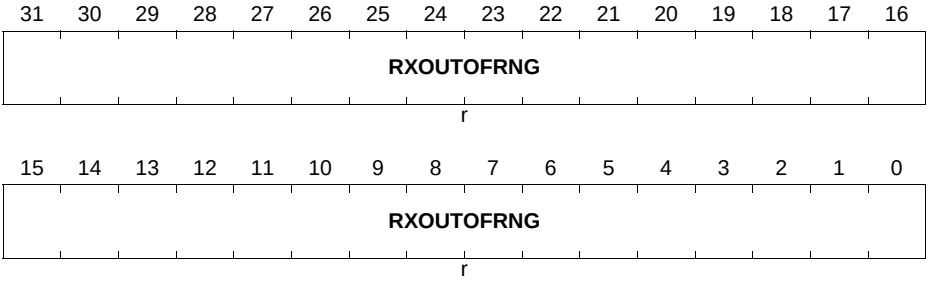


Field	Bits	Type	Description
RXLENERR	[31:0]	r	**This field indicates the number of frames received with length error (Length type field not equal to frame size) for all frames with valid length field.**

RX_OUT_OF_RANGE_TYPE_FRAMES

This register maintains the number of received frames with length field not equal to the valid frame size (greater than 1,500 but less than 1,536).

ETH0_RX_OUT_OF_RANGE_TYPE_FRAMES
Receive Frame Count for Out of Range Frames($1CC_H$) **Reset Value: $0000\ 0000_H$**

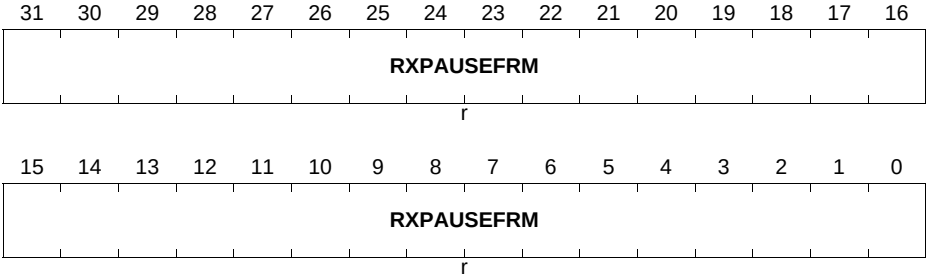


Field	Bits	Type	Description
RXOUTOFRNG	[31:0]	r	**This field indicates the number of received frames with length field not equal to the valid frame size (greater than 1,500 but less than 1,536).**

RX_PAUSE_FRAMES

This register maintains the number of received good and valid PAUSE frames.

ETH0_RX_PAUSE_FRAMES
Receive Frame Count for PAUSE Frames($1D0_H$) **Reset Value: $0000\ 0000_H$**

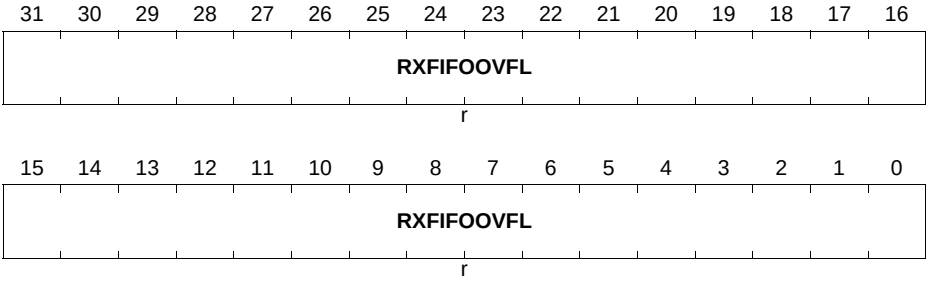


Field	Bits	Type	Description
RXPAUSEFRM	[31:0]	r	**This field indicates the number of received good and valid PAUSE frames.**

RX_FIFO_OVERFLOW_FRAMES

This register maintains the number of received frames missed because of FIFO overflow.

ETH0_RX_FIFO_OVERFLOW_FRAMES
Receive Frame Count for FIFO Overflow Frames($1D4_H$) Reset Value: $0000\ 0000_H$

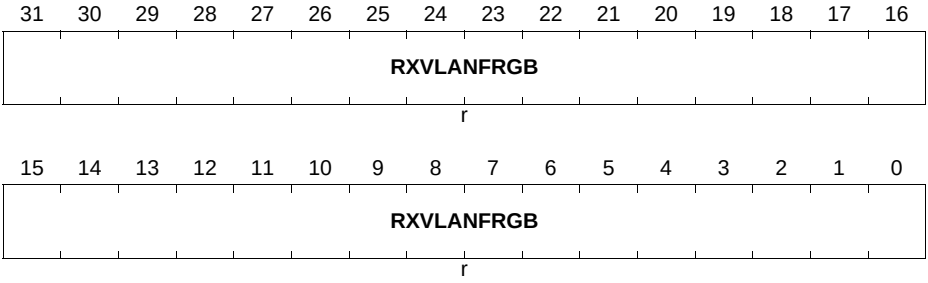


Field	Bits	Type	Description
RXFIFOOVFL	[31:0]	r	**This field indicates the number of received frames missed because of FIFO overflow.**

RX_VLAN_FRAMES_GOOD_BAD

This register maintains the number of received good and bad VLAN frames.

ETH0_RX_VLAN_FRAMES_GOOD_BAD
Receive Frame Count for Good and Bad VLAN Frames($1D8_H$) Reset Value: $0000\ 0000_H$

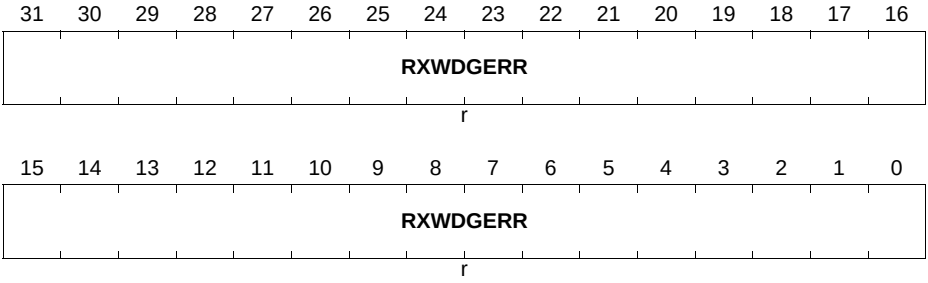


Field	Bits	Type	Description
RXVLANFRGB	[31:0]	r	**This field indicates the number of received good and bad VLAN frames.**

RX_WATCHDOG_ERROR_FRAMES

This register maintains the number of frames received with error because of the watchdog timeout error (frames with more than 2,048 bytes data load).

ETH0_RX_WATCHDOG_ERROR_FRAMES
Receive Frame Count for Watchdog Error Frames(1DC$_H$) Reset Value: 0000 0000$_H$

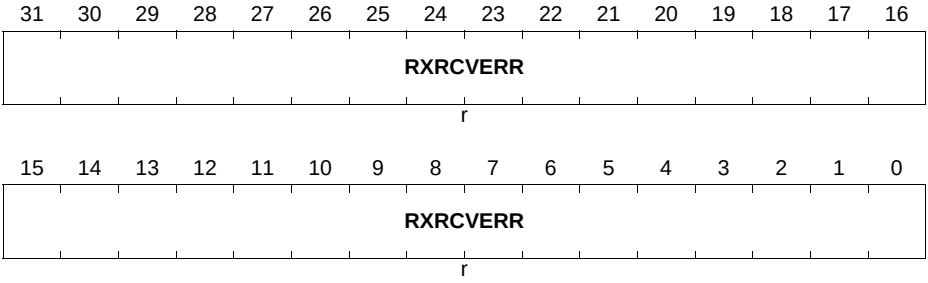


Field	Bits	Type	Description
RXWDGERR	[31:0]	r	**This field indicates the number of frames received with error because of the watchdog timeout error (frames with more than 2,048 bytes data load).**

RX_RECEIVE_ERROR_FRAMES

This register maintains the number of frames received with error because of the MII RXER error.

ETH0_RX_RECEIVE_ERROR_FRAMES
Receive Frame Count for Receive Error Frames($1E0_H$) **Reset Value: $0000\ 0000_H$**

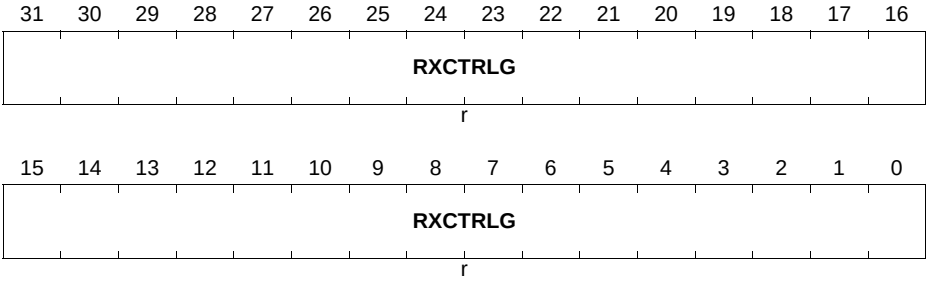


Field	Bits	Type	Description
RXRCVERR	[31:0]	r	**This field indicates the number of frames received with error because of the watchdog timeout error (frames with more than 2,048 bytes data load).**

RX_CONTROL_FRAMES_GOOD

This register maintains the number of godd control frames received.

ETH0_RX_CONTROL_FRAMES_GOOD
Receive Frame Count for Good Control Frames Frames($1E4_H$) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
RXCTRLG	[31:0]	r	**This field indicates the number of frames received with error because of the watchdog timeout error (frames with more than 2,048 bytes data load).**

MMC_IPC_RECEIVE_INTERRUPT_MASK

This register maintains the mask for the interrupt generated from the receive IPC statistic counters. This register is 32-bits wide.

ETH0_MMC_IPC_RECEIVE_INTERRUPT_MASK
MMC Receive Checksum Offload Interrupt Mask Register(200_H) Reset Value: 0000 0000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved_31_30		RXICMPEROIM	RXICMPGOIM	RXTCPEROIM	RXTCPGOIM	RXUDPEROIM	RXUDPGOIM	RXIPV6NOPAYOIM	RXIPV6HEROIM	RXIPV6GOIM	RXIPV4UDSBLOIM	RXIPV4FRAGOIM	RXIPV4NOPAYOIM	RXIPV4HEROIM	RXIPV4GOIM
r		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved_15_14		RXICMPERFIM	RXICMPGFIM	RXTCPERFIM	RXTCPGFIM	RXUDPERFIM	RXUDPGFIM	RXIPV6NOPAYFIM	RXIPV6HERFIM	RXIPV6GFIM	RXIPV4UDSBLFIM	RXIPV4FRAGFIM	RXIPV4NOPAYFIM	RXIPV4HERFIM	RXIPV4GFIM
r		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
RXIPV4GFIM	0	rw	**MMC Receive IPV4 Good Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RXIPV4_GOOD_FRAMES counter reaches half of the maximum value or the maximum value.
RXIPV4HERFIM	1	rw	**MMC Receive IPV4 Header Error Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RXIPV4_HEADER_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
RXIPV4NOPAYFIM	2	rw	**MMC Receive IPV4 No Payload Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RXIPV4_NO_PAYLOAD_FRAMES counter reaches half of the maximum value or the maximum value.

Field	Bits	Type	Description
RXIPV4FRAGFIM	3	rw	**MMC Receive IPV4 Fragmented Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RXIPV4_FRAGMENTED_FRAMES counter reaches half of the maximum value or the maximum value.
RXIPV4UDSBLFIM	4	rw	**MMC Receive IPV4 UDP Checksum Disabled Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RXIPV4_UDP_CHECKSUM_DISABLED_FRAMES counter reaches half of the maximum value or the maximum value.
RXIPV6GFIM	5	rw	**MMC Receive IPV6 Good Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RXIPV6_GOOD_FRAMES counter reaches half of the maximum value or the maximum value.
RXIPV6HERFIM	6	rw	**MMC Receive IPV6 Header Error Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RXIPV6_HEADER_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
RXIPV6NOPAYFIM	7	rw	**MMC Receive IPV6 No Payload Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RXIPV6_NO_PAYLOAD_FRAMES counter reaches half of the maximum value or the maximum value.
RXUDPGFIM	8	rw	**MMC Receive UDP Good Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RXUDP_GOOD_FRAMES counter reaches half of the maximum value or the maximum value.
RXUDPERFIM	9	rw	**MMC Receive UDP Error Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RXUDP_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.

Field	Bits	Type	Description
RXTCPGFIM	10	rw	**MMC Receive TCP Good Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RXTCP_GOOD_FRAMES counter reaches half of the maximum value or the maximum value.
RXTCPERFIM	11	rw	**MMC Receive TCP Error Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RXTCP_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
RXICMPGFIM	12	rw	**MMC Receive ICMP Good Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RXICMP_GOOD_FRAMES counter reaches half of the maximum value or the maximum value.
RXICMPERFIM	13	rw	**MMC Receive ICMP Error Frame Counter Interrupt Mask** Setting this bit masks the interrupt when the RXICMP_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
Reserved_15_14	[15:14]	r	**Reserved**
RXIPV4GOIM	16	rw	**MMC Receive IPV4 Good Octet Counter Interrupt Mask** Setting this bit masks the interrupt when the RXIPV4_GOOD_OCTETS counter reaches half of the maximum value or the maximum value.
RXIPV4HEROIM	17	rw	**MMC Receive IPV4 Header Error Octet Counter Interrupt Mask** Setting this bit masks the interrupt when the RXIPV4_HEADER_ERROR_OCTETS counter reaches half of the maximum value or the maximum value.
RXIPV4NOPAYOIM	18	rw	**MMC Receive IPV4 No Payload Octet Counter Interrupt Mask** Setting this bit masks the interrupt when the RXIPV4_NO_PAYLOAD_OCTETS counter reaches half of the maximum value or the maximum value.

Field	Bits	Type	Description
RXIPV4FRAGOIM	19	rw	**MMC Receive IPV4 Fragmented Octet Counter Interrupt Mask** Setting this bit masks the interrupt when the RXIPV4_FRAGMENTED_OCTETS counter reaches half of the maximum value or the maximum value.
RXIPV4UDSBLOIM	20	rw	**MMC Receive IPV4 UDP Checksum Disabled Octet Counter Interrupt Mask** Setting this bit masks the interrupt when the RXIPV4_UDP_CHECKSUM_DISABLE_OCTETS counter reaches half of the maximum value or the maximum value.
RXIPV6GOIM	21	rw	**MMC Receive IPV6 Good Octet Counter Interrupt Mask** Setting this bit masks the interrupt when the RXIPV6_GOOD_OCTETS counter reaches half of the maximum value or the maximum value.
RXIPV6HEROIM	22	rw	**MMC Receive IPV6 Header Error Octet Counter Interrupt Mask** Setting this bit masks interrupt when the RXIPV6_HEADER_ERROR_OCTETS counter reaches half of the maximum value or the maximum value.
RXIPV6NOPAYOIM	23	rw	**MMC Receive IPV6 No Payload Octet Counter Interrupt Mask** Setting this bit masks the interrupt when the RXIPV6_NO_PAYLOAD_OCTETS counter reaches half of the maximum value or the maximum value.
RXUDPGOIM	24	rw	**MMC Receive UDP Good Octet Counter Interrupt Mask** Setting this bit masks the interrupt when the RXUDP_GOOD_OCTETS counter reaches half of the maximum value or the maximum value.
RXUDPEROIM	25	rw	**MMC Receive UDP Error Octet Counter Interrupt Mask** Setting this bit masks the interrupt when the RXUDP_ERROR_OCTETS counter reaches half of the maximum value or the maximum value.

Field	Bits	Type	Description
RXTCPGOIM	26	rw	**MMC Receive TCP Good Octet Counter Interrupt Mask** Setting this bit masks the interrupt when the RXTCP_GOOD_OCTETS counter reaches half of the maximum value or the maximum value.
RXTCPEROIM	27	rw	**MMC Receive TCP Error Octet Counter Interrupt Mask** Setting this bit masks the interrupt when the RXTCP_ERROR_OCTETS counter reaches half of the maximum value or the maximum value.
RXICMPGOIM	28	rw	**MMC Receive ICMP Good Octet Counter Interrupt Mask** Setting this bit masks the interrupt when the RXICMP_GOOD_OCTETS counter reaches half of the maximum value or the maximum value.
RXICMPEROIM	29	rw	**MMC Receive ICMP Error Octet Counter Interrupt Mask** Setting this bit masks the interrupt when the RXICMP_ERROR_OCTETS counter reaches half of the maximum value or the maximum value.
Reserved_31_30	[31:30]	r	**Reserved**

MMC_IPC_RECEIVE_INTERRUPT

This register maintains the interrupt that the receive IPC statistic counters generate.

ETH0_MMC_IPC_RECEIVE_INTERRUPT
MMC Receive Checksum Offload Interrupt Register(208_H)Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved_31_30		RXICMPEROIS	RXICMPGOIS	RXTCPEROIS	RXTCPGOIS	RXUDPEROIS	RXUDPGOIS	RXIPV6NOPAYOIS	RXIPV6HEROIS	RXIPV6GOIS	RXIPV4UDSBLOIS	RXIPV4FRAGOIS	RXIPV4NOPAYOIS	RXIPV4HEROIS	RXIPV4GOIS
r		r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved_15_14		RXICMPERFIS	RXICMPGFIS	RXTCPERFIS	RXTCPGFIS	RXUDPERFIS	RXUDPGFIS	RXIPV6NOPAYFIS	RXIPV6HERFIS	RXIPV6GFIS	RXIPV4UDSBLFIS	RXIPV4FRAGFIS	RXIPV4NOPAYFIS	RXIPV4HERFIS	RXIPV4GFIS
r		r	r	r	r	r	r	r	r	r	r	r	r	r	r

Field	Bits	Type	Description
RXIPV4GFIS	0	r	**MMC Receive IPV4 Good Frame Counter Interrupt Status** This bit is set when the RXIPV4_GOOD_FRAMES counter reaches half of the maximum value or the maximum value.
RXIPV4HERFIS	1	r	**MMC Receive IPV4 Header Error Frame Counter Interrupt Status** This bit is set when the RXIPV4_HEADER_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
RXIPV4NOPAYFIS	2	r	**MMC Receive IPV4 No Payload Frame Counter Interrupt Status** This bit is set when the RXIPV4_NO_PAYLOAD_FRAMES counter reaches half of the maximum value or the maximum value.
RXIPV4FRAGFIS	3	r	**MMC Receive IPV4 Fragmented Frame Counter Interrupt Status** This bit is set when the RXIPV4_FRAGMENTED_FRAMES counter reaches half of the maximum value or the maximum value.

Field	Bits	Type	Description
RXIPV4UDSBLFIS	4	r	**MMC Receive IPV4 UDP Checksum Disabled Frame Counter Interrupt Status** This bit is set when the RXIPV4_UDP_CHECKSUM_DISABLED_FRAMES counter reaches half of the maximum value or the maximum value.
RXIPV6GFIS	5	r	**MMC Receive IPV6 Good Frame Counter Interrupt Status** This bit is set when the RXIPV6_GOOD_FRAMES counter reaches half of the maximum value or the maximum value.
RXIPV6HERFIS	6	r	**MMC Receive IPV6 Header Error Frame Counter Interrupt Status** This bit is set when the RXIPV6_HEADER_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
RXIPV6NOPAYFIS	7	r	**MMC Receive IPV6 No Payload Frame Counter Interrupt Status** This bit is set when the RXIPV6_NO_PAYLOAD_FRAMES counter reaches half of the maximum value or the maximum value.
RXUDPGFIS	8	r	**MMC Receive UDP Good Frame Counter Interrupt Status** This bit is set when the RXUDP_GOOD_FRAMES counter reaches half of the maximum value or the maximum value.
RXUDPERFIS	9	r	**MMC Receive UDP Error Frame Counter Interrupt Status** This bit is set when the RXUDP_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
RXTCPGFIS	10	r	**MMC Receive TCP Good Frame Counter Interrupt Status** This bit is set when the RXTCP_GOOD_FRAMES counter reaches half of the maximum value or the maximum value.

Field	Bits	Type	Description
RXTCPERFIS	11	r	**MMC Receive TCP Error Frame Counter Interrupt Status** This bit is set when the RXTCP_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
RXICMPGFIS	12	r	**MMC Receive ICMP Good Frame Counter Interrupt Status** This bit is set when the RXICMP_GOOD_FRAMES counter reaches half of the maximum value or the maximum value.
RXICMPERFIS	13	r	**MMC Receive ICMP Error Frame Counter Interrupt Status** This bit is set when the RXICMP_ERROR_FRAMES counter reaches half of the maximum value or the maximum value.
Reserved_15_14	[15:14]	r	**Reserved**
RXIPV4GOIS	16	r	**MMC Receive IPV4 Good Octet Counter Interrupt Status** This bit is set when the RXIPV4_GOOD_OCTETS counter reaches half of the maximum value or the maximum value.
RXIPV4HEROIS	17	r	**MMC Receive IPV4 Header Error Octet Counter Interrupt Status** This bit is set when the RXIPV4_HEADER_ERROR_OCTETS counter reaches half of the maximum value or the maximum value.
RXIPV4NOPAYOIS	18	r	**MMC Receive IPV4 No Payload Octet Counter Interrupt Status** This bit is set when the RXIPV4_NO_PAYLOAD_OCTETS counter reaches half of the maximum value or the maximum value.
RXIPV4FRAGOIS	19	r	**MMC Receive IPV4 Fragmented Octet Counter Interrupt Status** This bit is set when the RXIPV4_FRAGMENTED_OCTETS counter reaches half of the maximum value or the maximum value.

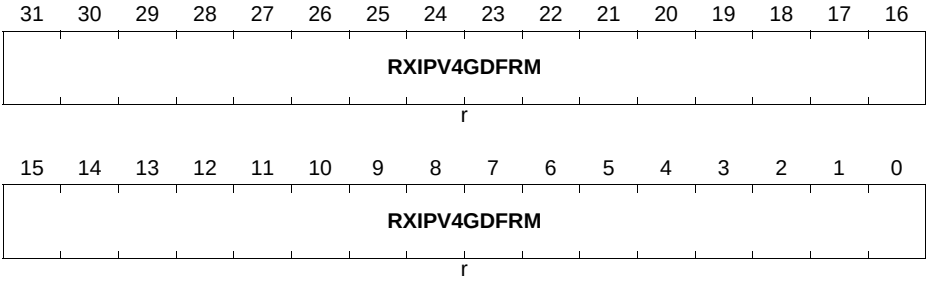
Field	Bits	Type	Description
RXIPV4UDSBLOIS	20	r	**MMC Receive IPV4 UDP Checksum Disabled Octet Counter Interrupt Status** This bit is set when the RXIPV4_UDP_CHECKSUM_DISABLE_OCTETS counter reaches half of the maximum value or the maximum value.
RXIPV6GOIS	21	r	**MMC Receive IPV6 Good Octet Counter Interrupt Status** This bit is set when the RXIPV6_GOOD_OCTETS counter reaches half of the maximum value or the maximum value.
RXIPV6HEROIS	22	r	**MMC Receive IPV6 Header Error Octet Counter Interrupt Status** This bit is set when the RXIPV6_HEADER_ERROR_OCTETS counter reaches half of the maximum value or the maximum value.
RXIPV6NOPAYOIS	23	r	**MMC Receive IPV6 No Payload Octet Counter Interrupt Status** This bit is set when the RXIPV6_NO_PAYLOAD_OCTETS counter reaches half of the maximum value or the maximum value.
RXUDPGOIS	24	r	**MMC Receive UDP Good Octet Counter Interrupt Status** This bit is set when the RXUDP_GOOD_OCTETS counter reaches half of the maximum value or the maximum value.
RXUDPEROIS	25	r	**MMC Receive UDP Error Octet Counter Interrupt Status** This bit is set when the RXUDP_ERROR_OCTETS counter reaches half the maximum value or the maximum value.
RXTCPGOIS	26	r	**MMC Receive TCP Good Octet Counter Interrupt Status** This bit is set when the RXTCP_GOOD_OCTETS counter reaches half the maximum value or the maximum value.

Field	Bits	Type	Description
RXTCPEROIS	27	r	**MMC Receive TCP Error Octet Counter Interrupt Status** This bit is set when the RXTCP_ERROR_OCTETS counter reaches half of the maximum value or the maximum value.
RXICMPGOIS	28	r	**MMC Receive ICMP Good Octet Counter Interrupt Status** This bit is set when the RXICMP_GOOD_OCTETS counter reaches half of the maximum value or the maximum value.
RXICMPEROIS	29	r	**MMC Receive ICMP Error Octet Counter Interrupt Status** This bit is set when the RXICMP_ERROR_OCTETS counter reaches half of the maximum value or the maximum value.
Reserved_31_30	[31:30]	r	**Reserved**

RXIPV4_GOOD_FRAMES

This register maintains the number of good IPv4 datagrams received with the TCP, UDP, or ICMP payload.

ETH0_RXIPV4_GOOD_FRAMES
RxIPv4 Good Frames Register (210$_H$) Reset Value: 0000 0000$_H$



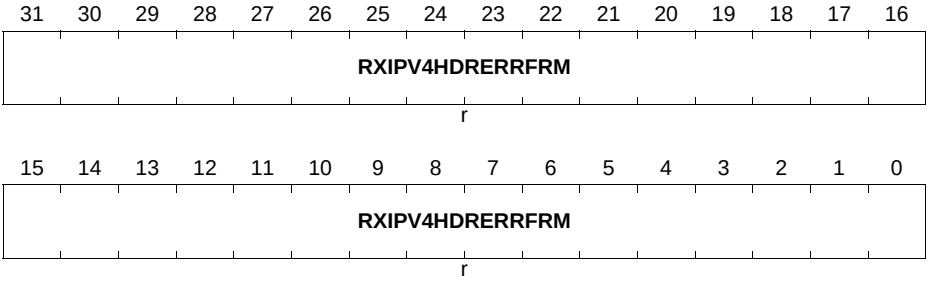
Field	Bits	Type	Description
RXIPV4GDFRM	[31:0]	r	**This field indicates the number of good IPv4 datagrams received with the TCP, UDP, or ICMP payload.**

RXIPV4_HEADER_ERROR_FRAMES

This register maintains the number of IPv4 datagrams received with header errors (checksum, length, or version mismatch).

ETH0_RXIPV4_HEADER_ERROR_FRAMES

Receive IPV4 Header Error Frame Counter Register(214_H)Reset Value: $0000\ 0000_H$

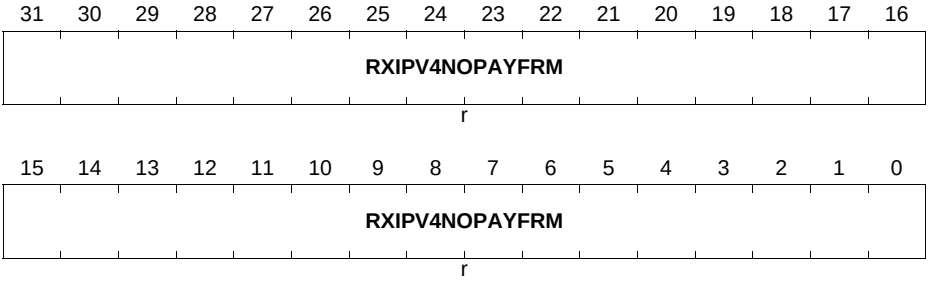


Field	Bits	Type	Description
RXIPV4HDRERRFRM	[31:0]	r	**This field indicates the number of IPv4 datagrams received with header errors (checksum, length, or version mismatch).**

RXIPV4_NO_PAYLOAD_FRAMES

This register maintains the number of received IPv4 datagram frames without a TCP, UDP, or ICMP payload processed by the Checksum engine.

ETH0_RXIPV4_NO_PAYLOAD_FRAMES
Receive IPV4 No Payload Frame Counter Register(218_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
RXIPV4NOPAYFRM	[31:0]	r	**This field indicates the number of IPv4 datagram frames received that did not have a TCP, UDP, or ICMP payload processed by the Checksum engine.**

RXIPV4_FRAGMENTED_FRAMES

This register maintains the number of good IPv4 datagrams received with fragmentation.

ETH0_RXIPV4_FRAGMENTED_FRAMES

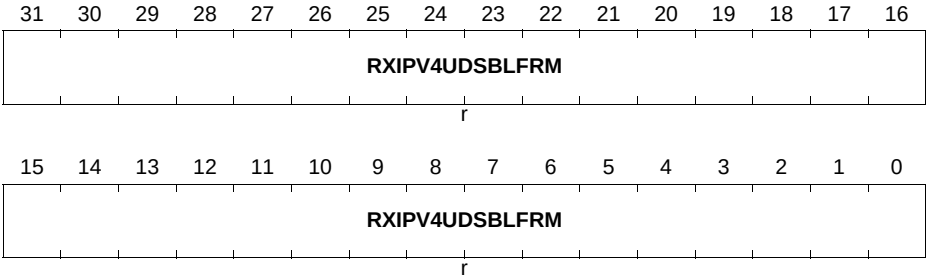
Receive IPV4 Fragmented Frame Counter Register($21C_H$) Reset Value: $0000\ 0000_H$

Field	Bits	Type	Description
RXIPV4FRAGFRM	[31:0]	r	**This field indicates the number of good IPv4 datagrams received with fragmentation.**

RXIPV4_UDP_CHECKSUM_DISABLED_FRAMES

This register maintains the number of received good IPv4 datagrams which have the UDP payload with checksum disabled.

ETH0_RXIPV4_UDP_CHECKSUM_DISABLED_FRAMES
Receive IPV4 UDP Checksum Disabled Frame Counter Register(220_H) Reset Value: $0000\ 0000_H$

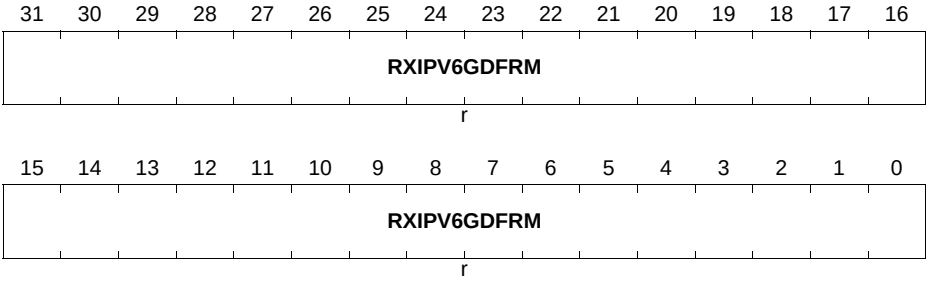


Field	Bits	Type	Description
RXIPV4UDSBLFRM	[31:0]	r	**This field indicates the number of received good IPv4 datagrams which have the UDP payload with checksum disabled.**

RXIPV6_GOOD_FRAMES

This register maintains the number of good IPv6 datagrams received with TCP, UDP, or ICMP payloads.

ETH0_RXIPV6_GOOD_FRAMES
RxIPv6 Good Frames Register (224$_H$) Reset Value: 0000 0000$_H$



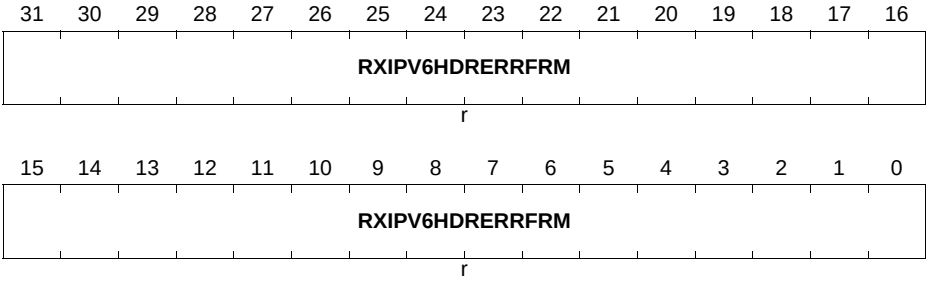
Field	Bits	Type	Description
RXIPV6GDFRM	[31:0]	r	**This field indicates the number of good IPv6 datagrams received with TCP, UDP, or ICMP payloads.**

RXIPV6_HEADER_ERROR_FRAMES

This register maintains the number of IPv6 datagrams received with header errors (length or version mismatch).

ETH0_RXIPV6_HEADER_ERROR_FRAMES

Receive IPV6 Header Error Frame Counter Register(228_H)Reset Value: $0000\ 0000_H$



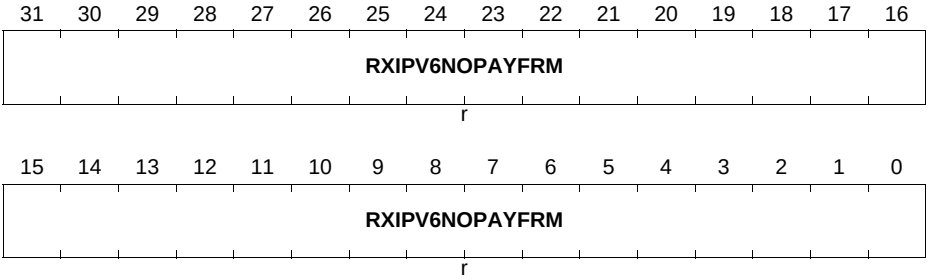
Field	Bits	Type	Description
RXIPV6HDRERRFRM	[31:0]	r	**This field indicates the number of IPv6 datagrams received with header errors (length or version mismatch).**

RXIPV6_NO_PAYLOAD_FRAMES

This register maintains the number of received IPv6 datagram frames without a TCP, UDP, or ICMP payload. This includes all IPv6 datagrams with fragmentation or security extension headers.

ETH0_RXIPV6_NO_PAYLOAD_FRAMES

Receive IPV6 No Payload Frame Counter Register($22C_H$) Reset Value: 0000 0000_H

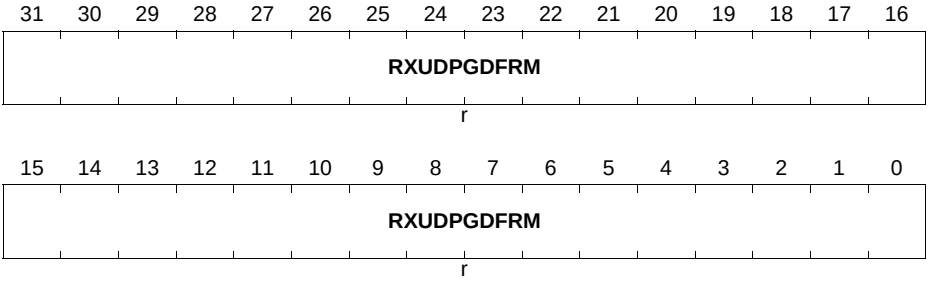


Field	Bits	Type	Description
RXIPV6NOPAYFRM	[31:0]	r	**This field indicates the number of IPv6 datagram frames received that did not have a TCP, UDP, or ICMP payload. This includes all IPv6 datagrams with fragmentation or security extension headers.**

RXUDP_GOOD_FRAMES

This register maintains the number of good IP datagrams with a good UDP payload. This counter is not updated when the counter is incremented.

ETH0_RXUDP_GOOD_FRAMES
RxUDP Good Frames Register (230$_H$) Reset Value: 0000 0000$_H$

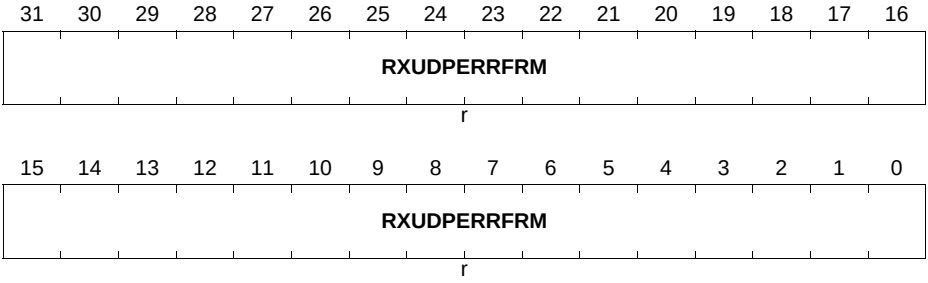


Field	Bits	Type	Description
RXUDPGDFRM	[31:0]	r	**This field indicates the number of good IP datagrams with a good UDP payload. This counter is not updated when the counter is incremented.**

RXUDP_ERROR_FRAMES

This register maintains the number of good IP datagrams whose UDP payload has a checksum error.

ETH0_RXUDP_ERROR_FRAMES
RxUDP Error Frames Register (234$_H$) Reset Value: 0000 0000$_H$

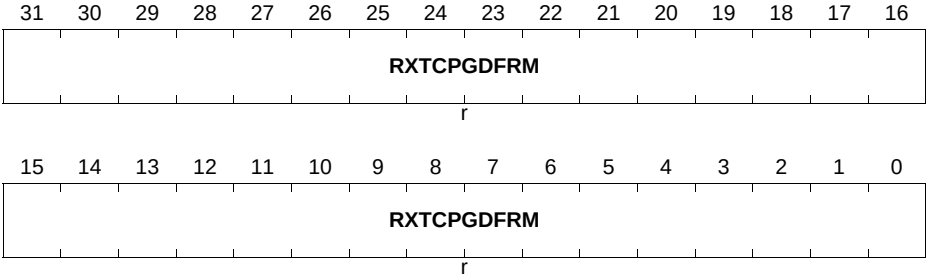


Field	Bits	Type	Description
RXUDPERRFRM	[31:0]	r	**This field indicates the number of good IP datagrams whose UDP payload has a checksum error.**

RXTCP_GOOD_FRAMES

This register maintains the number of good IP datagrams with a good TCP payload.

ETH0_RXTCP_GOOD_FRAMES
RxTCP Good Frames Register (238$_H$) Reset Value: 0000 0000$_H$

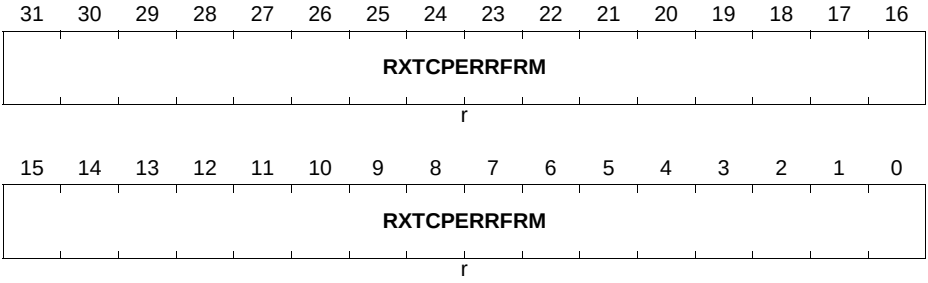


Field	Bits	Type	Description
RXTCPGDFRM	[31:0]	r	**This field indicates the number of good IP datagrams with a good TCP payload.**

RXTCP_ERROR_FRAMES

This register maintains the number of good IP datagrams whose TCP payload has a checksum error.

ETH0_RXTCP_ERROR_FRAMES
RxTCP Error Frames Register (23C$_H$) Reset Value: 0000 0000$_H$

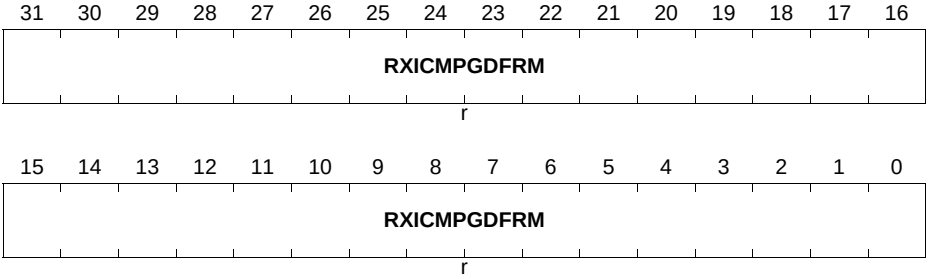


Field	Bits	Type	Description
RXTCPERRFRM	[31:0]	r	**This field indicates the number of good IP datagrams whose TCP payload has a checksum error.**

RXICMP_GOOD_FRAMES

This register maintains the number of good IP datagrams with a good ICMP payload.

ETH0_RXICMP_GOOD_FRAMES
RxICMP Good Frames Register (240$_H$) Reset Value: 0000 0000$_H$

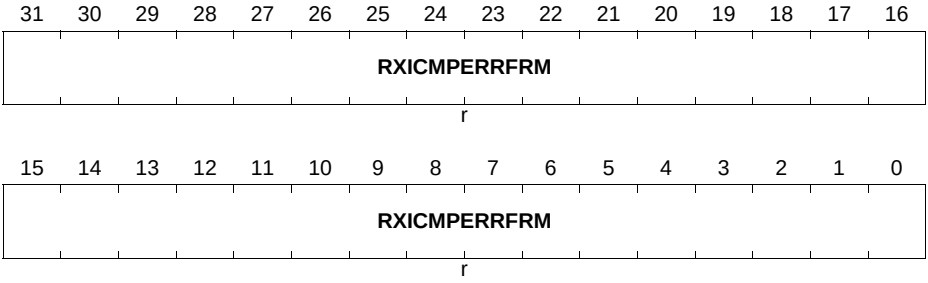


Field	Bits	Type	Description
RXICMPGDFRM	[31:0]	r	**This field indicates the number of good IP datagrams with a good ICMP payload.**

RXICMP_ERROR_FRAMES

This register maintains the number of good IP datagrams whose ICMP payload has a checksum error.

ETH0_RXICMP_ERROR_FRAMES
RxICMP Error Frames Register (244_H) Reset Value: $0000\ 0000_H$

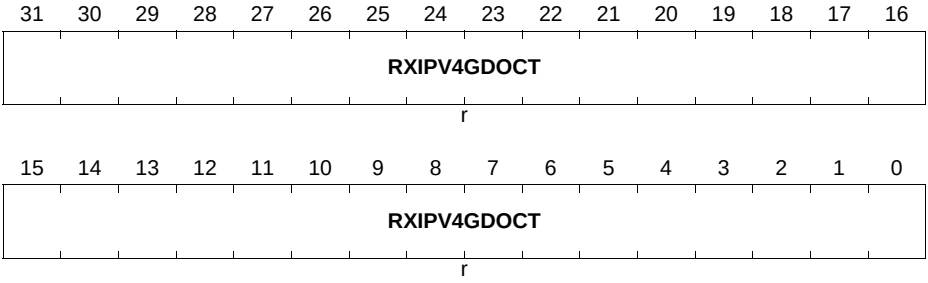


Field	Bits	Type	Description
RXICMPERRFRM	[31:0]	r	**This field indicates the number of good IP datagrams whose ICMP payload has a checksum error.**

RXIPV4_GOOD_OCTETS

This register maintains the number of bytes received in good IPv4 datagrams encapsulating TCP, UDP, or ICMP data.

ETH0_RXIPV4_GOOD_OCTETS
RxIPv4 Good Octets Register **(250_H)** **Reset Value: $0000\ 0000_H$**



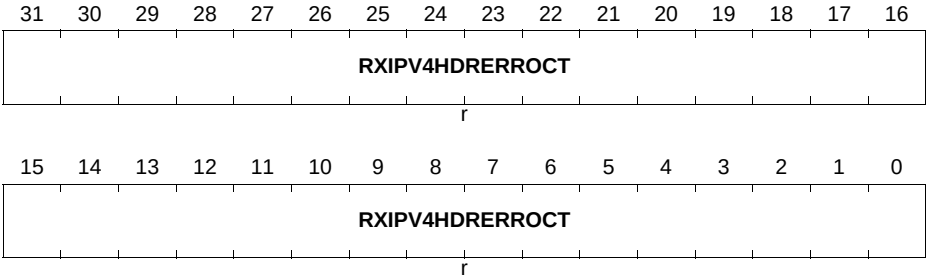
Field	Bits	Type	Description
RXIPV4GDOCT	[31:0]	r	**This field indicates the number of bytes received in good IPv4 datagrams encapsulating TCP, UDP, or ICMP data. The Ethernet header, FCS, pad, or IP pad** bytes are not included in this counter.

RXIPV4_HEADER_ERROR_OCTETS

This register maintains the number of bytes received in IPv4 datagrams with header errors (checksum, length, or version mismatch). The value in the Length field of the IPv4 header is used to update this counter.

ETH0_RXIPV4_HEADER_ERROR_OCTETS

Receive IPV4 Header Error Octet Counter Register(254_H) Reset Value: $0000\ 0000_H$

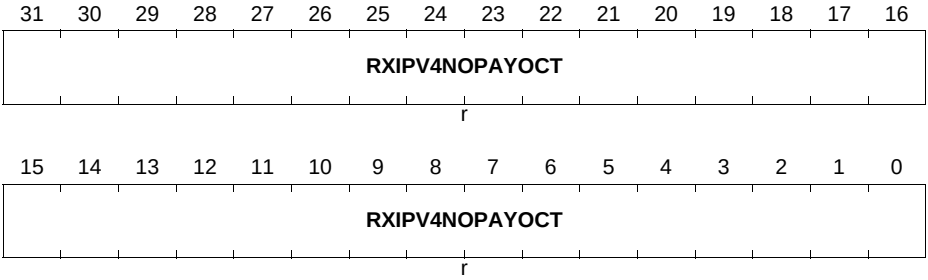


Field	Bits	Type	Description
RXIPV4HDRERROCT	[31:0]	r	**This field indicates the number of bytes received in the IPv4 datagrams with header errors (checksum, length, or version mismatch). The value in the Length field of IPv4 header is used to update this counter.** The Ethernet header, FCS, pad, or IP pad bytes are not included in this counter.

RXIPV4_NO_PAYLOAD_OCTETS

This register maintains the number of bytes received in IPv4 datagrams that did not have a TCP, UDP, or ICMP payload. The value in the IPv4 headers Length field is used to update this counter.

ETH0_RXIPV4_NO_PAYLOAD_OCTETS
Receive IPV4 No Payload Octet Counter Register(258_H) Reset Value: 0000 0000$_H$

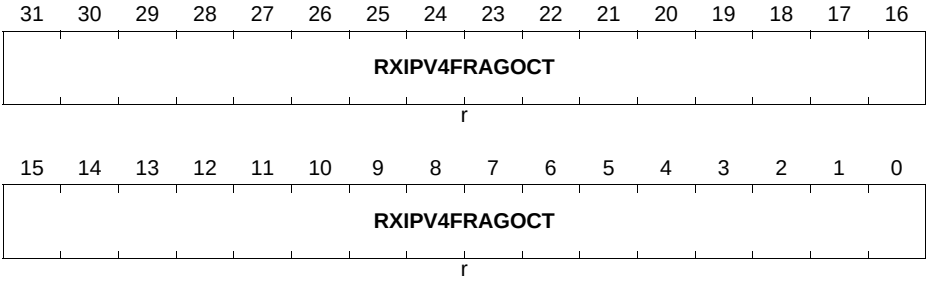


Field	Bits	Type	Description
RXIPV4NOPAYOCT	[31:0]	r	**This field indicates the number of bytes received in IPv4 datagrams that did not have a TCP, UDP, or ICMP payload. The value in the IPv4 headers Length field is used to update this counter.** The Ethernet header, FCS, pad, or IP pad bytes are not included in this counter.

RXIPV4_FRAGMENTED_OCTETS

This register maintains the number of bytes received in fragmented IPv4 datagrams. The value in the IPv4 headers Length field is used to update this counter.

ETH0_RXIPV4_FRAGMENTED_OCTETS
Receive IPV4 Fragmented Octet Counter Register($25C_H$) Reset Value: $0000\ 0000_H$



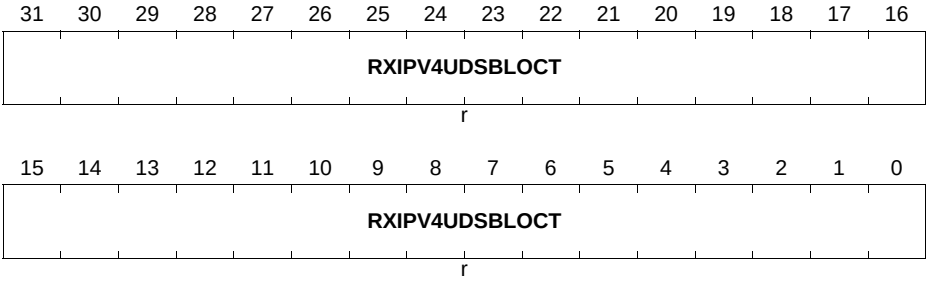
Field	Bits	Type	Description
RXIPV4FRAGOCT	[31:0]	r	**This field indicates the number of bytes received in fragmented IPv4 datagrams. The value in the IPv4 headers Length field is used to update this counter.** The Ethernet header, FCS, pad, or IP pad bytes are not included in this counter.

RXIPV4_UDP_CHECKSUM_DISABLE_OCTETS

This register maintains the number of bytes received in a UDP segment that had the UDP checksum disabled.

ETH0_RXIPV4_UDP_CHECKSUM_DISABLE_OCTETS

Receive IPV4 Fragmented Octet Counter Register(260_H) Reset Value: $0000\ 0000_H$

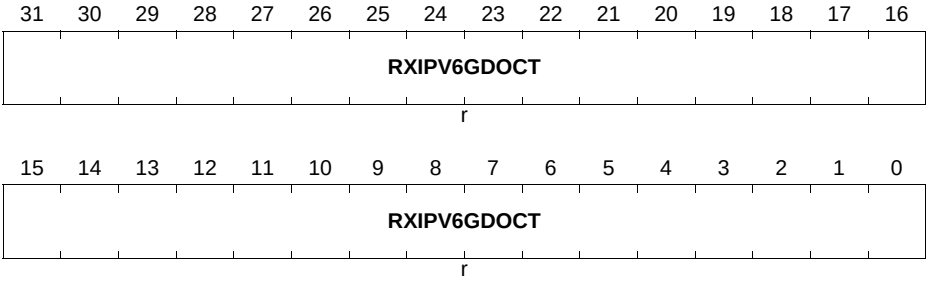


Field	Bits	Type	Description
RXIPV4UDSBLOCT	[31:0]	r	**This field indicates the number of bytes received in a UDP segment that had the UDP checksum disabled. This counter does not count the IP Header bytes. The Ethernet header, FCS, pad, or IP pad bytes are not included in this counter.**

RXIPV6_GOOD_OCTETS

This register maintains the number of bytes received in good IPv6 datagrams encapsulating TCP, UDP or ICMPv6 data.

ETH0_RXIPV6_GOOD_OCTETS
RxIPv6 Good Octets Register **(264_H)** **Reset Value: 0000 0000$_H$**



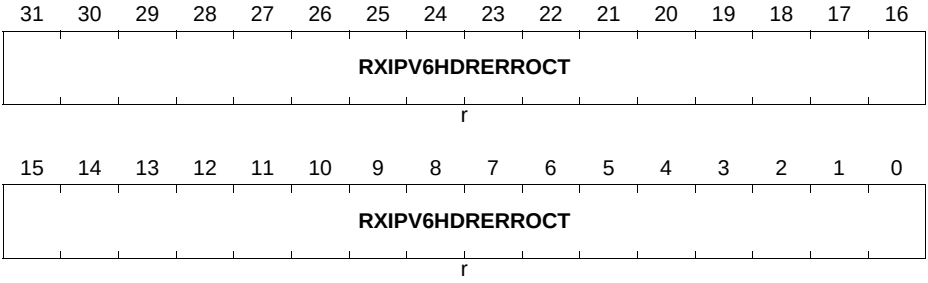
Field	Bits	Type	Description
RXIPV6GDOCT	[31:0]	r	**Thsi field indicates the number of bytes received in good IPv6 datagrams encapsulating TCP, UDP or ICMPv6 data.** This counter does not count the IP Header bytes. The Ethernet header, FCS, pad, or IP pad bytes are not included in this counter.

RXIPV6_HEADER_ERROR_OCTETS

This register maintains the number of bytes received in IPv6 datagrams with header errors (length or version mismatch).

ETH0_RXIPV6_HEADER_ERROR_OCTETS

Receive IPV6 Header Error Octet Counter Register(268_H) Reset Value: $0000\ 0000_H$

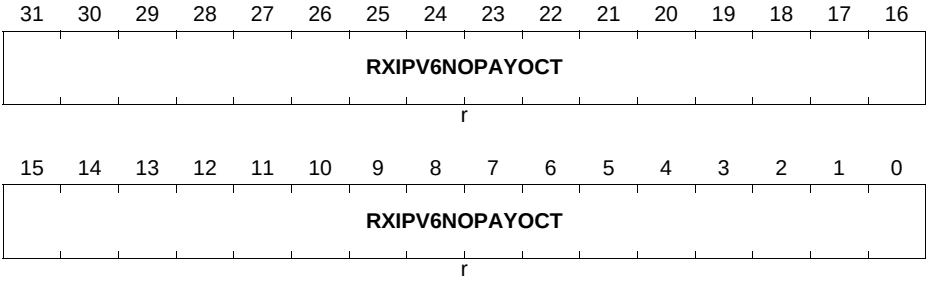


Field	Bits	Type	Description
RXIPV6HDRERROCT	[31:0]	r	**This field indicates the number of bytes received in IPv6 datagrams with header errors (length or version mismatch). The value in the IPv6 headers Length field is used to update this counter.** This counter does not count the IP Header bytes. The Ethernet header, FCS, pad, or IP pad bytes are not included in this counter.

RXIPV6_NO_PAYLOAD_OCTETS

This register maintains the number of bytes received in IPv6 datagrams that did not have a TCP, UDP, or ICMP payload.

ETH0_RXIPV6_NO_PAYLOAD_OCTETS
Receive IPV6 No Payload Octet Counter Register($26C_H$) Reset Value: $0000\ 0000_H$

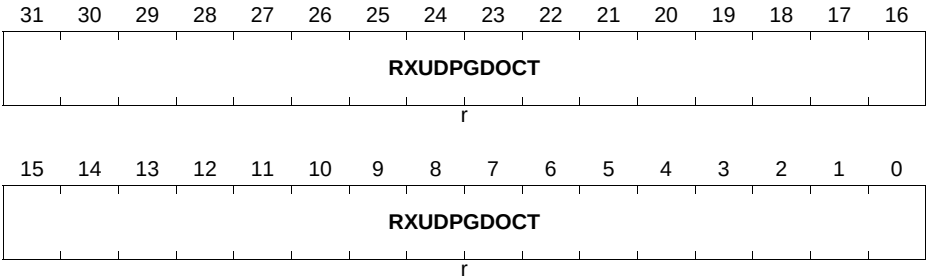


Field	Bits	Type	Description
RXIPV6NOPAYOCT	[31:0]	r	**This field indicates the number of bytes received in IPv6 datagrams that did not have a TCP, UDP, or ICMP payload. The value in the IPv6 headers Length field is used to update this counter.** This counter does not count the IP Header bytes. The Ethernet header, FCS, pad, or IP pad bytes are not included in this counter.

RXUDP_GOOD_OCTETS

This register maintains the number of bytes received in a good UDP segment.

ETH0_RXUDP_GOOD_OCTETS
Receive UDP Good Octets Register (270_H) **Reset Value: 0000 0000$_H$**

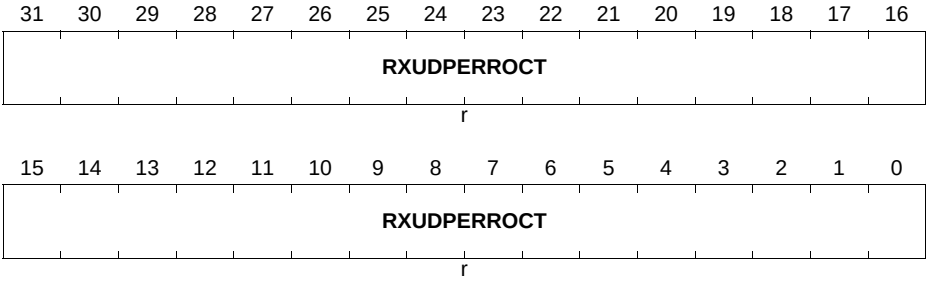


Field	Bits	Type	Description
RXUDPGDOCT	[31:0]	r	**This field indicates the number of bytes received in a good UDP segment. This counter does not count IP header bytes. The Ethernet header, FCS, pad, or IP pad bytes are not included in this counter.**

RXUDP_ERROR_OCTETS

This register maintains the number of bytes received in a UDP segment with checksum errors.

ETH0_RXUDP_ERROR_OCTETS
Receive UDP Error Octets Register (274$_H$) **Reset Value: 0000 0000$_H$**

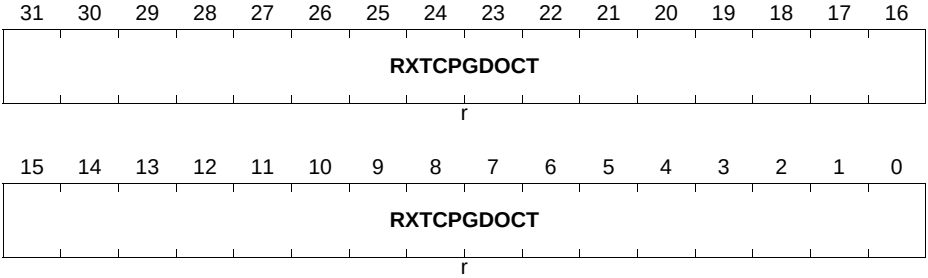


Field	Bits	Type	Description
RXUDPERROCT	[31:0]	r	**This field indicates the number of bytes received in a UDP segment with checksum errors. This counter does not count the IP Header bytes. The Ethernet header, FCS, pad, or IP pad bytes are not included in this counter.**

RXTCP_GOOD_OCTETS

This register maintains the number of bytes received in a good TCP segment.

ETH0_RXTCP_GOOD_OCTETS
Receive TCP Good Octets Register (278_H) **Reset Value: $0000\ 0000_H$**

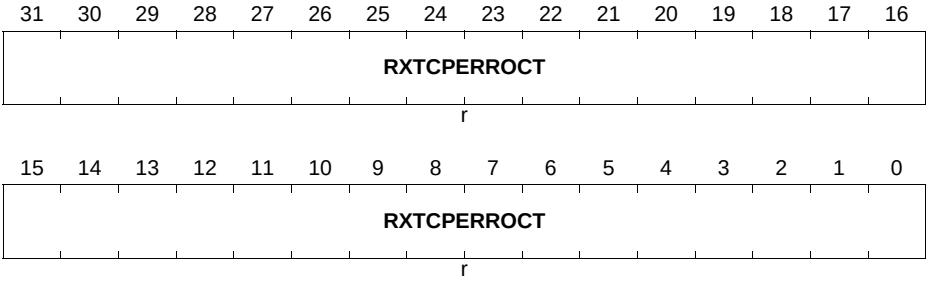


Field	Bits	Type	Description
RXTCPGDOCT	[31:0]	r	**This field indicates the number of bytes received in a good TCP segment. This counter does not count the IP Header bytes. The Ethernet header, FCS, pad, or IP pad bytes are not included in this counter.**

RXTCP_ERROR_OCTETS

This register maintains the number of bytes received in a TCP segment with checksum errors.

ETH0_RXTCP_ERROR_OCTETS
Receive TCP Error Octets Register ($27C_H$) **Reset Value: $0000\ 0000_H$**

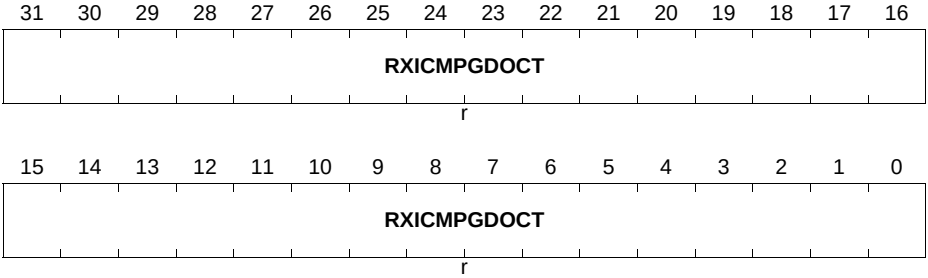


Field	Bits	Type	Description
RXTCPERROCT	[31:0]	r	**Thsi field indicates the number of bytes received in a TCP segment with checksum errors. This counter does not count the IP Header bytes. The Ethernet header, FCS, pad, or IP pad bytes are not included in this counter.**

RXICMP_GOOD_OCTETS

This register maintains the number of bytes received in a good ICMP segment.

ETH0_RXICMP_GOOD_OCTETS
Receive ICMP Good Octets Register (280$_H$) **Reset Value: 0000 0000$_H$**

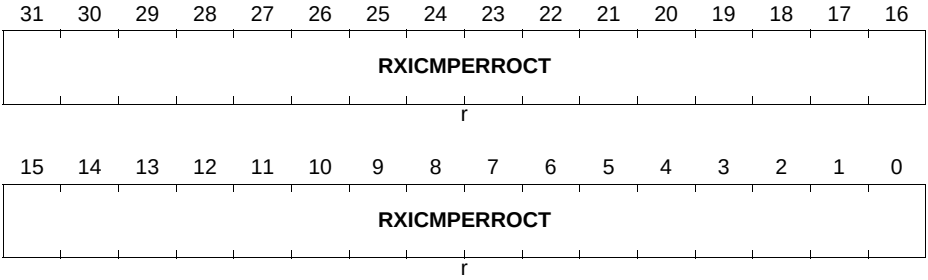


Field	Bits	Type	Description
RXICMPGDOCT	[31:0]	r	**This field indicates the number of bytes received in a good ICMP segment. This counter does not count the IP Header bytes. The Ethernet header, FCS, pad, or IP pad bytes are not included in this counter.**

RXICMP_ERROR_OCTETS

This register maintains the number of bytes received in a ICMP segment with checksum errors. This counter does not count the IP Header bytes. The Ethernet header, FCS, pad, or IP pad bytes are not included in this counter.

ETH0_RXICMP_ERROR_OCTETS
Receive ICMP Error Octets Register (284_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
RXICMPERROCT	[31:0]	r	**Number of bytes received in an ICMP segment with checksum errors**

TIMESTAMP_CONTROL

This register controls the operation of the System Time generator and the processing of PTP packets for timestamping in the Receiver. Note: * Bits[19:8] are reserved and read-only when Advanced Timestamp feature is not enabled.

ETH0_TIMESTAMP_CONTROL
Timestamp Control Register **(700_H)** **Reset Value: 0000 2000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved_23_19													TSENMACADDR	SNAPTYPSEL	
r													rw	rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TSMSTRENA	TSEVNTENA	TSIPV4ENA	TSIPV6ENA	TSIPENA	TSVER2ENA	TSCTRLSSR	TSENALL	Reserved_7_6		TSADDREG	TSTRIG	TSUPDT	TSINIT	TSCFUPDT	TSENA
rw	rw	rw	rw	rw	rw	rw	rw	r		rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
TSENA	0	rw	**Timestamp Enable** When set, the timestamp is added for the transmit and receive frames. When disabled, timestamp is not added for the transmit and receive frames and the Timestamp Generator is also suspended. You need to initialize the Timestamp (system time) after enabling this mode. On the receive side, the MAC processes the 1588 frames only if this bit is set.
TSCFUPDT	1	rw	**Timestamp Fine or Coarse Update** When set, this bit indicates that the system times update should be done using the fine update method. When reset, it indicates the system timestamp update should be done using the Coarse method.

Field	Bits	Type	Description
TSINIT	2	rw	**Timestamp Initialize** When set, the system time is initialized (overwritten) with the value specified in the SYSTEM_TIME_SECONDS_UPDATE Register and SYSTEM_TIME_NANOSECONDS_UPDATE Register. This bit should be read zero before updating it. This bit is reset when the initialization is complete.
TSUPDT	3	rw	**Timestamp Update** When set, the system time is updated (added or subtracted) with the value specified in System Time_Seconds_Update Register and SYSTEM_TIME_NANOSECONDS_UPDATE Register. This bit should be read zero before updating it. This bit is reset when the update is completed in hardware.
TSTRIG	4	rw	**Timestamp Interrupt Trigger Enable** When set, the timestamp interrupt is generated when the System Time becomes greater than the value written in the Target Time register. This bit is reset after the generation of the Timestamp Trigger Interrupt.
TSADDREG	5	rw	**Addend Reg Update** When set, the content of the TIMESTAMP_ADDEND register is updated in the PTP block for fine correction. This is cleared when the update is completed. This register bit should be zero before setting it.
Reserved_7_6	[7:6]	r	**Reserved**
TSENALL	8	rw	**Enable Timestamp for All Frames** When set, the timestamp snapshot is enabled for all frames received by the MAC.
TSCTRLSSR	9	rw	**Timestamp Digital or Binary Rollover Control** When set, the Timestamp Low register rolls over after 3B9A C9FFH value (that is, 1 nanosecond accuracy) and increments the timestamp (High) seconds. When reset, the rollover value of sub-second register is 7FFF FFFFH. The sub-second increment has to be programmed correctly depending on the PTP reference clock frequency and the value of this bit.

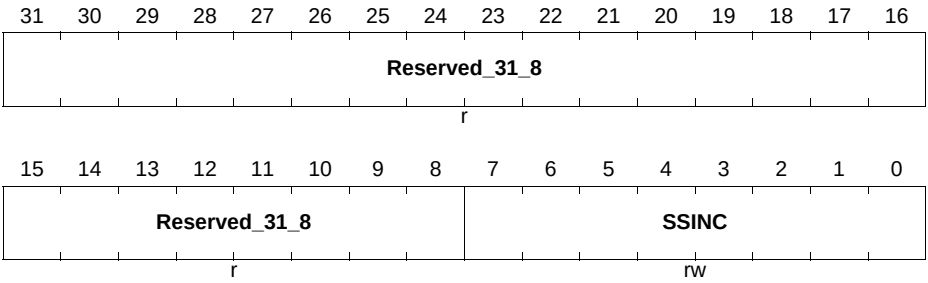
Field	Bits	Type	Description
TSVER2ENA	10	rw	**Enable PTP packet Processing for Version 2 Format** When set, the PTP packets are processed using the 1588 version 2 format. Otherwise, the PTP packets are processed using the version 1 format.
TSIPENA	11	rw	**Enable Processing of PTP over Ethernet Frames** When set, the MAC receiver processes the PTP packets encapsulated directly in the Ethernet frames. When this bit is clear, the MAC ignores the PTP over Ethernet packets.
TSIPV6ENA	12	rw	**Enable Processing of PTP Frames Sent Over IPv6-UDP** When set, the MAC receiver processes PTP packets encapsulated in UDP over IPv6 packets. When this bit is clear, the MAC ignores the PTP transported over UDP-IPv6 packets.
TSIPV4ENA	13	rw	**Enable Processing of PTP Frames Sent over IPv4-UDP** When set, the MAC receiver processes the PTP packets encapsulated in UDP over IPv4 packets. When this bit is clear, the MAC ignores the PTP transported over UDP-IPv4 packets. This bit is set by default.
TSEVNTENA	14	rw	**Enable Timestamp Snapshot for Event Messages** When set, the timestamp snapshot is taken only for event messages (SYNC, Delay_Req, Pdelay_Req, or Pdelay_Resp). When reset, the snapshot is taken for all messages except Announce, Management, and Signaling.
TSMSTRENA	15	rw	**Enable Snapshot for Messages Relevant to Master** When set, the snapshot is taken only for the messages relevant to the master node. Otherwise, the snapshot is taken for the messages relevant to the slave node.
SNAPTYPSEL	[17:16]	rw	**Select PTP packets for Taking Snapshots** These bits along with Bits 15 and 14 decide the set of PTP packet types for which snapshot needs to be taken.
TSENMACADDR	18	rw	**Enable MAC address for PTP Frame Filtering** When set, the DA MAC address (that matches any MAC Address register) is used to filter the PTP frames when PTP is directly sent over Ethernet.

Field	Bits	Type	Description
Reserved_23_19	[31:19]	r	**Reserved**

SUB_SECOND_INCREMENT

This register is present only when the IEEE 1588 timestamp feature is selected without an external timestamp input. In the Coarse Update mode (**TIMESTAMP_CONTROL**.TSCFUPDT bit), the value in this register is added to the system time every clock cycle of the PTP refference clock. In the Fine Update mode, the value in this register is added to the system time whenever the Accumulator gets an overflow.

ETH0_SUB_SECOND_INCREMENT
Sub-Second Increment Register (704$_H$) **Reset Value: 0000 0000$_H$**

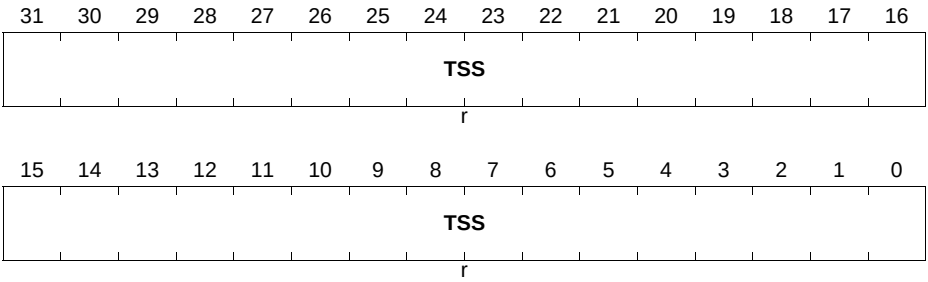


Field	Bits	Type	Description
SSINC	[7:0]	rw	**Sub-second Increment Value** The value programmed in this field is accumulated every clock cycle of the PTP refference clock with the contents of the sub-second register. For example, when PTP clock is 50 MHz (period is 20 ns), you should program 20 (14H) when the System Time-Nanoseconds register has an accuracy of 1 ns (TIMESTAMP_CONTROL.TSCTRLSSR bit is set). When Timestamp.Control.TSCTRLSSR is clear, the Nanoseconds register has a resolution of ~0.465ns. In this case, you should program a value of 43 (2BH) that is derived by 20ns/0.465.
Reserved_31_8	[31:8]	r	**Reserved**

SYSTEM_TIME_SECONDS

The System Time -Seconds register, along with System-TimeNanoseconds register, indicates the current value of the system time maintained by the MAC. Though it is updated on a continuous basis, there is some delay from the actual time because of clock domain transfer latencies .

ETH0_SYSTEM_TIME_SECONDS
System Time - Seconds Register (708_H) Reset Value: 0000 0000$_H$



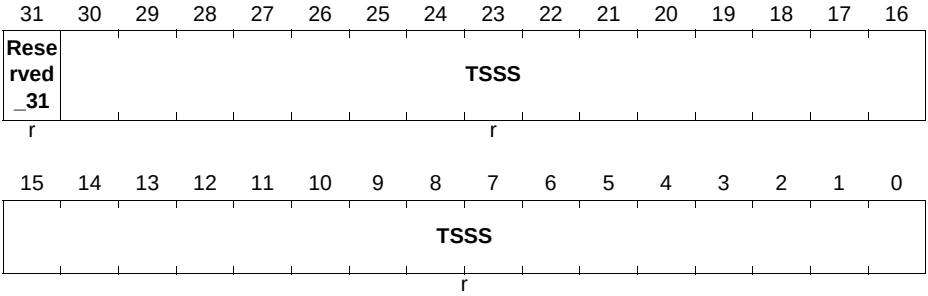
Field	Bits	Type	Description
TSS	[31:0]	r	**Timestamp Second** The value in this field indicates the current value in seconds of the System Time maintained by the MAC.

SYSTEM_TIME_NANOSECONDS

The value in this field has the sub second representation of time, with an accuracy of 0.46 ns. When TIMESTAMP_CONTROL.TSCTRLSSR is set, each bit represents 1 ns and the maximum value is 3B9A C9FFH, after which it rolls-over to zero.

ETH0_SYSTEM_TIME_NANOSECONDS

System Time Nanoseconds Register ($70C_H$) **Reset Value: $0000\ 0000_H$**



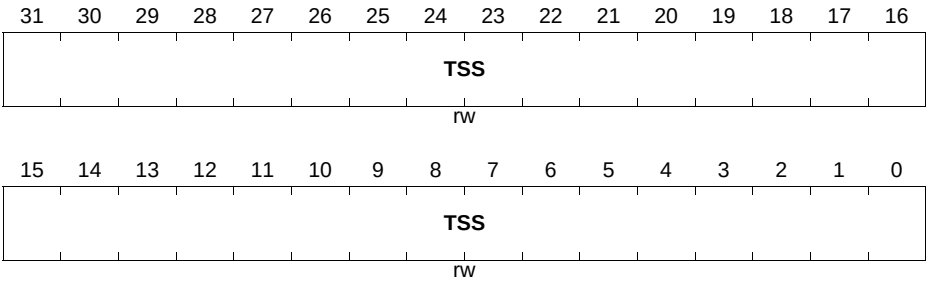
Field	Bits	Type	Description
TSSS	[30:0]	r	**Timestamp Sub Seconds** The value in this field has the sub second representation of time, with an accuracy of 0.46 ns. When TIMESTAMP_CONTROL.TSCTRLSSR is set, each bit represents 1 ns and the maximum value is 3B9A C9FFH, after which it rolls-over to zero.
Reserved_31	31	r	**Reserved**

SYSTEM_TIME_SECONDS_UPDATE

The System Time - Seconds Update register, along with the System_Time_Nanoseconds_Update register, initializes or updates the system time maintained by the MAC. You must write both of these registers before setting the TIMESTAMP_CONTROL.TSINIT or TIMESTAMP_CONTROL.TSUPDT bits.

ETH0_SYSTEM_TIME_SECONDS_UPDATE

System Time - Seconds Update Register(710_H) **Reset Value: $0000\ 0000_H$**



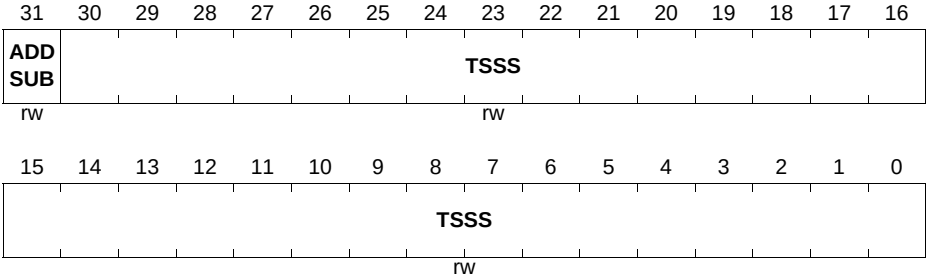
Field	Bits	Type	Description
TSS	[31:0]	rw	**Timestamp Second** The value in this field indicates the time in seconds to be initialized or added to the system time.

SYSTEM_TIME_NANOSECONDS_UPDATE

.

ETH0_SYSTEM_TIME_NANOSECONDS_UPDATE

System Time Nanoseconds Update Register(714_H) **Reset Value: $0000\ 0000_H$**

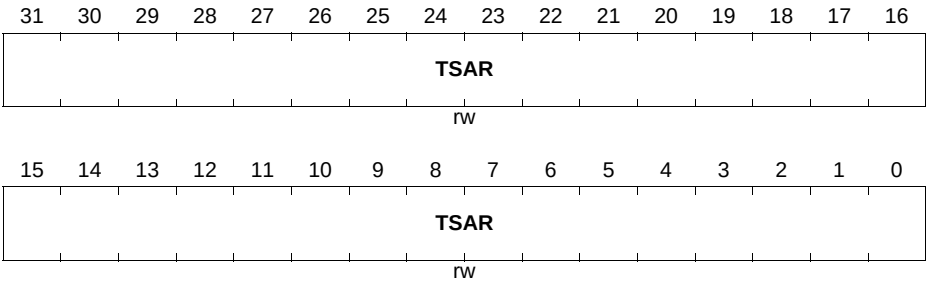


Field	Bits	Type	Description
TSSS	[30:0]	rw	**Timestamp Sub Second** The value in this field has the sub second representation of time, with an accuracy of 0.46 ns. When TIMESTAMP_CONTROL.TSCTRLSSR is set, each bit represents 1 ns and the programmed value should not exceed 3B9A C9FFH.
ADDSUB	31	rw	**Add or subtract time** When this bit is set, the time value is subtracted with the contents of the update register. When this bit is reset, the time value is added with the contents of the update register.

32-bit Register - TIMESTAMP_ADDEND

This register is present only when the IEEE 1588 Timestamp feature is selected without external timestamp input. This register value is used only when the system time is configured for Fine Update mode using TIMESTAMP_CONTROL.TSCFUPDT bit. This register content is added to a 32-bit accumulator in every clock cycle of the PTP refference clock and the system time is updated whenever the accumulator overflows.

ETH0_TIMESTAMP_ADDEND
Timestamp Addend Register **(718_H)** **Reset Value: 0000 0000$_H$**

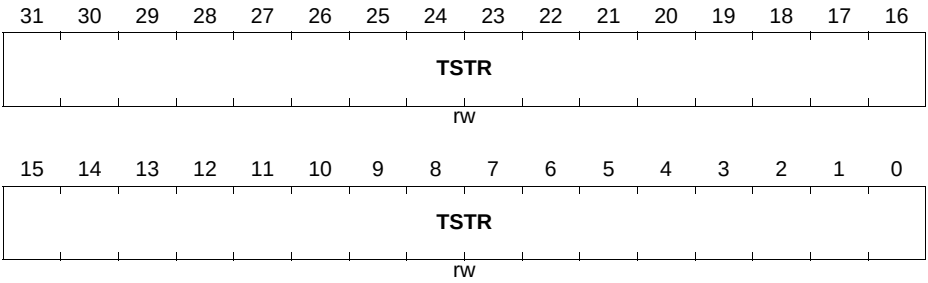


Field	Bits	Type	Description
TSAR	[31:0]	rw	**Timestamp Addend Register** This field indicates the 32-bit time value to be added to the Accumulator register to achieve time synchronization.

TARGET_TIME_SECONDS

The Target Time Seconds register, along with Target Time Nanoseconds register, is used to schedule an interrupt event triggered by the TimestampStatus.TSTARGT bit when Advanced Timestamping is enabled; otherwise, the INTERRUPT_STATUS.TSIS will trigger the interrupt when the system time exceeds the value programmed in these registers. This register is present only when the IEEE 1588 Timestamp feature is selected without external timestamp input.

ETH0_TARGET_TIME_SECONDS
Target Time Seconds Register **($71C_H$)** **Reset Value: 0000 0000$_H$**

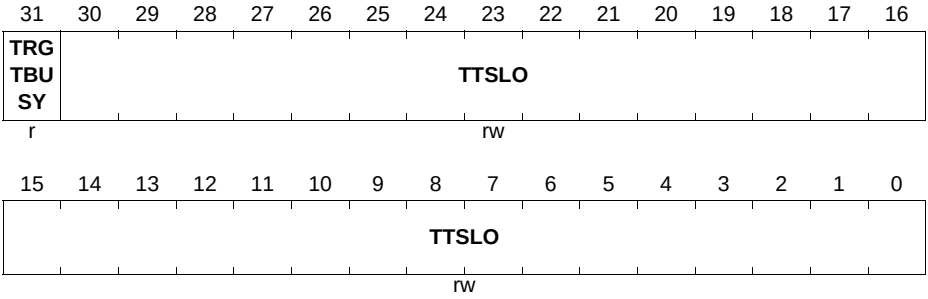


Field	Bits	Type	Description
TSTR	[31:0]	rw	**Target Time Seconds Register** This register stores the time in seconds. When the timestamp value matches or exceeds both Target Timestamp registers, then based on PPS_CONTROL.TRGTMODSEL0, the MAC starts or stops the PPS signal output and generates an interrupt (if enabled).

TARGET_TIME_NANOSECONDS

This register is present only when the IEEE 1588 Timestamp feature is selected without external timestamp input.

ETH0_TARGET_TIME_NANOSECONDS
Target Time Nanoseconds Register (720_H) **Reset Value: $0000\ 0000_H$**



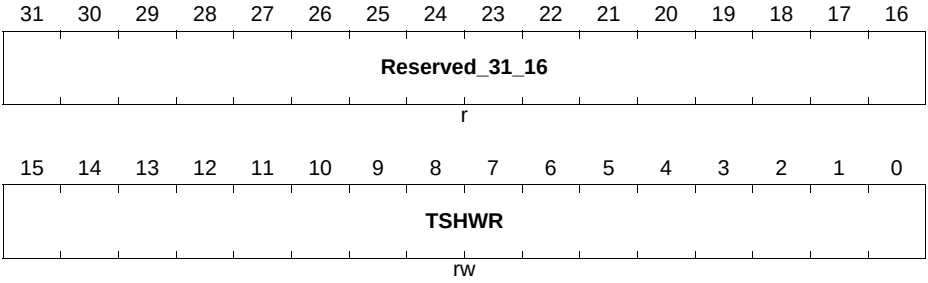
Field	Bits	Type	Description
TTSLO	[30:0]	rw	**Target Timestamp Low Register** This register stores the time in (signed) nanoseconds. When the value of the timestamp matches the both Target Timestamp registers, then based on the PPS_CONTROL.TPPSRGTMODSEL0 field , the MAC starts or stops the PPS signal output and generates an interrupt (if enabled). This value should not exceed 3B9A C9FFH when TIMESTAMP_CONTROL.TSCTRLSSR is set . The actual start or stop time of the PPS signal output may have an error margin up to one unit of sub-second increment value.

Field	Bits	Type	Description
TRGTBUSY	31	r	**Target Time Register Busy** The MAC sets this bit when the PPS_CONTROL.PPSCMD field is programmed to 010B or 011B. Programming the PPSCMD field to 010B or 011B, instructs the MAC to synchronize the Target Time Registers to the PTP clock domain. The MAC clears this bit after synchronizing the Target Time Registers to the PTP clock domain The application must not update the Target Time Registers when this bit is read as 1. Otherwise, the synchronization of the previous programmed time gets corrupted. This bit is reserved when the Enable Flexible Pulse-Per-Second Output feature is not selected.

SYSTEM_TIME_HIGHER_WORD_SECONDS

This register is present only when the IEEE 1588 Advanced Timestamp feature is selected without an external timestamp input.

ETH0_SYSTEM_TIME_HIGHER_WORD_SECONDS
System Time - Higher Word Seconds Register (724_H) **Reset Value: $0000\ 0000_H$**

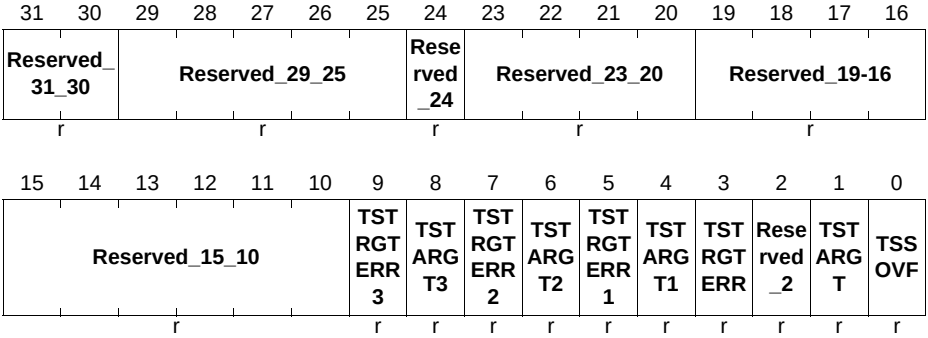


Field	Bits	Type	Description
TSHWR	[15:0]	rw	**Timestamp Higher Word Register** This field contains the most significant 16-bits of the timestamp seconds value. The register is directly written to initialize the value. This register is incremented when there is an overflow from the 32-bits of the SYSTEM_TIME_SECONDS register.
Reserved_31_16	[31:16]	r	**Reserved**

TIMESTAMP_STATUS

All bits except Bits[27:25] gets cleared when the CPU reads this register.

ETH0_TIMESTAMP_STATUS
Timestamp Status Register **(728_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
TSSOVF	0	r	**Timestamp Seconds Overflow** When set, this bit indicates that the seconds value of the timestamp (when supporting version 2 format) has overflowed beyond FFFF_FFFFH.
TSTARGT	1	r	**Timestamp Target Time Reached** When set, this bit indicates that the value of system time is greater or equal to the value specified in the Target_Time_ Seconds Register and Target Time Nanoseconds Register.
Reserved_2	2	r	**Reserved**
TSTRGTERR	3	r	**Timestamp Target Time Error** This bit is set when the target time, being programmed in Target Time Registers, is already elapsed. This bit is cleared when read by the application.

Field	Bits	Type	Description
TSTARGT1	4	r	**Timestamp Target Time Reached for Target Time PPS1** When set, this bit indicates that the value of system time is greater than or equal to the value specified in PPS1_ Target_Time_High Register and PPS1_Target_Time_Low Register.
TSTRGTERR1	5	r	**Timestamp Target Time Error** This bit is set when the target time, being programmed in Register 480 and Register 481, is already elapsed. This bit is cleared when read by the application.
TSTARGT2	6	r	**Timestamp Target Time Reached for Target Time PPS2** When set, this bit indicates that the value of system time is greater than or equal to the value specified in Register 488 [PPS2 Target Time High Register] and Register 489 [PPS2 Target Time Low Register].
TSTRGTERR2	7	r	**Timestamp Target Time Error** This bit is set when the target time, being programmed in Register 488 and Register 489, is already elapsed. This bit is cleared when read by the application.
TSTARGT3	8	r	**Timestamp Target Time Reached for Target Time PPS3** When set, this bit indicates that the value of system time is greater than or equal to the value specified in Register 496 [PPS3 Target Time High Register] and Register 497 [PPS3 Target Time Low Register].
TSTRGTERR3	9	r	**Timestamp Target Time Error** This bit is set when the target time, being programmed in Register 496 and Register 497, is already elapsed. This bit is cleared when read by the application.
Reserved_15_10	[15:10]	r	**Reserved**
Reserved_19-16	[19:16]	r	**Reserved**
Reserved_23_20	[23:20]	r	**Reserved**
Reserved_24	24	r	**Reserved**

Field	Bits	Type	Description
Reserved_29_25	[29:25]	r	**Reserved**
Reserved_31_30	[31:30]	r	**Reserved**

PPS_CONTROL

Note: * Bits[30:24] are valid only when four Flexible PPS outputs are selected. * Bits[22:16] are valid only when three or more Flexible PPS outputs are selected. * Bits[14:8] are valid only when two or more Flexible PPS outputs are selected. * Bits[6:4] are valid only when Flexible PPS feature is selected.

ETH0_PPS_CONTROL
PPS Control Register **($72C_H$)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved_31	TRGTMODSEL3		Reserved_28_27		PPSCMD3			Reserved_23	TRGTMODSEL2		Reserved_20_19		PPSCMD2		
r	r		r		r			r	r		r		r		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved_15	TRGTMODSEL1		Reserved_12_11		PPSCMD1			Reserved_7	TRGTMODSEL0		PPSEN0	PPSCTRL_PPSCMD			
r	r		r		r			r	r		r	rw			

Field	Bits	Type	Description
PPSCTRL_PPSCMD	[3:0]	rw	**PPSCTRL0 or PPSCMD0** PPSCTRL0: PPS0 Output Frequency Control This field controls the frequency of the PPS0 output (ptp_pps_o[0]) signal. The default value of PPSCTRL is 0000, and the PPS output is 1 pulse (of width clk_ptp_i) every second. For other values of PPSCTRL, the PPS output becomes a generated clock of following frequencies: -0001B: The binary rollover is 2 Hz, and the digital rollover is 1 Hz. -0010B: The binary rollover is 4 Hz, and the digital rollover is 2 Hz. -0011B: The binary rollover is 8 Hz, and the digital rollover is 4 Hz. -0100B: The binary rollover is 16 Hz, and the digital rollover is 8 Hz. -... -1111B: The binary rollover is 32.768 KHz, and the digital rollover is 16.384 KHz. See also [1)]

Field	Bits	Type	Description
PPSEN0	4	r	**Flexible PPS Output Mode Enable** When set low, Bits[3:0] function as PPSCTRL (backward compatible). When set high, Bits[3:0] function as PPSCMD.
TRGTMODSEL0	[6:5]	r	**Target Time Register Mode for PPS0 Output** This field indicates the Target Time registers (register 455 and 456) mode for PPS0 output signal: * 00B: Indicates that the Target Time registers are programmed only for generating the interrupt event. * 01B: Reserved * 10B: Indicates that the Target Time registers are programmed for generating the interrupt event and starting or stopping the generation of the PPS0 output signal. * 11B: Indicates that the Target Time registers are programmed only for starting or stopping the generation of the PPS0 output signal. No interrupt is asserted.
Reserved_7	7	r	**Reserved**
PPSCMD1	[10:8]	r	**Flexible PPS1 Output Control** This field controls the flexible PPS1 output (ptp_pps_o[1]) signal. This field is similar to PPSCMD0[2:0] in functionality.
Reserved_12_11	[12:11]	r	**Reserved**
TRGTMODSEL1	[14:13]	r	**Target Time Register Mode for PPS1 Output** This field indicates the Target Time registers (register 480 and 481) mode for PPS1 output signal. This field is similar to the TRGTMODSEL0 field.
Reserved_15	15	r	**Reserved**
PPSCMD2	[18:16]	r	**Flexible PPS2 Output Control** This field controls the flexible PPS2 output (ptp_pps_o[2]) signal. This field is similar to PPSCMD0[2:0] in functionality.
Reserved_20_19	[20:19]	r	**Reserved**

Field	Bits	Type	Description
TRGTMODSEL2	[22:21]	r	**Target Time Register Mode for PPS2 Output** This field indicates the Target Time registers (register 488 and 489) mode for PPS2 output signal. This field is similar to the TRGTMODSEL0 field.
Reserved_23	23	r	**Reserved**
PPSCMD3	[26:24]	r	**Flexible PPS3 Output Control** This field controls the flexible PPS3 output (ptp_pps_o[3]) signal. This field is similar to PPSCMD0[2:0] in functionality.
Reserved_28_27	[28:27]	r	**Reserved**
TRGTMODSEL3	[30:29]	r	**Target Time Register Mode for PPS3 Output** This field indicates the Target Time registers (register 496 and 497) mode for PPS3 output signal. This field is similar to the TRGTMODSEL0 field.
Reserved_31	31	r	**Reserved**

1) In the binary rollover mode, the PPS output (ptp_pps_o) has a duty cycle of 50 percent with these frequencies.

In the digital rollover mode, the PPS output frequency is an average number. The actual clock is of different frequency that gets synchronized every second. For example:

* When PPSCTRL = 0001, the PPS (1 Hz) has a low period of 537 ms and a high period of 463 ms

* When PPSCTRL = 0010, the PPS (2 Hz) is a sequence of:

- One clock of 50 percent duty cycle and 537 ms period

- Second clock of 463 ms period (268 ms low and 195 ms high)

* When PPSCTRL = 0011, the PPS (4 Hz) is a sequence of:

- Three clocks of 50 percent duty cycle and 268 ms period

- Fourth clock of 195 ms period (134 ms low and 61 ms high

This behavior is because of the non-linear toggling of bits in the digital rollover mode in System Time - Nanoseconds Register].

Flexible PPS0 Output Control

Programming these bits with a non-zero value instructs the MAC to initiate an event. Once the command is transferred or synchronized to the PTP clock domain, these bits get cleared automatically. The Software should ensure that these bits are programmed only when they are all-zero. The following list describes the values of PPSCMD0:

* 0000: No Command

* 0001: START Single Pulse

This command generates single pulse rising at the start point defined in Target Time Registers (TARGET_TIME_SECONDS and TARGET_TIME_NANOSECONDS) and of a duration defined in the PPS0 Width Register.

* 0010: START Pulse Train

This command generates the train of pulses rising at the start point defined in the Target Time Registers and of a duration defined in the PPS0 Width Register and repeated at interval defined in the PPS Interval Register. By default, the PPS pulse train is free-running unless stopped by 'STOP Pulse train at time' or 'STOP Pulse Train immediately' commands.

* 0011: Cancel START

This command cancels the START Single Pulse and START Pulse Train commands if the system time has not crossed the programmed start time.

* 0100: STOP Pulse train at time

This command stops the train of pulses initiated by the START Pulse Train command (PPSCMD = 0010) after the time programmed in the Target Time registers elapses.

* 0101: STOP Pulse Train immediately

This command immediately stops the train of pulses initiated by the START Pulse Train command (PPSCMD = 0010).

* 0110: Cancel STOP Pulse train

This command cancels the STOP pulse train at time command if the programmed stop time has not elapsed. The PPS pulse train becomes free-running on the successful execution of this command.

* 0111-1111: Reserved

BUS_MODE

The Bus Mode register establishes the bus operating modes for the DMA.

ETH0_BUS_MODE
Bus Mode Register **(1000_H)** **Reset Value: $0002\ 0101_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved_31_30		PRWG		TXPR	MB	AAL	8xPBL	USP	RPBL						FB
r		r		rw	rw	rw	rw	rw	rw						rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR		PBL						Reserved_7	DSL					DA	SWR
rw		rw						rw	rw					rw	rw

Field	Bits	Type	Description
SWR	0	rw	**Software Reset** When this bit is set, the MAC DMA Controller resets the logic and all internal registers of the MAC. It is cleared automatically after the reset operation has completed in all of the DWC_ETH clock domains. Before reprogramming any register of the DWC_ETH, you should read a zero (0) value in this bit . Note: * The reset operation is completed only when all resets in all active clock domains are de-asserted. Therefore, it is essential that all the PHY inputs clocks (applicable for the selected PHY interface) are present for the software reset completion.

Field	Bits	Type	Description
DA	1	rw	**DMA Arbitration Scheme** This bit specifies the arbitration scheme between the transmit and receive paths of Channel 0. * 0: Weighted round-robin with Rx:Tx or Tx:Rx. The priority between the paths is according to the priority specified in BUS_MODE.PR and priority weights specified in BUS_MODE.TXPR. * 1: Fixed priority. The transmit path has priority over receive path when BUS_MODE.TXPR is set. Otherwise, receive path has priority over the transmit path.
DSL	[6:2]	rw	**Descriptor Skip Length** This bit specifies the number of Words to skip between two unchained descriptors. The address skipping starts from the end of current descriptor to the start of next descriptor. When the DSL value is equal to zero, then the descriptor table is taken as contiguous by the DMA in Ring mode.
Reserved_7	7	rw	**Reserved**

Field	Bits	Type	Description
PBL	[13:8]	rw	**Programmable Burst Length** These bits indicate the maximum number of beats to be transferred in one DMA transaction. This is the maximum value that is used in a single block Read or Write. The DMA always attempts to burst as specified in PBL each time it starts a Burst transfer on the bus. PBL can be programmed with permissible values of 1, 2, 4, 8, 16, and 32. Any other value results in undefined behavior. When BUS_MODE.USP is set high, this BUS_MODE.PBL value is applicable only for Tx DMA transactions. If the number of beats to be transferred is more than 32, then perform the following steps: 1. Set the 8xPBL mode. 2. Set the PBL. For example, if the maximum number of beats to be transferred is 64, then first set 8xPBL to 1 and then set PBL to 8. The PBL values have the following limitation: The maximum number of possible beats (PBL) is limited by the size of the Tx FIFO and Rx FIFO in the MTL layer and the data bus width on the DMA. The FIFO has a constraint that the maximum beat supported is half the depth of the FIFO, except when specified. For different data bus widths and FIFO sizes, the valid PBL range (including x8 mode) is provided in the following list. If the PBL is common for both transmit and receive DMA, the minimum Rx FIFO and Tx FIFO depths must be considered. Note: In the half-duplex mode, the valid PBL range specified in the following list is applicable only for Tx FIFO.
PR	[15:14]	rw	**Priority Ratio** These bits control the priority ratio in the weighted round-robin arbitration between the Rx DMA and Tx DMA. These bits are valid only when Bit 1 (DA) is reset. The priority ratio is Rx:Tx or Tx:Rx depending on whether Bit 27 (TXPR) is reset or set. * 00B: The Priority Ratio is 1:1. * 01B: The Priority Ratio is 2:1. * 10B: The Priority Ratio is 3:1. * 11B: The Priority Ratio is 4:1.

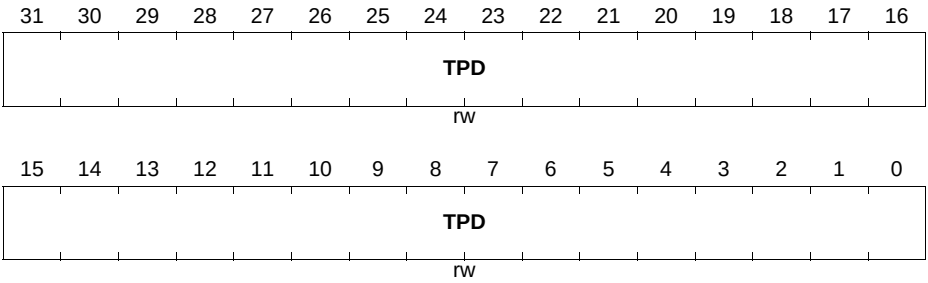
Field	Bits	Type	Description
FB	16	rw	**Fixed Burst** This bit controls whether the Bus Master interface performs fixed burst transfers or not. When set, the AHB interface uses only SINGLE, INCR4, INCR8, or INCR16 during start of the normal burst transfers. When reset, the AHB interface uses SINGLE and INCR burst transfer operations.
RPBL	[22:17]	rw	**Rx DMA PBL** This field indicates the maximum number of beats to be transferred in one Rx DMA transaction. This is the maximum value that is used in a single block Read or Write. The Rx DMA always attempts to burst as specified in the RPBL bit each time it starts a Burst transfer on the bus. You can program RPBL with values of 1, 2, 4, 8, 16, and 32. Any other value results in undefined behavior. This field is valid and applicable only when USP is set high.
USP	23	rw	**Use Seperate PBL** When set high, this bit configures the Rx DMA to use the value configured in Bits[22:17] as PBL. The BUS_MODE. PBL value is applicable only to the Tx DMA operations. When reset to low, the BUS_MODE.PBL value is applicable for both DMA engines.
8xPBL	24	rw	**8xPBL Mode** When set high, this bit multiplies the programmed PBL value (Bits[22:17] and Bits[13:8]) eight times. Therefore, the DMA transfers the data in 8, 16, 32, 64, 128, and 256 beats depending on the BUS_MODE.PBL value.
AAL	25	rw	**Address Aligned Beats** When this bit is set high and the BUS_MODE.FB bit is equal to 1, the AHB interface generates all bursts aligned to the start address LS bits. If the BUS_MODE.FB bit is equal to 0, the first burst (accessing the data buffer's start address) is not aligned, but subsequent bursts are aligned to the address.

Field	Bits	Type	Description
MB	26	rw	**Mixed Burst** When this bit is set high and the BUS_MODE.FB bit is low, the Bus Master interface starts all bursts of length more than 16 with INCR (undefined burst) whereas it reverts to fixed burst transfers (INCRx and SINGLE) for burst length of 16 and less.
TXPR	27	rw	**Transmit Priority** When set, this bit indicates that the transmit DMA has higher priority than the receive DMA during arbitration for the system-side bus.
PRWG	[29:28]	r	**Channel Priority Weights** This field sets the priority weights for Channel 0 during the round-robin arbitration between the DMA channels for the system bus. * 00B: The priority weight is 1. * 01B: The priority weight is 2. * 10B: The priority weight is 3. * 11B: The priority weight is 4.
Reserved_31_30	[31:30]	r	**Reserved**

TRANSMIT_POLL_DEMAND

The Transmit Poll Demand register enables the Tx DMA to check whether or not the DMA owns the current descriptor. The Transmit Poll Demand command is given to wake up the Tx DMA if it is in the Suspend mode. The Tx DMA can go into the Suspend mode because of an Underflow error in a transmitted frame or the unavailability of descriptors owned by it. You can give this command anytime and the Tx DMA resets this command when it again starts fetching the current descriptor from the XMC4500 memory.

ETH0_TRANSMIT_POLL_DEMAND
Transmit Poll Demand Register (1004$_H$) Reset Value: 0000 0000$_H$

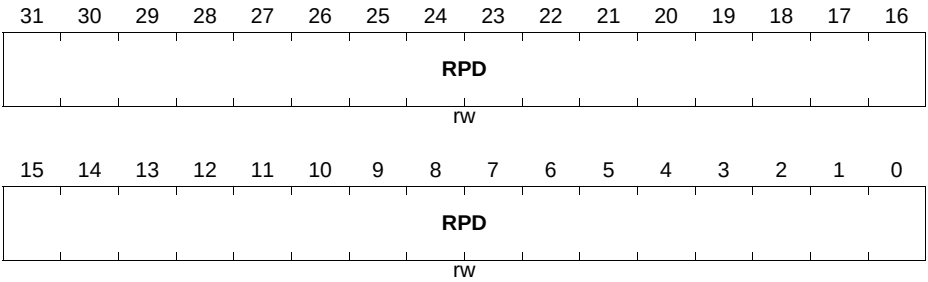


Field	Bits	Type	Description
TPD	[31:0]	rw	**Transmit Poll Demand** When these bits are written with any value, the DMA reads the current descriptor pointed to by the Current Host Transmit Descriptor Register. If that descriptor is not available (owned by the CPU), the transmission returns to the Suspend state and STATUS.TU is asserted. If the descriptor is available, the transmission resumes.

RECEIVE_POLL_DEMAND

The Receive Poll Demand register enables the receive DMA to check for new descriptors. This command is used to wake up the Rx DMA from the SUSPEND state. The RxDMA can go into the SUSPEND state only because of the unavailability of descriptors it owns.

ETH0_RECEIVE_POLL_DEMAND
Receive Poll Demand Register (1008$_H$) Reset Value: 0000 0000$_H$

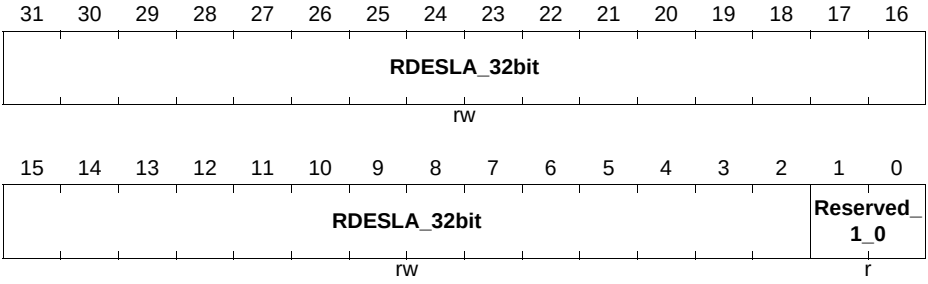


Field	Bits	Type	Description
RPD	[31:0]	rw	**Receive Poll Demand** When these bits are written with any value, the DMA reads the current descriptor pointed to by the Current Host Receive Descriptor Register. If that descriptor is not available (owned by the CPU), the reception returns to the Suspended state and STATUS.RU is not asserted. If the descriptor is available, the Rx DMA returns to the active state.

RECEIVE_DESCRIPTOR_LIST_ADDRESS

The Receive Descriptor List Address register points to the start of the Receive Descriptor List. The descriptor lists reside in the XMC4500's physical memory space and must be Word-aligned . The DMA internally converts it to bus width aligned address by making the corresponding LS bits low. Writing to this register is permitted only when reception is stopped. When stopped, this register must be written to before the receive Start command is given. You can write to this register only when Rx DMA has stopped, that is, Bit 1 (SR) is set to zero in the Operation Mode Register. When stopped, this register can be written with a new descriptor list address. When you set the SR bit to 1, the DMA takes the newly programmed descriptor base address. If this register is not changed when the SR bit is set to 0, then the DMA takes the descriptor address where it was stopped earlier.

ETH0_RECEIVE_DESCRIPTOR_LIST_ADDRESS
Receive Descriptor Address Register ($100C_H$) **Reset Value: $0000\ 0000_H$**

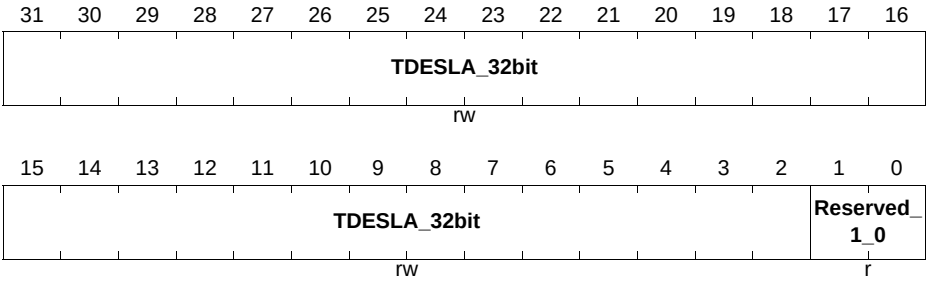


Field	Bits	Type	Description
Reserved_1_0	[1:0]	r	**Reserved**
RDESLA_32bit	[31:2]	rw	**Start of Receive List** This field contains the base address of the first descriptor in the Receive Descriptor list. The LSB bits (1:0) for 32-bit bus width are ignored and internally taken as all-zero by the DMA. Therefore, these LSB bits are read-only (RO).

TRANSMIT_DESCRIPTOR_LIST_ADDRESS

The Transmit Descriptor List Address register points to the start of the Transmit Descriptor List. The descriptor lists reside in the XMC4500's physical memory space and must be Word-aligned . The DMA internally converts it to bus width aligned address by making the corresponding LSB to low. You can write to this register only when the Tx DMA has stopped, that is, OPERATION_MODE.ST is set to zero. When stopped, this register can be written with a new descriptor list address. When you set the OPERATION_MODE.ST bit to 1, the DMA takes the newly programmed descriptor base address. If this register is not changed when the ST bit is set to 0, then the DMA takes the descriptor address where it was stopped earlier.

ETH0_TRANSMIT_DESCRIPTOR_LIST_ADDRESS
Transmit descripter Address Register (1010_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
Reserved_1_0	[1:0]	r	**Reserved**
TDESLA_32bit	[31:2]	rw	**Start of Transmit List** This field contains the base address of the first descriptor in the Transmit Descriptor list. The LSB bits (1:0) are ignored and are internally taken as all-zero by the DMA. Therefore, these LSB bits are read-only (RO).

STATUS

The STATUS register contains all status bits that the DMA reports to the CPU. The Software driver reads this register during an interrupt service routine or polling. Most of the fields in this register cause the CPU to be interrupted. The bits of this register are not cleared when read. Writing 1 to (unreserved) Bits[16:0] of this register clears these bits and writing 0 has no effect. Each field (Bits[16:0]) can be masked by masking the appropriate bit in the Interrupt Enable Register.

ETH0_STATUS
Status Register **(1014_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved_31	Reserved_30	TTI	EPI	EMI	Reserved_26	EB			TS			RS			NIS
r	r	r	r	r	r	r			r			r			rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AIS	ERI	FBI	Reserved_12_11		ETI	RWT	RPS	RU	RI	UNF	OVF	TJT	TU	TPS	TI
rw	rw	rw	r		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
TI	0	rw	**Transmit Interrupt** This bit indicates that the frame transmission is complete. When transmission is complete, the Bit 31 (Interrupt on Completion) of TDES1 is reset in the first descriptor, and the specific frame status information is updated in the descriptor.
TPS	1	rw	**Transmit Process Stopped** This bit is set when the transmission is stopped.
TU	2	rw	**Transmit Buffer Unavailable** This bit indicates that the CPU owns the Next Descriptor in the Transmit List and the DMA cannot acquire it. Transmission is suspended. The TS bit field explains the Transmit Process state transitions. To resume processing Transmit descriptors, the CPU should change the ownership of the descriptor by setting TDES0[31] and then issue a Transmit Poll Demand command.

Field	Bits	Type	Description
TJT	3	rw	**Transmit Jabber Timeout** This bit indicates that the Transmit Jabber Timer expired, which happens when the frame size exceeds 2,048 (10,240 bytes when the Jumbo frame is enabled). When the Jabber Timeout occurs, the transmission process is aborted and placed in the Stopped state. This causes the Transmit Jabber Timeout TDES0[14] flag to assert.
OVF	4	rw	**Receive Overflow** This bit indicates that the Receive Buffer had an Overflow during frame reception. If the partial frame is transferred to the application, the overflow status is set in RDES0[11].
UNF	5	rw	**Transmit Underflow** This bit indicates that the Transmit Buffer had an Underflow during frame transmission. Transmission is suspended and an Underflow Error TDES0[1] is set.
RI	6	rw	**Receive Interrupt** This bit indicates that the frame reception is complete. When reception is complete, the Bit 31 of RDES1 (Disable Interrupt on Completion) is reset in the last Descriptor, and the specific frame status information is updated in the descriptor. The reception remains in the Running state.
RU	7	rw	**Receive Buffer Unavailable** This bit indicates that the CPU owns the Next Descriptor in the Receive List and the DMA cannot acquire it. The Receive Process is suspended. To resume processing Receive descriptors, the CPU should change the ownership of the descriptor and issue a Receive Poll Demand command. If no Receive Poll Demand is issued, the Receive Process resumes when the next recognized incoming frame is received. This bit is set only when the previous Receive Descriptor is owned by the DMA.
RPS	8	rw	**Receive Process Stopped** This bit is asserted when the Receive Process enters the Stopped state.

Field	Bits	Type	Description
RWT	9	rw	**Receive Watchdog Timeout** This bit is asserted when a frame with length greater than 2,048 bytes is received (10, 240 when Jumbo Frame mode is enabled).
ETI	10	rw	**Early Transmit Interrupt** This bit indicates that the frame to be transmitted is fully transferred to the MTL Transmit FIFO.
Reserved_12_11	[12:11]	r	**Reserved**
FBI	13	rw	**Fatal Bus Error Interrupt** This bit indicates that a bus error occurred, as described in the EB bit field. When this bit is set, the corresponding DMA engine disables all of its bus accesses.
ERI	14	rw	**Early Receive Interrupt** This bit indicates that the DMA had filled the first data buffer of the packet. STATUS.RI automatically clears this bit.
AIS	15	rw	**Abnormal Interrupt Summary** Abnormal Interrupt Summary bit value is the logical OR of the following when the corresponding interrupt bits are enabled in Interrupt Enable Register: * STATUS.TPS: Transmit Process Stopped * STATUS.TJT: Transmit Jabber Timeout * STATUS.OVF : Receive FIFO Overflow * STATUS.UNF: Transmit Underflow * STATUS.RU: Receive Buffer Unavailable * STATUS.RPS: Receive Process Stopped * STATUS.RWT: Receive Watchdog Timeout * STATUS.ETI: Early Transmit Interrupt * STATUS.FBI: Fatal Bus Error Only unmasked bits affect the Abnormal Interrupt Summary bit. This is a sticky bit and must be cleared each time a corresponding bit, which causes AIS to be set, is cleared.

Field	Bits	Type	Description
NIS	16	rw	**Normal Interrupt Summary** Normal Interrupt Summary bit value is the logical OR of the following when the corresponding interrupt bits are enabled in the Interrupt Enable Register: * STATUS.TI: Transmit Interrupt * STATUS.TU: Transmit Buffer Unavailable * STATUS.RI: Receive Interrupt * STATUS.ERI: Early Receive Interrupt Only unmasked bits (interrupts for which interrupt enable is set in Register 7) affect the Normal Interrupt Summary bit. This is a sticky bit and must be cleared (by writing 1 to this bit) each time a corresponding bit, which causes NIS to be set, is cleared.
RS	[19:17]	r	**Received Process State** This field indicates the Receive DMA FSM state. This field does not generate an interrupt. * 000B: Stopped: Reset or Stop Receive Command issued * 001B: Running: Fetching Receive Transfer Descriptor * 010B: Reserved for future use * 011B: Running: Waiting for receive packet * 100B: Suspended: Receive Descriptor Unavailable * 101B: Running: Closing Receive Descriptor * 110B: TIME_STAMP write state * 111B: Running: Transferring the receive packet data from receive buffer to the XMC4500’s memory

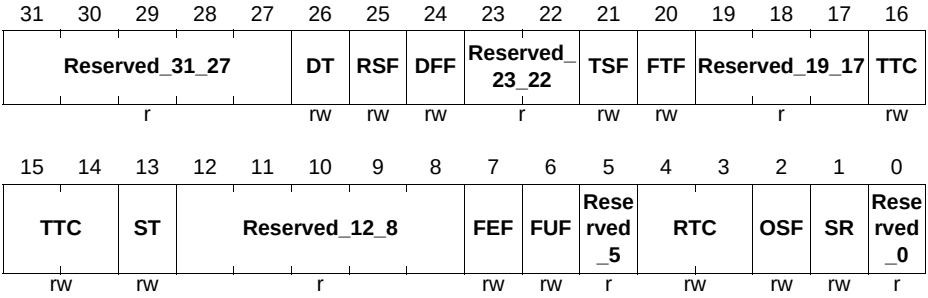
Field	Bits	Type	Description
TS	[22:20]	r	**Transmit Process State** This field indicates the Transmit DMA FSM state. This field does not generate an interrupt. * 000B: Stopped; Reset or Stop Transmit Command issued * 001B: Running; Fetching Transmit Transfer Descriptor * 010B: Running; Waiting for status * 011B: Running; Reading Data from the memory buffer and queuing it to transmit buffer (Tx FIFO) * 100B: TIME_STAMP write state * 101B: Reserved for future use * 110B: Suspended; Transmit Descriptor Unavailable or Transmit Buffer Underflow * 111B: Running; Closing Transmit Descriptor
EB	[25:23]	r	**Error Bits** This field indicates the type of error that caused a Bus Error, for example, error response on the AHB interface. This field is valid only when Bit 13 (FBI) is set. This field does not generate an interrupt. * Bit 23 - 1: Error during data transfer by the Tx DMA - 0: Error during data transfer by the Rx DMA * Bit 24 - 1: Error during read transfer - 0: Error during write transfer * Bit 25 - 1: Error during descriptor access - 0: Error during data buffer access
Reserved_26	26	r	**Reserved**
EMI	27	r	**ETH MMC Interrupt** This bit reflects an interrupt event in the MMC module of the DWC_ETH. The software must read the corresponding registers in the DWC_ETH to get the exact cause of interrupt and clear the source of interrupt to make this bit as 0. The interrupt signal from the DWC_ETH subsystem is high when this bit is high.

Field	Bits	Type	Description
EPI	28	r	**ETH PMT Interrupt** This bit indicates an interrupt event in the PMT module of the ETH. The software must read the PMT Control and STATUS Register in the MAC to get the exact cause of interrupt and clear its source to reset this bit to 0. The interrupt signal from the ETH subsystem is high when this bit is high. Note: This interrupt is different from the Power Management interrupt.
TTI	29	r	**Timestamp Trigger Interrupt** This bit indicates an interrupt event in the Timestamp Generator block of ETH. The software must read the corresponding registers in the ETH to get the exact cause of interrupt and clear its source to reset this bit to 0. When this bit is high, the interrupt signal from the ETH subsystem is high.
Reserved_30	30	r	**Reserved**
Reserved_31	31	r	**Reserved**

OPERATION_MODE

The Operation Mode register establishes the Transmit and Receive operating modes and commands. This register should be the last CSR to be written as part of the DMA initialization.

ETH0_OPERATION_MODE
Operation Mode Register **(1018_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
Reserved_0	0	r	**Reserved**

Field	Bits	Type	Description
SR	1	rw	**Start or Stop Receive** When this bit is set, the Receive process is placed in the Running state. The DMA attempts to acquire the descriptor from the Receive list and processes the incoming frames. The descriptor acquisition is attempted from the current position in the list, which is the address set by Receive Descriptor List Address Register or the position retained when the Receive process was previously stopped. If the DMA does not own the descriptor, reception is suspended and STATUS.RU is set. The Start Receive command is effective only when the reception has stopped. If the command is issued before setting Receive Descriptor List Address Register, the DMA behavior is unpredictable. When this bit is cleared, the Rx DMA operation is stopped after the transfer of the current frame. The next descriptor position in the Receive list is saved and becomes the current position after the Receive process is restarted. The Stop Receive command is effective only when the Receive process is in either the Running (waiting for receive packet) or in the Suspended state.
OSF	2	rw	**Operate on Second Frame** When this bit is set, it instructs the DMA to process the second frame of the Transmit data even before the status for the first frame is obtained.
RTC	[4:3]	rw	**Receive Threshold Control** These two bits control the threshold level of the MTL Receive FIFO. Transfer (request) to DMA starts when the frame size within the MTL Receive FIFO is larger than the threshold. In addition, full frames with length less than the threshold are transferred automatically. The value of 11 is not applicable if the configured Receive FIFO size is 128 bytes. These bits are valid only when the RSF bit is zero, and are ignored when the RSF bit is set to 1. * 00B: 64 * 01B: 32 * 10B: 96 * 11B: 128

Field	Bits	Type	Description
Reserved_5	5	r	**Reserved**
FUF	6	rw	**Forward Undersized Good Frames** When set, the Rx FIFO forwards Undersized frames (frames with no Error and length less than 64 bytes) including pad-bytes and CRC. When reset, the Rx FIFO drops all frames of less than 64 bytes, unless a frame is already transferred because of the lower value of Receive Threshold, for example, RTC = 01B.
FEF	7	rw	**Forward Error Frames** When this bit is reset, the Rx FIFO drops frames with error status (CRC error, collision error, MII_ER, giant frame, watchdog timeout, or overflow). However, if the start byte (write) pointer of a frame is already transferred to the read controller side (in Threshold mode), then the frame is not dropped. In the ETH-MTL configuration in which the Frame Length FIFO is also enabled during core configuration, the Rx FIFO drops the error frames if that frame's start byte is not transferred (output) on the ARI bus. When the FEF bit is set, all frames except runt error frames are forwarded to the DMA. If the RSF bit is set and the Rx FIFO overflows when a partial frame is written, then the frame is dropped irrespective of the FEF bit setting. However, if the RSF bit is reset and the Rx FIFO overflows when a partial frame is written, then a partial frame may be forwarded to the DMA.
Reserved_12_8	[12:8]	r	**Reserved**

Field	Bits	Type	Description
ST	13	rw	**Start or Stop Transmission Command** When this bit is set, transmission is placed in the Running state, and the DMA checks the Transmit List at the current position for a frame to be transmitted. Descriptor acquisition is attempted either from the current position in the list, which is the Transmit List Base Address set by Register 4 [Transmit Descriptor List Address Register], or from the position retained when transmission was stopped previously. If the DMA does not own the current descriptor, transmission enters the Suspended state and Bit 2 (Transmit Buffer Unavailable) of Register 5 [STATUS Register] is set. The Start Transmission command is effective only when transmission is stopped. If the command is issued before setting Register 4 [Transmit Descriptor List Address Register], then the DMA behavior is unpredictable. When this bit is reset, the transmission process is placed in the Stopped state after completing the transmission of the current frame. The Next Descriptor position in the Transmit List is saved, and it becomes the current position when transmission is restarted. To change the list address, you need to program Register 4 [Transmit Descriptor List Address Register] with a new value when this bit is reset. The new value is considered when this bit is set again. The stop transmission command is effective only when the transmission of the current frame is complete or the transmission is in the Suspended state.

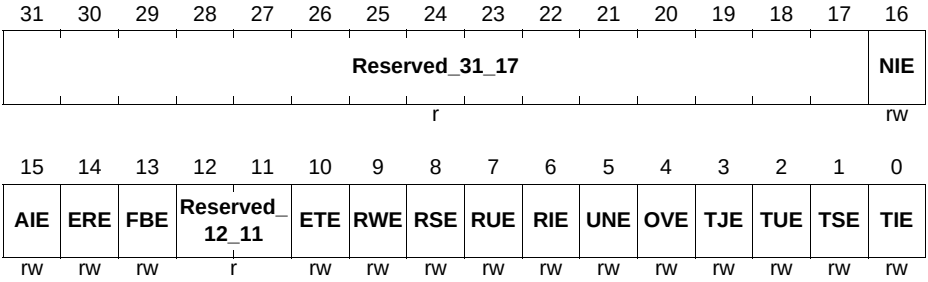
Field	Bits	Type	Description
TTC	[16:14]	rw	**Transmit Threshold Control** These bits control the threshold level of the MTL Transmit FIFO. Transmission starts when the frame size within the MTL Transmit FIFO is larger than the threshold. In addition, full frames with a length less than the threshold are also transmitted. These bits are used only when Bit 21 (TSF) is reset. * 000B: 64 * 001B: 128 * 010B: 192 * 011B: 256 * 100B: 40 * 101B: 32 * 110B: 24 * 111B: 16
Reserved_19_17	[19:17]	r	**Reserved**
FTF	20	rw	**Flush Transmit FIFO** When this bit is set, the transmit FIFO controller logic is reset to its default values and thus all data in the Tx FIFO is lost or flushed. This bit is cleared internally when the flushing operation is completed. The Operation Mode register should not be written to until this bit is cleared. The data which is already accepted by the MAC transmitter is not flushed. It is scheduled for transmission and results in underflow and runt frame transmission. Note: The flush operation is complete only when the Tx FIFO is emptied of its contents and all the pending Transmit Status of the transmitted frames are accepted by the CPU. To complete this flush operation, the PHY transmit clock is required to be active.
TSF	21	rw	**Transmit Store and Forward** When this bit is set, transmission starts when a full frame resides in the MTL Transmit FIFO. When this bit is set, the TTC values specified in Bits[16:14] are ignored. This bit should be changed only when the transmission is stopped.
Reserved_23_22	[23:22]	r	**Reserved**

Field	Bits	Type	Description
DFF	24	rw	**Disable Flushing of Received Frames** When this bit is set, the Rx DMA does not flush any frames because of the unavailability of receive descriptors or buffers as it does normally when this bit is reset.
RSF	25	rw	**Receive Store and Forward** When this bit is set, the MTL reads a frame from the Rx FIFO only after the complete frame has been written to it, ignoring the RTC bits. When this bit is reset, the Rx FIFO operates in the cut-through mode, subject to the threshold specified by the RTC bits.
DT	26	rw	**Disable Dropping of TCP/IP Checksum Error Frames** When this bit is set, the MAC does not drop the frames which only have errors detected by the Receive Checksum Offload engine. Such frames do not have any errors (including FCS error) in the Ethernet frame received by the MAC but have errors only in the encapsulated payload. When this bit is reset, all error frames are dropped if the FEF bit is reset.
Reserved_31_27	[31:27]	r	**Reserved**

INTERRUPT_ENABLE

The Interrupt Enable register enables the interrupts reported by **ETH0_STATUS** Register. Setting a bit to 1 enables a corresponding interrupt. After a hardware or software reset, all interrupts are disabled.

ETH0_INTERRUPT_ENABLE
Interrupt Enable Register **($101C_H$)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
TIE	0	rw	**Transmit Interrupt Enable** When this bit is set with Normal Interrupt Summary Enable (Bit 16), the Transmit Interrupt is enabled. When this bit is reset, the Transmit Interrupt is disabled.
TSE	1	rw	**Transmit Stopped Enable** When this bit is set with Abnormal Interrupt Summary Enable (Bit 15), the Transmission Stopped Interrupt is enabled. When this bit is reset, the Transmission Stopped Interrupt is disabled.
TUE	2	rw	**Transmit Buffer Unvailable Enable** When this bit is set with Normal Interrupt Summary Enable (Bit 16), the Transmit Buffer Unavailable Interrupt is enabled. When this bit is reset, the Transmit Buffer Unavailable Interrupt is disabled.
TJE	3	rw	**Transmit Jabber Timeout Enable** When this bit is set with Abnormal Interrupt Summary Enable (Bit 15), the Transmit Jabber Timeout Interrupt is enabled. When this bit is reset, the Transmit Jabber Timeout Interrupt is disabled.

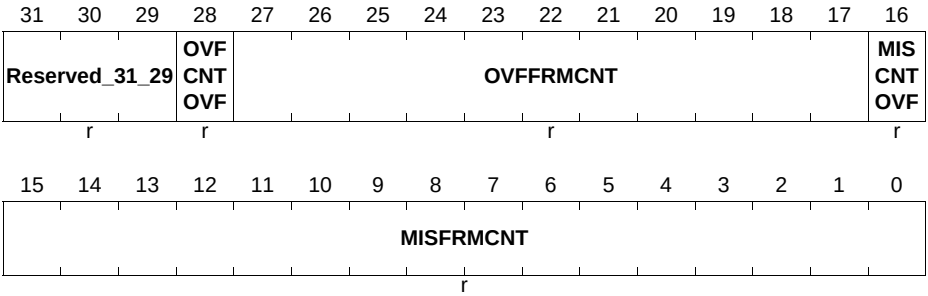
Field	Bits	Type	Description
OVE	4	rw	**Overflow Interrupt Enable** When this bit is set with Abnormal Interrupt Summary Enable (Bit 15), the Receive Overflow Interrupt is enabled. When this bit is reset, the Overflow Interrupt is disabled.
UNE	5	rw	**Underflow Interrupt Enable** When this bit is set with Abnormal Interrupt Summary Enable (Bit 15), the Transmit Underflow Interrupt is enabled. When this bit is reset, the Underflow Interrupt is disabled.
RIE	6	rw	**Receive Interrupt Enable** When this bit is set with Normal Interrupt Summary Enable (Bit 16), the Receive Interrupt is enabled. When this bit is reset, the Receive Interrupt is disabled.
RUE	7	rw	**Receive Buffer Unavailable Enable** When this bit is set with Abnormal Interrupt Summary Enable (Bit 15), the Receive Buffer Unavailable Interrupt is enabled. When this bit is reset, the Receive Buffer Unavailable Interrupt is disabled.
RSE	8	rw	**Receive Stopped Enable** When this bit is set with Abnormal Interrupt Summary Enable (Bit 15), the Receive Stopped Interrupt is enabled. When this bit is reset, the Receive Stopped Interrupt is disabled.
RWE	9	rw	**Receive Watchdog Timeout Enable** When this bit is set with Abnormal Interrupt Summary Enable (Bit 15), the Receive Watchdog Timeout Interrupt is enabled. When this bit is reset, the Receive Watchdog Timeout Interrupt is disabled.
ETE	10	rw	**Early Transmit Interrupt Enable** When this bit is set with an Abnormal Interrupt Summary Enable (Bit 15), the Early Transmit Interrupt is enabled. When this bit is reset, the Early Transmit Interrupt is disabled.
Reserved_12_11	[12:11]	r	**Reserved**

Field	Bits	Type	Description
FBE	13	rw	**Fatal Bus Error Enable** When this bit is set with Abnormal Interrupt Summary Enable (Bit 15), the Fatal Bus Error Interrupt is enabled. When this bit is reset, the Fatal Bus Error Enable Interrupt is disabled.
ERE	14	rw	**Early Receive Interrupt Enable** When this bit is set with Normal Interrupt Summary Enable (Bit 16), the Early Receive Interrupt is enabled. When this bit is reset, the Early Receive Interrupt is disabled.
AIE	15	rw	**Abnormal Interrupt Summary Enable** When this bit is set, abnormal interrupt summary is enabled. When this bit is reset, the abnormal interrupt summary is disabled. This bit enables the following interrupts in STATUS Register: * Transmit Process Stopped * Transmit Jabber Timeout * Receive Overflow * Transmit Underflow * Receive Buffer Unavailable * Receive Process Stopped * Receive Watchdog Timeout * Early Transmit Interrupt * Fatal Bus Error
NIE	16	rw	**Normal Interrupt Summary Enable** When this bit is set, normal interrupt summary is enabled. When this bit is reset, normal interrupt summary is disabled. This bit enables the following interrupts in Register 5 [STATUS Register]: * Transmit Interrupt * Transmit Buffer Unavailable * Receive Interrupt * Early Receive Interrupt
Reserved_31_17	[31:17]	r	**Reserved**

MISSED_FRAME_AND_BUFFER_OVERFLOW_COUNTER

The DMA maintains two counters to track the number of frames missed during reception. This register reports the current value of the counter. The counter is used for diagnostic purposes. Bits[15:0] indicate missed frames because of the RAM buffer being unavailable. Bits[27:17] indicate missed frames because of buffer overflow conditions (MTL and MAC) and runt frames (good frames of less than 64 bytes) dropped by the MTL.

ETH0_MISSED_FRAME_AND_BUFFER_OVERFLOW_COUNTER
Missed Frame and Buffer Overflow Counter Register (1020_H) Reset Value: 0000 0000_H



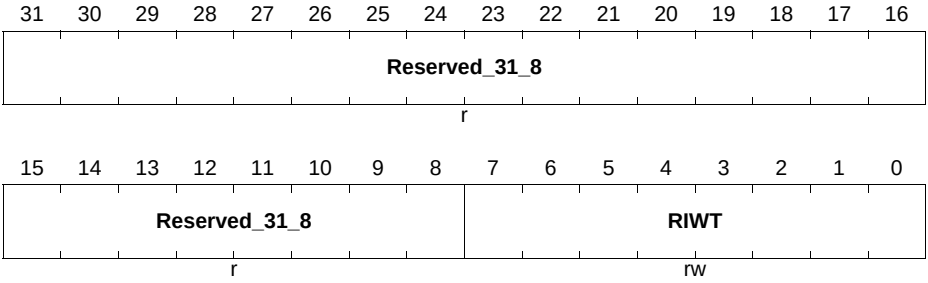
Field	Bits	Type	Description
MISFRMCNT	[15:0]	r	**This field indicates the number of frames missed by the controller because of the RAM Receive Buffer being unavailable. This counter is incremented each time the DMA discards an incoming frame. The counter is cleared when this register is read.**
MISCNTOVF	16	r	**Overflow bit for Missed Frame Counter**
OVFFRMCNT	[27:17]	r	**This field indicates the number of frames missed by the application. The counter is cleared when this register is read.**
OVFCNTOVF	28	r	**Overflow bit for FIFO Overflow Counter**
Reserved_31_29	[31:29]	r	**Reserved**

RECEIVE_INTERRUPT_WATCHDOG_TIMER

This register, when written with non-zero value, enables the watchdog timer for the Receive Interrupt (Bit 6) of STATUS Register]

ETH0_RECEIVE_INTERRUPT_WATCHDOG_TIMER

Receive Interrupt Watchdog Timer Register (1024_H) **Reset Value: $0000\ 0000_H$**

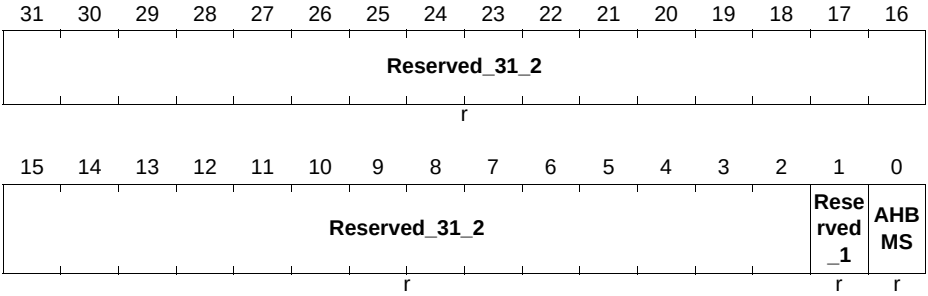


Field	Bits	Type	Description
RIWT	[7:0]	rw	**RI Watchdog Timer Count** This bit indicates the number of system clock cycles multiplied by 256 for which the watchdog timer is set. The watchdog timer gets triggered with the programmed value after the Rx DMA completes the transfer of a frame for which the RI status bit is not set because of the setting in the corresponding descriptor RDES1[31]. When the watchdog timer runs out, the RI bit is set and the timer is stopped. The watchdog timer is reset when the RI bit is set high because of automatic setting of RI as per RDES1[31] of any received frame.
Reserved_31_8	[31:8]	r	**Reserved**

AHB_Status

This register provides the active status of the AHB master interface. This register is useful for debugging purposes. In addition, this register is valid only in the Channel 0 DMA when multiple channels are present in the AV mode.

ETH0_AHB_Status
AHB Status Register **($102C_H$)** **Reset Value: 0000 0000$_H$**

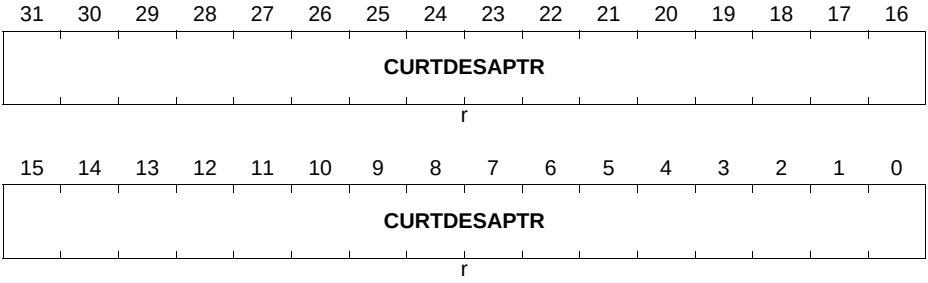


Field	Bits	Type	Description
AHBMS	0	r	**AHB Master Status** When high, it indicates that the AHB master interface FSMs are in the non-idle state.
Reserved_1	1	r	**Reserved**
Reserved_31_2	[31:2]	r	**Reserved**

CURRENT_HOST_TRANSMIT_DESCRIPTOR

The Current Host Transmit Descriptor register points to the start address of the current Transmit Descriptor read by the DMA.

ETH0_CURRENT_HOST_TRANSMIT_DESCRIPTOR
Current Host Transmit Descriptor Register (1048_H) **Reset Value: $0000\ 0000_H$**

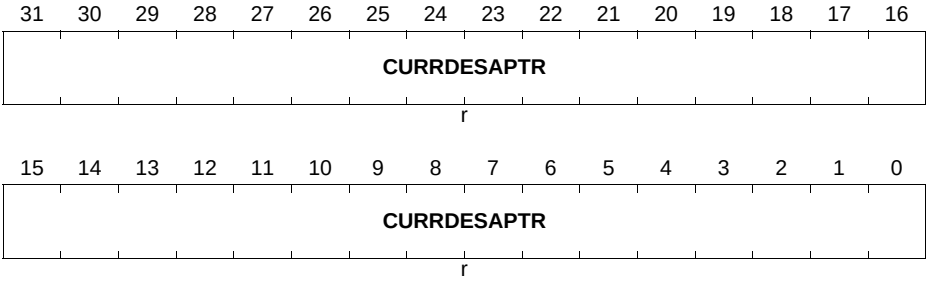


Field	Bits	Type	Description
CURTDESAPTR	[31:0]	r	**Host Transmit Descriptor Address Pointer** Cleared on Reset. Pointer updated by the DMA during operation.

CURRENT_HOST_RECEIVE_DESCRIPTOR

The Current Host Receive Descriptor register points to the start address of the current Receive Descriptor read by the DMA.

ETH0_CURRENT_HOST_RECEIVE_DESCRIPTOR
Current Host Receive Descriptor Register ($104C_H$) **Reset Value: $0000\ 0000_H$**



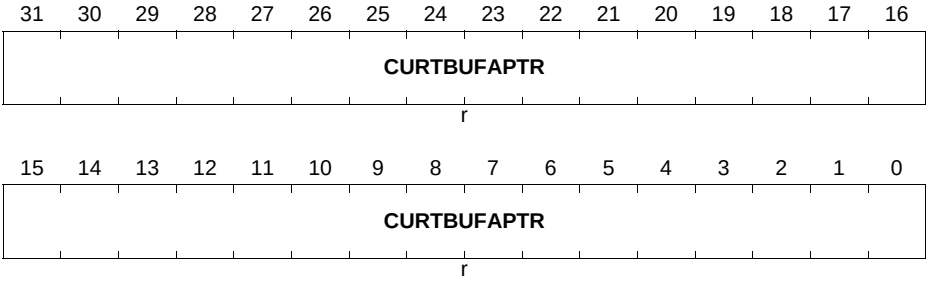
Field	Bits	Type	Description
CURRDESAPTR	[31:0]	r	**Host Receive Descriptor Address Pointer** Cleared on Reset. Pointer updated by the DMA during operation.

CURRENT_HOST_TRANSMIT_BUFFER_ADDRESS

The Current Host Transmit Buffer Address register points to the current Transmit Buffer Address being read by the DMA.

ETH0_CURRENT_HOST_TRANSMIT_BUFFER_ADDRESS

Current Host Transmit Buffer Address Register (1050_H) Reset Value: $0000\ 0000_H$

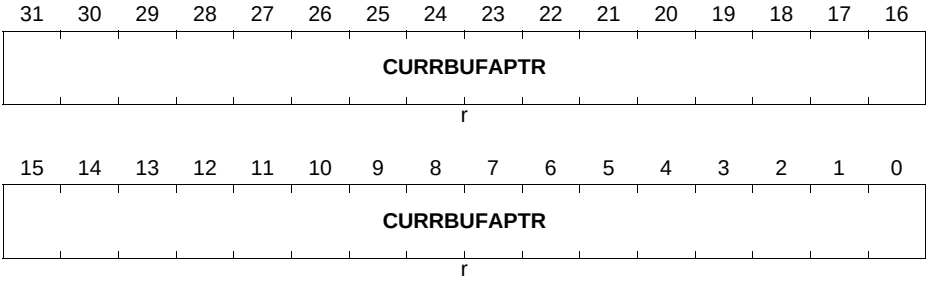


Field	Bits	Type	Description
CURTBUFAPTR	[31:0]	r	**Host Transmit Buffer Address Pointer** Cleared on Reset. Pointer updated by the DMA during operation.

CURRENT_HOST_RECEIVE_BUFFER_ADDRESS

The Current Host Receive Buffer Address register points to the current Receive Buffer address being read by the DMA.

ETH0_CURRENT_HOST_RECEIVE_BUFFER_ADDRESS
Current Host Receive Buffer Address Register (1054_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
CURRBUFAPTR	[31:0]	r	**Host Receive Buffer Address Pointer** Cleared on Reset. Pointer updated by the DMA during operation.

HW_FEATURE

This register indicates the presence of the optional features or functions of the DWC_ETH. The software driver can use this register to dynamically enable or disable the programs related to the optional blocks. Note: All bits are set or reset as per the selection of features during the DWC_ETH configuration.

ETH0_HW_FEATURE
HW Feature Register **(1058_H)** **Reset Value: 0305 2F35$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved_31	ACTPHYIF			SAVLANINS	FLEXIPPSEN	INTTSEN	ENHDESSEL	TXCHCNT		RXCHCNT		RXFIFOSIZE	RXTYP2COE	RXTYP1COE	TXCOESEL
r	r			r	r	r	r	r		r		rw	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AVSEL	EEESEL	TSVER2SEL	TSVER1SEL	MMCSEL	MGKSEL	RWKSEL	SMASEL	L3L4FLTREN	PCSSEL	ADDMACADRSEL	HASHSEL	EXTHASHEN	HDSEL	GMIISEL	MIISEL
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Field	Bits	Type	Description
MIISEL	0	r	**10 or 100 Mbps support**
GMIISEL	1	r	**1000 Mbps support**
HDSEL	2	r	**Half-Duplex support**
EXTHASHEN	3	r	**Expanded DA Hash Filter**
HASHSEL	4	r	**HASH Filter**
ADDMACADRSEL	5	r	**Multiple MAC Address Registers**
PCSSEL	6	r	**PCS registers (TBI, SGMII, or RTBI PHY interface)**
L3L4FLTREN	7	r	**Layer 3 and Layer 4 Filter Feature**
SMASEL	8	r	**SMA (MDIO) Interface**
RWKSEL	9	r	**PMT Remote Wakeup**
MGKSEL	10	r	**PMT Magic Packet**
MMCSEL	11	r	**RMON Module**

Field	Bits	Type	Description
TSVER1SEL	12	r	**Only IEEE 1588-2002 Timestamp**
TSVER2SEL	13	r	**IEEE 1588-2008 Advanced Timestamp**
EEESEL	14	r	**Energy Efficient Ethernet**
AVSEL	15	r	**AV Feature**
TXCOESEL	16	r	**Checksum Offload in Tx**
RXTYP1COE	17	r	**IP Checksum Offload (Type 1) in Rx**
RXTYP2COE	18	r	**IP Checksum Offload (Type 2) in Rx**
RXFIFOSIZE	19	rw	**Rx FIFO > 2,048 Bytes**
RXCHCNT	[21:20]	r	**Number of additional Rx channels**
TXCHCNT	[23:22]	r	**Number of additional Tx channels**
ENHDESSEL	24	r	**Alternate (Enhanced Descriptor)**
INTTSEN	25	r	**Timestamping with Internal System Time**
FLEXIPPSEN	26	r	**Flexible Pulse-Per-Second Output**
SAVLANINS	27	r	**Source Address or VLAN Insertion**
ACTPHYIF	[30:28]	r	**Active or Selected PHY interface** When you have multiple PHY interfaces in your configuration, this field indicates the sampled value of phy_intf_sel_i during reset de-assertion * 0000: MII * 0001: RMII * All Others: Reserved
Reserved_31	31	r	**Reserved**

15.7 Interconnects

The tables that refer to the “global pins” are the ones that contain the inputs/outputs of the ETH.

The GPIO connections are available in the Ports chapter.

15.7.1 ETH Pins

Table 15-28 ETH Pin Connectionsfor MIII

Global Inputs/Outputs	I/O	Connected To	Description
Control Signals			
ETH0.CRS(A)	I/O	PORT	Carrier Sense
ETH0.CRS(B)	I/O	PORT	Carrier Sense
ETH0.CRS(C)	I/O	PORT	Carrier Sense
ETH0.CRS(D)	I/O	PORT	Carrier Sense
ETH0.COL(A)	I/O	PORT	Collision Detect
ETH0.COL(B)	I/O	PORT	Collision Detect
ETH0.COL(C)	I/O	PORT	Collision Detect
ETH0.COL(D)	I/O	PORT	Collision Detect
ETH0.RXDV(A)	I/O	PORT	Receive Data Valid
ETH0.RXDV(B)	I/O	PORT	Receive Data Valid
ETH0.RXDV(C)	I/O	PORT	Receive Data Valid
ETH0.RXDV(D)	I/O	PORT	Receive Data Valid
ETH0.RXER(A)	I/O	PORT	Receive error
ETH0.RXER(B)	I/O	PORT	Receive error
ETH0.RXER(C)	I/O	PORT	Receive error
ETH0.RXER(D)	I/O	PORT	Receive error
Data Bus			
ETH0.RXD0(A)	I/O	PORT	Recieve data line
ETH0.RXD0(B)	I/O	PORT	Recieve data line
ETH0.RXD0(C)	I/O	PORT	Recieve data line
ETH0.RXD0(D)	I/O	PORT	Recieve data line
ETH0.RXD1(A)	I/O	PORT	Recieve data line
ETH0.RXD1(B)	I/O	PORT	Recieve data line

Table 15-28 ETH Pin Connectionsfor MIII (cont'd)

Global Inputs/Outputs	I/O	Connected To	Description
ETH0.RXD1(C)	I/O	PORT	Recieve data line
ETH0.RXD1(D)	I/O	PORT	Recieve data line
ETH0.RXD2(A)	I/O	PORT	Recieve data line
ETH0.RXD2(B)	I/O	PORT	Recieve data line
ETH0.RXD2(C)	I/O	PORT	Recieve data line
ETH0.RXD2(D)	I/O	PORT	Recieve data line
ETH0.RXD3(A)	I/O	PORT	Receive data line
ETH0.RXD3(B)	I/O	PORT	Receive data line
ETH0.RXD3(C)	I/O	PORT	Receive data line
ETH0.RXD3(D)	I/O	PORT	Receive data line
ETH0.TXEN(A)	I/O	PORT	Transmit enable
ETH0.TXEN(B)	I/O	PORT	Transmit enable
ETH0.TXEN(C)	I/O	PORT	Transmit enable
ETH0.TXEN(D)	I/O	PORT	Transmit enable
ETH0.TXER(A)	I/O	PORT	Transmit error
ETH0.TXER(B)	I/O	PORT	Transmit error
ETH0.TXER(C)	I/O	PORT	Transmit error
ETH0.TXER(D)	I/O	PORT	Transmit error
ETH0.TXD0(A)	I/O	PORT	Transmit Data Line
ETH0.TXD0(B)	I/O	PORT	Transmit Data Line
ETH0.TXD0(C)	I/O	PORT	Transmit Data Line
ETH0.TXD0(D)	I/O	PORT	Transmit Data Line
ETH0.TXD1(A)	I/O	PORT	Transmit data line
ETH0.TXD1(B)	I/O	PORT	Transmit data line
ETH0.TXD1(C)	I/O	PORT	Transmit data line
ETH0.TXD1(D)	I/O	PORT	Transmit data line
ETH0.TXD2(A)	I/O	PORT	Transmit data line
ETH0.TXD2(B)	I/O	PORT	Transmit data line
ETH0.TXD2(C)	I/O	PORT	Transmit data line
ETH0.TXD2(D)	I/O	PORT	Transmit data line

Table 15-28 ETH Pin Connectionsfor MIII (cont'd)

Global Inputs/Outputs	I/O	Connected To	Description
ETH0.TXD3(A)	I/O	PORT	Transmit data line
ETH0.TXD3(B)	I/O	PORT	Transmit data line
ETH0.TXD3(C)	I/O	PORT	Transmit data line
ETH0.TXD3(D)	I/O	PORT	Transmit data line
PHY Clocks			
ETH0.CLKTX(A)	I	PORT	PHY transmit clock
ETH0.CLKTX(B)	I	PORT	PHY transmit clock
ETH0.CLKTX(C)	I	PORT	PHY transmit clock
ETH0.CLKTX(D)	I	PORT	PHY transmit clock
ETH0.CLKRX(A)	I	PORT	PHY receive clock
ETH0.CLKRX(B)	I	PORT	PHY receive clock
ETH0.CLKRX(C)	I	PORT	PHY receive clock
ETH0.CLKRX(D)	I	PORT	PHY receive clock

Table 15-29 ETH Pin Connectionsfor RMIII

Global Inputs/Outputs	I/O	Connected To	Description
Control Signals			
ETH0.CRS_DV(A)	I/O	PORT	Carrier Sense Data Valid
ETH0.CRS_DV(B)	I/O	PORT	Carrier Sense Data Valid
ETH0.CRS_DV(C)	I/O	PORT	Carrier Sense Data Valid
ETH0.CRS_DV(D)	I/O	PORT	Carrier Sense Data Valid
ETH0.RXER(A)	I/O	PORT	Receive error
ETH0.RXER(B)	I/O	PORT	Receive error
ETH0.RXER(C)	I/O	PORT	Receive error
ETH0.RXER(D)	I/O	PORT	Receive error
Data Bus			
ETH0.RXD0(A)	I/O	PORT	Receive data line
ETH0.RXD0(B)	I/O	PORT	Receive data line
ETH0.RXD0(C)	I/O	PORT	Receive data line

Table 15-29 ETH Pin Connectionsfor RMIII (cont'd)

Global Inputs/Outputs	I/O	Connected To	Description
ETH0.RXD0(D)	I/O	PORT	Receive data line
ETH0.RXD1(A)	I/O	PORT	Receive data line
ETH0.RXD1(B)	I/O	PORT	Receive data line
ETH0.RXD1(C)	I/O	PORT	Receive data line
ETH0.RXD1(D)	I/O	PORT	Receive data line
ETH0.TXEN(A)	I/O	PORT	Transmit enable
ETH0.TXEN(B)	I/O	PORT	Transmit enable
ETH0.TXEN(C)	I/O	PORT	Transmit enable
ETH0.TXEN(D)	I/O	PORT	Transmit enable
ETH0.TXD0(A)	I/O	PORT	Transmit data line
ETH0.TXD0(B)	I/O	PORT	Transmit data line
ETH0.TXD0(C)	I/O	PORT	Transmit data line
ETH0.TXD0(D)	I/O	PORT	Transmit data line
ETH0.TXD1(A)	I/O	PORT	Transmit data line
ETH0.TXD1(B)	I/O	PORT	Transmit data line
ETH0.TXD1(C)	I/O	PORT	Transmit data line
ETH0.TXD1(D)	I/O	PORT	Transmit data line
PHY Clocks			
ETH0.CLK_RMII(A)	I	PORT	PHY Clock
ETH0.CLK_RMII(B)	I	PORT	PHY Clock
ETH0.CLK_RMII(C)	I	PORT	PHY Clock
ETH0.CLK_RMII(D)	I	PORT	PHY Clock

Table 15-30 ETH Pin Connectionsfor MDIO

Global Inputs/Outputs	I/O	Connected To	Description
Clock			
ETH0.MDC(A)	O	PORT	Management Data Clock
ETH0.MDC(B)	O	PORT	Management Data Clock
ETH0.MDC(C)	O	PORT	Management Data Clock

Table 15-30 ETH Pin Connectionsfor MDIO (cont'd)

Global Inputs/Outputs	I/O	Connected To	Description
ETH0.MDC(D)	O	PORT	Management Data Clock
Data			
ETH0.MDIO(A)	I/O	PORT	Management Data I/O
ETH0.MDIO(B)	I/O	PORT	Management Data I/O
ETH0.MDIO(C)	I/O	PORT	Management Data I/O
ETH0.MDIO(D)	I/O	PORT	Management Data I/O

16 Universal Serial Bus (USB)

The USB module is a Dual-Role Device (DRD) controller that supports both device and host functions and complies fully with the On-The-Go Supplement to the USB 2.0 Specification, Revision 1.3. It can also be configured as a host-only or device-only controller, fully compliant with the USB 2.0 Specification.

The USB core's USB 2.0 configurations support full-speed (12-Mbps) transfers.

The USB core is optimized for the following applications and systems:

- Portable electronic devices
- Point-to-point applications (direct connection to FS device)

References:

[15] USB 2.0 specification (April 27, 2000).

[16] On-The-Go Supplement to the USB 2.0 specification (Revision 1.3, December 5, 2006).

16.1 Overview

This section describes the features and provides a block diagram of the USB module.

16.1.1 Features

The USB module includes the following features:

- Complies with the USB 2.0 Specification
- Complies with the On-The-Go Supplement to the USB 2.0 Specification (Revision 1.3)
- Configurable as Device only, Host only or as an OTG Dual Role Device
- Support for the Full-Speed (12-Mbps) mode
- Provides a USB OTG FS PHY interface
- Supports up to 7 bidirectional endpoints, including control endpoint 0
- Supports up to 14 Host channels
- Supports Session Request Protocol (SRP).
- Supports Host Negotiation Protocol (HNP).
- Supports SOFs in Full-Speed modes.
- Supports clock gating for power saving
- Supports USB suspend/resume
- Supports DMA mode in:
  - Descriptor-Based Scatter/Gather DMA operation
  - Buffer DMA operation
- Dedicated transmit FIFO for each of the device IN endpoints in Slave and DMA modes. Each FIFO can hold multiple packets.

- Supports packet-based, Dynamic FIFO memory allocation for endpoints for small FIFOs and flexible, efficient use of RAM.
- Provides support to change an endpoint's FIFO memory size during transfers.
- Supports endpoint FIFO sizes that are not powers of 2, to allow the use of contiguous memory locations.

16.1.2 Block Diagram

Figure 16-1 shows the USB module block diagram.

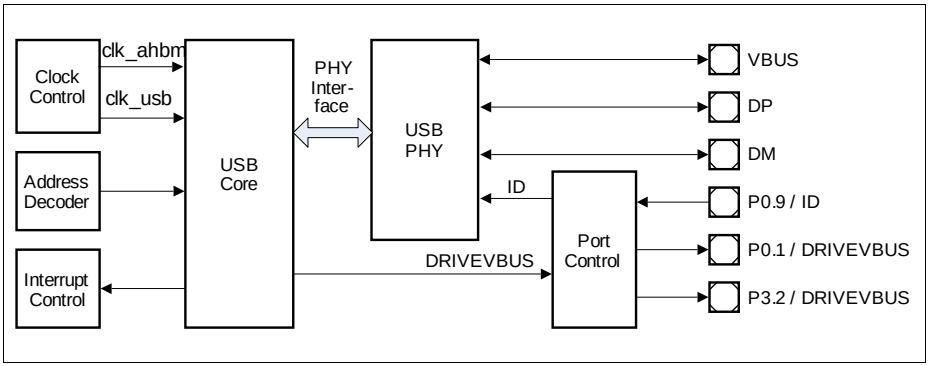


Figure 16-1 USB Module Block Diagram

16.2 Functional Description

This chapter describes the operation modes and the FIFO architecture of the USB module.

16.2.1 OTG Dual-Role Device (DRD)

The OTG DRD provides both Host and Device functions and supports the Host Negotiation Protocol (HNP) and Session Request Protocol (SRP). It is able to detect whether an A- or B-device is connected by sampling the ID input signal.

To drive the VBUS as an A-device, the OTG DRD requires an external charge pump, which is enabled through the output signal DRIVEBUS.

Figure 16-2 shows the connections of the DRD.

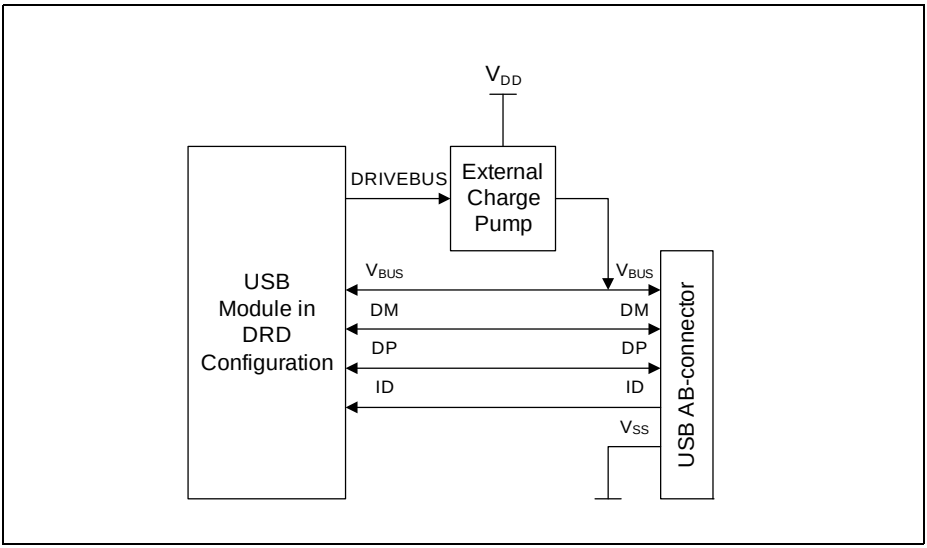


Figure 16-2 OTG DRD Connections

16.2.2 USB Host

The USB Host supports up to 14 Host channels, each configurable for the transfer type (Control, Bulk, Interrupt, or Isochronous) and direction (IN or OUT). To drive the VBUS, it requires an external charge pump, which is enabled through the output signal DRIVEBUS.

Figure 16-3 shows the connections of the USB Host.

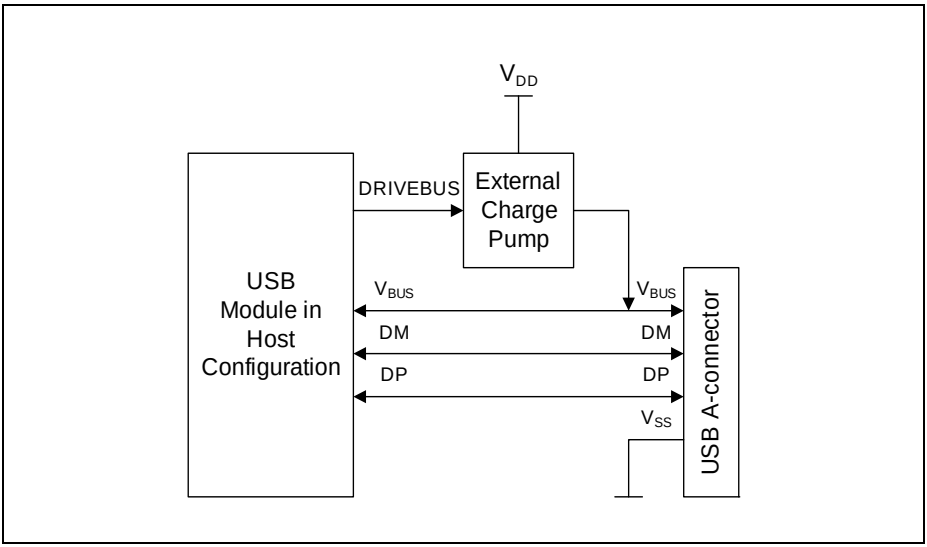


Figure 16-3 USB Host Connections

16.2.3 USB Device

The USB Device supports Control transfers through the bidirectional endpoint 0, and Bulk, Interrupt or Isochronous tranfers configurable from within the other 6 bidirectional endpoints. Being a self-powered device, it does not require an additional external voltage regulator.

Figure 16-4 shows the connections of the USB Device.

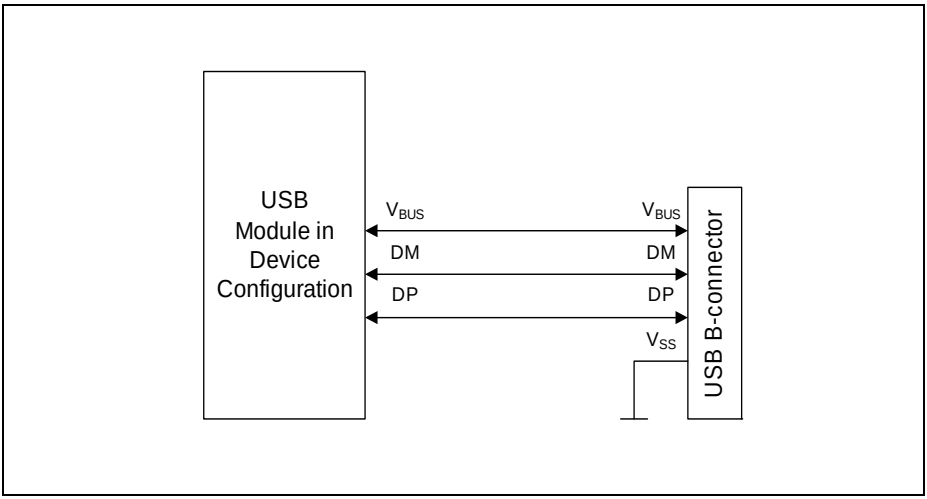


Figure 16-4 USB Device Connections

16.2.4 FIFO Architecture

This section describes the FIFO architecture in a USB Host and Device.

16.2.4.1 Host FIFO Architecture

The host uses one transmit FIFO for all non-periodic OUT transactions and one transmit FIFO for all periodic OUT transactions (periodic FIFOs 2 to n are only used in Device mode, where n is number of periodic IN endpoints in Device mode). These transmit FIFOs are used as transmit buffers to hold the data (payload of the transmit packet) to be transmitted over USB. The host pipes the USB transactions through Request queues (one for periodic and one for non-periodic). Each entry in the Request queue holds the IN or OUT channel number along with other information to perform a transaction on the USB. The order in which the requests are written into the queue determines the sequence of transactions on the USB. The host processes the periodic Request queue first, followed by the non-periodic Request queue, at the beginning of each frame.

The host uses one receive FIFO for all periodic and non-periodic transactions. The FIFO is used as a receive buffer to hold the received data (payload of the received packet) from the USB until it is transferred to the system memory. The status of each packet received also goes into the FIFO. The status entry holds the IN channel number along with other information, such as received byte count and validity status, to perform a transaction on the AHB.

16.2.4.2 Device FIFO Architecture

This section describes the USB device FIFO architecture.

Dedicated Transmit FIFO Operation

The core uses individual transmit FIFOs for each IN endpoint.

The core internally handles underrun condition during transmit and corrupts the packet (inverts the CRC) on the USB. If packet transmission results in an underrun condition (eventually resulting in packet corruption on the USB), the host can time out the endpoint after three consecutive errors.

Single Receive FIFO

The OTG device uses a single receive FIFO to receive the data for all the OUT endpoints. The receive FIFO holds the status of the received data packet, such as byte count, data PID and the validity of the received data. The DMA or the application reads the data out of the receive FIFO as it is received.

16.3 Programming Model

This chapter describes the programming requirements for the USB core in Host and Device modes.

16.3.1 Core Initialization

Each significant programming feature of the USB core is discussed in a separate section.

The application must perform the core initialization sequence. If the cable is connected during power-up, the Current Mode of Operation bit in the Core Interrupt register (GINTSTS.CurMod) reflects the mode. The USB core enters Host mode when an "A" plug is connected, or Device mode when a "B" plug is connected.

This section explains the initialization of the USB core after power-on. The application must follow the initialization sequence irrespective of Host or Device mode operation.

1. Program the following fields in the Global AHB Configuration (GAHBCFG) register.
   a) DMA Mode bit
   b) AHB Burst Length field
   c) Global Interrupt Mask bit = 1
   d) RxFIFO Non-Empty (GINTSTS.RxFLvl) (applicable only when the core is operating in Slave mode)
   e) Non-periodic TxFIFO Empty Level (can be enabled only when the core is operating in Slave mode as a host.)
   f) Periodic TxFIFO Empty Level (can be enabled only when the core is operating in Slave mode)
2. Program the following fields in GUSBCFG register.
   a) HNP Capable bit
   b) SRP Capable bit
   c) FS Time-Out Calibration field
   d) USB Turnaround Time field
3. The software must unmask the following bits in the GINTMSK register.
   a) OTG Interrupt Mask
   b) Mode Mismatch Interrupt Mask
4. The software can read the GINTSTS.CurMod bit to determine whether the USB core is operating in Host or Device mode. The software then follows either the **“Host Initialization” on Page 16-13** or **“Device Initialization” on Page 16-69** sequence.

Note: The core is designed to be interrupt-driven. Polling interrupt mechanism is not recommended: this may result in undefined resolutions.

16.3.2 Modes of Operation

16.3.2.1 Overview: DMA/Slave modes

The application can operate the core in either of two modes:

- In **DMA Mode** — the core fetches the data to be transmitted or updates the received data on the AHB.
- In **Slave Mode** — the application initiates the data transfers for data fetch and store.

The application cannot operate the core using a combination of DMA and Slave simultaneously. The application can operate the DMA in:

- Scatter/Gather mode (a Descriptor-Based mode).
- Buffer-pointer based mode.

After the option is selected for Scatter/Gather DMA with configuration-time programming, the mode can be controlled with the bit DescDMA, as described in **“Device Configuration Register (DCFG)” on Page 16-289**.

Note: The DescDMA field must be modified only once after a reset.

16.3.2.2 DMA Mode

In this mode, the OTG host uses the AHB master Interface for transmit packet data fetch (AHB to USB) and receive data update (USB to AHB). The AHB master uses the programmed DMA address (HCDMAx register in host mode and DIEPDMAx/DOEPDMAx register in device mode) to access the data buffers. When Scatter/Gather DMA is enabled in device mode DIEPDMAx/DOEPDMAx registers are used to access the base descriptor.

Transfer-Level Operation

In DMA mode, the application is interrupted only after the programmed transfer size is transmitted or received (provided the USB core detects no NAK/NYET/Timeout/Error response in Host mode, or Timeout/CRC Error in Device mode). The application must handle all transaction errors. In Device mode with dedicated FIFOs, all the USB errors are handled by the core itself.

Transaction-Level Operation

This mode is similar to transfer-level operation with the programmed transfer size equal to one packet size (either maximum packet size, or a short packet size). When Scatter/Gather DMA is enabled, the transfer size is extracted from the descriptors.

16.3.2.3 Slave Mode

In Slave mode, the application can operate the USB core either in transaction-level (packet-level) operation or in pipelined transaction-level operation.

Transaction-Level Operation

The application handles one data packet at a time per channel/endpoint in transaction-level operations. Based on the handshake response received on the USB, the application determines whether to retry the transaction or proceed with the next, until the end of the transfer. The application is interrupted on completion of every packet. The application performs transaction-level operations for a channel/endpoint for a transmission (host: OUT/ device: IN) or reception (host: IN / device: OUT) as shown in **Figure 16-5** and **Figure 16-6**.

Transaction-Level Operation: Host Mode

For an OUT transaction, the application enables the channel and writes the data packet into the corresponding (Periodic or Non-periodic) transmit FIFO. The USB core automatically writes the channel number into the corresponding (Periodic or Non-periodic) Request Queue, along with the last DWORD write of the packet. For an IN transaction, the application enables the channel and the USB core automatically writes the channel number into the corresponding Request queue. The application must wait for the packet received interrupt, then empty the packet from the receive FIFO.

Transaction-Level Operation: Device Mode

For an IN transaction, the application enables the endpoint, writes the data packet into the corresponding transmit FIFO, and waits for the packet completion interrupt from the core. For an OUT transaction, the application enables the endpoint, waits for the packet received interrupt from the core, then empties the packet from the receive FIFO.

Note: The application has to finish writing one complete packet before switching to a different channel/endpoint FIFO. Violating this rule results in an error

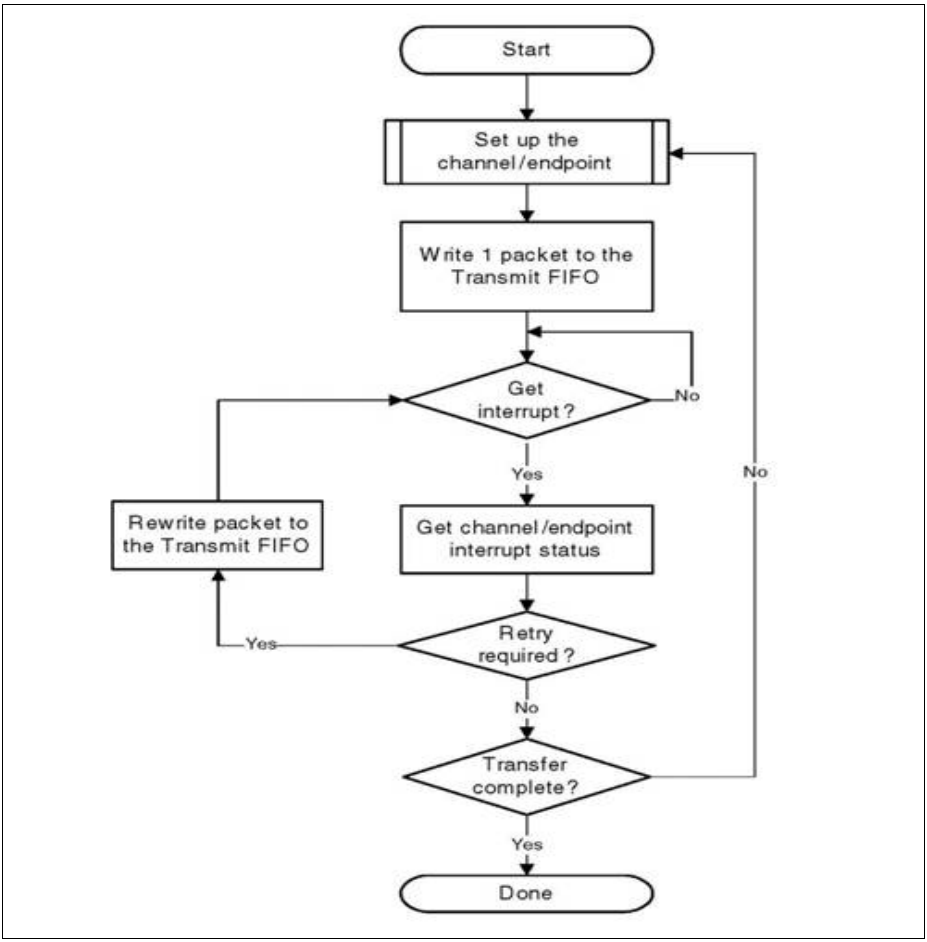


Figure 16-5 Transmit Transaction-Level Operation in Slave Mode

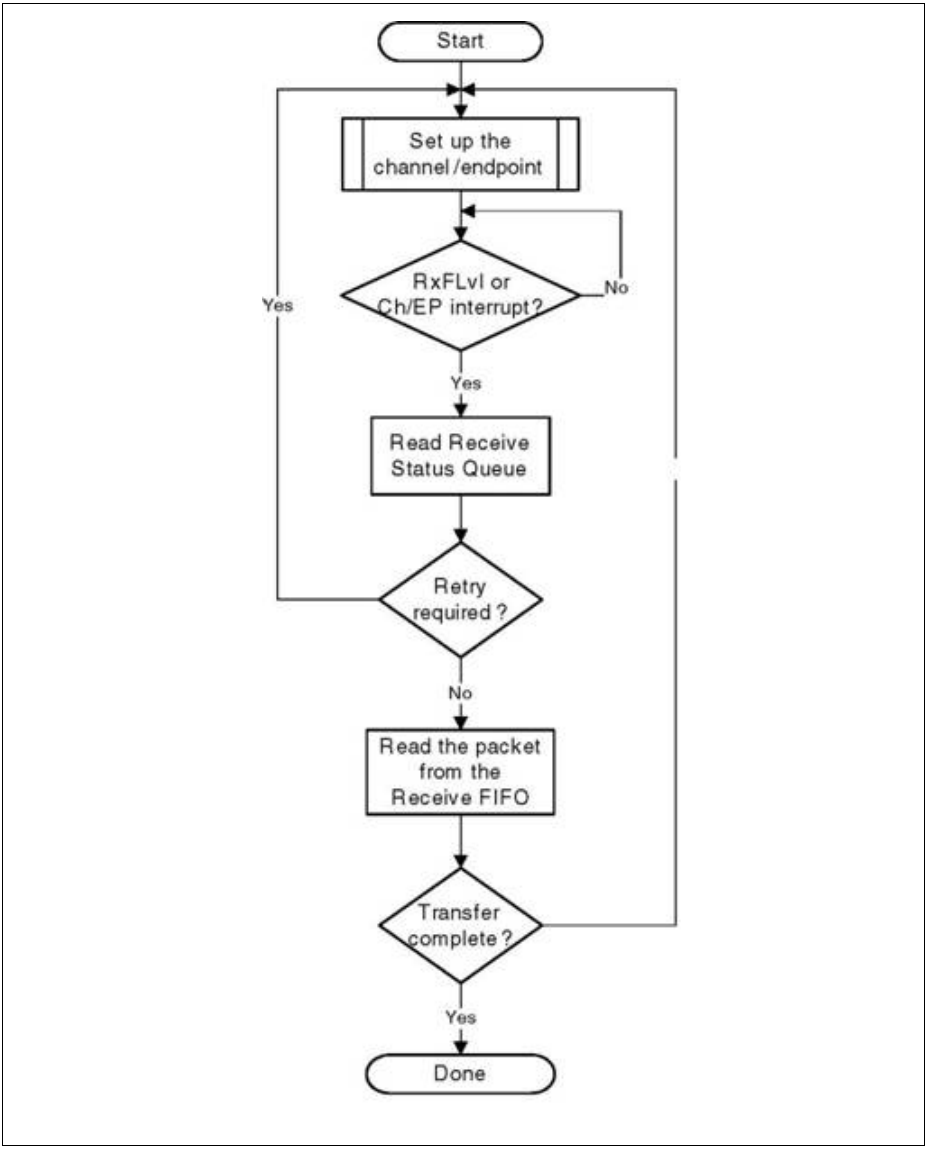


Figure 16-6 Receive Transaction-Level Operation in Slave Mode

Pipelined Transaction-Level Operation

The application can pipeline more than one transaction (IN or OUT) with pipelined transaction-level operation, which is analogous to Transfer mode in DMA mode. In pipelined transaction-level operation, the application can program the core to perform multiple transactions. The advantage of this mode compared to transaction-level operation is that the application is not interrupted on a packet basis.

Pipelined Transaction-Level Operation: Host Mode

For an OUT transaction, the application sets up a transfer and enables the channel. The application can write multiple packets back-to-back for the same channel into the transmit FIFO, based on the space availability. It can also pipeline OUT transactions for multiple channels by writing into the HCHARn register, followed by a packet write to that channel. The core writes the channel number, along with the last DWORD write for the packet, into the Request queue and schedules transactions on the USB in the same order.

For an IN transaction, the application sets up a transfer and enables the channel, and the USB core writes the channel number into the Request queue. The application can schedule IN transactions on multiple channels, provided space is available in the Request queue. The core initiates an IN token on the USB only when there is enough space to receive at least of one maximum-packet-size packet of the channel in the top of the Request queue.

Pipelined Transaction-Level Operation: Device Mode

For an IN transaction, the application sets up a transfer and enables the endpoint. The application can write multiple packets back-to-back for the same endpoint into the transmit FIFO, based on available space. It can also pipeline IN transactions for multiple channels by writing into the DIEPCTLx register followed by a packet write to that endpoint. The core writes the endpoint number, along with the last DWORD write for the packet into the Request queue. The core transmits the data in the transmit FIFO when an IN token is received on the USB.

For an OUT transaction, the application sets up a transfer and enables the endpoint. The core receives the OUT data into the receive FIFO, when it has available space. As the packets are received into the FIFO, the application must empty data from it.

From this point on in this chapter, the terms "Pipelined Transaction mode" and "Transfer mode" are used interchangeably.

16.4 Host Programming Model

16.4.1 Host Initialization

To initialize the core as host, the application must perform the following steps.

1. Program GINTMSK.PrtIntMsk to unmask.
2. Program the HCFG register to select full-speed host.
3. Program the HPRT.PrtPwr bit to 1_B. This drives VBUS on the USB.
4. Wait for the HPRT.PrtConnDet interrupt. This indicates that a device is connected to the port.
5. Program the HPRT.PrtRst bit to 1_B. This starts the reset process.
6. Wait at least 10 ms for the reset process to complete.
7. Program the HPRT.PrtRst bit to 0_B.
8. Wait for the HPRT.PrtEnChng interrupt.
9. Read the HPRT.PrtSpd field to get the enumerated speed.
10. Program the HFIR register with a value corresponding to the selected PHY clock.[1)]
11. Program the GRXFSIZ register to select the size of the receive FIFO.
12. Program the GNPTXFSIZ register to select the size and the start address of the Non-periodic Transmit FIFO for non-periodic transactions.
13. Program the HPTXFSIZ register to select the size and start address of the Periodic Transmit FIFO for periodic transactions.

To communicate with devices, the system software must initialize and enable at least one channel as described in **“Channel Initialization” on Page 16-13**".

16.4.2 Channel Initialization

The application must initialize one or more channels before it can communicate with connected devices. To initialize and enable a channel, the application must perform the following steps.

1. Program the GINTMSK register to unmask the following:
2. Channel Interrupt
   a) Non-periodic Transmit FIFO Empty for OUT transactions (applicable for Slave mode that operates in pipelined transaction-level with the Packet Count field programmed with more than one).
   b) Non-periodic Transmit FIFO Half-Empty for OUT transactions (applicable for Slave mode that operates in pipelined transaction-level with the Packet Count field programmed with more than one).
3. Program the HAINTMSK register to unmask the selected channels' interrupts.
4. Program the HCINTMSK register to unmask the transaction-related interrupts of interest given in the Host Channel Interrupt register.
5. Program the selected channel's HCTSIZx register.
   Note that this step is not applicable in Scatter/Gather DMA mode. Program the

1) At this point, the host is up and running and the port register begins to report device disconnects, etc. The port is active with SOFs occurring down the enabled port.

register with the total transfer size, in bytes, and the expected number of packets, including short packets. The application must program the PID field with the initial data PID (to be used on the first OUT transaction or to be expected from the first IN transaction).

6. Program the selected channels' HCDMAx register(s) with the buffer start address (Scatter/Gather DMA mode only).
7. Program the HCCHARx register of the selected channel with the device's endpoint characteristics, such as type, speed, direction, and so forth. (The channel can be enabled by setting the Channel Enable bit to 1_B only when the application is ready to transmit or receive any packet).

Repeat steps 1-7 for other channels.

Note: De-allocate channel means after the transfer has completed, the channel is disabled. When the application is ready to start the next transfer, the application re-initializes the channel by following these steps.

16.4.3 Halting a Channel

The application can disable any channel by programming the HCCHARx register with the HCCHARx.ChDis and HCCHARx.ChEna bits set to 1_B. This enables the USB host to flush the posted requests (if any) and generates a Channel Halted interrupt. The application must wait for the HCINTx.ChHltd interrupt before reallocating the channel for other transactions. The USB host does not interrupt the transaction that has been already started on USB.

In Slave mode operation, before disabling a channel, the application must ensure that there is at least one free space available in the Non-periodic Request Queue (when disabling a non-periodic channel) or the Periodic Request Queue (when disabling a periodic channel). The application can simply flush the posted requests when the Request queue is full (before disabling the channel), by programming the HCCHARx register with the HCCHARx.ChDis bit set to 1_B, and the HCCHARx.ChEna bit reset to 0_B.

The core generates a RxFLvl interrupt when there is an entry in the queue. The application must read/pop the GRXSTSP register to generate the Channel Halted interrupt.

To disable a channel in DMA mode operation, the application need not check for space in the Request queue. The USB host checks for space in which to write the Disable request on the disabled channel's turn during arbitration. Meanwhile, all posted requests are dropped from the Request queue when the HCCHARx.ChDis bit is set to 1_B.

The application is expected to disable a channel under any of the following conditions:

1. When a HCINTx.XferCompl interrupt is received during a non-periodic IN transfer or high- bandwidth interrupt IN transfer (Slave mode only)
2. When a HCINTx.STALL, HCINTx.XactErr, HCINTx.BblErr, or HCINTx.DataTglErr interrupt is received for an IN or OUT channel (Slave mode only). For high-bandwidth

interrupt INs in Slave mode, once the application has received a DataTglErr interrupt it must disable the channel and wait for a Channel Halted interrupt. The application must be able to receive other interrupts (DataTglErr, Nak, Data, XactErr, BabbleErr) for the same channel before receiving the halt.

3. When a GINTSTS.DisconnInt (Disconnect Device) interrupt is received. The application must check for the HPRT.PrtConnSts, because when the device directly connected to the host is disconnected, HPRT.PrtConnSts is reset. The software must issue a soft reset to ensure that all channels are cleared. When the device is reconnected, the host must issue a USB Reset.
4. When the application aborts a transfer before normal completion (Slave and DMA modes).

Note

In buffer DMA mode, the following guidelines must be considered:

- Channel disable must not be programmed for non-split periodic channels. At the end of the next frame (in the worst case), the core generates a channel halted and disables the channel automatically.
- For split enabled channels (both non-periodic and periodic), channel disable must not be programmed randomly. However, channel disable can be programmed for specific scenarios such as NAK and FrmOvrn as defined in the Host programming model.

16.4.4 Selecting the Queue Depth

Choose the Periodic and Non-periodic Request Queue depths carefully to match the number of periodic/non-periodic endpoints accessed.

The Non-periodic Request Queue depth affects the performance of non-periodic transfers. The deeper the queue (along with sufficient FIFO size), the more often the core is able to pipeline non-periodic transfers. If the queue size is small, the core is able to put in new requests only when the queue space is freed up.

The core's Periodic Request Queue depth is critical to performing periodic transfers as scheduled. Select the periodic queue depth, based on the number of periodic transfers scheduled in a frame. In Slave mode, however, the application must also take into account the disable entry that must be put into the queue. So, if there are two non-high-bandwidth periodic endpoints, the Periodic Request Queue depth must be at least 4. If at least one high-bandwidth endpoint supported, the queue depth must be 8. If the Periodic Request Queue depth is smaller than the periodic transfers scheduled in a frame, a frame overrun condition results.

16.4.5 Handling Babble Conditions

USB core handles two cases of babble: packet babble and port babble. Packet babble occurs if the device sends more data than the maximum packet size for the channel. Port babble occurs if the core continues to receive data from the device at EOF2 (the end of frame 2, which is very close to SOF).

When USB core detects a packet babble, it stops writing data into the Rx buffer and waits for the end of packet (EOP). When it detects an EOP, it flushes already-written data in the Rx buffer and generates a Babble interrupt to the application.

When USB core detects a port babble, it flushes the RxFIFO and disables the port. The core then generates a Port Disabled Interrupt (GINTSTS.PrtInt, HPRT.PrtEnChng). On receiving this interrupt, the application must determine that this is not due to an overcurrent condition (another cause of the Port Disabled interrupt) by checking HPRT.PrtOvrCurrAct, then perform a soft reset. The core does not send any more tokens after it has detected a port babble condition.

16.4.6 Handling Disconnects

If the device is disconnected suddenly, a GINTSTS.DisconnInt interrupt is generated. When the application receives this interrupt, it must issue a soft reset by programming the GRSTCTL.CSftRst bit.

16.4.7 Host Programming Operations

Table 16-1 provides links to the programming sequence for the different types of USB transactions.

Table 16-1 Host Programming Operations

Mode	IN	OUT/SETUP
Control		
Slave	**“Bulk and Control IN Transactions in Host Slave Mode” on Page 16-23**	**“Bulk and Control OUT/SETUP Transactions in Host Slave Mode” on Page 16-20**
Buffer DMA	**“Bulk and Control IN Transactions in DMA Mode” on Page 16-31**	**“Bulk and Control OUT/SETUP Transactions in Host DMA Mode” on Page 16-25**
Scatter Gather DMA Mode	**“Asynchronous Transfers” on Page 16-62**"	**“Asynchronous Transfers” on Page 16-62**
Bulk		

Table 16-1 Host Programming Operations (cont'd)

Mode	IN	OUT/SETUP
Slave	**"Bulk and Control IN Transactions in Host Slave Mode" on Page 16-23**	**"Bulk and Control OUT/SETUP Transactions in Host Slave Mode" on Page 16-20**
Buffer DMA	**"Bulk and Control IN Transactions in DMA Mode" on Page 16-31**	**"Bulk and Control OUT/SETUP Transactions in Host DMA Mode" on Page 16-25**
Scatter Gather DMA Mode	**"Asynchronous Transfers" on Page 16-62"**	**"Asynchronous Transfers" on Page 16-62**
Interrupt		
Slave	**"Interrupt IN Transactions in Slave Mode" on Page 16-36**	**"Interrupt OUT Transactions in Slave Mode" on Page 16-33**
Buffer DMA	**"Interrupt IN Transactions in DMA Mode" on Page 16-41**	**"Interrupt OUT Transactions in DMA Mode" on Page 16-38**
Scatter Gather DMA Mode	**"Periodic Transfers" on Page 16-64**	**"Periodic Transfers" on Page 16-64**
Isochronous		
Slave	**"Isochronous IN Transactions in Slave Mode" on Page 16-46**	**"Isochronous OUT Transactions in Slave Mode" on Page 16-43**
Buffer DMA	**"Isochronous IN Transactions in Host DMA Mode" on Page 16-50**	**"Isochronous OUT Transactions in DMA Mode" on Page 16-48**
Scatter Gather DMA Mode	**"Periodic Transfers" on Page 16-64**	**"Periodic Transfers" on Page 16-64**

16.4.7.1 Writing the Transmit FIFO in Slave Mode

Figure 16-7 shows the flow diagram for writing to the transmit FIFO in Slave mode. The USB host automatically writes an entry (OUT request) to the Periodic/Non-periodic Request Queue, along with the last DWORD write of a packet. The application must ensure that at least one free space is available in the Periodic/Non-periodic Request Queue before starting to write to the transmit FIFO. The application must always write to the transmit FIFO in DWORDs. If the packet size is non-DWORD aligned, the application must use padding. The USB host determines the actual packet size based on the programmed maximum packet size and transfer size.

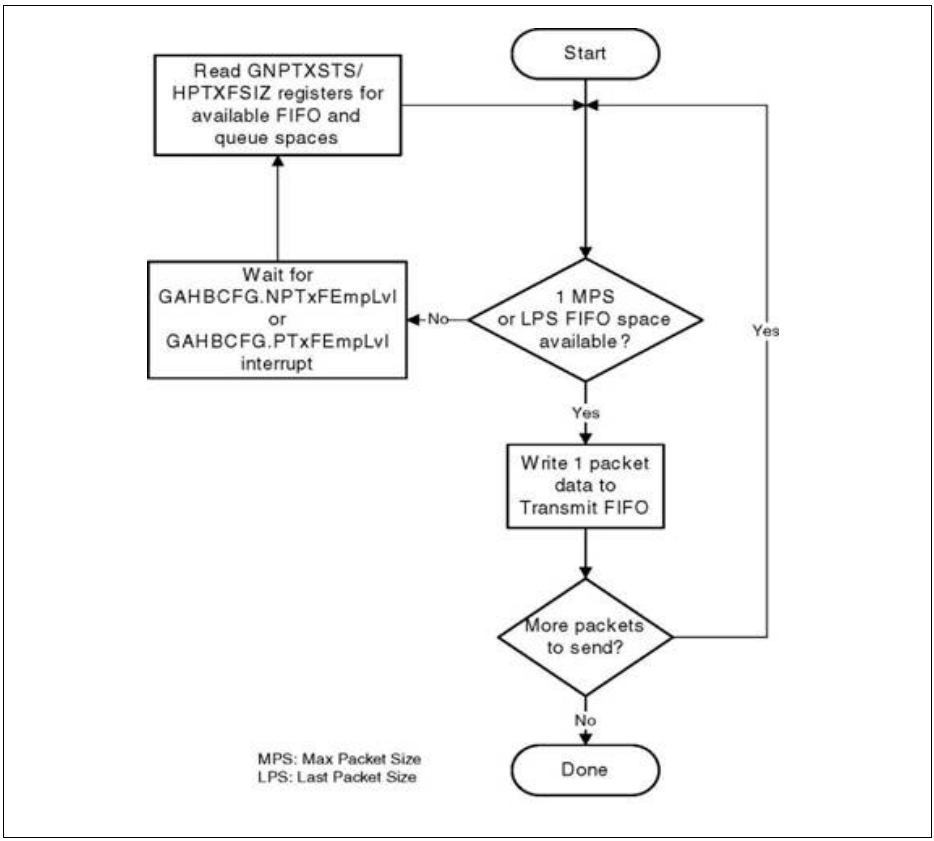


Figure 16-7 Transmit FIFO Write Task in Slave Mode

16.4.7.2 Reading the Receive FIFO in Slave Mode

Figure 16-8 shows the flow diagram for reading the receive FIFO in Slave mode. The application must ignore all packet statuses other than IN Data Packet (0010_B).

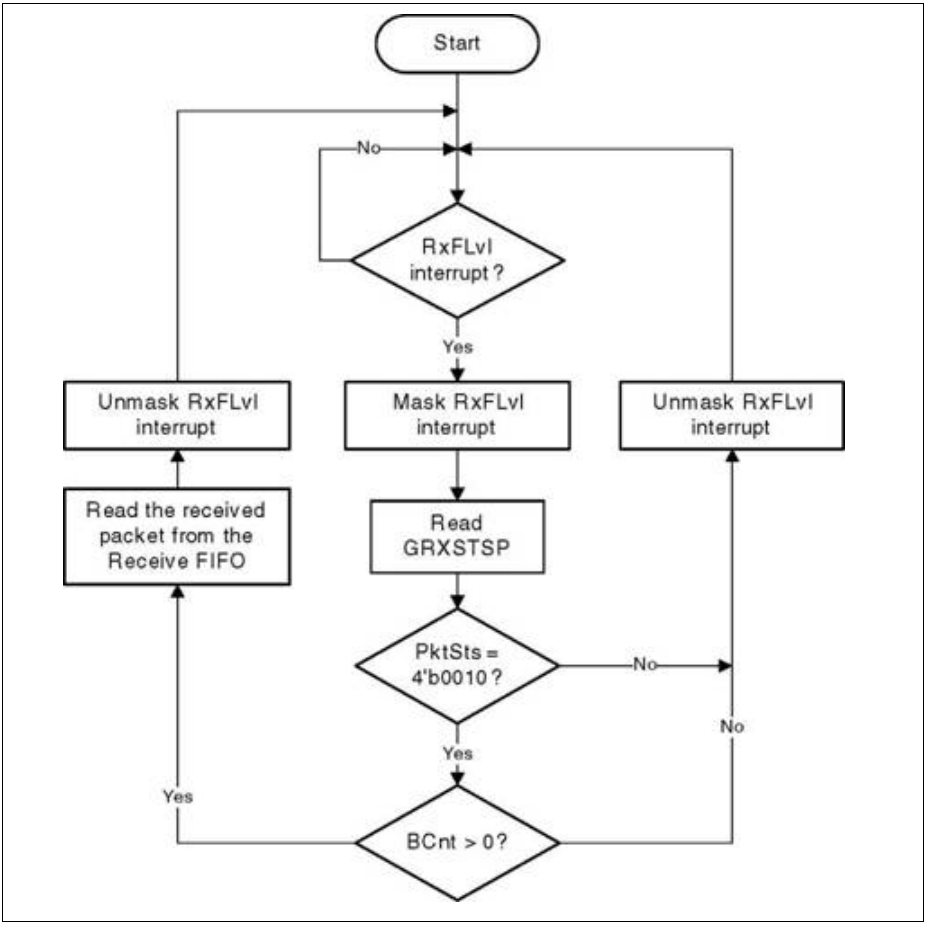


Figure 16-8 Receive FIFO Read Task in Slave Mode

16.4.7.3 Control Transactions in Host Slave Mode

Setup, Data, and Status stages of a control transfer must be performed as three separate transfers. Setup-, Data- or Status-stage OUT transactions are performed similarly to the bulk OUT transactions explained in **“Bulk and Control OUT/SETUP Transactions in Host Slave Mode” on Page 16-20**. Data- or Status-stage IN transactions are performed similarly to the bulk IN transactions explained in **“Bulk and Control IN Transactions in Host Slave Mode” on Page 16-23**. For all three stages, the application is expected to set the HCCHAR1.EPType field to Control. During the Setup stage, the application is expected to set the HCTSIZ1.PID field to SETUP.

16.4.7.4 Bulk and Control OUT/SETUP Transactions in Host Slave Mode

A typical bulk or control OUT/SETUP pipelined transaction-level operation in Slave mode is shown in **Figure 16-9**. See channel 1 (ch_1). Two bulk OUT packets are transmitted. A control SETUP transaction operates the same way but has only one packet. The assumptions are:

- The application is attempting to send two maximum-packet-size packets (transfer size = 1,024 bytes).
- The Non-periodic Transmit FIFO can hold two packets (128 bytes for FS).
- The Non-periodic Request Queue depth = 4.

Normal Bulk and Control OUT/SETUP Operations

The sequence of operations in **Figure 16-9** (channel 1) is as follows:

1. Initialize channel 1 as explained in **“Channel Initialization” on Page 16-13**.
2. Write the first packet for channel 1.
3. Along with the last DWORD write, the core writes an entry to the Non-periodic Request Queue.
4. As soon as the non-periodic queue becomes non-empty, the core attempts to send an OUT token in the current frame.
5. Write the second (last) packet for channel 1.
6. The core generates the XferCompl interrupt as soon as the last transaction is completed successfully.
7. In response to the XferCompl interrupt, de-allocate the channel for other transfers

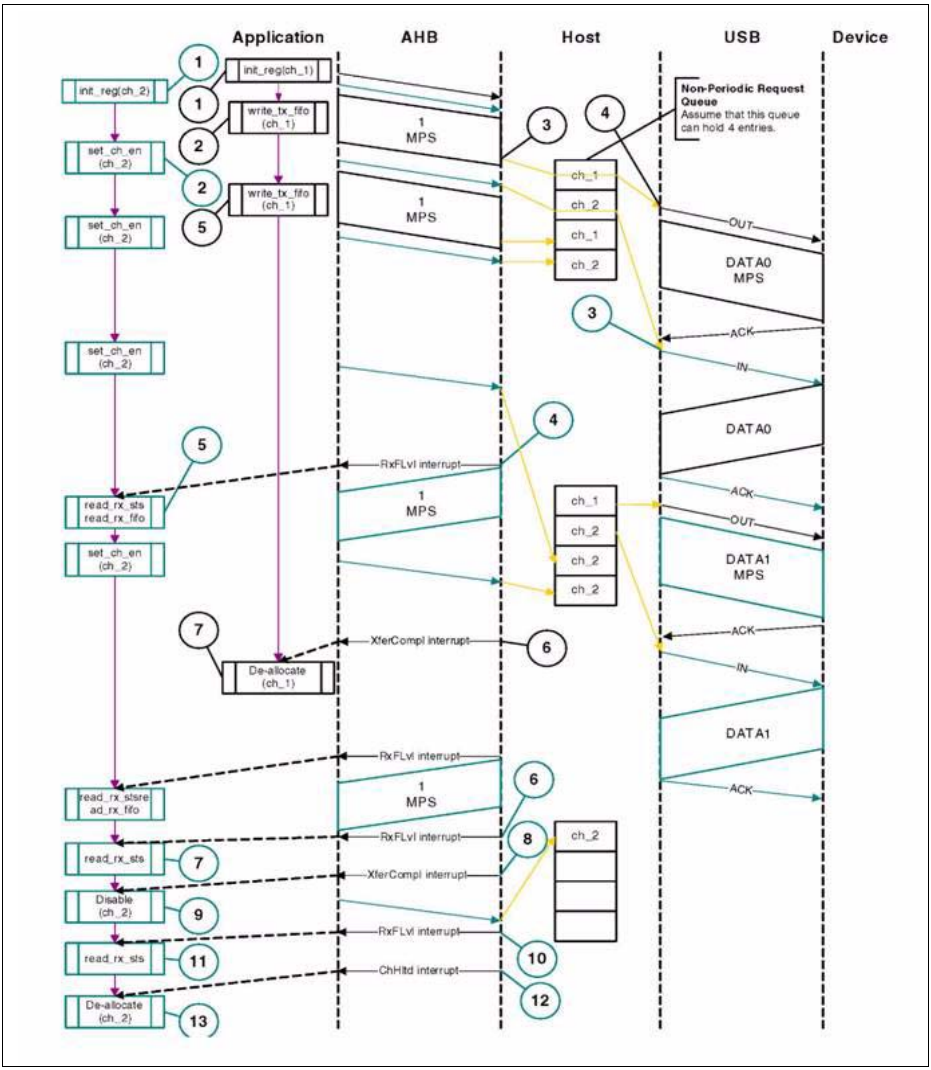


Figure 16-9 Normal Bulk/Control OUT/SETUP and Bulk/Control IN Transactions in Slave Mode

Handling Interrupts

The channel-specific interrupt service routine for bulk and control OUT/SETUP transactions in Slave mode is shown in the following code samples.

Interrupt Service Routine for Bulk/Control OUT/SETUP Transactions in Slave Mode

Bulk/Control OUT/SETUP

```
Unmask (NAK/XactErr/NYET/STALL/XferCompl)
if (XferCompl)
      {
      Reset Error Count
      Mask ACK
      De-allocate Channel
      }
else if (STALL)
      {
      Transfer Done = 1
      Unmask ChHltd
      Disable Channel
      }
else if (NAK or XactErr or NYET)
      {
      Rewind Buffer Pointers
      Unmask ChHltd
      Disable Channel
      if (XactErr)
            {
            Increment Error Count
            Unmask ACK
            }
      else
            {
            Reset Error Count
            }
      }
else if (ChHltd)
      {
      Mask ChHltd
      if (Transfer Done or (Error_count == 3))
```

```
            {
            De-allocate Channel
            }
      else
            {
            Re-initialize Channel
            }
      }
else if (ACK)
      {
      Reset Error Count
      Mask ACK
      }
```

The application is expected to write the data packets into the transmit FIFO when space is available in the transmit FIFO and the Request queue. The application can make use of GINTSTS.NPTxFEmp interrupt to find the transmit FIFO space.

The application is expected to write the requests as and when the Request queue space is available and until the XferCompl interrupt is received.

The application must clear and never modify the DoPing bit after enabling the channel and until the XferCompl or ChHltd interrupt is received. The core uses the DoPing bit to flush the excessive IN requests after receiving the last or short packet.

16.4.7.5 Bulk and Control IN Transactions in Host Slave Mode

A typical bulk or control IN pipelined transaction-level operation in Slave mode is shown in **Figure 16-9**. See channel 2 (ch_2). The assumptions are:

- The application is attempting to receive two maximum-sized packets (transfer size = 1,024 bytes).
- The receive FIFO can contain at least one maximum-packet-size packet and two status DWORDs per packet (72 bytes for FS).
- The Non-periodic Request Queue depth = 4.

Normal Bulk and Control IN Operations

The sequence of operations in **Figure 16-9** (channel 2) is as follows:

1. Initialize channel 2 as explained in **"Channel Initialization" on Page 16-13**.
2. Set the HCCHAR2.ChEna bit to write an IN request to the Non-periodic Request Queue.
3. The core attempts to send an IN token after completing the current OUT transaction.
4. The core generates an RxFLvl interrupt as soon as the received packet is written to the receive FIFO.

5. In response to the RxFLvl interrupt, mask the RxFLvl interrupt and read the received packet status to determine the number of bytes received, then read the receive FIFO accordingly. Following this, unmask the RxFLvl interrupt.
6. The core generates the RxFLvl interrupt for the transfer completion status entry in the receive FIFO.
7. The application must read and ignore the receive packet status when the receive packet status is not an IN data packet (GRXSTSR.PktSts! = 0010_B).
8. The core generates the XferCompl interrupt as soon as the receive packet status is read.
9. In response to the XferCompl interrupt, disable the channel (see **“Halting a Channel” on Page 16-14**) and stop writing the HCCHAR2 register for further requests. The core writes a channel disable request to the non-periodic request queue as soon as the HCCHAR2 register is written.
10. The core generates the RxFLvl interrupt as soon as the halt status is written to the receive FIFO.
11. Read and ignore the receive packet status.
12. The core generates a ChHltd interrupt as soon as the halt status is popped from the receive FIFO.
13. In response to the ChHltd interrupt, de-allocate the channel for other transfers.

Note: For Bulk/Control IN transfers, the application must write the requests when the Request queue space is available, and until the XferCompl interrupt is received.

Handling Interrupts

The channel-specific interrupt service routine for bulk and control IN transactions in Slave mode is shown in the following code samples.

Interrupt Service Routine for Bulk/Control IN Transactions in Slave Mode

```
Unmask (XactErr/XferCompl/BblErr/STALL/DataTglErr)
if (XferCompl)
      {
      Reset Error Count
      Unmask ChHltd
      Disable Channel
      Reset Error Count
      Mask ACK
      }
else if (XactErr or BblErr or STALL)
      {
      Unmask ChHltd
      Disable Channel
      if (XactErr)
            {
```

```
            Increment Error Count
            Unmask ACK
            }
      }
else if (ChHltd)
      {
      Mask ChHltd
      if (Transfer Done or (Error_count == 3
            {
            De-allocate Channel
            }
      else
            {
            Re-initialize Channel
            }
      }
else if (ACK)
      {
      Reset Error Count
      Mask ACK
      }
else if (DataTglErr)
      {
      Reset Error Count
      }
```

16.4.7.6 Control Transactions in Host DMA Mode

Setup, Data, and Status stages of a control transfer must be performed as three separate transfers. Setup- and Data- or Status-stage OUT transactions are performed similarly to the bulk OUT transactions explained in **“Bulk and Control OUT/SETUP Transactions in Host DMA Mode” on Page 16-25**. Data- or Status-stage IN transactions are performed similarly to the bulk IN transactions explained in **“Bulk and Control IN Transactions in DMA Mode” on Page 16-31**. For all three stages, the application is expected to set the HCCHAR1.EPType field to Control. During the Setup stage, the application is expected to set the HCTSIZ1.PID field to SETUP.

16.4.7.7 Bulk and Control OUT/SETUP Transactions in Host DMA Mode

This section discusses the following topics:

- **Overview**
- **Normal Bulk and Control OUT/SETUP Operations**
- **NAK and NYET Handling With Internal DMA**

- **Handling Interrupts**

Overview

- The application is attempting to send two maximum-packet-size packets (transfer size = 1,024 bytes).
- The Non-periodic Transmit FIFO can hold two packets (128 bytes for FS).
- The Non-periodic Request Queue depth = 4.

Normal Bulk and Control OUT/SETUP Operations

The sequence of operations in **Figure 16-9** (channel 1) is as follows:

1. Initialize and enable channel 1 as explained in **"Channel Initialization" on Page 16-13**.
2. The USB host starts fetching the first packet as soon as the channel is enabled. For internal DMA mode, the USB host uses the programmed DMA address to fetch the packet.
3. After fetching the last DWORD of the second (last) packet, the USB host masks channel 1 internally for further arbitration.
4. The USB host generates a ChHltd interrupt as soon as the last packet is sent.
5. In response to the ChHltd interrupt, de-allocate the channel for other transfers.

The channel-specific interrupt service routine for bulk and control OUT/SETUP transactions in DMA mode is shown in **"Handling Interrupts" on Page 16-29**.

NAK and NYET Handling With Internal DMA

1. The USB Host sends a Bulk OUT Transaction.
2. The Device responds with NAK or NYET.
3. If the application has unmasked NAK or NYET, the core generates the corresponding interrupt(s) to the application.
   The application is not required to service these interrupts, since the core takes care of rewinding of buffer pointers and re-initializing the Channel without application intervention.
4. The core automatically issues a ping token.
5. When the Device returns an ACK, the core continues with the transfer.

Note: The application must use the Do Ping bit to set the ping bit (HCTSIZ0[31]) for the next transfer and not rely on the NYET status. This ensures that the last response sent from the device (NYET/ACK) does not matter for a new transfer.

Optionally, the application can utilize these interrupts. If utilized by the application:

- The NAK or NYET interrupt is masked by the application.

- The core does not generate a separate interrupt when NAK or NYET is received by the Host functionality.

Application Programming Flow

1. The application programs a channel to do a bulk transfer for a particular data size in each transaction.
   a) Packet Data size can be up to 512KBytes
   b) Zero-length data must be programmed as a separate transaction.
2. Program the transfer size register with:
   a) Transfer size
   b) Packet Count
3. Program the DMA address.
4. Program the HCCHAR to enable the channel.
5. The application is not required to set the HCCHARx.DoPng bit for NAK/NYET responses. The core sends a Ping token automatically when the device responds with a NAK/NYET for OUT transfers. The core keeps sending the Ping token until an ACK response is received.
6. The Interrupt handling by the application is as depicted in the flow diagram.

Note: The NAK/NYET interrupts are still generated internally. The application can mask off these interrupts from reaching it. The application can use these interrupts optionally

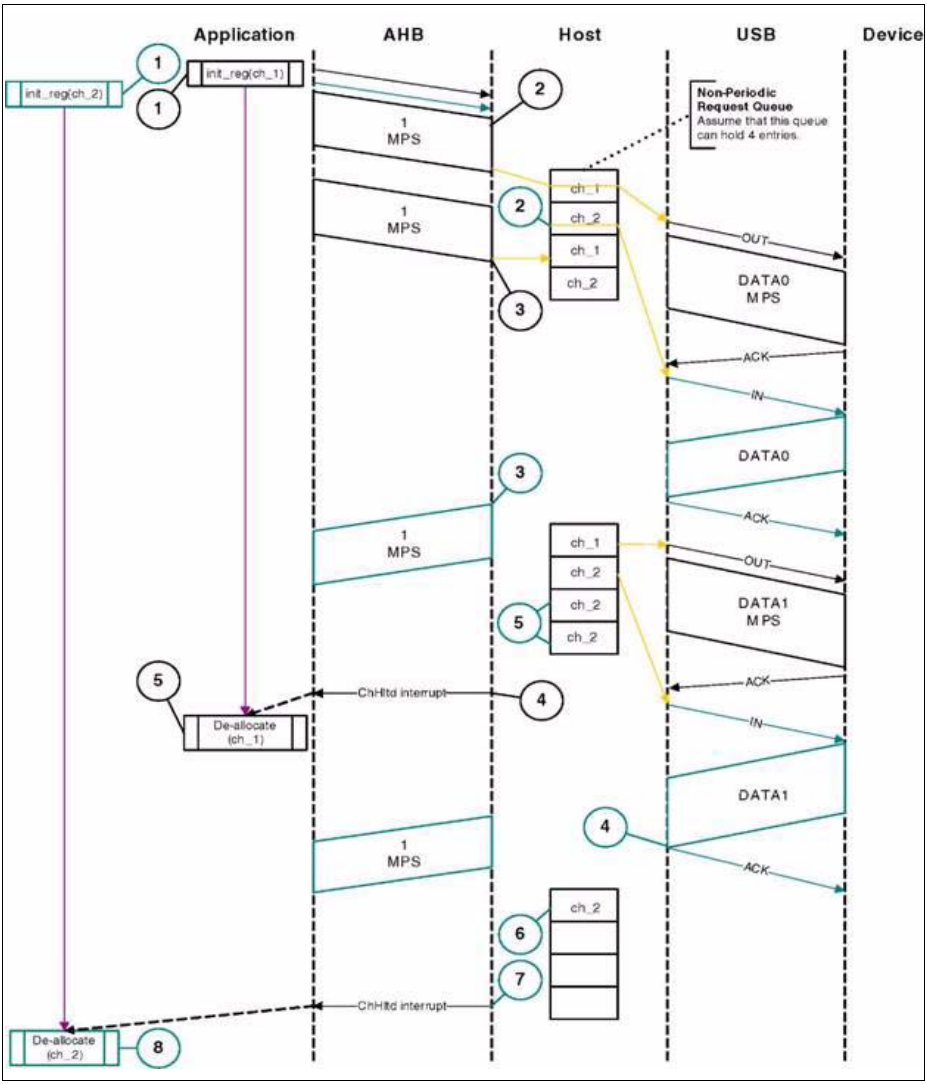


Figure 16-10 Normal Bulk/Control OUT/SETUP and Bulk/Control IN Transactions in DMA Mode

Handling Interrupts

The channel-specific interrupt service routine for bulk and control OUT/SETUP transactions in DMA mode is shown in the following code samples.

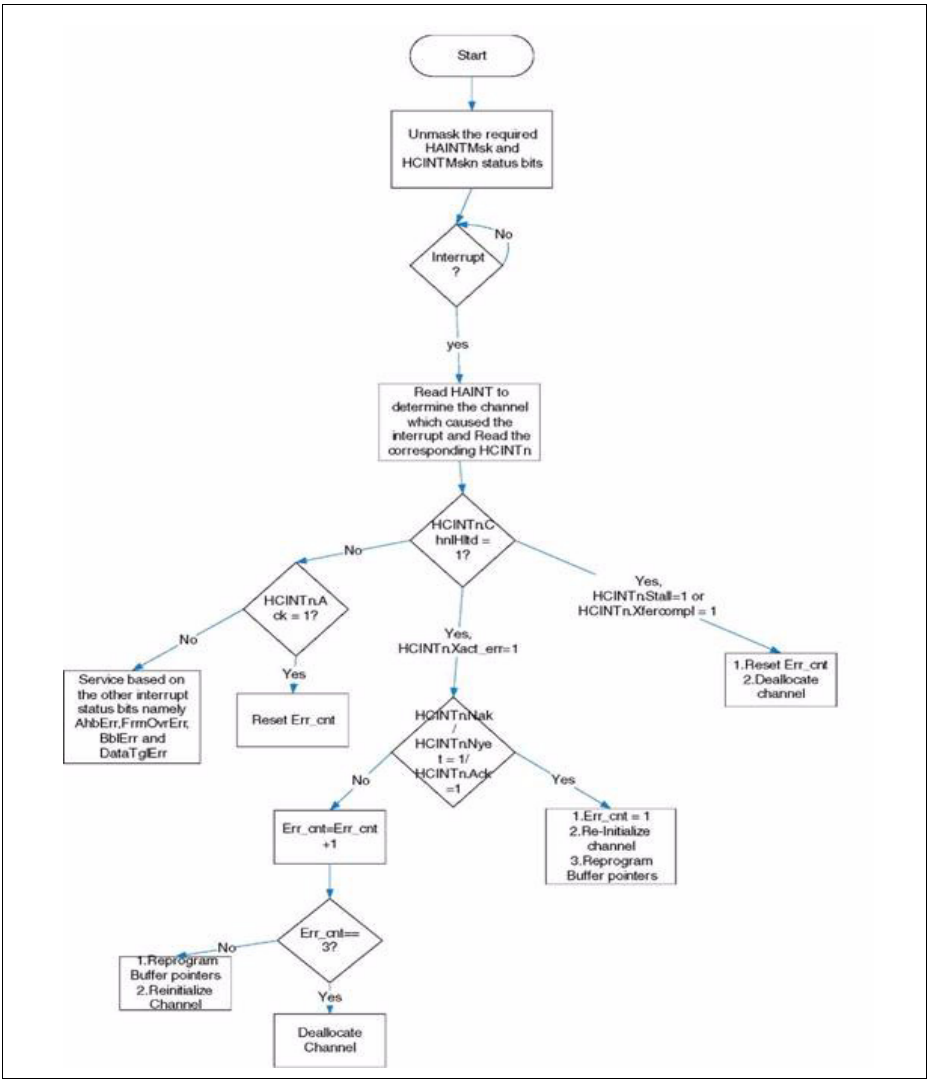


Figure 16-11 Interrupt Service Routine for Bulk/Control OUT Transaction in DMA Mode

In **Figure 16-11** that the Interrupt Service Routine is not required to handle NAK or NYET responses. The core internally sets the HCCHARx.DoPng bit once a NAK/NYET is received. The HCCHARx.DoPng is cleared only when the Ping token receives an ACK response. The application is not required to set the HCCHARx.DoPng bit for NAK/NYET scenarios. This is the difference of proposed flow with respect to current flow. Similar flow is applicable for Control flow also.

The NAK/NYET status bits in HCINTx registers are updated. The application can unmask these interrupts when it requires the core to generate an interrupt for NAK/NYET. The NAK/NYET status is updated because during Xact_err scenarios, this status provides a means for the application to determine whether the Xact_err occurred three times consecutively or there were NAK/NYET responses in between two Xact_err. This provides a mechanism for the application to reset the error counter accordingly. The application must read the NAK / NYET /ACK along with the xact_err. If NAK / NYET/ ACK is not set, the Xact_err count must be incremented otherwise application must initialize the Xact_err count to 1.

Bulk/Control OUT/SETUP

```
Unmask (ChHltd)
if (ChHltd)
     {
     if (XferCompl or STALL)
          {
          Reset Error Count (Error_count=1)
          Mask ACK
          De-allocate Channel
          }
     else if (XactErr)
          {
          if (Nak/Nyet/Ack)
               {
               Error_count = 1
               Re-initialize Channel
               Rewind Buffer Pointers }
               }
          else
          {
               Error_count = Error_count + 1
               if (Error_count == 3)
               {
                    De allocate channel
               }
               else
```

```
                {
                     Re-initialize Channel
                     Rewind Buffer Pointers
                }
           }
      }
}
else if (ACK)
{
      Reset Error Count (Error_count=1)
      Mask ACK
}
```

As soon as the channel is enabled, the core attempts to fetch and write data packets, in multiples of the maximum packet size, to the transmit FIFO when space is available in the transmit FIFO and the Request queue. The core stops fetching as soon as the last packet is fetched.

While continuing the transfer to a high-speed device, the application must set the DoPing bit before enabling the channel if the previous transaction ended with XacrErr response. In this case, the core starts with the ping protocol, then automatically switches to Data Transfer mode.

16.4.7.8 Bulk and Control IN Transactions in DMA Mode

A typical bulk or control IN operation in DMA mode is shown in **Figure 16-10**. See channel 2 (ch_2).

The assumptions are:

- The application is attempting to receive two maximum-packet-size packets (transfer size = 1,024 bytes).
- The receive FIFO can hold at least one maximum-packet-size packet and two status DWORDs per packet (72 bytes for FS).
- The Non-periodic Request Queue depth = 4.

Normal Bulk and Control IN Operations

The sequence of operations in **Figure 16-10** (channel 2) is as follows:

1. Initialize and enable channel 2 as explained in **“Channel Initialization” on Page 16-13**.
2. The USB host writes an IN request to the Request queue as soon as channel 2 receives the grant from the arbiter. (Arbitration is performed in a round-robin fashion, with fairness.).

3. The USB host starts writing the received data to the system memory as soon as the last byte is received with no errors.
4. When the last packet is received, the USB host sets an internal flag to remove any extra IN requests from the Request queue.
5. The USB host flushes the extra requests.
6. The final request to disable channel 2 is written to the Request queue. At this point, channel 2 is internally masked for further arbitration.
7. The USB host generates the ChHltd interrupt as soon as the disable request comes to the top of the queue.
8. In response to the ChHltd interrupt, de-allocate the channel for other transfers.

Handling Interrupts

The channel-specific interrupt service routine for bulk and control IN transactions in DMA mode is shown in the following flow:

Interrupt Service Routines for Bulk/Control Bulk/Control IN Transactions in DMA Mode

Bulk/Control IN

```
Unmask (ChHltd)
      if (ChHltd) {
            if (XferCompl or STALL or BblErr) {
                  Reset Error Count Mask ACK De-allocate Channel }
            else if (XactErr) {
                  if (Error_count == 2) {
                        De-allocate Channel
                  }
                  else {
                        Unmask ACK
                        Unmask NAK
                        Unmask DataTglErr
                        Increment Error
                        Count Re-initialize Channel
                  }
            }
      }
else if (ACK or NAK or DataTglErr) {
      Reset Error Count
      Mask ACK
      Mask NAK Mask DataTglErr
```

```
}
```

The application must clear and never modify the DoPing bit after enabling the channel and until the ChHltd interrupt is received. The core uses the DoPing excessive IN requests after receiving the last or short packet.

16.4.7.9 Interrupt OUT Transactions in Slave Mode

A typical interrupt OUT operation in Slave mode is shown in **Figure 16-12**. See channel 1 (ch_1). The assumptions are:

- The application is attempting to send one packet in every frame (up to 1 maximum packet size), starting with the odd frame (transfer size = 1,024 bytes).
- The Periodic Transmit FIFO can hold one packet (1 KB for FS).
- Periodic Request Queue depth = 4.

Normal Interrupt OUT Operation

The sequence of operations in **Figure 16-12** (channel 1) is as follows:

1. Initialize and enable channel 1 as explained in **"Channel Initialization" on Page 16-13**. The application must set the HCCHAR1.OddFrm bit.
2. Write the first packet for channel 1. For a high-bandwidth interrupt transfer, the application must write the subsequent packets up to MC (maximum number of packets to be transmitted in the next frame times before switching to another channel).
3. Along with the last DWORD write of each packet, the USB host writes an entry to the Periodic Request Queue.
4. The USB host attempts to send an OUT token in the next (odd) frame.
5. The USB host generates an XferCompl interrupt as soon as the last packet is transmitted successfully.
6. In response to the XferCompl interrupt, reinitialize the channel for the next transfer.

Handling Interrupts

The channel-specific interrupt service routine for Interrupt OUT transactions in Slave mode is shown in the following flow:

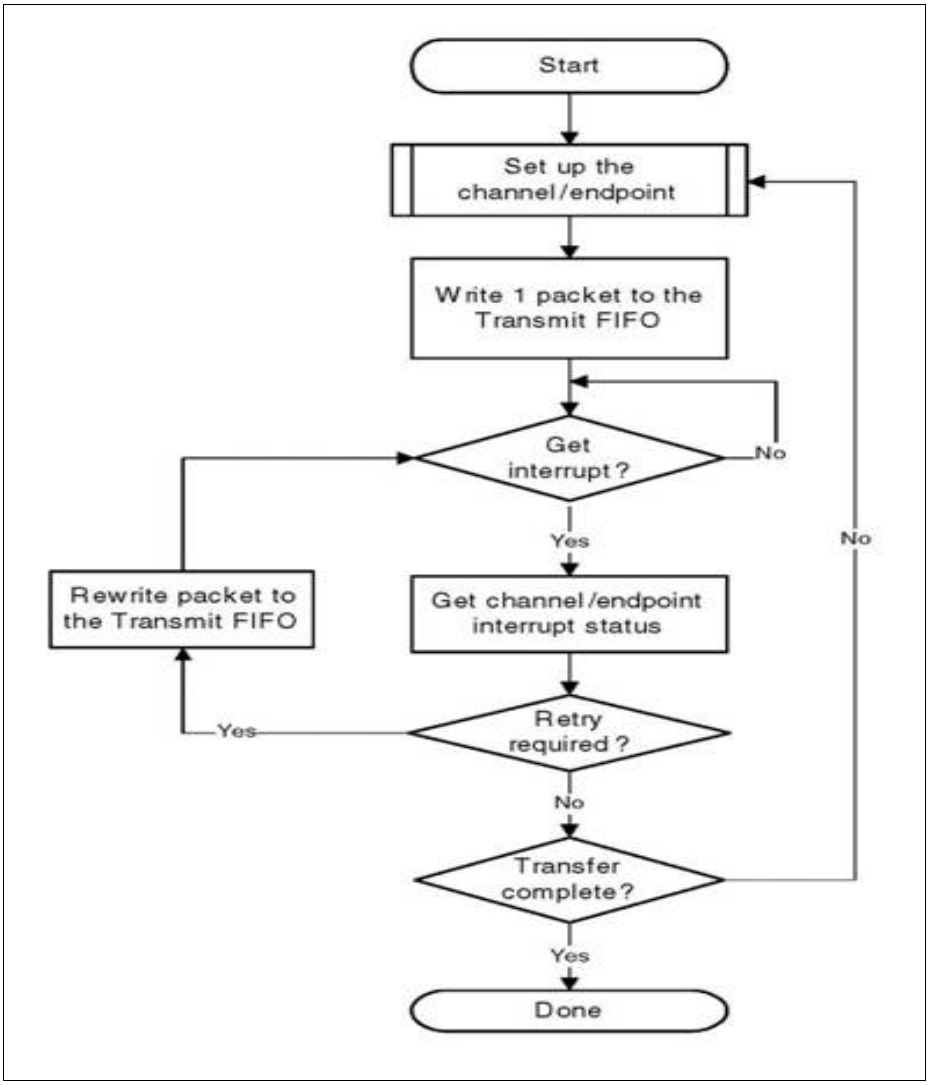


Figure 16-12 Normal Interrupt OUT/IN Transactions in Slave Mode

Interrupt Service Routine for Interrupt OUT Transactions in Slave Mode

Interrupt OUT

```
Unmask (NAK/XactErr/STALL/XferCompl/FrmOvrun)
if (XferCompl)
           {
           Reset Error Count
           Mask ACK
           De-allocate Channel
           }
else       if (STALL or FrmOvrun)
           {
           Mask ACK
           Unmask ChHltd
           Disable Channel
           if ( STALL)
                {
           Transfer Done = 1
                }
           }
     else if (NAK or XactErr)
           {
           Rewind Buffer Pointers
                Reset Error Count
                Mask ACK
                Unmask ChHltd
                Disable Channel
                }
           else if (ChHltd)
                {
                Mask ChHltd
                if (Transfer Done or (Error_count == 3))
                     {
                     De-allocate Channel
                     }
           else
                {
                Re-initialize  Channel  (in  next  b_interval  -  1
uF/F)
                }
           }
     else if (ACK)
           {
           Reset Error Count
```

```
Mask ACK
}
```

The application is expected to write the data packets into the transmit FIFO when the space is available in the transmit FIFO and the Request queue up to the count specified in the MC field before switching to another channel. The application uses the GINTSTS.NPTxFEmp interrupt to find the transmit FIFO space.

16.4.7.10 Interrupt IN Transactions in Slave Mode

A typical interrupt-IN operation in Slave mode is shown in **Figure 16-12**. See channel 2 (ch_2). The assumptions are:

- The application is attempting to receive one packet (up to 1 maximum packet size) in every frame, starting with odd. (transfer size = 1,024 bytes).
- The receive FIFO can hold at least one maximum-packet-size packet and two status DWORDs per packet (1,031 bytes for FS).
- Periodic Request Queue depth = 4.

Normal Interrupt IN Operation

The sequence of operations in **Figure 16-12** (channel 2) is as follows:

1. Initialize channel 2 as explained in **"Channel Initialization" on Page 16-13**. The application must set the HCCHAR2.OddFrm bit.
2. Set the HCCHAR2.ChEna bit to write an IN request to the Periodic Request Queue. For a high- bandwidth interrupt transfer, the application must write the HCCHAR2 register MC (maximum number of expected packets in the next frame) times before switching to another channel.
3. The USB host writes an IN request to the Periodic Request Queue for each HCCHAR2 register write with a ChEna bit set.
4. The USB host attempts to send an IN token in the next (odd) frame.
5. As soon as the IN packet is received and written to the receive FIFO, the USB host generates an RxFLvl interrupt.
6. In response to the RxFLvl interrupt, read the received packet status to determine the number of bytes received, then read the receive FIFO accordingly. The application must mask the RxFLvl interrupt before reading the receive FIFO, and unmask after reading the entire packet.
7. The core generates the RxFLvl interrupt for the transfer completion status entry in the receive FIFO. The application must read and ignore the receive packet status when the receive packet status is not an IN data packet (GRXSTSR.PktSts! = 0010_B).
8. The core generates an XferCompl interrupt as soon as the receive packet status is read.
9. In response to the XferCompl interrupt, read the HCTSIZ2.PktCnt field. If HCTSIZ2.PktCnt!= 0, disable the channel (as explained in **"Halting a Channel" on Page 16-14**) before re-initializing the channel for the next transfer, if any). If

HCTSIZ2.PktCnt == 0, reinitialize the channel for the next transfer. This time, the application must reset the HCCHAR2.OddFrm bit.

Handling Interrupts

The channel-specific interrupt service routine for an interrupt IN transaction in Slave mode is as follows.

Interrupt IN

```
Unmask (NAK/XactErr/XferCompl/BblErr/STALL/FrmOvrun/DataTglErr)
if (XferCompl)
          {
          Reset Error Count
          Mask ACK
          if (HCTSIZx.PktCnt == 0)
               {
               De-allocate Channel
               }
          else
               {
               Transfer Done = 1
               Unmask ChHltd
               Disable Channel
               }
          }
else if (STALL or FrmOvrun or NAK or DataTglErr or BblErr)
          {
          Mask ACK
          Unmask ChHltd
          Disable Channel
          if (STALL or BblErr)
               {
               Reset Error Count
               Transfer Done = 1
               }
          else if (!FrmOvrun)
               {
               Reset Error Count
               }
          }
     else if (XactErr)
          {
          Increment Error Count
```

```
            Unmask ACK
            Unmask ChHltd
            Disable Channel
            }
      else if (ChHltd)
            {
            Mask ChHltd
            if (Transfer Done or (Error_count == 3))
                  {
                  De-allocate Channel
                  }
            else Re-initialize Channel (in next b_interval - 1
uF/F)
            }
      }
else if (ACK)
      {
      Reset Error Count
      Mask ACK
      }
```

The application is expected to write the requests for the same channel when the Request queue space is available up to the count specified in the MC field before switching to another channel (if any).

16.4.7.11 Interrupt OUT Transactions in DMA Mode

A typical interrupt OUT operation in DMA mode is shown in **Figure 16-13**. See channel 1 (ch_1). The assumptions are:

- The application is attempting to transmit one packet in every frame (up to 1 maximum packet size of 1,024 bytes).
- The Periodic Transmit FIFO can hold one packet (1 KB for FS).
- Periodic Request Queue depth = 4.

Normal Interrupt OUT Operation

1. Initialize and enable channel 1 as explained in **“Channel Initialization” on Page 16-13**.
2. The USB host starts fetching the first packet as soon the channel is enabled and writes the OUT request along with the last DWORD fetch. In high-bandwidth transfers, the USB host continues fetching the next packet (up to the value specified in the MC field) before switching to the next channel.
3. The USB host attempts to send the OUT token in the beginning of the next odd frame.

4. After successfully transmitting the packet, the USB host generates a ChHltd interrupt.
5. In response to the ChHltd interrupt, reinitialize the channel for the next transfer.

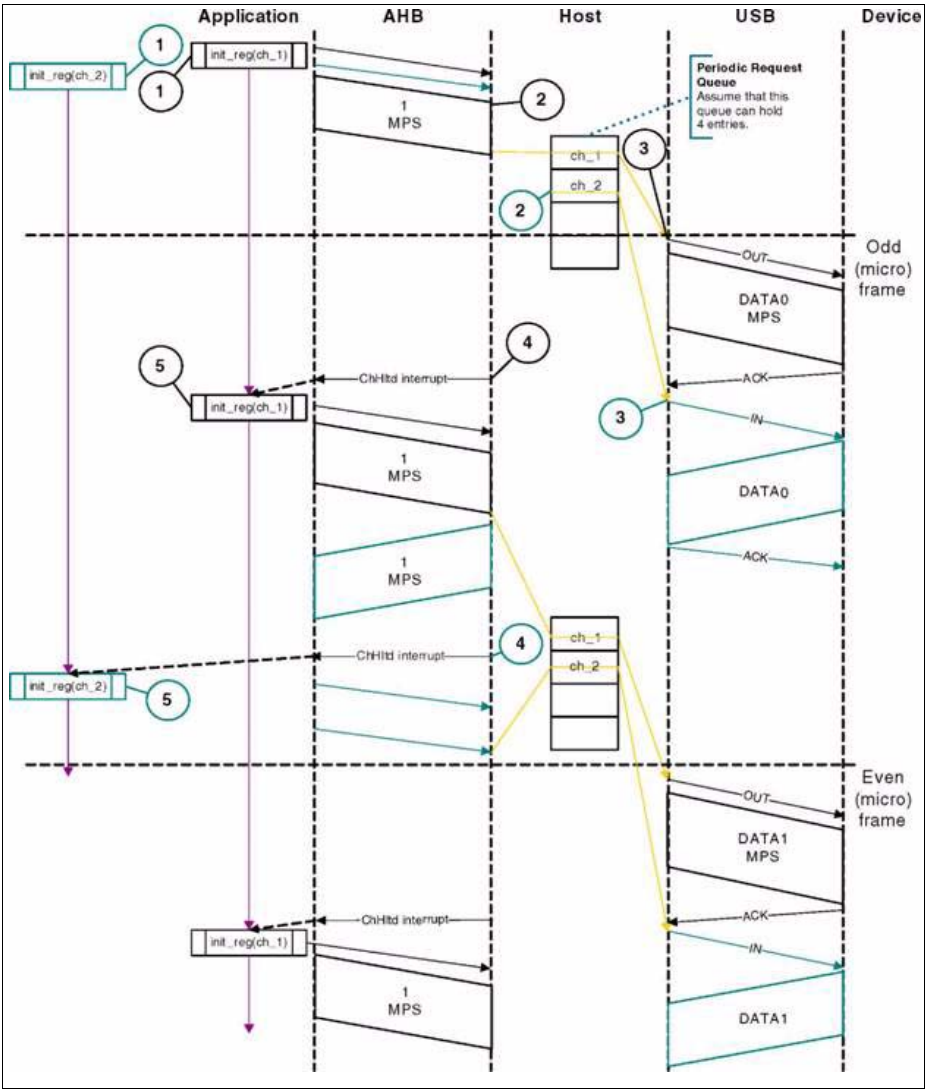


Figure 16-13 Normal Interrupt OUT/IN Transactions in DMA Mode

Handling Interrupts

The following code sample shows the channel-specific ISR for an interrupt OUT transaction in DMA mode.

Interrupt Service Routine for Interrupt OUT Transactions in DMA Mode

Interrupt OUT

```
Unmask (ChHltd)
     if (ChHltd)
          {
          if (XferCompl)
               {
               Reset Error Count
               Mask ACK
               if (Transfer Done)
                    {
                    De-allocate Channel
                    }
               else
                    {
                    Re-initialize Channel (in next b_interval -
1 uF/F)
                    }
               }
          else if (STALL)
               {
               Transfer Done = 1
               Reset Error Count
               Mask ACK
               De-allocate Channel
               }
          else if (NAK or FrmOvrun)
               {
               Mask ACK
               Rewind Buffer Pointers
               Re-initialize Channel (in next b_interval - 1
uF/F)
               if (NAK)
                    {
                    Reset Error Count
```

```
                    }
               }
          else if (XactErr)
               {
               if (Error_count == 2)
               {
               De-allocate Channel
               }
          else
               {
               Increment Error Count
               Rewind Buffer Pointers
               Unmask ACK
               Re-initialize Channel (in next b_interval - 1
uF/F)
               }
          }
     }
else if (ACK)
     {
     Reset Error Count
     Mask ACK
     }
```

As soon as the channel is enabled, the core attempts to fetch and write data packets, in maximum packet size multiples, to the transmit FIFO when the space is available in the transmit FIFO and the Request queue. The core stops fetching as soon as the last packet is fetched (the number of packets is determined by the MC field of the HCCHARx register).

16.4.7.12 Interrupt IN Transactions in DMA Mode

A typical interrupt IN operation in DMA mode is shown in **Figure 16-13**. See channel 2 (ch_2). The assumptions are:

- The application is attempting to receive one packet in every frame (up to 1 maximum packet size of 1,024 bytes).
- The receive FIFO can hold at least one maximum-packet-size packet and two status DWORDs per packet (1,032 bytes for FS).
- Periodic Request Queue depth = 4.

Normal Interrupt IN Operation

The sequence of operations in **Figure 16-13** on **Page 16-39** (channel 2) is as follows:

1. Initialize and enable channel 2 as explained in **“Channel Initialization” on Page 16-13**.
2. The USB host writes an IN request to the Request queue as soon as the channel 2 gets the grant from the arbiter (round-robin with fairness). In high-bandwidth transfers, the USB host writes consecutive writes up to MC times.
3. The USB host attempts to send an IN token at the beginning of the next (odd) frame.
4. As soon the packet is received and written to the receive FIFO, the USB host generates a ChHltd interrupt.
5. In response to the ChHltd interrupt, reinitialize the channel for the next transfer.

Handling Interrupts

The channel-specific interrupt service routine for Interrupt IN transactions in DMA mode is as follows.

Interrupt Service Routine for Interrupt IN Transactions in DMA Mode

```
Unmask (ChHltd)
if (ChHltd)
          {
          if (XferCompl)
               {
               Reset Error Count
               Mask ACK
               if (Transfer Done)
                    {
                    De-allocate Channel
                    }
               else
                    {
                    Re-initialize Channel (in next b_interval -
1 uF/F)
                    }
               }
     else if (STALL or BblErr)
          {
          Reset Error Count
          Mask ACK
          De-allocate Channel
          }
     else if (NAK or DataTglErr or FrmOvrun)
          {
          Mask ACK
          Re-initialize Channel (in next b_interval - 1 uF/F)
```

```
          if (DataTglErr or NAK)
               {
               Reset Error Count
               }
          }
     else if (XactErr)
          {
          if (Error_count == 2)
               {
               De-allocate Channel
               }
          else
               {
               Increment Error Count
               Unmask ACK
               Re-initialize Channel (in next b_interval - 1
uF/F)
               }
          }
     }
else if (ACK)
     {
     Reset Error Count
     Mask ACK
```

As soon as the channel is enabled, the core attempts to write the requests into the Request queue when the space is available up to the count specified in the MC field.

16.4.7.13 Isochronous OUT Transactions in Slave Mode

A typical isochronous OUT operation in Slave mode is shown in **Figure 16-14**. See channel 1 (ch_1). The assumptions are:

- The application is attempting to send one packet every frame (up to 1 maximum packet size), starting with an odd frame. (transfer size = 1,024 bytes).
- The Periodic Transmit FIFO can hold one packet (1 KB for FS).
- Periodic Request Queue depth = 4.

Normal Isochronous OUT Operation

The sequence of operations in **Figure 16-14** (channel 1) is as follows:

1. Initialize and enable channel 1 as explained in **"Channel Initialization" on Page 16-13**. The application must set the HCCHAR1.OddFrm bit.
2. Write the first packet for channel 1. For a high-bandwidth isochronous transfer, the application must write the subsequent packets up to MC (maximum number of

packets to be transmitted in the next frame) times before switching to another channel.

3. Along with the last DWORD write of each packet, the USB host writes an entry to the Periodic Request Queue.
4. The USB host attempts to send the OUT token in the next frame (odd).
5. The USB host generates the XferCompl interrupt as soon as the last packet is transmitted successfully.
6. In response to the XferCompl interrupt, reinitialize the channel for the next transfer.

Handling Interrupts

The channel-specific interrupt service routine for isochronous OUT transactions in Slave mode is shown in the following flow:

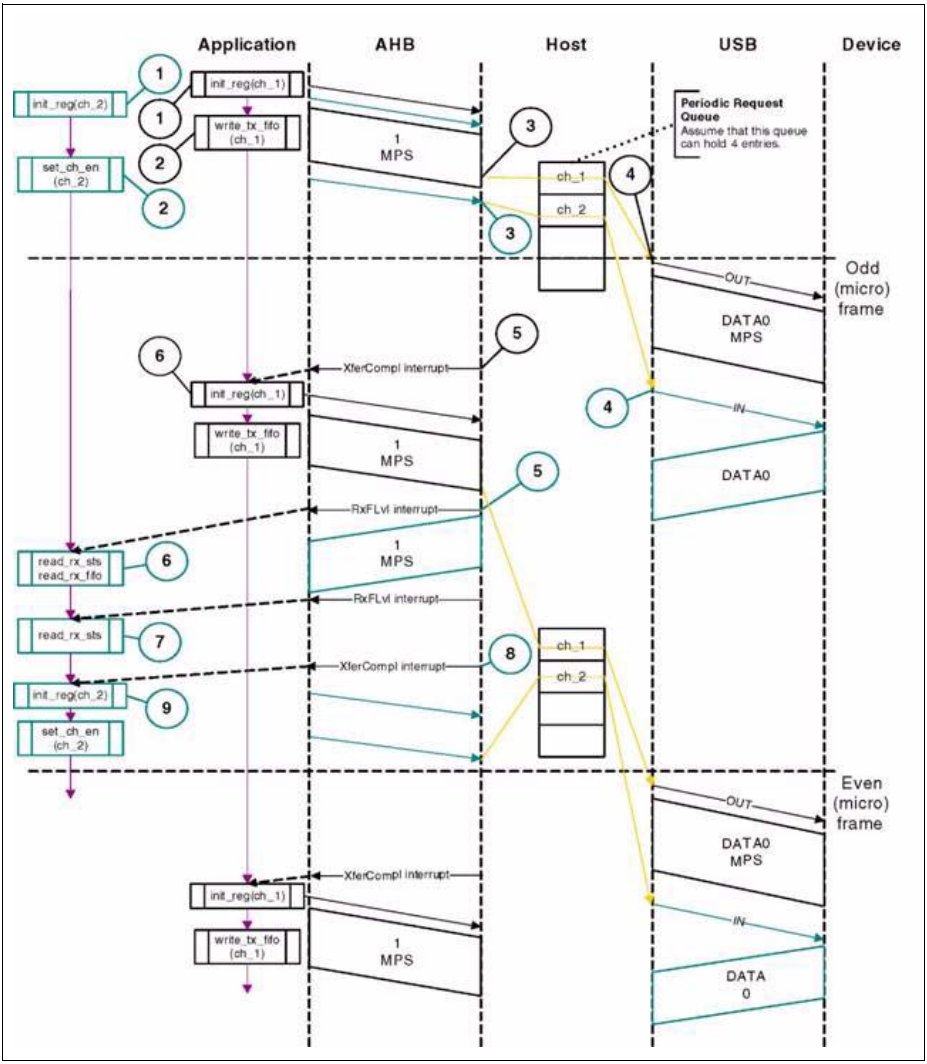


Figure 16-14 Normal Isochronous OUT/IN Transactions in Slave Mode

Interrupt Service Routine for Isochronous OUT Transactions in Slave Mode

Isochronous OUT

```
Unmask (FrmOvrun/XferCompl)
if (XferCompl)
          {
          De-allocate Channel
          }
else if (FrmOvrun)
          {
          Unmask ChHltd
          Disable Channel
          }
     else if (ChHltd)
          {
          Mask ChHltd
          De-allocate Channel
          }
```

16.4.7.14 Isochronous IN Transactions in Slave Mode

A typical isochronous IN operation in Slave mode is shown in **Figure 16-14**. See channel 2 (ch_2). The assumptions are:

- The application is attempting to receive one packet (up to 1 maximum packet size) in every frame starting with the next odd frame. (transfer size = 1,024 bytes).
- The receive FIFO can hold at least one maximum-packet-size packet and two status DWORDs per packet (1,031 bytes for FS).
- Periodic Request Queue depth = 4.

Normal Isochronous IN Operation

The sequence of operations in **Figure 16-14** (channel 2) is as follows:

1. Initialize channel 2 as explained in **"Channel Initialization" on Page 16-13**. The application must set the HCCHAR2.OddFrm bit.
2. Set the HCCHAR2.ChEna bit to write an IN request to the Periodic Request Queue. For a high- bandwidth isochronous transfer, the application must write the HCCHAR2 register MC (maximum number of expected packets in the next frame) times before switching to another channel.
3. The USB host writes an IN request to the Periodic Request Queue for each HCCHAR2 register write with the ChEna bit set.
4. The USB host attempts to send an IN token in the next odd frame.

5. As soon as the IN packet is received and written to the receive FIFO, the USB host generates an RxFLvl interrupt.
6. In response to the RxFLvl interrupt, read the received packet status to determine the number of bytes received, then read the receive FIFO accordingly. The application must mask the RxFLvl interrupt before reading the receive FIFO, and unmask it after reading the entire packet.
7. The core generates an RxFLvl interrupt for the transfer completion status entry in the receive FIFO. This time, the application must read and ignore the receive packet status when the receive packet status is not an IN data packet (GRXSTSR.PktSts!= 0010_B).
8. The core generates an XferCompl interrupt as soon as the receive packet status is read.
9. In response to the XferCompl interrupt, read the HCTSIZ2.PktCnt field. If HCTSIZ2.PktCnt!= 0, disable the channel (as explained in **“Halting a Channel” on Page 16-14**) before re-initializing the channel for the next transfer, if any. If HCTSIZ2.PktCnt == 0, reinitialize the channel for the next transfer. This time, the application must reset the HCCHAR2.OddFrm bit.

Handling Interrupts

The channel-specific interrupt service routine for an isochronous IN transaction in Slave mode is as follows.

Isochronous IN

```
Unmask (XactErr/XferCompl/FrmOvrun/BblErr)
if ( XferCompl or FrmOvrun)
      {
      if (XferCompl and (HCTSIZx.PktCnt == 0))
            {
            Reset Error Count
            De-allocate Channel
            }
      else
            {
            Unmask ChHltd
            Disable Channel
            }
      }
else if (XactErr or BblErr)
      {
      Increment Error Count
      Unmask ChHltd
      Disable Channel
```

```
      }
else if (ChHltd)
      {
      Mask ChHltd
      if (Transfer Done or (Error_count == 3))
            {
            De-allocate Channel
            }
      else
            {
            Re-initialize Channel
            }
      }
```

16.4.7.15 Isochronous OUT Transactions in DMA Mode

A typical isochronous OUT operation in DMA mode is shown in **Figure 16-15**. See channel 1 (ch_1). The assumptions are:

- The application is attempting to transmit one packet every frame (up to 1 maximum packet size of 1,024 bytes).
- The Periodic Transmit FIFO can hold one packet (1 KB for FS).
- Periodic Request Queue depth = 4.

Normal Isochronous OUT Operation

1. Initialize and enable channel 1 as explained in **"Channel Initialization" on Page 16-13**.
2. The USB host starts fetching the first packet as soon as the channel is enabled, and writes the OUT request along with the last DWORD fetch. In high-bandwidth transfers, the USB host continues fetching the next packet (up to the value specified in the MC field) before switching to the next channel.
3. The USB host attempts to send an OUT token in the beginning of the next (odd) frame.
4. After successfully transmitting the packet, the USB host generates a ChHltd interrupt.
5. In response to the ChHltd interrupt, reinitialize the channel for the next transfer.

Handling Interrupts

The channel-specific interrupt service routine for Isochronous OUT transactions in DMA mode is shown in the following flow:

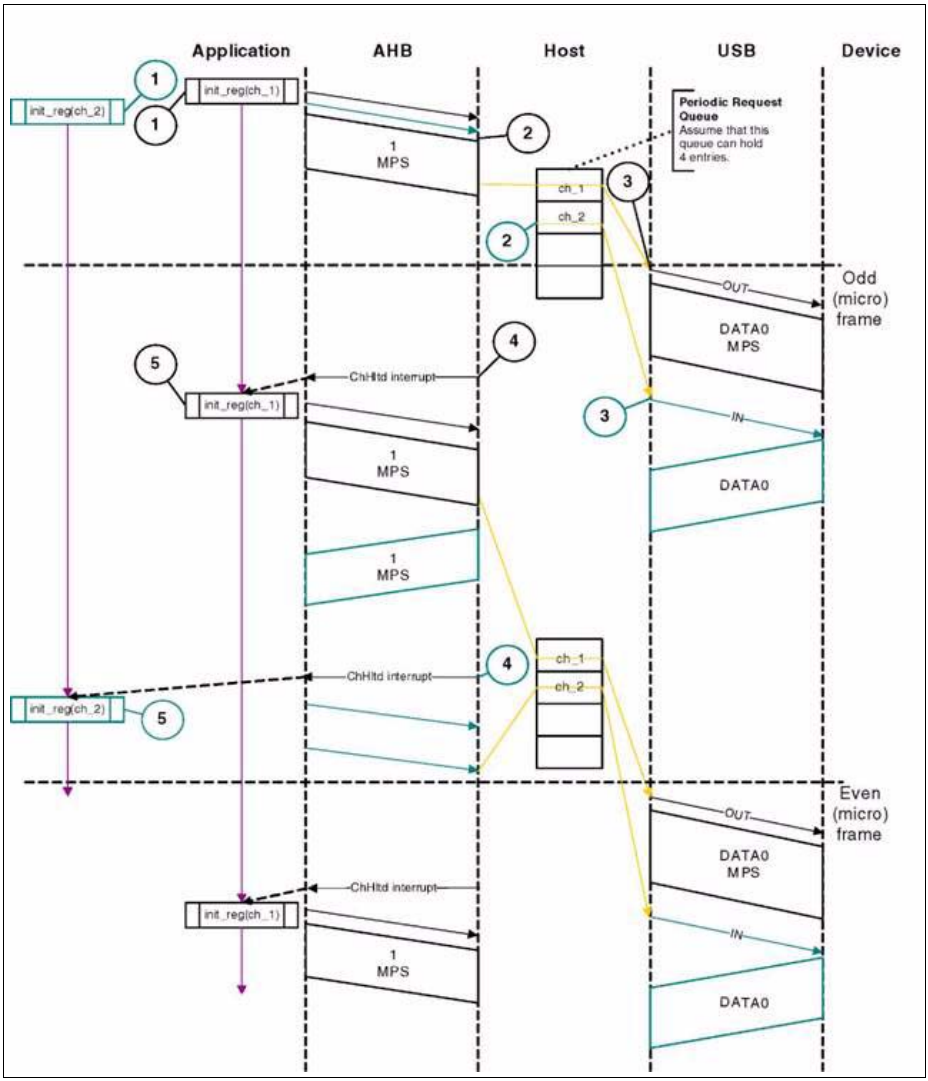


Figure 16-15 Normal Isochronous OUT/IN Transactions in DMA Mode

Interrupt Service Routine for Isochronous OUT Transactions in DMA Mode

Isochronous OUT

```
Unmask (ChHltd)
if (ChHltd)
      {
      if (XferCompl or FrmOvrun)
            {
            De-allocate Channel
            }
      }
```

16.4.7.16 Isochronous IN Transactions in Host DMA Mode

A typical isochronous IN operation in DMA mode is shown in **Figure 16-15**. See channel 2 (ch_2). The assumptions are:

- The application is attempting to receive one packet in every frame (up to 1 maximum packet size of 1,024 bytes).
- The receive FIFO can hold at least one maximum-packet-size packet and two status DWORDS per packet (1,032 bytes for FS).
- Periodic Request Queue depth = 4.

Normal Isochronous IN Operation

The sequence of operations in **Figure 16-15** (channel 2) is as follows:

1. Initialize and enable channel 2 as explained in **"Channel Initialization" on Page 16-13**.
2. The USB host writes an IN request to the Request queue as soon as the channel 2 gets the grant from the arbiter (round-robin with fairness). In high-bandwidth transfers, the USB host performs consecutive writes up to MC times.
3. The USB host attempts to send an IN token at the beginning of the next (odd) frame.
4. As soon the packet is received and written to the receive FIFO, the USB host generates a ChHltd interrupt.
5. In response to the ChHltd interrupt, reinitialize the channel for the next transfer.

Handling Interrupts

The channel-specific interrupt service routine for an isochronous IN transaction in DMA mode is as follows.

Isochronous IN

```
Unmask (ChHltd)
if (ChHltd)
```

```
      {
      if ( XferCompl or FrmOvrun)
            {
            if (XferCompl and (HCTSIZx.PktCnt == 0))
                  {
                  Reset Error Count
                  De-allocate Channel
                  }
            else
                  {
                  De-allocate Channel
                  }
            }
      else if (XactErr or BblErr)
            {
            if (Error_count == 2)
                  {
                  De-allocate Channel
                  }
            else
            {
            Increment Error Count
            Re-enable Channel (in next b_interval - 1 uF/F)
            }
      }
}
```

16.5 Host Scatter-Gather DMA Mode

16.5.1 Overview

Note the following points when the host scatter-gather DMA mode is used:

- USB core supports non-DWORD aligned address access in Scatter/Gather DMA in Host mode only
- NAK/NYET scenario is handled by USB core in Scatter/Gather DMA mode without the application's intervention.
- CONCAT mode is not supported for any of the flows, that is, a single packet cannot span more than one descriptor.

16.5.2 SPRAM Requirements

For each channel, the current descriptor pointer and descriptor status are cached to avoid additional requests to system memory. These are stored in the SPRAM. In addition, the HCDMAx registers are also implemented in the SPRAM.

16.5.3 Descriptor Memory Structures

In Scatter/Gather DMA mode, the core implements a true scatter-gather memory distribution in which data buffers are scattered over the system memory. However, the descriptors themselves are continuous. Each channel memory structure is implemented as a contiguous list of descriptors; each descriptor points to a data buffer of predefined size. In addition to the buffer pointer (1 DWORD), the descriptor also has a status quadlet (1 DWORD). When the list is implemented as a ring buffer, the list processor switches to the first element of the list when it encounters last bit. All channels (control, bulk, interrupt, and isochronous) implement these structures in memory.

Note: The descriptors are stored in continuous locations. For example, descriptor 1 is stored in 32'h0000_0000, descriptor 2 is stored in 32'h0000_0008, descriptor 3 in 32'h0000_0010 and so on. The descriptors are always DWORD aligned.

The descriptor memory structures are displayed in **Figure 16-16**.

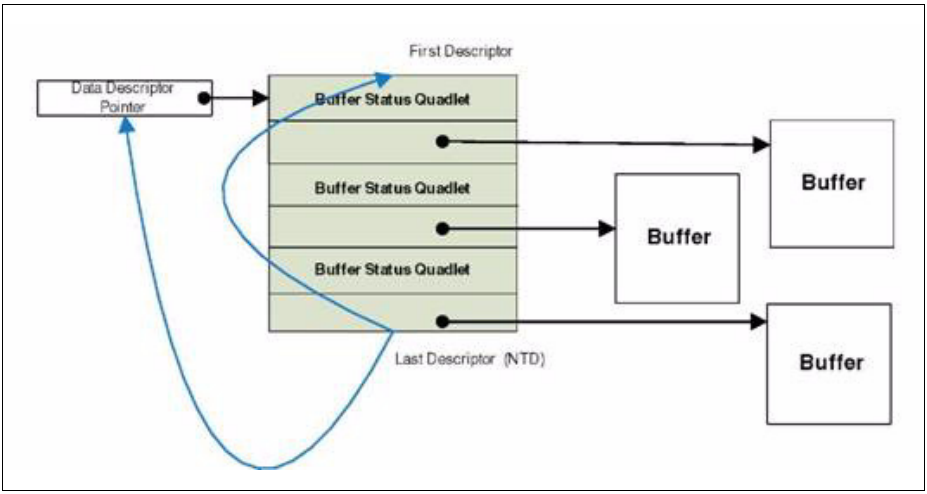


Figure 16-16 Descriptor Memory Structures

All channels must implement the following memory structure:

- Each channel has one memory structure
- Each data buffer must have a descriptor associated with it to provide the status of the buffer. The buffer itself contains only raw data.
- Each buffer descriptor is two quadlets in length. When the descriptor is ready, the DMA fetches and processes its data buffer. The buffers to which the descriptor points hold packet data for non- isochronous channels and packet data corresponding to the frame data for isochronous channels.
- The handshake between the application and core is accomplished by the Active Bit field in the status quadlet of the descriptor as described below:
- A=1 indicates that the descriptor is ready.
- A=0 indicates that the descriptor is not ready.

The IN and OUT data memory structures are shown in **Figure 16-17**. The figure shows the definition of status quadlet bits for non-ISO and ISO channels.

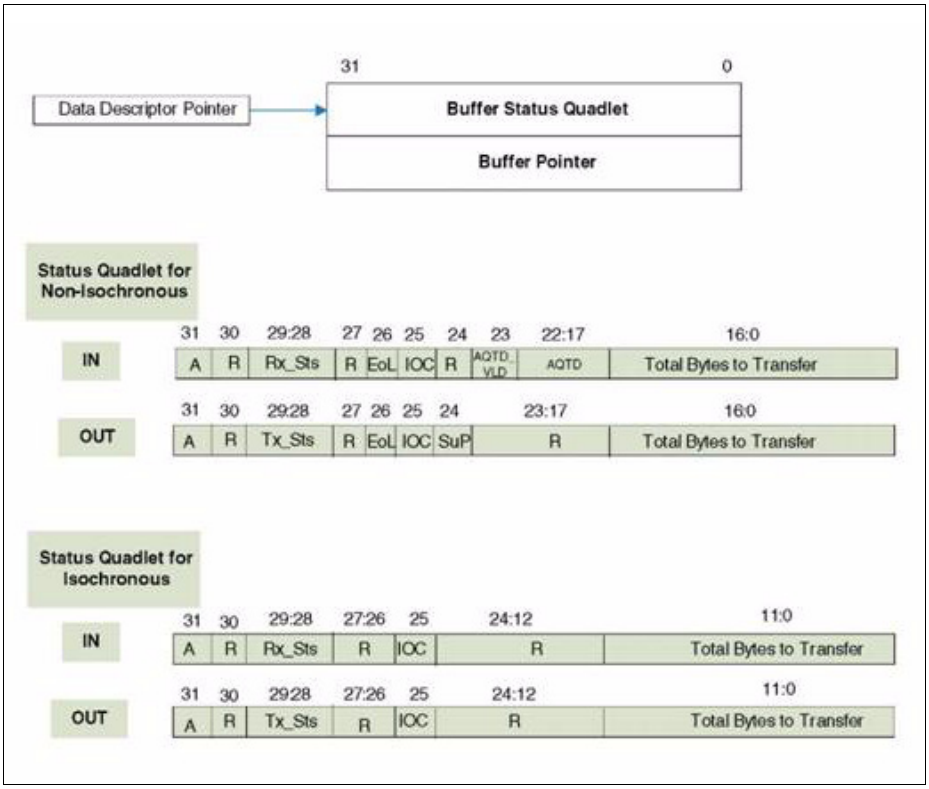


Figure 16-17 Memory Structure

In addition, a Frame list in memory for Isochronous and Interrupt channels contains information on the channels that need to be scheduled in a frame. For periodic channels, USB core reads the list corresponding to the frame number and schedules the channel that has Ch_sch=1 in the appropriate frame. **Figure 16-18** shows the frame list for periodic channels.

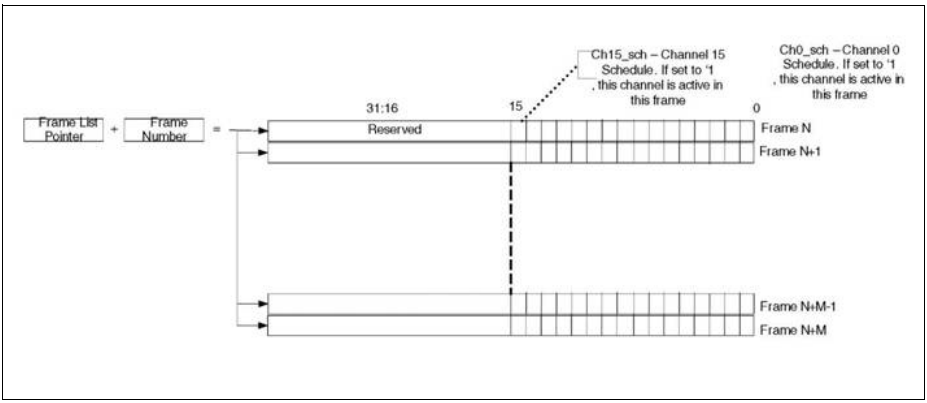


Figure 16-18 Frame List for Periodic Channels

16.5.4 IN Memory Structure

All channels that support IN direction transactions (channels that receive data from the USB device) must implement a memory structure with the following characteristics:

- Each data buffer must have a descriptor associated with it to provide the status of the buffer. The buffer itself contains only raw data.
- Each buffer descriptor is two quadlets in length. **Table 16-2** displays the IN Data Memory Structure fields.

Table 16-2 IN Data Memory Structure Values

Bit	Bit ID	Description
A[31]	Active Bit	This 1 -bit value indicates whether the descriptor is ready. For non-isochronous channels, this bit indicates the following: 0_B Descriptor is not ready 1_B Descriptor is ready. USB core can start processing the descriptor. For Isochronous channels, this bit indicates the following: 0_B Isochronous channel is not scheduled for the corresponding frame/fame. 1_B Isochronous channel is not scheduled for the corresponding frame/fame. The application sets this bit when the descriptor is ready. USB core resets this bit while closing the descriptor. The application needs to set this bit as a last step after the entire descriptor is ready. The core resets this bit as a final step of processing the descriptor. This bit is accessed by both the core and the application.
Rx Sts [29:28]	Receive Status	This 2-bit field describes the status of the received data. The core updates this field when the descriptor is closed. PKTERR is set by the core when there was an error while receiving the packet. When updated with PKTERR, it is an indication that IN data has been received with errors. The error includes Xact_err scenarios. BUFERR is set by the core when AHB error is encountered during buffer access. The possible combinations are: • 00_B Success, No AHB or packet errors • 01_B PKTERR. • 10_B Reserved • 11_B Reserved This field has to be initialized to 00_B by the application and updated by the core subsequently.

Table 16-2 IN Data Memory Structure Values (cont'd)

Bit	Bit ID	Description	
EoL [26]	End of List	For Non Isochronous, it indicates that this is the last descriptor in the list, if set. The core does not generate a BNA interrupt for the next descriptor, if it is unavailable. For Isochronous, this field is reserved. This field is controlled by the application.	
IOC [25]	Interrupt On complete	Set by the application, this bit indicates that the core must generate a transfer complete interrupt (XferCompl) after this descriptor is finished.	
[24]	Varies	Non Isochronous Reserved	**Isochronous** **Bit: [24:23]** **Bit ID:** R: Reserved
[23]	Varies	**Non Isochronous IN** **Bit: [23]** Bit ID: AQTD_VALID Alternate Queue Transfer Descriptor Valid. When set by the application, if a Short packet is received, the core jumps to a new descriptor in the same list. The new descriptor address is obtained by replacing the CTD value of the corresponding channel with the AQTD value. When the application resets this bit, the core ignores AQTD.	
[22:17]	Varies	**Non Isochronous IN** **Bit: [23]** **Bit ID:** AQTD_VALID Alternate Queue Transfer Descriptor Valid. This is valid only if AQTD_VALID is set. This field gives the offset value in DWORDS. The core will use this offset to jump to a new descriptor address in the same list.	**Isochronous IN Bit** **Bit: [22:12]** **Bit ID:** R Reserved

Table 16-2 IN Data Memory Structure Values (cont'd)

Bit	Bit ID	Description	
[16:12]	Varies	**Non Isochronous IN** **Bit:** [16:0] **Bit ID:** Total bytes to transfer This 17-bit value can take values from 0 to 128K-1 bytes, depending on the transfer size of data received from the USB device. The application programs the expected transfer size. When the descriptor is closed, this indicates remainder of the transfer size. This field must be in multiple of MPS for the corresponding end point. The MPS for the various packet types are as follows: • Control - – LS - 8 bytes – FS - 8,16,32,64 bytes • Bulk – – FS - 8,16,32,64 bytes • Interrupt – LS - up to 8 bytes – FS – up to 64 bytes	
[11]	Varies		**Isochronous IN** **Bit: 11:0** **Bit ID:** Received
[10:0]	Varies		**Isochronous IN** **Bit: [11:0]** **Bit ID:** Total bytes to transfer This 11-bit value can take values from 0 to 4K bytes, depending on the packet size of data received from the USB host. The application programs the expected transfer size. When the descriptor is closed, it indicates remainder of the transfer size. The maximum payload size of each ISO packet as per USB specification 2.0 is as follows. • FS - up to 1023 bytes *Note: Note: A value of 0 indicates zero bytes of data, 1 indicates 1 byte of data, and so on.*

Table 16-3 displays the out buffer pointer field description.

Table 16-3 IN Buffer Pointer

Buf Addr[31:0]	Buffer Address	The Buffer pointer field in the descriptor is 32 bits wide and contains the address where the received data is to be stored in the system memory. The buffer address does not need to be aligned with DWORD.

16.5.5 OUT Memory Structure

All channels that support OUT direction transactions (channels that transmit data to the USB device) must implement the following memory structure:

- Each buffer must have a descriptor associated with it.
- The application fills the data buffer, updates its status in the descriptor, and enables the channel.
- The DMA fetches this descriptor and processes it, moving on in this manner until it reaches the end of the descriptor chain.
- The buffer to which the descriptor points to holds packet data for non-isochronous channels and frame data for isochronous channels.

Table 16-4 displays the OUT Data Memory Structure fields. Bits that are not present are reserved to be set to zero by the application for writes and ignored during reads.

Table 16-4 OUT Data Memory Structure Values

Bit	Bit ID	Description
A[31]	Active Bit	This 1 -bit value indicates whether the descriptor is ready. For non-isochronous channels, this bit indicates the following: 0_B Descriptor is not ready 1_B Descriptor is ready. USB core can start processing the descriptor. For Isochronous channels, this bit indicates the following: 0_B Isochronous channel is not scheduled for the corresponding frame. 1_B Isochronous channel is not scheduled for the corresponding frame. The application sets this bit when the descriptor is ready. USB core resets this bit while closing the descriptor. The application needs to set this bit as a last step after the entire descriptor is ready. The core resets this bit as a final step of processing the descriptor.
Tx Sts [29:28]	Transmit Status	The status of the transmitted data. This reflects if the OUT data has been transmitted correctly or with errors. BUFERR is set by core when there is a AHB error during buffer access along with asserting AHBERR interrupt (HCINTx register) for the corresponding channel. PKTERR is set by the core when there was an error while transmitting the packet. The error includes Xact_err scenarios. The possible combinations are as follows: 00_B Success, No AHB errors 01_B PKTERR 10_B Reserved 11_B Reserved This field has to be initialized to 00_B by the application and updated by the core subsequently.
EoL [26]	End of List	For Non Isochronous, it indicates that this is the last descriptor in the list, if set. The core does not generate a BNA interrupt for the next descriptor, if it is unavailable. For Isochronous, this field is reserved. This field is controlled by the application.
IOC[25]	Interrupt On complete	Set by the application, this bit indicates that the core must generate a transfer complete interrupt after this descriptor is finished.

Table 16-4 OUT Data Memory Structure Values (cont'd)

Bit	Bit ID	Description	
[24][1)]	SuP	**Non Isochronous OUT** Setup Packet When set, it indicates that the buffer data pointed by this descriptor is a setup packet of 8 bytes	**Isochronous** **Reserved: [24:12]**
[23][1)]	R	**Non Isochronous OUT** Setup Packet **Bit:** Reserved [23] **Bit ID:** Reserved	
[22:17][1)]	Varies	Non Isochronous OUT Bit Reserved: [22:17] Bit ID: Reserved	
[16:12][1)]	Varies	**Non Isochronous OUT OUT** **Bit:** [16:0] **Bit ID:** Total bytes to transfer. This 17-bit value can take values from 0 to 128K-1 bytes, indicating the number of bytes of data to be transmitted to the USB device. Note: A Value of 0 indicates zero bytes of data, 1 indicates 1 byte of data and so on	**Isochronous** **Bit [11:0]** **Bit ID:** Total bytes to transfer. This 12-bit value can take values from 0 to 4K bytes, indicating the number of bytes of data to be trvValue of 0 indicates zero bytes of data, 1 indicates 1 byte of data, and so on.
[11:0][1)]	Varies		

1) The meaning of this field varies. See description.

Table 16-5 displays the out buffer pointer field description.

Table 16-5 IN Buffer Pointer

Buf Addr[31:0]	Buffer Address	The Buffer pointer field in the descriptor is 32 bits wide and contains the address where the transmit data is to be stored in the system memory. The buffer address does not need to be aligned with DWORD.

16.5.6 Host Scatter-Gather DMA Mode Programming Model

This section describes the programming requirements for the USB core operating in descriptor DMA host mode. It describes how to initialize the channel and provides information on asynchronous transfers (bulk and control) and periodic transfers (isochronous and interrupt).

16.5.6.1 Channel Initialization

The application must initialize one or more channels before it can communicate with connected devices. To initialize and enable a channel, the application must perform the following steps.

- Program the periodic frame list array (for periodic channels).
- Program the HFLBAddr register with the base address of the periodic frame list array (for periodic channels).
- Program the HCFG register with PerSchedEn bit set.
- Program at least one transfer descriptor in the system memory.
- Program the HCDMAx with the pointer to the corresponding descriptor.
- Program the GINTMSK register to unmask the Channel Interrupts.
- Program the HAINTMSK register to unmask the selected channels' interrupts.
- Program the HCINTMSK register to unmask the ChHalt, XferCompl, and BNA.
- Program the HCTSIZx register with initial data PID and SCHED_INFO (for periodic channels). In addition, depending on the number of transfer descriptors, program the NTD field.
- Program the HCCHARx register with the device's endpoint characteristics, such as type, speed, direction, and so on (The channel can be enabled by setting the channel enable bit to 1_B only when the application is ready to transmit or receive any packet).

16.5.6.2 Asynchronous Transfers

When the application enables an asynchronous (Bulk and Control) channel by writing into the HCCHARx register, the host controller begins servicing the asynchronous channel. It reads the referenced (CTD) transfer descriptor qTDn (pointed to by the HCDMAx register). If the read qTDn is active, the host controller caches the qTDn and then schedules a transaction. If the read qTDn is inactive, the host controller disables the channel and generates a Buffer Not Available (BNA) interrupt.

If multiple asynchronous channels are enabled simultaneously, the host controller caches the referenced transfer descriptor of the entire enabled channels. The host controller schedules transactions for each enabled channel in round-robin fashion.

When the host controller completes the transfer for a channel, it updates the status quadlet of the processed qTDn in the system memory.

For a normal completion, the host controller updates the status of the qTDn with no errors. The host controller completes a transfer normally if one of the following events occurs:

- Short or zero length packet is received for an IN channel.
- The allocated buffer is fulfilled with the received data packets for an IN channel.
- The allocated buffer is fully transferred to the device for an OUT channel.

When a transfer is completed normally, the host controller attempts to process the next qTDn from the descriptor list, if the End of List (EOL) bit is not set in the completed qTDn.

where m = AQTD (if IN channel with AQTD_VLD=1 received a short packet) or m = (n + 1) mod (NTD + 1)

If EOL is set, the host controller disables the channel and generates a Channel Halt interrupt. The transfer complete interrupt is generated for the following conditions.

- IOC is set.
- Short or zero length packet is received for an IN channel.
- EOL is set.

For an abnormal completion, the host controller updates the status of the qTDn with PKT_ERR. The host controller completes a transfer abnormally if one of the following events occurs:

- STALL response is received from the device.
- Excessive transaction errors occurred.
- Babble detected.

When a transfer is completed abnormally, the host controller disables the channel and then generates a Channel Halt interrupt with the appropriate status in HCINT register.

Asynchronous Transfer Descriptor

The application must use separate qTD for different stages of control transfers. A three stage control transfer uses three qTDs. The same qTD can be reused for performing different stages of control transfer. The combination of EPType, EPDir fields of the HCCHARx register, and the SuP flag of the qTD decides the stage of the control transfer. See **Table 16-6**.

Table 16-6 Asynchronous Transfer Descriptor

HCCHARx.EP Type	HCCHARx.EP Dir	qTD.SuP	Control Stage
00_B	0	1	SETUP
00_B	0	0	Data stage OUT / Status stage OUT
00_B	1	0	Data stage IN / Status stage IN
00_B	1	1	Invalid

The host controller executes a zero-length OUT transaction if the "Num bytes to transmit" field of the qTD is initialized to zero for an OUT channel. For an IN channel, the "Num of bytes received" field of the qTD must be always initialized to an integer multiple of the maximum packet size.

The application can use one or multiple qTDs for bulk IN and OUT transfers. The number of qTDs depends on the available consecutive data buffer space and the size of the transfer. Each qTD can support up to 64KB of consecutive data buffer space.

16.5.6.3 Periodic Transfers

The periodic schedule is used to manage all isochronous and interrupt transfer streams. The base of the periodic schedule is the periodic frame list. The periodic schedule is referenced from the register space using the HFLBAddrBase and the HFNUM registers. The periodic schedule is based on an array of scheduled channels called the periodic frame list. The periodic frame list implements a sliding window of transactions over time. When the application enables the periodic schedule (PerSchedEna) in the HCFG register, the host controller attempts to read an entry from the frame list that corresponds to the next running frame number at the beginning of each frame.

The periodic frame list can be programmed to 8, 16, 32, or 64 elements. The size of the periodic frame list should be large enough to support the required b-interval of the least frequent channel. The least significant bits [15:0] in the periodic frame list elements are used to identify the scheduled periodic channels (0 through 15) for that corresponding frame. For example if channel 2 and 6 are periodic channels scheduled for a frame then the corresponding entry in the periodic frame list will be 0000_0044_H.

The host controller should program the SCHED_INFO to 1111_1111_B when operating in Full Speed for all the enabled periodic channels.

Isochronous Transactions

When the application enables an isochronous channel by writing into the HCCHARx register, the host controller begins servicing the isochronous channel based on the programmed scheduling (periodic frame list and SCHED_INFO). The application must use separate qTD for each frame. Each qTD handles a frame of transactions. The application is expected to allocate a qTD with Active bit zero even if no transaction is scheduled for a frame. The position of the active qTD determines the b-interval of the isochronous channel.

The host controller supports high-bandwidth isochronous transfer via the multi-count (MC) field of the HCCHARx register. The Multi Count represents a transaction count per frame for the endpoint. If the multi- count is zero, the operation of the host controller is undefined. Multi-count greater than one is not applicable for the FS host.

For OUT transfers, the value of the "Num bytes to transmit" field represents the total bytes to be sent during the frame. The application is expected to program the Mult count

field to be the maximum number of packets to be sent in any frame. The host controller automatically selects the number of packets and its data PID based on the programmed Xfer Size.

For IN transfers, the host controller issues Mult count transactions. The application is expected to initialize the "Num bytes received" field to (MC * MaxPktSize).

The host controller does not execute all Multi-count transactions if:

- The channel is an OUT and the "Num bytes to transmit" goes to zero before all the Multi-count transactions have executed (ran out of transmit data) or
- The channel is an IN and the endpoint delivers a short packet, or an error occurs on a transaction before all the Multi-count transaction have been executed.
- The channel is an IN and the endpoint delivers a packet with DATA0 PID before all the Multi-count transaction have been executed.

Each transfer descriptor (qTD) describes one frame of transactions. The host controller will cache one transfer descriptor in a frame prior to the scheduled frame.

When the application is adding new isochronous transactions to the schedule, it always performs a read of the HFNUM register to determine the current frame and frame the host is currently executing. Because there is no information about where in the frame the host controller is, a constant uncertainty factor of one frame for FS is assumed.

The end of frame (FS) may occur before all of the transaction opportunities are executed. When this happens, the host controller closes the corresponding descriptor and proceeds to processing the next scheduled descriptor. If the scheduled descriptor is not fetched by the host controller due to high system latency, the host controller does not execute any transaction for that scheduled frame and will skip the descriptor without any update (that is, without clearing the Active bit).

When a transfer is completed normally, the host controller generates the transfer complete interrupt only if IOC is set in the completed qTD.

When a transfer is completed abnormally (STALL response or Babble), the host controller disables the channel and then generates a Channel Halt interrupt with the appropriate status in HCINT register. The host controller updates the status of the qTD with PKT_ERR if one of the following conditions occurs:

- STALL response is received from the device
- Error packet received
- Babble detected
- Unable to complete all the transactions in the scheduled frame

An example for the FS isochronous scheduling is shown in **Figure 16-19**. In this figure, channels 2 and 15 are isochronous channels with b-interval 1ms and 4ms respectively. The host controller fetches only the qTDs that corresponds to the scheduled frame (Periodic Frame List entry). The host controller initiates the qTD fetch in the frame prior to the scheduled frame. If the qTD is active and belongs to an OUT channel, the host

controller also fetches the corresponding data in the previous frame. If this qTD is not active, the host controller ignores the qTD and does not generate any BNA interrupt.

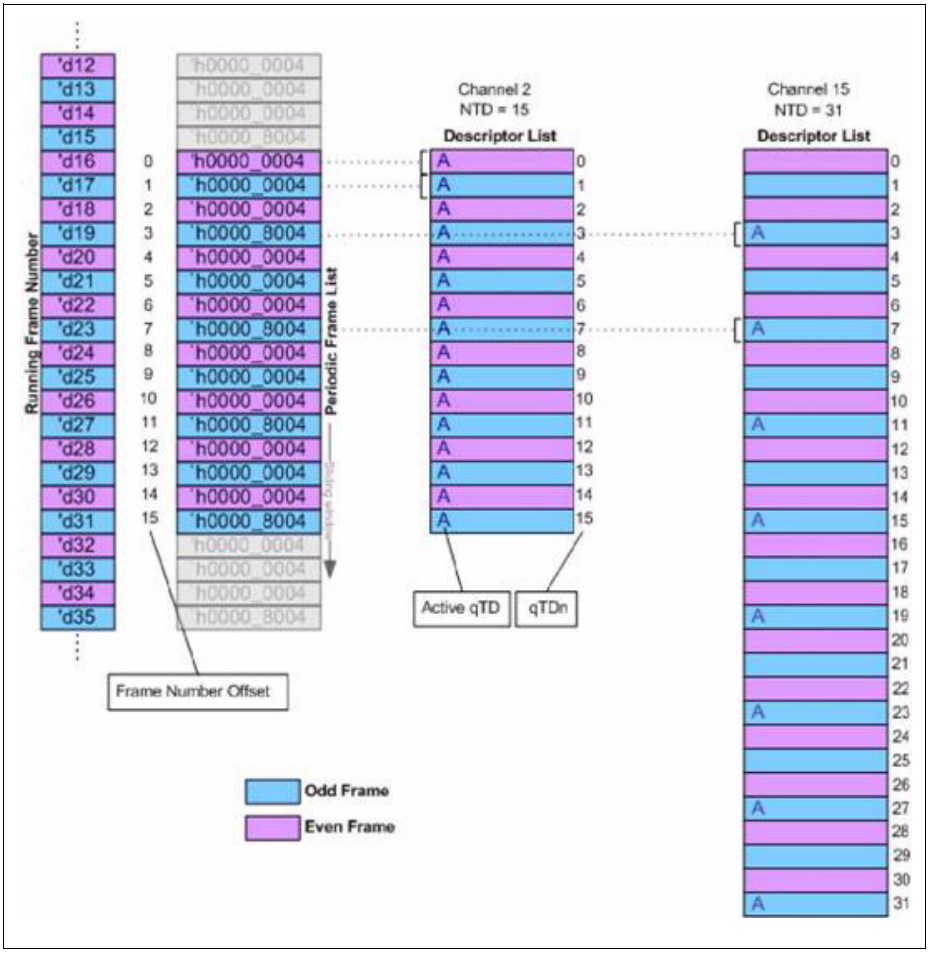


Figure 16-19 Full Speed Isochronous Transfer Scheduling

Interrupt Transactions

When the application enables an interrupt channel by writing into the HCCHARx register, the host controller begins servicing the interrupt channel based on the programmed scheduling (periodic frame list and SCHLD_INFO). It reads the referenced (CTD) transfer descriptor qTDn (pointed by the HCDMAx register) in the frame prior to the scheduled frame.

If the read qTDn is active, the host controller caches the qTDn and then schedules a transaction. If the read qTDn is inactive, the host controller disables the channel and generates a Buffer Not Available (BNA) interrupt.

When the host controller completes the transfer, it updates the status quadlet of the processed qTDn in the system memory.

For a normal completion, the host controller updates the status of the qTDn with no errors. The host controller completes a transfer normally if one of the following events occurs:

- Short or zero length packet is received for an IN channel.
- The allocated buffer is fulfilled with the received data packets for an IN channel.
- The allocated buffer is fully transferred to the device for an OUT channel.

When a transfer is completed normally, the host controller attempts to process the next qTDm from the descriptor list if the End of List (EOL) bit is not set in the completed qTDn.

Where m = (n + 1) mod (NTD + 1)

If EOL is set, the host controller disables the channel and generates Channel Halt interrupt. The transfer complete interrupt will be generated for the following conditions.

- IOC is set.
- Short or zero length packet is received for an IN channel.
- EOL is set.

For an abnormal completion, the host controller updates the status of the qTDn with PKT_ERR. The host controller completes a transfer abnormally if one of the following events occurs:

- STALL response is received from the device.
- Excessive transaction errors occurred.
- Babble detected.

When a transfer is completed abnormally, the host controller disables the channel and then generates a Channel Halt interrupt with the appropriate status in the HCINT register.

The host controller supports high-bandwidth interrupt transfer through the Multi-count (MC) field of HCCHARx register. The Multi-count represents a transaction count per frame for the endpoint. If the Multi-count is zero, the operation of the host controller is undefined. Multi-count greater than one is not applicable for FS host.

The host controller does not execute all Multi-count transactions in a frame if:

- The channel is an OUT and the "Num bytes to transmit" goes to zero before all the Multi-count transactions have executed (ran out of transmit data) or
- The channel is an IN and the endpoint delivers a short packet, or an error occurs on a transaction before all the Multi-count transaction have been executed.
- The channel is an IN and the "Num bytes received" goes to zero before all the Multi-count transaction are executed (ran out of receive buffer space).

16.6 Device Programming Model

16.6.1 Device Initialization

The application must perform the following steps to initialize the core at device on, power on, or after a mode change from Host to Device.

1. Program the following fields in DCFG register.
   a) DescDMA bit
   b) Device Speed
   c) NonZero Length Status OUT Handshake
   d) Periodic Frame Interval (If Periodic Endpoints are supported)
2. Program the GINTMSK register to unmask the following interrupts.
   a) USB Reset
   b) Enumeration Done
   c) Early Suspend
   d) USB Suspend
   e) SOF
3. Wait for the GINTSTS.USBReset interrupt, which indicates a reset has been detected on the USB and lasts for about 10 ms. On receiving this interrupt, the application must perform the steps listed in **“Initialization on USB Reset” on Page 16-69**.
4. Wait for the GINTSTS.EnumerationDone interrupt. This interrupt indicates the end of reset on the USB. On receiving this interrupt, the application must read the DSTS register to determine the enumeration speed and perform the steps listed in **“Initialization on Enumeration Completion” on Page 16-70**.

At this point, the device is ready to accept SOF packets and perform control transfers on control endpoint 0.

16.6.2 Endpoint Initialization

16.6.2.1 Initialization on USB Reset

1. Set the NAK bit for all OUT endpoints
   a) DOEPCTLx.SNAK = 1 (for all OUT endpoints)
2. Unmask the following interrupt bits:
   a) DAINTMSK.INEP0 = 1 (control 0 IN endpoint)
   b) DAINTMSK.OUTEPO = 1 (control 0 OUT endpoint)
   c) DOEPMSK.SETUP = 1
   d) DOEPMSK.XferCompl = 1

e) DIEPMSK.XferCompl = 1
f) DIEPMSK.TimeOut = 1

3. To transmit or receive data, the device must initialize more registers as specified in **"Device DMA/Slave Mode Initialization" on Page 16-72**
4. Set up the Data FIFO RAM for each of the FIFOs
   a) Program the GRXFSIZ Register, to be able to receive control OUT data and setup data. At a minimum, this must be equal to 1 max packet size of control endpoint 0 + 2 DWORDs (for the status of the control OUT data packet) + 10 DWORDs (for setup packets).
   b) Program the dedicated FIFO size register (depending on the FIFO number chosen) in Dedicated FIFO operation, to be able to transmit control IN data. At a minimum, this must be equal to 1 max packet size of control endpoint 0.
5. (This step is not required if the Scatter/Gather DMA mode is used.) Program the following fields in the endpoint-specific registers for control OUT endpoint 0 to receive a SETUP packet
   a) DOEPTSIZ0.SetUP Count = 3 (to receive up to 3 back-to-back SETUP packets)
   b) In DMA mode, DOEPDMAO register with a memory address to store any SETUP packets received

At this point, all initialization required to receive SETUP packets is done, except for enabling control OUT endpoint 0 in DMA mode.

16.6.2.2 Initialization on Enumeration Completion

1. On the Enumeration Done interrupt (GINTSTS.EnumDone, read the DSTS register to determine the enumeration speed.
2. Program the DIEPCTLO.MPS field to set the maximum packet size. This step configures control endpoint 0. The maximum packet size for a control endpoint depends on the enumeration speed.
3. In DMA mode, program the DOEPCTLO register to enable control OUT endpoint O, to receive a SETUP packet. In Scatter/Gather DMA mode, the descriptors must be set up in memory before enabling the endpoint.
   a) DOEPCTLO. EPEna = 1

At this point, the device is ready to receive SOF packets and is configured to perform control transfers on control endpoint O.

16.6.2.3 Initialization on SetAddress Command

This section describes what the application must do when it receives a SetAddress command in a SETUP packet.

1. Program the DCFG register with the device address received in the SetAddress command

2. Program the core to send out a status IN packet.

16.6.2.4 Initialization on SetConfiguration/SetInterface Command

This section describes what the application must do when it receives a SetConfiguration or SetInterface command in a SETUP packet.

1. When a SetConfiguration command is received, the application must program the endpoint registers to configure them with the characteristics of the valid endpoints in the new configuration.
2. When a SetInterface command is received, the application must program the endpoint registers of the endpoints affected by this command.
3. Some endpoints that were active in the prior configuration or alternate setting are not valid in the new configuration or alternate setting. These invalid endpoints must be deactivated.
4. For details on a particular endpoint's activation or deactivation, see **“Endpoint Activation” on Page 16-71** and **“Endpoint Deactivation” on Page 16-72**.
5. Unmask the interrupt for each active endpoint and mask the interrupts for all inactive endpoints in the DAINTMSK register.
6. Set up the Data FIFO RAM for each FIFO. See **“Data FIFO RAM Allocation” on Page 16-201** for more detail.
7. After all required endpoints are configured, the application must program the core to send a status IN packet.

At this point, the device core is configured to receive and transmit any type of data packet.

16.6.2.5 Endpoint Activation

This section describes the steps required to activate a device endpoint or to configure an existing device endpoint to a new type.

1. Program the characteristics of the required endpoint into the following fields of the DIEPCTLx register (for IN or bidirectional endpoints) or the DOEPCTLx register (for OUT or bidirectional endpoints).
   a) Maximum Packet Size
   b) USB Active Endpoint = 1
   c) Endpoint Start Data Toggle (for interrupt and bulk endpoints)
   d) Endpoint Type
   e) TxFIFO Number
2. Once the endpoint is activated, the core starts decoding the tokens addressed to that endpoint and sends out a valid handshake for each valid token received for the endpoint.

16.6.2.6 Endpoint Deactivation

This section describes the steps required to deactivate an existing endpoint.

1. In the endpoint to be deactivated, clear the USB Active Endpoint bit in the DIEPCTLx register (for IN or bidirectional endpoints) or the DOEPCTLx register (for OUT or bidirectional endpoints).
2. Once the endpoint is deactivated, the core ignores tokens addressed to that endpoint, resulting in a timeout on the USB.

16.6.2.7 Device DMA/Slave Mode Initialization

The application must meed the following conditions to set up the device core to handle traffic.

- In Slave mode, GINTMSK.NPTxFEmpMsk, and GINTMSK.RxFLvlMsk must be unset.
- In DMA mode, the aforementioned interrupts must be masked.

16.6.3 Device Programming Operations (Non-Descriptor DMA Mode)

Table 16-7 provides links to the programming sequence for different USB transaction types.

Table 16-7 Device Programming Operations

Device Mode	IN	SETUP	OUT
Control			
Slave	"Generic Non-Periodic (Bulk and Control) IN Data Transfers" on Page 16-111	"OUT Data Transfers in Device Slave and Buffer DMA Modes" on Page 16-76	"Generic Non-Isochronous OUT Data Transfers in DMA and Slave Modes" on Page 16-88
DMA	"Generic Non-Periodic (Bulk and Control) IN Data Transfers" on Page 16-111	"OUT Data Transfers in Device Slave and Buffer DMA Modes" on Page 16-76	"Generic Non-Isochronous OUT Data Transfers in DMA and Slave Modes" on Page 16-88
Bulk			
Slave	"Generic Non-Periodic (Bulk and Control) IN Data Transfers" on Page 16-111	-	"Generic Non-Isochronous OUT Data Transfers in DMA and Slave Modes" on Page 16-88
DMA	"Generic Non-Periodic (Bulk and Control) IN Data Transfers" on Page 16-111	-	"Generic Non-Isochronous OUT Data Transfers in DMA and Slave Modes" on Page 16-88

Table 16-7 Device Programming Operations (cont'd)

Device Mode	IN	SETUP	OUT
Interrupt			
Slave	**"Generic Periodic IN (Interrupt and Isochronous) Data Transfers" on Page 16-118** **"Generic Periodic IN Data Transfers Using the Periodic Transfer Interrupt Feature" on Page 16-120**	-	**"Generic Non-Isochronous OUT Data Transfers in DMA and Slave Modes" on Page 16-88** **"Generic Interrupt OUT Data Transfers Using Periodic Transfer Interrupt Feature" on Page 16-93**
DMA	**"Generic Periodic IN (Interrupt and Isochronous) Data Transfers" on Page 16-118** **"Generic Periodic IN Data Transfers Using the Periodic Transfer Interrupt Feature" on Page 16-120**	-	**"Generic Non-Isochronous OUT Data Transfers in DMA and Slave Modes" on Page 16-88** **"Generic Interrupt OUT Data Transfers Using Periodic Transfer Interrupt Feature" on Page 16-93**

Table 16-7 Device Programming Operations (cont'd)

Device Mode	IN	SETUP	OUT
Isochronous			
Slave	**"Generic Periodic IN (Interrupt and Isochronous) Data Transfers" on Page 16-118**	-	**"Generic Isochronous OUT Data Transfer in DMA and Slave Modes" on Page 16-92** **"Incomplete Isochronous OUT Data Transfers in DMA and Slave Modes" on Page 16-100**
DMA	**"Generic Periodic IN (Interrupt and Isochronous) Data Transfers" on Page 16-118** **"Generic Periodic IN Data Transfers Using the Periodic Transfer Interrupt Feature" on Page 16-120**	-	**"Generic Isochronous OUT Data Transfer in DMA and Slave Modes" on Page 16-92** **"Incomplete Isochronous OUT Data Transfers in DMA and Slave Modes" on Page 16-100**

16.6.3.1 OUT Data Transfers in Device Slave and Buffer DMA Modes

This section describes the internal data flow and application-level operations during data OUT transfers and setup transactions.

Control Setup Transactions

This section describes how the core handles SETUP packets and the application's sequence for handling setup transactions. To initialize the core after power-on reset, the application must follow the sequence in **“Core Initialization” on Page 16-7**. Before it can communicate with the host, it must initialize an endpoint as described in **“Endpoint Initialization” on Page 16-69**. See **“Packet Read from FIFO in Slave Mode” on Page 16-84**.

Application Requirements

1. To receive a SETUP packet, the DOEPTSIZx.SUPCnt field in a control OUT endpoint must be programmed to a non-zero value. When the application programs the SUPCnt field to a non-zero value, the core receives SETUP packets and writes them to the receive FIFO, irrespective of the DOEPCTLx.NAK status and DOEPCTLx.EPEna bit setting. The SUPCnt field is decremented every time the control endpoint receives a SETUP packet. If the SUPCnt field is not programmed to a proper value before receiving a SETUP packet, the core still receives the SETUP packet and decrements the SUPCnt field, but the application possibly is not be able to determine the correct number of SETUP packets received in the Setup stage of a control transfer.
   a) DOEPTSIZx.SUPCnt = 3
2. In DMA mode, the OUT endpoint must also be enabled, to transfer the received SETUP packet data from the internal receive FIFO to the external memory.
   a) DOEPCTLx.EPEna = 1_B
3. The application must always allocate some extra space in the Receive Data FIFO, to be able to receive up to three SETUP packets on a control endpoint.
   a) The space to be Reserved is (4 * n) + 6 DWORDs, where n is the number of control endpoints supported by the device. Three DWORDs are required for the first SETUP packet, 1 DWORD is required for the Setup Stage Done DWORD, and 6 DWORDs are required to store two extra SETUP packets among all control endpoints.
   b) 3 DWORDs per SETUP packet are required to store 8 bytes of SETUP data and 4 bytes of SETUP status (Setup Packet Pattern). The core reserves this space in the receive data
   c) FIFO to write SETUP data only, and never uses this space for data packets.
4. In Slave mode, the application must read the 2 DWORDs of the SETUP packet from the receive FIFO. In DMA mode, the core writes the 2 DWORDs of SETUP data to the memory.

5. The application must read and discard the Setup Stage Done DWORD from the receive FIFO.

Internal Data Flow

1. When a SETUP packet is received, the core writes the received data to the receive FIFO, without checking for available space in the receive FIFO and irrespective of the endpoint's NAK and Stall bit settings.
   a) The core internally sets the IN NAK and OUT NAK bits for the control IN/OUT endpoints on which the SETUP packet was received.
2. For every SETUP packet received on the USB, 3 DWORDs of data is written to the receive FIFO, and the SUPCnt field is decremented by 1.
   a) The first DWORD contains control information used internally by the core
   b) The second DWORD contains the first 4 bytes of the SETUP command
   c) The third DWORD contains the last 4 bytes of the SETUP command
3. When the Setup stage changes to a Data IN/OUT stage, the core writes an entry (Setup Stage Done DWORD) to the receive FIFO, indicating the completion of the Setup stage.
4. On the AHB side, SETUP packets are emptied either by the DMA or the application. In DMA mode, the SETUP packets (2 DWORDs) are written to the memory location programmed in the DOEPDMAx register, only if the endpoint is enabled. If the endpoint is not enabled, the data remains in the receive FIFO until the enable bit is set.
5. When either the DMA or the application pops the Setup Stage Done DWORD from the receive FIFO, the core interrupts the application with a DOEPINTx.SETUP interrupt, indicating it can process the received SETUP packet.
   a) The core clears the endpoint enable bit for control OUT endpoints.

Application Programming Sequence

1. Program the DOEPTSIZx register.
   a) DOEPTSIZx.SUPCnt = 3
2. In DMA mode, program the DOEPDMAx register and DOEPCTLx register with the endpoint characteristics and set the Endpoint Enable bit (DOEPCTLx. EPEna).
   a) Endpoint Enable = 1
3. In Slave mode, wait for the GINTSTS.RxFLvl interrupt and empty the data packets from the receive FIFO, as explained in **“Packet Read from FIFO in Slave Mode” on Page 16-84**. This step can be repeated many times.
4. Assertion of the DOEPINTx.SETUP interrupt marks a successful completion of the SETUP Data Transfer.
   a) On this interrupt, the application must read the DOEPTSIZx register to determine the number of SETUP packets received and process the last received SETUP packet.

b) In DMA mode, the application must also determine if the interrupt bit DOEPINTx.Back2BackSETup is set. This bit is set if the core has received more than three back- to-back SETUP packets. If this is the case, the application must ignore the DOEPTSIZx.SUPCnt value and use the DOEPDMAx directly to read out the last SETUP packet received. DOEPDMAx-8 provides the pointer to the last valid SETUP data.

Note: If the application has not enabled EP0 before the host sends the SETUP packet, the core ACKs the SETUP packet and stores it in the FIFO, but does not write to the memory until EP0 is enabled. When the application enables the EP0 (first enable) and clears the NAK bit at the same time the Host sends DATA OUT, the DATA OUT is stored in the RxFIFO. The OTG core then writes the setup data to the memory and disables the endpoint. Though the application expects a Transfer Complete interrupt for the Data OUT phase, this does not occur, because the SETUP packet, rather than the DATA OUT packet, enables EP0 the first time. Thus, the DATA OUT packet is still in the RxFIFO until the application re-enables EP0. The application must enable EP0 one more time for the core to process the DATA OUT packet.

Figure 16-20 charts this flow.

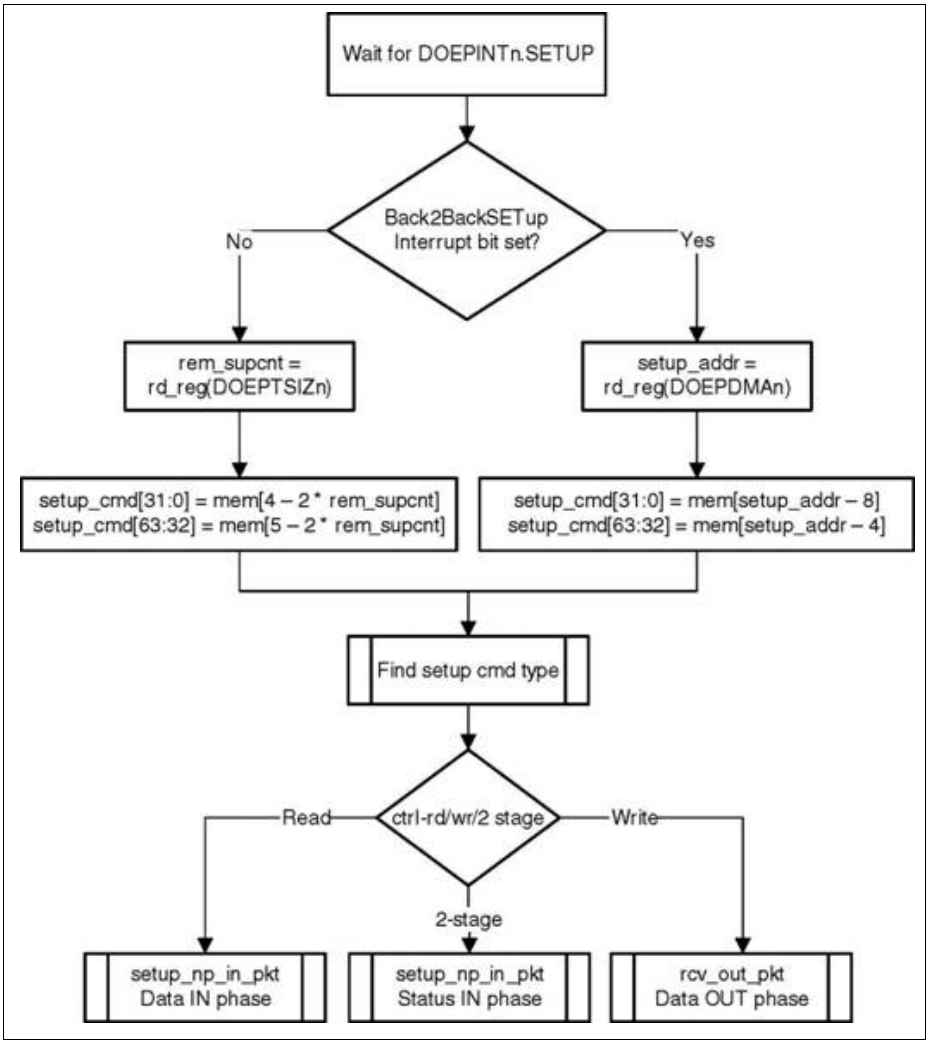


Figure 16-20 Processing a SETUP Packet

Handling More Than Three Back-to-Back SETUP Packets

Per the USB 2.0 specification, normally, during a SETUP packet error, a host does not send more than three back-to-back SETUP packets to the same endpoint. However, the USB 2.0 specification does not limit the number of back-to-back SETUP packets a host

can send to the same endpoint. When this condition occurs, the USB core generates an interrupt (DOEPINTx.Back2BackSETup). In DMA mode, the core also rewinds the DMA address for that endpoint (DOEPDMAx) and overwrites the first SETUP packet in system memory with the fourth, second with the fifth, and so on. If the Back2BackSETup interrupt is asserted, the application must read the OUT endpoint DMA register (DOEPDMAx) to determine the final SETUP data in system memory.

In DMA mode, the application can mask the Back2BackSETup interrupt, but after receiving the DOEPINT.SETUP interrupt, the application can read the DOEPINT.Back2BackSETup interrupt bit. In Slave mode, the application can use the GINTSTS.RxFLvl interrupt to read out the SETUP packets from the FIFO whenever the core receives the SETUP packet.

16.6.3.2 Control Transfers in Device Mode

This section describes the various types of control transfers.

Control Write Transfers (SETUP, Data OUT, Status IN)

This section describes control write transfers.

Application Programming Sequence

1. Assertion of the DOEPINTx.SETUP Packet interrupt indicates that a valid SETUP packet has been transferred to the application. See **“OUT Data Transfers in Device Slave and Buffer DMA Modes” on Page 16-76** for more details. At the end of the Setup stage, the application must reprogram the DOEPTSIZx.SUPCnt field to 3 to receive the next SETUP packet.
2. If the last SETUP packet received before the assertion of the SETUP interrupt indicates a data OUT phase, program the core to perform a control OUT transfer as explained in **“Generic Non-Isochronous OUT Data Transfers in DMA and Slave Modes” on Page 16-88**.
   In DMA mode, the application must reprogram the DOEPDMAx register to receive a control OUT data packet to a different memory location.
3. In a single OUT data transfer on control endpoint 0, the application can receive up to 64 bytes. If the application is expecting more than 64 bytes in the Data OUT stage, the application must re-enable the endpoint to receive another 64 bytes, and must continue to do so until it has received all the data in the Data stage.
4. Assertion of the DOEPINTx.Transfer Compl interrupt on the last data OUT transfer indicates the completion of the data OUT phase of the control transfer.
5. On completion of the data OUT phase, the application must do the following.
   a) To transfer a new SETUP packet in DMA mode, the application must re-enable the control OUT endpoint as explained in section in section "**“OUT Data Transfers in Device Slave and Buffer DMA Modes” on Page 16-76**.
      - DOEPCTLx.EPEna = 1_B
   b) To execute the received Setup command, the application must program the required registers in the core. This step is optional, based on the type of Setup command received.
6. For the status IN phase, the application must program the core as described in **“Generic Non-Periodic (Bulk and Control) IN Data Transfers” on Page 16-111** to perform a data IN transfer.
7. Assertion of the DIEPINTx.Transfer Compl interrupt indicates completion of the status IN phase of the control transfer.

Control Read Transfers (SETUP, Data IN, Status OUT)

This section describes control write transfers.

Application Programming Sequence

1. Assertion of the DOEPINTx.SETUP Packet interrupt indicates that a valid SETUP packet has been transferred to the application. See "**“OUT Data Transfers in Device Slave and Buffer DMA Modes” on Page 16-76** for more details. At the end of the Setup stage, the application must reprogram the DOEPTSIZx.SUPCnt field to 3 to receive the next SETUP packet.
2. If the last SETUP packet received before the assertion of the SETUP interrupt indicates a data IN phase, program the core to perform a control IN transfer as explained in **“Generic Non-Periodic (Bulk and Control) IN Data Transfers” on Page 16-111**.
3. On a single IN data transfer on control endpoint 0, the application can transmit up to 64 bytes. To transmit more than 64 bytes in the Data IN stage, the application must re-enable the endpoint to transmit another 64 bytes, and must continue to do so, until it has transmitted all the data in the Data stage.
4. The DIEPINTx.Transfer Compl interrupt on the last IN data transfer marks the completion of the control transfer's Data stage.
5. To perform a data OUT transfer in the status OUT phase, the application must program the core as described in "**“OUT Data Transfers in Device Slave and Buffer DMA Modes” on Page 16-76**.
   a) The application must program the DCFG.NZStsOUTHShk handshake field to a proper setting before transmitting an data OUT transfer for the Status stage.
   b) In DMA mode, the application must reprogram the DOEPDMAx register to receive the control OUT data packet to a different memory location.
6. Assertion of the DOEPINTx.Transfer Compl interrupt indicates completion of the status OUT phase of the control transfer. This marks the successful completion of the control read transfer.
   a) To transfer a new SETUP packet in DMA mode, the application must re-enable the control OUT endpoint as explained in "**“OUT Data Transfers in Device Slave and Buffer DMA Modes” on Page 16-76**.
      - DOEPCTLx.EPEna = 1_B

Two-Stage Control Transfers (SETUP/Status IN)

This section describes two-stage control transfers.

Application Programming Sequence

1. Assertion of the DOEPINTx.SetUp interrupt indicates that a valid SETUP packet has been transferred to the application. See "**“OUT Data Transfers in Device Slave and Buffer DMA Modes” on Page 16-76** for more detail. To receive the next SETUP packet, the application must reprogram the DOEPTSIZx.SUPCnt field to 3 at the end of the Setup stage.

2. Decode the last SETUP packet received before the assertion of the SETUP interrupt. If the packet indicates a two-stage control command, the application must do the following.
   a) To transfer a new SETUP packet in DMA mode, the application must re-enable the control OUT endpoint. See "**“OUT Data Transfers in Device Slave and Buffer DMA Modes” on Page 16-76** for details.
      - DOEPCTLx.EPEna = 1_B
   b) Depending on the type of Setup command received, the application can be required to program registers in the core to execute the received Setup command.
3. For the status IN phase, the application must program the core described in **“Generic Non-Periodic (Bulk and Control) IN Data Transfers” on Page 16-111** to perform a data IN transfer.
4. Assertion of the DIEPINTx.Transfer Compl interrupt indicates the completion of the status IN phase of the control transfer.

Example: Two-Stage Control Transfer

These notes refer to **Figure 16-21**.

1. SETUP packet #1 is received on the USB and is written to the receive FIFO, and the core responds with an ACK handshake. This handshake is lost and the host detects a timeout.
2. The SETUP packet in the receive FIFO results in a GINTSTS.RxFLvl interrupt to the application, causing the application to empty the receive FIFO.
3. SETUP packet #2 on the USB is written to the receive FIFO, and the core responds with an ACK handshake.
4. The SETUP packet in the receive FIFO sends the application the GINTSTS.RxFLvl interrupt and the application empties the receive FIFO.
5. After the second SETUP packet, the host sends a control IN token for the status phase. The core issues a NAK response to this token, and writes a Setup Stage Done entry to the receive FIFO. This entry results in a GINTSTS.RxFLvl interrupt to the application, which empties the receive FIFO. After reading out the Setup Stage Done DWORD, the core asserts the DOEPINTx.SetUp packet interrupt to the application.
6. On this interrupt, the application processes SETUP Packet #2, decodes it to be a two-stage control command, and clears the control IN NAK bit.
   a) DIEPCTLx.CNAK = 1
7. When the application clears the IN NAK bit, the core interrupts the application with a DIEPINTx.INTknTXFEmp. On this interrupt, the application enables the control IN endpoint with a DIEPTSIZx.XferSize of 0 and a DIEPTSIZx.PktCnt of 1. This results in a zero-length data packet for the status IN token on the USB.
8. At the end of the status IN phase, the core interrupts the application with a DIEPINTx.XferCompl interrupt.

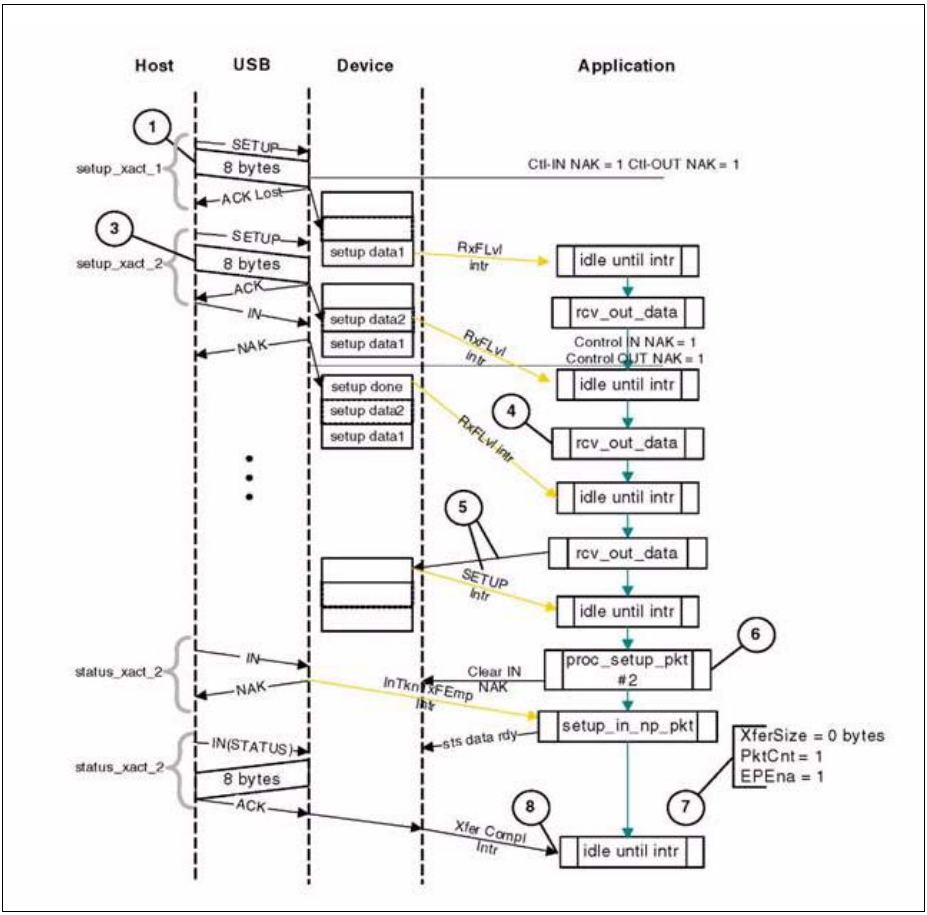


Figure 16-21 Two-Stage Control Transfer

Packet Read from FIFO in Slave Mode

This section describes how to read packets (OUT data and SETUP packets) from the receive FIFO in Slave mode.

1. On catching a GINTSTS.RxFLvl interrupt, the application must read the Receive Status Pop register (GRXSTSP).
2. The application can mask the GINTSTS.RxFLvl interrupt by writing to GINTMSK.RxFLvl = 0_B, until it has read the packet from the receive FIFO.

3. If the received packet's byte count is not 0, the byte count amount of data is popped from the receive Data FIFO and stored in memory. If the received packet byte count is 0, no data is popped from the Receive Data FIFO.
4. The receive FIFO's packet status readout indicates one of the following.
5. Global OUT NAK Pattern: PktSts = Global OUT NAK, BCnt = 11'h000, EPNum = Dont Care (4'h0), DPID = Dont Care (00_B). This data indicates that the global OUT NAK bit has taken effect.
   a) SETUP Packet Pattern: PktSts = SETUP, BCnt = 11'h008, EPNum = Control EP Num, DPID = D0. This data indicates that a SETUP packet for the specified endpoint is now available for reading from the receive FIFO.
   b) Setup Stage Done Pattern: PktSts = Setup Stage Done, BCnt = 11'h0, EPNum = Control EP Num, DPID = Don't Care (00_B). This data indicates that the Setup stage for the specified endpoint has completed and the Data stage has started. After this entry is popped from the receive FIFO, the core asserts a Setup interrupt on the specified control OUT endpoint.
   c) Data OUT Packet Pattern: PktSts = DataOUT, BCnt = size of the Received data OUT packet (0 < BCnt <1,024), EPNum = EPNum on which the packet was received, DPID = Actual Data PID.
   d) Data Transfer Completed Pattern: PktSts = Data OUT Transfer Done, BCnt = 11'h0, EPNum = OUT EP Num on which the data transfer is complete, DPID = Dont Care (00_B). This data indicates that a OUT data transfer for the specified OUT endpoint has completed. After this entry is popped from the receive FIFO, the core asserts a Transfer Completed interrupt on the specified OUT endpoint.
   The encoding for the PktSts is listed in **“Receive Status Debug Read/Status Read and Pop Registers (GRXSTSR/GRXSTSP)” on Page 16-246**.
6. After the data payload is popped from the receive FIFO, the GINTSTS.RxFLvl interrupt must be unmasked.
7. Steps 1-5 are repeated every time the application detects assertion of the interrupt line due to GINTSTS.RxFLvl. Reading an empty receive FIFO can result in undefined core behavior.

Figure 16-22 provides a flow chart of this procedure.

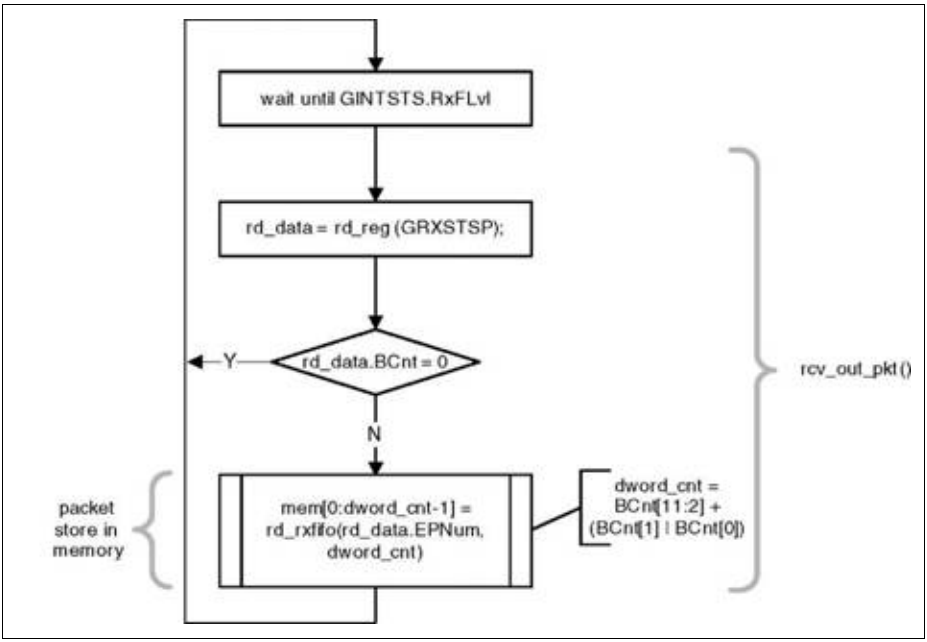


Figure 16-22 Receive FIFO Packet Read in Slave Mode

Setting the Global OUT NAK

Internal Data Flow

1. When the application sets the Global OUT NAK (DCTL.SGOUTNak), the core stops writing data, except SETUP packets, to the receive FIFO. Irrespective of the space availability in the receive FIFO, non-isochronous OUT tokens receive a NAK handshake response, and the core ignores isochronous OUT data packets
2. The core writes the Global OUT NAK pattern to the receive FIFO. The application must reserve enough receive FIFO space to write this data pattern. See **"Data FIFO RAM Allocation" on Page 16-201**.
3. When either the core (in DMA mode) or the application (in Slave mode) pops the Global OUT NAK pattern DWORD from the receive FIFO, the core sets the GINTSTS.GOUTNakEff interrupt.

4. Once the application detects this interrupt, it can assume that the core is in Global OUT NAK mode. The application can clear this interrupt by clearing the DCTL.SGOUTNak bit.

Application Programming Sequence

1. To stop receiving any kind of data in the receive FIFO, the application must set the Global OUT NAK bit by programming the following field.
   a) DCTL.SGOUTNak = 1_B
2. Wait for the assertion of the interrupt GINTSTS.GOUTNakEff. When asserted, this interrupt indicates that the core has stopped receiving any type of data except SETUP packets.
3. The application can receive valid OUT packets after it has set DCTL.SGOUTNak and before the core asserts the GINTSTS.GOUTNakEff interrupt.
4. The application can temporarily mask this interrupt by writing to the GINTMSK.GINNakEffMsk bit.
   a) GINTMSK.GINNakEffMsk = 0_B
5. Whenever the application is ready to exit the Global OUT NAK mode, it must clear the DCTL.SGOUTNak bit. This also clears the GINTSTS.GOUTNakEff interrupt.
   a) DCTL.CGOUTNak = 1_B
6. If the application has masked this interrupt earlier, it must be unmasked as follows:
   a) GINTMSK.GINNakEffMsk = 1_B

Disabling an OUT Endpoint

The application must use this sequence to disable an OUT endpoint that it has enabled.

Application Programming Sequence

1. Before disabling any OUT endpoint, the application must enable Global OUT NAK mode in the core, as described in **“Setting the Global OUT NAK” on Page 16-86**.
   a) DCTL.DCTL.SGOUTNak = 1_B
2. Wait for the GINTSTS.GOUTNakEff interrupt
3. Disable the required OUT endpoint by programming the following fields.
   a) DOEPCTLx.EPDisable = 1_B
   b) DOEPCTLx.SNAK = 1_B
4. Wait for the DOEPINTx.EPDisabled interrupt, which indicates that the OUT endpoint is completely disabled. When the EPDisabled interrupt is asserted, the core also clears the following bits.
   a) DOEPCTLx.EPDisable = 0_B
   b) DOEPCTLx.EPEnable = 0_B
5. The application must clear the Global OUT NAK bit to start receiving data from other non-disabled OUT endpoints.
   a) DCTL.SGOUTNak = 0_B

Stalling a Non-Isochronous OUT Endpoint

This section describes how the application can stall a non-isochronous endpoint.

1. Put the core in the Global OUT NAK mode, as described in **“Setting the Global OUT NAK” on Page 16-86**.
2. Disable the required endpoint, as described in **“Disabling an OUT Endpoint” on Page 16-87**.
   a) When disabling the endpoint, instead of setting the DOEPCTL.SNAK bit, set DOEPCTL.STALL = 1.
   - The Stall bit always takes precedence over the NAK bit.
3. When the application is ready to end the STALL handshake for the endpoint, the DOEPCTLx.STALL bit must be cleared.
4. If the application is setting or clearing a STALL for an endpoint due to a SetFeature.Endpoint Halt or ClearFeature.Endpoint Halt command, the Stall bit must be set or cleared before the application sets up the Status stage transfer on the control endpoint.

Generic Non-Isochronous OUT Data Transfers in DMA and Slave Modes

This section describes a regular non-isochronous OUT data transfer (control, bulk, or interrupt).

Application Requirements

1. Before setting up an OUT transfer, the application must allocate a buffer in the memory to accommodate all data to be received as part of the OUT transfer, then program that buffer's size and start address (in DMA mode) in the endpoint-specific registers.
2. For OUT transfers, the Transfer Size field in the endpoint's Transfer Size register must be a multiple of the maximum packet size of the endpoint, adjusted to the DWORD boundary.
   a) transfer size[epnum] = n * (mps[epnum] + 4 - (mps[epnum] mod 4))
   b) packet count[epnum] = n
   c) n > 0
3. In DMA mode, the core stores a received data packet in the memory, always starting on a DWORD boundary. If the maximum packet size of the endpoint is not a multiple of 4, the core inserts byte pads at end of a maximum-packet-size packet up to the end of the DWORD.
4. On any OUT endpoint interrupt, the application must read the endpoint's Transfer Size register to calculate the size of the payload in the memory. The received payload size can be less than the programmed transfer size.
   a) Payload size in memory = application-programmed initial transfer size - core updated final transfer size

b) Number of USB packets in which this payload was received = application-programmed initial packet count - core updated final packet count

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the endpoint-specific registers, clear the NAK bit, and enable the endpoint to receive the data.
2. Once the NAK bit is cleared, the core starts receiving data and writes it to the receive FIFO, as long as there is space in the receive FIFO. For every data packet received on the USB, the data packet and its status are written to the receive FIFO. Every packet (maximum packet size or short packet) written to the receive FIFO decrements the Packet Count field for that endpoint by 1.
   a) OUT data packets received with Bad Data CRC are flushed from the receive FIFO automatically.
   b) After sending an ACK for the packet on the USB, the core discards non-isochronous OUT data packets that the host, which cannot detect the ACK, re-sends. The application does not detect multiple back-to-back data OUT packets on the same endpoint with the same data PID. In this case the packet count is not decremented.
   c) If there is no space in the receive FIFO, isochronous or non-isochronous data packets are ignored and not written to the receive FIFO. Additionally, non-isochronous OUT tokens receive a NAK handshake reply.
   d) In all the above three cases, the packet count is not decremented because no data is written to the receive FIFO.
3. When the packet count becomes 0 or when a short packet is received on the endpoint, the NAK bit for that endpoint is set. Once the NAK bit is set, the isochronous or non-isochronous data packets are ignored and not written to the receive FIFO, and non-isochronous OUT tokens receive a NAK handshake reply.
4. After the data is written to the receive FIFO, either the application (in Slave mode) or the core's DMA engine (in External or Internal DMA mode), reads the data from the receive FIFO and writes it to external memory, one packet at a time per endpoint.
5. At the end of every packet write on the AHB to external memory, the transfer size for the endpoint is decremented by the size of the written packet.
6. The OUT Data Transfer Completed pattern for an OUT endpoint is written to the receive FIFO on one of the following conditions.
   a) The transfer size is 0 and the packet count is 0
   b) The last OUT data packet written to the receive FIFO is a short packet (0 ^packet size < maximum packet size)
7. When either the application or the DMA pops this entry (OUT Data Transfer Completed), a Transfer Completed interrupt is generated for the endpoint and the endpoint enable is cleared.

Application Programming Sequence

1. Program the DOEPTSIZx register for the transfer size and the corresponding packet count. Additionally, in DMA mode, program the DOEPDMAx register.
2. Program the DOEPCTLx register with the endpoint characteristics, and set the Endpoint Enable and ClearNAK bits.
   a) DOEPCTLx.EPEna = 1
   b) DOEPCTLx.CNAK = 1
3. In Slave mode, wait for the GINTSTS.Rx StsQ level interrupt and empty the data packets from the receive FIFO as explained in **“Packet Read from FIFO in Slave Mode” on Page 16-84**.
   a) This step can be repeated many times, depending on the transfer size.
4. Asserting the DOEPINTx.XferCompl interrupt marks a successful completion of the non- isochronous OUT data transfer.
5. Read the DOEPTSIZx register to determine the size of the received data payload.

Slave Mode Bulk OUT Transaction

Figure 16-23 depicts the reception of a single bulk OUT data packet from the USB to the AHB and describes the events involved in the process.

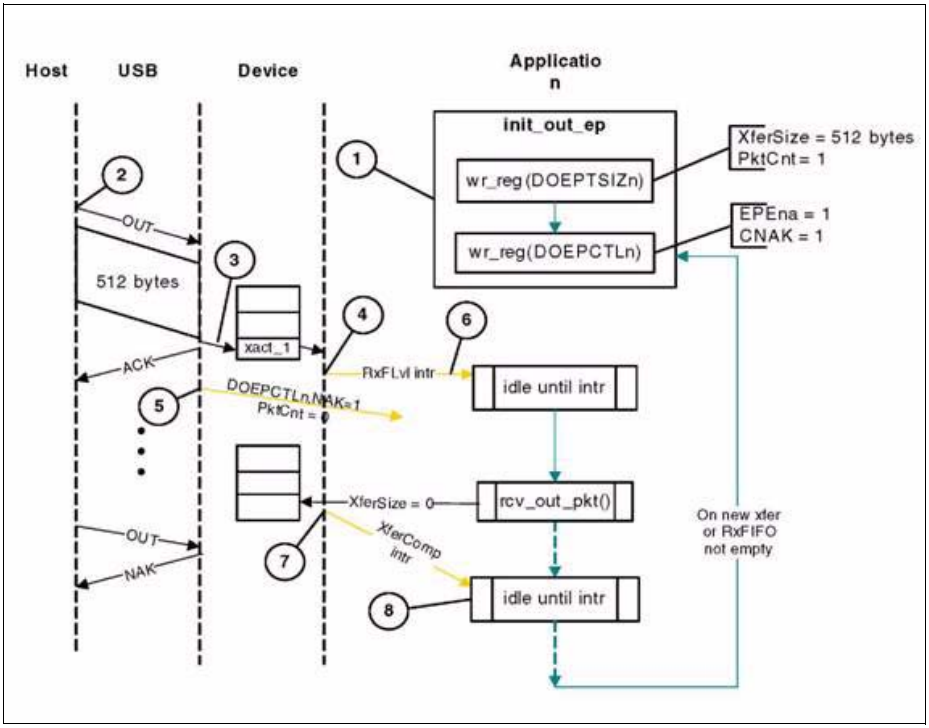


Figure 16-23 Slave Mode Bulk OUT Transaction

After a SetConfiguration/SetInterface command, the application initializes all OUT endpoints by setting DOEPCTLx.CNAK = 1 and DOEPCTLx.EPEna = 1, and setting a suitable XferSize and PktCnt in the DOEPTSIZx register.

1. Host attempts to send data (OUT token) to an endpoint.
2. When the core receives the OUT token on the USB, it stores the packet in the RxFIFO because space is available there.
3. After writing the complete packet in the RxFIFO, the core then asserts the GINTSTS.RxFLvl interrupt.
4. On receiving the PktCnt number of USB packets, the core sets the NAK bit for this endpoint internally to prevent it from receiving any more packets.
5. The application processes the interrupt and reads the data from the RxFIFO.

6. When the application has read all the data (equivalent to XferSize), the core generates a DOEPINTx.XferCompl interrupt.
7. The application processes the interrupt and uses the setting of the DOEPINTx.XferCompl interrupt bit to determine that the intended transfer is complete.

Generic Isochronous OUT Data Transfer in DMA and Slave Modes

This section describes a regular isochronous OUT data transfer.

Application Requirements

1. All the application requirements for non-isochronous OUT data transfers also apply to isochronous OUT data transfers
2. For isochronous OUT data transfers, the Transfer Size and Packet Count fields must always be set to the number of maximum-packet-size packets that can be received in a single frame and no more. Isochronous OUT data transfers cannot span more than 1 frame.
   a) 1 <= packet count[epnum] <= 3
3. In Slave mode, when isochronous OUT endpoints are supported in the device, the application must read all isochronous OUT data packets from the receive FIFO (data and status) before the end of the periodic frame (GINTSTS.EOPF interrupt). In DMA mode, the application must guarantee enough bandwidth to allow emptying the isochronous OUT data packet from the receive FIFO before the end of each periodic frame.
4. To receive data in the following frame, an isochronous OUT endpoint must be enabled after the GINTSTS.EOPF and before the GINTSTS.SOF.

Internal Data Flow

1. The internal data flow for isochronous OUT endpoints is the same as that for non-isochronous OUT endpoints, but for a few differences.
2. When an isochronous OUT endpoint is enabled by setting the Endpoint Enable and clearing the NAK bits, the Even/Odd frame bit must also be set appropriately. The core receives data on a isochronous OUT endpoint in a particular frame only if the following condition is met.
   a) DOEPCTLx.Even/Odd frame = DSTS.SOFFN[0]
3. When either the application or the external/internal DMA completely reads an isochronous OUT data packet (data and status) from the receive FIFO, the core updates the DOEPTSIZx.Received DPID field with the data PID of the last isochronous OUT data packet read from the receive FIFO.

Application Programming Sequence

1. Program the DOEPTSIZx register for the transfer size and the corresponding packet count. When in DMA mode, also program the DOEPDMAx register.
2. Program the DOEPCTLx register with the endpoint characteristics and set the Endpoint Enable, ClearNAK, and Even/Odd frame bits.
   a) Endpoint Enable = 1
   b) CNAK=1
   c) Even/Odd frame = (0: Even/1: Odd)
3. In Slave mode, wait for the GINTSTS.Rx StsQ level interrupt and empty the data packets from the receive FIFO as explained in **“Packet Read from FIFO in Slave Mode” on Page 16-84**.
   a) This step can be repeated many times, depending on the transfer size.
4. The assertion of the DOEPINTx.XferCompl interrupt marks the completion of the isochronous OUT data transfer. This interrupt does not necessarily mean that the data in memory is good.
5. This interrupt can not always be detected for isochronous OUT transfers. Instead, the application can detect the GINTSTS.incomplete Isochronous OUT data interrupt. See **“Incomplete Isochronous OUT Data Transfers in DMA and Slave Modes” on Page 16-100**, for more details
6. Read the DOEPTSIZx register to determine the size of the received transfer and to determine the validity of the data received in the frame. The application must treat the data received in memory as valid only if one of the following conditions is met.
   a) DOEPTSIZx.RxDPID = D0 and the number of USB packets in which this payload was received = 1
   b) DOEPTSIZx.RxDPID = D1 and the number of USB packets in which this payload was received = 2
   c) DOEPTSIZx.RxDPID = D2 and the number of USB packets in which this payload was received = 3
7. The number of USB packets in which this payload was received = App Programmed Initial Packet Count - Core Updated Final Packet Count
   The application can discard invalid data packets.

Generic Interrupt OUT Data Transfers Using Periodic Transfer Interrupt Feature

This section describes a regular INTR OUT data transfer with the Periodic Transfer Interrupt feature.

Application Requirements

1. Before setting up a periodic OUT transfer, the application must allocate a buffer in the memory to accommodate all data to be received as part of the OUT transfer, then program that buffer's size and start address in the endpoint-specific registers.

2. For Interrupt OUT transfers, the Transfer Size field in the endpoint's Transfer Size register must be a multiple of the maximum packet size of the endpoint, adjusted to the DWORD boundary. The Transfer Size programmed can span across multiple frames based on the periodicity after which the application want to receive the DOEPINTx.XferCompl interrupt
3. transfer size[epnum] = n * (mps[epnum] + 4 - (mps[epnum] mod 4))
4. packet count[epnum] = n
5. n > 0 (Higher value of n reduces the periodicity of the DOEPINTx.XferCompl interrupt)
6. 1 < packet count[epnum] < n (Higher value of n reduces the periodicity of the DOEPINTx.XferCompl interrupt)
7. In DMA mode, the core stores a received data packet in the memory, always starting on a DWORD boundary. If the maximum packet size of the endpoint is not a multiple of 4, the core inserts byte pads at end of a maximum-packet-size packet up to the end of the DWORD. The application will not be informed about the frame number on which a specific packet has been received.
8. On DOEPINTx.XferCompl interrupt, the application must read the endpoint's Transfer Size register to calculate the size of the payload in the memory. The received payload size can be less than the programmed transfer size.
9. Payload size in memory = application-programmed initial transfer size - core updated final transfer size
10. Number of USB packets in which this payload was received = application-programmed initial packet count - core updated final packet count.
11. If for some reason, the host stops sending tokens, there are no interrupts to the application, and the application must timeout on its own.
12. The assertion of the DOEPINTx.XferCompl interrupt marks the completion of the interrupt OUT data transfer. This interrupt does not necessarily mean that the data in memory is good.
13. Read the DOEPTSIZx register to determine the size of the received transfer and to determine the validity of the data received in the frame.

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the endpoint-specific registers, clear the NAK bit, and enable the endpoint to receive the data.
2. The application must mask the GINTSTS.incompISOOUT.
3. The application must enable the DCTL.IgnrFrmNum
4. When an interrupt OUT endpoint is enabled by setting the Endpoint Enable and clearing the NAK bits, the Even/Odd frame will be ignored by the core.
5. Once the NAK bit is cleared, the core starts receiving data and writes it to the receive FIFO, as long as there is space in the receive FIFO. For every data packet received on the USB, the data packet and its status are written to the receive FIFO. Every

packet (maximum packet size or short packet) written to the receive FIFO decrements the Packet Count field for that endpoint by 1.
6. OUT data packets received with Bad Data CRC or any packet error are flushed from the receive FIFO automatically.
7. Interrupt packets with PID errors are not passed to application. Core discards the packet, sends ACK and does not decrement packet count.
8. If there is no space in the receive FIFO, interrupt data packets are ignored and not written to the receive FIFO. Additionally, interrupt OUT tokens receive a NAK handshake reply.
9. When the packet count becomes 0 or when a short packet is received on the endpoint, the NAK bit for that endpoint is set. Once the NAK bit is set, the isochronous or interrupt data packets are ignored and not written to the receive FIFO, and interrupt OUT tokens receive a NAK handshake reply.
10. After the data is written to the receive FIFO, the core's DMA engine reads the data from the receive FIFO and writes it to external memory, one packet at a time per endpoint.
11. At the end of every packet write on the AHB to external memory, the transfer size for the endpoint is decremented by the size of the written packet.
12. The OUT Data Transfer Completed pattern for an OUT endpoint is written to the receive FIFO on one of the following conditions.
13. The transfer size is 0 and the packet count is 0.
14. The last OUT data packet written to the receive FIFO is a short packet (0 < packet size < maximum packet size)
15. When either the application or the DMA pops this entry (OUT Data Transfer Completed), a Transfer Completed interrupt is generated for the endpoint and the endpoint enable is cleared.

Generic Isochronous OUT Data Transfers Using Periodic Transfer Interrupt Feature

This section describes a regular isochronous OUT data transfer with the Periodic Transfer Interrupt feature.

Application Requirements

1. Before setting up ISOC OUT transfers spanned across multiple frames, the application must allocate buffer in the memory to accommodate all data to be received as part of the OUT transfers, then program that buffer's size and start address in the endpoint-specific registers.
   a) The application must mask the GINTSTS.incomp ISO OUT.
   b) The application must enable the DCTL.IgnrFrmNum
2. For ISOC transfers, the Transfer Size field in the DOEPTSIZx.XferSize register must be a multiple of the maximum packet size of the endpoint, adjusted to the DWORD boundary. The Transfer Size programmed can span across multiple frames based on

the periodicity after which the application wants to receive the DOEPINTx.XferCompl interrupt
a) transfer size[epnum] = n * (mps[epnum] + 4 - (mps[epnum] mod 4))
b) packet count[epnum] = n
c) n > 0 (Higher value of n reduces the periodicity of the DOEPINTx.XferCompl interrupt)
d) 1 =< packet count[epnum] =< n (Higher value of n reduces the periodicity of the DOEPINTx.XferCompl interrupt).

3. In DMA mode, the core stores a received data packet in the memory, always starting on a DWORD boundary. If the maximum packet size of the endpoint is not a multiple of 4, the core inserts byte pads at end of a maximum-packet-size packet up to the end of the DWORD. The application will not be informed about the frame number and the PID value on which a specific OUT packet has been received.
4. The assertion of the DOEPINTx.XferCompl interrupt marks the completion of the isochronous OUT data transfer. This interrupt does not necessarily mean that the data in memory is good.
   a) On DOEPINTx.XferCompl, the application must read the endpoint's Transfer Size register to calculate the size of the payload in the memory.
   b) Payload size in memory = application-programmed initial transfer size - core updated final transfer size
   c) Number of USB packets in which this payload was received = application-programmed initial packet count - core updated final packet count.
   d) If for some reason, the host stop sending tokens, there will be no interrupt to the application, and the application must timeout on its own.
5. The assertion of the DOEPINTx.XferCompl can also mark a packet drop on USB due to unavailability of space in the RxFifo or due to any packet errors.
   a) The application must read the DOEPINTx.PktDrpSts (DOEPINTx.Bit[11] is now used as the DOEPINTx.PktDrpSts) register to differentiate whether the DOEPINTx.XferCompl was generated due to the normal end of transfer or due to dropped packets. In case of packets being dropped on the USB due to unavailability of space in the RxFifo or due to any packet errors the endpoint enable bit is cleared.
   b) In case of packet drop on the USB application must re-enable the endpoint after recalculating the values DOEPTSIZx.XferSize and DOEPTSIZx.PktCnt.
   c) Payload size in memory = application-programmed initial transfer size - core updated final transfer size
   d) Number of USB packets in which this payload was received = application-programmed initial packet count - core updated final packet count.

Note: Due to application latencies it is possible that DOEPINT.XferComplete interrupt is generated without DOEPINT.PktDrpSts being set, This scenario is possible only if back-to-back packets are dropped for consecutive frames and the PktDrpSts is merged, but the XferSize and PktCnt values for the endpoint are nonzero. In this

case, the application must proceed further by programming the PktCnt and XferSize register for the next frame, as it would if PktDrpSts were being set.

Figure 16-24 gives the application flow for Isochronous OUT Periodic Transfer Interrupt feature.

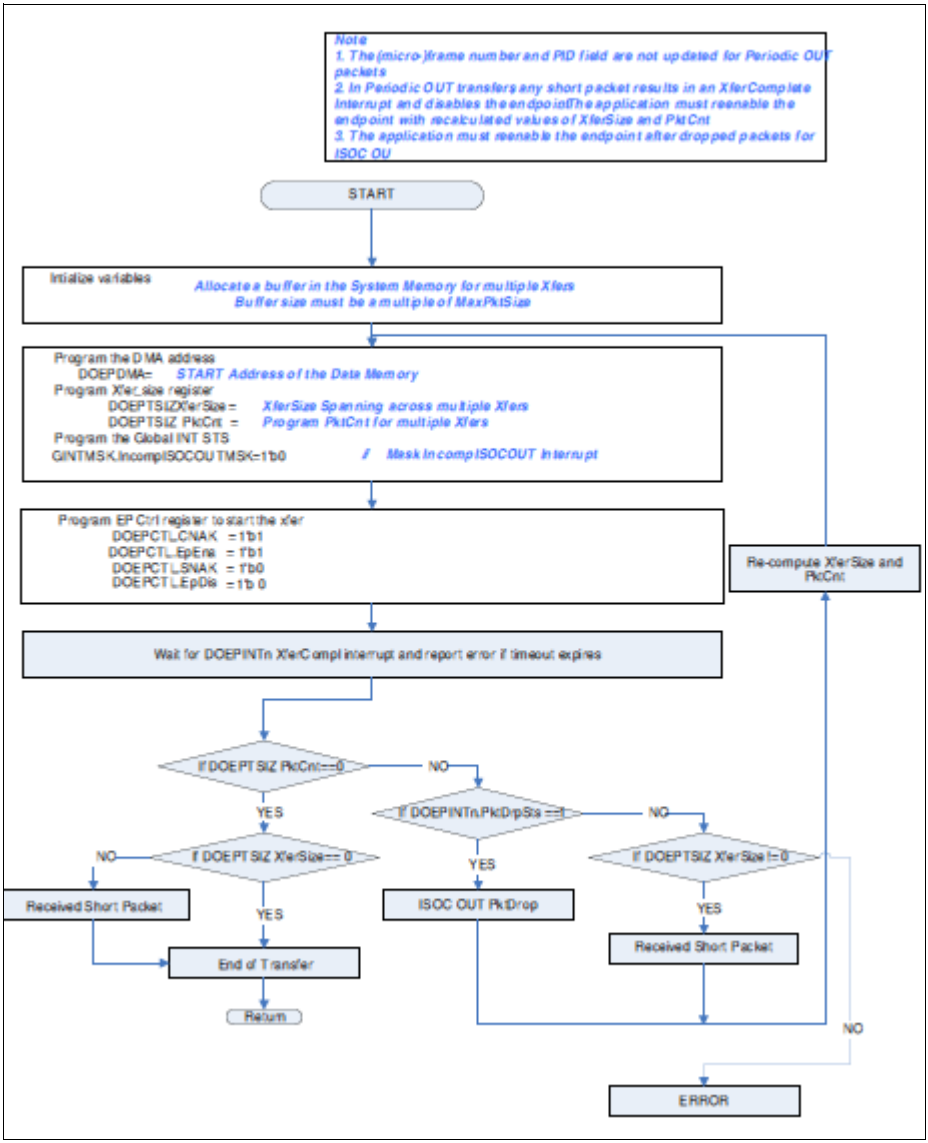


Figure 16-24 ISOC OUT Application Flow for Periodic Transfer Interrupt Feature

Internal Data Flow

1. The application must set the Transfer Size, Packets to be received in a frame and Packet Count Fields in the endpoint-specific registers, clear the NAK bit, and enable the endpoint to receive the data.
2. When an isochronous OUT endpoint is enabled by setting the Endpoint Enable and clearing the NAK bits, the Even/Odd frame will be ignored by the core.
3. Once the NAK bit is cleared, the core starts receiving data and writes it to the receive FIFO, as long as there is space in the receive FIFO. For every data packet received on the USB, the data packet and its status are written to the receive FIFO. Every packet (maximum packet size or short packet) written to the receive FIFO decrements the Packet Count field for that endpoint by 1.
4. When the packet count becomes 0 or when a short packet is received on the endpoint, the NAK bit for that endpoint is set. Once the NAK bit is set, the ISOC packets are ignored and not written to the receive FIFO.
5. After the data is written to the receive FIFO, the core's DMA engine, reads the data from the receive FIFO and writes it to external memory, one packet at a time per endpoint.
6. At the end of every packet write on the AHB to external memory, the transfer size for the endpoint is decremented by the size of the written packet.
7. The OUT Data Transfer Completed pattern for an OUT endpoint is written to the receive FIFO on one of the following conditions.
   a) The transfer size is 0 and the packet count is 0
   b) The last OUT data packet written to the receive FIFO is a short packet (0 < packet size < maximum packet size).
8. When the DMA pops this entry (OUT Data Transfer Completed), a Transfer Completed interrupt is generated for the endpoint or the endpoint enable is cleared.
9. OUT data packets received with Bad Data CRC or any packet error are flushed from the receive FIFO automatically.
   a) In these two cases, the packet count and transfer size registers are not decremented because no data is written to the receive FIFO.

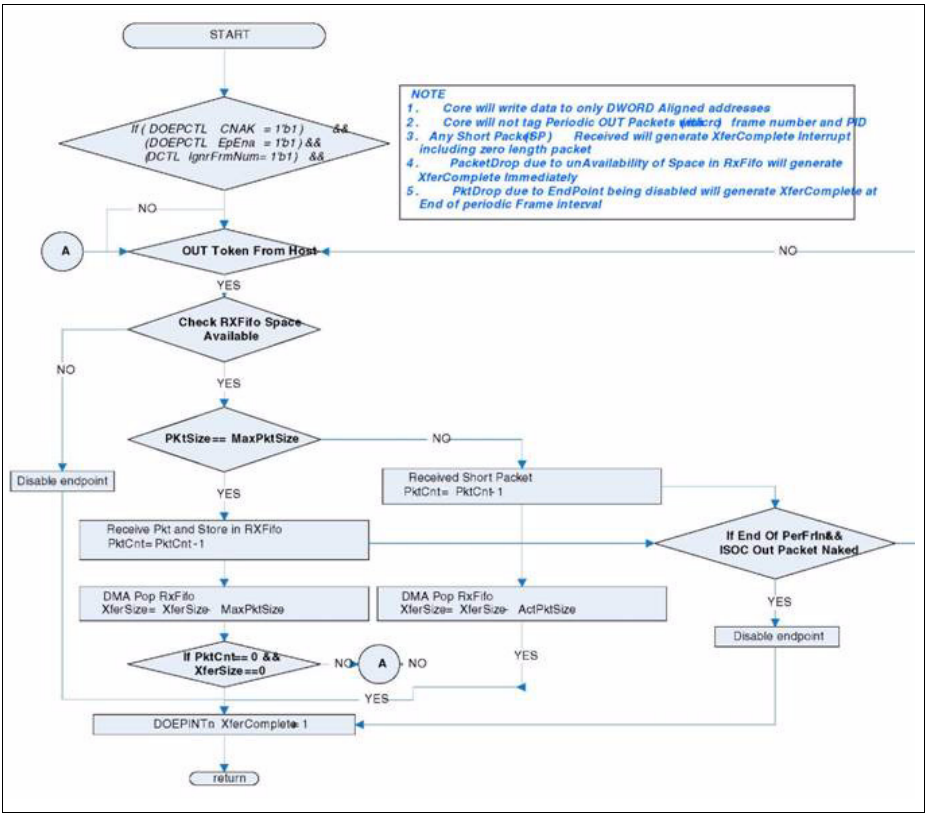


Figure 16-25 Isochronous OUT Core Internal Flow for Periodic Transfer Interrupt Feature

Incomplete Isochronous OUT Data Transfers in DMA and Slave Modes

This section describes the application programming sequence when isochronous OUT data packets are dropped inside the core.

Internal Data Flow

1. For isochronous OUT endpoints, the DOEPINTx.XferCompl interrupt possibly is not always asserted. If the core drops isochronous OUT data packets, the application could fail to detect the DOEPINTx.XferCompl interrupt under the following circumstances.
   a) When the receive FIFO cannot accommodate the complete ISO OUT data packet, the core drops the received ISO OUT data.

b) When the isochronous OUT data packet is received with CRC errors
c) When the isochronous OUT token received by the core is corrupted
d) When the application is very slow in reading the data from the receive FIFO

2. When the core detects an end of periodic frame before transfer completion to all isochronous OUT endpoints, it asserts the GINTSTS.incomplete Isochronous OUT data interrupt, indicating that a DOEPINTx.XferCompl interrupt is not asserted on at least one of the isochronous OUT endpoints. At this point, the endpoint with the incomplete transfer remains enabled, but no active transfers remains in progress on this endpoint on the USB.
3. This step is applicable only if the OTG core is operating in slave mode. Application Programming Sequence
4. This step is applicable only if the OTG core is operating in slave mode. Asserting the GINTSTS.incomplete Isochronous OUT data interrupt indicates that in the current frame, at least one isochronous OUT endpoint has an incomplete transfer.
5. If this occurs because isochronous OUT data is not completely emptied from the endpoint, the application must ensure that the DMA or the application empties all isochronous OUT data (data and status) from the receive FIFO before proceeding.
   a) When all data is emptied from the receive FIFO, the application can detect the DOEPINTx.XferCompl interrupt. In this case, the application must re-enable the endpoint to receive isochronous OUT data in the next frame, as described in **“Generic Isochronous OUT Data Transfer in DMA and Slave Modes” on Page 16-92**.
6. When it receives a GINTSTS.incomplete Isochronous OUT data interrupt, the application must read the control registers of all isochronous OUT endpoints (DOEPCTLx) to determine which endpoints had an incomplete transfer in the current frame. An endpoint transfer is incomplete if both the following conditions are met.
   a) DOEPCTLx.Even/Odd frame bit = DSTS.SOFFN[0]
   b) DOEPCTLx.Endpoint Enable = 1
7. The previous step must be performed before the GINTSTS.SOF interrupt is detected, to ensure that the current frame number is not changed.
8. For isochronous OUT endpoints with incomplete transfers, the application must discard the data in the memory and disable the endpoint by setting the DOEPCTLx.Endpoint Disable bit.
9. Wait for the DOEPINTx.Endpoint Disabled interrupt and enable the endpoint to receive new data in the next frame as explained in **“Generic Isochronous OUT Data Transfer in DMA and Slave Modes” on Page 16-92**.
   a) Because the core can take some time to disable the endpoint, the application possibly is not able to receive the data in the next frame after receiving bad isochronous data.

16.6.3.3 IN Data Transfers in Device Slave and Buffer DMA Modes

This section describes the internal data flow and application-level operations during IN data transfers.

- **Packet Write in Slave Mode**
- **Setting IN Endpoint NAK**
- **IN Endpoint Disable**
- **Bulk IN Stall**
- **Incomplete Isochronous IN Data Transfers**
- **Stalling Non-Isochronous IN Endpoints**
- **Non-Periodic IN Endpoint Sequencing**
- **Worst-Case Response Time**
- **Choosing the Value of GUSBCFG.USBTrdTim**
- **Handling Babble Conditions**
- **Generic Non-Periodic (Bulk and Control) IN Data Transfers**
- **Examples**
- **Generic Periodic IN (Interrupt and Isochronous) Data Transfers**
- **Generic Periodic IN Data Transfers Using the Periodic Transfer Interrupt Feature**

Packet Write in Slave Mode

This section describes how the application writes data packets to the endpoint FIFO in Slave mode with dedicated transmit FIFOs.

1. The application can either choose polling or interrupt mode.
   a) In polling mode, application monitors the status of the endpoint transmit data FIFO, by reading the DTXFSTSx register, to determine, if there is enough space in the data FIFO.
   b) In interrupt mode, application waits for the DIEPINTx.TxFEmp interrupt and then reads the DTXFSTSx register, to determine, if there is enough space in the data FIFO.
   c) To write a single non-zero length data packet, there must be space to write the entire packet is the data FIFO.
   d) For writing zero length packet, application must not look for FIFO space.
2. Using one of the above mentioned methods, when the application determines that there is enough space to write a transmit packet, the application must first write into the endpoint control register, before writing the data into the data FIFO. The application, typically must do a read modify write on the DIEPCTLx, to avoid modifying the contents of the register, except for setting the Endpoint Enable bit.

The application can write multiple packets for the same endpoint, into the transmit FIFO, if space is available. For periodic IN endpoints, application must write packets only for one frame. It can write packets for the next periodic transaction, only after getting transfer complete for the previous transaction.

Setting IN Endpoint NAK

Internal Data Flow

1. When the application sets the IN NAK for a particular endpoint, the core stops transmitting data on the endpoint, irrespective of data availability in the endpoint's transmit FIFO.
2. Non-isochronous IN tokens receive a NAK handshake reply
   a) Isochronous IN tokens receive a zero-data-length packet reply
3. The core asserts the DIEPINTx.IN NAK Effective interrupt in response to the DIEPCTL.Set NAK bit.
4. Once this interrupt is seen by the application, the application can assume that the endpoint is in IN NAK mode. This interrupt can be cleared by the application by setting the DIEPCTLx. Clear NAK bit.

Application Programming Sequence

1. To stop transmitting any data on a particular IN endpoint, the application must set the IN NAK bit. To set this bit, the following field must be programmed.
   a) DIEPCTLx.SetNAK = 1_B
2. Wait for assertion of the DIEPINTx.NAK Effective interrupt. This interrupt indicates the core has stopped transmitting data on the endpoint.
3. The core can transmit valid IN data on the endpoint after the application has set the NAK bit, but before the assertion of the NAK Effective interrupt.
4. The application can mask this interrupt temporarily by writing to the DIEPMSK.NAK Effective bit.
   a) DIEPMSK.NAK Effective = 0_B
5. To exit Endpoint NAK mode, the application must clear the DIEPCTLx.NAK status. This also clears the DIEPINTx.NAK Effective interrupt.
   a) DIEPCTLx.ClearNAK = 1_B
6. If the application masked this interrupt earlier, it must be unmasked as follows:
   a) DIEPMSK.NAK Effective = 1_B

IN Endpoint Disable

Use the following sequence to disable a specific IN endpoint (periodic/non-periodic) that has been previously enabled in dedicated FIFO operation.

Application Programming Sequence:

1. In Slave mode, the application must stop writing data on the AHB, for the IN endpoint to be disabled.
2. The application must set the endpoint in NAK mode. See **“Setting IN Endpoint NAK” on Page 16-103**.

a) DIEPCTLx.SetNAK = 1_B
3. Wait for DIEPINTx.NAK Effective interrupt.
4. Set the following bits in the DIEPCTLx register for the endpoint that must be disabled.
   a) DIEPCTLx.Endpoint Disable = 1
   b) DIEPCTLx.SetNAK = 1
5. Assertion of DIEPINTx.Endpoint Disabled interrupt indicates that the core has completely disabled the specified endpoint. Along with the assertion of the interrupt, the core also clears the following bits.
   a) DIEPCTLx.EPEnable = 0_B
   b) DIEPCTLx.EPDisable = 0_B
6. The application must read the DIEPTSIZx register for the periodic IN EP, to calculate how much data on the endpoint was transmitted on the USB.
7. The application must flush the data in the Endpoint transmit FIFO, by setting the following fields in the GRSTCTL register.
   a) GRSTCTL.TxFIFONum = Endpoint Transmit FIFO Number
   b) GRSTCTL.TxFFlush = 1

The application must poll the GRSTCTL register, until the TxFFlush bit is cleared by the core, which indicates the end of flush operation. To transmit new data on this endpoint, the application can re-enable the endpoint at a later point.

Bulk IN Stall

These notes refer to **Figure 16-26**.

1. The application has scheduled an IN transfer on receiving the DIEPINTx.InTknRcvd When TxFIFO Empty interrupt.
2. When the transfer is in progress, the application must force a STALL on the endpoint. This could be because the application has received a SetFeature.Endpoint Halt command. The application sets the Stall bit, disables the endpoint and waits for the DIEPINTx.Endpoint Disabled interrupt. This generates STALL handshakes for the endpoint on the USB.
3. On receiving the interrupt, the application flushes the Non-periodic Transmit FIFO and clears the DCTL.GlobalINNPNAK bit.
4. On receiving the ClearFeature.Endpoint Halt command, the application clears the Stall bit.
5. The endpoint behaves normally and the application can re-enable the endpoint for new transfers

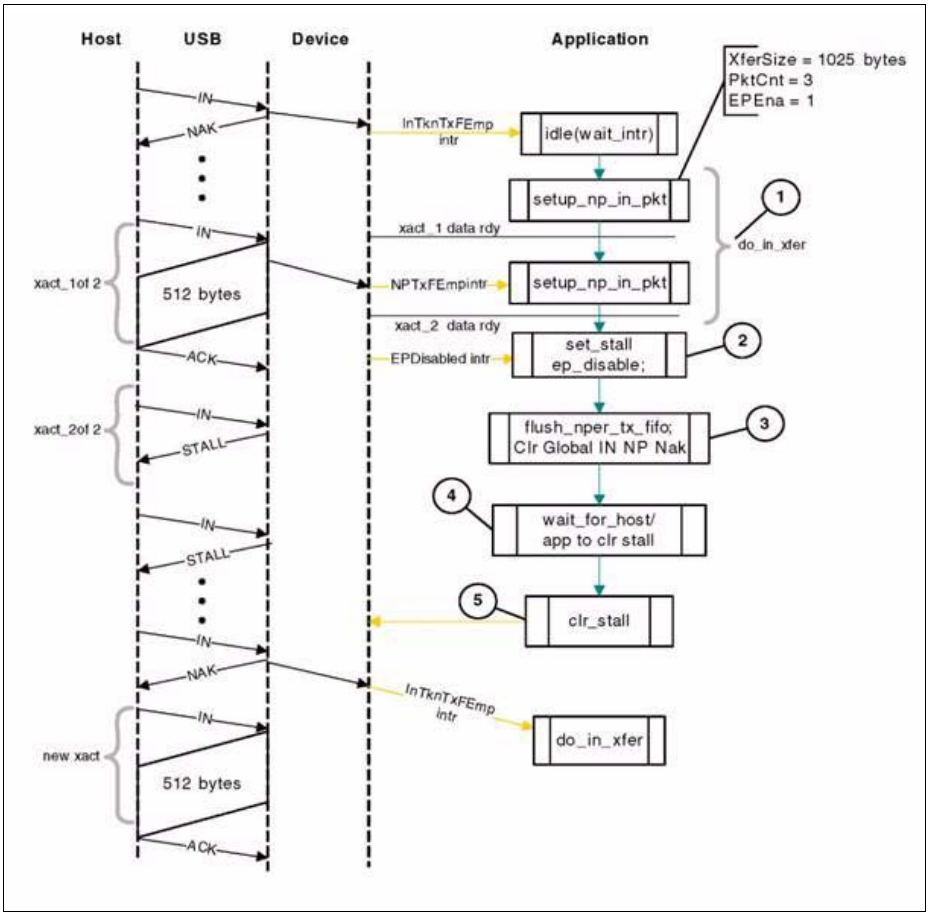


Figure 16-26 Bulk IN Stall

Incomplete Isochronous IN Data Transfers

This section describes what the application must do on an incomplete isochronous IN data transfer.

Internal Data Flow

1. An isochronous IN transfer is treated as incomplete in one of the following conditions.
   a) The core receives a corrupted isochronous IN token on at least one isochronous IN endpoint. In this case, the application detects a GINTSTS.incomplete Isochronous IN Transfer interrupt.
   b) The application or DMA is slow to write the complete data payload to the transmit FIFO and an IN token is received before the complete data payload is written to the FIFO. In this case, the application detects a DIEPINTx.IN Tkn Rcvd When TxFIFO Empty interrupt. The application can ignore this interrupt, as it eventually results in a GINTSTS.incomplete Isochronous IN Transfer interrupt at the end of periodic frame.
   - The core transmits a zero-length data packet on the USB in response to the received IN token.
2. In either of the aforementioned cases, in Slave mode, the application must stop writing the data payload to the transmit FIFO as soon as possible.
3. The application must set the NAK bit and the disable bit for the endpoint. In DMA mode, the core automatically stops fetching the data payload when the endpoint disable bit is set.
4. The core disables the endpoint, clears the disable bit, and asserts the Endpoint Disable interrupt for the endpoint.

Application Programming Sequence

1. The application can ignore the DIEPINTx.IN Tkn Rcvd When TxFIFO empty interrupt on any isochronous IN endpoint, as it eventually results in a GINTSTS.incomplete Isochronous IN Transfer interrupt.
2. Assertion of the GINTSTS.incomplete Isochronous IN Transfer interrupt indicates an incomplete isochronous IN transfer on at least one of the isochronous IN endpoints.
3. The application must read the Endpoint Control register for all isochronous IN endpoints to detect endpoints with incomplete IN data transfers.
4. In Slave mode, the application must stop writing data to the Periodic Transmit FIFOs associated with these endpoints on the AHB.
5. In both modes of operation, program the following fields in the DIEPCTLx register to disable the endpoint. See **“IN Endpoint Disable” on Page 16-103** for more details.
   a) DIEPCTLx.SetNAK = 1
   b) DIEPCTLx.Endpoint Disable = 1
6. The DIEPINTx.Endpoint Disabled interrupt's assertion indicates that the core has disabled the endpoint.

a) At this point, the application must flush the data in the associated transmit FIFO or overwrite the existing data in the FIFO by enabling the endpoint for a new transfer in the next frame. To flush the data, the application must use the GRSTCTL register.

Stalling Non-Isochronous IN Endpoints

This section describes how the application can stall a non-isochronous endpoint.

Application Programming Sequence

1. Disable the IN endpoint to be stalled. See **“IN Endpoint Disable” on Page 16-103** for more details. Set the Stall bit as well.
2. DIEPCTLx.Endpoint Disable = 1, when the endpoint is already enabled
   a) DIEPCTLx.STALL = 1
   b) The Stall bit always takes precedence over the NAK bit
3. Assertion of the DIEPINTx.Endpoint Disabled interrupt indicates to the application that the core has disabled the specified endpoint.
4. The application must flush the Non-periodic or Periodic Transmit FIFO, depending on the endpoint type. In case of a non-periodic endpoint, the application must re-enable the other non-periodic endpoints, which do not need to be stalled, to transmit data.
5. Whenever the application is ready to end the STALL handshake for the endpoint, the DIEPCTLx.STALL bit must be cleared.
6. If the application sets or clears a STALL for an endpoint due to a SetFeature.Endpoint Halt command or ClearFeature.Endpoint Halt command, the Stall bit must be set or cleared before the application sets up the Status stage transfer on the control endpoint.

Special Case: Stalling the Control IN/OUT Endpoint

The core must stall IN/OUT tokens if, during the Data stage of a control transfer, the host sends more IN/OUT tokens than are specified in the SETUP packet. In this case, the application must to enable DIEPINTx.INTknTXFEmp and DOEPINTx.OUTTknEPdis interrupts during the Data stage of the control transfer, after the core has transferred the amount of data specified in the SETUP packet. Then, when the application receives this interrupt, it must set the STALL bit in the corresponding endpoint control register, and clear this interrupt.

Non-Periodic IN Endpoint Sequencing

In DMA mode, the DIEPCTLx.NextEp value programmed controls the order in which the core fetches non- periodic data for IN endpoints.

If application requires the core to fetch data for the non-periodic IN endpoints in a certain endpoint order, it must program the DIEPCTLx.NextEP field accordingly before enabling the endpoints. To enable a single endpoint enabled at a time the application must set the

DIEPCTLx.NextEP field to the endpoint number itself. The core uses the NextEP field irrespective of the DIEPCTLx.EPEna bit.

Worst-Case Response Time

When the USB core acts as a device, there is a worst case response time for any tokens that follow an isochronous OUT. This worst case response time depends on the AHB clock frequency.

The core registers are in the AHB domain, and the core does not accept another token before updating these register values. The worst case is for any token following an isochronous OUT, because for an isochronous transaction, there is no handshake and the next token could come sooner. This worst case value is 7 PHY clocks when the AHB clock is the same as the PHY clock. When AHB clock is faster, this value is smaller.

If this worst case condition occurs, the core responds to bulk/ interrupt tokens with a NAK and drops isochronous and SETUP tokens. The host interprets this as a timeout condition for SETUP and retries the SETUP packet. For isochronous transfers, the incomplISOCIN and incomplISOCOUT interrupts inform the application that isochronous IN/OUT packets were dropped.

Choosing the Value of GUSBCFG.USBTrdTim

The value in GUSBCFG.USBTrdTim is the time it takes for the MAC, in terms of PHY clocks after it has received an IN token, to get the FIFO status, and thus the first data from PFC (Packet FIFO Controller) block. This time involves the synchronization delay between the PHY and AHB clocks. The worst case delay for this is when the AHB clock is the same as the PHY clock. In this case, the delay is 5 clocks. If the PHY clock is running at 60 MHz and the AHB is running at 30 MHz, this value is 9 clocks.

Once the MAC receives an IN token, this information (token received) is synchronized to the AHB clock by the PFC (the PFC runs on the AHB clock). The PFC then reads the data from the SPRAM and writes it into the dual clock source buffer. The MAC then reads the data out of the source buffer (4 deep).

If the AHB is running at a higher frequency than the PHY, the application can use a smaller value for GUSBCFG.USBTrdTim. **Figure 16-27** explains the 5-clock delay. This diagram has the following signals:

- tkn_rcvd: Token received information from MAC to PFC
- dynced_tkn_rcvd: Doubled sync tkn_rcvd, from pclk to hclk domain
- spr_read: Read to SPRAM
- spr_addr: Address to SPRAM
- spr_rdata: Read data from SPRAM
- srcbuf_push: Push to the source buffer
- srcbuf_rdata: Read data from the source buffer. Data seen by MAC

The application can use the following formula to calculate the value of GUSBCFG.USBTrdTim:

4 * AHB Clock + 1 PHY Clock
= (2 clock sync + 1 clock memory address + 1 clock memory data from sync RAM) + (1 PHY Clock (next PHY clock MAC can sample the 2-clock FIFO output)

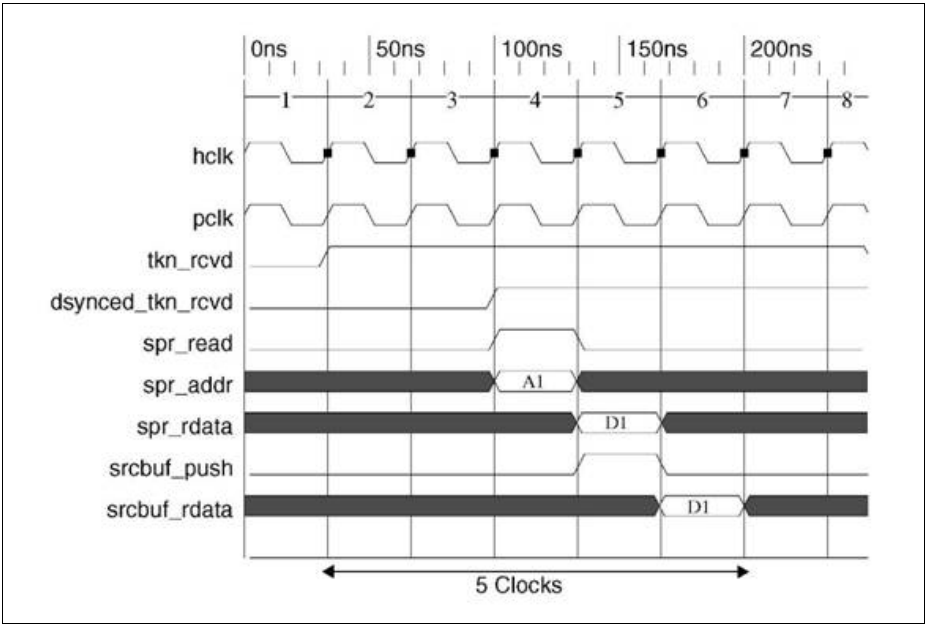


Figure 16-27 USBTrdTim Max Timing Case

Handling Babble Conditions

If USB core receives a packet that is larger than the maximum packet size for that endpoint, the core stops writing data to the Rx buffer and waits for the end of packet (EOP). When the core detects the EOP, it flushes the packet in the Rx buffer and does not send any response to the host.

If the core continues to receive data at the EOF2 (the end of frame 2, which is very close to SOF), the core generates an early_suspend interrupt (GINTSTS.ErlySusp). On receiving this interrupt, the application must check the erratic_error status bit (DSTS.ErrticErr). If this bit is set, the application must take it as a long babble and perform a soft reset.

Generic Non-Periodic (Bulk and Control) IN Data Transfers

This section describes a regular non-periodic IN data transfer.

Application Requirements

1. Before setting up an IN transfer, the application must ensure that all data to be transmitted as part of the IN transfer is part of a single buffer, and must program the size of that buffer and its start address (in DMA mode) to the endpoint-specific registers.
2. For IN transfers, the Transfer Size field in the Endpoint Transfer Size register denotes a payload that constitutes multiple maximum-packet-size packets and a single short packet. This short packet is transmitted at the end of the transfer.
   a) To transmit a few maximum-packet-size packets and a short packet at the end of the transfer:
   b) Transfer size[epnum] = n * mps[epnum] + sp
      (where n is an integer > 0, and 0 <sp < mps[epnum])
      - If (sp > 0), then packet count[epnum] = n + 1.
        Otherwise, packet count[epnum] = n
   c) To transmit a single zero-length data packet:
      - Transfer size[epnum] = 0
      - Packet count[epnum] = 1
   d) To transmit a few maximum-packet-size packets and a zero-length data packet at the end of the transfer, the application must split the transfer in two parts. The first sends maximum-packet- size data packets and the second sends the zero-length data packet alone.
      - First transfer: transfer size[epnum] = n * mps[epnum]; packet count = n;
      - Second transfer: transfer size[epnum] = 0; packet count = 1;
3. In DMA mode, the core fetches an IN data packet from the memory, always starting at a DWORD boundary. If the maximum packet size of the IN endpoint is not a multiple of 4, the application must arrange the data in the memory with pads inserted at the end of a maximum-packet-size packet so that a new packet always starts on a DWORD boundary.
4. Once an endpoint is enabled for data transfers, the core updates the Transfer Size register. At the end of IN transfer, which ended with an Endpoint Disabled interrupt, the application must read the Transfer Size register to determine how much data posted in the transmit FIFO was already sent on the USB.
5. Data fetched into transmit FIFO = Application-programmed initial transfer size - core-updated final transfer size
   a) Data transmitted on USB = (application-programmed initial packet count - Core updated final packet count) * mps[epnum]
   b) Data yet to be transmitted on USB = (Application-programmed initial transfer size - data transmitted on USB)

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the endpoint-specific registers and enable the endpoint to transmit the data.
2. In Slave mode, the application must also write the required data to the transmit FIFO for the endpoint. In DMA mode, the core fetches the data from memory according to the application setting for the endpoint.
3. Every time a packet is written into the transmit FIFO, either by the core's internal DMA (in DMA mode) or the application (in Slave Mode), the transfer size for that endpoint is decremented by the packet size. The data is fetched from the memory (DMA/ Application), until the transfer size for the endpoint becomes 0. After writing the data into the FIFO, the "number of packets in FIFO" count is incremented (this is a 3-bit count, internally maintained by the core for each IN endpoint transmit FIFO. The maximum number of packets maintained by the core at any time in an IN endpoint FIFO is eight). For zero-length packets, a separate flag is set for each FIFO, without any data in the FIFO.
4. Once the data is written to the transmit FIFO, the core reads it out upon receiving an IN token. For every non-isochronous IN data packet transmitted with an ACK handshake, the packet count for the endpoint is decremented by one, until the packet count is zero. The packet count is not decremented on a TIMEOUT.
5. For zero length packets (indicated by an internal zero length flag), the core sends out a zero-length packet for the IN token and decrements the Packet Count field.
6. If there is no data in the FIFO for a received IN token and the packet count field for that endpoint is zero, the core generates a IN Tkn Rcvd When FIFO Empty Interrupt for the endpoint, provided the endpoint NAK bit is not set. The core responds with a NAK handshake for non-isochronous endpoints on the USB.
7. In Dedicated FIFO operation, the core internally rewinds the FIFO pointers and no timeout interrupt is generated except for Control IN endpoint.
8. When the transfer size is 0 and the packet count is 0, the transfer complete interrupt for the endpoint is generated and the endpoint enable is cleared.

Application Programming Sequence

1. Program the DIEPTSIZx register with the transfer size and corresponding packet count. In DMA mode, also program the DIEPDMAx register.
2. Program the DIEPCTLx register with the endpoint characteristics and set the CNAK and Endpoint Enable bits.
   In DMA mode, ensure that the NextEp field is programmed so that the core fetches the data for IN endpoints in the correct order. See **“Non-Periodic IN Endpoint Sequencing” on Page 16-108** for details.
3. When using dedicated FIFO operation in slave mode, when transmitting non-zero length data packet, the application must poll the DTXFSTSx register (where n is the FIFO number associated with that endpoint) to determine whether there is enough

space in the data FIFO. The application can optionally use DIEPINTx.TxFEmp before writing the data.

Examples

Slave Mode Bulk IN Transaction

These notes refer to **Figure 16-28**.

1. The host attempts to read data (IN token) from an endpoint.
2. On receiving the IN token on the USB, the core returns a NAK handshake, because no data is available in the transmit FIFO.
3. To indicate to the application that there was no data to send, the core generates a DIEPINTx.IN Token Rcvd When TxFIFO Empty interrupt.
4. When data is ready, the application sets up the DIEPTSIZx register with the Transfer Size and Packet Count fields.
5. The application writes one maximum packet size or less of data to the Non-periodic TxFIFO.
6. The host reattempts the IN token.
7. Because data is now ready in the FIFO, the core now responds with the data and the host ACKs it.
8. Because the XferSize is now zero, the intended transfer is complete. The device core generates a DIEPINTx.XferCompl interrupt.
9. The application processes the interrupt and uses the setting of the DIEPINTx.XferCompl interrupt bit to determine that the intended transfer is complete.

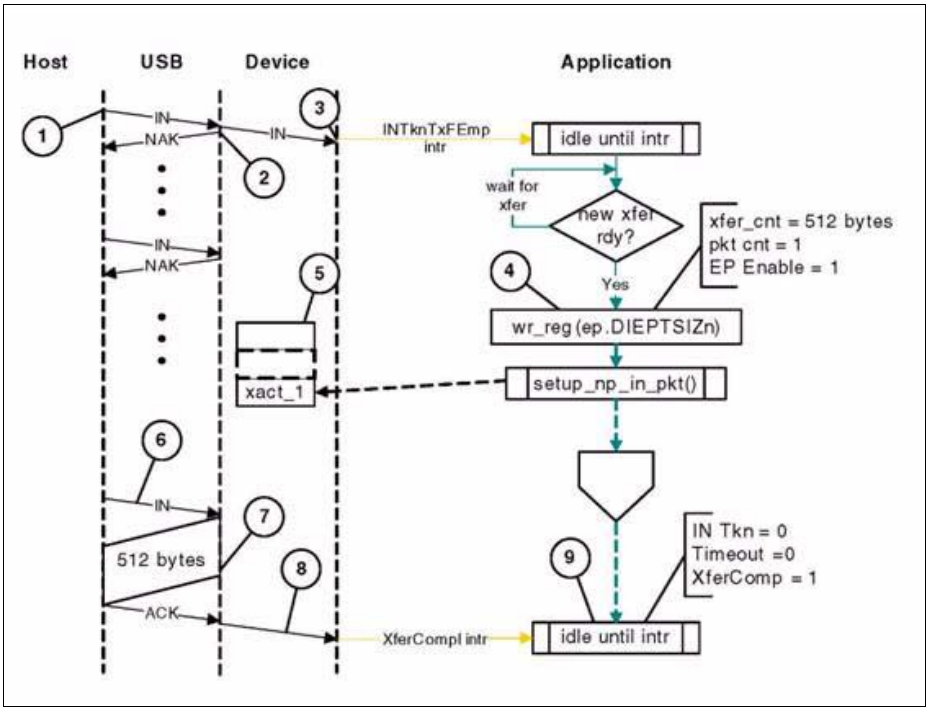


Figure 16-28 Slave Mode Bulk IN Transaction

Slave Mode Bulk IN Transfer (Pipelined Transaction)

These notes refer to **Figure 16-29**.

1. The host attempts to read data (IN token) from an endpoint.
2. On receiving the IN token on the USB, the core returns a NAK handshake, because no data is available in the transmit FIFO.
3. To indicate that there was no data to send, the core generates an DIEPINTx.InTkn Rcvd When TxFIFO Empty interrupt.
4. When data is ready, the application sets up the DIEPTSIZx register with the transfer size and packet count.
5. The application writes one maximum packet size or less of data to the Non-periodic TxFIFO.
6. The host reattempts the IN token.
7. Because data is now ready in the FIFO, the core responds with the data, and the host ACKs it.

8. When the TxFIFO level falls below the halfway mark, the core generates a GINTSTS.NonPeriodic TxFIFO Empty interrupt. This triggers the application to start writing additional data packets to the FIFO.
9. A data packet for the second transaction is ready in the TxFIFO.
10. A data packet for third transaction is ready in the TxFIFO while the data for the second packet is being sent on the bus.
11. The second data packet is sent to the host.
12. The last short packet is sent to the host.
13. Because the last packet is sent and XferSize is now zero, the intended transfer is complete. The core generates a DIEPINTx.XferCompl interrupt.
14. The application processes the interrupt and uses the setting of the DIEPINTx.XferCompl interrupt bit to determine that the intended transfer is complete

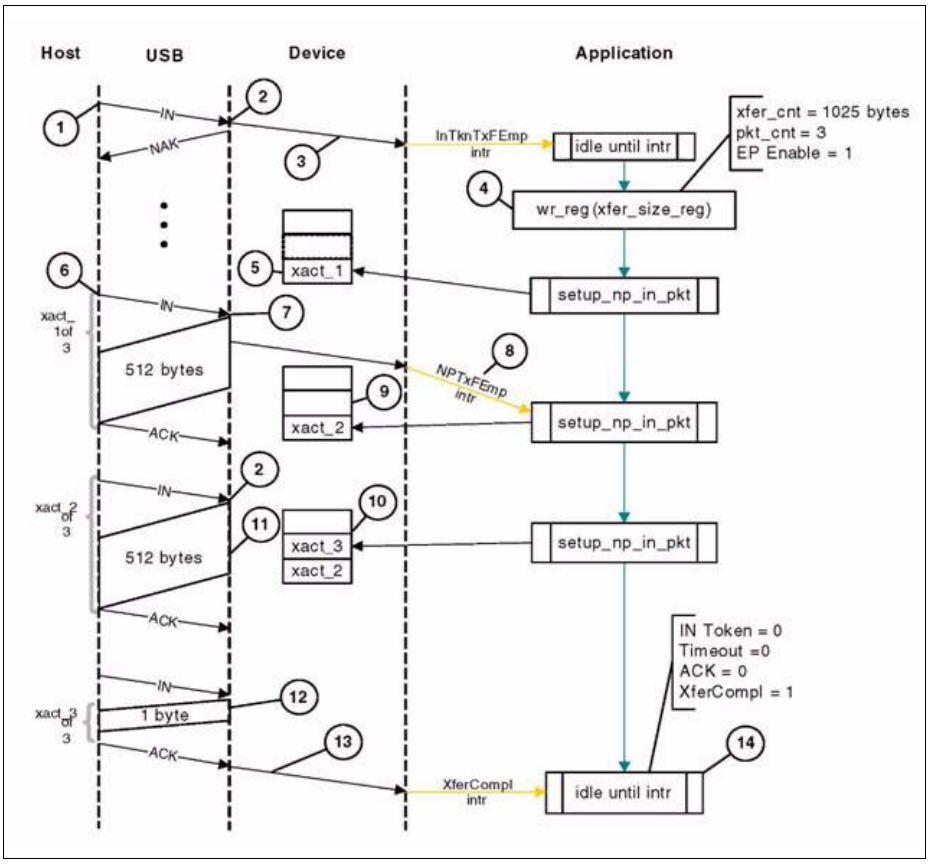


Figure 16-29 Slave Mode Bulk IN Transfer (Pipelined Transaction

Slave Mode Bulk IN Two-Endpoint Transfer

These notes refer to **Figure 16-30**.

1. The host attempts to read data (IN token) from endpoint 1.
2. On receiving the IN token on the USB, the core returns a NAK handshake, because no data is available in the transmit FIFO for endpoint 1, and generates a DIEPINTl.InTkn Rcvd When TxFIFO Empty interrupt.
3. The application processes the interrupt and initializes DIEPTSIZ1 register with the Transfer Size and Packet Count fields. The application starts writing the transaction data to the transmit FIFO.
4. The application writes one maximum packet size or less of data for endpoint 1 to the Non-periodic TxFIFO.
5. Meanwhile, the host attempts to read data (IN token) from endpoint 2.
6. On receiving the IN token on the USB, the core returns a NAK handshake, because no data is available in the transmit FIFO for endpoint 2, and the core generates a DIEPINT2.InTkn Rcvd When TxFIFO Empty interrupt.
7. Because the application has completed writing the packet for endpoint 1, it initializes the DIEPTSIZ2 register with the Transfer Size and Packet Count fields. The application starts writing the transaction data into the transmit FIFO for endpoint 2.
8. The host repeats its attempt to read data (IN token) from endpoint 1.
9. Because data is now ready in the TxFIFO, the core returns the data, which the host ACKs.
10. Meanwhile, the application has initialized the data for the next two packets in the TxFIFO (ep2.xact1 and ep1.xact2, in order).
11. The host repeats its attempt to read data (IN token) from endpoint 2.
12. Because endpoint 2's data is ready, the core responds with the data (ep2.xact_1), which the host ACKs.
13. Meanwhile, the application has initialized the data for the next two packets in the TxFIFO (ep2.xact2 and ep1.xact3, in order). The application has finished initializing data for the two endpoints involved in this scenario.
14. The host repeats its attempt to read data (IN token) from endpoint 1.
15. Because data is now ready in the FIFO, the core responds with the data, which the host ACKs.
16. The host repeats its attempt to read data (IN token) from endpoint 2.
17. With data now ready in the FIFO, the core responds with the data, which the host ACKs.
18. With the last packet for endpoint 2 sent and its XferSize now zero, the intended transfer is complete. The core generates a DIEPINT2.XferCompl interrupt for this endpoint.
19. The application processes the interrupt and uses the setting of the DIEPINT2.XferCompl interrupt bit to determine that the intended transfer on endpoint 2 is complete.
20. The host repeats its attempt to read data (IN token) from endpoint 1 (last transaction).

21. With data now ready in the FIFO, the core responds with the data, which the host ACKs.
22. Because the last endpoint one packet has been sent and XferSize is now zero, the intended transfer is complete. The core generates a DIEPINT1.XferCompl interrupt for this endpoint.
23. The application processes the interrupt and uses the setting of the DIEPINT1.XferCompl interrupt bit to determine that the intended transfer on endpoint 1 is complete.

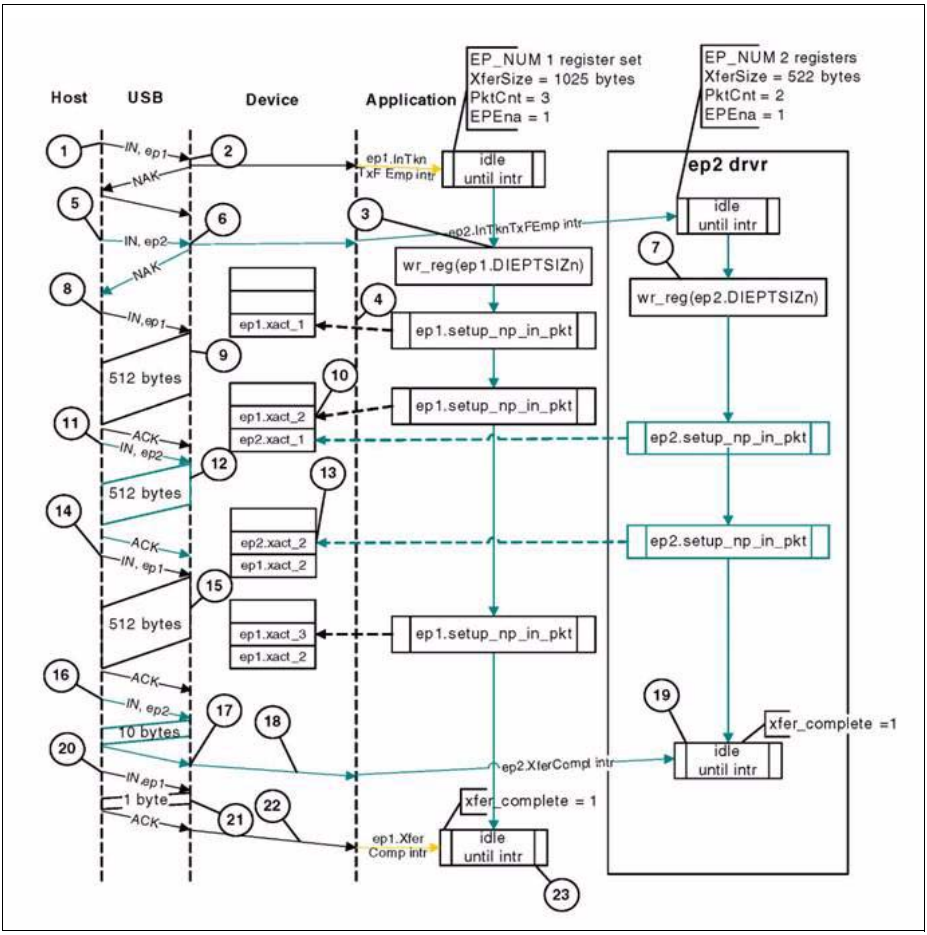


Figure 16-30 Slave Mode Bulk IN Two-Endpoint Transfer

Generic Periodic IN (Interrupt and Isochronous) Data Transfers

This section describes a typical periodic IN data transfer.

Application Requirements

1. Application requirements 1, 2, 3, and 4 of **"Generic Non-Periodic (Bulk and Control) IN Data Transfers" on Page 16-111** also apply to periodic IN data transfers, except for a slight modification of Requirement 2.
   a) The application can only transmit multiples of maximum-packet-size data packets or multiples of maximum-packet-size packets, plus a short packet at the end. To transmit a few maximum- packet-size packets and a short packet at the end of the transfer, the following conditions must be met.
      - transfer size[epnum] = n * mps[epnum] + sp
        (where n is an integer > 0, and 0 <sp < mps[epnum])
      - If (sp > 0), packet count[epnum] = n + 1
        Otherwise, packet count[epnum] = n;
      - mc[epnum] = packet count[epnum]

   b) The application cannot transmit a zero-length data packet at the end of transfer. It can transmit a single zero-length data packet by it self. To transmit a single zero-length data packet,
   c) transfer size[epnum] = 0
      - packet count[epnum] = 1
      - mc[epnum] = packet count[epnum]
2. The application can only schedule data transfers 1 frame at a time.
   a) (DIEPTSIZx.MC - 1) * DIEPCTLx.MPS <DIEPTSIZx.XferSiz <DIEPTSIZx.MC * DIEPCTLx.MPS
   b) DIEPTSIZx. PktCnt = DIEPTSIZx.MC
   c) If DIEPTSIZx.XferSiz < DIEPTSIZx.MC * DIEPCTLx.MPS, the last data packet of the transfer is a short packet.
3. This step is not applicable for isochronous data transfers, only for interrupt transfers. The application can schedule data transfers for multiple frames, only if multiples of max packet sizes (up to 3 packets), must be transmitted every frame. This is can be done, only when the core is operating in DMA mode. This is not a recommended mode though.
   a) ((n*DIEPTSIZx.MC) - 1)*DIEPCTLx.MPS <= DIEPTSIZx.Transfer Size <= n*DIEPTSIZx.MC*DIEPCTLx.MPS
   b) DIEPTSIZx.Packet Count = n*DIEPTSIZx.MC
   c) n is the number of frames for which the data transfers are scheduled

   Data Transmitted per frame in this case would be DIEPTSIZx.MC*DIEPCTLx.MPS, in all the frames except the last one. In the frame "n", the data transmitted would be (DIEPTSIZx.TransferSize - (n-1)*DIEPTSIZx.MC*DIEPCTLx.MPS)

4. For Periodic IN endpoints, the data must always be prefetched 1 frame ahead for transmission in the next frame. This can be done, by enabling the Periodic IN endpoint 1 frame ahead of the frame in which the data transfer is scheduled.
5. The complete data to be transmitted in the frame must be written into the transmit FIFO (either by the application or the DMA), before the Periodic IN token is received. Even when 1 DWORD of the data to be transmitted per frame is missing in the transmit FIFO when the Periodic IN token is received, the core behaves as when the FIFO was empty. When the transmit FIFO is empty,
6. A zero data length packet would be transmitted on the USB for ISO IN endpoints
   a) A NAK handshake would be transmitted on the USB for INTR IN endpoints
7. For a High Bandwidth IN endpoint with three packets in a frame, the application can program the endpoint FIFO size to be 2*max_pkt_size and have the third packet load in after the first packet has been transmitted on the USB.

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the endpoint-specific registers and enable the endpoint to transmit the data.
2. In Slave mode, the application must also write the required data to the associated transmit FIFO for the endpoint. In DMA mode, the core fetches the data for the endpoint from memory, according to the application setting.
3. Every time either the core's internal DMA (in DMA mode) or the application (in Slave mode) writes a packet to the transmit FIFO, the transfer size for that endpoint is decremented by the packet size. The data is fetched from DMA or application memory until the transfer size for the endpoint becomes 0.
4. When an IN token is received for an periodic endpoint, the core transmits the data in the FIFO, if available. If the complete data payload (complete packet, in dedicated FIFO mode) for the frame is not present in the FIFO, then the core generates an IN Tkn Rcvd When TxF Empty Interrupt for the endpoint.
   a) A zero-length data packet is transmitted on the USB for isochronous IN endpoints
   b) A NAK handshake is transmitted on the USB for interrupt IN endpoints
5. The packet count for the endpoint is decremented by 1 under the following conditions:
   a) For isochronous endpoints, when a zero- or non-zero-length data packet is transmitted
   b) For interrupt endpoints, when an ACK handshake is transmitted
   c) When the transfer size and packet count are both 0, the Transfer Completed interrupt for the endpoint is generated and the endpoint enable is cleared.
6. At the "Periodic frame Interval" (controlled by DCFG.PerFrint), when the core finds non-empty any of the isochronous IN endpoint FIFOs scheduled for the current frame non-empty, the core generates a GINTSTS.incompISOIN interrupt.

Application Programming Sequence (Transfer Per Frame)

1. Program the DIEPTSIZx register. In DMA mode, also program the DIEPDMAx register.
2. Program the DIEPCTLx register with the endpoint characteristics and set the CNAK and Endpoint Enable bits.
3. In Slave mode, write the data to be transmitted in the next frame to the transmit FIFO as described in **“Packet Write in Slave Mode” on Page 16-102**.
4. Asserting the DIEPINTx.In Token Rcvd When TxF Empty interrupt indicates that either the DMA or application has not yet written all data to be transmitted to the transmit FIFO.
5. If the interrupt endpoint is already enabled when this interrupt is detected, ignore the interrupt. If it is not enabled, enable the endpoint so that the data can be transmitted on the next IN token attempt.
   a) If the isochronous endpoint is already enabled when this interrupt is detected, see **“Incomplete Isochronous IN Data Transfers” on Page 16-107** for more details.
6. The core handles timeouts internally, without application intervention.
7. Asserting the DIEPINTx.XferCompl interrupt with no DIEPINTx.In Tkn Rcvd When TxF Empty interrupt indicates the successful completion of an isochronous IN transfer. A read to the DIEPTSIZx register must indicate transfer size = 0 and packet count = 0, indicating all data is transmitted on the USB.
8. Asserting the DIEPINTx.XferCompl interrupt, with or without the DIEPINTx.In Tkn Rcvd When TxF Empty interrupt, indicates the successful completion of an interrupt IN transfer. A read to the DIEPTSIZx register must indicate transfer size = 0 and packet count = 0, indicating all data is transmitted on the USB.
9. Asserting the GINTSTS.incomplete Isochronous IN Transfer interrupt with none of the aforementioned interrupts indicates the core did not receive at least 1 periodic IN token in the current frame.
10. For isochronous IN endpoints, see **“Incomplete Isochronous IN Data Transfers” on Page 16-107**, for more details.

Generic Periodic IN Data Transfers Using the Periodic Transfer Interrupt Feature

This section describes a typical Periodic IN (ISOC / INTR) data transfer with the Periodic Transfer Interrupt feature.

1. Before setting up an IN transfer, the application must ensure that all data to be transmitted as part of the IN transfer is part of a single buffer, and must program the size of that buffer and its start address (in DMA mode) to the endpoint-specific registers.
   a) The application must mask the GINTSTS.incompISOIN.
2. For IN transfers, the Transfer Size field in the Endpoint Transfer Size register denotes a payload that constitutes multiple maximum-packet-size packets and a single short packet. This short packet is transmitted at the end of the transfer.

a) To transmit a few maximum-packet-size packets and a short packet at the end of the transfer:
- Transfer size[epnum] = n * mps[epnum] + sp
(where n is an integer > 0, and 0 < sp < mps[epnum]. A higher value of n reduces the periodicity of the DOEPINTx.XferCompl interrupt)
- If (sp > 0), then packet count[epnum] = n + 1. Otherwise, packet count[epnum] = n

b) To transmit a single zero-length data packet:
- Transfer size[epnum] = 0
- Packet count[epnum] = 1

c) To transmit a few maximum-packet-size packets and a zero-length data packet at the end of the transfer, the application must split the transfer in two parts. The first sends maximum-packet- size data packets and the second sends the zero-length data packet alone.
- First transfer: transfer size[epnum] = n * mps[epnum]; packet count = n;
- Second transfer: transfer size[epnum] = 0; packet count = 1;

d) The application can only transmit multiples of maximum-packet-size data packets or multiples of maximum-packet-size packets, plus a short packet at the end. To transmit a few maximum- packet-size packets and a short packet at the end of the transfer, the following conditions must be met.
- transfer size[epnum] = n * mps[epnum] + sp (where n is an integer > 0, and 0 < sp < mps[epnum])
- If (sp > 0), packet count[epnum] = n + 1 Otherwise, packet count[epnum] = n;
- mc[epnum] = number of packets to be sent out in a frame.

e) The application cannot transmit a zero-length data packet at the end of transfer. It can transmit a single zero-length data packet by itself. To transmit a single zero-length data packet,
- transfer size[epnum] = 0
- packet count[epnum] = 1
- mc[epnum] = packet count[epnum]

3. In DMA mode, the core fetches an IN data packet from the memory, always starting at a DWORD boundary. If the maximum packet size of the IN endpoint is not a multiple of 4, the application must arrange the data in the memory with pads inserted at the end of a maximum-packet-size packet so that a new packet always starts on a DWORD boundary.
4. Once an endpoint is enabled for data transfers, the core updates the Transfer Size register. At the end of IN transfer, which ended with an Endpoint Disabled interrupt, the application must read the Transfer Size register to determine how much data posted in the transmit FIFO was already sent on the USB.
   a) Data fetched into transmit FIFO = Application-programmed initial transfer size - core-updated final transfer size
   b) Data transmitted on USB = (application-programmed initial packet count - Core updated final packet count) * mps[epnum]

c) Data yet to be transmitted on USB = (Application-programmed initial transfer size - data transmitted on USB)

5. The application can schedule data transfers for multiple frames, only if multiples of max packet sizes (up to 3 packets), must be transmitted every frame. This is can be done, only when the core is operating in DMA mode.
   a) ((n*DIEPTSIZx.MC) - 1)*DIEPCTLx.MPS <= DIEPTSIZx.Transfer Size <= n*DIEPTSIZx.MC*DIEPCTLx.MPS
   b) DIEPTSIZx.Packet Count = n*DIEPTSIZx.MC
   c) n is the number of frames for which the data transfers are scheduled. Data Transmitted per frame in this case is DIEPTSIZx.MC*DIEPCTLx.MPS in all frames except the last one. In frame n, the data transmitted is (DIEPTSIZx.TransferSize - (n - 1) * DIEPTSIZx.MC * DIEPCTLx.MPS)
6. For Periodic IN endpoints, the data must always be prefetched 1 frame ahead for transmission in the next frame. This can be done, by enabling the Periodic IN endpoint 1 frame ahead of the frame in which the data transfer is scheduled.
7. The complete data to be transmitted in the frame must be written into the transmit FIFO, before the Periodic IN token is received. Even when 1 DWORD of the data to be transmitted per frame is missing in the transmit FIFO when the Periodic IN token is received, the core behaves as when the FIFO was empty. When the transmit FIFO is empty,
   a) A zero data length packet would be transmitted on the USB for ISOC IN endpoints
   b) A NAK handshake would be transmitted on the USB for INTR IN endpoints
   c) DIEPTSIZx.PktCnt is not decremented in this case.
8. For a High Bandwidth IN endpoint with three packets in a frame, the application can program the endpoint FIFO size to be 2 * max_pkt_size and have the third packet load in after the first packet has been transmitted on the USB.

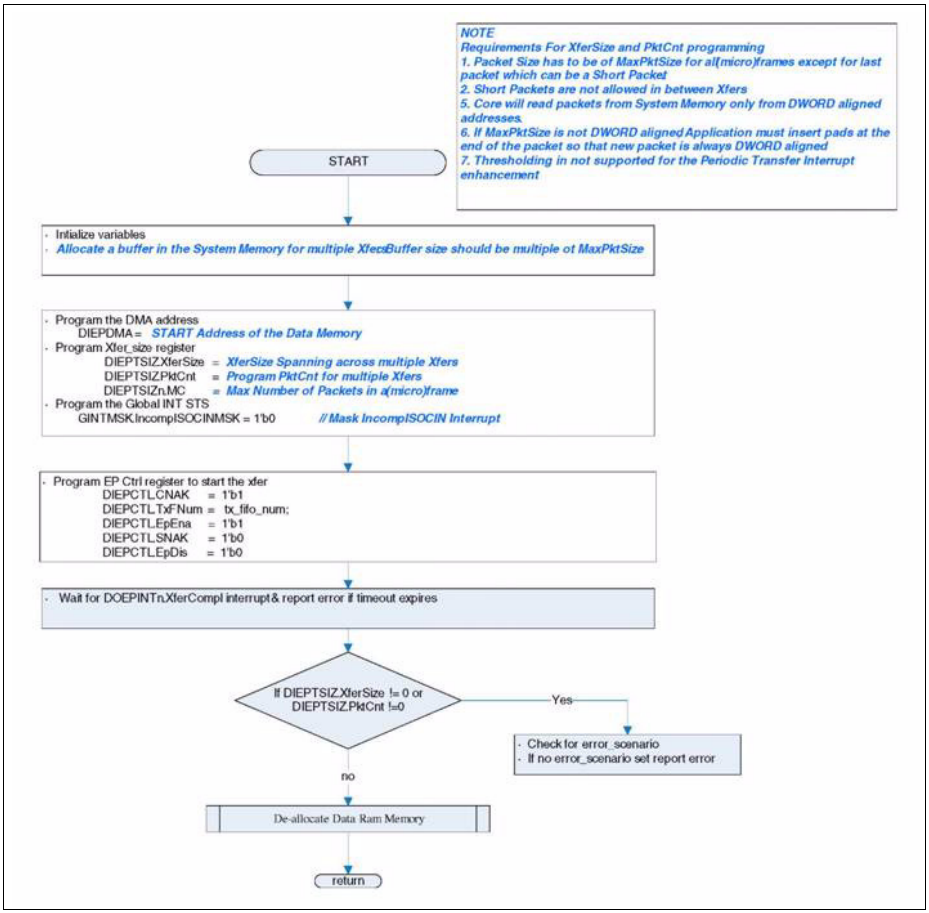


Figure 16-31 Periodic IN Application Flow for Periodic Transfer Interrupt Feature

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the endpoint-specific registers and enable the endpoint to transmit the data.
   a) The application must mask the GINTSTS.incompISOOUT.
   b) The application must enable the DCTL.IgnrFrmNum
2. When an isochronous OUT endpoint is enabled by setting the Endpoint Enable and clearing the NAK bits, the Even/Odd frame will be ignored by the core.
   a) Subsequently the core updates the Even / Odd bit on its own
3. Every time either the core's internal DMA writes a packet to the transmit FIFO, the transfer size for that endpoint is decremented by the packet size. The data is fetched from DMA or application memory until the transfer size for the endpoint becomes 0.
4. When an IN token is received for a periodic endpoint, the core transmits the data in the FIFO, if available. If the complete packet for the frame is not present in the FIFO, then the core generates an IN Tkn Rcvd When TxFifo Empty Interrupt for the endpoint.
   a) A zero-length data packet is transmitted on the USB for isochronous IN endpoints
   b) A NAK handshake is transmitted on the USB for interrupt IN endpoints
5. If an IN token comes for an endpoint on the bus, and if the corresponding TxFIFO for that endpoint has at least 1 packet available, and if the DIEPCTLx.NAK bit is not set, and if the internally maintained even/odd bit match with the bit 0 of the current frame number, then the core will send this data out on the USB. The core will also decrement the packet count. Core also toggles the MultCount in DIEPCTLx register and based on the value of MultCount the next PID value is sent.
   a) If the IN token results in a timeout (core did not receive the handshake or handshake error), core rewind the FIFO pointers. Core does not decrement packet count. It does not toggle PID. DIEPINTx.TimeOUt interrupt will be set which the application could check.
   b) At the end of periodic frame interval (Based on the value programmed in the DCFG.PerFrint register, core will internally set the even/ odd internal bit to match the next frame.
6. The packet count for the endpoint is decremented by 1 under the following conditions:
   a) For isochronous endpoints, when a zero- or non-zero-length data packet is transmitted
   b) For interrupt endpoints, when an ACK handshake is transmitted
7. The data PID of the transmitted data packet is based on the value of DIEPTSIZx.MC programmed by the application. In case the DIEPTSIZx.MC value is set to 3 then, for a particular frame the core expects to receive 3 Isochronous IN token for the respective endpoint. The data PIDs transmitted will be D2 followed by D1 and D0 respectively for the tokens.
   a) If any of the tokens responded with a zero-length packet due to non-availability of data in the TxFIFO, the packet is sent in the next frame with the pending data PID.

For example, in a frame, the first received token is responded to with data and data PID value D2. If the second token is responded to with a zero-length packet, the host is expected not to send any more tokens for the respective endpoint in the current frame. When a token arrives in the next frame it will be responded to with the pending data PID value of D1.

b) Similarly the second token of the current frame gets responded with D0 PID. The host is expected to send only two tokens for this frame as the first token got responded with D1 PID.

8. When the transfer size and packet count are both 0, the Transfer Completed interrupt for the endpoint is generated and the endpoint enable is cleared.
9. The GINTSTS.incompISOIN will be masked by the application hence at the Periodic Frame interval (controlled by DCFG.PerFrint), even though the core finds non-empty any of the isochronous IN endpoint FIFOs, GINTSTS.incompISOIN interrupt will not be generated.

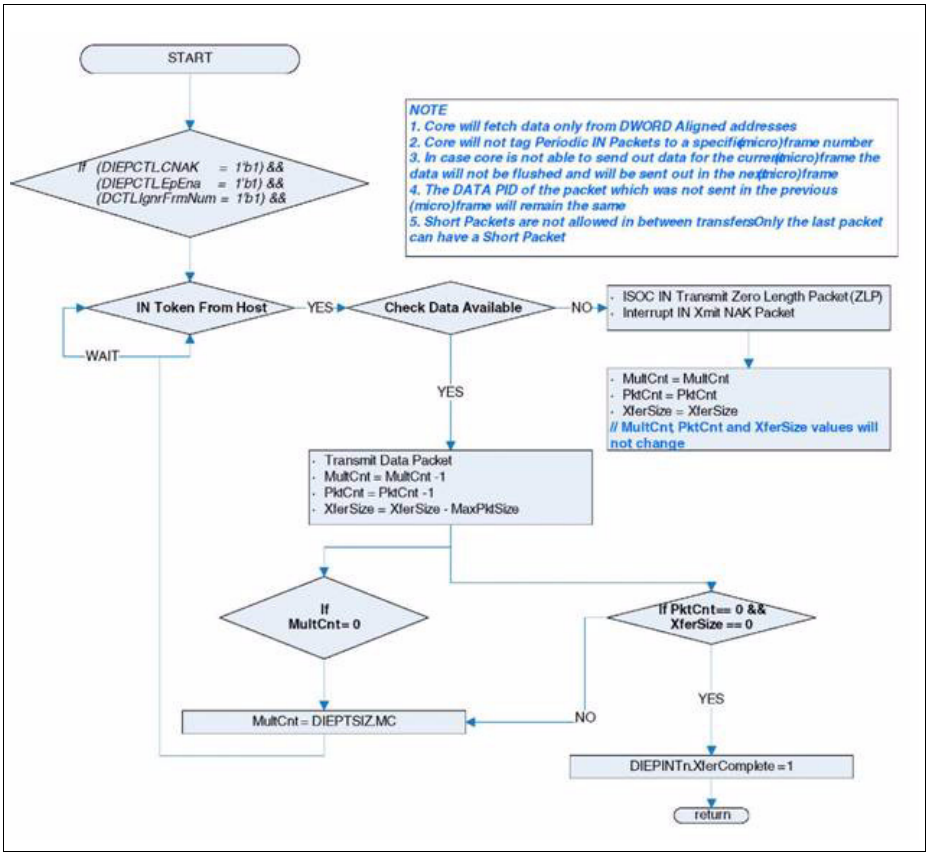


Figure 16-32 Periodic IN Core Internal Flow for Periodic Transfer Interrupt Feature

16.7 Device Scatter-Gather DMA Mode

16.7.1 Scatter/Gather DMA Mode

When the Scatter/Gather DMA mode is enabled data buffers are presented through descriptor structures

1. The application prepares the descriptors, and sets the bit DIEPCTLx/DOEPCTLx.EPEna.
2. DMA fetches the corresponding descriptor (initially determined by DIEPDMAx/DOEPDMAx).

3. DMA internally sets the transfer size from descriptor back to DIEPTSIZx/DOEPTSIZx.

Note: The registers DIEPTSIZx/DOEPTSIZx must not be written by the application in Scatter/Gather DMA mode.

From this point, the current OTG flow executes.

Once the transfer size data is moved by DMA, the DMA checks for further links in the descriptor chain.

If this is the last descriptor, the DMA sets the DIOEPINTn.XferCompl interrupt. If there are further active links, the DMA continues to process them.

In Scatter/Gather DMA mode, the core implements a true scatter-gather memory distribution in which data buffers are scattered over the system memory. Each endpoint memory structure is implemented as a contiguous list of descriptors, in which each descriptor points to a data buffer of predefined size. In addition to the buffer pointer (1 DWORD), the descriptor also has a status quadlet (1 DWORD). When the list is implemented as a ring buffer, the list processor switches to the first element of the list when it encounters last bit. All endpoints (control, bulk, interrupt, and isochronous) implement these structures in memory.

Note: The descriptors are stored in continuos locations. For example descriptor 1 is stored in 0000_0000_H, descriptor 2 is stored in 0000_0008_H, descriptor 3 in 0000_0010_H and so on. The descriptors are always DWORD aligned.

16.7.2 SPRAM Requirements

For each endpoint the current descriptor pointer and descriptor status are cached to avoid additional requests to system memory. These are stored in SPRAM. In addition DIEPDMAx/DOEPDMAx registers are also implemented in SPRAM.

16.7.3 Descriptor Memory Structures

The descriptor memory structures are displayed in **Figure 16-33**.

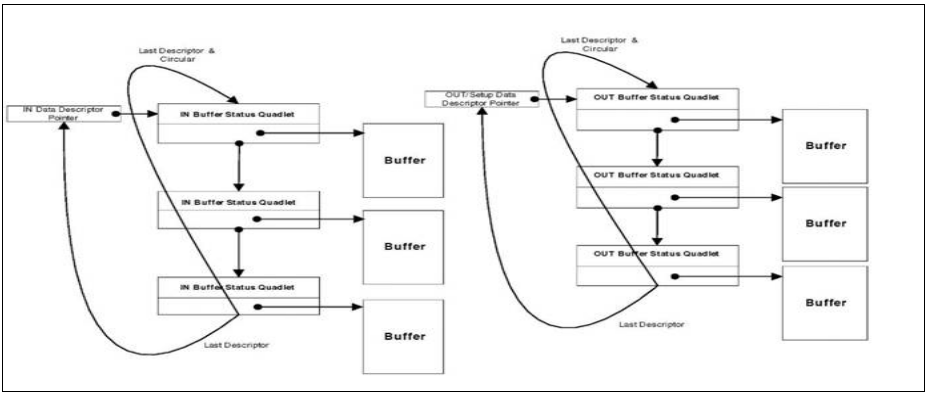


Figure 16-33 Descriptor Memory Structures

16.7.3.1 OUT Data Memory Structure

All endpoints that support OUT direction transactions (endpoints that receive data from the USB host), must implement a memory structure with the following characteristics:

- Each data buffer must have a descriptor associated with it to provide the status of the buffer. The buffer itself contains only raw data.
- Each buffer descriptor is two quadlets in length.

When the buffer status of the first descriptor is host Ready, the DMA fetches and processes its data buffer; otherwise the DMA optionally skips to the next descriptor until it reaches the end of the descriptor chain. The buffers to which the descriptor points hold packet data for non-isochronous endpoints and frame (FS)/µframe (FS) data for isochronous endpoints.

Host Ready — indicates that the descriptor is available for the DMA to process.

DMA Busy — indicates that the DMA is still processing the descriptor.

DMA Done—indicates that the buffer data transfer is complete.

Host Busy—indicates that the application is processing the descriptor.

The OUT data memory structure is shown in **Figure 16-34**, which shows the definition of status quadlet bits for non-ISO and ISO end points

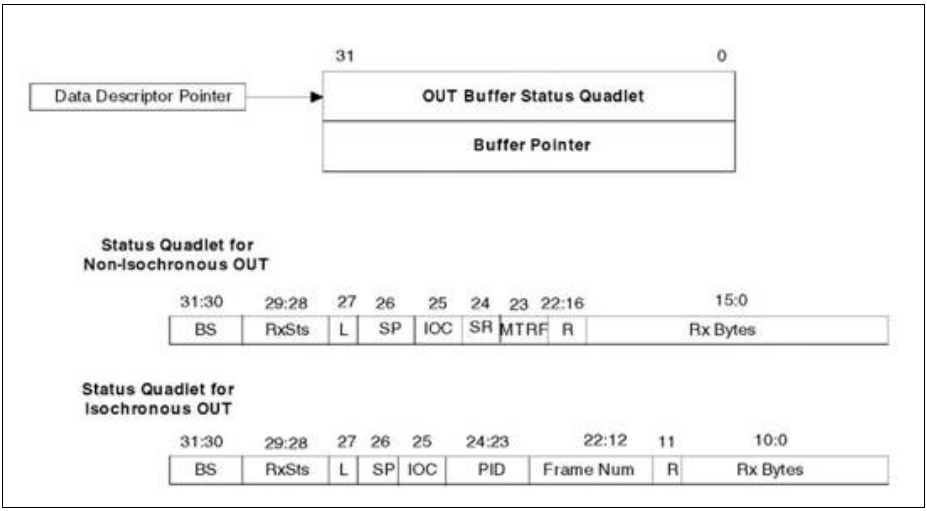


Figure 16-34 Out Data Memory Structure

The status quadlet interpretation depends on the end point type field (DOEPCTLx.EPType) for the corresponding end point. For example, if an end point is OUT and periodic, then the status quadlet is interpreted as Status Quadlet for Isochronous OUT.

Table 16-8 displays the OUT Data Memory Structure fields.

Note: Note that some fields change depending on the mode.

Table 16-8 OUT Data Memory Structure Values

Bit	Bit ID	Description
BS [31:30]	Buffer Status	This 2-bit value describes data buffer status. Possible options are: 00_B Host Ready 01_B DMA Busy 10_B DMA Done 11_BHost Busy Application sets to Host Ready if the descriptor is ready or to Host Busy if the descriptor is not ready. Core sets to DMA busy if the descriptor is being serviced or to DMA Done if the transfer finished associated with the descriptor. The application needs to make these bits as 00_B (Host Ready) as a last step after preparing the entire descriptor ready.Once the software makes these bits as Host Ready then it must not alter the descriptor until DMA completes
Rx Sts [29:28]	Receive Statu	This 2-bit value describes the status of the received data. Core updates this when the descriptor is closed. This reflects whether OUT data has been received correctly or with errors. BUFERR is set by the core when AHB error is encountered during buffer access. BUFERR is set by the core after asserting AHBErr for the corresponding end point.The possible combinations are: • 00_B Success, No AHB errors • 01_B Reserved • 10_B Reserved • 11_B BUFERR
L [27]	Last	Set by the application, this bit indicates that this descriptor is the last one in the chain. Note - L Bit is interpreted by the core even when BS value is other than Host ready. For example, BNA is set, the core keeps traversing all the descriptors until it encounters a descriptor whose L bit is set after which the core disables the corresponding endpoint.
SP[26]	Short Packet	Set by the Core, this bit indicates that this descriptor closed after short packet. When reset it indicates that the descriptor is closed after requested amount of data is received.

Table 16-8 OUT Data Memory Structure Values (cont'd)

Bit	Bit ID	Description	
IOC[25]	Interrupt On complete	Set by the application, this bit indicates that the core must generate a transfer complete interrupt(XferCompl) after this descriptor is finished.	
[24][1)]	Varies	**Non Isochronous Out** **Bit: SR[24]** **Bit ID:** Setup Packet Received Set by the Core, this bit indicates that this buffer holds 8 bytes of setup data. There is only one setup packet per descriptor. On reception of a setup packet, the descriptor is closed and the corresponding endpoint is disabled after SETUP_COMPLETE status is seen in the Rx fifo. The core puts a SETUP_COMPLETE status into the Rx FIFO when it sees the first IN/OUT token after the SETUP packet for that particular endpoint. However, if the L bit of the descriptor is set, the endpoint is disabled and the descriptor is closed irrespective of the SETUP_COMPLETE status. The application has to re-enable for receiving any OUT data for the control transfer. (It also need to reprogram the descriptor start address) Note - Because of the above behavior, the core can receive any number of back to back setup packets and one descriptor for every setup packet is used.	**Isochronous Out** **Bit: Reserved [24:23]** **Bit ID:** This field is reserved and the core writes 00_B.

Table 16-8 OUT Data Memory Structure Values (cont'd)

Bit	Bit ID	Description	
[23][1)]	Varies	**Non Isochronous Out** **Bit: MTRF[23]** **Bit ID:** Multiple Transfer Set by the application, this bit indicates the Core can continue processing the list after it encountered last descriptor.This is to support multiple transfers without application intervention. Reserved for ISO OUT and Control OUT endpoints.	See description for bit [24]
[22:16][1)]	Varies	**Non Isochronous Out** **Bit:** [22:12] **Bit ID:** R Reserved	**Isochronous Out** **Bit: Frame Number [22:12]** **Bit ID:** Frame number The 11-bit frame number corresponds to full speed frame number.

Table 16-8 OUT Data Memory Structure Values (cont'd)

Bit	Bit ID	Description	
[15:12][1)]	Varies	**Non Isochronous Out** **Bit:** Rx Bytes [15:0] **Bit ID:** Received number of bytes remaining This 16-bit value can take values from 0 to (64K-1) bytes, depending on the transfer size of data received from the USB host. The application programs the expected transfer size. When the descriptor is done this indicates remainder of the transfer size. Here, Rx Bytes must be in terms of multiple of MPS for the corresponding end point. The MPS for the various packet types are as follows: • Control – LS - 8 bytes – FS - 8,16,32,64 bytes • Bulk – FS - 8,16,32,64 bytes • Interrupt – LS - up to 8 bytes – FS - up to 64 bytes **Note**: In case of Interrupt packets, the MPS may not be a multiple of 4. If the MPS in an interrupt packet is not a multiple of 4, then a single interrupt packet corresponds to a single descriptor. If MPS is a multiple of 4 for an interrupt packet, then a single descriptor can have multiple MPS packets.	See description for bits [22:16]
[11][1)]	Varies		**Isochronous Out** **Bit:** 11 **Bit ID:** Reserved
[10:0][1)]	Varies		Isochronous Out Bit: Rx Bytes [10:0] Bit ID: Received number of bytes This 11 -bit value can take values from 0 to (2K-1) bytes, depending on the packet size of data received from the USB host.Application programs the expected transfer size. When the descriptor is done this indicates remainder of the transfer size. The maximum payload size of each ISO packet as per USB specification 2.0 is as follows. FS - up to 1023 bytes **Note**: A Value of 0 indicates zero bytes of data, 1 indicates 1 byte of data and so on.

1) The meaning of this field varies. See description.

Table 16-9 displays the matrix of L bit and MTRF bit options.

Table 16-9 OUT - L Bit and MTRF Bit

L Bit	MTRF bit	Functionality
1	1	Continue to process the list after the last descriptor encountered. Use DOEPDMAx as next descriptor. The Endpoint is not disabled.
1	0	For non-Isochronous endpoints, Stop processing list after last descriptor encountered. The application intervenes and programs the list pointer into DOEPDMAx register when a list is created in a new location otherwise enables the endpoint. Start processing when the endpoint is enabled again with DOEPDMAx register pointing to start of list. For Isochronous endpoints, the DMA engine always goes back to the base descriptor address after the last descriptor.
0	1	If a short packet is received or expected transfer is done, close the current descriptor, continue with the next descriptor. If a short packet or Zero length packet is received, the corresponding endpoint is not disabled.
0	0	After processing the current descriptor go to next descriptor. If a short packet OR zero length packet is received disable the endpoint and a transfer complete interrupt is generated irrespective of IOC bit setting for that descriptor.

Table 16-10 displays the out buffer pointer field description.

Note: For Bulk and Interrupt End Points, if MTRF bit is set for the last descriptor in a list, then all the descriptors in that list need to have their MTRF bit set.

Table 16-10 OUT Buffer Pointer

Buf Addr[31:0]	Buffer Address	The Buffer pointer field in the descriptor is 32 bits wide and contains the address where the received data is to be stored in the system memory. The starting buffer address must be DWORD aligned. The buffer size must be also DWORD aligned.

16.7.3.2 Isochronous OUT

- The application must create one descriptor per packet.
- End point is not disabled by the core based on L bit. The DMA always goes back to the base descriptor address after the last descriptor.
- The bit MTRF is not applicable.

16.7.3.3 Non-Isochronous OUT

- The core uses one descriptor per setup packet.
- The core closes the descriptor after receiving a short packet.
- Bit combinations for L and MTRF appear in **Table 16-9**.
- Multiple Interrupt packets in the same buffer is allowed only if the MPS is multiple of 4.

16.7.3.4 IN Data Memory Structure

All endpoints that support IN direction transactions (transmitting data to the USB host) must implement the following memory structure. Each buffer must have a descriptor associated with it. The application fills the data buffer, updates its status in the descriptor, and enables the endpoint. The DMA fetches this descriptor and processes it, moving on in this fashion until it reaches the end of the descriptor chain. The buffer to which the descriptor points to hold packet data for non-isochronous endpoints and frame data for isochronous endpoints.

The definition of status quadlet bits for non-periodic and periodic end points are as shown in the figure. The status quadlet interpretation depends on the end point type field (DIEPCTLx.EPType) for the corresponding end point. For example, if an end point is IN and periodic, then the status quadlet is interpreted as "Status Quadlet for Isochronous IN".

The IN data memory structure is shown in **Figure 16-35**.

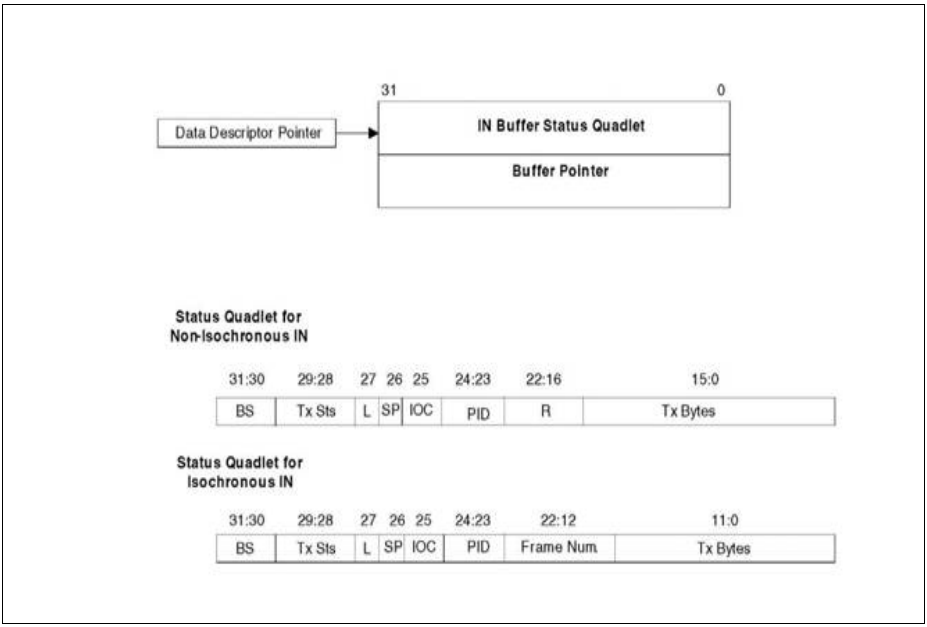


Figure 16-35 IN Data Memory Structure

Table 16-11 displays the IN Data Memory Structure fields.

Note: Some fields change depending on the mode

Table 16-11 IN Data Memory Structure Values

Bit	Bit ID	Description
BS [31:30]	Buffer Status	This 2-bit value describes the status of the data buffer. The possible options are: • 00_B Host ready • 01_B DMA busy • 10_B DMA done • 11_B Host busy The application needs to make these bits as 00_B (Host Ready) as a last step after preparing the entire descriptor ready.Once the software makes these bits as HostReady then it must not alter the descriptor until DMA done
Tx Sts [29:28]	Transmit Status	The status of the transmitted data. This reflects if the IN data has been transmitted correctly or with errors. BUFERR is set by core when there is a AHB error during buffer access. When iIgnrFrmNum is not set, BUFFLUSH is set by the core when • the core is fetching data pertaining to the current frame (N) and finds that the frame has incremented (N+1) during the data fetch • or • when it fetches a descriptor for which the frame number has already elapsed. The possible combinations are: • 00_B Success, No AHB errors • 01_B BUFFLUSH • 10_B Reserved • 11_B BUFERR
L [27]	Last	When set by the application, this bit indicates that this descriptor is the last one in the chain.
SP[26]	Short Packet	When set, this bit indicates that this descriptor points to a short packet or a zero length packet. If there is more than one packet in the descriptor, it indicates that the last packet is a short packet or a zero length packet.
IOC[25]	Interrupt On complete	When set by the application, this bit indicates that the core must generate a transfer complete interrupt after this descriptor is finished.

Table 16-11 IN Data Memory Structure Values (cont'd)

Bit	Bit ID	Description	
[24:23][1)]	Varies	**Non Isochronous In** **Bit:** Reserved[24:16] **Bit ID:** Reserved	**Isochronous In** **Bit:** Reserved[24:23] **Bit ID:** Reserved
[22:12][1)]	Varies		**Isochronous In** **Bit:** Frame Number [22:12] **Bit ID:** This field must correspond to the 11-bit full speed frame number.
[15:12][1)]	Varies	**Non Isochronous In** **Bit:** Tx bytes [15:0] **Bit ID:** Number of bytes to be transmitted This 16-bit value can take values from 0 to (64K-1) bytes, indicating the number of bytes of data to be transmitted to the USB host. **Note**: A Value of 0 indicates zero bytes of data, 1 indicates 1 byte of data and so on.	
[11:0][1)]	Varies		**Isochronous In** **Bit:** Tx bytes [11:0] **Bit ID:** Number of bytes to transmit Tx bytes [11:0] Number of bytes to be transmitted This 12-bit value can take values from 0 to (4K-1) bytes, indicating the number of bytes of data to be transmitted to the USB host. **Note**: A Value of 0 indicates zero bytes of data, 1 indicates 1 byte of data and so on.

1) The meaning of this field varies. See description.

Table 16-12 displays the matrix of IN - L Bit, SP Bit and Tx bytes options.

Table 16-12 IN - L Bit, SP Bit and Tx bytes

L Bit	SP bit	Tx Bytes	Functionality
0	1	Multiple of endpoint maximum packet size	Transmit a zero length packet after the last packet
0	1	Not multiple of maximum packet size	Send short packet at the end after normal packets are sent out. Then move onto next descriptor
0	1	0	Transmit zero length packet. Then move on to next descriptor.
0	0	Multiple of endpoint maximum packet size	Send normal packets and then move to next descriptor.
0	0	Not a multiple of maximum packet size	Transmit the normal packets and concatenate the remaining bytes with next buffer from the next descriptor. This combination is valid only for bulk end points.
0	0	0	Invalid. The behavior of the core is undefined.
1	1	Multiple of endpoint maximum packet size	Transmit a zero length packet after the last packet If this IN descriptor is for a ISO endpoint, then move onto the first descriptor in the list. If this IN descriptor is for a non-ISO endpoint, then stop processing this list and disable the corresponding end point.
1	1	Not multiple of maximum packet size	Send short packet after sending the normal packets If this IN descriptor is for a ISO endpoint, move onto the first descriptor in the list. If this IN descriptor is for a non-ISO endpoint, then stop processing this list and disable the corresponding end point.

Table 16-12 IN - L Bit, SP Bit and Tx bytes (cont'd)

L Bit	SP bit	Tx Bytes	Functionality
1	1	0	Transmit zero length packet If this IN descriptor is for a ISO endpoint, move onto the first descriptor in the list. If this IN descriptor is for a non-ISO endpoint, then stop processing this list and disable the corresponding end point.
1	0	Multiple of endpoint maximum packet size	Send normal packets If this IN descriptor is for a ISO endpoint, Move onto the first descriptor in the list after current transfer done. If this IN descriptor is for a non-ISO endpoint, then stop processing the list and disable the corresponding end point.
1	0	Not multiple of maximum packet size.	Invalid. The behavior of the core is undefined for these values.
1	0	0	invalid. The behavior of the core is undefined for these values.

The descriptions provided for the different combinations in **Table 16-12** depend on the previous descriptor L, SP, and Tx Bytes values. Consider **Table 16-13**. The MPS for this example is 512.

Table 16-13 IN - Buffer Pointer

DESC NO	L bit	SP bit	Txbytes	Description
1	0	0	520	Send a normal packet of size 512, and concatenate the remaining 8 bytes with the next descriptor's buffer data
2	0	1	512	For this combination of L,SP and TxBytes, as per the above table, we need to send a zero length packet instead of a short packet. However, a normal packet followed by a short packet of length 8-bytes is sent. This is to illustrate the context dependency based on previous descriptor L,SP and TxByte combinations.

Table 16-14 displays the IN buffer pointer field description.

Table 16-14 IN Buffer Pointer

Bit	Bit ID	Description
Buf Addr[31:0]	Buffer Address	The Buffer pointer field in the descriptor is 32 bits wide and contains the address where the transmit data is stored in the system memory. The address can be non- DWORD aligned.

16.7.3.5 Descriptor Update Interrupt Enable Modes

If IOC bit is set for a descriptor and if the corresponding Transfer Completed Interrupt Mask (XferComplMask) is unmasked, this interrupt (DIOEPINTn.XferCompl) is asserted while closing that descriptor.

16.7.3.6 DMA Arbitration in Scatter/Gather DMA Mode

The arbiter grants receive higher priority than transmit. Within transmit, the priority is as follows.

- The highest priority is given to periodic endpoints. The periodic endpoints are serviced in a round robin fashion.
- The non periodic endpoints are serviced after the periodic scheduling interval has elapsed. The duration of the periodic scheduling interval is programmable, as specified by register bits DCFG[25:24]. When the periodic interval is active, the periodic endpoints are given priority.
- Amongst the periodic endpoints, the priority is round robin.
- Amongst the non periodic endpoints, the Global Multi Count field in the Device Control Register (DCTL) specifies the number of packets that need to be serviced for that end point before moving to the next endpoint.

The arbiter disables an endpoint and moves on to the next endpoint in the following scenarios as well, for all the endpoint types:

- Descriptor Fetch and AHB Error occurs.
- Buffer Not Available (BNA), such as when buffer status is Host busy.
- AHB Error during Descriptor update stage and Data transfer stage.

16.7.3.7 Buffer Data Access on AHB in Scatter/Gather DMA Mode

The buffer address whose data needs to be accessed in the system memory can be non DWORD aligned for transmit.

For buffer data read, the core arranges the buffer data to form a quadlet internally before populating the TXFIFO within the core as per the following scenarios

- The packet starts in a non DWORD aligned address, the core does two reads on AHB before appending the relevant bytes to form a quadlet internally. Hence the core stores the bytes before pushing to the TXFIFO.
- The packet ends in a non DWORD aligned address and it is not the end of the buffer or expected transfer, the core may switch to service another end point and come back to service the initial end point. In this case, the core reads the same DWORD location again and then samples only the relevant bytes. This eliminates the storage of the bytes for the initial end point.

For buffer data write, the core always performs DWORD accesses.

16.7.4 Control Transfer Handling

Control transfers (3-Stage Control R/WR or 2-Stage), can be handled effectively in the Descriptor-Based Scatter/Gather DMA mode by following the procedure explained in this section. By following this procedure the application is able to handle all normal control transfer flow and any of the following abnormal cases.

- More than one SETUP packet (back to back) — Host could send any number of SETUP packets back to back, before sending any IN/OUT token. In this case, the application is suppose to take the last SETUP packet, and ignore the others.
- More OUT/IN tokens during data phase than what is specified in the wlength field — If the host sends more OUT/IN data tokens than what is specified in the wlength field of the SETUP data, then the device must STALL.
- Premature SETUP packet during data/status phase — Device application must be able to handle this SETUP packet and ignore the previous control transfer.
- Lost ACK for the last data packet of a Three-Stage Control Read Status Stage.

16.7.5 Interrupt Usage for Control Transfers

The application checks the following OUT interrupts status bits for the proper decoding of control transfers.

- DIEPINTx.XferCompl (Transfer complete, based on IOC bit in the descriptor)
- DIEPINTx.InTknTxfEmp (In token received when Tx FIFO is empty)
- DOEPINTx.XferCompl (Transfer complete, based on IOC bit in the descriptor)
- DOEPINTx.SetUp (Setup Complete interrupt, generated when the core receives IN/OUT token after a SETUP packet.
- DOEPINTx.StsPhseRcvd (Status phase received interrupt (Also called SI), generated when host has switched to status phase of a Control Write transfer).

The core performs some optimization of these interrupt settings, when it sees multiple interrupt bits need to be set for OUT endpoints. This reduces the number of valid combinations of interrupts and simplifies the application.

The core gives priority for DOEPINTx.XferCompl over DOEPINTx.SetUp and DOEPINTx.StsPhseRcvd (SI) interrupts. When setting the XferCompl interrupts, it clears the SetUP and SI interrupt bits.

- The core gives priority to DOEPINTx.SI interrupt over DOEPINTx.SetUp. When setting DOEPINTx.StsPhseRcvd (SI), the core clears DOEPINTx.SetUp interrupt bit.

Based on this, the application needs only to decode the combinations of interrupts for OUT endpoints shown in **Table 16-15**.

Table 16-15 Combinations of OUT Endpoint Interrupts for Control Transfer

StsPhse Rcvd (SI)	SetUp (SPD)	XferCompl (IOC)	Description	Template Used
0	0	1	Core has updated the OUT descriptor. Check the "SR" (Setup Received) bit in the descriptor to see if the data is for a SETUP or OUT transaction.	Case A
0	1	0	Setup Phase Done Interrupt for the previously decoded SETUP packet.	Case B
0	1	1	The core has updated the OUT descriptor for a SETUP packet, and the core is indicating a SETUP complete status also.	Case C
1	0	0	Host has switched to Status phase of a Control OUT transfer	Case D
1	0	1	Core has updated the OUT descriptor. Check the SR" (Setup Received) bit in the descriptor to see if the data is for a SETUP or OUT transaction. Also, the host has already switched to Control Write Status phase.	Case E

16.7.6 Application Programming Sequence

This section describes the application programming sequence to take care of normal and abnormal Control transfer scenarios.

All the control transfer cases can be handled by five separate descriptor lists. The descriptor lists are shown in **Figure 16-36**.

- Three lists are for SETUP. The SETUP descriptors also take data for the Status stage of Control Read.
- The first two (index 0 and 1) act in a ping-pong fashion.

- The third list is an empty list, linked to one of the OUT descriptors when premature SETUP comes during the data/ status phase.
- Two lists are for IN and OUT data respectively.
- **Figure 16-36** displays setup_index 0, 1, and 2 as elements of array of pointers called setup_index. The first two elements of this array point to SETUP descriptors. The third element of this array is initially a NULL pointer, but is eventually linked to a SETUP descriptor. These array elements could also point to a descriptor for Control Read Status phase.

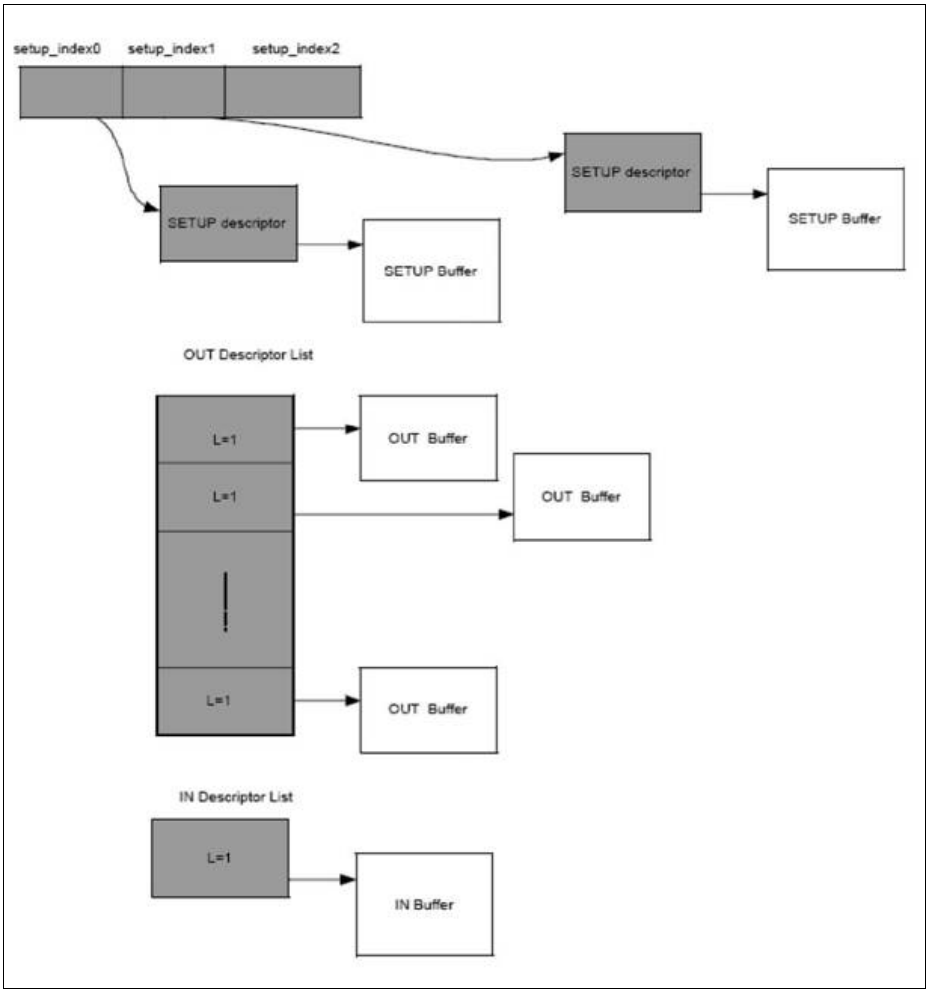


Figure 16-36 Descriptor Lists for Handling Control Transfers

The following are the steps that need to be followed by the application driver.

1. **Set up Desc for SETUP/Ctrl-Rd-Sts** — Setup 2 descriptor lists in memory for taking in SETUP packets. Each of this list must have only one descriptor, with the descriptor fields set to the following
   a) Rx_bytes — Set it to Max packet size of the control endpoint.
   b) IOC =1.
   c) MTRF=0.

d) L=1.

2. **Enable DMA**—If current setup_index =0, then setup_index=1. The application ping-pongs between these two descriptors. Program the address of the current setup descriptor (specified by setup_index) to DOEPDMAx. Write to DOEPCTLx with the following fields.
   a) DOEPCTL.MPS — Max Packet size of the endpoint
   b) DOEPCTL.EPEna — Set to 1 to enable the DMA for the endpoint.
3. **Wait for Interrupt**—Wait for OUT endpoint interrupt (GINTSTS.OEPInt). Then read the corresponding DOEPINT.
4. If Control Read Data Stage in progress
   a) Case A—Check SR bit (In this case SR bit is set, because the host cannot send OUT at this point. If it sends OUT it is NAKed. GOTO Step 24.
   b) Case B —GOTO Step 26.
   c) Case C:-Check SR bit (In this case SR bit is set because host cannot send OUT packets without SETUP at this stage). GOTO Step 24.
   d) Case D — Cannot happen at this stage because SI cannot come alone without a SETUP, at this stage.
   e) Case E — Indicates that host has switched to another SETUP (Three-Stage control write) and then has switched to status phase without and data phase (core clears SUP with SI in this case). Decode SETUP packet and if ok, GOTO Step 11.

      else If Ctrl Write Status Stage in progress OR Two-Stage Status Stage in progress

   f) Case A—Check SR bit (In this case SR bit is set, because the host cannot send OUT at this point. If it sends OUT it is NAKed.) GOTO Step 24.
   g) Case B — (Could happen for Two-Stage Ctrl Transfer.) GOTO Step 26.
   h) Case C—GOTO Step 24.
   i) Case D — Clear SI interrupt and wait Step 3.
   j) Case E — Cannot happen at this stage.

      else

   k) Case A—GOTO Check Desc.
   l) Case B — Normally, this does not occur at this stage. Either IOC comes first or IOC comes with SUP (Case C).
   m)Case C— GOTO Check Desc.
   n) Case D — Cannot happen at this point.
   o) Case E — If SR==1, Indicates Three stage control Transfer SETUP and that the host has switched to status phase. Decode the SETUP packet and Goto Step 11.
   p) Check Desc

      Read the Descriptor status quadlet corresponding to the setup_index and check the SR field. (Application might also want to check the BS and RxSts fields and

take necessary actions if there is any abnormalities). If SR field is 1 GOTO Step 5 (If Step Step 20 is active, terminate it). If SR field is 0 GOTO Step 22 (Control Rd Status phase) (This must also terminate Step 20).

5. Decode SETUP—Decode the SETUP packet. If it is a Three-Stage Control Write, GOTO Step 20. If it is a Three-Stage Control Read, GOTO Step 15. If it is a Two-Stage Control transfer, GOTO Step 11 (Same as Status stage for 3-Stage Control Write).
6. Desc list for Ctrl Wr data— Setup descriptor list for Control write data phase. This must be based on the Wlength field in the SETUP data. The descriptors in the list must be setup such that there must be one descriptor per packet. Each of these descriptors must have the control fields set as follows.
   a) Rx_Bytes — Set to the Max Packet Size of the control Endpoint.
   b) IOC = 1
   c) MTRF = 0.
   d) L=1.
   e) At this point we are not enabling and clearing the NAK for the IN endpoint for status phase.This is because, the status phase for Control Write can be ACKed only after decoding the complete data for the data phase.
   GOTO Step 7.
7. Enable DMA for Ctrl Wr Data—Write the start address of this list to DOEPDMAx. Program the DOEPCTLx with the following bits set
   a) DOEPCTL.MPS — Max Packet size of the endpoint
   b) DOEPCTL.EPEna — Set to 1 to enable the DMA for the endpoint. GOTO Step 8.
8. Wait for Ctrl Wr Data Interrupt—Wait for OUT endpoint interrupt (GINTSTS.OEPInt). Then read the corresponding DOEPINTx.
   a) Case A—check the SR field. Also clear DOEPINTx.XferCompl by writing to DOEPINTx.(Application might also want to check the BS and RxSts fields and take necessary actions if there is any abnormalities). If SR field is 0 GOTO Step 9.If SR field is 1, GOTO Step 23. (This indicates that the host has switched to a new control transfer).
   b) Case B —GOTO Step 25.
   c) Case C—GOTO Step 23. (This indicates that the host has switched to a new control transfer).
   d) Case D — Host has switched to status phase. Decode the data received so far. GOTO Step 10.
   e) Case E — Check SR bit. If SR==0, decode the data received so far. GOTO Step 10. If SR==1, decode the SETUP packet and Goto Step10.
9. Check Desc — If it's not the last packet of data phase, Re-enable the endpoint and clear the Nak. This is because the core sets NAK after receiving each OUT packet for control write data phase. This is to allow application to STALL in case the host sends more data than what is specified in the Wlength field. GOTO Step 8. Re-enabling and clearing the NK involves the following steps.
   a) Write to DOEPDMA with the new descriptor address.

b) Write to DOEPCTLx with the following fields.
- DOEPCTL.MPS — Max Packet size of the endpoint
- DOEPCTL.CNAK—Set to 1 to clear the NAK.
- DOEPCTL.EPEna — Set to 1 to enable the DMA for the endpoint.

If it is the last packet of the data phase, GOTO Step 10.

10. **STALL Extra Bytes**— Write to DOEPCTLx with Stall set so that the core could STALL any further OUT tokens from host.If the received Bytes so far is greater than what is specified in Wlength field OR is there were any unsupported commands in the data phase, then write to DIEPCTLx with the Stall bit set so that the Status phase could be Stalled.(The STALL bit is automatically cleared by the core with the next SETUP). GOTO Step 11.
11. **Disc list for Ctrl Wr Sts**— The following two process must run in parallel. This is because, we are preparing for the status phase (IN) of Control write but at the same time the host could send another SETUP. So IN and OUT descriptor list must be ready.
    a) Do Step 2— Step 5 (This is for handling SETUP or Ctrl Wr Status). If the OUT DMA is already enabled (OUT DMA was enabled for data phase of Three-Stage Control Write, but there was a premature status phase), GOTO Step 3.
    b) Setup descriptor list for Status phase IN, depending on the data in the status phase. Normally it is always a zero length packet.
    c) Tx_Bytes — Size of status phase,
    d) BS — Host Ready,
    e) L=1.
    f) IOC=1.
    g) SP=1 (Depending on the Tx_Bytes).
    h) Write to DIEPDMAx with the start address of the descriptor.Write to DIEPCTLx clear the NAK and enable the endpoint. Flush the corresponding TX FIFO.
    i) If SI has not been received in the data stage prior to the status stage, then wait for SI before clearing the NAK(DIEPCTLx.CNAK=1)
    j) DIEPCTLx.EpEna=1.
    k) GOTO Step 12.
12. **Wait for Interrupt**—Wait for IN endpoint interrupt (GINTSTS.IEPInt).
13. If IN endpoint INterrupt, and DIEPINTx.XferCompl, then GOTO Step 14.
14. **Check Desc** —Read the Status field of the descriptor. Check Tx_bytes in the descriptor.(Application might also want to check the BS and RxSts fields and take necessary actions if there is any abnormalities). This is end of Three-Stage Control Write OR Two-Stage Control transfer. We are now ready for the next control transfer (Already taken care by process "a" is Step 11.
15. **Desc for Ctrl Rd Data**—The following two steps must be run in parallel. This is because, we are preparing for Data phase of Control read, but at the same time, the host could abnormally abort this control transfer and send a SETUP, or switch to status phase.

16. Do Step 2— Step 5 (This is for handling SETUP and also Control Read Status phase.).
17. Setup descriptor list for Data phase IN, depending on the WLength field in the SETUP data. The setup can be for a single descriptor OR multiple descriptors. If it is multiple descriptors, ensure that IOC for the last descriptor is set.
    a) Tx_Bytes — Size of data phase (Wlength field).
    b) BS — Host Ready
    c) L=1.
    d) IOC=1. It is mandatory to set the IOC when it is the last descriptor.
    e) SP=1 (Depending on the Tx_Bytes).
    f) Write to DIEPDMAx with the start address of the descriptor list.
    g) Write to DIEPCTLx clear the NAK and enable the endpoint.
    h) Flush the corresponding TX FIFO.
    i) DIEPCTLx.MPS = Max_packet size of the endpoint,
    j) DIEPCTLx.CNAK=1 only if SPD already set (Case C in Step 3).
    k) Also set the DOEPCTLx.CNAK for the corresponding OUT endpoint after SPD because a premature status stage (OUT) can come which must be acked.
    l) DIEPCTLx.EpEna=1.
    m)GOTO Step 18.
18. **Wait for Interrupt**—Wait for IN endpoint interrupt (GINTSTS.IEPInt)
19. If IN endpoint interrupt, read the corresponding DIEPINTx and if XferCompl is set GOTO Step 20.
20. **Check_Desc**—Wait for the DIEPINTx.IOC interrupt. Go to Step 21.
21. **Set_Stall**—Write to DIEPCTLx with STALL bit set. (The STALL bit is automatically cleared by the core with the next SETUP). The function of this process initiated in step Step 15 is over, and must be terminated. The next control transfer is already taken care by the process that is running from Step 2.
22. **Ctrl Rd Sts Desc Check** — Read the descriptor to check the Rxbytes and also check the SP field. The Three-Stage control Read is complete here. GOTO Step 2, in preparation for the next SETUP.
23. The unexpected SETUP packet now received during the control write data phase, is sitting in the descriptor allocated for Data. Link this to the setup descriptor pointer. setup_desc_index = 2. Point setup_desc_index to the current OUT descriptor (which has the SETUP). GOTO Step 5.
24. Disable IN Endpoint DMA. Core flushes the corresponding Tx FIFO in order to flush the data that was meant for Control Write Status phase OR Control Read data phase. If Step 12 or Step 18 is active, terminate it. GOTO Step .
25. Read Modify write DOEPCTLx to clear the NAK. Then GOTO Step 8 again.
    a) DOEPCTLx.CNAK—Set to 1 to clear the NAK.
26. Read Modify write DIEPCTLx to clear the NAK. Then GOTO Step 3 again.
    a) DIEPCTLx.CNAK—Set to 1 to clear the NAK.
27. Read Modify write DIEPCTLx to clear the NAK. Then Step 12 again
    a) DIEPCTLx.CNAK—Set to 1 to clear the NAK.

b) DOEPCTLx.CNAK:Set to 1 to clear the NAK for the out endpoint. This clears the NAK to accept status stage data in case of control read.

16.7.7 Internal Data Flow

This section explains the cores internal data flow for control transfers.

16.7.7.1 Three-Stage Control Write

Figure 16-37 displays the core behavior for Three-Stage control write transfers.

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint. Additionally, the clearing of the NAK bit is blocked by the core until the following SPD or SI is read by the application and cleared.
2. The DMA detects the RxFIFO as non-empty and does the following:
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
3. On receiving the first data phase OUT token after the SETUP, the core push a SETUP_COMPLETE status into the RxFIFO.Core NAKs the data phase OUT tokens because of the NAK set on receiving the SETUP packet.
4. The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2).
5. The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status out of the RxFIFO.
6. Application clears NAK for the data phase, after receiving DOEPINTx.SetUp interrupt.
7. The core ACKs and the next OUT token because the NAK has been cleared (provided there is enough space in the RxFIFO.
8. DMA detects the OUT packet in RxFIFO and starts transferring the OUT packet to the system memory.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the OUT packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close descriptor with DMA_DONE status.
9. The core NAKs the next OUT token because the core internally sets the NAK after every control write data phase packets. This is to allow application to Stall any extra tokens.
10. The core generates DOEPINT.XferCompl after closing the OUT descriptor (Step 8).
11. Application clear NAK on receiving DOEPINTx.XferCompl interrupt.
12. Host starts the Status phase by sending the IN token which is NAKed by the core. The core push DATA_PHASE_DONE status into the RxFIFO.

13. The core generates DOEPINTx.XferCompl for the last OUT packet transfer to system memory.
14. The core generates DOEPINT.StsPhsRcvd interrupt after the DMA has popped the DATA_PHASE_DONE status from the RxFIFO.
15. Application clears the NAK and enables the IN endpoint for status phase.
16. The core starts fetching the data for the Status phase
    a) Fetch the descriptor pointed by DIEPDMA.
    b) Fetch the packet (if size >0) to Tx fifo.
    c) Close the descriptor with DMA_DONE status
    d) The core generates DIEPINTx.XferCompl interrupt after closing the descriptor.
17. The core sends out data in response to the Status Phase IN token.

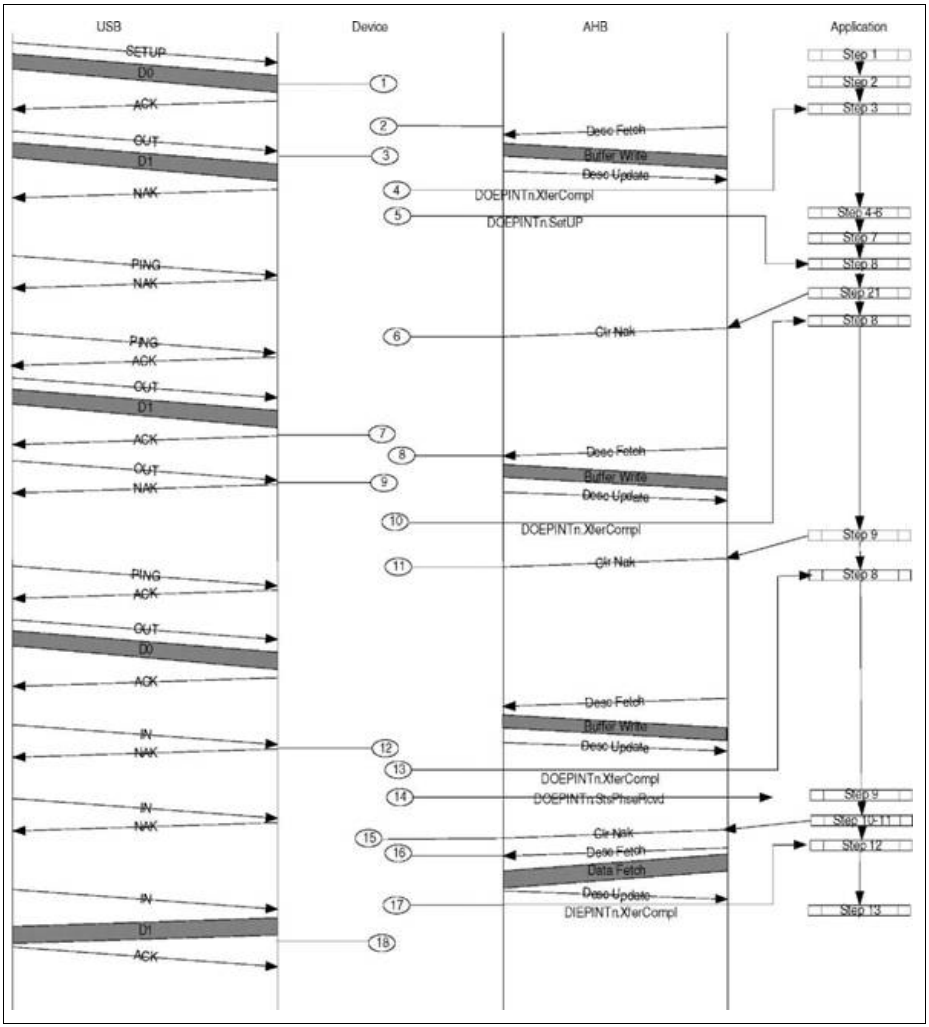


Figure 16-37 Three-Stage Control Write

16.7.7.2 Three-Stage Control Read

Figure 16-38 displays the core flow for three-stage control read transfers

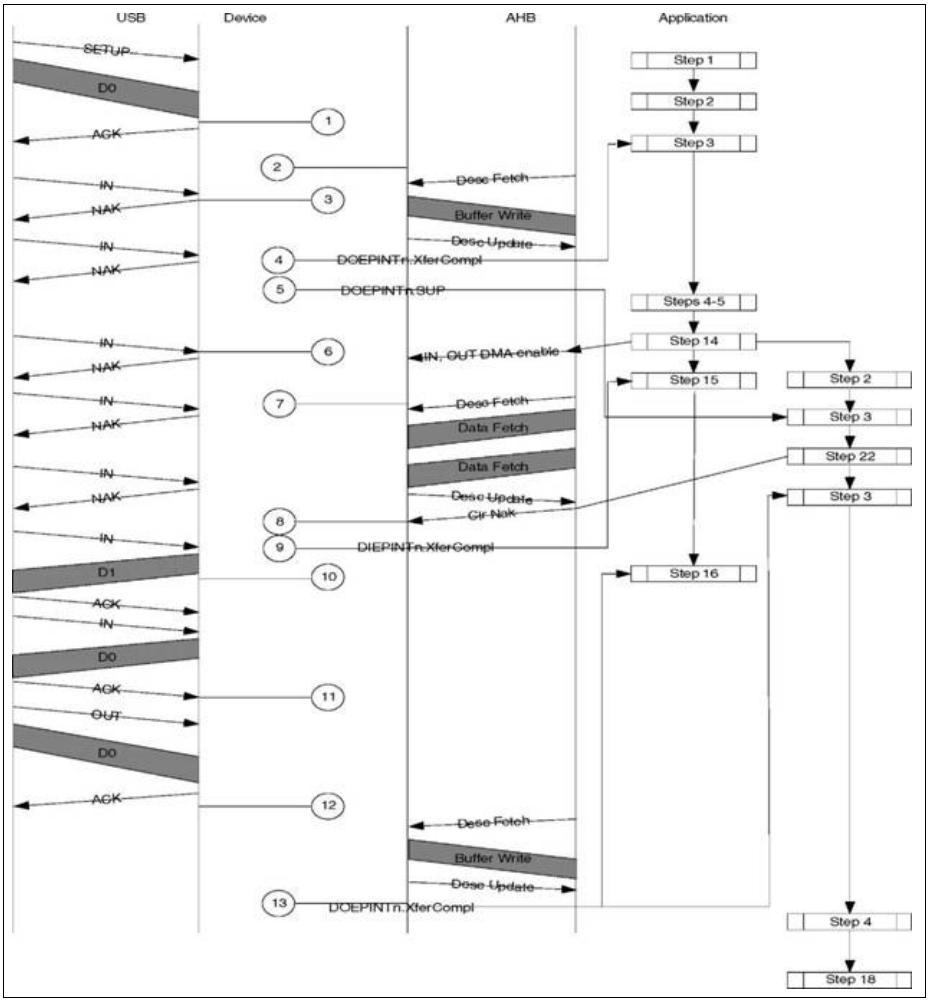


Figure 16-38 Three-Stage Control Read

In this example, it is assumed that the data phase consists of 2 packets, and the application allocates these two packets in a single buffer.

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint.

2. The DMA detects the RxFIFO as non-empty and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
3. On receiving the first data phase OUT token after the SETUP, the core push a SETUP_COMPLETE status into the RxFIFO.Core NAKs the data phase OUT tokens because of the NAK set on receiving the SETUP packet.
4. The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2).
5. The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status out of the RxFIFO.
6. Data phase IN tokens are NAKed until this point because the NAK has not yet been cleared by the application.
7. The core starts fetching the IN data after the application enables IN DMA (In this example it is assumed that multiple packets are in the same buffer. But it could also be in different buffers). This involves the following steps
   a) Fetch the descriptor pointed by DIEPDMA.
   b) Fetch the data into the corresponding Tx FIFO.
   c) Close the descriptor with DMA_DONE status...
8. The application clears the NAK after receiving the setup complete (DOEPINT.SetUp) interrupt. The application also clears NAK of the OUT End point to accept the status phase.
9. After all the data has been fetched for the descriptor (Step 7), core generates DIEPINT.XferCompl interrupt.
10. The core sends data in response to the IN token for the data phase.
11. The core sends out the last packet of the IN data phase.
12. The core ACKs the status phase.
13. The core generates DOEPINTx.XferCompl interrupt after transferring the data received for the status phase to system memory.

16.7.7.3 Two-Stage Control Transfer

Figure 16-39 displays the core behavior for Two Stage control read transfers.

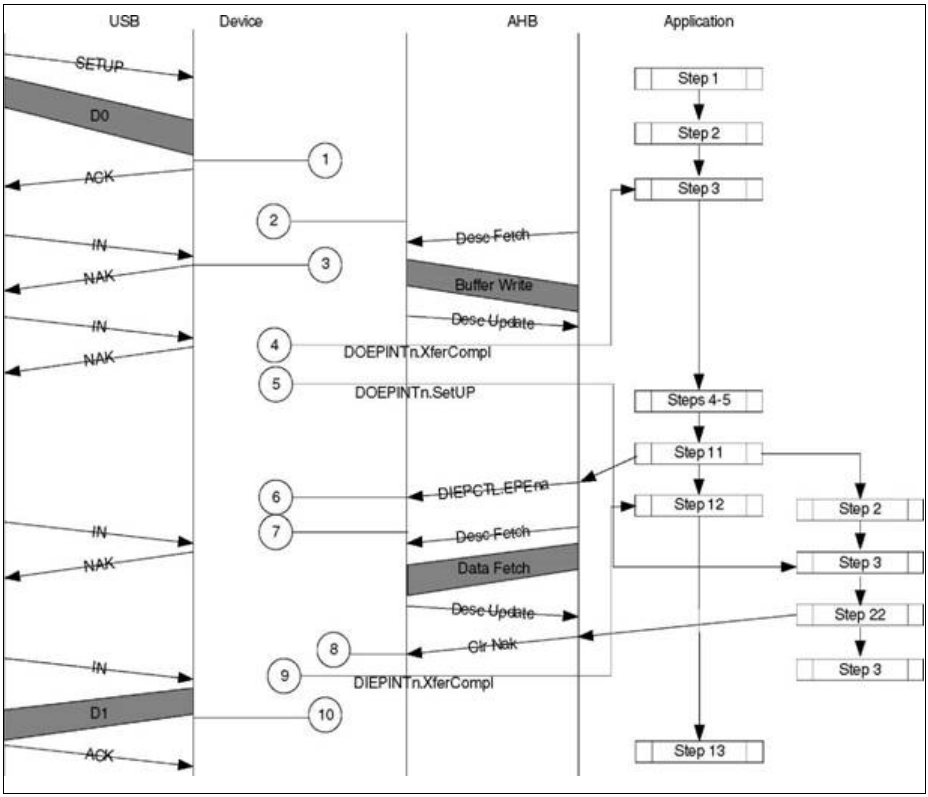


Figure 16-39 Two-Stage Control Transfer

This example shows the core behavior for a Two-Stage Control transfer.

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint.
2. The DMA detects the RxFIFO as non-empty and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
   d) The core receives the status phase IN token, which it NAKs. Core also pushes SETUP_COMPLETE status into the Rx FIFO.

e) The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2)
f) The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status out of the RxFIFO.

3. Application enables the IN endpoint for status phase.
4. The core starts fetching the descriptor for IN endpoint.
5. Application clears the NAK for IN endpoint after getting the DOEPINTx.SetUP interrupt (Step 4).
6. The core generates DIEPINTx.XferCompl after updating the descriptor after IN data fetch.
7. The core sends out data for the status phase IN token from host.

16.7.7.4 Back to Back SETUP During Control Write

This example shows the core receiving 2 Back to Back SETUP tokens fo3 Three-Stage Control write.

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint.
2. The DMA detects the RxFIFO as non-empty and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
   d) The core receives another SETUP, and pushes the data into the Rx FIFO, also sets the NAK.
   e) The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2).
3. On receiving the first data phase OUT token after the SETUP, the core push a SETUP_COMPLETE status into the RxFIFO.Core NAKs the data phase OUT tokens because of the NAK set on receiving the SETUP packet.
4. The DMA detects the RxFIFO as non-empty (because of the 2nd SETUP packet) and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
   d) The core generates DOEPINT.XferCompl interrupt after having transferred the second SETUP packet into memory (Step 6)
   e) The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status out of the RxFIFO.
5. Application clears NAK for the data phase, after receiving DOEPINTx.SetUp interrupt.

6. The core ACKs the next OUT/Ping token after the NAK has been cleared by the application.
7. The DMA detects the RxFIFO as non-empty (because of the OUT packet) and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
   d) The core generates DOEPINT.XferCompl interrupt after having transferred the OUT packet into memory (Step 11) and closing the descriptor.

The remaining steps are similar to Steps 11-18 of **“Application Programming Sequence” on Page 16-143**.

This example shows the core behavior for a Two-Stage Control transfer.

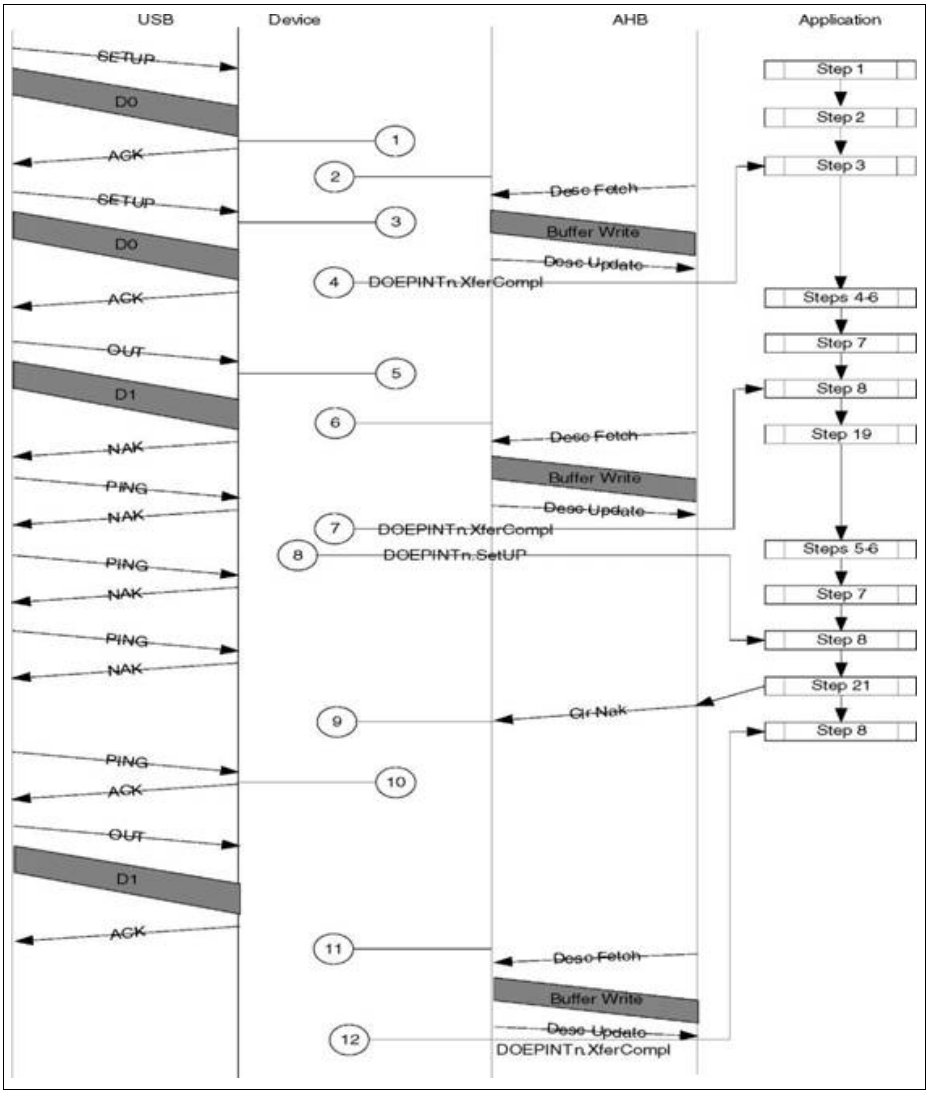


Figure 16-40 Back-to-Back SETUP Packet Handling During Control Write

16.7.7.5 Back-to-Back SETUPs During Control Read

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint.
2. The DMA detects the RxFIFO as non-empty and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
   d) The core receives another SETUP, and pushes the data into the Rx FIFO, also sets the NAK.
3. The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2).
4. Host sends IN token for the data phase which is NAKed by the core, because NAK is set in Setp3. The core pushes SETUP_COMPLETE status into RxFIFO.
5. After the application has re-enabled the OUT DMA (Application flow Step 2) core detects RxFIFO as non-empty because of the second SETUP packet and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
   d) The core generates DOEPINTx.XferCompl interrupt after having transferred the SETUP packet into memory (Step6).
   e) The core starts fetching data for IN endpoint because the IN endpoint was enabled by application in Step-14.
6. On seeing DOEPINTx.XferCompl (Step 7) and finding that it is a SETUP packet, application disables the endpoint in Step 20.
7. The core generates DOEPINTx.SetUP (Setup complete) interrupt after popping the SETUP_COMPLETE status from the RxFIFO.
8. The core generates endpoint disabled interrupt (as a result of application setting disable bit in step 9)
9. The core generates DIEPINTx.XferCompl after completing the IN data fetch and updating the descriptor.
10. application clears NAK after seeing setup_complete interrupt (generated in Step 10. The flow after this is same as steps 9 - 13 of **“Internal Data Flow” on Page 16-150**

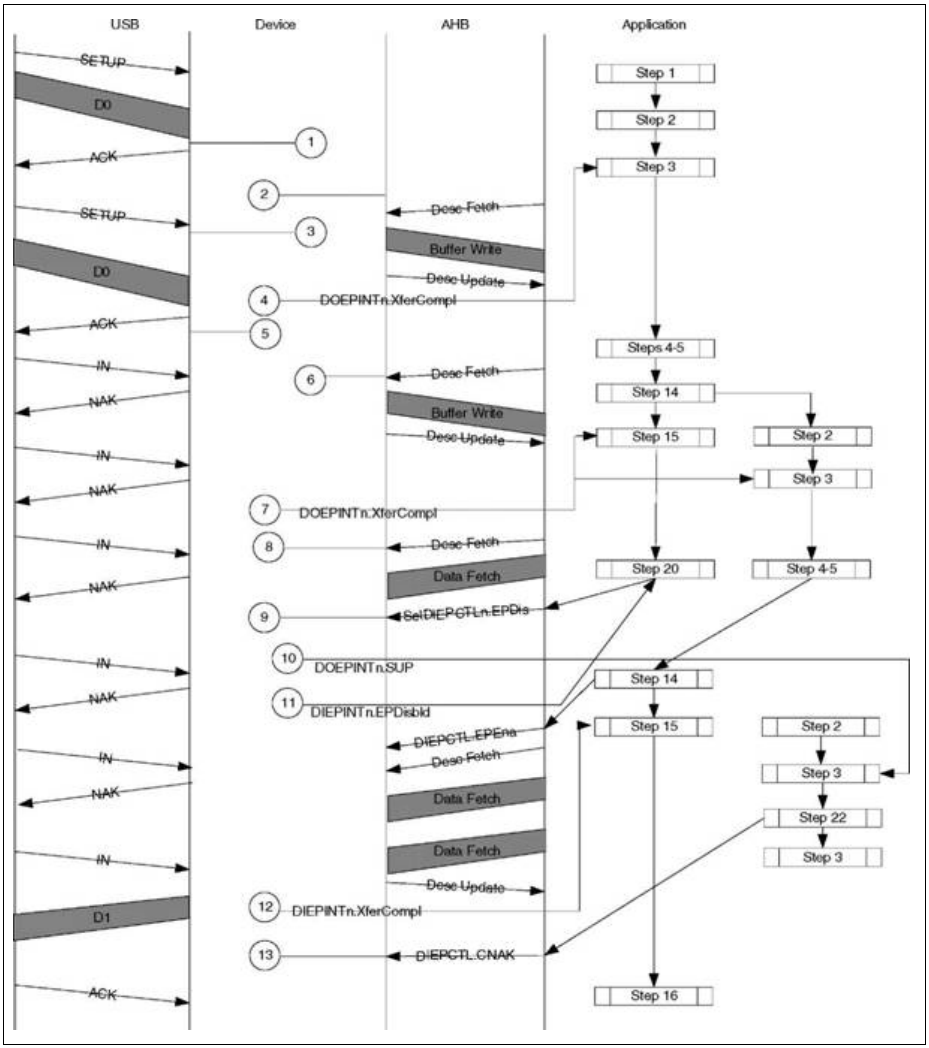


Figure 16-41 Back-to-Back SETUP During Control Read

16.7.7.6 Extra Tokens During Control Write Data Phase

This example assumes a three-stage control write transfer with only Wlength field in the SETUP indicating only 1 packet in the data phase. But the host sends an additional OUT packets which the core STALLs.

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint.
2. The DMA detects the RxFIFO as non-empty and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
3. On receiving the first data phase OUT token after the SETUP, the core push a SETUP_COMPLETE status into the RxFIFO.Core NAKs the data phase OUT tokens because of the NAK set on receiving the SETUP packet.
4. The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2).
5. The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status out of the RxFIFO.
6. Application clears NAK for the data phase, after receiving DOEPINTx.SetUp interrupt.
7. The core ACKs and the next OUT token because the NAK has been cleared (provided there is enough space in the RxFIFO.
8. DMA starts transferring the OUT packet to the system memory.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the OUT packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close descriptor with DMA_DONE status.
   d) The core generates DOEPINTx.XferCompl interrupt after having transferred the OUT packet to the system memory. Since there were only one packet in the data phase, the data phase is complete here.
   e) The core initially NAK's the extra tokens send by the host, because the core internally sets NAK after each OUT packet for the data phase of control write.
9. Application sets STALL to stall any extra tokens.
10. The core stalls the next OUT/PING token.
11. Host switches to next control transfer, core ACKs the SETUP. This SETUP packet is transferred to the system memory buffer originally allocated for Status phase.
12. The core generates DOEPINTx.XferCompl interrupt after transferring the SETUP packet to the system memory.

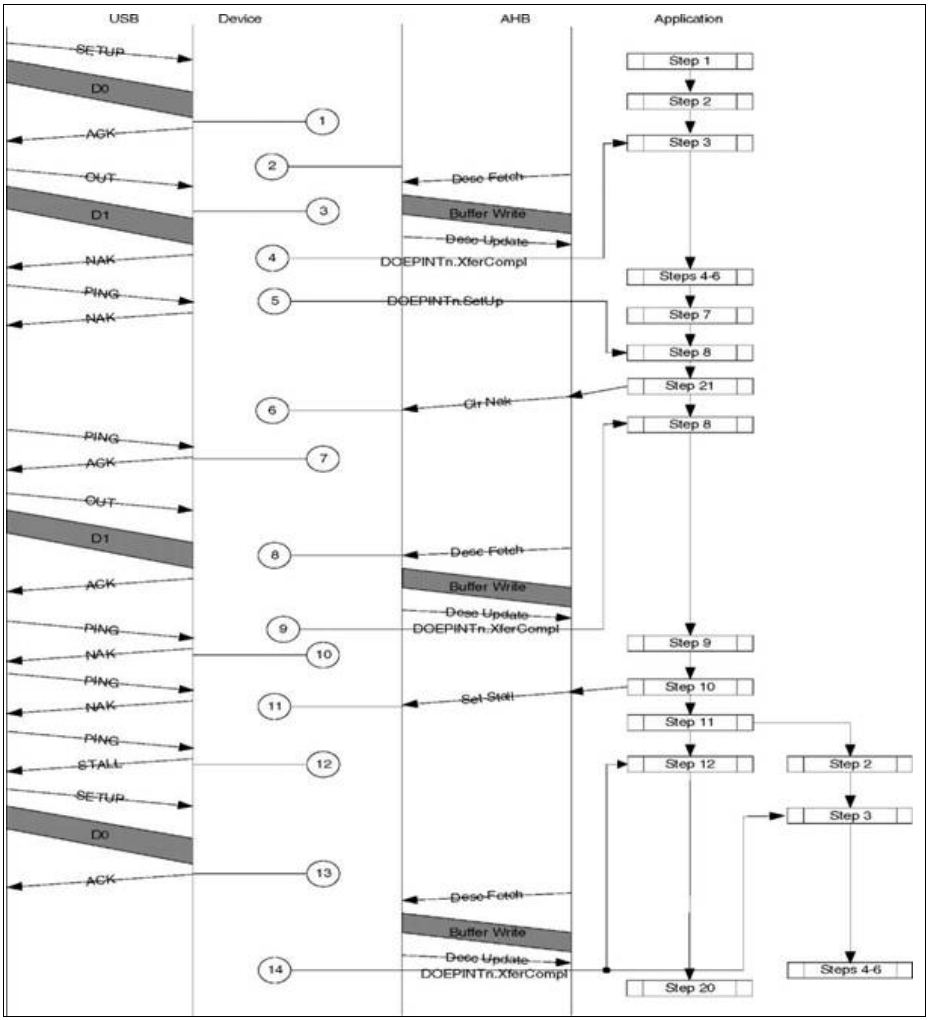


Figure 16-42 Extra Tokens During Control Write Data Phase

16.7.7.7 Extra Tokens During Control Read Data Phase

In this example, it is assumed that the data phase consists of 2 packets, and the application allocates these two packets in a single buffer. After the data phase is complete and the two packets have been transferred, the core sends an extra IN token and then the application sets Stall.

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint.
2. The DMA detects the RxFIFO as non-empty and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
3. On receiving the first data phase OUT token after the SETUP, the core push a SETUP_COMPLETE status into the RxFIFO.Core NAKs the data phase OUT tokens because of the NAK set on receiving the SETUP packet.
4. The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2).
5. The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status out of the RxFIFO.
6. Data phase IN tokens are NAKed until this point because the NAK has not yet been cleared by the application.
7. The core starts fetching the IN data after the application enables IN DMA (In this example it is assumed that multiple packets are in the same buffer. But it could also be in different buffers). This involves the following steps
   a) Fetch the descriptor pointed by DIEPDMA.
   b) Fetch the data into the corresponding Tx FIFO.
   c) Close the descriptor with DMA_DONE status.
8. The application clear the NAK after receiving the setup complete (DOEPINT.SetUp) interrupt. Set the Stall bit after all the Data has been pushed in the FIFO
9. After all the data has been fetched for the descriptor (Step 7), core generates DIEPINT.XferCompl interrupt.
10. The core sends data in response to the IN token for the data phase.
11. The core sends out the last packet of the IN data phase.
12. Host sends an extra token.
13. The core Stalls the IN token and also automatically Stalls the Status phase if the Host switches to the Status phase.

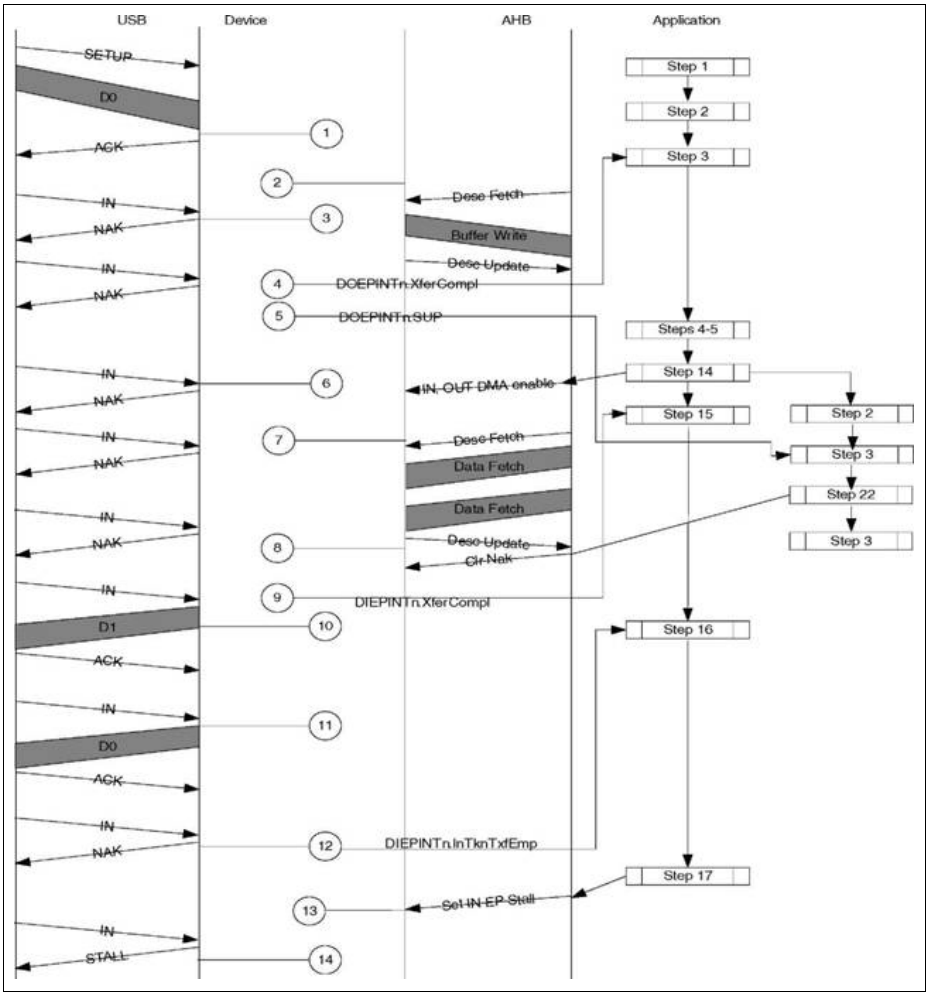


Figure 16-43 Extra IN Tokens During Control Read Data Phase

16.7.7.8 Premature SETUP During Control Write Data Phase

This example shows a Three-Stage Control Write transfer with host sending a premature Control Write SETUP packet during the data phase.

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint.
2. The DMA detects the RxFIFO as non-empty and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
3. On receiving the first data phase OUT token after the SETUP, the core push a SETUP_COMPLETE status into the RxFIFO.Core NAKs the data phase OUT tokens because of the NAK set on receiving the SETUP packet.
4. The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2).
5. The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status out of the RxFIFO.
6. The core receives a SETUP packet during the data phase. This is an unexpected SETUP packet.On receiving this SETUP, the SETUP data is pushed into the RxFIFO and the core again sets NAK on both IN and OUT endpoints of the control endpoint (NAK was already set because of the first SETUP packet received).
7. Application decodes the previous DOEPINT.SetUp interrupt and clears the NAK, unaware of the fact that there is another SETUP packet sitting in the RxFIFO for the same control endpoint. On seeing this condition, core does not allow clearing of the NAK bit, and masks the clearing of NAK. The core takes this decision based on the fact that a SETUP_COMPLETE status is pending in the RXFIFO.
8. The DMA detects the RxFIFO as non-empty (because of the unexpected SETUP) and does following
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
   d) The core NAKs the data phase OUT token because NAK bit clearing by the application did not take effect (as explained in Step 7).
   e) The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 8).
   f) The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status (for the unexpected SETUP packet received) out of the RxFIFO.
9. Application clears the NAK after decoding the latest SETUP packet. This time, the core does not mask the clearing of the NAK because there are no more SETUP_COMPLETE status sitting in the RxFIFO.

10. The core ACKs the next OUT/PING token of the data phase.
11. DMA starts transferring the OUT packet to the system memory.
    a) Fetch the descriptor pointed by DOEPMA.
    b) Transfer the OUT packet from the RxFIFO to the buffer pointed by the descriptor.
    c) Close descriptor with DMA_DONE status.
    d) The core generates DOEPINTx.XferCompl interrupt after having transferred the OUT packet to the system memory.
    e) The remaining steps are similar to Steps 11-18 of **“Application Programming Sequence” on Page 16-143**

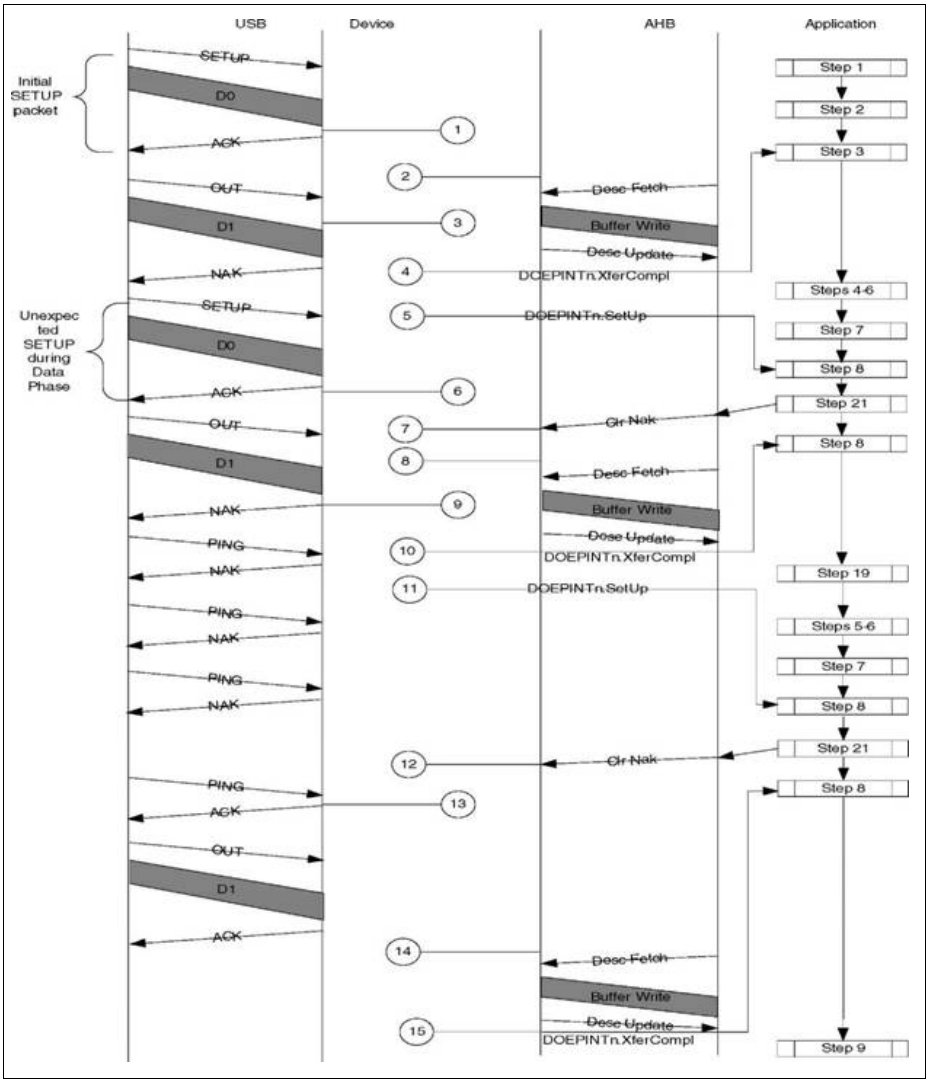


Figure 16-44 Premature SETUP During Control Write Data Phase

16.7.7.9 Premature SETUP During Control Read Data Phase

In this example, it is assumed that the data phase consists of 2 packets, and the application allocates these two packets in a single buffer. The host switches to a new control read command after having send two IN tokens during the data phase.

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint.
2. The DMA detects the RxFIFO as non-empty and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
3. On receiving the first data phase IN token after the SETUP, the core push a SETUP_COMPLETE status into the RxFIFO.Core NAKs the data phase IN tokens because of the NAK set on receiving the SETUP packet.
4. The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2).
5. The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status out of the RxFIFO.
6. Host switches to a new Control transfer by sending a SETUP token. This is the premature SETUP packet. Core sets NAK on both IN and OUT control endpoints.
7. The core fetch the data for IN control endpoint after application enables the IN endpoint.
8. The core push SETUP_COMPLETE status into Rx FIFO on seeing the IN token for data phase.
9. Application clears NAK as a result of DOEPINT.SetUP (Setup complete) interrupt generated in Step 5. But core masks this clearing of setup_complete interrupt, because there is already one SETUP packet sitting in the Rx FIFO.
10. The core generates DIEPINTx.XFERCompl after closing the IN endpoint descriptor (for Step 7)
11. The core generates DOEPINTx.XferCompl after transferring the premature SETUP packet to system memory and closing the descriptor.
12. The core generates SETUP complete interrupt.
13. Application enables IN endpoint DMA for data phase.
14. The core fetches descriptor and data for IN endpoint.
15. Application clears IN endpoint NAK after receiving DOEPINTx.SetUP (Setup complete) interrupt. This time, the core does not mask the clearing of the Nak because NO SETUP packet is remaining in the Rx FIFO.
16. The core generates DIEPINTx.XferCompl interrupt after fetching the data and closing the descriptor. The remaining steps are same as steps 11 to 13 of **“Internal Data Flow” on Page 16-150**.

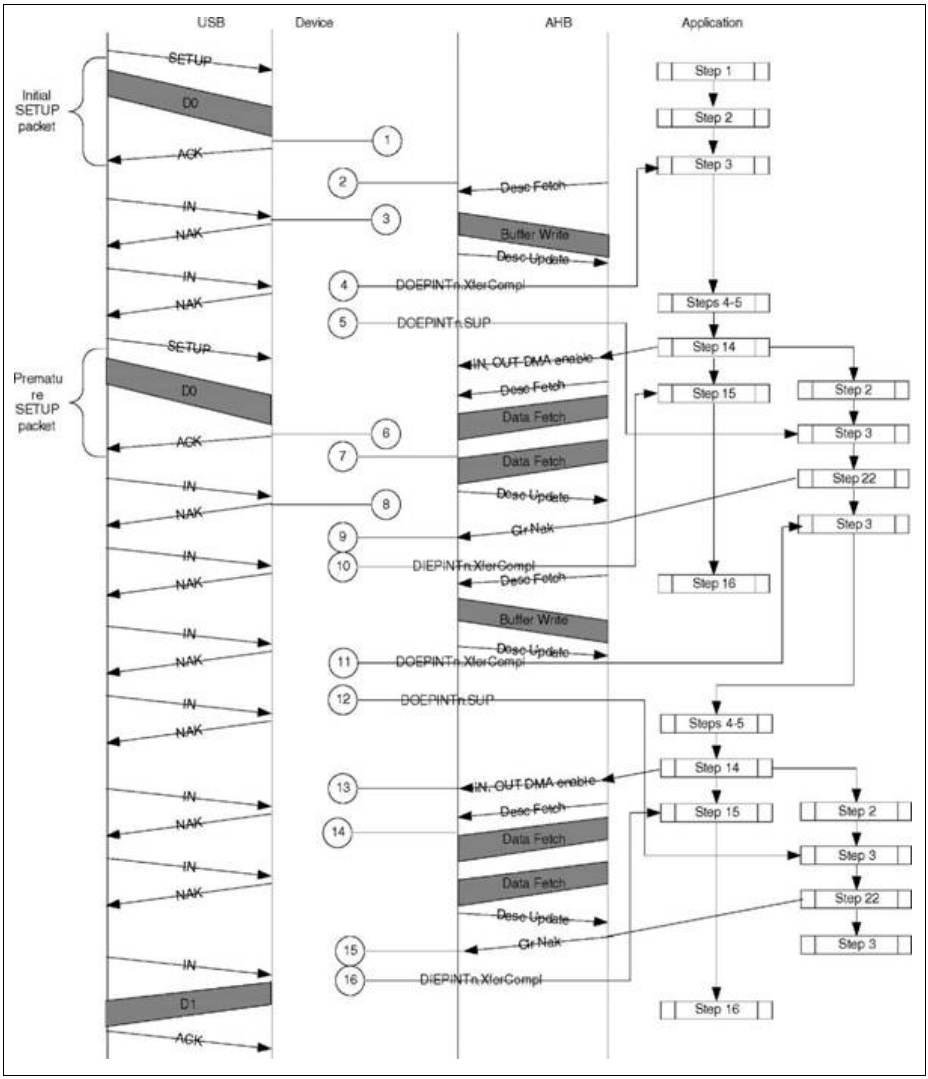


Figure 16-45 Premature SETUP During Control Read Data Phase

16.7.7.10 Premature Status During Control Write

This example assumes a Three-Stage control write transfer with only Wlength field in the SETUP indicating two packets in the data phase. But the host switch to data phase after the first packet of the data phase is complete.

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint.
2. The DMA detects the RxFIFO as non-empty and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
3. On receiving the first data phase OUT token after the SETUP, the core push a SETUP_COMPLETE status into the RxFIFO.Core NAKs the data phase OUT tokens because of the NAK set on receiving the SETUP packet.
4. The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2).
5. The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status out of the RxFIFO.
6. Application clears NAK for the data phase, after receiving DOEPINTx.SetUp interrupt (Step 5).
7. The core ACKs and the next OUT token because the NAK has been cleared (provided there is enough space in the RxFIFO.
8. DMA sees TxFIFO non empty and starts transferring the OUT packet to the system memory.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the OUT packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close descriptor with DMA_DONE status.
9. Host switch to status phase (IN token) without completing the data phase.
10. The core generates DOEPINTx.XferComp after closing the descriptor after the data fetch.
11. Application sets up descriptor, enables IN endpoint and clear NAK.
12. The core starts to fetch the descriptor and data for the status phase once application has enabled the IN endpoint.
13. The core generates DIEPINTx.XferCompl after doing the data fetch and the descriptor update (step 12)
14. The core sends data out in response to status phase IN token.

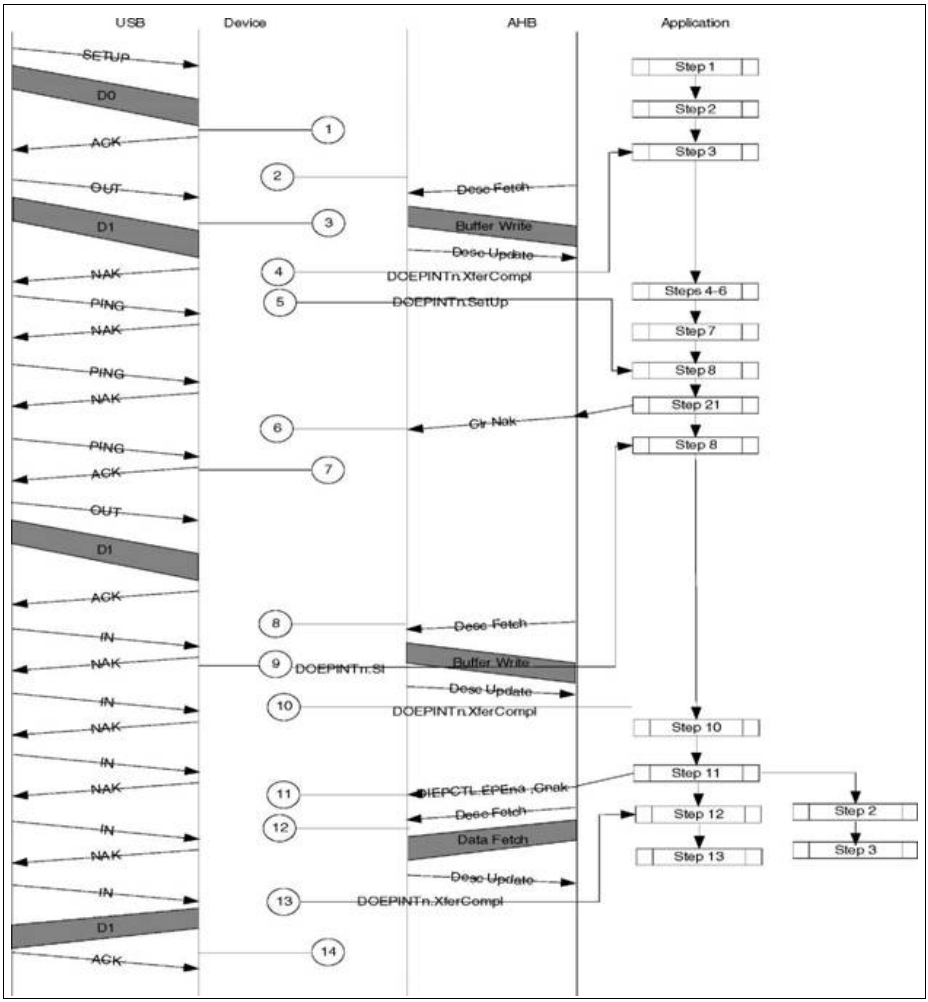


Figure 16-46 Premature Status Phase During Control Write

16.7.7.11 Premature Status During Control Read

In this example, it is assumed that the data phase consists of two packets, and the application allocates these two packets in a single buffer. After one packet in the data phase, host switches to status phase.

1. On receiving SETUP, the data is pushed into the Rx FIFO and the core sets NAK on both IN and OUT endpoint of that control endpoint.
2. The DMA detects the RxFIFO as non-empty and does the following.
   a) Fetch the descriptor pointed by DOEPMA.
   b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
   c) Close the descriptor with DMA_DONE status.
3. On receiving the first data phase IN token after the SETUP, the core push a SETUP_COMPLETE status into the RxFIFO.Core NAKs the data phase IN tokens because of the NAK set on receiving the SETUP packet.
4. The core generates DOEPINT.XferCompl interrupt after having transferred the SETUP packet into memory (Step 2).
5. The core generates DOEPINT.SetUp interrupt after the DMA has popped the SETUP_COMPLETE status out of the RxFIFO.
6. Data phase IN tokens are NAKed until this point because the NAK has not yet been cleared by the application.
7. The core starts fetching the IN data after the application enables IN DMA (In this example it is assumed that multiple packets are in the same buffer. But it could also be in different buffers). This involves the following steps
   a) Fetch the descriptor pointed by DIEPDMA.
   b) Fetch the data into the corresponding Tx FIFO.
   c) Close the descriptor with DMA_DONE status.
8. Application clear the NAK after receiving the setup complete (DOEPINT.SetUp) interrupt.
9. After all the data has been fetched for the descriptor (Step 7), core generates DIEPINT.XferCompl interrupt.
10. The core sends data in response to the IN token for the data phase.
11. Host switches to status phase and sends the status phase OUT token. Core ACks the OUT packet because the NAK has already been cleared.
12. The DMA detects the RxFIFO as non-empty (because of the status phase data) and does following.
    a) Fetch the descriptor pointed by DOEPMA.
    b) Transfer the SETUP packet from the RxFIFO to the buffer pointed by the descriptor.
    c) Close the descriptor with DMA_DONE status.
    d) The core generates DOEPINTx.XferCompl after transferring the status phase data to system memory and closing the descriptor.

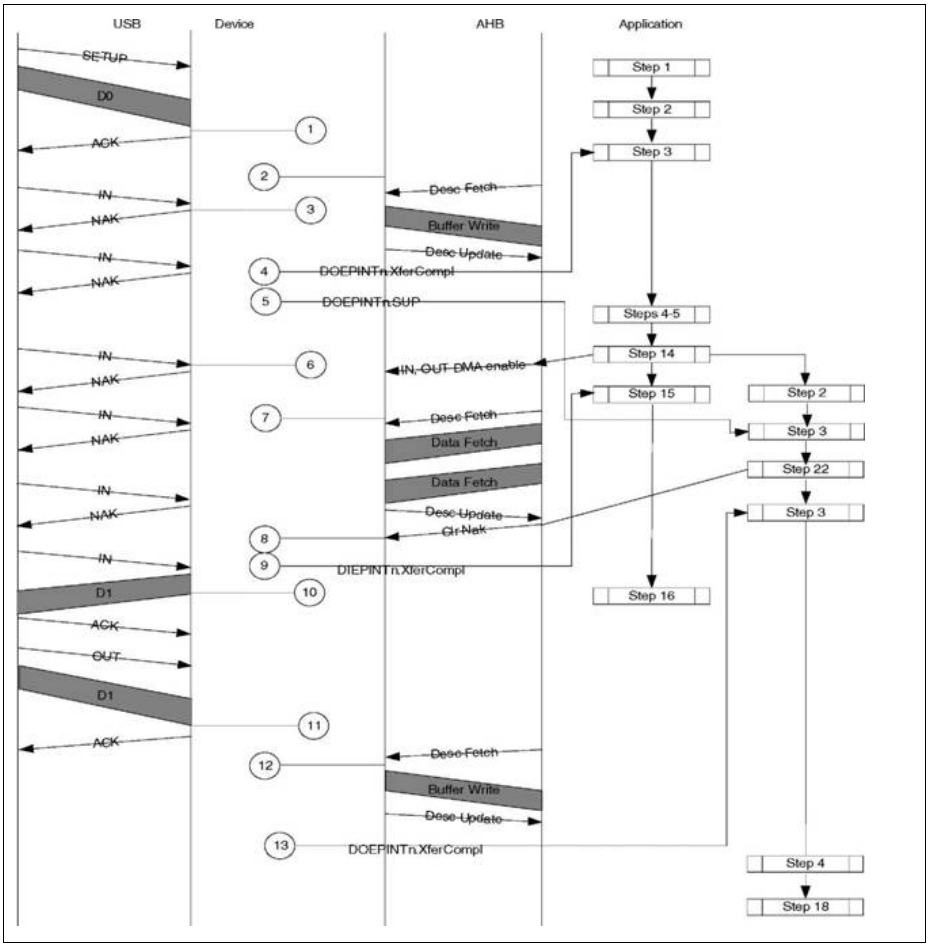


Figure 16-47 Premature Status Phase During Control Read

16.7.7.12 Lost ACK During Last Packet of Control Read

This is similar to the previous section. **Figure 16-48** shows this.

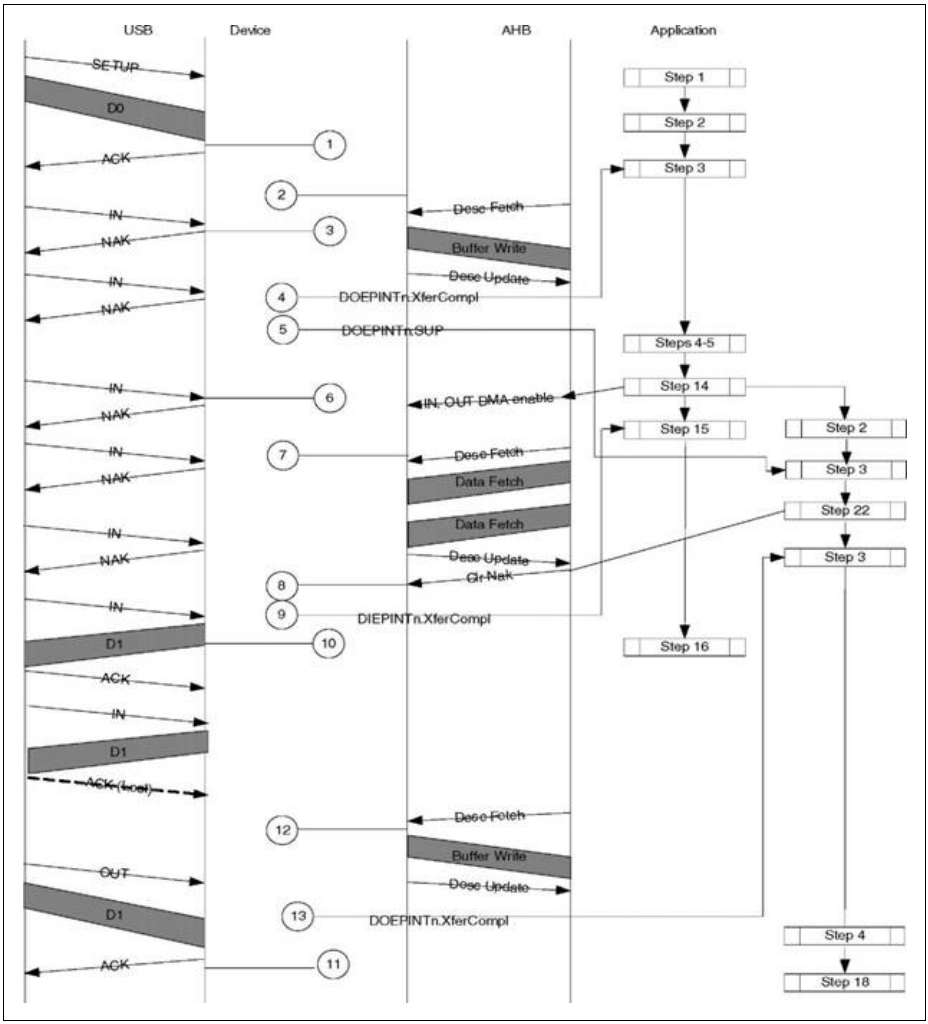


Figure 16-48 Lost ACK During Last Packet of Control Read

16.7.8 Bulk IN Transfer Data Transaction in Scatter-Gather DMA Mode

16.7.8.1 Interrupt usage

The following interrupts are of relevance.

1. DIEPINTx.XferCompl (Transfer complete, based on IOC bit in the descriptor)
2. DIEPINTx.BNA (Buffer Not Available)

16.7.8.2 Application Programming Sequence

This section describes the application programming sequence for Bulk IN transfer scenarios.

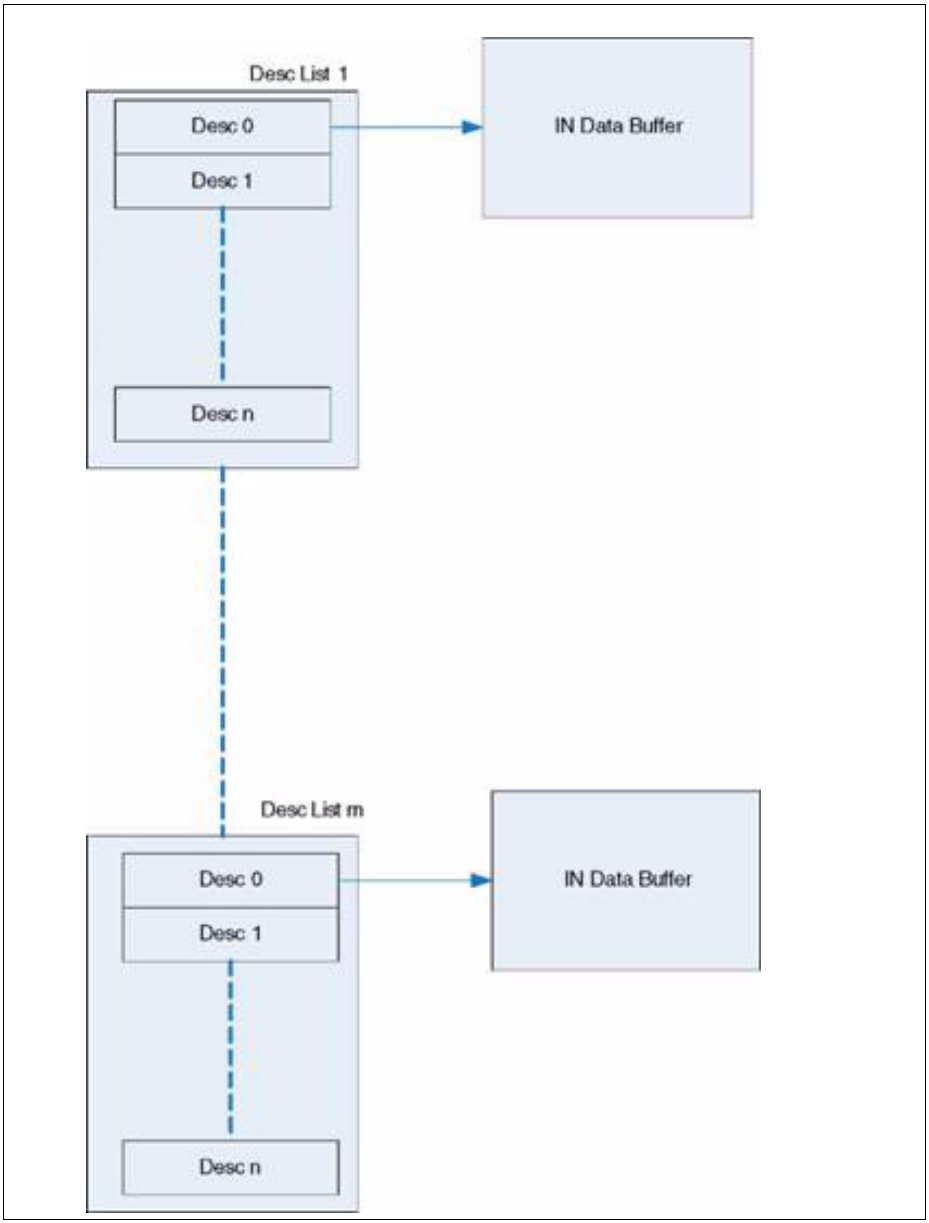


Figure 16-49 IN Descriptor List

1. Prepare Descriptor(s):
2. The application creates descriptor list(s) in the system memory pertaining to an Endpoint.
3. Each descriptor list may have up to n descriptors and there may be up to m descriptor lists.
4. Application may choose to set the IOC bit of the corresponding descriptor. If the IOC is set for the last descriptor of the list, the core generates DIEPINTx.XferCompl interrupt after the entire list is processed.
5. Program DIEPDMAx:
   a) Application programs the base address of the descriptor in the corresponding IN Endpoint DIEPDMAx register.
6. Enable DMA:
   a) Application programs the corresponding endpoint DIEPCTLx register with the following
      - DIEPCTLx.MPS — Max Packet size of the endpoint
      - DIEPCTLx.CNAK—Set to 1 to clear the NAK
      - DIEPCTLx.EPEna — Set to 1 to enable the DMA for the endpoint.
7. Wait for Interrupt:
   a) On reception of DIEPINTx.XferCompl, application must check the Buffer status and Tx Status field of the descriptor to ascertain that the descriptor closed normally.

DIEPINTx.BNA interrupt gets generated by the core when it encounters a descriptor in the list whose Buffer Status field is not Host Ready. In this case, the application is suppose to read the DIEPDMAx register to ascertain the address for which the BNA interrupt is asserted to take corrective action.

16.7.8.3 Internal Flow

Bulk IN Transfers

The core handles Bulk IN transfers internally as functionally depicted in **Figure 16-50** (Non ISO IN Descriptor/Data Processing). **Figure 16-51** depicts this flow.

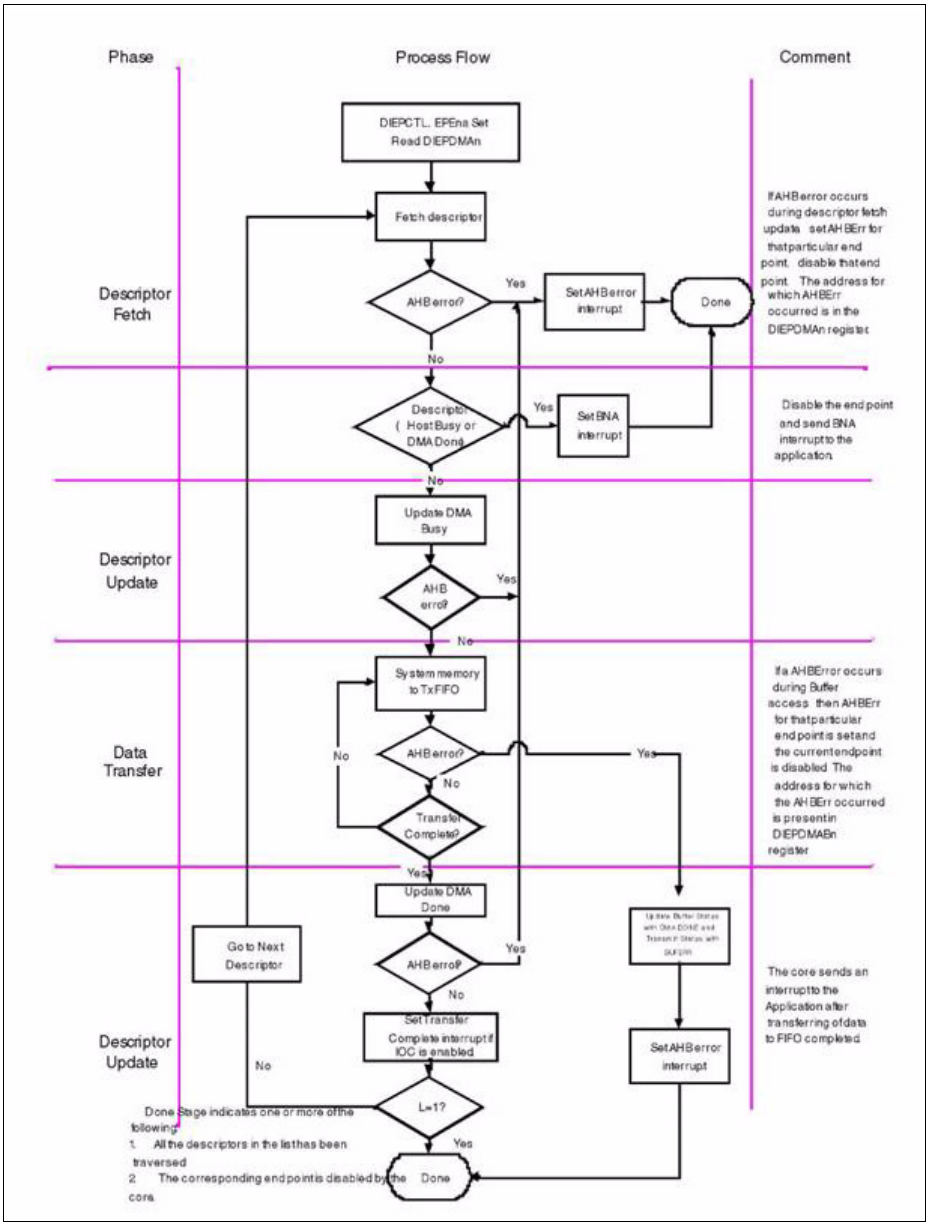


Figure 16-50 Non ISO IN Descriptor/Data Processing

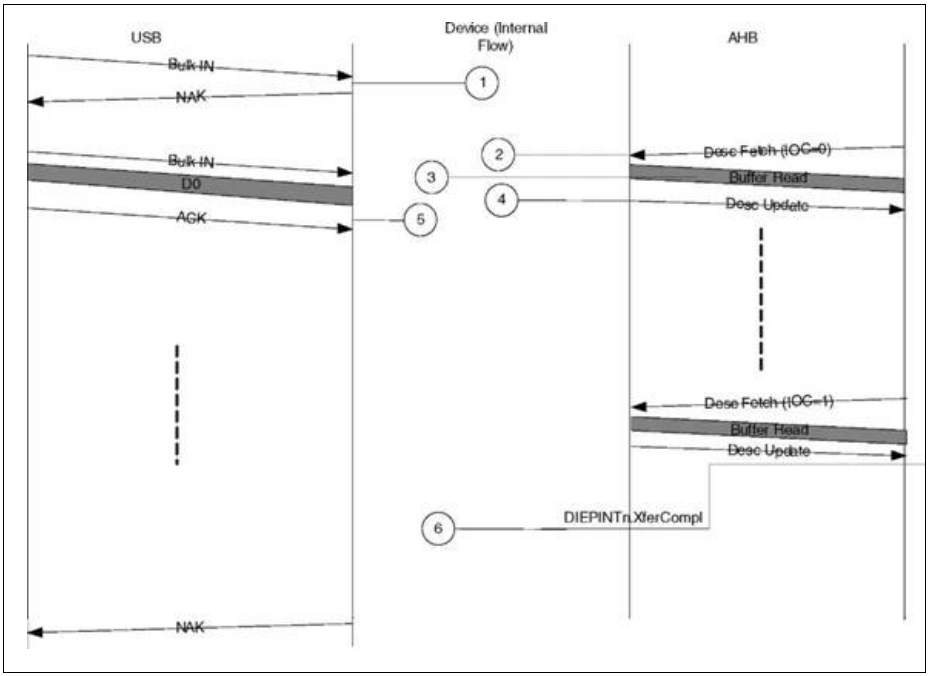


Figure 16-51 Bulk IN Transfers

1. When a BULK IN token is received on an end point before the corresponding DMA is enabled, (DIEPCTLx.EPEna = 0_B), it is NAKed on USB.
2. As a result of application enabling the DMA for the corresponding end point (DIEPCTLx.EPEna=1), the core fetches the descriptor and processes it.
3. The DMA fetches the data from the system memory and populates its internal FIFO with this data.
4. After fetching all the data from a descriptor, the core closes the descriptor with a DMA_DONE status.
5. On reception of BULK IN tokens on USB, data is sent to the USB Host.
6. After the last descriptor in the chain is processed, the core generates DIEPINTx.XferCompl interrupt provided the IOC bit for the last descriptor is set.

16.7.9 Bulk OUT Data Transaction in Scatter-Gather Mode

16.7.9.1 Interrupt Usage

The following interrupts are of relevance.

1. DOEPINTx.XferCompl (Transfer complete, based on IOC bit in the descriptor)
2. DOEPINTx.BNA (Buffer Not Available)

16.7.9.2 Application Programming Sequence

This section describes the application programming sequence to take care of Bulk OUT transfer scenarios.

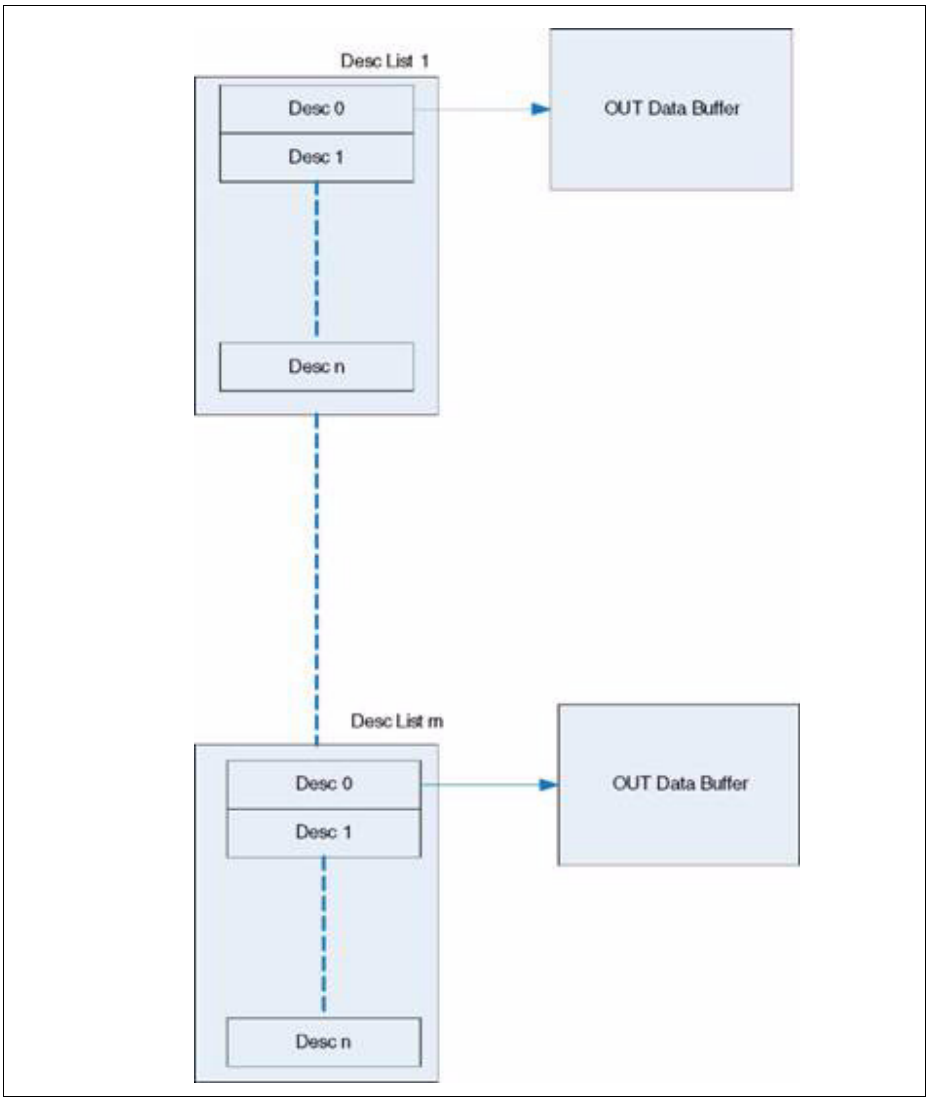


Figure 16-52 OUT Descriptor List

1. Prepare Descriptor(s):
2. The application creates descriptor list(s) in the system memory pertaining to an Endpoint.
3. Each descriptor list may have up to n descriptors and there may be up to m descriptor lists.
4. Application may choose to set the IOC bit of the corresponding descriptor. If the IOC is set for the last descriptor of the list, the core generates DOEPINTx.XferCompl interrupt after the entire list is processed.
   a) a. Based on L bit and MTRF bit combinations, the core may disable the end point. Refer to **Table 16-8 "OUT Data Memory Structure Values" on Page 16-130** for bit field descriptions.
5. Program DOEPDMAx:
   a) Application programs the base address of the descriptor in the corresponding OUT Endpoint DOEPDMAx register.
6. Enable DMA:
   a) Application programs the corresponding endpoint DOEPCTLx register with the following
      - DOEPCTL.MPS — Max Packet size of the endpoint
      - DOEPCTL.CNAK—Set to 1 to clear the NAK
      - DOEPCTL.EPEna — Set to 1 to enable the DMA for the endpoint.
7. Wait for Interrupt:
   a) On reception of DOEPINTx.XferCompl, application must check the Buffer status and Rx Status field of the descriptor to ascertain that the descriptor closed normally.

DOEPINTx.BNA interrupt gets generated by the core when it encounters a descriptor in the list whose Buffer Status field is not Host Ready. In this case, the application is suppose to read the DOEPDMAx register to ascertain the address for which the BNA interrupt is asserted to take corrective action.

16.7.9.3 Internal Flow

The core handles Bulk OUT transfers internally as depicted in **Figure 16-53**. **Figure 16-54** also diagrams this flow.

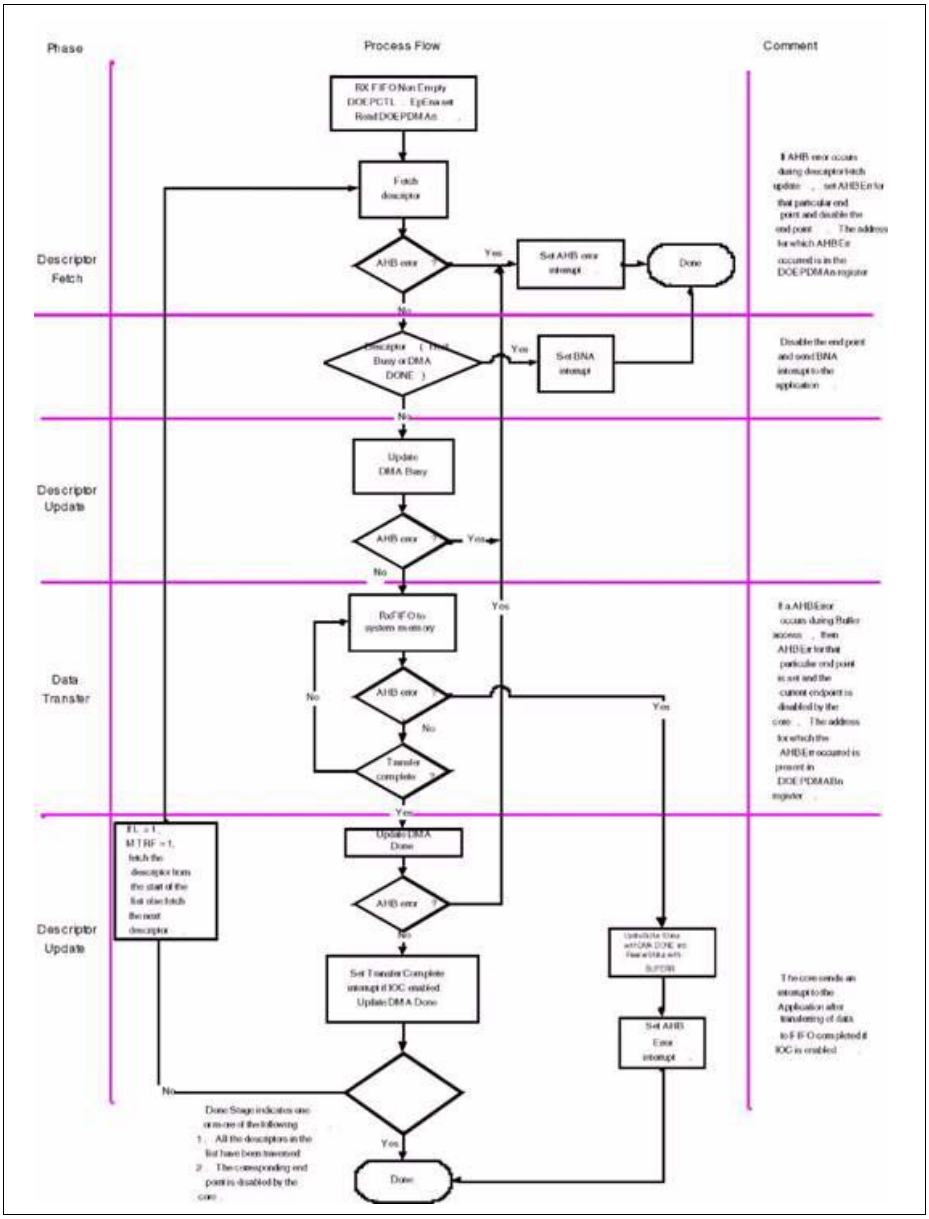


Figure 16-53 Non ISO OUT Descriptor/Data Buffer Processing

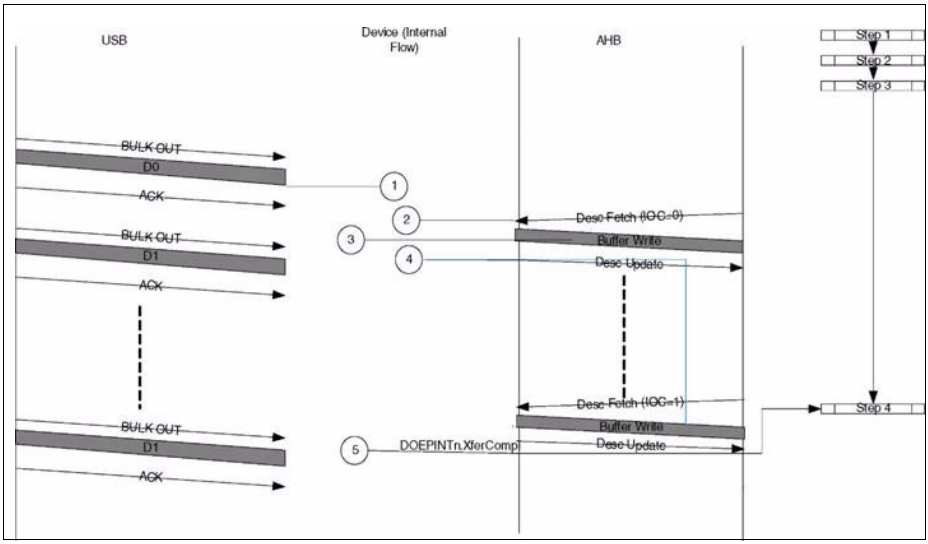


Figure 16-54 Bulk OUT Transfers

1. When a BULK OUT token is received on an end point, the core stores the received data internally in a FIFO.
2. As a result of application enabling the DMA for the corresponding end point (DOEPCTLx.EPEna=1), the core fetches the descriptor and processes it.
3. The DMA transfers the data from the internal FIFO to system memory.
4. After transferring all the data from the FIFO, the core closes the descriptor with a DMA_DONE status.
5. After the last descriptor in the chain is processed, the core generates DOEPINTx.XferCompl interrupt provided the IOC bit for the last descriptor is set.

16.7.10 Interrupt IN Data Transaction in Scatter/Gather DMA Mode

Application programming for Interrupt IN transfers is as with the Bulk IN transfer sequence. The core handles Interrupt IN transfers internally in the same way it handles Bulk IN transfers

16.7.11 Interrupt OUT Transfer

Application programming for Interrupt OUT transfers is as with the Bulk OUT transfer sequence. The core handles Interrupt OUT transfers internally in the same way it handles Bulk OUT Transfers

16.7.12 Isochronous IN Transfer

The application programming for Isochronous IN transfers is in the same manner as Bulk IN transfer sequence.

The following behavior is of importance while working with Isochronous IN end points

DCTL.IgnrFrmNum = $\mathbf{1_B}$

The way the core handles Isochronous IN transfers internally in the same way as it handles Bulk IN Transfers.

DCTL.IgnrFrmNum = 0_B

The core closes the descriptor and clears the corresponding fetched data in the FIFO if the USB frame number to which the descriptor belongs is elapsed.

16.7.12.1 Isochronous Transfers in Scatter/Gather (Descriptor DMA) Mode

This topic includes descriptions of both isochronous IN and OUT transfers

Isochronous IN

In the case of ISO IN After descriptor is fetched, the frame number field M is compared with current USB frame number N.

If the frame number in the fetched descriptor is already elapsed (M<N) then the descriptor is closed with status changed to DMA Done.

If the frame number in the fetched descriptor is for future (N>M+1) then the descriptor is left untouched. The Core suspends and re-look at this descriptor contents in the next frame.

- If the frame number in the fetched descriptor is for current or next frame (N=M or M+1) then the descriptor is further processed as per the flow chart. At the end of data transfer from memory to TxFIFO the above check must be performed. And if the data fetch finished in the subsequent frame, data must be flushed and descriptor must be closed (DMA Done) with BUFFLUSH status.
- For ISO IN, the application creates a series of descriptors (D,D+1,D+2) for a given periodic end point corresponding to successive frames (N,N+1,N+2).

Note: The series of descriptors does not correspond to the series of frames in the same order.

For example, D and D + 1 may correspond to N, D + 2 may correspond to N + 1 and so on except in the case where the application can create more than one descriptor for the same frame. The core fetches the descriptor and compares the frame/ ^frame number field with the current frame/ ^frame number.

If the fetched descriptor corresponds to a frame which has already elapsed, the core updates the descriptor with DMA Done Buffer status and proceeds to the next descriptor.

If the next descriptor fetched indicates that it corresponds to frame number N or N + 1, it services it. In the process of fetching the descriptors, if the core determines that the descriptor corresponds to a future frame/ ^.frame (> N + 1), it does not service the descriptor in that frame/^frame. Instead, it moves on to the next periodic endpoint or non-periodic endpoint without disabling the current periodic endpoint. It revisits this endpoint in the next frame/ ^.frame and repeats the process.

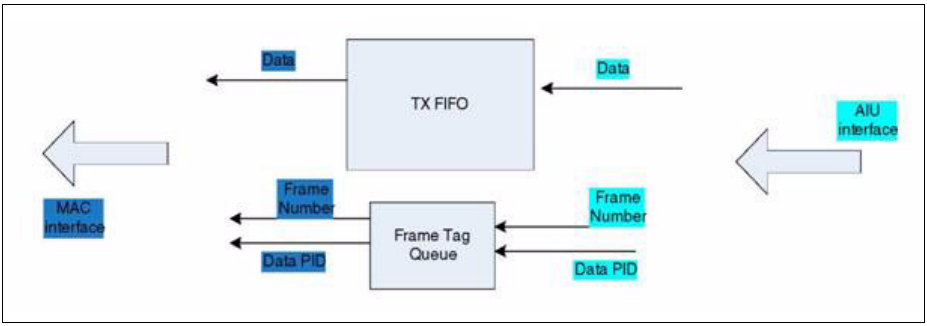


Figure 16-55 ISO IN Data Flow

Application Programming Sequence

This section describes the application programming sequence for Isochronous IN transfer scenarios.

Prepare Descriptor(s)

The application creates descriptor list(s) in the system memory pertaining to an Endpoint.

Each descriptor list may have up to n descriptors and there may be up to m descriptor lists.

Application may choose to set the IOC bit of the corresponding descriptor. If the IOC is set for the last descriptor of the list, the core generates DIEPINTx.XferCompl interrupt after the entire list is processed.

1. Program DIEPDMAx:
   a) Application programs the base address of the descriptor in the corresponding IN Endpoint DIEPDMAx register.
2. Enable DMA:
   a) Application programs the corresponding endpoint DIEPCTLx register with the following
      - DIEPCTLx.MPS — Max Packet size of the endpoint
      - DIEPCTLx.CNAK—Set to 1 to clear the NAK
      - DIEPCTLx.EPEna — Set to 1 to enable the DMA for the endpoint.

3. Wait for Interrupt:
   a) On reception of DIEPINTx.XferCompl, application must check the Buffer status and Tx Status field of the descriptor to ascertain that the descriptor closed normally.

DIEPINTx.BNA interrupt gets generated by the core when it encounters a descriptor in the list whose Buffer Status field is not Host Ready. In this case, the application is suppose to read the DIEPDMAx register to ascertain the address for which the BNA interrupt is asserted to take corrective action.

16.7.12.2 Internal Flow

The core handles isochronous IN transfers internally as functionally depicted in **Figure 16-56**. **Figure 16-57** also diagrams this flow.

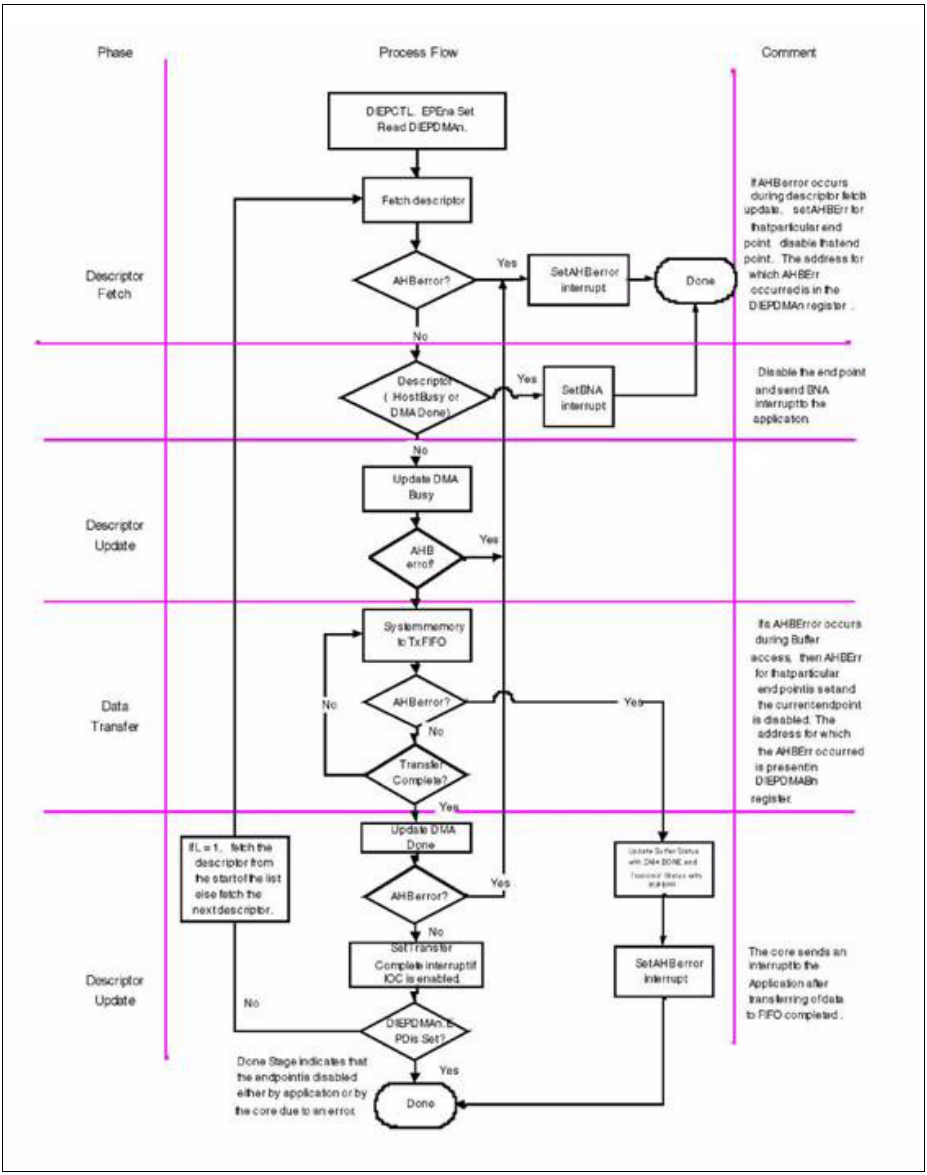


Figure 16-56 ISO IN Descriptor/Data Processing

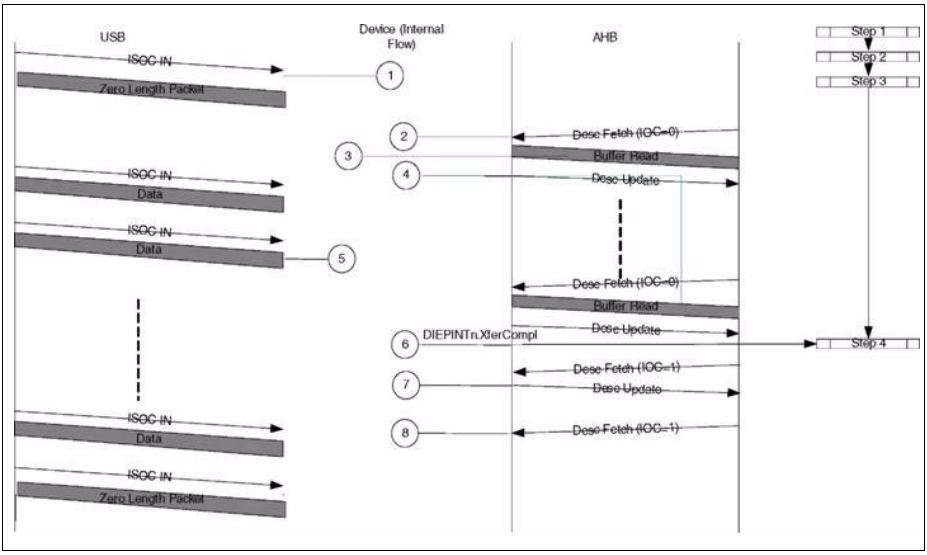


Figure 16-57 Isochronous IN Transfers

1. When an Isochronous IN token is received on an end point before the corresponding DMA is enabled, (DIEPCTLx.EPEna = 0_B), zero length packet is sent on USB.
2. As a result of application enabling the DMA for the corresponding end point (DIEPCTLx.EPEna=1), the core fetches the descriptor. If the descriptor belongs to the current or the next USB frame number, the core processes it.
3. The DMA fetches the data pointed by the above descriptor from the system memory and populates its internal FIFO with this data.
4. After fetching all the data, the core closes the descriptor with a DMA_DONE status.
5. On reception of Isochronous IN tokens on USB, data is sent to the USB Host.
6. After the last descriptor in the chain is processed, the core generates DIEPINTx.XferCompl interrupt provided the IOC bit for the last descriptor is set.
7. When the DMA fetches a descriptor whose USB frame number has been already elapsed, it closes that descriptor with a DMA_DONE status without fetching the data for that descriptor.
8. When the DMA fetches a descriptor which has a future USB frame number, it does not service it in the current context. It services it in the future.

16.7.13 Isochronous OUT Transfer

The application programming for isochronous out transfers is in the same manner as Bulk OUT transfer sequence, except that the application creates only 1 packet per descriptor for an isochronous OUT endpoint. The core handles isochronous OUT transfers internally in the same way it handles Bulk OUT transfers, and as depicted in **Figure 16-58**.

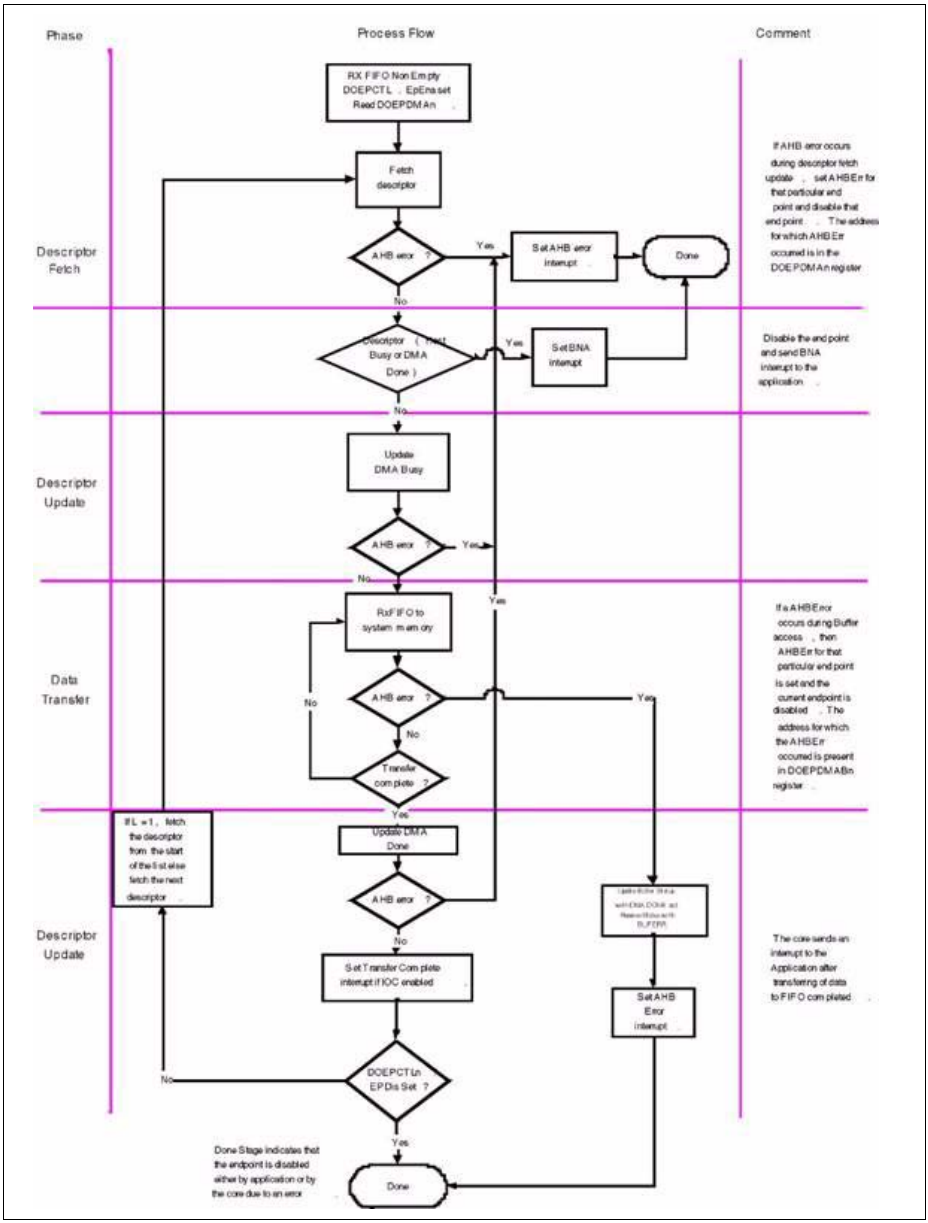


Figure 16-58 Isochronous OUT Descriptor/Data Buffer Processing

Isochronous OUT

For ISO OUT transactions, the core transfers the packets from the Rx FIFO to the system memory and updates the frame number field of the descriptor with the frame number in which the packet was received.The frame number for which data is received is extracted from the Receive Status queue and written back to the descriptor.

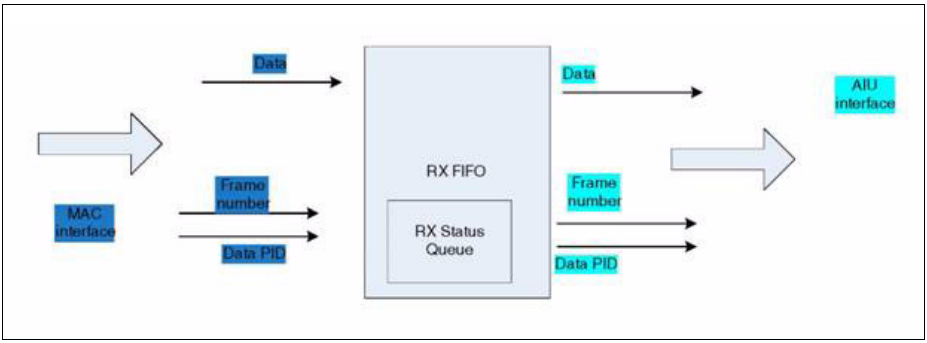


Figure 16-59 ISO Out Data Flow

Note: Incomplete Isochronous Interrupt (GINTSTS.incomplete) is not generated in Scatter/Gather DMA mode. Received isochronous packets are sent unmodified to the application memory, with the corresponding frame number updated in the descriptor status.

16.8 OTG Revision 1.3 Programming Model

This section describes the OTG programming model when the OTG core is configured to support OTG Revision 1.3 of the specification.

The USB core is an OTG device supporting HNP and SRP. When the core is connected to an "A" plug, it is referred to as an A-device. When the core is connected to a "B" plug it is referred to as a B-device. In Host mode, the USB core turns off VBUS to conserve power. SRP is a method by which the B-device signals the A-device to turn on VBUS power. A device must perform both data-line pulsing and VBUS pulsing, but a host can detect either data-line pulsing or VBUS pulsing for SRP. HNP is a method by which the B-device negotiates and switches to host role. In Negotiated mode after HNP, the B-device suspends the bus and reverts to the device role.

16.8.1 A-Device Session Request Protocol

The application must set the SRP-Capable bit in the Core USB Configuration register. This enables the USB core to detect SRP as an A-device.

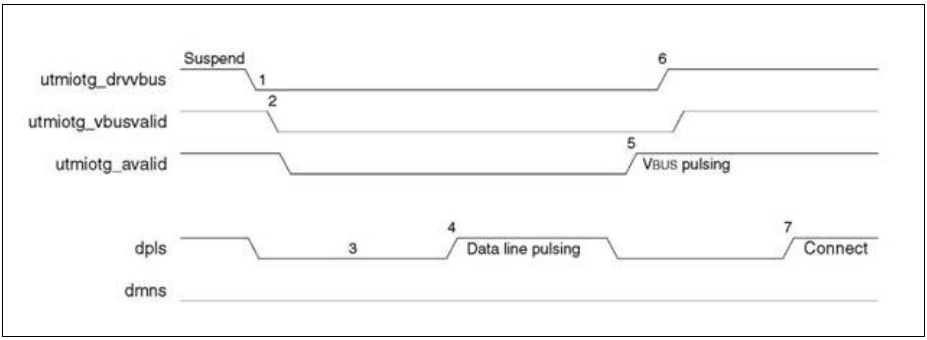


Figure 16-60 A-Device SRP

1. To save power, the application suspends and turns off port power when the bus is idle by writing the port Suspend and Port Power bits in the Host Port Control and Status register.
2. PHY indicates port power off by deasserting the utmi_vbusvalid signal.
3. The device must detect SE0 for at least 2 ms to start SRP when Vbus power is off.
4. To initiate SRP, the device turns on its data line pull-up resistor for 5 to 10 ms. The USB core detects data-line pulsing.
5. The device drives VBUS above the A-device session valid (2.0 V minimum) for VBUS pulsing.
   The USB core interrupts the application on detecting SRP. The Session Request Detected bit is set in Global Interrupt Status register (GINTSTS.SessReqInt).
6. The application must service the Session Request Detected interrupt and turn on the Port Power bit by writing the Port Power bit in the Host Port Control and Status register. The PHY indicates port power-on by asserting utmi_vbusvalid signal.
7. When the USB is powered, the device connects, completing the SRP process.

16.8.2 B-Device Session Request Protocol

The application must set the SRP-Capable bit in the Core USB Configuration register. This enables the USB core to initiate SRP as a B-device. SRP is a means by which the USB core can request a new session from the host.

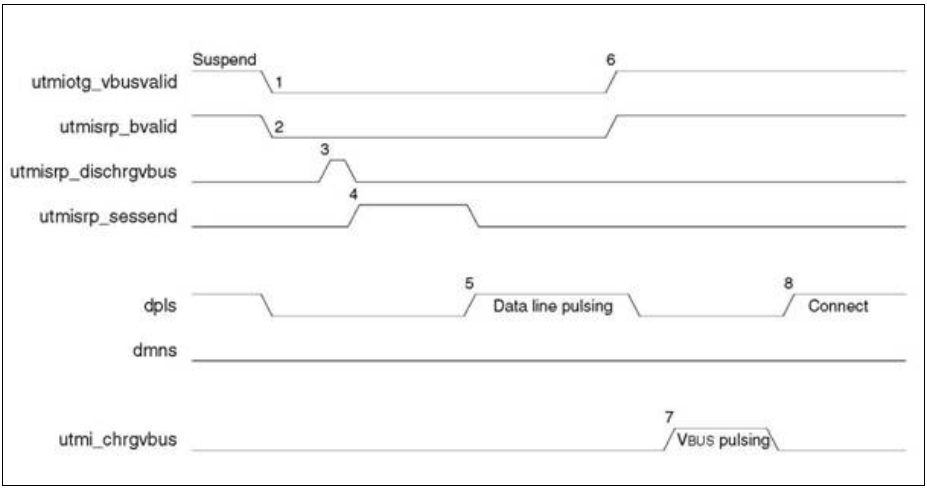


Figure 16-61 B-Device SRP

1. To save power, the host suspends and turns off port power when the bus is idle. PHY indicates port power off by deasserting the utmi_vbusvalid signal.The USB core sets the Early Suspend bit in the Core Interrupt register after 3 ms of bus idleness. Following this, the USB core sets the USB Suspend bit in the Core Interrupt register.The PHY indicates the end of the B-device session by deasserting the utmi_bvalid signal.
2. The USB core asserts the utmi_dischrgvbus signal to indicate to the PHY to speed up VBUS discharge.
3. The PHY indicates the session's end by asserting the utmi_sessend signal. This is the initial condition for SRP. The USB core requires 2 ms of SE0 before initiating SRP.
   For a USB 1.1 full-speed serial transceiver, the application must wait until VBUS discharges to 0.2 V after GOTGCTL.BSesVld is deasserted.
4. The application initiates SRP by writing the Session Request bit in the OTG Control and Status register. The USB core perform data-line pulsing followed by VBUS pulsing.
5. The host detects SRP from either the data-line or VBUS pulsing, and turns on VBUS. The PHY indicates VBUS power-on by asserting utmi_vbusvalid.
6. The USB core performs VBUS pulsing by asserting utmi_chrgvbus.
   The host starts a new session by turning on VBUS, indicating SRP success. The USB core interrupts the application by setting the Session Request Success Status Change bit in the OTG Interrupt Status register. The application reads the Session Request Success bit in the OTG Control and Status register.
7. When the USB is powered, the USB core connects, completing the SRP process.

16.8.3 A-Device Host Negotiation Protocol

HNP switches the USB host role from the A-device to the B-device. The application must set the HNP- Capable bit in the Core USB Configuration register to enable the USB core to perform HNP as an A-device.

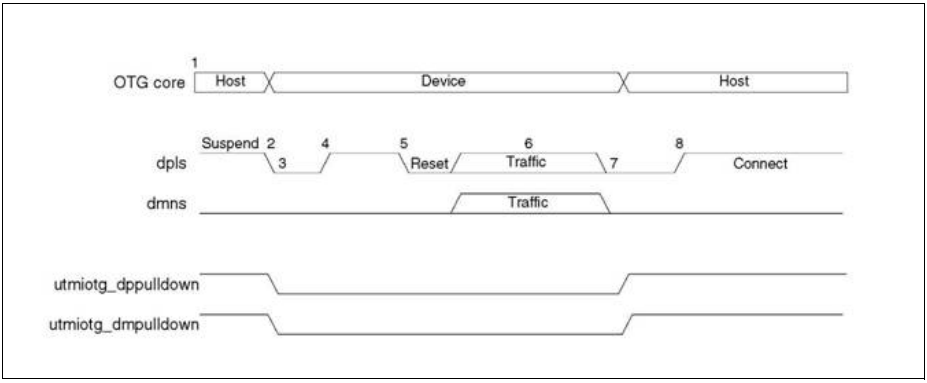


Figure 16-62 A-Device HNP

1. The USB core sends the B-device a SetFeature b_hnp_enable descriptor to enable HNP support. The B-device's ACK response indicates that the B-device supports HNP. The application must set Host Set HNP Enable bit in the OTG Control and Status register to indicate to the USB core that the B-device supports HNP.
2. When it has finished using the bus, the application suspends by writing the Port Suspend bit in the Host Port Control and Status register.
3. When the B-device observes a USB suspend, it disconnects, indicating the initial condition for HNP. The B-device initiates HNP only when it must switch to the host role; otherwise, the bus continues to be suspended.
   The USB core sets the Host Negotiation Detected interrupt in the OTG Interrupt Status register, indicating the start of HNP.
   The USB core deasserts the utmiotg_dppulldown and utmiotg_dmpulldown signals to indicate a device role. The PHY enable the D+ pull-up resistor indicates a connect for B-device.
   The application must read the Current Mode bit in the OTG Control and Status register to determine Device mode operation.
4. The B-device detects the connection, issues a USB reset, and enumerates the USB core for data traffic.
5. The B-device continues the host role, initiating traffic, and suspends the bus when done.
   The USB core sets the Early Suspend bit in the Core Interrupt register after 3 ms of bus idleness. Following this, the USB core sets the USB Suspend bit in the Core Interrupt register.

6. In Negotiated mode, the USB core detects the suspend, disconnects, and switches back to the host role. The USB core asserts the utmiotg_dppulldown and utmiotg_dmpulldown signals to indicate its assumption of the host role.
7. The USB core sets the Connector ID Status Change interrupt in the OTG Interrupt Status register. The application must read the connector ID status in the OTG Control and Status register to determine the USB core's operation as an A-device. This indicates the completion of HNP to the application. The application must read the Current Mode bit in the OTG Control and Status register to determine Host mode operation.
8. The B-device connects, completing the HNP process.

16.8.4 B-Device Host Negotiation Protocol

HNP switches the USB host role from B-device to A-device. The application must set the HNP-Capable bit in the Core USB Configuration register to enable the USB core to perform HNP as a B-device.

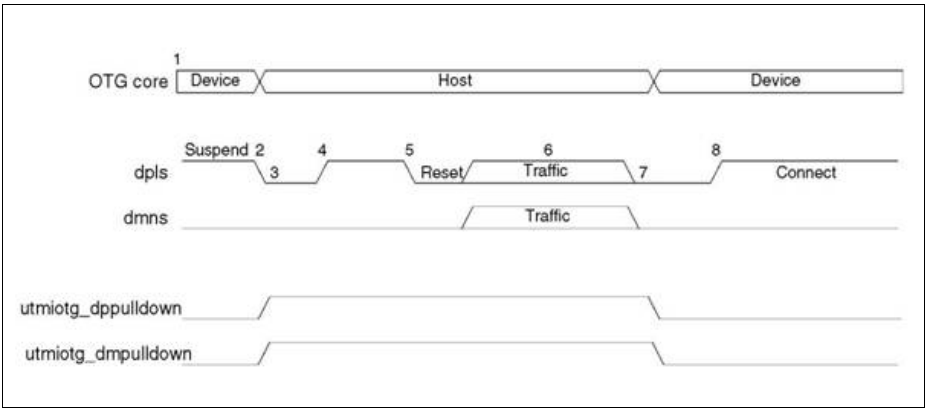


Figure 16-63 B-Device HNP

1. The A-device sends the SetFeature b_hnp_enable descriptor to enable HNP support. The USB core's ACK response indicates that it supports HNP. The application must set the Device HNP Enable bit in the OTG Control and Status register to indicate HNP support.
   The application sets the HNP Request bit in the OTG Control and Status register to indicate to the USB core to initiate HNP.
2. When it has finished using the bus, the A-device suspends by writing the Port Suspend bit in the Host Port Control and Status register.
   The USB core sets the Early Suspend bit in the Core Interrupt register after 3 ms of bus idleness. Following this, the USB core sets the USB Suspend bit in the Core Interrupt register.

The USB core disconnects and the A-device detects SE0 on the bus, indicating HNP.
The USB core asserts the utmiotg_dppulldown and utmiotg_dmpulldown signals to indicate its assumption of the host role.
The A-device responds by activating its D+ pull-up resistor within 3 ms of detecting SE0. The USB core detects this as a connect.
The USB core sets the Host Negotiation Success Status Change interrupt in the OTG Interrupt Status register, indicating the HNP status. The application must read the Host Negotiation Success bit in the OTG Control and Status register to determine host negotiation success. The application must read the Current Mode bit in the Core Interrupt register (GINTSTS) to determine Host mode operation.

3. The application sets the reset bit (HPRT.PrtRst) and the USB core issues a USB reset and enumerates the A-device for data traffic
4. The USB core continues the host role of initiating traffic, and when done, suspends the bus by writing the Port Suspend bit in the Host Port Control and Status register.
5. In Negotiated mode, when the A-device detects a suspend, it disconnects and switches back to the host role. The USB core deasserts the utmiotg_dppulldown and utmiotg_dmpulldown signals to indicate the assumption of the device role.
6. The application must read the Current Mode bit in the Core Interrupt (GINTSTS) register to determine the Host mode operation.
7. The USB core connects, completing the HNP process.

16.9 Clock Gating Programming Model

When the USB is suspended or the session is not valid, the PHY is driven into Suspend mode, and the PHY clock is stopped to reduce power consumption in the PHY and the USB core. The PHY clock is turned off for as long as the core asserts the suspend signal.

To further reduce power consumption, the USB core also supports AHB clock gating. The AHB clock to some of the USB internal modules can be gated by writing to the Gate Hclk bit in the Power and Clock Gating Control register.

The following sections show the procedures to use the clock gating feature.

16.9.1 Host Mode Suspend and Resume With Clock Gating

Sequence of operations:

1. The application sets the Port Suspend bit in the Host Port Control and Signal register, and the core drives a USB suspend.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the core asserts the suspend signal to the PHY, and the PHY clock stops. The application sets the Gate hclk bit in the Power and Clock Gating Control register, the core gates the hclk (hclk_gated) to AHB- domain modules other than the BIU.
3. The core remains in Suspend mode.
4. The application clears the Gate hclk and Stop PHY Clock bits, and the PHY clock is generated.

5. The application sets the Port Resume bit, and the core starts driving Resume signaling.
6. The application clears the Port Resume bit after at least 20 ms.
7. The core is in normal operating mode.

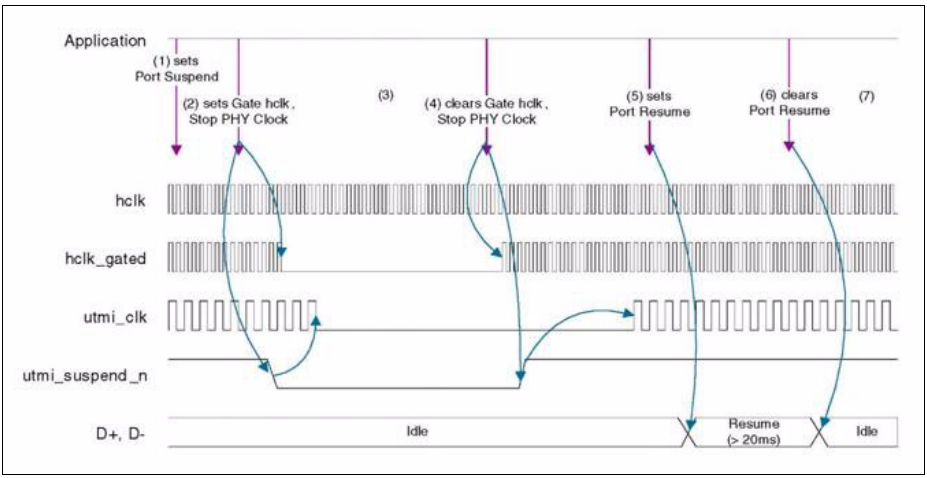


Figure 16-64 Host Mode Suspend and Resume With Clock Gating

16.9.2 Host Mode Suspend and Remote Wakeup With Clock Gating

Sequence of operations:

1. The application sets the Port Suspend bit in the Host Port CSR, and the core drives a USB suspend.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the core asserts the suspend_n signal to the PHY, and the PHY clock stops. The application sets the Gate hclk bit in the Power and Clock Gating Control register, and the core gates the hclk (hclk_ctl) to AHB- domain modules other than the BIU.
3. The core remains in Suspend mode.
4. The Remote Wakeup signaling from the device is detected. The core deasserts the suspend_n signal to the PHY to generate the PHY clock. The core generates a Remote Wakeup Detected interrupt.
5. The application clears the Gate hclk and Stop PHY Clock bits. The core sets the Port Resume bit.
6. The application clears the Port Resume bit after at least 20 ms.
7. The core is in normal operating mode.

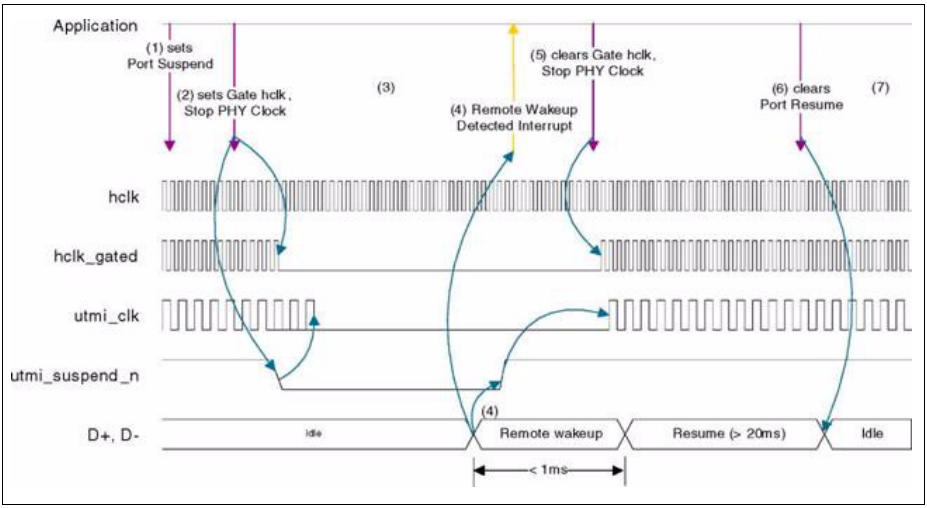


Figure 16-65 Host Mode Suspend and Remote Wakeup With Clock Gating

16.9.3 Host Mode Session End and Start With Clock Gating

Sequence of operations:

1. The application sets the Port Suspend bit in the Host Port CSR, and the core drives a USB suspend.
2. The application clears the Port Power bit. The core turns off VBUS.
3. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the core asserts the suspend_n signal to the PHY, and the PHY clock stops. The application sets the Gate hclk bit in the Power and Clock Gating Control register, and the core gates the hclk (hclk_ctl) to AHB- domain modules other than the BIU.
4. The core remains in Low-Power mode.
5. The application clears the Gate hclk bit and the application clears the Stop PHY Clock bit to start the PHY clock.
6. The application sets the Port Power bit to turn on VBUS.
7. The core detects device connection and drives a USB reset.
8. The core is in normal operating mode.

16.9.4 Host Mode Session End and SRP With Clock Gating

Sequence of operations:

1. The application sets the Port Suspend bit in the Host Port CSR, and the core drives a USB suspend.
2. The application clears the Port Power bit. The core turns off VBUS.

3. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the core asserts the suspend_n signal to the PHY, and the PHY clock stops. The application sets the Gate hclk bit in the Power and Clock Gating Control register, and the core gates the hclk (hclk_ctl) to AHB- domain modules other than the BIU.
4. The core remains in Low-Power mode.
5. SRP (data line pulsing) from the device is detected. The core deasserts the suspend_n signal to the PHY to generate the PHY clock. An SRP Request Detected interrupt is generated.
6. The application clears the Gate hclk bit and the Stop PHY Clock bit.
7. The core sets the Port Power bit to turn on VBUS.
8. The core detects device connection and drives a USB reset.
9. The core is in normal operating mode.

16.9.5 Device Mode Suspend and Resume With Clock Gating

Sequence of operations:

1. The core detects a USB suspend and generates a Suspend Detected interrupt.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the core asserts the suspend_n signal to the PHY, and the PHY clock stops. The application sets the Gate hclk bit in the Power and Clock Gating Control register, and the core gates the hclk (hclk_ctl) to AHB- domain modules other than the BIU.
3. The core remains in Suspend mode.
4. The Resume signaling from the host is detected. The core deasserts the suspend_n signal to the PHY to generate the PHY clock. A Resume Detected interrupt is generated.
5. The application clears the Gate hclk bit and the Stop PHY Clock bit.
6. The host finishes Resume signaling.
7. The core is in normal operating mode.

16.9.6 Device Mode Suspend and Remote Wakeup With Clock Gating

Sequence of operations:

1. The core detects a USB suspend and generates a Suspend Detected interrupt.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the core asserts the suspend_n signal to the PHY, and the PHY clock stops. The application sets the Gate hclk bit in the Power and Clock Gating Control register, the core gates the hclk (hclk_ctl) to AHB-domain modules other than the BIU.
3. The core remains in Suspend mode.
4. The application clears the Gate hclk bit and the Stop PHY Clock bit.
5. The application sets the Remote Wakeup bit in the Device Control register, the core starts driving Remote Wakeup signaling.
6. The host drives Resume signaling.
7. The core is in normal operating mode.

16.9.7 Device Mode Session End and Start With Clock Gating

Sequence of operations:

1. The core detects a USB suspend, and generates a Suspend Detected interrupt. The host turns off VBUS.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the core asserts the suspend_n signal to the PHY, and the PHY clock stops. The application sets the Gate hclk bit in the Power and Clock Gating Control register, and the core gates the hclk (hclk_ctl) to AHB- domain modules other than the BIU.
3. The core remains in Low-Power mode.
4. The new session is detected (bsessvld is high). The core deasserts the suspend_n signal to the PHY to generate the PHY clock. A New Session Detected interrupt is generated.
5. The application clears the Gate hclk and Stop PHY Clock bits.
6. The core detects USB reset.
7. The core is in normal operating mode

16.9.8 Device Mode Session End and SRP With Clock Gating

Sequence of operations:

1. The core detects a USB suspend, and generates a Suspend Detected interrupt. The host turns off VBUS.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the core asserts the suspend_n signal to the PHY, and the PHY clock stops. The application sets the Gate hclk bit in the Power and Clock Gating Control register, and the core gates the hclk (hclk_ctl) to AHB- domain modules other than the BIU.
3. The core remains in Low-Power mode.
4. The application clears the Gate hclk and Stop PHY Clock bits.
5. The application sets the SRP Request bit, and the core drives data line and VBUS pulsing.
6. The host turns on Vbus, detects device connection, and drives a USB reset.
7. The core is in normal operating mode.

16.10 FIFO RAM Allocation

16.10.1 Data FIFO RAM Allocation

The RAM must be allocated among different FIFOs in the core before any transactions can start. The application must follow this procedure every time it changes core FIFO RAM allocation.

The application must allocate data RAM per FIFO based on the AHB's operating frequency, the PHY Clock frequency, the available AHB bandwidth, and the performance

required on the USB. Based on the above mentioned criteria, the application must provide a table as described below with RAM sizes for each FIFO in each mode.

USB core shares a single SPRAM between transmit FIFO(s) and receive FIFO.

In DMA mode — The SPRAM is also used for storing some register information.

In non Scatter Gather mode — The Device mode Endpoint DMA address registers (DI/OEPDMAn) and Host mode Channel DMA registers (HCDMA) are stored in the SPRAM.

In Scatter Gather DMA mode—The Base descriptor address, the Current descriptor address, the current buffer address and the descriptor status quadlet information for each endpoint/channel are stored in the SPRAM.

- These register information are stored at the end of the SPRAM after the space allocated for receive and Transmit FIFO. These register space must also be taken into account when calculating the total FIFO depth of the core as explained in the following sections.

In addition, the registers DIEPDMAx/DOEPDMAx are maintained in RAM regardless of the enabled/disabled setting for Dynamic FIFO Sizing.

The following rules apply while calculating how much RAM space must be allocated to store these registers.

Host Mode

- Slave mode only: No space needed.
- Buffer DMA mode: One location per channel.
- Scatter/Gather DMA mode: Four locations per channel as listed below.
  - Location for storing current descriptor address.
  - Location for storing current buffer address.
  - Location for storing the status quadlet that is used by the List processor
  - Location for storing the transfer size used by the token request block

For example in Scatter/Gather DMA mode, if there are ten channels, then the last forty SPRAM locations are reserved for storing these values.

Device Mode

- Slave mode only: No space needed.
- Buffer DMA mode: One location per end point direction.
- Scatter/Gather DMA mode: Four locations per endpoint direction:
  – Location for storing base descriptor address.
  – Location for storing current descriptor address.
  – Location for storing the current buffer address.
  – Location to store the descriptor status quadlet

For example in Scatter/Gather DMA mode, if there are five bidirectional endpoints, then the last forty SPRAM locations are reserved for storing these values.

16.10.1.1 Device Mode RAM Allocation

When allocating data RAM for FIFOs in Device mode when dedicated TX FIFO is used, keep in mind these factors:

1. Receive FIFO RAM Allocation:
   a) RAM for SETUP Packets: 4 * n + 6 locations must be Reserved in the receive FIFO to receive up to n SETUP packets on control endpoints, where n is the number of control endpoints the device core supports. The core does not use these locations, which are Reserved for SETUP packets, to write any other data.
   b) One location for Global OUT NAK
   c) Status information is written to the FIFO along with each received packet. Therefore, a minimum space of (Largest Packet Size / 4) + 1 must be allotted to receive packets. If a high-bandwidth endpoint is enabled, or multiple isochronous endpoints are enabled, then at least two (Largest Packet Size / 4) + 1 spaces must be allotted to receive back-to-back packets. Typically, two (Largest Packet Size / 4) + 1 spaces are recommended so that when the previous packet is being transferred to AHB, the USB can receive the subsequent packet. If AHB latency is high, enough space must be allocated to receive multiple packets. This is critical to prevent dropping any isochronous packets.
   d) Along with each endpoint's last packet, transfer complete status information is also pushed to the FIFO. Typically, one location for each OUT endpoint is recommended.
2. Transmit FIFO RAM Allocation:
   a) The minimum RAM space required for each IN Endpoint Transmit FIFO is the maximum packet size for that particular IN endpoint.
   b) More space allocated in the transmit IN Endpoint FIFO results in a better performance on the USB and can hide latencies on the AHB.

Table 16-16 FIFO Name - Data RAM Size 2

FIFO Name	Data RAM Size
Receive data FIFO	rx_fifo_size. This must include RAM for setup packets, OUT endpoint control information and data OUT packets, as mentioned earlier.
Transmit FIFO 0	tx_fifo_size[0]
Transmit FIFO 1	tx_fifo_size[1]
Transmit FIFO 2	tx_fifo_size[2]
...	...
Transmit FIFO i	tx_fifo_size[i]

With this information, the following registers must be programmed as follows:

1. Receive FIFO Size Register (GRXFSIZ)
   GRXFSIZ.Receive FIFO Depth = rx_fifo_size;
2. Device IN Endpoint Transmit FIFO0 Size Register (GNPTXFSIZ)
   GNPTXFSIZ.non-periodic Transmit FIFO Depth = tx_fifo_size[0];
   GNPTXFSIZ.non-periodic Transmit RAM Start Address = rx_fifo_size;
3. Device IN Endpoint Transmit FIFO#1 Size Register (DIEPTXF1)
   DIEPTXF1. Transmit RAM Start Address = GNPTXFSIZ.FIFO0 Transmit RAM Start Address + tx_fifo_size[0];
4. Device IN Endpoint Transmit FIFO#2 Size Register (DIEPTXF2)
   DIEPTXF2.Transmit RAM Start Address = DIEPTXF1.Transmit RAM Start Address + tx_fifo_size[1];
5. Device IN Endpoint Transmit FIFO#i Size Register (DIEPTXFi)
   DIEPTXFm.Transmit RAM Start Address = DIEPTXFi-1.Transmit RAM Start Address + tx_fifo_size[i-1];
6. The transmit FIFOs and receive FIFO must be flushed after the RAM allocation is done, for the proper functioning of the FIFOs.
   a) GRSTCTL.TxFNum = 10_H
   b) GRSTCTL.TxFFlush = 1_B
   c) GRSTCTL.RxFFlush = 1_B
   d) The application must wait until the TxFFlush bit and the RxFFlush bits are cleared before performing any operation on the core.

16.10.1.2 Host Mode RAM Allocation

Considerations for allocating data RAM for Host Mode FIFOs are listed here:

Receive FIFO RAM allocation:

Status information is written to the FIFO along with each received packet. Therefore, a minimum space of (Largest Packet Size / 4) + 2 must be allotted to receive packets. If a high-bandwidth channel is enabled, or multiple isochronous channels are enabled, then at least two (Largest Packet Size / 4) + 2 spaces must be allotted to receive back-to-back packets. Typically, two (Largest Packet Size / 4) + 2 spaces are recommended so that when the previous packet is being transferred to AHB, the USB can receive the subsequent packet. If AHB latency is high, enough space must be allocated to receive multiple packets.

Along with each host channel's last packet, information on transfer complete status and channel halted is also pushed to the FIFO. So two locations must be allocated for this.

For handling NAK/NYET in Buffer DMA mode, the application must determine the number of Control/Bulk OUT endpoint data that must fit into the TX_FIFO at the same instant. Based on this, one location each is required for Control/Bulk OUT endpoints.

For example, when the host addresses one Control OUT endpoint and three Bulk OUT endpoints, and all these must fit into the non-periodic TX_FIFO at the same time, then four extra locations are required in the RX FIFO to store the rewind status information for each of these endpoints.

Transmit FIFO RAM allocation

The minimum amount of RAM required for the Host Non-periodic Transmit FIFO is the largest maximum packet size among all supported non-periodic OUT channels.

More space allocated in the Transmit Non-periodic FIFO results in better performance on the USB and can hide AHB latencies. Typically, two Largest Packet Sizes' worth of space is recommended, so that when the current packet is under transfer to the USB, the AHB can get the next packet. If the AHB latency is large, then enough space must be allocated to buffer multiple packets.

The minimum amount of RAM required for Host periodic Transmit FIFO is the largest maximum packet size among all supported periodic OUT channels. If there is at lease one High Bandwidth Isochronous OUT endpoint, then the space must be at least two times the maximum packet size of that channel.

Internal Register Storage Space Allocation

When operating in Buffer DMA mode, the DMA address register for each host channel (HCDMAx) is stored in the SPRAM. One location for each channel must be reserved for this.

When operating in Scatter/Gather DMA mode, four locations per channel must be reserved.

Table 16-17 FIFO Name - Data RAM Size 3

FIFO Name	Data RAM Size
Receive Data FIFO	rx_fifo_size
Non-periodic Transmit FIFO	tx_fifo_size[0]
IN Endpoint Transmit FIFO	tx_fifo_size[1]

With this information, the following registers must be programmed:

1. Receive FIFO Size Register (GRXFSIZ)
   a) `GRXFSIZ.RxFDep= rx_fifo_size;`
2. Non-periodic Transmit FIFO Size Register (GNPTXFSIZ)
   a) `GNPTXFSIZ.NPTxFDe=tx_fifo_size[0];`
   b) `GNPTXFSIZ.NPTxFStAddr = rx_fifo_size;`
3. Host Periodic Transmit FIFO Size Register (HPTXFSIZ)
   a) `HPTXFSIZ.PTxFSize = tx_fifo_size[1];`
   b) `HPTXFSIZ.PTxFStAddr= GNPTXFSIZ.NPTxFStAddr + tx_fifo_size[0];`
4. The transmit FIFOs and receive FIFO must be flushed after RAM allocation for proper FIFO function.
   a) GRSTCTL.TxFNum = 10_H
   b) GRSTCTL.TxFFlush = 1_B
   c) GRSTCTL.RxFFlush = 1_B
   d) The application must wait until the TxFFlush bit and the RxFFlush bits are cleared before performing any operation on the core.

16.10.2 Dynamic FIFO Allocation

The application can change the RAM allocation for each FIFO during the operation of the core.

16.10.2.1 Host Mode

In Host mode, before changing FIFO data RAM allocation, the application must determine the following. All channels are disabled

- All FIFOs are empty

Once these conditions are met, the application can reallocate FIFO data RAM as explained in **“Data FIFO RAM Allocation” on Page 16-201**.

After reallocating the FIFO data RAM, the application must flush all FIFOs in the core using the GRSTCTL.TxFIFO Flush and GRSTCTL.RxFIFO Flush fields. Flushing is required to reset the pointers in the FIFOs for proper FIFO operation after reallocation.

16.10.2.2 Device Mode

In Device mode, before changing FIFO data RAM allocation, the application must determine the following.

- All IN and OUT endpoints are disabled
- NAK mode is enabled in the core on all IN endpoints
- Global OUT NAK mode is enabled in the core
- All FIFOs are empty

Once these conditions are met, the application can reallocate FIFO data RAM as explained in **“Data FIFO RAM Allocation” on Page 16-201**. When NAK mode is enabled in the core, the core responds with a NAK handshake on all tokens received on the USB, except for SETUP packets.

After the reallocating the FIFO data RAM, the application must flush all FIFOs in the core using the GRSTCTL.TxFIFO Flush and GRSTCTL.RxFIFO Flush fields. Flushing is required to reset the pointers in the FIFOs for proper FIFO operation after reallocation.

The Core Interrupt Handler

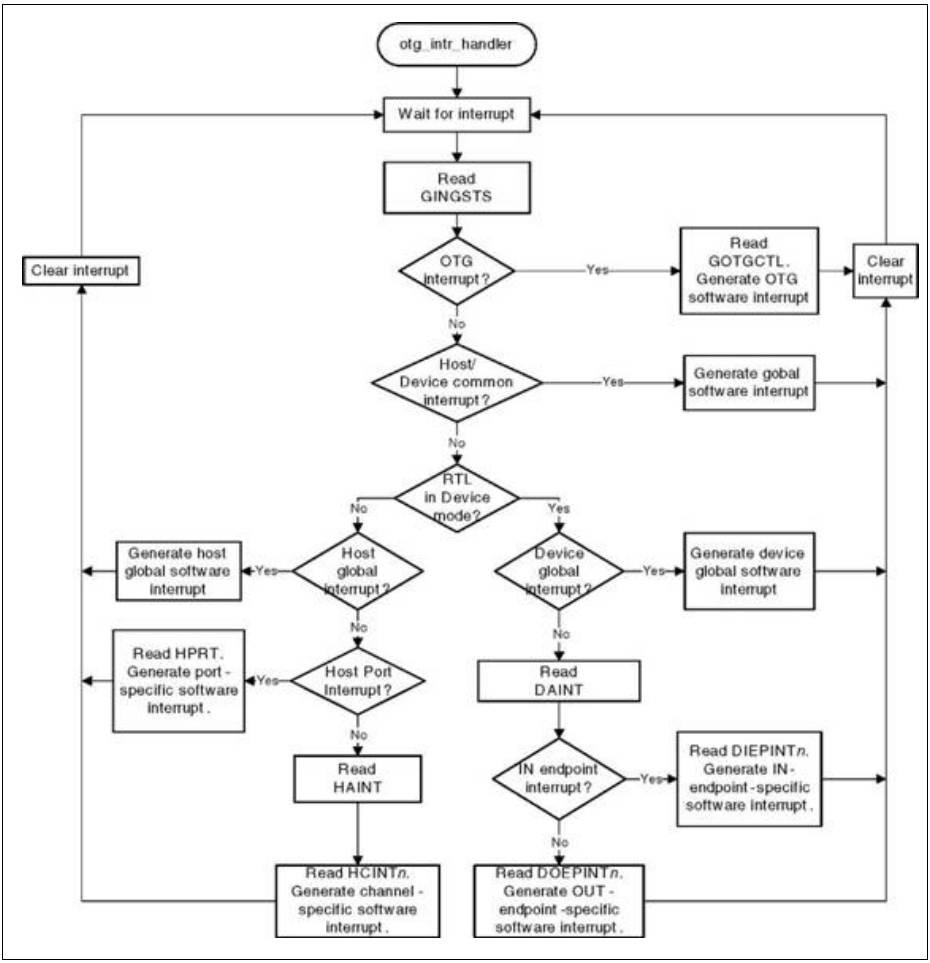


Figure 16-66 Core Interrupt Handler

16.11 Service Request Generation

The USB module provides a single service request output connected to an interrupt node in the Nested Vectored Interrupt Controller (NVIC)

Figure 16-67 displays the USB core interrupt hierarchy.

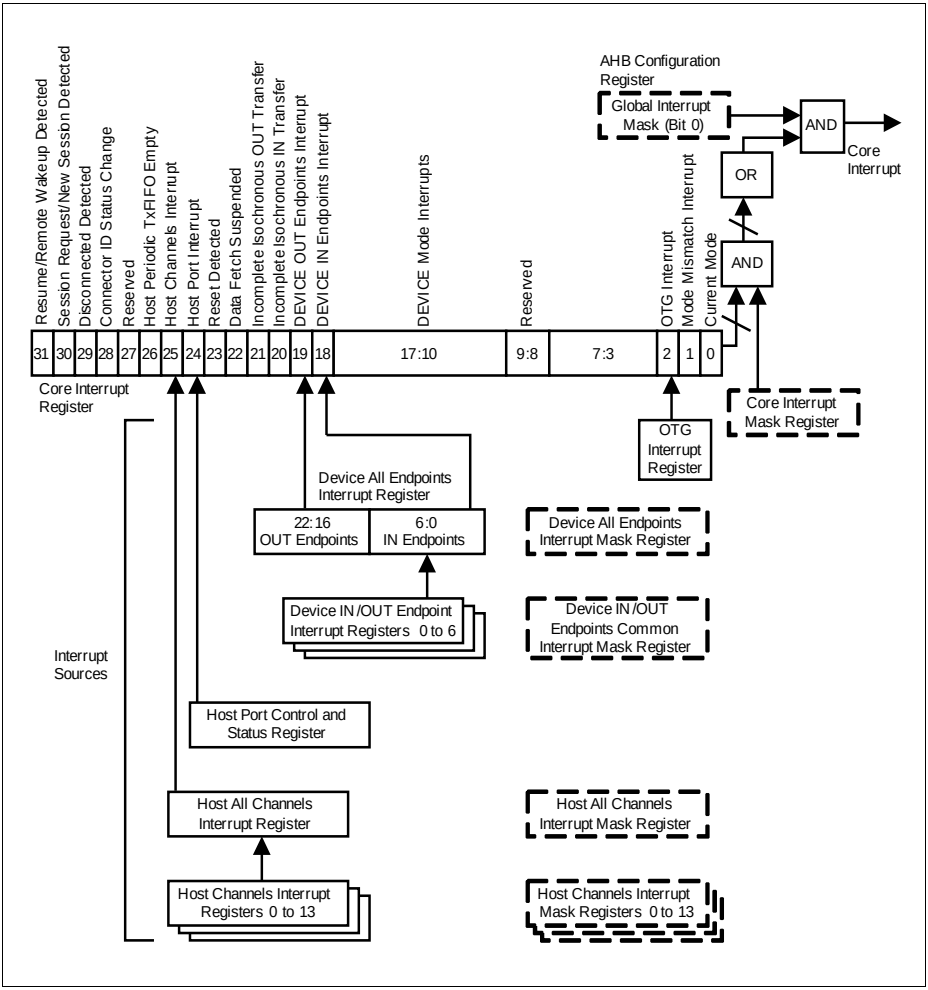


Figure 16-67 Interrupt Hierarchy

16.12 Debug Behaviour

The USB module is not affected when the CPU enters HALT mode.

16.13 Power, Reset and Clock

When the USB module is programmed as a host, an external charge pump is required to drive the VBUS.

The module, including all registers, can be reset to its default state by a system reset or a software reset triggered through the setting of corresponding bits in PRSETx registers.

The module has the following input clocks:

- clk_ahbm: the module clock, which is also referred to as hclk in this chapter
- clk_usb: the 48 MHz PHY clock., which is also referred to as phy_clk in this chapter

In addition, the module internally generates:

- hclk_gated: hclk gated for power optimization

16.14 Initialization and System Dependencies

The USB core is held in reset after a start-up from a system or software reset. The USB PHY is also by default in the power-down state. Therefore, the application has to apply the following initialization sequence before programming the USB core:

- Release reset of USB core by writing a 1 to the USBRS bit in SCU_PRCLR2 register
- Enable the 48 MHz PHY clock by configuring the USB PLL in SCU, see clock control section in SCU chapter
- Remove the USB PHY from power-down by writing 1s to the USBOTGEN and USBPHYPDQ bits in SCU_PWSET register

16.15 Registers

Register Overview

The application controls the USB core by reading from and writing to the Control and Status Registers (CSRs) through the AHB Slave interface. These registers are 32 bits wide and the addresses are 32-bit block aligned.

Only the Core Global, Power and Clock Gating, Data FIFO Access, and Host Port registers can be accessed in both Host and Device modes. When the USB core is operating in one mode, either Device or Host, the application must not access registers from the other mode. If an illegal access occurs, a Mode Mismatch interrupt is generated and reflected in the Core Interrupt register (GINTSTS.ModeMis).

When the core switches from one mode to another, the registers in the new mode must be reprogrammed as they would be after a power-on reset.

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 16-18 Registers Address Space

Module	Base Address	End Address	Note
USB0	$5004\ 0000_H$	$5007\ FFFF_H$	

Figure 16-68 shows the CSR address map. Host and Device mode registers occupy different addresses.

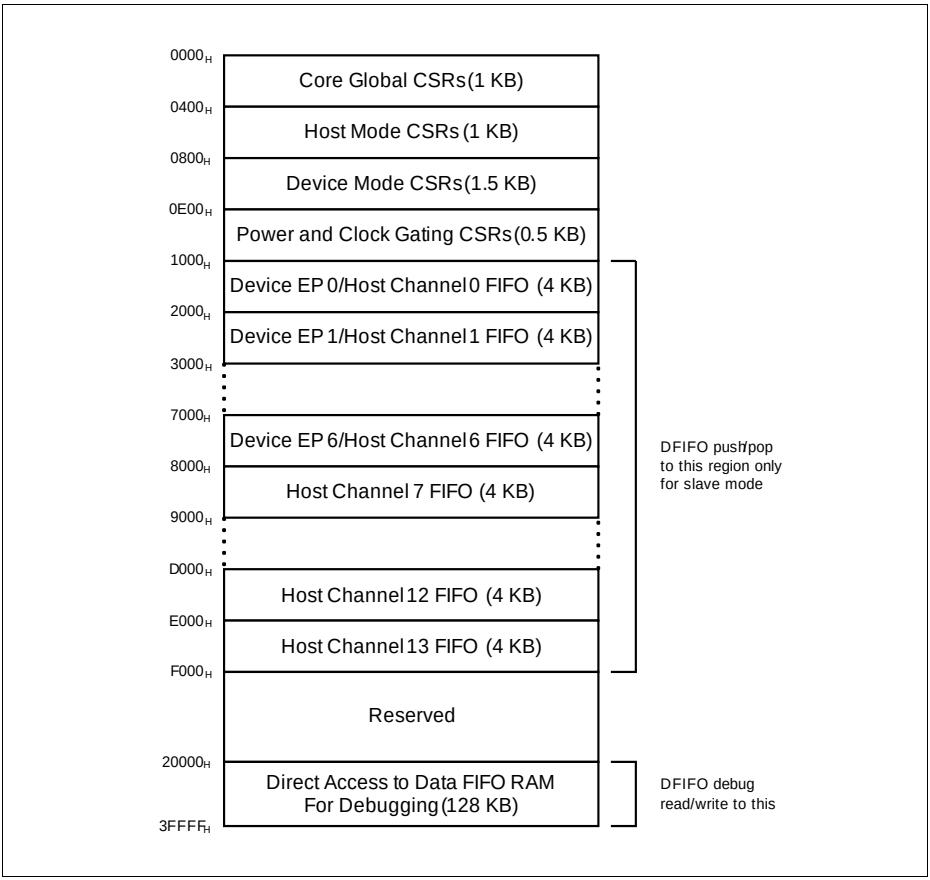


Figure 16-68 CSR Memory Map

The first letter of the register name is a prefix for the register type:

- G: Core Global
- H: Host mode
- D: Device mode

Note: FIFO size and FIFO depth are used interchangeably.

Table 16-19 Register Overview

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
USB Global Registers					
GOTGCTL	Control and Status Register	000_H	U, PV	U, PV	Page 16-218
GOTGINT	OTG Interrupt Register	004_H	U, PV	U, PV	Page 16-224
GAHBCFG	AHB Configuration Register	008_H	U, PV	U, PV	Page 16-226
GUSBCFG	USB Configuration Register	$00C_H$	U, PV	U, PV	Page 16-228
GRSTCTL	Reset Register	010_H	U, PV	U, PV	Page 16-231
GINTSTS	Interrupt Register	014_H	U, PV	U, PV	Page 16-235
GINTMSK	Interrupt Mask Register	018_H	U, PV	U, PV	Page 16-243
GRXSTSR	Receive Status Debug Read Register	$01C_H$	U, PV	U, PV	Page 16-246
GRXSTSP	Status Read and Pop Register	020_H	U, PV	U, PV	Page 16-246
GRXFSIZ	Receive FIFO Size Register	024_H	U, PV	U, PV	Page 16-250
GNPTXFSIZ	Non-Periodic Transmit FIFO Size Register	028_H	U, PV	U, PV	Page 16-251
GNPTXSTS	Non-Periodic Transmit FIFO/Queue Status Register	$02C_H$	U, PV	U, PV	Page 16-253
Reserved	Reserved	030_H-038_H	nBE	nBE	
GUID	User ID Register	$03C_H$	U, PV	U, PV	Page 16-255
Reserved	Reserved	040_H-058_H	nBE	nBE	
GDFIFOCFG	DFIFO Software Config Register	$05C_H$	U, PV	U, PV	Page 16-256
Reserved	Reserved	060_H-$0FC_H$	nBE	nBE	
HPTXFSIZ	Host Periodic Transmit FIFO Size Register	100_H	U, PV	U, PV	Page 16-257
DIEPTXFn	Device IN Endpoint Transmit FIFO Size Register	104_H-124_H	U, PV	U, PV	Page 16-258

Table 16-19 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
Reserved	Reserved	128_H-$3FF_H$	nBE	nBE	
USB Host Mode Registers					
HCFG	Host Configuration Register	400_H	U, PV	U, PV	**Page 16-258**
HFIR	Host Frame Interval Register	404_H	U, PV	U, PV	**Page 16-262**
HFNUM	Host Frame Number/Frame Time Remaining Register	408_H	U, PV	U, PV	**Page 16-263**
Reserved	Reserved	$40C_H$	nBE	nBE	
HPTXSTS	Host Periodic Transmit FIFO/Queue Status Register	410_H	U, PV	U, PV	**Page 16-264**
HAINT	Host All Channels Interrupt Register	414_H	U, PV	U, PV	**Page 16-266**
HAINTMSK	Host All Channels Interrupt Mask Register	418_H	U, PV	U, PV	**Page 16-267**
HFLBADDR	Host Frame List Base Address Register	$41C_H$	U, PV	U, PV	**Page 16-268**
Reserved	Reserved	420_H-$43C_H$	nBE	nBE	
HPRT	Host Port Control and Status Register	440_H	U, PV	U, PV	**Page 16-269**
Reserved	Reserved	444_H-$4FC_H$	nBE	nBE	
HCCHARx	Host Channel-n Characteristics Register	500_H + n*20	U, PV	U, PV	**Page 16-273**
Reserved	Reserved	504_H + n*20	nBE	nBE	
HCINTx	Host Channel-n Interrupt Register	508_H + n*20	U, PV	U, PV	**Page 16-276**
HCINTMSKx	Host Channel-n Interrupt Mask Register	$50C_H$ + n*20	U, PV	U, PV	**Page 16-279**
HCTSIZx	Host Channel-n Transfer Size Register	510_H + n*20	U, PV	U, PV	**Page 16-281**

Table 16-19 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
HCDMAx	Host Channel-n DMA Address Register	514_H + n*20	U, PV	U, PV	Page 16-284
Reserved	Reserved	518_H + n*20	nBE	nBE	
HCDMABx	Host Channel-n DMA Buffer Address Register	$51C_H$ + n*20	U, PV	U, PV	Page 16-288
Reserved	Reserved	780_H-$7FF_H$	nBE	nBE	
USB Device Mode Registers					
DCFG	Device Configuration Register	800_H	U, PV	U, PV	Page 16-289
DCTL	Device Control Register	804_H	U, PV	U, PV	Page 16-293
DSTS	Device Status Register	808_H	U, PV	U, PV	Page 16-297
Reserved	Reserved	$80C_H$	nBE	nBE	
DIEPMSK	Device IN Endpoint Common Interrupt Mask Register	810_H	U, PV	U, PV	Page 16-299
DOEPMSK	Device OUT Endpoint Common Interrupt Mask Register	814_H	U, PV	U, PV	Page 16-301
DAINT	Device All Endpoints Interrupt Register	818_H	U, PV	U, PV	Page 16-303
DAINTMSK	Device All Endpoints Interrupt Mask Register	$81C_H$	U, PV	U, PV	Page 16-304
Reserved	Reserved	820_H-824_H	nBE	nBE	
DVBUSDIS	Device VBUS Discharge Time Register	828_H	U, PV	U, PV	Page 16-305
DVBUSPULSE	Device VBUS Pulsing Time Register	$82C_H$	U, PV	U, PV	Page 16-306
Reserved	Reserved	830_H	nBE	nBE	
DIEPEMPMSK	Device IN Endpoint FIFO Empty Interrupt Mask Register	834_H	U, PV	U, PV	Page 16-307

Table 16-19 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
Reserved	Reserved	838_H-$8FC_H$	nBE	nBE	
DIEPCTL0	Device Control IN Endpoint 0 Control Register	900_H	U, PV	U, PV	Page 16-308
DIEPCTLx	Device Endpoint n Control Register	900_H + n*20_H	U, PV	U, PV	Page 16-314
Reserved	Reserved	904_H + n*20_H	nBE	nBE	
DIEPINTx	Device Endpoint-n Interrupt Register	908_H + n*20_H	U, PV	U, PV	Page 16-324
Reserved	Reserved	$90C_H$ + n*20_H	nBE	nBE	
DIEPTSIZ0	Device Endpoint 0 Transfer Size Register	910_H	U, PV	U, PV	Page 16-330
DIEPTSIZx	Device Endpoint-n Transfer Size Register	910_H + n*20_H	U, PV	U, PV	Page 16-332
DIEPDMAx	Device Endpoint-n DMA Address Register	914_H + n*20_H	U, PV	U, PV	Page 16-336
DTXFSTSx	Device IN Endpoint Transmit FIFO Status Register	918_H + n*20_H	U, PV	U, PV	Page 16-338
DIEPDMABx	Device Endpoint-n DMA Buffer Address Register	$91C_H$ + n*20_H	U, PV	U, PV	Page 16-337
Reserved	Reserved	$9E0_H$-AFC_H	nBE	nBE	
DOEPCTL0	Device Control OUT Endpoint 0 Control Register	$B00_H$	U, PV	U, PV	Page 16-311
DOEPCTLx	Device Endpoint-n Control Register	$B00_H$ + n*20_H	U, PV	U, PV	Page 16-314
Reserved	Reserved	$B04_H$ + n*20_H	nBE	nBE	
DOEPINTx	Device Endpoint-n Interrupt Register	$B08_H$ + n*20_H	U, PV	U, PV	Page 16-324

Table 16-19 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
Reserved	Reserved	$B0C_H$ + n*20_H	nBE	nBE	
DOEPTSIZ0	Device Endpoint 0 Transfer Size Register	$B10_H$	U, PV	U, PV	Page 16-330
DOEPTSIZx	Device Endpoint-n Transfer Size Register	$B10_H$ + n*20_H	U, PV	U, PV	Page 16-332
DOEPDMAx	Device Endpoint-n DMA Address Register	$B14_H$ + n*20_H	U, PV	U, PV	Page 16-336
Reserved	Reserved	$B18_H$ + n*20_H	nBE	nBE	
DOEPDMABx	Device Endpoint-n DMA Buffer Address Register	$B1C_H$ + n*20_H	U, PV	U, PV	Page 16-337
Reserved	Reserved	$BE0_H$-DFC_H	nBE	nBE	
USB Power and Gating Register					
PCGCR	Power and Clock Gating Control Register	$E00_H$	U, PV	U, PV	Page 16-338
Reserved	Reserved	$E04_H$-FFC_H	nBE	nBE	

Data FIFO (DFIFO) Access Register Map

These registers, available in both Host and Device modes, are used to read or write the FIFO space for a specific endpoint or a channel, in a given direction. If a host channel is of type IN, the FIFO can only be read on the channel. Similarly, if a host channel is of type OUT, the FIFO can only be written on the channel.

Table 16-20 Data FIFO (DFIFO) Access Register Map

FIFO Access Register Section	Address Range	Access
Device IN Endpoint 0/Host OUT Channel 0: DFIFO Write Access Device OUT Endpoint 0/Host IN Channel 0: DFIFO Read Access	1000_H-$1FFC_H$	WO/RO
Device IN Endpoint 1/Host OUT Channel 1: DFIFO Write Access Device OUT Endpoint 1/Host IN Channel 1: DFIFO Read Access	2000_H-$2FFC_H$	WO/RO
...	...	...

Table 16-20 Data FIFO (DFIFO) Access Register Map (cont'd)

FIFO Access Register Section	Address Range	Access
Device IN Endpoint 6/Host OUT Channel 6: DFIFO Write Access Device OUT Endpoint 6/Host IN Channel 6: DFIFO Read Access	7000_H- $7FFC_H$	WO/RO
Host OUT Channel 7: DFIFO Write Access Host IN Channel 7: DFIFO Read Access	8000_H- $8FFC_H$	WO/RO
...	...	...
Host OUT Channel 13: DFIFO Write Access Host IN Channel 13: DFIFO Read Access	$E000_H$- $EFFC_H$	WO/RO

Access Restriction

Note: The USB registers are accessible only through word accesses. Half-word and byte accesses on USB registers will not generate a bus error. Write to unused address space will not cause an error but be ignored.

16.15.1 Register Description

This section describes Core Global, Device Mode, Host Mode, and Power and Clock Gating CSRs.

Note: Always program Reserved fields with 0s. Treat read values from Reserved fields as unknowns (Xs).

Global Registers

These registers are available in both Host and Device modes, and do not need to be reprogrammed when switching between these modes.

Control and Status Register (GOTGCTL)

The OTG Control and Status register controls the behavior and reflects the status of the OTG function of the core.

GOTGCTL
Control and Status Register **(000_H)** **Reset Value: 0001 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0											OTG Ver	BSesVld	ASesVld	DbncTime	ConIDSts
r											rw	rh	rh	rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				DevHNPEn	HstSetHNPEn	HNPReq	HstNegScs	BvalidOvVal	BvalidOvEn	AvalidOvVal	AvalidOvEn	VbvalidOvVal	VbvalidOvEn	SesReq	SesReqScs
r				rw	rw	rw	rh	rw	rw	rw	rw	rw	rw	rw	rh

Field	Bits	Type	Description
SesReqScs	0	rh	**Session Request Success** The core sets this bit when a session request initiation is successful. 0_B Session request failure 1_B Session request success This bit is used in Device only.
SesReq	1	rw	**Session Request** The application sets this bit to initiate a session request on the USB. The application can clear this bit by writing a 0 when the Host Negotiation Success Status Change bit in the OTG Interrupt register (GOTGINT.HstNegSucStsChng) is set. The core clears this bit when the HstNegSucStsChng bit is cleared. Since the USB 1.1 Full-Speed Serial Transceiver interface is used to initiate the session request, the application must wait until the Vbus discharges to 0.2 V, after the B-Session Valid bit in this register (GOTGCTL.BSesVld) is cleared. 0_B No session request 1_B Session request This bit is used in Device only.

Field	Bits	Type	Description
VbvalidOvEn	2	rw	**VBUS Valid Override Enable** This bit is used to enable/disable the software to override the vbus valid signal using the GOTGCTL.VbvalidOvVal. 0_B Override is disabled and vbus valid signal from the PHY is used internally by the core. 1_B Internally vbus valid received from the PHY is overridden with GOTGCTL.VbvalidOvVal. This bit is used in Host only.
VbvalidOvVal	3	rw	**VBUS Valid Override Value** This bit is used to set the override value for vbus valid signal when GOTGCTL.VbvalidOvEn is set. 0_B vbusvalid value is 0_B when GOTGCTL.VbvalidOvEn = 1 1_B vbusvalid value is 1_B when GOTGCTL.VbvalidOvEn = 1 This bit is used in Host only.
AvalidOvEn	4	rw	**A-Peripheral Session Valid Override Enable** This bit is used to enable/disable the software to override the Avalid signal using the GOTGCTL.AvalidOvVal. 0_B Override is disabled and Avalid signal from the PHY is used internally by the core. 1_B Internally Avalid received from the PHY is overridden with GOTGCTL.AvalidOvVal. This bit is used in Host only.
AvalidOvVal	5	rw	**A-Peripheral Session Valid Override Value** This bit is used to set the override value for Avalid signal when GOTGCTL.AvalidOvEn is set. 0_B Avalid value is 0_B when GOTGCTL.AvalidOvEn = 1 1_B Avalid value is 1_B when GOTGCTL.AvalidOvEn = 1 This bit is used in Host only.
BvalidOvEn	6	rw	**B-Peripheral Session Valid Override Enable** This bit is used to enable/disable the software to override the Bvalid signal using the GOTGCTL.BvalidOvVal. 0_B Override is disabled and Bvalid signal from the PHY is used internally by the core. 1_B Internally Bvalid received from the PHY is overridden with GOTGCTL.BvalidOvVal. This bit is used in Device only.

Field	Bits	Type	Description
BvalidOvVal	7	rw	**B-Peripheral Session Valid Override Value** This bit is used to set the override value for Bvalid signal when GOTGCTL.BvalidOvEn is set. 0_B Bvalid value is 0_B when GOTGCTL.BvalidOvEn = 1 1_B Bvalid value is 1_B when GOTGCTL.BvalidOvEn = 1 This bit is used in Device only.
HstNegScs	8	rh	**Host Negotiation Success** The core sets this bit when host negotiation is successful. The core clears this bit when the HNP Request (HNPReq) bit in this register is set. 0_B Host negotiation failure 1_B Host negotiation success This bit is used in Device only.
HNPReq	9	rw	**HNP Request** The application sets this bit to initiate an HNP request to the connected USB host. The application can clear this bit by writing a 0 when the Host Negotiation Success Status Change bit in the OTG Interrupt register (GOTGINT.HstNegSucStsChng) is set. The core clears this bit when the HstNegSucStsChng bit is cleared. 0_B No HNP request 1_B HNP request This bit is used in Device only.
HstSetHNPEn	10	rw	**Host Set HNP Enable** The application sets this bit when it has successfully enabled HNP (using the SetFeature.SetHNPEnable command) on the connected device. 0_B Host Set HNP is not enabled 1_B Host Set HNP is enabled This bit is used in Host only.
DevHNPEn	11	rw	**Device HNP Enabled** The application sets this bit when it successfully receives a SetFeature.SetHNPEnable command from the connected USB host. 0_B HNP is not enabled in the application 1_B HNP is enabled in the application This bit is used in Device only.

Field	Bits	Type	Description
ConIDSts	16	rh	**Connector ID Status** Indicates the connector ID status on a connect event. 0_B The USB core is in A-Device mode 1_B The USB core is in B-Device mode
DbncTime	17	rh	**Long/Short Debounce Time** Indicates the debounce time of a detected connection. 0_B Long debounce time, used for physical connections (100 ms + 2.5 µs) 1_B Short debounce time, used for soft connections (2.5 µs) This bit is used in Host only.
ASesVld	18	rh	**A-Session Valid** Indicates the Host mode transceiver status. 0_B A-session is not valid 1_B A-session is valid *Note: If the OTG features (such as SRP and HNP) are not enabled, the read reset value will be 1.* This bit is used in Host only.
BSesVld	19	rh	**B-Session Valid** Indicates the Device mode transceiver status. 0_B B-session is not valid. 1_B B-session is valid. In OTG mode, this bit can be used to determine if the device is connected or disconnected. *Note: If the OTG features (such as SRP and HNP) are not enabled, the read reset value will be 1.* This bit is used in Device only.

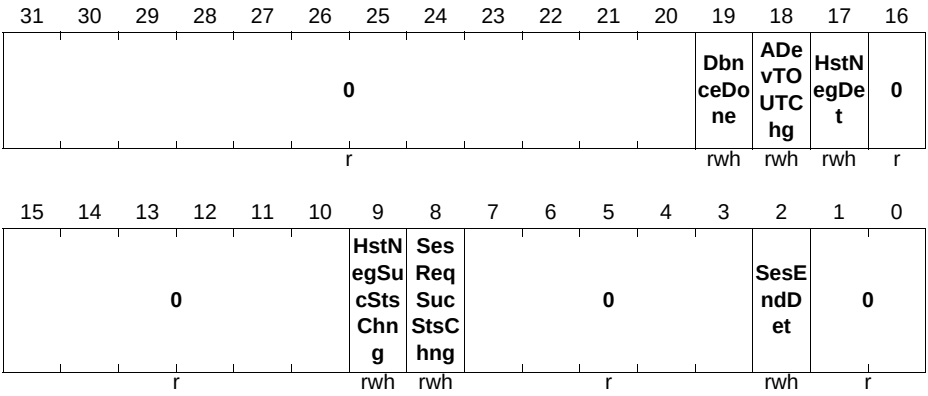
Field	Bits	Type	Description
OTGVer	20	rw	**OTG Version** Indicates the OTG revision. 0_B OTG Version 1.3. In this version the core supports Data line pulsing and VBus pulsing for SRP. 1_B OTG Version 2.0. In this version the core supports only Data line pulsing for SRP. *Note: XMC4500 supports only OTG Version 1.3. Therefore, the OTGVer bit should always be written with 0.*
0	[15:12], [31:21]	r	**Reserved** Read as 0; should be written with 0.

Interrupt Register (GOTGINT)

The application reads this register whenever there is an OTG interrupt and clears the bits in this register to clear the OTG interrupt.

Note: All bits in this register are set only by hardware and cleared only by a software write of 1 to the bit.

GOTGINT
OTG Interrupt Register **(004_H)** **Reset Value: 0000 0000_H**



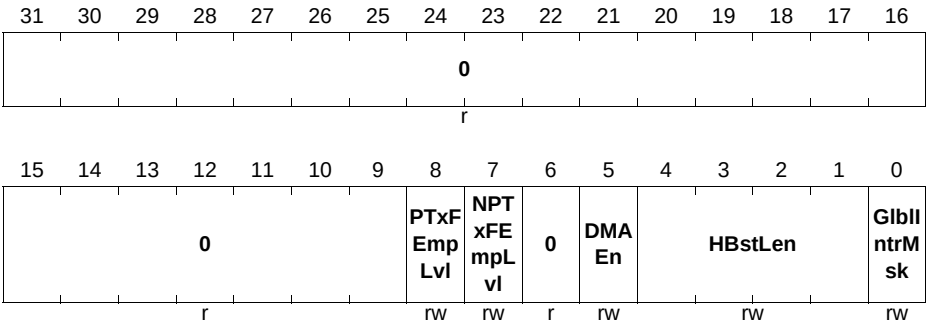
Field	Bits	Type	Description
SesEndDet	2	rwh	**Session End Detected** The core sets this bit when the bvalid signal is deasserted. This bit is used in Device only.
SesReqSucStsChng	8	rwh	**Session Request Success Status Change** The core sets this bit on the success or failure of a session request. The application must read the Session Request Success bit in the OTG Control and Status register (GOTGCTL.SesReqScs) to check for success or failure.
HstNegSucStsChng	9	rwh	**Host Negotiation Success Status Change** The core sets this bit on the success or failure of a USB host negotiation request. The application must read the Host Negotiation Success bit of the OTG Control and Status register (GOTGCTL.HstNegScs) to check for success or failure.

Field	Bits	Type	Description
HstNegDet	17	rwh	**Host Negotiation Detected** The core sets this bit when it detects a host negotiation request on the USB.
ADevTOUTChg	18	rwh	**A-Device Timeout Change** The core sets this bit to indicate that the A-device has timed out while waiting for the B-device to connect.
DbnceDone	19	rwh	**Debounce Done** The core sets this bit when the debounce is completed after the device connect. The application can start driving USB reset after seeing this interrupt. This bit is only valid when the HNP Capable or SRP Capable bit is set in the Core USB Configuration register (GUSBCFG.HNPCap or GUSBCFG.SRPCap, respectively). This bit is used in Host only.
0	[31:20], [16:10], [7:3], [1:0]	r	**Reserved** Read as 0; should be written with 0.

AHB Configuration Register (GAHBCFG)

This register can be used to configure the core after power-on or a change in mode. This register mainly contains AHB system-related configuration parameters. Do not change this register after the initial programming. The application must program this register before starting any transactions on either the AHB or the USB.

GAHBCFG
AHB Configuration Register **(008_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
GlblIntrMsk	0	rw	**Global Interrupt Mask** The application uses this bit to mask or unmask the interrupt line assertion to itself. Irrespective of this bit's setting, the interrupt status registers are updated by the core. 0_B Mask the interrupt assertion to the application. 1_B Unmask the interrupt assertion to the application.
HBstLen	[4:1]	rw	**Burst Length/Type** This field is used in DMA mode to indicate the AHB Master burst type. 0000_B Single 0001_B INCR 0011_B INCR4 0101_B INCR8 0111_B INCR16 Others: Reserved
DMAEn	5	rw	**DMA Enable** 0_B Core operates in Slave mode 1_B Core operates in a DMA mode

Field	Bits	Type	Description
NPTxFEmpLvl	7	rw	**Non-Periodic TxFIFO Empty Level** This bit indicates when IN endpoint Transmit FIFO empty interrupt (DIEPINTx.TxFEmp) is triggered. 0_B DIEPINTx.TxFEmp interrupt indicates that the IN Endpoint TxFIFO is half empty 1_B DIEPINTx.TxFEmp interrupt indicates that the IN Endpoint TxFIFO is completely empty This bit is used only in Device Slave mode.
PTxFEmpLvl	8	rw	**Periodic TxFIFO Empty Level** Indicates when the Periodic TxFIFO Empty Interrupt bit in the Core Interrupt register (GINTSTS.PTxFEmp) is triggered. 0_B GINTSTS.PTxFEmp interrupt indicates that the Periodic TxFIFO is half empty 1_B GINTSTS.PTxFEmp interrupt indicates that the Periodic TxFIFO is completely empty This bit is used only in Host Slave mode.
0	[31:9], 6	r	**Reserved** Read as 0; should be written with 0.

USB Configuration Register (GUSBCFG)

This register can be used to configure the core after power-on or a changing to Host mode or Device mode. It contains USB and USB-PHY related configuration parameters. The application must program this register before starting any transactions on either the AHB or the USB. Do not make changes to this register after the initial programming.

GUSBCFG
USB Configuration Register **($00C_H$)** **Reset Value: $0000\ 1440_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CTP	ForceDevMode	ForceHstMode	TxEndDelay	0											OtgI2CSel
rw	rw	rw	rw	r											rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		USBTrdTim				HNPCap	SRPCap	0	PHYSel	0			TOutCal		
r		rw				rw	rw	r	r	r			rw		

Field	Bits	Type	Description
TOutCal	[2:0]	rw	**FS Timeout Calibration** The number of PHY clocks that the application programs in this field is added to the full-speed interpacket timeout duration in the core to account for any additional delays introduced by the PHY. This can be required, because the delay introduced by the PHY in generating the line state condition can vary from one PHY to another. The USB standard timeout value for full-speed operation is 16 to 18 (inclusive) bit times. The application must program this field based on the speed of enumeration. The number of bit times added per PHY clock is 0.25 bit times
PHYSel	6	r	**USB 1.1 Full-Speed Serial Transceiver Select** This bit is always read as 1 to indicate a full-speed transceiver is selected. 0_B Reserved 1_B USB 1.1 full-speed serial transceiver

Field	Bits	Type	Description
SRPCap	8	rw	**SRP-Capable** The application uses this bit to control the USB core SRP capabilities. If the core operates as a non-SRP-capable B-device, it cannot request the connected A-device (host) to activate VBUS and start a session. 0_B SRP capability is not enabled. 1_B SRP capability is enabled.
HNPCap	9	rw	**HNP-Capable** The application uses this bit to control the USB core's HNP capabilities. 0_B HNP capability is not enabled. 1_B HNP capability is enabled.
USBTrdTim	[13:10]	rw	**USB Turnaround Time** Sets the turnaround time in PHY clocks. Specifies the response time for a MAC request to the Packet FIFO Controller (PFC) to fetch data from the DFIFO (SPRAM). *Note: USB turnaround time is critical for certification where long cables and 5-Hubs are used. See "Choosing the Value of GUSBCFG.USBTrdTim" on page 446.* This bit is used in Device Only.
OtgI2CSel	16	rw	**UTMIFS Interface Select** The application uses this bit to select the USB 1.1 Full-Speed interface. 0_B UTMI USB 1.1 Full-Speed interface for OTG signals 1_B Reserved This bit should always be written with 0.
TxEndDelay	28	rw	**Tx End Delay** Writing a 1 to this bit enables the TxEndDelay timers in the core as per the section 4.1.5 on Opmode of the USB 2.0 Transceiver Macrocell Interface (UTMI) version 1.05. 0_B Normal mode 1_B Introduce Tx end delay timers This bit is used in Device Only.

Field	Bits	Type	Description
ForceHst Mode	29	rw	**Force Host Mode** Writing a 1 to this bit forces the core to host mode irrespective of connected plug. 0_B Normal Mode 1_B Force Host Mode After setting the force bit, the application must wait at least 25 ms before the change to take effect.
ForceDev Mode	30	rw	**Force Device Mode** Writing a 1 to this bit forces the core to device mode irrespective of connected plug. 0_B Normal Mode 1_B Force Device Mode After setting the force bit, the application must wait at least 25 ms before the change to take effect.
CTP	31	rw	**Corrupt Tx packet** This bit is for debug purposes only. Never set this bit to 1.
0	[27:17], [15:14], 7, [5:3]	r	**Reserved** Read as 0; should be written with 0.

Reset Register (GRSTCTL)

The application uses this register to reset various hardware features inside the core.

GRSTCTL
Reset Register **(010_H)** **Reset Value: $1000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AHBIdle	**DMAReq**	**0**													
r	r	r													

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					**TxFNum**					**TxFFlsh**	**RxFFlsh**	**0**	**FrmCntrRst**	**0**	**CSftRst**
r					rw					rwh	rwh	r	rwh	r	rwh

Field	Bits	Type	Description
CSftRst	0	rwh	**Core Soft Reset** Resets the hclk and phy_clock domains as follows: • Clears the interrupts and all the CSR registers except the following register bits: – PCGCCTL.GateHclk – GUSBCFG.PHYSel – HCFG.FSLSPclkSel – DCFG.DevSpd – GGPIO • All module state machines (except the AHB Slave Unit) are reset to the IDLE state, and all the transmit FIFOs and the receive FIFO are flushed. • Any transactions on the AHB Master are terminated as soon as possible, after gracefully completing the last data phase of an AHB transfer. Any transactions on the USB are terminated immediately. The application can write to this bit any time it wants to reset the core. This is a self-clearing bit and the core clears this bit after all the necessary logic is reset in the core, which can take several clocks, depending on the current state of the core. Once this bit is cleared software must wait at least 3 PHY clocks before doing any access to the PHY domain (synchronization delay). Software must also must check that bit 31 of this register is 1 (AHB Master is IDLE) before starting any operation. Typically software reset is used during software development.
FrmCntrRst	2	rwh	**Host Frame Counter Reset** The application writes this bit to reset the frame number counter inside the core. When the frame counter is reset, the subsequent SOF sent out by the core has a frame number of 0. This bit is used in Host only. This bit is set only by software and cleared only by hardware.

Field	Bits	Type	Description
RxFFlsh	4	rwh	**RxFIFO Flush** The application can flush the entire RxFIFO using this bit, but must first ensure that the core is not in the middle of a transaction. The application must only write to this bit after checking that the core is neither reading from the RxFIFO nor writing to the RxFIFO. The application must wait until the bit is cleared before performing any other operations. This bit requires 8 clocks (slowest of PHY or AHB clock) to clear. This bit is set only by software and cleared only by hardware.
TxFFlsh	5	rwh	**TxFIFO Flush** This bit selectively flushes a single or all transmit FIFOs, but cannot do so if the core is in the midst of a transaction. The application must write this bit only after checking that the core is neither writing to the TxFIFO nor reading from the TxFIFO. Verify using these registers: • Read—NAK Effective Interrupt ensures the core is not reading from the FIFO • Write—GRSTCTL.AHBIdle ensures the core is not writing anything to the FIFO. Flushing is normally recommended when FIFOs are re-configured. FIFO flushing is also recommended during device endpoint disable. The application must wait until the core clears this bit before performing any operations. This bit requires 8 clocks (slowest of PHY or AHB clock) to clear. This bit is set only by software and cleared only by hardware.

Field	Bits	Type	Description
TxFNum	[10:6]	rw	**TxFIFO Number** This is the FIFO number that must be flushed using the TxFIFO Flush bit. This field must not be changed until the core clears the TxFIFO Flush bit. 00_H Non-periodic TxFIFO flush in Host mode or Tx FIFO 0 flush in device mode 01_H Periodic TxFIFO flush in Host mode or Tx FIFO 1 flush in device mode 02_H Tx FIFO 2 flush in device mode $...{}_H$ ... $0F_H$ Tx FIFO 15 flush in device mode 10_H Flush all the transmit FIFOs in device or host mode.
DMAReq	30	r	**DMA Request Signal** Indicates that the DMA request is in progress. Used for debug.
AHBIdle	31	r	**AHB Master Idle** Indicates that the AHB Master State Machine is in the IDLE condition.
0	[29:11], 3, 1	r	**Reserved** Read as 0; should be written with 0.

Interrupt Register (GINTSTS)

This register interrupts the application for system-level events in the current mode (Device mode or Host mode). It is shown in **Figure 16-67**.

Some of the bits in this register are valid only in Host mode, while others are valid in Device mode only. This register also indicates the current mode.

Note: In the GINTSTS register, interrupt status bits with access type 'rwh' are set by hardware. To clear these bits, the application must write 1 into these bits.

The FIFO status interrupts are read only; once software reads from or writes to the FIFO while servicing these interrupts, FIFO interrupt conditions are cleared automatically.

The application must clear the GINTSTS register at initialization before unmasking the interrupt bit to avoid any interrupts generated prior to initialization.

GINTSTS
Interrupt Register [HOSTMODE] (014_H) Reset Value: 1400 0020$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WkUpInt	**SessReqInt**	**DisconnInt**	**ConIDStsChng**	**0**	**PTxFEmp**	**HChlInt**	**PrtInt**	**0**		**incomplP**	**0**				
rwh	rwh	rwh	rwh	r	rh	rh	rh	r		rwh	r				

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0										**1**	**RxFLvl**	**Sof**	**OTGInt**	**ModeMis**	**CurMod**
r										r	rh	rwh	rh	rwh	rh

Field	Bits	Type	Description
CurMod	0	rh	**Current Mode of Operation** Indicates the current mode. 0_B Device mode 1_B Host mode
ModeMis	1	rwh	**Mode Mismatch Interrupt** The core sets this bit when the application is trying to access: • A Host mode register, when the core is operating in Device mode • A Device mode register, when the core is operating in Host mode The register access is completed on the AHB with an OKAY response, but is ignored by the core internally and does not affect the operation of the core.
OTGInt	2	rh	**OTG Interrupt** The core sets this bit to indicate an OTG protocol event. The application must read the OTG Interrupt Status (GOTGINT) register to determine the exact event that caused this interrupt. The application must clear the appropriate status bit in the GOTGINT register to clear this bit.
Sof	3	rwh	**Start of Frame** In Host mode, the core sets this bit to indicate that an SOF is transmitted on the USB.
RxFLvl	4	rh	**RxFIFO Non-Empty** Indicates that there is at least one packet pending to be read from the RxFIFO.
incomplP	21	rwh	**Incomplete Periodic Transfer** In **Host mode**, the core sets this interrupt bit when there are incomplete periodic transactions still pending which are scheduled for the current frame.
PrtInt	24	rh	**Host Port Interrupt** The core sets this bit to indicate a change in port status of one of the USB core ports in Host mode. The application must read the Host Port Control and Status (HPRT) register to determine the exact event that caused this interrupt. The application must clear the appropriate status bit in the Host Port Control and Status register to clear this bit.

Field	Bits	Type	Description
HChInt	25	rh	**Host Channels Interrupt** The core sets this bit to indicate that an interrupt is pending on one of the channels of the core (in Host mode). The application must read the Host All Channels Interrupt (HAINT) register to determine the exact number of the channel on which the interrupt occurred, and then read the corresponding Host Channel-n Interrupt (HCINTx) register to determine the exact cause of the interrupt. The application must clear the appropriate status bit in the HCINTx register to clear this bit.
PTxFEmp	26	rh	**Periodic TxFIFO Empty** This interrupt is asserted when the Periodic Transmit FIFO is either half or completely empty and there is space for at least one entry to be written in the Periodic Request Queue. The half or completely empty status is determined by the Periodic TxFIFO Empty Level bit in the Core AHB Configuration register (GAHBCFG.PTxFEmpLvl).
ConIDSts Chng	28	rwh	**Connector ID Status Change** This interrupt is asserted when there is a change in connector ID status.
DisconnInt	29	rwh	**Disconnect Detected Interrupt** This interrupt is asserted when a device disconnect is detected.
SessReqInt	30	rwh	**Session Request/New Session Detected Interrupt** In Host mode, this interrupt is asserted when a session request is detected from the device. In Device mode, this interrupt is asserrted when the Bvalid signal goes high.
WkUpInt	31	rwh	**Resume/Remote Wakeup Detected Interrupt** Wakeup Interrupt during Suspend state. • Device Mode - This interrupt is asserted only when Host Initiated Resume is detected on USB. • Host Mode - This interrupt is asserted only when Device Initiated Remote Wakeup is detected on USB.

Field	Bits	Type	Description
1	5	r	**Reserved** Read as 1; should be written with 1.
0	27, [23:22], [20:6]	r	**Reserved** Read as 0; should be written with 0.

GINTSTS
Interrupt Register [DEVICEMODE] (014_H) **Reset Value: $1400\ 0020_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WkUpInt	**SessReqInt**	**0**	**ConIDStsChng**	**0**	**1**	**0**				**incomplSOOUT**	**incomplSOIN**	**OEPInt**	**IEPInt**	**0**	
rwh	rwh	r	rwh	r	r	r				rwh	rwh	r	r	r	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EOPF	**ISOOutDrop**	**EnumDone**	**USBRst**	**USBSusp**	**ErlySusp**	**0**		**GOUTNakEff**	**GINNakEff**	**1**	**RxFLvl**	**Sof**	**OTGInt**	**ModeMis**	**CurMod**
rwh	rwh	rwh	rwh	rwh	rwh	r		rh	rh	r	rh	rwh	rh	rwh	rh

Field	Bits	Type	Description
CurMod	0	rh	**Current Mode of Operation** Indicates the current mode. 0_B Device mode 1_B Host mode
ModeMis	1	rwh	**Mode Mismatch Interrupt** The core sets this bit when the application is trying to access: • A Host mode register, when the core is operating in Device mode • A Device mode register, when the core is operating in Host mode The register access is completed on the AHB with an OKAY response, but is ignored by the core internally and does not affect the operation of the core.

Field	Bits	Type	Description
OTGInt	2	rh	**OTG Interrupt** The core sets this bit to indicate an OTG protocol event. The application must read the OTG Interrupt Status (GOTGINT) register to determine the exact event that caused this interrupt. The application must clear the appropriate status bit in the GOTGINT register to clear this bit.
Sof	3	rwh	**Start of Frame** In Device mode, the core sets this bit to indicate that an SOF token has been received on the USB. The application can read the Device Status register to get the current frame number.
RxFLvl	4	rh	**RxFIFO Non-Empty** Indicates that there is at least one packet pending to be read from the RxFIFO.
GINNakEff	6	rh	**Global IN Non-Periodic NAK Effective** Indicates that the Set Global Non-periodic IN NAK bit in the Device Control register (DCTL.SGNPInNak), set by the application, has taken effect in the core. That is, the core has sampled the Global IN NAK bit set by the application. This bit can be cleared by clearing the Clear Global Non-periodic IN NAK bit in the Device Control register (DCTL.CGNPInNak). This interrupt does not necessarily mean that a NAK handshake is sent out on the USB. The STALL bit takes precedence over the NAK bit.
GOUTNakEff	7	rh	**Global OUT NAK Effective** Indicates that the Set Global OUT NAK bit in the Device Control register (DCTL.SGOUTNak), set by the application, has taken effect in the core. This bit can be cleared by writing the Clear Global OUT NAK bit in the Device Control register (DCTL.CGOUTNak).
ErlySusp	10	rwh	**Early Suspend** The core sets this bit to indicate that an Idle state has been detected on the USB for 3 ms.

Field	Bits	Type	Description
USBSusp	11	rwh	**USB Suspend** The core sets this bit to indicate that a suspend was detected on the USB. The core enters the Suspended state when there is no activity on the two USB data signals for an extended period of time.
USBRst	12	rwh	**USB Reset** The core sets this bit to indicate that a reset is detected on the USB.
EnumDone	13	rwh	**Enumeration Done** The core sets this bit to indicate that speed enumeration is complete. The application must read the Device Status (DSTS) register to obtain the enumerated speed.
ISOOutDrop	14	rwh	**Isochronous OUT Packet Dropped Interrupt** The core sets this bit when it fails to write an isochronous OUT packet into the RxFIFO because the RxFIFO does not have enough space to accommodate a maximum packet size packet for the isochronous OUT endpoint.
EOPF	15	rwh	**End of Periodic Frame Interrupt** Indicates that the period specified in the Periodic Frame Interval field of the Device Configuration register (DCFG.PerFrInt) has been reached in the current frame.
IEPInt	18	r	**IN Endpoints Interrupt** The core sets this bit to indicate that an interrupt is pending on one of the IN endpoints of the core (in Device mode). The application must read the Device All Endpoints Interrupt (DAINT) register to determine the exact number of the IN endpoint on which the interrupt occurred, and then read the corresponding Device IN Endpoint-n Interrupt (DIEPINTx) register to determine the exact cause of the interrupt. The application must clear the appropriate status bit in the corresponding DIEPINTx register to clear this bit.

Field	Bits	Type	Description
OEPInt	19	r	**OUT Endpoints Interrupt** The core sets this bit to indicate that an interrupt is pending on one of the OUT endpoints of the core (in Device mode). The application must read the Device All Endpoints Interrupt (DAINT) register to determine the exact number of the OUT endpoint on which the interrupt occurred, and then read the corresponding Device OUT Endpoint-n Interrupt (DOEPINTx) register to determine the exact cause of the interrupt. The application must clear the appropriate status bit in the corresponding DOEPINTx register to clear this bit.
incomplSOIN	20	rwh	**Incomplete Isochronous IN Transfer** The core sets this interrupt to indicate that there is at least one isochronous IN endpoint on which the transfer is not completed in the current frame. This interrupt is asserted along with the End of Periodic Frame Interrupt (EOPF) bit in this register. *Note: This interrupt is not asserted in Scatter/Gather DMA mode.*
incomplSOOUT	21	rwh	**Incomplete Isochronous OUT Transfer** In the Device mode, the core sets this interrupt to indicate that there is at least one isochronous OUT endpoint on which the transfer is not completed in the current frame. This interrupt is asserted along with the End of Periodic Frame Interrupt (EOPF) bit in this register.
ConIDStsChng	28	rwh	**Connector ID Status Change** This interrupt is asserted when there is a change in connector ID status.
SessReqInt	30	rwh	**Session Request/New Session Detected Interrupt** In Host mode, this interrupt is asserted when a session request is detected from the device. In Device mode, this interrupt is asserrted when the Bvalid signal goes high.
WkUpInt	31	rwh	**Resume/Remote Wakeup Detected Interrupt** Wakeup Interrupt during Suspend state. • Device Mode - This interrupt is asserted only when Host Initiated Resume is detected on USB. • Host Mode - This interrupt is asserted only when Device Initiated Remote Wakeup is detected on USB.

Field	Bits	Type	Description
1	26, 5	r	**Reserved** Read as 1; should be written with 1.
0	29, 27, [25:22], [17:16], [9:8]	r	**Reserved** Read as 0; should be written with 0.

Interrupt Mask Register (GINTMSK)

This register works with the Interrupt Register (**"Interrupt Register (GINTSTS)" on Page 16-235**) to interrupt the application. When an interrupt bit is masked, the interrupt associated with that bit is not generated. However, the GINTSTS register bit corresponding to that interrupt is still set.

- Mask interrupt: 0_B
- Unmask interrupt: 1_B

GINTMSK
Interrupt Mask Register [HOSTMODE](018_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WkUpIntMsk	SessReqIntMsk	DisconnIntMsk	ConIDStsChngMsk	0	PTxFEmpMsk	HChIntMsk	PrtIntMsk	0		incomplPMsk	0				
rw	rw	rw	rw	r	rw	rw	rw	r		rw	r				

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0											RxFLvlMsk	SofMsk	OTGIntMsk	ModeMisMsk	0
r											rw	rw	rw	rw	r

Field	Bits	Type	Description
ModeMisMsk	1	rw	**Mode Mismatch Interrupt Mask**
OTGIntMsk	2	rw	**OTG Interrupt Mask**
SofMsk	3	rw	**Start of Frame Mask**
RxFLvlMsk	4	rw	**Receive FIFO Non-Empty Mask**
incomplPMsk	21	rw	**Incomplete Periodic Transfer Mask**
PrtIntMsk	24	rw	**Host Port Interrupt Mask**
HChIntMsk	25	rw	**Host Channels Interrupt Mask**
PTxFEmpMsk	26	rw	**Periodic TxFIFO Empty Mask**
ConIDStsChngMsk	28	rw	**Connector ID Status Change Mask**

Field	Bits	Type	Description
DisconnIntMsk	29	rw	**Disconnect Detected Interrupt Mask**
SessReqIntMsk	30	rw	**Session Request/New Session Detected Interrupt Mask**
WkUpIntMsk	31	rw	**Resume/Remote Wakeup Detected Interrupt Mask**
0	27, [23:22], [20:5], 0	r	**Reserved** Read as 0; should be written with 0.

GINTMSK
Interrupt Mask Register [DEVICEMODE] (018_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WkUpIntMsk	**SessReqIntMsk**	**DisconnIntMsk**	**ConIDStsChngMsk**	**0**						**incomplSOOUTMsk**	**incomplSOINMsk**	**OEPIntMsk**	**IEPIntMsk**	**0**	
rw	rw	rw	rw	r						rw	rw	rw	rw	r	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EOPFMsk	**ISOOutDropMsk**	**EnumDoneMsk**	**USBRstMsk**	**USBSuspMsk**	**ErlySuspMsk**	**0**		**GOUTNakEffMsk**	**GINNakEffMsk**	**0**	**RxFLvlMsk**	**SofMsk**	**OTGIntMsk**	**ModeMisMsk**	**0**
rw	rw	rw	rw	rw	rw	r		rw	rw	r	rw	rw	rw	rw	r

Field	Bits	Type	Description
ModeMisMsk	1	rw	**Mode Mismatch Interrupt Mask**
OTGIntMsk	2	rw	**OTG Interrupt Mask**
SofMsk	3	rw	**Start of Frame Mask**
RxFLvlMsk	4	rw	**Receive FIFO Non-Empty Mask**
GINNakEffMsk	6	rw	**Global Non-periodic IN NAK Effective Mask**

Field	Bits	Type	Description
GOUTNakEffMsk	7	rw	**Global OUT NAK Effective Mask**
ErlySuspMsk	10	rw	**Early Suspend Mask**
USBSuspMsk	11	rw	**USB Suspend Mask**
USBRstMsk	12	rw	**USB Reset Mask**
EnumDoneMsk	13	rw	**Enumeration Done Mask**
ISOOutDropMsk	14	rw	**Isochronous OUT Packet Dropped Interrupt Mask**
EOPFMsk	15	rw	**End of Periodic Frame Interrupt Mask** Mode: Device only Reset: 0_B
IEPIntMsk	18	rw	**IN Endpoints Interrupt Mask**
OEPIntMsk	19	rw	**OUT Endpoints Interrupt Mask**
incompISOINMsk	20	rw	**Incomplete Isochronous IN Transfer Mask**
incompISOOUTMsk	21	rw	**Incomplete Isochronous OUT Transfer Mask**
ConIDStsChngMsk	28	rw	**Connector ID Status Change Mask**
DisconnIntMsk	29	rw	**Disconnect Detected Interrupt Mask**
SessReqIntMsk	30	rw	**Session Request/New Session Detected Interrupt Mask**
WkUpIntMsk	31	rw	**Resume/Remote Wakeup Detected Interrupt Mask**
0	[27:22], [17:16], [9:8], 5, 0	r	**Reserved** Read as 0; should be written with 0.

Receive Status Debug Read/Status Read and Pop Registers (GRXSTSR/GRXSTSP)

A read to the Receive Status Debug Read register returns the contents of the top of the Receive FIFO. A read to the Receive Status Read and Pop register additionally pops the top data entry out of the RxFIFO.

The receive status contents must be interpreted differently in Host and Device modes. The core ignores the receive status pop/read when the receive FIFO is empty and returns a value of $0000'0000_H$. The application must only pop the Receive Status FIFO when the Receive FIFO Non-Empty bit of the Core Interrupt register (GINTSTS.RxFLvl) is asserted.

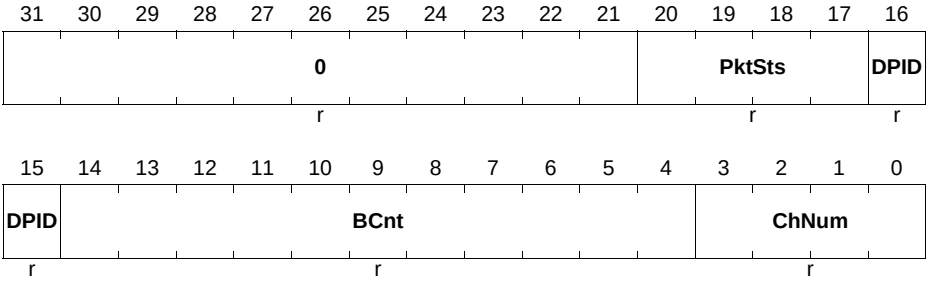
Notes

1. *Use of these fields vary based on whether the OTG core is functioning as a host or a device.*
2. *Do not read this register's reset value before configuring the core because the read value is "X".*

Receive Status Debug Read/Status Read and Pop Registers in Host Mode

GRXSTSR
Receive Status Debug Read Register [HOSTMODE]
($01C_H$) **Reset Value: $0000\ 0000_H$**

GRXSTSP
Receive Status Read and Pop Register [HOSTMODE]
(020_H) **Reset Value: $0000\ 0000_H$**

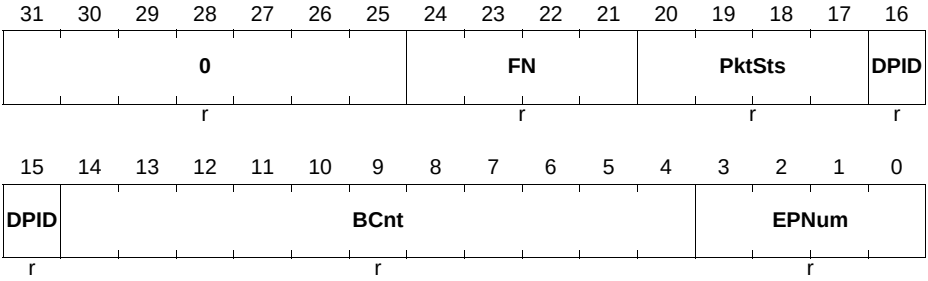


Field	Bits	Type	Description
ChNum	[3:0]	r	**Channel Number** Indicates the channel number to which the current received packet belongs.
BCnt	[14:4]	r	**Byte Count** Indicates the byte count of the received IN data packet.
DPID	[16:15]	r	**Data PID** Indicates the Data PID of the received packet 00_B DATA0 10_B DATA1 01_B DATA2 11_B MDATA
PktSts	[20:17]	r	**Packet Status** Indicates the status of the received packet 0010_B IN data packet received 0011_B IN transfer completed (triggers an interrupt) 0101_B Data toggle error (triggers an interrupt) 0111_B Channel halted (triggers an interrupt) Others: Reserved
0	[31:21]	r	**Reserved** Read as 0; should be written with 0.

Receive Status Debug Read/Status Read and Pop Registers in Device Mode (GRXSTSR/GRXSTSP)

GRXSTSR
Receive Status Debug Read Register [DEVICEMODE]
($01C_H$) **Reset Value: $0000\ 0000_H$**

GRXSTSP
Receive Status Read and Pop Register [DEVICEMODE]
(020_H) **Reset Value: $0000\ 0000_H$**



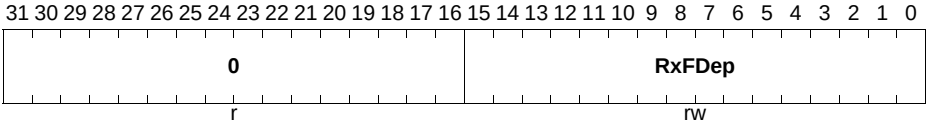
Field	Bits	Type	Description
EPNum	[3:0]	r	**Endpoint Number** Indicates the endpoint number to which the current received packet belongs.
BCnt	[14:4]	r	**Byte Count** Indicates the byte count of the received data packet.
DPID	[16:15]	r	**Data PID** Indicates the Data PID of the received OUT data packet 00_B DATA0 10_B DATA1 01_B DATA2 11_B MDATA
PktSts	[20:17]	r	**Packet Status** Indicates the status of the received packet 0001_B Global OUT NAK (triggers an interrupt) 0010_B OUT data packet received 0011_B OUT transfer completed (triggers an interrupt) 0100_B SETUP transaction completed (triggers an interrupt) 0110_B SETUP data packet received Others: Reserved

Field	Bits	Type	Description
FN	[24:21]	r	**Frame Number** This is the least significant 4 bits of the frame number in which the packet is received on the USB. This field is supported only when isochronous OUT endpoints are supported.
0	[31:25]	r	**Reserved** Read as 0; should be written with 0.

Receive FIFO Size Register (GRXFSIZ)

The application can program the RAM size that must be allocated to the RxFIFO.

GRXFSIZ
Receive FIFO Size Register **(024_H)** **Reset Value: 0000 011A$_H$**



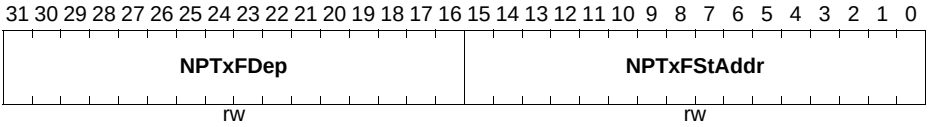
Field	Bits	Type	Description
RxFDep	[15:0]	rw	**RxFIFO Depth** This value is in terms of 32-bit words. • Minimum value is 16 • Maximum value is 282 Programmed values must not exceed the maximum value.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

Non-Periodic Transmit FIFO Size Register (GNPTXFSIZ)

The application can program the RAM size and the memory start address for the Non-periodic TxFIFO.

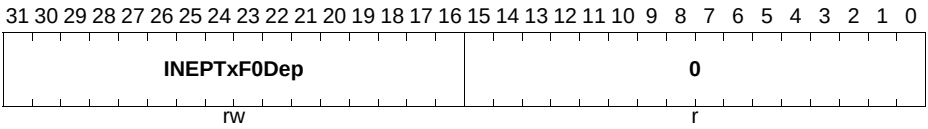
Note: The fields of this register change, depending on host or device mode.

GNPTXFSIZ
Non-Periodic Transmit FIFO Size Register [HOSTMODE]
(028_H) **Reset Value: 0010 011A$_H$**



Field	Bits	Type	Description
NPTxFStAddr	[15:0]	rw	**Non-periodic Transmit RAM Start Address** This field contains the memory start address for Non-periodic Transmit FIFO RAM. Programmed values must not exceed the power-on value.
NPTxFDep	[31:16]	rw	**Non-periodic TxFIFO Depth** This value is in terms of 32-bit words. • Minimum value is 16 • Maximum value is 16 Programmed values must not exceed the maximum value.

GNPTXFSIZ
Non-Periodic Transmit FIFO Size Register [DEVICEMODE]
(028_H) **Reset Value: 0010 0000$_H$**



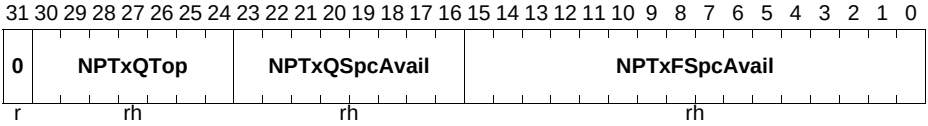
Field	Bits	Type	Description
INEPTxF0 Dep	[31:16]	rw	**IN Endpoint TxFIFO 0 Depth** This value is in terms of 32-bit words. • Minimum value is 16 • Maximum value is 16 Programmed values must not exceed the maximum value.
0	[15:0]	r	**Reserved** Read as 0; should be written with 0.

Non-Periodic Transmit FIFO/Queue Status Register (GNPTXSTS)

This register is valid only in Host.

This read-only register contains the free space information for the Non-periodic TxFIFO and the Non- periodic Transmit Request Queue.

GNPTXSTS
Non-Periodic Transmit FIFO/Queue Status Register($02C_H$) **Reset Value: $0008\ 0010_H$**



Field	Bits	Type	Description
NPTxFSpcAvail	[15:0]	rh	**Non-periodic TxFIFO Space Avail** Indicates the amount of free space available in the Non-periodic TxFIFO. Values are in terms of 32-bit words. 0_H Non-periodic TxFIFO is full 1_H 1 word available 2_H 2 words available Others: Up to n words can be selected (0 < n < 16); selections greater than n are reserved
NPTxQSpcAvail	[23:16]	rh	**Non-periodic Transmit Request Queue Space Available** Indicates the amount of free space available in the Non-periodic Transmit RequestQueue. This queue holds both IN and OUT requests in Host mode. 0_H Non-periodic Transmit Request Queue is full 1_H 1 location available 2_H 2 locations available Others: Up to n locations can be selected (0 < n < 8); selections greater than n are reserved

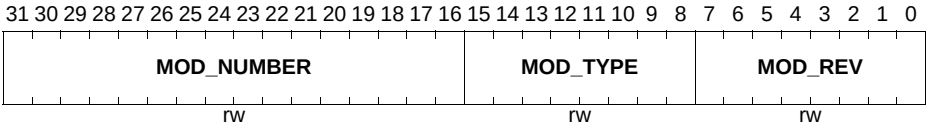
Field	Bits	Type	Description
NPTxQTop	[30:24]	rh	**Top of the Non-periodic Transmit Request Queue** Entry in the Non-periodic Tx Request Queue that is currently being processed by the MAC. • Bits [30:27]: Channel/endpoint number • Bits [26:25]: 00_B IN/OUT token 01_B Zero-length transmit packet (device IN/host OUT) 10_B Reserved 11_B Channel halt command • Bit [24]: Terminate (last entry for selected channel/endpoint)
0	31	r	**Reserved** Read as 0; should be written with 0.

USB Module Identification Register (GUID)

This register contains the USB module version and revision and should not be overwritten by application.

GUID

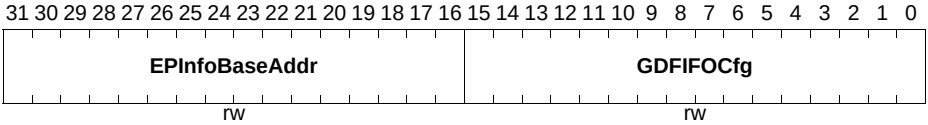
USB Module Identification Register (03C$_H$) **Reset Value: 00AE C0XX$_H$**



Field	Bits	Type	Description
MOD_REV	[7:0]	rw	**Module Revision** Indicates the revision number of the implementation. This information depends on the design step.
MOD_TYPE	[15:8]	rw	**Module Type** This internal marker is fixed to C0$_H$.
MOD_NUMBER	[31:16]	rw	**Module Number** Indicates the module identification number.

Global DFIFO Software Config Register (GDFIFOCFG)

GDFIFOCFG
Global DFIFO Software Config Register(05C$_H$) **Reset Value: 027A 02B2$_H$**

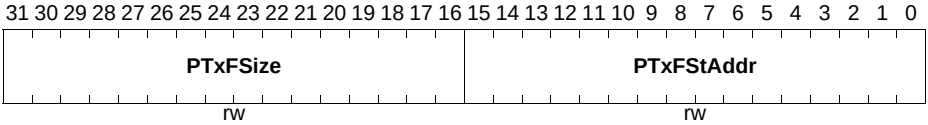


Field	Bits	Type	Description
GDFIFOCfg	[15:0]	rw	**GDFIFOCfg** This field is for dynamic programming of the DFIFO Size. This value takes effect only when the application programs a non zero value to this register. The value programmed must conform to the guidelines described in **“Data FIFO RAM Allocation” on Page 16-201**. The USB core does not have any corrective logic if the FIFO sizes are programmed incorrectly.
EPInfoBaseAddr	[31:16]	rw	**EPInfoBaseAddr** This field provides the start address of the RAM space allocated to store register information in DMA mode. See **“Data FIFO RAM Allocation” on Page 16-201**.

Host Periodic Transmit FIFO Size Register (HPTXFSIZ)

This register holds the size and the memory start address of the Periodic TxFIFO.

HPTXFSIZ
Host Periodic Transmit FIFO Size Register(100_H) **Reset Value: $0100\ 012A_H$**

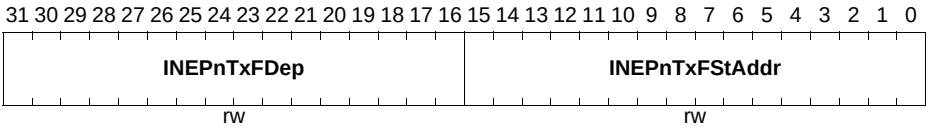


Field	Bits	Type	Description
PTxFStAddr	[15:0]	rw	**Host Periodic TxFIFO Start Address** The power-on reset value of this register is the sum of the Largest Rx Data FIFO Depth and Largest Non-periodic Tx Data FIFO Depth. Programmed values must not exceed the power-on value .
PTxFSize	[31:16]	rw	**Host Periodic TxFIFO Depth** This value is in terms of 32-bit words. Minimum value is 16 Maximum value is 256 Programmed values must not exceed the maximum value.

Device IN Endpoint Transmit FIFO Size Register (DIEPTXFn)

This register holds the size and memory start address of IN endpoint TxFIFOs implemented in Device mode. Each FIFO holds the data for one IN endpoint. This register is repeated for instantiated IN endpoint FIFOs 1 to 6. For IN endpoint FIFO 0 use GNPTXFSIZ register for programming the size and memory start address.

DIEPTXFx (x=1-6)
Device IN Endpoint Transmit FIFO Size Register(100_H + x*04_H) **Reset Value: 0100 011A_H**



Field	Bits	Type	Description
INEPnTxFStAddr	[15:0]	rw	**IN Endpoint FIFOn Transmit RAM Start Address** This field contains the memory start address for IN endpoint Transmit FIFOn (1 < n ≤ 6). Programmed values must not exceed the reset value.
INEPnTxFDep	[31:16]	rw	**IN Endpoint TxFIFO Depth** This value is in terms of 32-bit words. • Minimum value is 16 • Maximum value is 256 Programmed values must not exceed the maximum value.

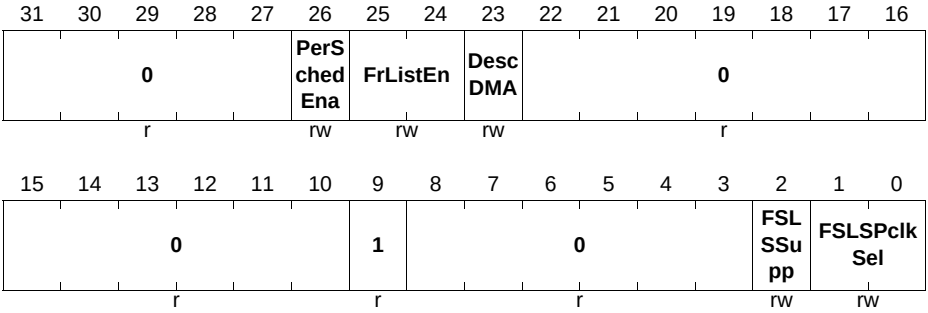
Host Mode Registers

These registers affect the operation of the core in the Host mode. Host mode registers must not be accessed in Device mode, as the results are undefined. Host Mode registers can be categorized as follows:

Host Configuration Register (HCFG)

This register configures the core after power-on. Do not make changes to this register after initializing the host.

HCFG
Host Configuration Register **(400_H)** **Reset Value: 0000 0200$_H$**



Field	Bits	Type	Description
FSLSPclkSel	[1:0]	rw	**FS PHY Clock Select** 01_B PHY clock is running at 48 MHz Others: Reserved
FSLSSupp	2	rw	**FS-Only Support** The application uses this bit to control the core's enumeration speed. Using this bit, the application can make the core enumerate as a FS host, even if the connected device supports HS traffic. Do not make changes to this field after initial programming. 0_B FS-only, connected device can supports also only FS. 1_B FS-only, even if the connected device can support HS

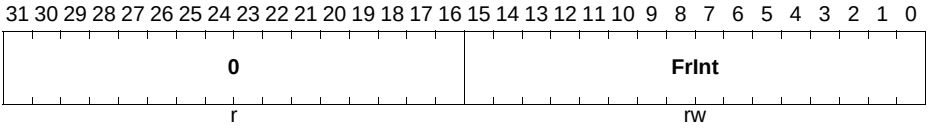
Field	Bits	Type	Description
DescDMA	23	rw	**Enable Scatter/gather DMA in Host mode** The application can set this bit during initialization to enable the Scatter/Gather DMA operation. *Note: This bit must be modified only once after a reset. The following combinations are available for programming:* • GAHBCFG.DMAEn=0, HCFG.DescDMA=0 => Slave mode • GAHBCFG.DMAEn=0, HCFG.DescDMA=1 => Invalid • GAHBCFG.DMAEn=1, HCFG.DescDMA=0 => Buffered DMA mode • GAHBCFG.DMAEn=1, HCFG.DescDMA=1 => Scatter/Gather DMA mode In non Scatter/Gather DMA mode, this bit is reserved.
FrListEn	[25:24]	rw	**Frame List Entries** This field is valid only in Scatter/Gather DMA mode. The value in the register specifies the number of entries in the Frame list. 00_B 8 Entries 01_B 16 Entries 10_B 32 Entries 11_B 64 Entries In modes other than Scatter/Gather DMA mode, these bits are reserved.
PerSchedEna	26	rw	**Enable Periodic Scheduling** Applicable in Scatter/Gather DMA mode only. Enables periodic scheduling within the core. Initially, the bit is reset. The core will not process any periodic channels. As soon as this bit is set, the core will get ready to start scheduling periodic channels and sets HCFG.PerSchedStat. The setting of HCFG.PerSchedStat indicates the core has enabled periodic scheduling. Once HCFG.PerSchedEna is set, the application is not supposed to again reset the bit unless HCFG.PerSchedStat is set. As soon as this bit is reset, the core will get ready to stop scheduling periodic channels and resets HCFG.PerSchedStat. In non Scatter/Gather DMA mode, this bit is reserved.

Field	Bits	Type	Description
1	9	r	**Reserved** Read as 1; should be written with 1.
0	[31:27], [22:10], [8:3]	r	**Reserved** Read as 0; should be written with 0.

Host Frame Interval Register (HFIR)

This register stores the frame interval information for the current speed to which the USB core has enumerated.

HFIR
Host Frame Interval Register **(404_H)** **Reset Value: 0000 EA60$_H$**

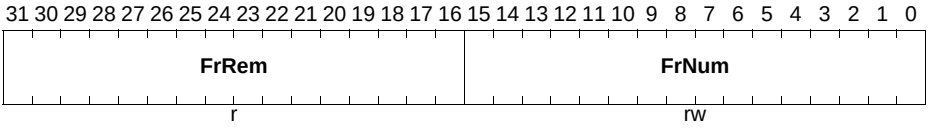


Field	Bits	Type	Description
FrInt	[15:0]	rw	**Frame Interval** The value that the application programs to this field specifies the interval between two consecutive SOFs. This field contains the number of PHY clocks that constitute the required frame interval. The default value set in this field for a FS operation when the PHY clock frequency is 60 MHz. The application can write a value to this register only after the Port Enable bit of the Host Port Control and Status register (HPRT.PrtEnaPort) has been set. If no value is programmed, the core calculates the value based on the PHY clock specified in the FS PHY Clock Select field of the Host Configuration register (HCFG.FSLSPclkSel). Do not change the value of this field after the initial configuration. • 1 ms * (PHY clock frequency for FS)
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

Host Frame Number/Frame Time Remaining Register (HFNUM)

This register indicates the current frame number. It also indicates the time remaining (in terms of the number of PHY clocks) in the current frame.

HFNUM
Host Frame Number/Frame Time Remaining Register **(408_H)** **Reset Value: 0000 3FFF$_H$**

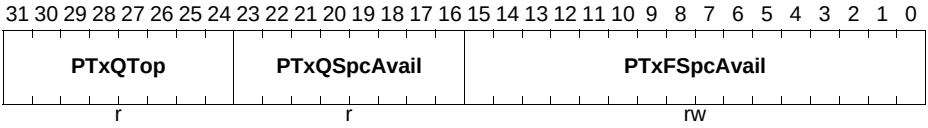


Field	Bits	Type	Description
FrNum	[15:0]	rw	**Frame Number** This field increments when a new SOF is transmitted on the USB, and is reset to 0_H when it reaches $3FFF_H$.
FrRem	[31:16]	r	**Frame Time Remaining** Indicates the amount of time remaining in the current frame, in terms of PHY clocks. This field decrements on each PHY clock. When it reaches zero, this field is reloaded with the value in the Frame Interval register and a new SOF is transmitted on the USB.

Host Periodic Transmit FIFO/Queue Status Register (HPTXSTS)

This read-only register contains the free space information for the Periodic TxFIFO and the Periodic Transmit Request Queue.

HPTXSTS
Host Periodic Transmit FIFO/
Queue Status Register **(410_H)** **Reset Value: $0008\ 0100_H$**



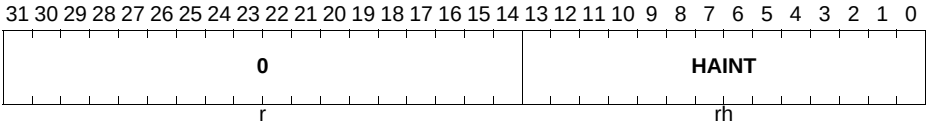
Field	Bits	Type	Description
PTxFSpc Avail	[15:0]	rw	**Periodic Transmit Data FIFO Space Available** Indicates the number of free locations available to be written to in the Periodic TxFIFO. Values are in terms of 32-bit words 0_B Periodic TxFIFO is full 1_H 1 word available 2_H 2 words available Others: Up to n words can be selected (0 < n < 256); selections greater than n are reserved

Field	Bits	Type	Description
PTxQSpcAvail	[23:16]	r	**Periodic Transmit Request Queue Space Available** Indicates the number of free locations available to be written in the Periodic Transmit Request Queue. This queue holds both IN and OUT requests. 0_H Periodic Transmit Request Queue is full 1_H 1 location available 2_H 2 locations available Others: Up to n locations can be selected (0 < n < 8); selections greater than n are reserved
PTxQTop	[31:24]	r	**Top of the Periodic Transmit Request Queue** This indicates the entry in the Periodic Tx Request Queue that is currently being processes by the MAC. This register is used for debugging. Bit [31]: Odd/Even frame 0_B send in even frame 1_B send in odd frame Bits [30:27]: Channel/endpoint number Bits [26:25]: Type 00_B IN/OUT 01_B Zero-length packet 10_B Reserved 11_B Disable channel command Bit [24]: Terminate (last entry for the selected channel/endpoint)

Host All Channels Interrupt Register (HAINT)

When a significant event occurs on a channel, the Host All Channels Interrupt register interrupts the application using the Host Channels Interrupt bit of the Core Interrupt register (GINTSTS.HChInt). This is shown in **Figure 16-67 “Interrupt Hierarchy” on Page 16-209**. There is one interrupt bit per channel, up to a maximum of 14 bits. Bits in this register are set and cleared when the application sets and clears bits in the corresponding Host Channel- n Interrupt register.

HAINT
Host All Channels Interrupt Register (414_H) **Reset Value: $0000\ 0000_H$**



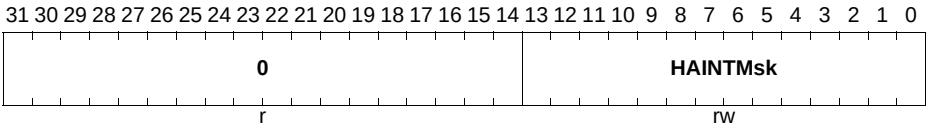
Field	Bits	Type	Description
HAINT	[13:0]	rh	**Channel Interrupts** One bit per channel: Bit 0 for Channel 0, bit 13 for Channel 13
0	[31:14]	r	**Reserved** Read as 0; should be written with 0.

Host All Channels Interrupt Mask Register (HAINTMSK)

The Host All Channel Interrupt Mask register works with the Host All Channel Interrupt register to interrupt the application when an event occurs on a channel. There is one interrupt mask bit per channel, up to a maximum of 14 bits.

- Mask interrupt: 0_B
- Unmask interrupt: 1_B

HAINTMSK
Host All Channels Interrupt Mask Register(418_H) **Reset Value: $0000\ 0000_H$**

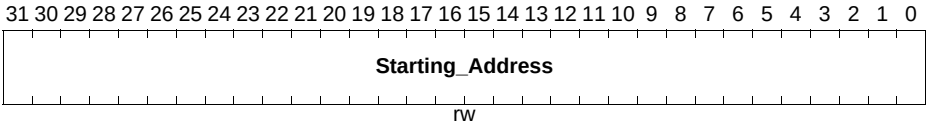


Field	Bits	Type	Description
HAINTMsk	[13:0]	rw	**Channel Interrupt Mask** One bit per channel: Bit 0 for channel 0, bit 13 for channel 13
0	[31:14]	r	**Reserved** Read as 0; should be written with 0.

Host Frame List Base Address Register (HFLBADDR)

This register holds the starting address of the Frame list information. It is present only in case of Scatter/Gather DMA and used only for Isochronous and Interrupt Channels. The register is implemented in RAM.

HFLBADDR
Host Frame List Base Address Register($41C_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
Starting_ Address	[31:0]	rw	**Starting Address** The starting address of the Frame list.

Host Port Control and Status Register (HPRT)

This register is available only in Host mode. Currently, the OTG Host supports only one port.

A single register holds USB port-related information such as USB reset, enable, suspend, resume, connect status, and test mode for each port. It is shown in **Figure 16-67 "Interrupt Hierarchy" on Page 16-209**. The bits PrtOvrCurrChng, PrtEnChng and PrtConnDet in this register can trigger an interrupt to the application through the Host Port Interrupt bit of the Core Interrupt register (GINTSTS.PrtInt). On a Port Interrupt, the application must read this register and clear the bit that caused the interrupt. For these bits, the application must write a 1 to the bit to clear the interrupt.

HPRT
Host Port Control and Status Register(440_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0													PrtSpd		0
r													rh		r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			PrtPwr	PrtLnSts		0	PrtRst	PrtSusp	PrtRes	PrtOvrCurrChng	PrtOvrCurrAct	PrtEnChng	PrtEna	PrtConnDet	PrtConnSts
r			rwh	rh		r	rw	rwh	rwh	rwh	r	rwh	rwh	rwh	rh

Field	Bits	Type	Description
PrtConnSts	0	rh	**Port Connect Status** 0_B No device is attached to the port. 1_B A device is attached to the port.
PrtConnDet	1	rwh	**Port Connect Detected** The core sets this bit when a device connection is detected to trigger an interrupt to the application using the Host Port Interrupt bit of the Core Interrupt register (GINTSTS.PrtInt). This bit is set only by hardware and cleared only by a software write of 1 to the bit.

Field	Bits	Type	Description
PrtEna	2	rwh	**Port Enable** A port is enabled only by the core after a reset sequence, and is disabled by an overcurrent condition, a disconnect condition, or by the application clearing this bit. The application cannot set this bit by a register write. It can only clear it to disable the port. This bit does not trigger any interrupt to the application. 0_B Port disabled 1_B Port enabled
PrtEnChng	3	rwh	**Port Enable/Disable Change** The core sets this bit when the status of the Port Enable bit [2] of this register changes. This bit is set only by hardware and cleared only by a software write of 1 to the bit.
PrtOvrCurrAct	4	r	**Port Overcurrent Active** Indicates the overcurrent condition of the port. 0_B No overcurrent condition 1_B Overcurrent condition
PrtOvrCurrChng	5	rwh	**Port Overcurrent Change** The core sets this bit when the status of the Port Overcurrent Active bit (bit 4) in this register changes. This bit is set only by hardware and cleared only by a software write of 1 to the bit.
PrtRes	6	rwh	**Port Resume** The application sets this bit to drive resume signaling on the port. The core continues to drive the resume signal until the application clears this bit. If the core detects a USB remote wakeup sequence, as indicated by the Port Resume/Remote Wakeup Detected Interrupt bit of the Core Interrupt register (GINTSTS.WkUpInt), the core starts driving resume signaling without application intervention and clears this bit when it detects a disconnect condition. The read value of this bit indicates whether the core is currently driving resume signaling. 0_B No resume driven 1_B Resume driven This bit can be set and cleared by both hardware and software.

Field	Bits	Type	Description
PrtSusp	7	rwh	**Port Suspend** The application sets this bit to put this port in Suspend mode. The core only stops sending SOFs when this is set. To stop the PHY clock, the application must set the Port Clock Stop bit, which asserts the suspend input pin of the PHY. The read value of this bit reflects the current suspend status of the port. This bit is cleared by the core after a remote wakeup signal is detected or the application sets the Port Reset bit or Port Resume bit in this register or the Resume/Remote WakeupDetected Interrupt bit or Disconnect Detected Interrupt bit in the Core Interrupt register (GINTSTS.WkUpInt or GINTSTS.DisconnInt, respectively). 0_B Port not in Suspend mode 1_B Port in Suspend mode This bit is set only by software and cleared only by hardware.
PrtRst	8	rw	**Port Reset** When the application sets this bit, a reset sequence is started on this port. The application must time the reset period and clear this bit after the reset sequence is complete. 0_B Port not in reset 1_B Port in reset To start a reset on the port, the application must leave this bit set for at least the minimum duration mentioned below, as specified in the USB 2.0 specification, Section 7.1.7.5. The application can leave it set for another 10 ms in addition to the required minimum duration, before clearing the bit, even though there is no maximum limit set by the USB standard. • Full speed: 10 ms
PrtLnSts	[11:10]	rh	**Port Line Status** Indicates the current logic level USB data lines Bit [10]: Logic level of D+ Bit [11]: Logic level of D-

Field	Bits	Type	Description
PrtPwr	12	rwh	**Port Power** The application uses this field to control power to this port, and the core can clear this bit on an over current condition. 0_B Power off 1_B Power on This bit is set only by software and can be cleared by hardware or a software write of 0 to the bit.
PrtSpd	[18:17]	rh	**Port Speed** Indicates the speed of the device attached to this port. 01_B Full speed Other values are reserved.
0	[31:19], [16:13], 9	r	**Reserved** Read as 0; should be written with 0.

Host Channel-n Characteristics Register (HCCHARx)

HCCHARx (x=0-13)
Host Channel-x Characteristics Register(500_H + x*20_H) Reset Value: 0000 0000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ChEna	**ChDis**	**OddFrm**	**DevAddr**							**MC_EC**		**EPType**		**0**	
rwh	rwh	rw	rw							rw		rw		r	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EPDir	**EPNum**				**MPS**										
rw	rw				rw										

Field	Bits	Type	Description
MPS	[10:0]	rw	**Maximum Packet Size** Indicates the maximum packet size of the associated endpoint.
EPNum	[14:11]	rw	**Endpoint Number** Indicates the endpoint number on the device serving as the data source or sink.
EPDir	15	rw	**Endpoint Direction** Indicates whether the transaction is IN or OUT. 0_B OUT 1_B IN
EPType	[19:18]	rw	**Endpoint Type** Indicates the transfer type selected. 00_B Control 01_B Isochronous 10_B Bulk 11_B Interrupt

Field	Bits	Type	Description
MC_EC	[21:20]	rw	**Multi Count / Error Count** This field indicates to the host the number of transactions that must be executed per frame for this periodic endpoint. For non periodic transfers, this field is used only in DMA mode, and specifies the number packets to be fetched for this channel before the internal DMA engine changes arbitration. 00_B Reserved. This field yields undefined results. 01_B 1 transaction 10_B 2 transactions to be issued for this endpoint per frame 11_B 3 transactions to be issued for this endpoint per frame
DevAddr	[28:22]	rw	**Device Address** This field selects the specific device serving as the data source or sink.
OddFrm	29	rw	**Odd Frame** This field is set (reset) by the application to indicate that the OTG host must perform a transfer in an odd frame. This field is applicable for only periodic (isochronous and interrupt) transactions. 0_B Even frame 1_B Odd frame This field is not applicable for Scatter/Gather DMA mode and need not be programmed by the application and is ignored by the core.
ChDis	30	rwh	**Channel Disable** The application sets this bit to stop transmitting/receiving data on a channel, even before the transfer for that channel is complete. The application must wait for the Channel Disabled interrupt before treating the channel as disabled. This bit can be set by software and can be cleared by both hardware and software.

Field	Bits	Type	Description
ChEna	31	rwh	**Channel Enable** When Scatter/Gather mode is enabled 0_B Indicates that the descriptor structure is not yet ready. 1_B Indicates that the descriptor structure and data buffer with data is setup and this channel can access the descriptor. When Scatter/Gather mode is disabled This field is set by the application and cleared by the OTG host. 0_B Channel disabled 1_B Channel enabled This bit is set only by software and cleared only by hardware.
0	[17:16]	r	**Reserved** Read as 0; should be written with 0.

Host Channel-n Interrupt Register (HCINTx)

This register indicates the status of a channel with respect to USB- and AHB-related events. It is shown in **Figure 16-67 "Interrupt Hierarchy" on Page 16-209**. The application must read this register when the Host Channels Interrupt bit of the Core Interrupt register (GINTSTS.HChInt) is set. Before the application can read this register, it must first read the Host All Channels Interrupt (HAINT) register to get the exact channel number for the Host Channel-n Interrupt register. The application must clear the appropriate bit in this register to clear the corresponding bits in the HAINT and GINTSTS registers.

Note: All bits of the access type 'rwh' in this register are set only by hardware and cleared only by a software write of 1 to the bits.

HCINTx (x=0-13)
Host Channel-x Interrupt Register(508_H + x*20_H) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		DESC_LST_ROLLIntr	XCS_XACT_ERR	BNAIntr	DataTglErr	FrmOvrun	BblErr	XactErr	NYET	ACK	NAK	STALL	AHBErr	ChHltd	XferCompl
r		rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
XferCompl	0	rwh	**Transfer Completed** For Scatter/Gather DMA mode, it indicates that current descriptor processing got completed with IOC bit set in its descriptor. In non Scatter/Gather DMA mode, it indicates that Transfer completed normally without any errors.
ChHltd	1	rwh	**Channel Halted** In non Scatter/Gather DMA mode, it indicates the transfer completed abnormally either because of any USB transaction error or in response to disable request by the application or because of a completed transfer. In Scatter/Gather DMA mode, this indicates that transfer completed due to any of the following • EOL being set in descriptor • AHB error • Excessive transaction errors • In response to disable request by the application • Babble • Stall • Buffer Not Available (BNA)
AHBErr	2	rwh	**AHB Error** This is generated only in DMA mode when there is an AHB error during AHB read/write. The application can read the corresponding channel's DMA address register to get the error address.
STALL	3	rwh	**STALL Response Received Interrupt** In Scatter/Gather DMA mode, the interrupt due to this bit is masked in the core.
NAK	4	rwh	**NAK Response Received Interrupt** In Scatter/Gather DMA mode, the interrupt due to this bit is masked in the core.
ACK	5	rwh	**ACK Response Received/Transmitted Interrupt** In Scatter/Gather DMA mode, the interrupt due to this bit is masked in the core.
NYET	6	rwh	**NYET Response Received Interrupt** In Scatter/Gather DMA mode, the interrupt due to this bit is masked in the core.

Field	Bits	Type	Description
XactErr	7	rwh	**Transaction Error** Indicates one of the following errors occurred on the USB. • CRC check failure • Timeout • Bit stuff error • False EOP In Scatter/Gather DMA mode, the interrupt due to this bit is masked in the core.
BblErr	8	rwh	**Babble Error** In Scatter/Gather DMA mode, the interrupt due to this bit is masked in the core.
FrmOvrun	9	rwh	**Frame Overrun** In Scatter/Gather DMA mode, the interrupt due to this bit is masked in the core
DataTglErr	10	rwh	**Data Toggle Error** In Scatter/Gather DMA mode, the interrupt due to this bit is masked in the core.
BNAIntr	11	rwh	**BNA (Buffer Not Available) Interrupt** This bit is valid only when Scatter/Gather DMA mode is enabled. The core generates this interrupt when the descriptor accessed is not ready for the Core to process. BNA will not be generated for Isochronous channels. For non Scatter/Gather DMA mode, this bit is reserved.
XCS_XACT_ERR	12	rwh	**Excessive Transaction Error** This bit is valid only when Scatter/Gather DMA mode is enabled. The core sets this bit when 3 consecutive transaction errors occurred on the USB bus. XCS_XACT_ERR will not be generated for Isochronous channels. For non Scatter/Gather DMA mode, this bit is reserved.
DESC_LST_ROLLIntr	13	rwh	**Descriptor rollover interrupt** This bit is valid only when Scatter/Gather DMA mode is enabled. The core sets this bit when the corresponding channel's descriptor list rolls over. For non Scatter/Gather DMA mode, this bit is reserved.
0	[31:14]	r	**Reserved** Read as 0; should be written with 0.

Host Channel-n Interrupt Mask Register (HCINTMSKx)

This register reflects the mask for each channel status described in register HCINTx.

- Mask interrupt: 0_B
- Unmask interrupt: 1_B

HCINTMSKx (x=0-13)
Host Channel-x Interrupt Mask Register($50C_H$ + x*20_H) Reset Value: 0000 0000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		DESC_LST_ROLLIntr	0	BNAIntrMsk	DataTglErrMsk	FrmOvrunMsk	BblErrMsk	XactErrMsk	NyetMsk	AckMsk	NakMsk	StallMsk	AHBErrMsk	ChHltdMsk	XferComplMsk
r		rw	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
XferComplMsk	0	rw	**Transfer Completed Mask**
ChHltdMsk	1	rw	**Channel Halted Mask**
AHBErrMsk	2	rw	**AHB Error Mask**
StallMsk	3	rw	**STALL Response Received Interrupt Mask** This bit is not applicable in Scatter/Gather DMA mode.
NakMsk	4	rw	**NAK Response Received Interrupt Mask** This bit is not applicable in Scatter/Gather DMA mode.
AckMsk	5	rw	**ACK Response Received/Transmitted Interrupt Mask** This bit is not applicable in Scatter/Gather DMA mode.
NyetMsk	6	rw	**NYET Response Received Interrupt Mask** This bit is not applicable in Scatter/Gather DMA mode.
XactErrMsk	7	rw	**Transaction Error Mask** This bit is not applicable in Scatter/Gather DMA mode

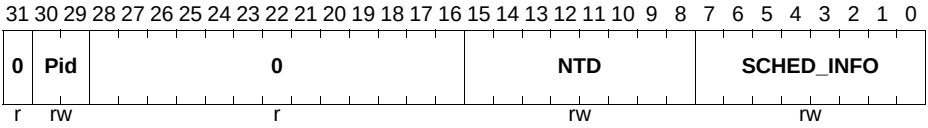
Field	Bits	Type	Description
BblErrMsk	8	rw	**Babble Error Mask** This bit is not applicable in Scatter/Gather DMA mode.
FrmOvrunMsk	9	rw	**Frame Overrun Mask** This bit is not applicable in Scatter/Gather DMA mode.
DataTglErrMsk	10	rw	**Data Toggle Error Mask** This bit is not applicable in Scatter/Gather DMA mode.
BNAIntrMsk	11	rw	**BNA (Buffer Not Available) Interrupt mask register** This bit is valid only when Scatter/Gather DMA mode is enabled. In non Scatter/Gather DMA mode, this bit is reserved
DESC_LST_ROLLIntrMsk	13	rw	**Descriptor rollover interrupt Mask register** This bit is valid only when Scatter/Gather DMA mode is enabled. In non Scatter/Gather DMA mode, this bit is reserved.
0	[31:14], 12	r	**Reserved** Read as 0; should be written with 0.

Host Channel-n Transfer Size Register (HCTSIZx)

The HCTSIZx register description depends on the selected DMA mode.

In **Scatter/Gather mode**, HCTSIZx is defined as follows:

HCTSIZx (x=0-13)
Host Channel-x Transfer Size Register [SCATGATHER]
(510_H + x*20_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
SCHED_INFO	[7:0]	rw	**Schedule information** This field should be written with $1111'1111_B$ for a FS Host.
NTD	[15:8]	rw	**Number of Transfer Descriptors (Non Isochronous)** This value is in terms of number of descriptors. Maximum number of descriptor that can be present in the list is 64. The values can be from 0 to 63. 0_D 1 descriptor $..._D$ ... 63_D 64 descriptors This field indicates the total number of descriptors present in that list. The core will wrap around after servicing NTD number of descriptors for that list. **(Isochronous)** This field indicates the number of descriptors present in that list frame The possible values are 1_D 2 descriptors 3_D 4 descriptors 7_D 8 descriptors 15_D 16 descriptors 31_D 32 descriptors 63_D 64 descriptors

Field	Bits	Type	Description
Pid	[30:29]	rw	**PID** The application programs this field with the type of PID to use for the initial transaction.The host maintains this field for the rest of the transfer. 00_B DATA0 01_B DATA2 10_B DATA1 11_B MDATA (non-control)
0	31, [28:16]	r	**Reserved** Read as 0; should be written with 0.

Host Channel-n Transfer Size Register (HCTSIZx)

In **Buffer DMA Mode**, HCTSIZx is defined as follows:

HCTSIZx (x=0-13)
Host Channel-x Transfer Size Register [BUFFERMODE]
(510_H + x*20_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**Pid**		**PktCnt**										**XferSize**		
r	rw		rw										rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XferSize															
rw															

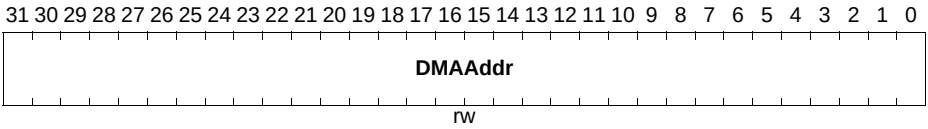
Field	Bits	Type	Description
XferSize	[18:0]	rw	**Transfer Size** For an OUT, this field is the number of data bytes the host sends during the transfer. For an IN, this field is the buffer size that the application has reserved for the transfer. The application is expected to program this field as an integer multiple of the maximum packet size for IN transactions (periodic and non-periodic).
PktCnt	[28:19]	rw	**Packet Count** This field is programmed by the application with the expected number of packets to be transmitted (OUT) or received (IN). The host decrements this count on every successful transmission or reception of an OUT/IN packet. Once this count reaches zero, the application is interrupted to indicate normal completion.
Pid	[30:29]	rw	**PID** The application programs this field with the type of PID to use for the initial transaction.The host maintains this field for the rest of the transfer. 00_B DATA0 01_B DATA2 10_B DATA1 11_B MDATA (non-control)/SETUP (control)
0	31	r	**Reserved** Read as 0; should be written with 0.

Host Channel-n DMA Address Register (HCDMAx)

This register is used by the OTG host in DMA mode to maintain the current buffer pointer for IN/OUT transactions. The starting DMA address must be DWORD-aligned. The HCDMAx register description depends on the selected DMA mode.

In **Buffer DMA Mode**, HCDMAx is defined as follows:

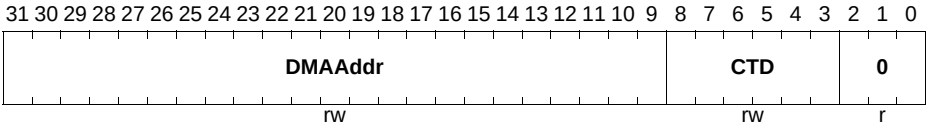
HCDMAx (x=0-13)
Host Channel-x DMA Address Register [BUFFERMODE]
(514_H + x*20_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
DMAAddr	[31:0]	rw	**DMA Address** This field holds the start address in the external memory from which the data for the endpoint must be fetched or to which it must be stored. This register is incremented on every AHB transaction.

In **Scatter/Gather DMA Mode**, HCDMAx is defined as follows:

HCDMAx (x=0-13)
Host Channel-x DMA Address Register [SCATGATHER]
(514_H + x*20_H) **Reset Value: 0000 0000_H**



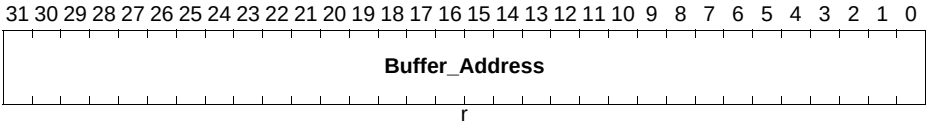
Field	Bits	Type	Description
CTD	[8:3]	rw	**Current Transfer Desc:** **Non Isochronous:** This value is in terms of number of descriptors. The values can be from 0 to 63. 0_D 1 descriptor $..._D$ ... 63_D 64 descriptors This field indicates the current descriptor processed in the list. This field is updated both by application and the core. For example, if the application enables the channel after programming CTD=5, then the core will start processing the 6th descriptor. The address is obtained by adding a value of (8 bytes*5=) 40 to DMAAddr. **Isochronous:** For isochronous transfers, the bits are [N-1:3]. CTD for isochronous is based on the current frame value. Need to be set to zero by application.

Field	Bits	Type	Description
DMAAddr	[31:9]	rw	**DMA Address** **Non-Isochronous:** This field holds the start address of the 512 bytes page. The first descriptor in the list should be located in this address. The first descriptor may be or may not be ready. The core starts processing the list from the CTD value. **Isochronous:** For isochronous transfers, the bits are [31:N]. This field holds the address of the 2*(nTD+1) bytes of locations in which the isochronous descriptors are present where N is based on nTD as listed below: nTD=1 => N=4 nTD=3 => N=5 nTD=7 => N=6 nTD=15 => N=7 nTD=31 => N=8 nTD=63 => N=9 *Note: For Scatter/Gather DMA mode, this address is the start of the page address where the descriptor list is located.*
0	[2:0]	r	**Reserved** Read as 0; should be written with 0.

Host Channel-n DMA Buffer Address Register (HCDMABx)

This register is present only in case of Scatter/Gather DMA. It is implemented in RAM. This register holds the current buffer address.

HCDMABx (x=0-13)
Host Channel-x DMA Buffer Address Register($51C_H$ + x*20_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
Buffer_Address	[31:0]	r	**Buffer Address** Holds the current buffer address. This register is updated as and when the data transfer for the corresponding end point is in progress. Reset: "X" if not programmed as the register is in SPRAM

Device Mode Registers

These registers are visible only in Device mode and must not be accessed in Host mode, as the results are unknown. Some of them affect all the endpoints uniformly, while others affect only a specific endpoint. Device Mode registers fall into two categories:

Device Logical IN Endpoint-Specific Registers

One set of endpoint registers is instantiated per logical endpoint. A logical endpoint is unidirectional: it can be either IN or OUT. To represent a bidirectional endpoint, two logical endpoints are required, one for the IN direction and the other for the OUT direction. This is also true for control endpoints.

The registers and register fields described in this section can pertain to IN or OUT endpoints, or both, or specific endpoint types as noted.

Device Configuration Register (DCFG)

This register configures the core in Device mode after power-on or after certain control commands or enumeration. Do not make changes to this register after initial programming.

DCFG
Device Configuration Register **(800_H)** **Reset Value: $0820\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0				1	0	PerSchIntvl		DescDMA	0	1	0			0	
r				r	r	rw		rw	r	r	r			r	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			PerFrInt		DevAddr							0	NZStsOUTHShk	DevSpd	
r			rw		rw							r	rw	rw	

Field	Bits	Type	Description
DevSpd	[1:0]	rw	**Device Speed** Indicates the speed at which the application requires the core to enumerate, or the maximum speed the application can support. However, the actual bus speed is determined only after the chirp sequence is completed, and is based on the speed of the USB host to which the core is connected. See **"Device Initialization" on Page 16-69** for details. 00_B Reserved 01_B Reserved 10_B Reserved 11_B Full speed (USB 1.1 transceiver clock is 48 MHz)
NZStsOUTHShk	2	rw	**Non-Zero-Length Status OUT Handshake** The application can use this field to select the handshake the core sends on receiving a nonzero-length data packet during the OUT transaction of a control transfer's Status stage. 1_B Send a STALL handshake on a nonzero-length status OUT transaction and do not send the received OUT packet to the application. 0_B Send the received OUT packet to the application (zero-length or nonzero-length) and send a handshake based on the NAK and STALL bits for the endpoint in the Device Endpoint Control register.
DevAddr	[10:4]	rw	**Device Address** The application must program this field after every SetAddress control command.
PerFrInt	[12:11]	rw	**Periodic Frame Interval** Indicates the time within a frame at which the application must be notified using the End Of Periodic Frame Interrupt. This can be used to determine if all the isochronous traffic for that frame is complete. 00_B 80% of the frame interval 01_B 85% 10_B 90% 11_B 95%

Field	Bits	Type	Description
DescDMA	23	rw	**Enable Scatter/Gather DMA in Device mode.** The application can set this bit during initialization to enable the Scatter/Gather DMA operation. *Note: This bit must be modified only once after a reset.* The following combinations are available for programming: • GAHBCFG.DMAEn=0,DCFG.DescDMA=0 => Slave mode • GAHBCFG.DMAEn=0,DCFG.DescDMA=1 => Invalid • GAHBCFG.DMAEn=1 ,DCFG.DescDMA=0 => Buffered DMA mode • GAHBCFG.DMAEn=1 ,DCFG.DescDMA=1 => Scatter/Gather DMA mode
PerSchIntvl	[25:24]	rw	**Periodic Scheduling Interval** PerSchIntvl must be programmed only for Scatter/Gather DMA mode. Description: This field specifies the amount of time the Internal DMA engine must allocate for fetching periodic IN endpoint data. Based on the number of periodic endpoints, this value must be specified as 25, 50 or 75% of frame. • When any periodic endpoints are active, the internal DMA engine allocates the specified amount of time in fetching periodic IN endpoint data. • When no periodic endpoints are active, then the internal DMA engine services non-periodic endpoints, ignoring this field. • After the specified time within a frame, the DMA switches to fetching for non-periodic endpoints. 00_B 25% of frame. 01_B 50% of frame. 10_B 75% of frame. 11_B Reserved.

Field	Bits	Type	Description
1	27, 21	r	**Reserved** Read as 1; should be written with 1.
0	[31:28], 26, 22, [20:18], [17:13], 3	r	**Reserved** Read as 0; should be written with 0.

Device Control Register (DCTL)

DCTL
Device Control Register **(804_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															NakOnBble
r															rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IgnrFrmNum	GMC		0		CGOUTNak	SGOUTNak	CGNPInNak	SGNPInNak	0			GOUTNakSts	GNPINNakSts	SftDiscon	RmtWkUpSig
rw	rw		r		w	w	w	w	r			rh	rh	rw	rw

Field	Bits	Type	Description
RmtWkUpSig	0	rw	**Remote Wakeup Signaling** When the application sets this bit, the core initiates remote signaling to wake the USB host. The application must set this bit to instruct the core to exit the Suspend state. As specified in the USB 2.0 specification, the application must clear this bit 1 to15 ms after setting it.
SftDiscon	1	rw	**Soft Disconnect** The application uses this bit to signal the USB core to do a soft disconnect. As long as this bit is set, the host does not see that the device is connected, and the device does not receive signals on the USB. The core stays in the disconnected state until the application clears this bit. The minimum duration for which the core must keep this bit set is specified in **Table 16-21**. 0_B Normal operation. When this bit is cleared after a soft disconnect, the core drives a device connect event to the USB host. When the device is reconnected, the USB host restarts device enumeration. 1_B The core drives a device disconnect event to the USB host.

Field	Bits	Type	Description
GNPINNakSts	2	rh	**Global Non-periodic IN NAK Status** 0_B A handshake is sent out based on the data availability in the transmit FIFO. 1_B A NAK handshake is sent out on all non-periodic IN endpoints, irrespective of the data availability in the transmit FIFO.
GOUTNakSts	3	rh	**Global OUT NAK Status** 0_B A handshake is sent based on the FIFO Status and the NAK and STALL bit settings. 1_B No data is written to the RxFIFO, irrespective of space availability. Sends a NAK handshake on all packets, except on SETUP transactions. All isochronous OUT packets are dropped.
SGNPInNak	7	w	**Set Global Non-periodic IN NAK** A write to this field sets the Global Non-periodic IN NAK.The application uses this bit to send a NAK handshake on all non-periodic IN endpoints. The core can also set this bit when a timeout condition is detected on a non-periodic endpoint in shared FIFO operation. The application must set this bit only after making sure that the Global IN NAK Effective bit in the Core Interrupt Register (GINTSTS.GINNakEff) is cleared.
CGNPInNak	8	w	**Clear Global Non-periodic IN NAK** A write to this field clears the Global Non-periodic IN NAK.
SGOUTNak	9	w	**Set Global OUT NAK** A write to this field sets the Global OUT NAK. The application uses this bit to send a NAK handshake on all OUT endpoints. The application must set the this bit only after making sure that the Global OUT NAK Effective bit in the Core Interrupt Register (GINTSTS.GOUTNakEff) is cleared.
CGOUTNak	10	w	**Clear Global OUT NAK** A write to this field clears the Global OUT NAK.

Field	Bits	Type	Description
GMC	[14:13]	rw	**Global Multi Count** GMC must be programmed only once after initialization. Applicable only for Scatter/Gather DMA mode. This indicates the number of packets to be serviced for that end point before moving to the next end point. It is only for non-periodic end points. 00_B Invalid. 01_B 1 packet. 10_B 2 packets. 11_B 3 packets. When Scatter/Gather DMA mode is disabled, this field is reserved. and reads 00_B.
IgnrFrmNum	15	rw	**Ignore frame number for isochronous endpoints in case of Scatter/Gather DMA** *Note: When this bit is enabled, there must be only one packet per descriptor.* 0_B The core transmits the packets only in the frame number in which they are intended to be transmitted. 1_B The core ignores the frame number, sending packets immediately as the packets are ready. Scatter/Gather: In Scatter/Gather DMA mode, when this bit is enabled, the packets are not flushed when an ISOC IN token is received for an elapsed frame. When Scatter/Gather DMA mode is disabled, this field is used by the application to enable periodic transfer interrupt. The application can program periodic endpoint transfers for multiple frames. 0_B Periodic transfer interrupt feature is disabled; the application must program transfers for periodic endpoints every frame 1_B Periodic transfer interrupt feature is enabled; the application can program transfers for multiple frames for periodic endpoints. In non-Scatter/Gather DMA mode, the application receives transfer complete interrupt after transfers for multiple frames are completed.

Field	Bits	Type	Description
NakOnBble	16	rw	**Set NAK automatically on babble** The core sets NAK automatically for the endpoint on which babble is received.
0	[31:17], [12:11], [6:4]	r	**Reserved** Read as 0; should be written with 0.

Table 16-21 lists the minimum duration under various conditions for which the Soft Disconnect (SftDiscon) bit must be set for the USB host to detect a device disconnect. To accommodate clock jitter, it is recommended that the application add some extra delay to the specified minimum duration.

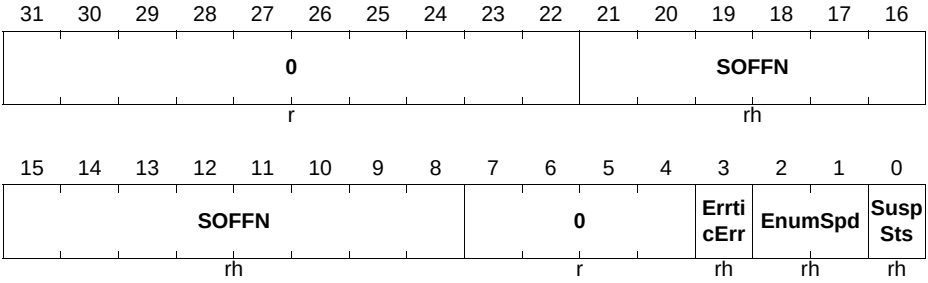
Table 16-21 Minimum Duration for Soft Disconnect

Operating Speed	Device State	Minimum Duration
Full speed	Suspended	1 ms + 2.5 µs
Full speed	Idle	2.5 µs
Full speed	Not Idle or Suspended (Performing transactions)	2.5 µs

Device Status Register (DSTS)

This register indicates the status of the core with respect to USB-related events. It must be read on interrupts from Device All Interrupts (DAINT) register.

DSTS
Device Status Register **(808_H)** **Reset Value: $0000\ 0002_H$**



Field	Bits	Type	Description
SuspSts	0	rh	**Suspend Status** In Device mode, this bit is set as long as a Suspend condition is detected on the USB. The core enters the Suspended state when there is no activity on the two USB data signals for an extended period of time. The core comes out of the suspend: • When there is any activity on the two USB data signals • When the application writes to the Remote Wakeup Signaling bit in the Device Control register (DCTL.RmtWkUpSig)
EnumSpd	[2:1]	rh	**Enumerated Speed** Indicates the speed at which the USB core has come up after speed detection through a chirp sequence. 00_B Reserved 01_B Reserved 10_B Reserved 11_B Full speed (PHY clock is running at 48 MHz)

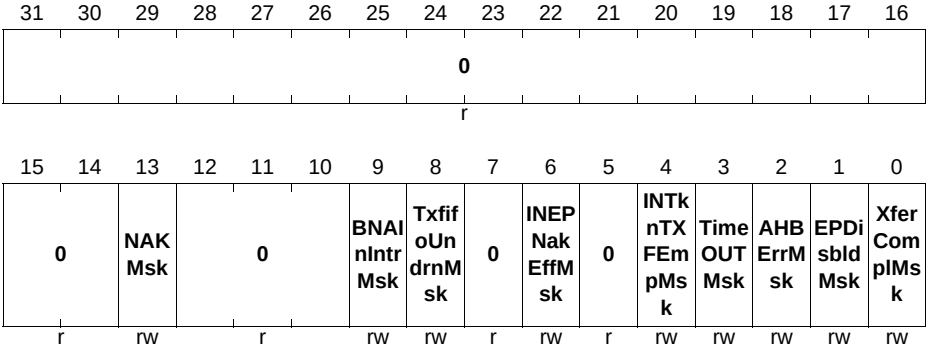
Field	Bits	Type	Description
ErrticErr	3	rh	**Erratic Error** The core sets this bit to report any erratic errors. Due to erratic errors, the USB core goes into Suspended state and an interrupt is generated to the application with Early Suspend bit of the Core Interrupt register (GINTSTS.ErlySusp). If the early suspend is asserted due to an erratic error, the application can only perform a soft disconnect recover.
SOFFN	[21:8]	rh	**Frame Number of the Received SOF** When the core is operating at full speed, this field contains a frame number.
0	[31:22], [7:4]	r	**Reserved** Read as 0; should be written with 0.

Device IN Endpoint Common Interrupt Mask Register (DIEPMSK)

This register works with each of the Device IN Endpoint Interrupt (DIEPINTx) registers for all endpoints to generate an interrupt per IN endpoint. The IN endpoint interrupt for a specific status in the DIEPINTx register can be masked by writing to the corresponding bit in this register. Status bits are masked by default.

- Mask interrupt: 0_B
- Unmask interrupt: 1_B

DIEPMSK
Device IN Endpoint Common Interrupt Mask Register **(810_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
XferComplMsk	0	rw	**Transfer Completed Interrupt Mask**
EPDisbldMsk	1	rw	**Endpoint Disabled Interrupt Mask**
AHBErrMsk	2	rw	**AHB Error Mask**
TimeOUTMsk	3	rw	**Timeout Condition Mask** (Non-isochronous endpoints)
INTknTXFEmpMsk	4	rw	**IN Token Received When TxFIFO Empty Mask**
INEPNakEffMsk	6	rw	**IN Endpoint NAK Effective Mask**
TxfifoUndrnMsk	8	rw	**Fifo Underrun Mask**
BNAInIntrMsk	9	rw	**BNA Interrupt Mask**

Field	Bits	Type	Description
NAKMsk	13	rw	**NAK interrupt Mask**
0	[31:14], [12:10], 7, 5	r	**Reserved** Read as 0; should be written with 0.

Device OUT Endpoint Common Interrupt Mask Register (DOEPMSK)

This register works with each of the Device OUT Endpoint Interrupt (DOEPINTx) registers for all endpoints to generate an interrupt per OUT endpoint. The OUT endpoint interrupt for a specific status in the DOEPINTx register can be masked by writing into the corresponding bit in this register. Status bits are masked by default.

- Mask interrupt: 0_B
- Unmask interrupt: 1_B

DOEPMSK
Device OUT Endpoint Common Interrupt Mask Register **(814_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	NYETMsk	NAKMsk	BbleErrMsk	0		BnaOutIntrMsk	OutPktErrMsk	0	Back2BackSETup	0	OUTTknEPdisMsk	SetUPMsk	AHBErrMsk	EPDisbldMsk	XferComplMsk
r	rw	rw	rw	r		rw	rw	r	rw	r	rw	rw	rw	rw	rw

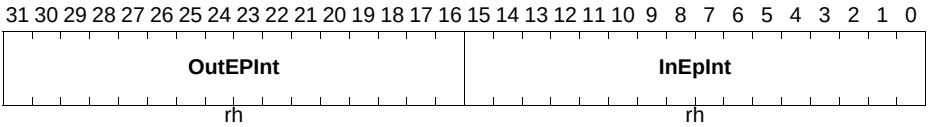
Field	Bits	Type	Description
XferComplMsk	0	rw	**Transfer Completed Interrupt Mask**
EPDisbldMsk	1	rw	**Endpoint Disabled Interrupt Mask**
AHBErrMsk	2	rw	**AHB Error**
SetUPMsk	3	rw	**SETUP Phase Done Mask** Applies to control endpoints only.
OUTTknEPdisMsk	4	rw	**OUT Token Received when Endpoint Disabled Mask** Applies to control OUT endpoints only.
Back2BackSETup	6	rw	**Back-to-Back SETUP Packets Received Mask** Applies to control OUT endpoints only.
OutPktErrMsk	8	rw	**OUT Packet Error Mask**
BnaOutIntrMsk	9	rw	**BNA interrupt Mask**
BbleErrMsk	12	rw	**Babble Interrupt Mask**
NAKMsk	13	rw	**NAK Interrupt Mask**

Field	Bits	Type	Description
NYETMsk	14	rw	**NYET Interrupt Mask**
0	[31:15], [11:10], 7, 5	r	**Reserved** Read as 0; should be written with 0.

Device All Endpoints Interrupt Register (DAINT)

When a significant event occurs on an endpoint, a Device All Endpoints Interrupt register interrupts the application using the Device OUT Endpoints Interrupt bit or Device IN Endpoints Interrupt bit of the Core Interrupt register (GINTSTS.OEPInt or GINTSTS.IEPInt, respectively). This is shown in **Figure 16-67 "Interrupt Hierarchy" on Page 16-209**. There is one interrupt bit per endpoint, up to a maximum of 7 bits for OUT endpoints and 7 bits for IN endpoints. For a bidirectional endpoint, the corresponding IN and OUT interrupt bits are used. Bits in this register are set and cleared when the application sets and clears bits in the corresponding Device Endpoint-n Interrupt register (DIEPINTx/DOEPINTx).

DAINT
Device All Endpoints Interrupt Register(818_H) **Reset Value: $0000\ 0000_H$**



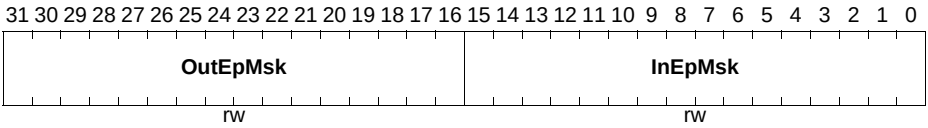
Field	Bits	Type	Description
InEpInt	[15:0]	rh	**IN Endpoint Interrupt Bits** One bit per IN Endpoint: Bit 0 for IN endpoint 0, bit 6 for endpoint 6. Bits [15:7] are not used.
OutEPInt	[31:16]	rh	**OUT Endpoint Interrupt Bits** One bit per OUT endpoint: Bit 16 for OUT endpoint 0, bit 22 for OUT endpoint 6. Bits [31:23] are not used.

Device All Endpoints Interrupt Mask Register (DAINTMSK)

The Device Endpoint Interrupt Mask register works with the Device Endpoint Interrupt register to interrupt the application when an event occurs on a device endpoint. However, the Device All Endpoints Interrupt (DAINT) register bit corresponding to that interrupt is still set.

- Mask Interrupt: 0_B
- Unmask Interrupt: 1_B

DAINTMSK
Device All Endpoints Interrupt Mask Register($81C_H$) **Reset Value: 0000 0000$_H$**

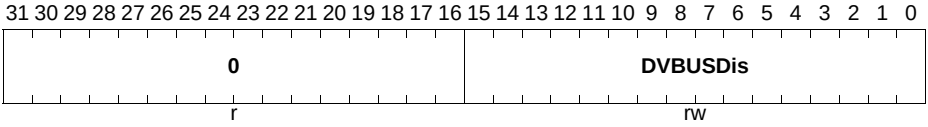


Field	Bits	Type	Description
InEpMsk	[15:0]	rw	**IN EP Interrupt Mask Bits** One bit per IN Endpoint: Bit 0 for IN EP 0, bit 6 for IN EP 6. Bits [15:7] are not used.
OutEpMsk	[31:16]	rw	**OUT EP Interrupt Mask Bits** One per OUT Endpoint: Bit 16 for OUT EP 0, bit 22 for OUT EP 6. Bits [31:23] are not used.

Device VBUS Discharge Time Register (DVBUSDIS)

This register specifies the VBUS discharge time after VBUS pulsing during SRP.

DVBUSDIS
Device VBUS Discharge Time Register(828_H) **Reset Value: 0000 17D7$_H$**

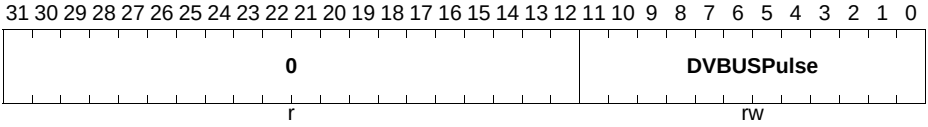


Field	Bits	Type	Description
DVBUSDis	[15:0]	rw	**Device Vbus Discharge Time** Specifies the Vbus discharge time after Vbus pulsing during SRP. This value equals: Vbus discharge time in PHY clocks / 1,024 The reset value is based on PHY operating at 60 MHz. Depending on the Vbus load, this value might need adjustment.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

Device VBUS Pulsing Time Register (DVBUSPULSE)

This register specifies the VBUS pulsing time during SRP.

DVBUSPULSE
Device VBUS Pulsing Time Register ($82C_H$) **Reset Value: 0000 $05B8_H$**



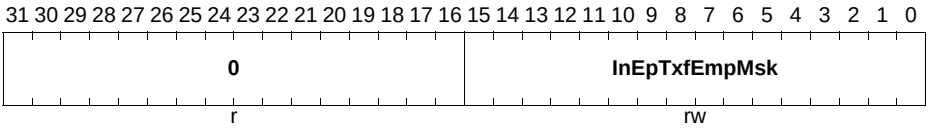
Field	Bits	Type	Description
DVBUSPulse	[11:0]	rw	**Device Vbus Pulsing Time** Specifies the Vbus pulsing time during SRP. This value equals: Vbus pulsing time in PHY clocks / 1,024 The reset value is based on PHY operating at 60 MHz.
0	[31:12]	r	**Reserved** Read as 0; should be written with 0.

Device IN Endpoint FIFO Empty Interrupt Mask Register (DIEPEMPMSK)

This register is used to control the IN endpoint FIFO empty interrupt generation (DIEPINTx.TxfEmp).

- Mask interrupt: 0_B
- Unmask interrupt: 1_B

DIEPEMPMSK
Device IN Endpoint FIFO Empty Interrupt Mask Register **(834_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
InEpTxfEmpMsk	[15:0]	rw	**IN EP Tx FIFO Empty Interrupt Mask Bits** These bits acts as mask bits for DIEPINTx. TxFEmp interrupt One bit per IN Endpoint: • Bit 0 for IN endpoint 0 • ... • Bit 6 for endpoint 6 Bits [15:7] are not used.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

Device Control IN Endpoint 0 Control Register (DIEPCTL0)

This section describes the Control IN Endpoint 0 Control register. Non-zero control endpoints use registers for endpoints 1-6.

DIEPCTL0
Device Control IN Endpoint 0 Control Register(900_H) **Reset Value: $0000\ 8000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EPEna	EPDis	0		SNAK	CNAK	TxFNum				Stall	0	EPType		NAKSts	0
rwh	rwh	r		w	w	rw				rwh	r	r		rh	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USBActEP	0													MPS	
r	r													rw	

Field	Bits	Type	Description
MPS	[1:0]	rw	**Maximum Packet Size** Applies to IN and OUT endpoints. The application must program this field with the maximum packet size for the current logical endpoint. 00_B 64 bytes 01_B 32 bytes 10_B 16 bytes 11_B 8 bytes
USBActEP	15	r	**USB Active Endpoint** This bit is always set to 1, indicating that control endpoint 0 is always active in all configurations and interfaces.
NAKSts	17	rh	**NAK Status** Indicates the following: 0_B The core is transmitting non-NAK handshakes based on the FIFO status 1_B The core is transmitting NAK handshakes on this endpoint. When this bit is set, either by the application or core, the core stops transmitting data, even if there is data available in the TxFIFO. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.

Field	Bits	Type	Description
EPType	[19:18]	r	**Endpoint Type** Hardcoded to 00_B for control.
Stall	21	rwh	**STALL Handshake** The application can only set this bit, and the core clears it, when a SETUP token is received for this endpoint. If a NAK bit, Global Non-periodic IN NAK, or Global OUT NAK is set along with this bit, the STALL bit takes priority. This bit is set only by software and cleared only by hardware.
TxFNum	[25:22]	rw	**TxFIFO Number** • This value is set to the FIFO number that is assigned to IN Endpoint 0.
CNAK	26	w	**Clear NAK** A write to this bit clears the NAK bit for the endpoint.
SNAK	27	w	**Set NAK** A write to this bit sets the NAK bit for the endpoint. Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The core can also set this bit for an endpoint after a SETUP packet is received on that endpoint.
EPDis	30	rwh	**Endpoint Disable** The application sets this bit to stop transmitting data on an endpoint, even before the transfer for that endpoint is complete. The application must wait for the Endpoint Disabled interrupt before treating the endpoint as disabled. The core clears this bit before setting the Endpoint Disabled Interrupt. The application must set this bit only if Endpoint Enable is already set for this endpoint. This bit is set only by software and cleared only by hardware.

Field	Bits	Type	Description
EPEna	31	rwh	**Endpoint Enable** • When Scatter/Gather DMA mode is enabled, for IN endpoints this bit indicates that the descriptor structure and data buffer with data ready to transmit is setup. • When Scatter/Gather DMA mode is disabled—such as in buffer-pointer based DMA mode—this bit indicates that data is ready to be transmitted on the endpoint. The core clears this bit before setting the following interrupts on this endpoint: • Endpoint Disabled • Transfer Completed This bit is set only by software and cleared only by hardware.
0	[29:28], 20, 16, [14:2]	r	**Reserved** Read as 0; should be written with 0.

Device Control OUT Endpoint 0 Control Register (DOEPCTL0)

This section describes the Control OUT Endpoint 0 Control register. Non-zero control endpoints use registers for endpoints 1-6.

DOEPCTL0
Device Control OUT Endpoint 0 Control Register($B00_H$) Reset Value: $0000\ 8000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EPEna	EPDis	0		SNAK	CNAK	0				Stall	Snp	EPType		NAKSts	0
rwh	r	r		w	w	r				rwh	rw	r		rh	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USBActEP	0													MPS	
r	r													r	

Field	Bits	Type	Description
MPS	[1:0]	r	**Maximum Packet Size** The maximum packet size for control OUT endpoint 0 is the same as what is programmed in control IN Endpoint 0. 00_B 64 bytes 01_B 32 bytes 10_B 16 bytes 11_B 8 bytes
USBActEP	15	r	**USB Active Endpoint** This bit is always set to 1, indicating that a control endpoint 0 is always active in all configurations and interfaces.
NAKSts	17	rh	**NAK Status** Indicates the following: 0_B The core is transmitting non-NAK handshakes based on the FIFO status. 1_B The core is transmitting NAK handshakes on this endpoint. When either the application or the core sets this bit, the core stops receiving data, even if there is space in the RxFIFO to accommodate the incoming packet. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.

Field	Bits	Type	Description
EPType	[19:18]	r	**Endpoint Type** Hardcoded to 00 for control.
Snp	20	rw	**Snoop Mode** This bit configures the endpoint to Snoop mode. In Snoop mode, the core does not check the correctness of OUT packets before transferring them to application memory.
Stall	21	rwh	**STALL Handshake** The application can only set this bit, and the core clears it, when a SETUP token is received for this endpoint. If a NAK bit or Global OUT NAK is set along with this bit, the STALL bit takes priority. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake. This bit is set only by software and cleared only by hardware.
CNAK	26	w	**Clear NAK** A write to this bit clears the NAK bit for the endpoint.
SNAK	27	w	**Set NAK** A write to this bit sets the NAK bit for the endpoint. Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The core can also set bit on a Transfer Completed interrupt, or after a SETUP is received on the endpoint.
EPDis	30	r	**Endpoint Disable** The application cannot disable control OUT endpoint 0.

Field	Bits	Type	Description
EPEna	31	rwh	**Endpoint Enable** When Scatter/Gather DMA mode is enabled, for OUT endpoints this bit indicates that the descriptor structure and data buffer to receive data is setup. • When Scatter/Gather DMA mode is disabled—(such as for buffer-pointer based DMA mode)—this bit indicates that the application has allocated the memory to start receiving data from the USB. The core clears this bit before setting any of the following interrupts on this endpoint: • SETUP Phase Done • Endpoint Disabled • Transfer Completed *Note: In DMA mode, this bit must be set for the core to transfer SETUP data packets into memory.* This bit is set only by software and cleared only by hardware.
0	[29:28], [25:22], 16, [14:2]	r	**Reserved** Read as 0; should be written with 0.

Device Endpoint-n Control Register (DIEPCTLx/DOEPCTLx)

The application uses this register to control the behavior of each logical endpoint other than endpoint 0.

Note: The fields of the DIEPCTLx/DOEPCTLx register change, depending on interrupt/bulk or isochronous/control endpoint.

DIEPCTLx (x=1-6)
Device Endpoint-x Control Register [INTBULK]
(900_H + x*20_H) **Reset Value: 0000 0000_H**

DOEPCTLx (x=1-6)
Device Endpoint-x Control Register [INTBULK]
($B00_H$ + x*20_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EPEna	**EPDis**	**SetD1PID**	**SetD0PID**	**SNAK**	**CNAK**	**TxFNum**				**Stall**	**Snp**	**EPType**		**NAKSts**	**DPID**
rwh	rwh	w	w	w	w	rw				rw	rw	rw		rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USBActEP	**0**				**MPS**										
rwh	r				rw										

Field	Bits	Type	Description
MPS	[10:0]	rw	**Maximum Packet Size** Applies to IN and OUT endpoints. The application must program this field with the maximum packet size for the current logical endpoint. This value is in bytes.
USBActEP	15	rwh	**USB Active Endpoint** Applies to IN and OUT endpoints. Indicates whether this endpoint is active in the current configuration and interface. The core clears this bit for all endpoints (other than EP 0) after detecting a USB reset. After receiving the SetConfiguration and SetInterface commands, the application must program endpoint registers accordingly and set this bit. This bit is set only by software and can be cleared by hardware or a software write of 0 to the bit.

Field	Bits	Type	Description
DPID	16	rh	**Endpoint Data PID** Applies to interrupt/bulk IN and OUT endpoints only. Contains the PID of the packet to be received or transmitted on this endpoint. The application must program the PID of the first packet to be received or transmitted on this endpoint, after the endpoint is activated. The applications use the SetD1PID and SetD0PID fields of this register to program either DATA0 or DATA1 PID. 0_B DATA0 1_B DATA1 This field is applicable both for Scatter/Gather DMA mode and non-Scatter/Gather DMA mode.
NAKSts	17	rh	**NAK Status** Applies to IN and OUT endpoints. Indicates the following: 0_B The core is transmitting non-NAK handshakes based on the FIFO status. 1_B The core is transmitting NAK handshakes on this endpoint. When either the application or the core sets this bit: The core stops receiving any data on an OUT endpoint, even if there is space in the RxFIFO to accommodate the incoming packet. For non-isochronous IN endpoints: The core stops transmitting any data on an IN endpoint, even if there data is available in the TxFIFO. For isochronous IN endpoints: The core sends out a zero-length data packet, even if there data is available in the TxFIFO. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.
EPType	[19:18]	rw	**Endpoint Type** Applies to IN and OUT endpoints. This is the transfer type supported by this logical endpoint. 00_B Control 01_B Isochronous 10_B Bulk 11_B Interrupt

Field	Bits	Type	Description
Snp	20	rw	**Snoop Mode** Applies to OUT endpoints only. This bit configures the endpoint to Snoop mode. In Snoop mode, the core does not check the correctness of OUT packets before transferring them to application memory.
Stall	21	rw	**STALL Handshake** Applies to non-control, non-isochronous IN and OUT endpoints only. The application sets this bit to stall all tokens from the USB host to this endpoint. If a NAK bit, Global Non-periodic IN NAK, or Global OUT NAK is set along with this bit, the STALL bit takes priority. Only the application can clear this bit, never the core.
TxFNum	[25:22]	rw	**TxFIFO Number** **T**hese bits specify the FIFO number associated with this endpoint. Each active IN endpoint must be programmed to a separate FIFO number. This field is valid only for IN endpoints.
CNAK	26	w	**Clear NAK** Applies to IN and OUT endpoints. A write to this bit clears the NAK bit for the endpoint.
SNAK	27	w	**Set NAK** Applies to IN and OUT endpoints.A write to this bit sets the NAK bit for the endpoint. Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The core can also set this bit for OUT endpoints on a Transfer Completed interrupt, or after a SETUP is received on the endpoint.
SetD0PID	28	w	**Set DATA0 PID** Applies to interrupt/bulk IN and OUT endpoints only. Writing to this field sets the Endpoint Data PID (DPID) field in this register to DATA0. This field is applicable both for Scatter/Gather DMA mode and non-Scatter/Gather DMA mode.

Field	Bits	Type	Description
SetD1PID	29	w	**29 Set DATA1 PID** Applies to interrupt/bulk IN and OUT endpoints only. Writing to this field sets the Endpoint Data PID (DPID) field in this register to DATA1. This field is applicable both for Scatter/Gather DMA mode and non-Scatter/Gather DMA mode.
EPDis	30	rwh	**Endpoint Disable** Applies to IN and OUT endpoints. The application sets this bit to stop transmitting/receiving data on an endpoint, even before the transfer for that endpoint is complete. The application must wait for the Endpoint Disabled interrupt before treating the endpoint as disabled. The core clears this bit before setting the Endpoint Disabled interrupt. The application must set this bit only if Endpoint Enable is already set for this endpoint. This bit is set only by software and cleared only by hardware.

Field	Bits	Type	Description
EPEna	31	rwh	**Endpoint Enable** Applies to IN and OUT endpoints. • When Scatter/Gather DMA mode is enabled, • For IN endpoints this bit indicates that the descriptor structure and data buffer with data ready to transmit is setup. • For OUT endpoint it indicates that the descriptor structure and data buffer to receive data is setup. • When Scatter/Gather DMA mode is enabled—such as for buffer-pointer based DMA mode: – For IN endpoints, this bit indicates that data is ready to be transmitted on the endpoint. – For OUT endpoints, this bit indicates that the application has allocated the memory to start receiving data from the USB. – The core clears this bit before setting any of the following interrupts on this endpoint: • SETUP Phase Done • Endpoint Disabled • Transfer Completed *Note: For control endpoints in DMA mode, this bit must be set to be able to transfer SETUP data packets in memory.* This bit is set only by software and cleared only by hardware.
0	[14:11]	r	**Reserved** Read as 0; should be written with 0.

DIEPCTLx (x=1-6)
Device Endpoint-x Control Register [ISOCONT]
(900_H + x*20_H) **Reset Value: 0000 0000_H**

DOEPCTLx (x=1-6)
Device Endpoint-x Control Register [ISOCONT]
($B00_H$ + x*20_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EPEna	EPDis	SetOddFr	SetEvenFr	SNAK	CNAK	TxFNum				Stall	Snp	EPType		NAKSts	EO_FrNum
rwh	rwh	w	w	w	w	rw				rwh	rw	rw		rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USBActEP	0				MPS										
rwh	r				rw										

Field	Bits	Type	Description
MPS	[10:0]	rw	**Maximum Packet Size** Applies to IN and OUT endpoints. The application must program this field with the maximum packet size for the current logical endpoint. This value is in bytes.
USBActEP	15	rwh	**USB Active Endpoint** Applies to IN and OUT endpoints. Indicates whether this endpoint is active in the current configuration and interface. The core clears this bit for all endpoints (other than EP 0) after detecting a USB reset. After receiving the SetConfiguration and SetInterface commands, the application must program endpoint registers accordingly and set this bit. This bit is set only by software and can be cleared by hardware or a software write of 0 to the bit.

Field	Bits	Type	Description
EO_FrNum	16	rh	**Even/Odd Frame** Applies to isochronous IN and OUT endpoints only. In non-Scatter/Gather DMA mode, the bit Indicates the frame number in which the core transmits/receives isochronous data for this endpoint. The application must program the even/odd frame number in which it intends to transmit/receive isochronous data for this endpoint using the SetEvnFr and SetOddFr fields in this register. 0_B Even frame 1_B Odd rame When Scatter/Gather DMA mode is enabled, this field is reserved. The frame number in which to send data is provided in the transmit descriptor structure. The frame in which data is received is updated in receive descriptor structure.
NAKSts	17	rh	**NAK Status** Applies to IN and OUT endpoints. Indicates the following: 0_B The core is transmitting non-NAK handshakes based on the FIFO status. 1_B The core is transmitting NAK handshakes on this endpoint. When either the application or the core sets this bit: The core stops receiving any data on an OUT endpoint, even if there is space in the RxFIFO to accommodate the incoming packet. For non-isochronous IN endpoints: The core stops transmitting any data on an IN endpoint, even if there data is available in the TxFIFO. For isochronous IN endpoints: The core sends out a zero-length data packet, even if there data is available in the TxFIFO. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.

Field	Bits	Type	Description
EPType	[19:18]	rw	**Endpoint Type** Applies to IN and OUT endpoints. This is the transfer type supported by this logical endpoint. 00_B Control 01_B Isochronous 10_B Bulk 11_B Interrupt
Snp	20	rw	**Snoop Mode** Applies to OUT endpoints only. This bit configures the endpoint to Snoop mode. In Snoop mode, the core does not check the correctness of OUT packets before transferring them to application memory.
Stall	21	rwh	**STALL Handshake** Applies to control endpoints only. The application can only set this bit, and the core clears it, when a SETUP token is received for this endpoint. If a NAK bit, Global Non-periodic IN NAK, or Global OUT NAK is set along with this bit, the STALL bit takes priority. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake. This bit is set only by software and cleared only by hardware.
TxFNum	[25:22]	rw	**TxFIFO Number** **T**hese bits specify the FIFO number associated with this endpoint. Each active IN endpoint must be programmed to a separate FIFO number. This field is valid only for IN endpoints.
CNAK	26	w	**Clear NAK** Applies to IN and OUT endpoints. A write to this bit clears the NAK bit for the endpoint.

Field	Bits	Type	Description
SNAK	27	w	**Set NAK** Applies to IN and OUT endpoints.A write to this bit sets the NAK bit for the endpoint. Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The core can also set this bit for OUT endpoints on a Transfer Completed interrupt, or after a SETUP is received on the endpoint.
SetEvenFr	28	w	**In non-Scatter/Gather DMA mode: Set Even frame** Applies to isochronous IN and OUT endpoints only. Writing to this field sets the Even/Odd frame (EO_FrNum) field to even frame. When Scatter/Gather DMA mode is enabled, this field is reserved. The frame number in which to send data is in the transmit descriptor structure. The frame in which to receive data is updated in receive descriptor structure.
SetOddFr	29	w	**Set Odd frame** Applies to isochronous IN and OUT endpoints only. Writing to this field sets the Even/Odd frame (EO_FrNum) field to odd frame. This field is not applicable for Scatter/Gather DMA mode.
EPDis	30	rwh	**Endpoint Disable** Applies to IN and OUT endpoints. The application sets this bit to stop transmitting/receiving data on an endpoint, even before the transfer for that endpoint is complete. The application must wait for the Endpoint Disabled interrupt before treating the endpoint as disabled. The core clears this bit before setting the Endpoint Disabled interrupt. The application must set this bit only if Endpoint Enable is already set for this endpoint. This bit is set only by software and cleared only by hardware.

Field	Bits	Type	Description
EPEna	31	rwh	**Endpoint Enable** Applies to IN and OUT endpoints. • When Scatter/Gather DMA mode is enabled, • For IN endpoints this bit indicates that the descriptor structure and data buffer with data ready to transmit is setup. • For OUT endpoint it indicates that the descriptor structure and data buffer to receive data is setup. • When Scatter/Gather DMA mode is enabled—such as for buffer-pointer based DMA mode: – For IN endpoints, this bit indicates that data is ready to be transmitted on the endpoint. – For OUT endpoints, this bit indicates that the application has allocated the memory to start receiving data from the USB. – The core clears this bit before setting any of the following interrupts on this endpoint: • SETUP Phase Done • Endpoint Disabled • Transfer Completed *Note: For control endpoints in DMA mode, this bit must be set to be able to transfer SETUP data packets in memory.* This bit is set only by software and cleared only by hardware.
0	[14:11]	r	**Reserved** Read as 0; should be written with 0.

Device Endpoint-n Interrupt Register (DIEPINTx/DOEPINTx)

This register indicates the status of an endpoint with respect to USB- and AHB-related events. It is shown in **Figure 16-67 "Interrupt Hierarchy" on Page 16-209**. The application must read this register when the OUT Endpoints Interrupt bit or IN Endpoints Interrupt bit of the Core Interrupt register (GINTSTS.OEPInt or GINTSTS.IEPInt, respectively) is set. Before the application can read this register, it must first read the Device All Endpoints Interrupt (DAINT) register to get the exact endpoint number for the Device Endpoint-n Interrupt register. The application must clear the appropriate bit in this register to clear the corresponding bits in the DAINT and GINTSTS registers.

Note: In the DIEPINTx/DOEPINTx registers, status bits with access type 'rwh' are set by hardware. To clear these bits, the application must write 1 into these bits.

DIEPINTx (x=0-6)
Device Endpoint-x Interrupt Register (908_H + x*20_H) **Reset Value: 0000 0080_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						BNAIntr	0	TxFEmp	INEPNakEff	0	INTknTXFEmp	TimeOUT	AHBErr	EPDisbld	XferCompl
r						rwh	r	r	rwh	r	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
XferCompl	0	rwh	**Transfer Completed Interrupt** Applies to IN and OUT endpoints. • When Scatter/Gather DMA mode is enabled • For IN endpoint this field indicates that the requested data from the descriptor is moved from external system memory to internal FIFO. • For OUT endpoint this field indicates that the requested data from the internal FIFO is moved to external system memory. This interrupt is generated only when the corresponding endpoint descriptor is closed, and the IOC bit for the corresponding descriptor is set. • When Scatter/Gather DMA mode is disabled, this field indicates that the programmed transfer is complete on the AHB as well as on the USB, for this endpoint.
EPDisbld	1	rwh	**Endpoint Disabled Interrupt** Applies to IN and OUT endpoints. This bit indicates that the endpoint is disabled per the application's request.
AHBErr	2	rwh	**AHB Error** Applies to IN and OUT endpoints. This is generated only in DMA mode when there is an AHB error during an AHB read/write. The application can read the corresponding endpoint DMA address register to get the error address.
TimeOUT	3	rwh	**Timeout Condition** • Applies only to Control IN endpoints. • In Scatter/Gather DMA mode, the TimeOUT interrupt is not asserted. Indicates that the core has detected a timeout condition on the USB for the last IN token on this endpoint.
INTknTXFEmp	4	rwh	**IN Token Received When TxFIFO is Empty** Indicates that an IN token was received when the associated TxFIFO (periodic/non-periodic) was empty. This interrupt is asserted on the endpoint for which the IN token was received.

Field	Bits	Type	Description
INEPNakEff	6	rwh	**IN Endpoint NAK Effective** Applies to periodic IN endpoints only. This bit can be cleared when the application clears the IN endpoint NAK by writing to DIEPCTLx.CNAK. This interrupt indicates that the core has sampled the NAK bit set (either by the application or by the core). The interrupt indicates that the IN endpoint NAK bit set by the application has taken effect in the core. This interrupt does not guarantee that a NAK handshake is sent on the USB. A STALL bit takes priority over a NAK bit. This bit is applicable only when the endpoint is enabled.
TxFEmp	7	r	**Transmit FIFO Empty** This bit is valid only for IN Endpoints This interrupt is asserted when the TxFIFO for this endpoint is either half or completely empty. The half or completely empty status is determined by the TxFIFO Empty Level bit in the Core AHB Configuration register (GAHBCFG.NPTxFEmpLvl)).
BNAIntr	9	rwh	**BNA (Buffer Not Available) Interrupt** The core generates this interrupt when the descriptor accessed is not ready for the Core to process, such as Host busy or DMA done This bit is valid only when Scatter/Gather DMA mode is enabled.
0	[31:10], 8, 5	r	**Reserved** Read as 0; should be written with 0.

DOEPINTx (x=0-6)

Device Endpoint-x Interrupt Register ($B08_H$ + x*20_H) **Reset Value: 0000 0080_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	NYETIntrpt	NAKIntrpt	BbleErrIntrpt	PktDrpStss	0	BNAIntr	0		Back2BackSETup	StsPhseRcvd	OUTTknEPdis	SetUp	AHBErr	EPDisbld	XferCompl
r	rwh	rwh	rwh	rwh	r	rwh	r		rw	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
XferCompl	0	rwh	**Transfer Completed Interrupt** Applies to IN and OUT endpoints. • When Scatter/Gather DMA mode is enabled • For IN endpoint this field indicates that the requested data from the descriptor is moved from external system memory to internal FIFO. • For OUT endpoint this field indicates that the requested data from the internal FIFO is moved to external system memory. This interrupt is generated only when the corresponding endpoint descriptor is closed, and the IOC bit for the corresponding descriptor is set. • When Scatter/Gather DMA mode is disabled, this field indicates that the programmed transfer is complete on the AHB as well as on the USB, for this endpoint.
EPDisbld	1	rwh	**Endpoint Disabled Interrupt** Applies to IN and OUT endpoints. This bit indicates that the endpoint is disabled per the application's request.
AHBErr	2	rwh	**AHB Error** Applies to IN and OUT endpoints. This is generated only in DMA mode when there is an AHB error during an AHB read/write. The application can read the corresponding endpoint DMA address register to get the error address.

Field	Bits	Type	Description
SetUp	3	rwh	**SETUP Phase Done** Applies to control OUT endpoints only. Indicates that the SETUP phase for the control endpoint is complete and no more back-to-back SETUP packets were received for the current control transfer. On this interrupt, the application can decode the received SETUP data packet.
OUTTknEPdis	4	rwh	**OUT Token Received When Endpoint Disabled** Indicates that an OUT token was received when the endpoint was not yet enabled. This interrupt is asserted on the endpoint for which the OUT token was received.
StsPhseRcvd	5	rwh	**Status Phase Received For Control Write** This interrupt is valid only for Control OUT endpoints and only in Scatter Gather DMA mode. This interrupt is generated only after the core has transferred all the data that the host has sent during the data phase of a control write transfer, to the system memory buffer. The interrupt indicates to the application that the host has switched from data phase to the status phase of a Control Write transfer. The application can use this interrupt to ACK or STALL the Status phase, after it has decoded the data phase. This is applicable only in case of Scatter Gather DMA mode.
Back2BackSETup	6	rw	**Back-to-Back SETUP Packets Received** Applies to Control OUT endpoints only. This bit indicates that the core has received more than three back-to-back SETUP packets for this particular endpoint. For information about handling this interrupt, see **“Handling More Than Three Back-to-Back SETUP Packets” on Page 16-79**.
BNAIntr	9	rwh	**BNA (Buffer Not Available) Interrupt** The core generates this interrupt when the descriptor accessed is not ready for the Core to process, such as Host busy or DMA done This bit is valid only when Scatter/Gather DMA mode is enabled.

Field	Bits	Type	Description
PktDrpSts	11	rwh	**Packet Dropped Status** This bit indicates to the application that an ISOC OUT packet has been dropped. This bit does not have an associated mask bit and does not generate an interrupt. This bit is valid in non Scatter/Gather DMA mode when periodic transfer interrupt feature is selected.
BbleErrIntrpt	12	rwh	**BbleErr (Babble Error) interrupt** The core generates this interrupt when babble is received for the endpoint.
NAKIntrpt	13	rwh	**NAK interrupt** The core generates this interrupt when a NAK is transmitted or received by the device.In case of isochronous IN endpoints the interrupt gets generated when a zero length packet is transmitted due to un-availability of data in the TXFIFO.
NYETIntrpt	14	rwh	**NYET interrupt** The core generates this interrupt when a NYET response is transmitted for a non isochronous OUT endpoint.
0	[31:15], 10, [8:7]	r	**Reserved** Read as 0; should be written with 0.

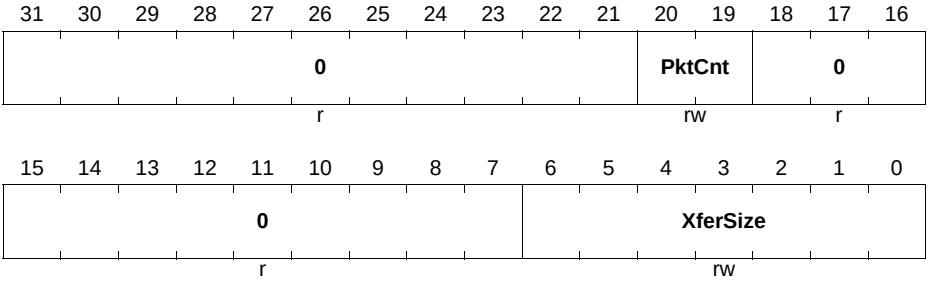
Device Endpoint 0 Transfer Size Register (DIEPTSIZ0/DOEPTSIZ0)

The application must modify this register before enabling endpoint 0. Once endpoint 0 is enabled using Endpoint Enable bit of the Device Control Endpoint 0 Control registers (DIEPCTL0.EPEna/DOEPCTL0.EPEna), the core modifies this register. The application can only read this register once the core has cleared the Endpoint Enable bit.

Non-zero endpoints use the registers for endpoints 1-6.

When Scatter/Gather DMA mode is enabled, this register must not be programmed by the application. If the application reads this register when Scatter/Gather DMA mode is enabled, the core returns all zeros.

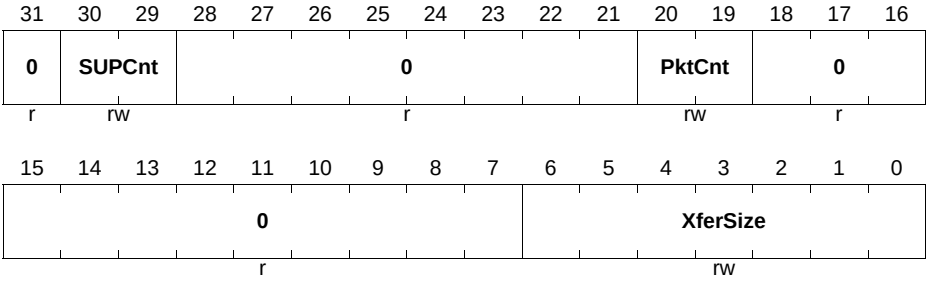
DIEPTSIZ0
Device IN Endpoint 0 Transfer Size Register(910_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
XferSize	[6:0]	rw	**Transfer Size** Indicates the transfer size in bytes for endpoint 0. The core interrupts the application only after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet. The core decrements this field every time a packet from the external memory is written to the TxFIFO.
PktCnt	[20:19]	rw	**Packet Count** Indicates the total number of USB packets that constitute the Transfer Size amount of data for endpoint 0. This field is decremented every time a packet (maximum size or short packet) is read from the TxFIFO.
0	[31:21], [18:7]	r	**Reserved** Read as 0; should be written with 0.

DOEPTSIZ0

Device OUT Endpoint 0 Transfer Size Register($B10_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
XferSize	[6:0]	rw	**Transfer Size** Indicates the transfer size in bytes for endpoint 0. The core interrupts the application only after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet. The core decrements this field every time a packet is read from the RxFIFO and written to the external memory.
PktCnt	[20:19]	rw	**Packet Count** This field is decremented to zero after a packet is written into the RxFIFO.
SUPCnt	[30:29]	rw	**SETUP Packet Count** This field specifies the number of back-to-back SETUP data packets the endpoint can receive. 01_B 1 packet 10_B 2 packets 11_B 3 packets
0	31, [28:21], [18:7]	r	**Reserved** Read as 0; should be written with 0.

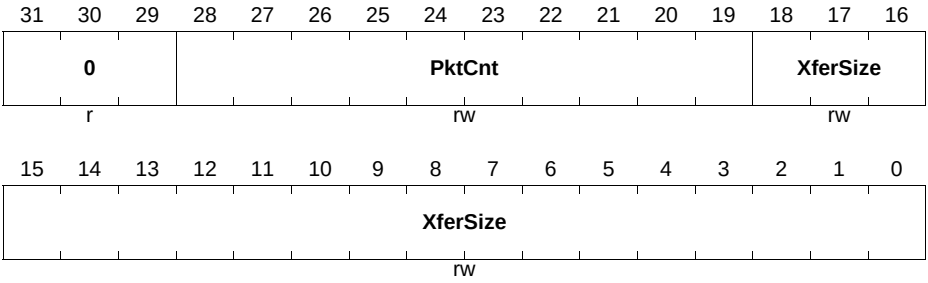
Device Endpoint-n Transfer Size Register (DIEPTSIZx/DOEPTSIZx)

The application must modify this register before enabling the endpoint. Once the endpoint is enabled using Endpoint Enable bit of the Device Endpoint-n Control registers (DIEPCTLx.EPEna/DOEPCTLx.EPEna), the core modifies this register. The application can only read this register once the core has cleared the Endpoint Enable bit.

This register is used only for endpoints other than Endpoint 0.

Note: When Scatter/Gather DMA mode is enabled, this register must not be programmed by the application. If the application reads this register when Scatter/Gather DMA mode is enabled, the core returns all zeros

DIEPTSIZx (x=1-6)
Device Endpoint-x Transfer Size Register(910_H + x*20_H) Reset Value: 0000 0000_H

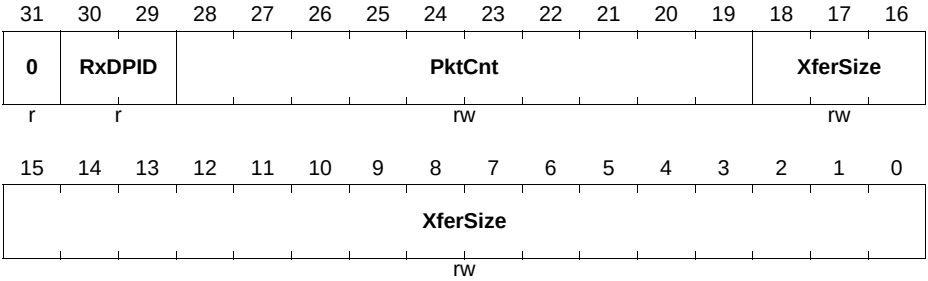


Field	Bits	Type	Description
XferSize	[18:0]	rw	**Transfer Size** This field contains the transfer size in bytes for the current endpoint. The core only interrupts the application after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet. • IN Endpoints: The core decrements this field every time a packet from the external memory is written to the TxFIFO.

Field	Bits	Type	Description
PktCnt	[28:19]	rw	**Packet Count** Indicates the total number of USB packets that constitute the Transfer Size amount of data for this endpoint. • IN Endpoints: This field is decremented every time a packet (maximum size or short packet) is read from the TxFIFO..
0	[31:29]	r	**Reserved** Read as 0; should be written with 0.

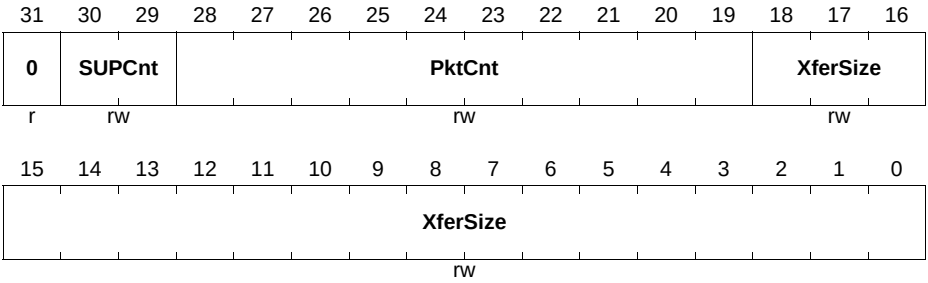
Note: The fields of the DOEPTSIZx register change, depending on isochronous or control OUT endpoint.

DOEPTSIZx (x=1-6)
Device Endpoint-x Transfer Size Register [ISO]
($B10_H$ + x*20_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
XferSize	[18:0]	rw	**Transfer Size** This field contains the transfer size in bytes for the current endpoint. The core only interrupts the application after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet. • OUT Endpoints: The core decrements this field every time a packet is read from the RxFIFO and written to the external memory.
PktCnt	[28:19]	rw	**Packet Count** Indicates the total number of USB packets that constitute the Transfer Size amount of data for this endpoint. • OUT Endpoints: This field is decremented every time a packet (maximum size or short packet) is written to the RxFIFO.
RxDPID	[30:29]	r	**Received Data PID** Applies to isochronous OUT endpoints only. This is the data PID received in the last packet for this endpoint. 00_B DATA0 01_B DATA2 10_B DATA1 11_B MDATA
0	31	r	**Reserved** Read as 0; should be written with 0.

DOEPTSIZx (x=1-6)
Device Endpoint-x Transfer Size Register [CONT]
($B10_H$ + x*20_H) **Reset Value: 0000 0000_H**

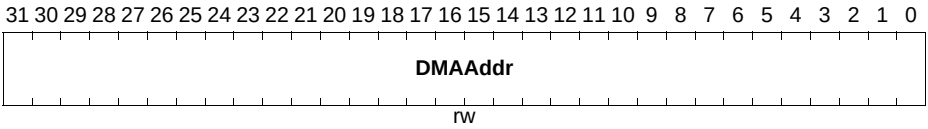


Field	Bits	Type	Description
XferSize	[18:0]	rw	**Transfer Size** This field contains the transfer size in bytes for the current endpoint. The core only interrupts the application after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet. • OUT Endpoints: The core decrements this field every time a packet is read from the RxFIFO and written to the external memory.
PktCnt	[28:19]	rw	**Packet Count** Indicates the total number of USB packets that constitute the Transfer Size amount of data for this endpoint. • OUT Endpoints: This field is decremented every time a packet (maximum size or short packet) is written to the RxFIFO.
SUPCnt	[30:29]	rw	**SETUP Packet Count** Applies to control OUT Endpoints only. This field specifies the number of back-to-back SETUP data packets the endpoint can receive. 00_B 1 packet 00_B 2 packets 00_B 3 packets
0	31	r	**Reserved** Read as 0; should be written with 0.

Device Endpoint-n DMA Address Register (DIEPDMAx/DOEPDMAx)

These registers are implemented in RAM.

DIEPDMAx (x=0-6)
Device Endpoint-x DMA Address Register(914_H + x*20_H) **Reset Value: XXXX XXXX$_H$**
DOEPDMAx (x=0-6)
Device Endpoint-x DMA Address Register($B14_H$ + x*20_H) **Reset Value: XXXX XXXX$_H$**

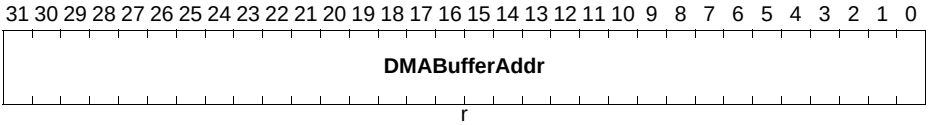


Field	Bits	Type	Description
DMAAddr	[31:0]	rw	**DMA Address** Holds the start address of the external memory for storing or fetching endpoint data. *Note: For control endpoints, this field stores control OUT data packets as well as SETUP transaction data packets. When more than three SETUP packets are received back-to-back, the SETUP data packet in the memory is overwritten.* This register is incremented on every AHB transaction. The application can give only a DWORD-aligned address. • When Scatter/Gather DMA mode is not enabled, the application programs the start address value in this field. • When Scatter/Gather DMA mode is enabled, this field indicates the base pointer for the descriptor list.

Device Endpoint-n DMA Buffer Address Register (DIEPDMABx/DOEPDMABx)

These fields are present only in case of Scatter/Gather DMA. These registers are implemented in RAM.

DIEPDMABx (x=0-6)
Device Endpoint-x DMA Buffer Address Register($91C_H$ + x*20_H) **Reset Value: XXXX XXXX$_H$**
DOEPDMABx (x=0-6)
Device Endpoint-x DMA Buffer Address Register($B1C_H$ + x*20_H) **Reset Value: XXXX XXXX$_H$**

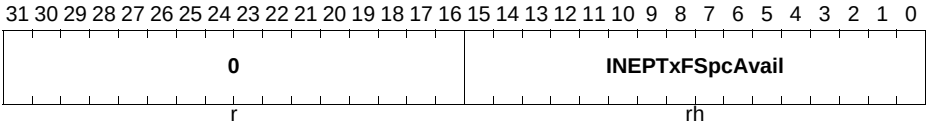


Field	Bits	Type	Description
DMABufferAddr	[31:0]	r	**DMA Buffer Address** Holds the current buffer address.This register is updated as and when the data transfer for the corresponding end point is in progress. This register is present only in Scatter/Gather DMA mode. Otherwise this field is reserved.

Device IN Endpoint Transmit FIFO Status Register (DTXFSTSx)

This read-only register contains the free space information for the Device IN endpoint TxFIFO.

DTXFSTSx (x=0-6)
Device IN Endpoint Transmit FIFO Status Register(918_H + x*20_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
INEPTxFSpcAvail	[15:0]	rh	**IN Endpoint TxFIFO Space Avail** Indicates the amount of free space available in the Endpoint TxFIFO. Values are in terms of 32-bit words. 0_H Endpoint TxFIFO is full 1_H 1 word available 2_H 2 words available Others: Up to n words can be selected (0 < n < 256); selections greater than n are reserved
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

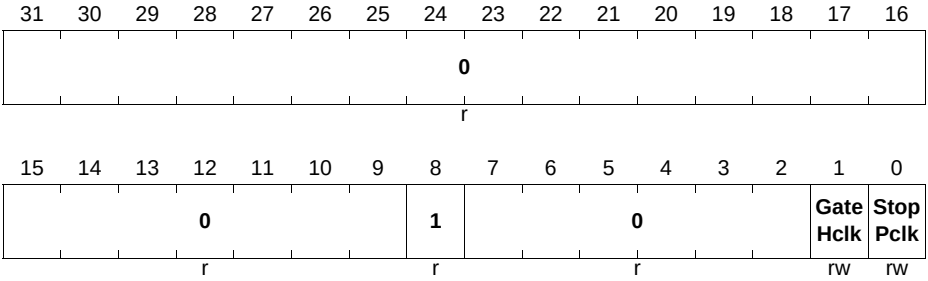
Power and Clock Gating Registers

There is a single register for power and clock gating. It is available in both Host and Device modes.

Power and Clock Gating Control Register (PCGCCTL)

This register is available in Host and Device modes. The application can use this register to control the core's clock gating features.

PCGCCTL
Power and Clock Gating Control Register(E00$_H$) **Reset Value: 0000 0100$_H$**



Field	Bits	Type	Description
StopPclk	0	rw	**Stop Pclk** The application sets this bit to stop the PHY clock (phy_clk) when the USB is suspended, the session is not valid, or the device is disconnected. The application clears this bit when the USB is resumed or a new session starts.
GateHclk	1	rw	**Gate Hclk** The application sets this bit to gate hclk to modules other than the AHB Slave and Master and wakeup logic when the USB is suspended or the session is not valid. The application clears this bit when the USB is resumed or a new session starts.
1	8	r	**Reserved** Read as 1; should be written with 1.
0	[31:9], [7:2]	r	**Reserved** Read as 0; should be written with 0.

16.16 Interconnects

The interconnects section describes the connectivity of the module.

Table 16-22 Pin Connections

Input/Output	I/O	Connected To	Description
USB0.ID	I	P0.9	ID pad signal
USB0.D+	I/O	USB_DP	Data + signal
USB0.D-	I/O	USB_DM	Data - signal
USB0.VBUS	I/O	VBUS	VBUS signal
USB0.DRIVEVBUS	O	P0.1 P3.2	Drive VBUS signal

17 Universal Serial Interface Channel (USIC)

The **U**niversal **S**erial **I**nterface **C**hannel module (USIC) is a flexible interface module covering several serial communication protocols. A USIC module contains two independent communication channels named USICx_CH0 and USICx_CH1, with x being the number of the USIC module (e.g. channel 0 of USIC module 0 is referenced as USIC0_CH0). The user can program during run-time which protocol will be handled by each communication channel and which pins are used.

References

The following documents are referenced for further information

[17] IIC Bus Specification (Philips Semiconductors v2.1)

[18] IIS Bus Specification (Philips Semiconductors June 5 1996 revision)

17.1 Overview

This section gives an overview about the feature set of the USIC.

17.1.1 Features

Each USIC channel can be individually configured to match the application needs, e.g. the protocol can be selected or changed during run time without the need for a reset. The following protocols are supported:

- **UART** (ASC, asynchronous serial channel)
  - Module capability: receiver/transmitter with max. baud rate f_{PB} / 4
  - Wide baud rate range down to single-digit baud rates
  - Number of data bits per data frame: 1 to 63
  - MSB or LSB first
- **LIN** Support by hardware (Local Interconnect Network)
  - Data transfers based on ASC protocol
  - Baud rate detection possible by built-in capture event of baud rate generator
  - Checksum generation under software control for higher flexibility
- **SSC/SPI** (synchronous serial channel with or without slave select lines)
  - Standard, Dual and Quad SPI format supported
  - Module capability: maximum baud rate f_{PB} / 2, limited by loop delay
  - Number of data bits per data frame 1 to 63, more with explicit stop condition
  - Parity bit generation supported
  - MSB or LSB first
- **IIC** (Inter-IC Bus)
  - Application baud rate 100 kbit/s to 400 kbit/s
  - 7-bit and 10-bit addressing supported
  - Full master and slave device capability

- **IIS** (infotainment audio bus)
  – Module capability: maximum baud rate f_{PB} / 2

Note: The real baud rates that can be achieved in a real application depend on the operating frequency of the device, timing parameters as described in the Data Sheet, signal delays on the PCB and timings of the peer device.

In addition to the flexible choice of the communication protocol, the USIC structure has been designed to reduce the system load (CPU load) allowing efficient data handling. The following aspects have been considered:

- **Data buffer capability**
  The standard buffer capability includes a double word buffer for receive data and a single word buffer for transmit data. This allows longer CPU reaction times (e.g. interrupt latency).
- **Additional FIFO buffer capability**
  In addition to the standard buffer capability, the received data and the data to be transmitted can be buffered in a FIFO buffer structure. The size of the receive and the transmit FIFO buffer can be programmed independently. Depending on the application needs, a total buffer capability of 64 data words can be assigned to the receive and transmit FIFO buffers of a USIC module (the two channels of the USIC module share the 64 data word buffer).
  In addition to the FIFO buffer, a bypass mechanism allows the introduction of high-priority data without flushing the FIFO buffer.
- **Transmit control information**
  For each data word to be transmitted, a 5-bit transmit control information has been added to automatically control some transmission parameters, such as word length, frame length, or the slave select control for the SPI protocol. The transmit control information is generated automatically by analyzing the address where the user software has written the data word to be transmitted (32 input locations = 2^5 = 5 bit transmit control information).
  This feature allows individual handling of each data word, e.g. the transmit control information associated to the data words stored in a transmit FIFO can automatically modify the slave select outputs to select different communication targets (slave devices) without CPU load. Alternatively, it can be used to control the frame length.
- **Flexible frame length control**
  The number of bits to be transferred within a data frame is independent of the data word length and can be handled in two different ways. The first option allows automatic generation of frames up to 63 bits with a known length. The second option supports longer frames (even unlimited length) or frames with a dynamically controlled length.
- **Interrupt capability**
  The events of each USIC channel can be individually routed to one of 6 service request outputs SR[5:0] available for each USIC module, depending on the

application needs. Furthermore, specific start and end of frame indications are supported in addition to protocol-specific events.

- **Flexible interface routing**
  Each USIC channel offers the choice between several possible input and output pins connections for the communications signals. This allows a flexible assignment of USIC signals to pins that can be changed without resetting the device.
- **Input conditioning**
  Each input signal is handled by a programmable input conditioning stage with programmable filtering and synchronization capability.
- **Baud rate generation**
  Each USIC channel contains its own baud rate generator. The baud rate generation can be based either on the internal module clock or on an external frequency input. This structure allows data transfers with a frequency that can not be generated internally, e.g. to synchronize several communication partners.
- **Transfer trigger capability**
  In master mode, data transfers can be triggered by events generated outside the USIC module, e.g. by an input pin or a timer unit (transmit data validation). This feature allows time base related data transmission.
- **Debugger support**
  The USIC offers specific addresses to read out received data without interaction with the FIFO buffer mechanism. This feature allows debugger accesses without the risk of a corrupted receive data sequence.

To reach a desired baud rate, two criteria have to be respected, the module capability and the application environment. The module capability is defined with respect to the module's input clock frequency, being the base for the module operation. Although the module's capability being much higher (depending on the module clock and the number of module clock cycles needed to represent a data bit), the reachable baud rate is generally limited by the application environment. In most cases, the application environment limits the maximum reachable baud rate due to driver delays, signal propagation times, or due to EMI reasons.

Note: Depending on the selected additional functions (such as digital filters, input synchronization stages, sample point adjustment, data structure, etc.), the maximum reachable baud rate can be limited. Please also take care about additional delays, such as (internal or external) propagation delays and driver delays (e.g. for collision detection in ASC mode, for IIC, etc.).

A block diagram of the USIC module/channel structure is shown in **Figure 17-1**.

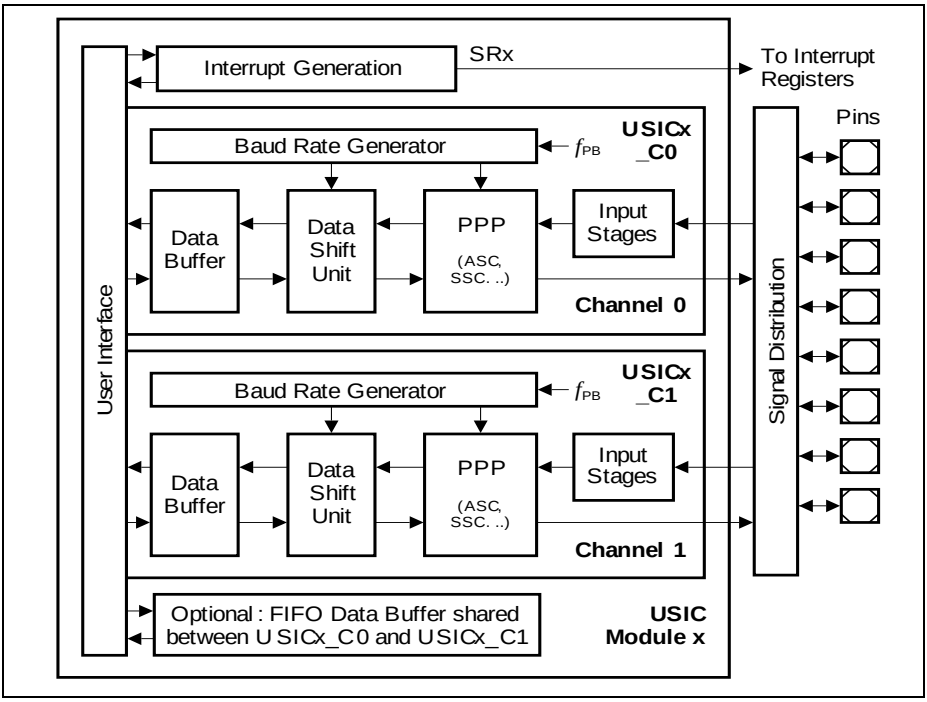


Figure 17-1 USIC Module/Channel Structure

17.2 Operating the USIC

This section describes how to operate the USIC communication channel.

17.2.1 USIC Structure Overview

This section introduces the USIC structure.

17.2.1.1 Channel Structure

The USIC module contains two independent communication channels, with a structure as shown in **Figure 17-1**.

The data shift unit and the data buffering of each channel support full-duplex data transfers. The protocol-specific actions are handled by the protocol pre-processors (PPP). In order to simplify data handling, an additional FIFO data buffer is optionally available for each USIC module to store transmit and receive data for each channel.

Due to the independent channel control and baud rate generation, the communication protocol, baud rate and the data format can be independently programmed for each communication channel.

17.2.1.2 Input Stages

For each protocol, the number of input signals used depends on the selected protocol. Each input signal is handled by an input stage (called DXn, where n=0-5) for signal conditioning, such as input selection, polarity control, or a digital input filter. They can be classified according to their meaning for the protocols, see **Table 17-1**.

The inputs marked as “optional” are not needed for the standard function of a protocol and may be used for enhancements. The descriptions of protocol-specific items are given in the related protocol chapters. For the external frequency input, please refer to the baud rate generator section, and for the transmit data validation, to the data handling section.

Table 17-1 Input Signals for Different Protocols

Selected Protocol	Shift Data Input(s) (handled by DX0, DX3, DX4 and DX5)[1)]	Shift Clock Input (handled by DX1)	Shift Control Input (handled by DX2)
ASC, LIN	RXD	optional: external frequency input or TXD collision detection	optional: transmit data validation
Standard SSC, SPI (Master)	DIN0 (MRST, MISO)	optional: external frequency input or delay compensation	optional: transmit data validation or delay compensation
Standard SSC, SPI (Slave)	DIN0 (MTSR, MOSI)	SCLKIN	SELIN
Dual-SSC, SPI (Master)	DIN[1:0] (MRST[1:0], MISO[1:0])	optional: external frequency input or delay compensation	optional: transmit data validation or delay compensation
Dual-SSC, SPI (Slave)	DIN[1:0] (MTSR[1:0], MOSI[1:0])	SCLKIN	SELIN
Quad-SSC, SPI (Master)	DIN[3:0] (MRST[3:0], MISO[3:0])	optional: external frequency input or delay compensation	optional: transmit data validation or delay compensation
Quad-SSC, SPI (Slave)	DIN[3:0] (MTSR[3:0], MOSI[3:0])	SCLKIN	SELIN
IIC	SDA	SCL	optional: transmit data validation
IIS (Master)	DIN0	optional: external frequency input or delay compensation	optional: transmit data validation or delay compensation
IIS (Slave)	DIN0	SCLKIN	WAIN

1) ASC, IIC, IIS and standard SSC protocols use only DX0 as the shift data input.

Note: To allow a certain flexibility in assigning required USIC input functions to port pins of the device, each input stage can select the desired input location among several possibilities.
The available USIC signals and their port locations are listed in the interconnects section, see ***Page 17-225****.*

17.2.1.3 Output Signals

For each protocol, up to 14 protocol-related output signals are available. The number of actually used outputs depends on the selected protocol. They can be classified according to their meaning for the protocols, see **Table 17-2**.

The outputs marked as “optional” are not needed for the standard function of a protocol and may be used for enhancements. The descriptions of protocol-specific items are given in the related protocol chapters. The MCLKOUT output signal has a stable frequency relation to the shift clock output (the frequency of MCLKOUT can be higher than for SCLKOUT) for synchronization purposes of a slave device to a master device. If the baud rate generator is not needed for a specific protocol (e.g. in SSC slave mode), the SCLKOUT and MCLKOUT signals can be used as clock outputs with 50% duty cycle with a frequency that can be independent from the communication baud rate.

Table 17-2 Output Signals for Different Protocols

Selected Protocol	Shift Data Output(s) DOUT[3:0]	Shift Clock Output SCLKOUT	Shift Control Outputs SELO[7:0]	Master Clock Output MCLKOUT
ASC, LIN	TXD	not used	not used	optional: master time base
Standard SSC, SPI (Master)	DOUT0 (MTSR, MOSI)	master shift clock	slave select, chip select	optional: master time base
Standard SSC, SPI (Slave)	DOUT0 (MRST, MISO)	optional: independent clock output	not used	optional: independent clock output
Dual-SSC, SPI (Master)	DOUT[1:0] (MTSR[1:0], MOSI[1:0])	master shift clock	slave select, chip select	optional: master time base
Dual-SSC, SPI (Slave)	DOUT[1:0] (MRST[1:0], MISO[1:0])	optional: independent clock output	not used	optional: independent clock output

Table 17-2 Output Signals for Different Protocols (cont'd)

Selected Protocol	Shift Data Output(s) DOUT[3:0]	Shift Clock Output SCLKOUT	Shift Control Outputs SELO[7:0]	Master Clock Output MCLKOUT
Quad-SSC, SPI (Master)	DOUT[3:0] (MTSR[3:0], MOSI[3:0])	master shift clock	slave select, chip select	optional: master time base
Quad-SSC, SPI (Slave)	DOUT[3:0] (MRST[3:0], MISO[3:0])	optional: independent clock output	not used	optional: independent clock output
IIC	SDA	SCL	not used	optional: master time base
IIS (master)	DOUT0	master shift clock	WA	optional: master time base
IIS (slave)	DOUT0	optional: independent clock output	not used	optional: independent clock output

Note: To allow a certain flexibility in assigning required USIC output functions to port pins of the device, most output signals are made available on several port pins. The port control itself defines pin-by-pin which signal is used as output signal for a port pin (see port chapter).
The available USIC signals and their port locations are listed in the interconnects section, see ***Page 17-225****.*

17.2.1.4 Baud Rate Generator

Each USIC Channel contains a baud rate generator structured as shown in **Figure 17-2**. It is based on coupled divider stages, providing the frequencies needed for the different protocols. It contains:

- A fractional divider to generate the input frequency $f_{PIN} = f_{FD}$ for baud rate generation based on the internal system frequency f_{PB}.
- The DX1 input to generate the input frequency $f_{PIN} = f_{DX1}$ for baud rate generation based on an external signal.
- Two protocol-related counters: the divider mode counter to provide the master clock signal MCLK, the shift clock signal SCLK, and other protocol-related signals; and the capture mode timer for time interval measurement, e.g. baud rate detection.
- A time quanta counter associated to the protocol pre-processor defining protocol-specific timings, such shift control signals or bit timings, based on the input frequency f_{CTQIN}.

- The output signals MCLKOUT and SCLKOUT of the protocol-related divider that can be made available on pins. In order to adapt to different applications, some output characteristics of these signals can be configured.
  For device-specific details about availability of USIC signals on pins please refer to the interconnects section.

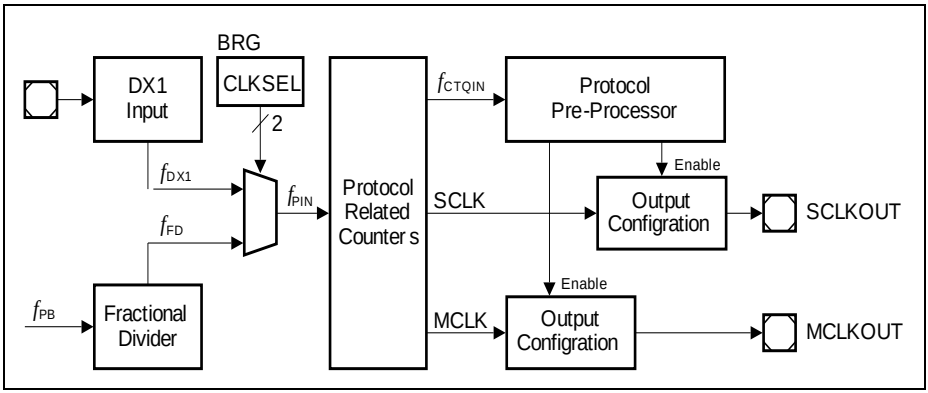


Figure 17-2 Baud Rate Generator

17.2.1.5 Channel Events and Interrupts

The notification of the user about events occurring during data traffic and data handling is based on:

- Data transfer events related to the transmission or reception of a data word, independent of the selected protocol.
- Protocol-specific events depending on the selected protocol.
- Data buffer events related to data handling by the optional FIFO data buffers.

17.2.1.6 Data Shifting and Handling

The data handling of the USIC module is based on an independent data shift unit (DSU) and a buffer structure that is similar for the supported protocols. The data shift and buffer registers are 16-bit wide (maximum data word length), but several data words can be concatenated to achieve longer data frames. The DSU inputs are the shift data (handled by input stage DX0, DX3, DX4 and DX5), the shift clock (handled by the input stage DX1), and the shift control (handled by the input stage DX2). The signal DOUT[3:0] represents the shift data outputs.

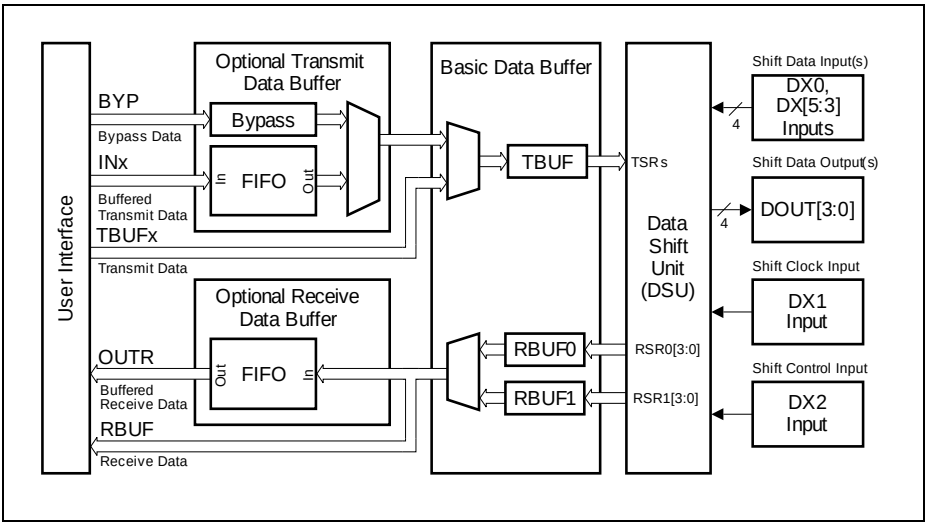


Figure 17-3 Principle of Data Buffering

The principle of data handling comprises:

- A transmitter with transmit shift registers (TSR and TSR[3:0]) in the DSU and a transmit data buffer (TBUF). A data validation scheme allows triggering and gating of data transfers by external events under certain conditions.
- A receiver with two alternating sets of receive shift registers (RSR0[3:0] and RSR1[3:0]) in the DSU and a double receive buffer structure (RBUF0, RBUF1). The alternating receive shift registers support the reception of data streams and data frames longer than one data word.
- Optional transmit and receive data buffers according to the first-in-first-out principle (FIFO), that are not necessarily available in all devices. For device-specific details about availability of the FIFO buffer please refer to the USIC implementation chapter.
- A user interface to handle data, interrupts, and status and control information.

Basic Data Buffer Structure

The read access to received data and the write access of data to be transmitted can be handled by a basic data buffer structure.

The received data stored in the receiver buffers RBUF0/RBUF1 can be read directly from these registers. In this case, the user has to take care about the reception sequence to read these registers in the correct order. To simplify the use of the receive buffer structure, register RBUF has been introduced. A read action from this register delivers the data word received first (oldest data) to respect the reception sequence. With a read access from at least the low byte of RBUF, the data is automatically declared to be no

longer new and the next received data word becomes visible in RBUF and can be read out next.

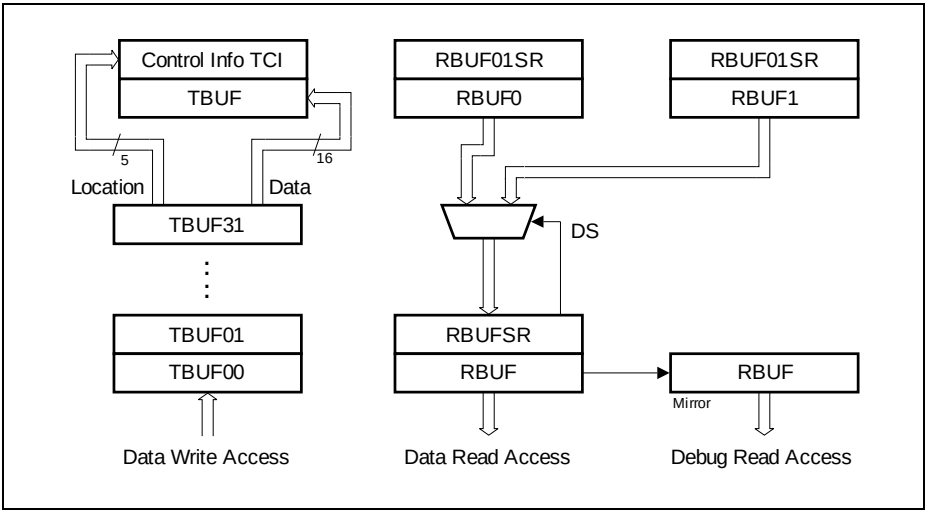


Figure 17-4 Data Access Structure without additional Data Buffer

It is recommended to read the received data words by accesses to RBUF and to avoid handling of RBUF0 and RBUF1. The USIC module also supports the use of debug accesses to receive data words. Debugger read accesses should not disturb the receive data sequence and, as a consequence, should not target RBUF. Therefore, register RBUFD has been introduced. It contains the same value as RBUF, but a read access from RBUFD does not change the status of the data (same data can be read several times). In addition to the received data, some additional status information about each received data word is available in the receiver buffer status register RBUF01SR (related to data in RBUF0 and RBUF1) and RBUFSR (related to data in RBUF).

Transmit data can be loaded to TBUF by software by writing to the transmit buffer input locations TBUFx (x = 00-31), consisting of 32 consecutive addresses. The data written to one of these input locations is stored in the transmit buffer TBUF. Additionally, the address of the written location is evaluated and can be used for additional control purposes. This 5-bit wide information (named **T**ransmit **C**ontrol **I**nformation TCI) can be used for different purposes in different protocols.

FIFO Buffer Structure

To allow easier data setup and handling, an additional data buffering mechanism can be optionally supported. The data buffer is based on the first-in-first-out principle (FIFO) that ensures that the sequence of transferred data words is respected.

If a FIFO buffer structure is used, the data handling scheme (data with associated control information) is similar to the one without FIFO. The additional FIFO buffer can be independently enabled/disabled for transmission and reception (e.g. if data FIFO buffers are available for a specific USIC channel, it is possible to configure the transmit data path without and the receive data path with FIFO buffering).

The transmit FIFO buffer is addressed by using 32 consecutive address locations for INx instead of TBUFx (x=00-31) regardless of the FIFO depth. The 32 addresses are used to store the 5-bit TCI (together with the written data) associated with each FIFO entry.

The receive FIFO can be read out at two independent addresses, OUTR and OUTDR instead of RBUF and RBUFD. A read from the OUTR location triggers the next data packet to be available for the next read (general FIFO mechanism). In order to allow non-intrusive debugging (without risk of data loss), a second address location (OUTDR) has been introduced. A read at this location delivers the same value as OUTR, but without modifying the FIFO contents.

The transmit FIFO also has the capability to bypass the data stream and to load bypass data to TBUF. This can be used to generate high-priority messages or to send an emergency message if the transmit FIFO runs empty. The transmission control of the FIFO buffer can also use the transfer trigger and transfer gating scheme of the transmission logic for data validation (e.g. to trigger data transfers by events).

Note: The available size of a FIFO data buffer for a USIC channel depends on the specific device. Please refer to the implementation chapter for details about available FIFO buffer capability.

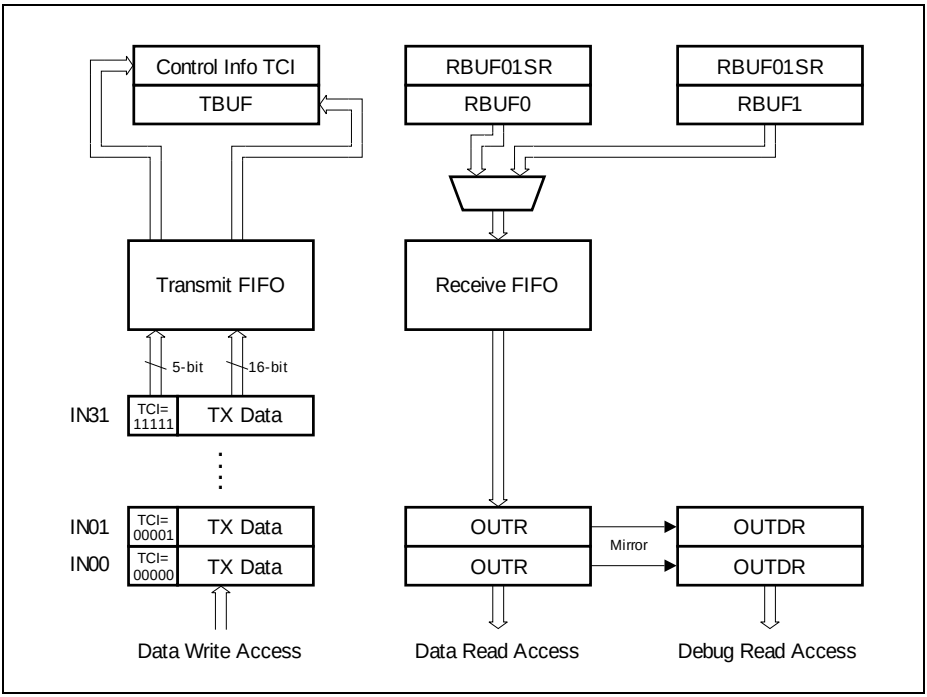


Figure 17-5 Data Access Structure with FIFO

17.2.2 Operating the USIC Communication Channel

This section describes how to operate a USIC communication channel, including protocol control and status, mode control and interrupt handling. The following aspects have to be taken into account:

- Enable the USIC module for operation and configure the behavior for the different device operation modes (see **Page 17-15**).
- Configure the pinning (refer to description in the corresponding protocol section).
- Configure the data structure (shift direction, word length, frame length, polarity, etc.).
- Configure the data buffer structure of the optional FIFO buffer area. A FIFO buffer can only be enabled if the related bit in register CCFG is set.
- Select a protocol by CCR.MODE. A protocol can only be selected if the related bit in register CCFG is set.

17.2.2.1 Protocol Control and Status

The protocol-related control and status information are located in the protocol control register PCR and in the protocol status register PSR. These registers are shared between the available protocols. As a consequence, the meaning of the bit positions in these registers is different within the protocols.

Use of PCR Bits

The signification of the bits in register PCR is indicated by the protocol-related alias names for the different protocols.

- PCR for the ASC protocol (see **Page 17-64**)
- PCR for the SSC protocol (see **Page 17-96**)
- PCR for the IIC protocol (see **Page 17-127**)
- PCR for the IIS protocol (see **Page 17-145**)

Use of PSR Flags

The signification of the flags in register PSR is indicated by the protocol-related alias names for the different protocols.

- PSR flags for the ASC protocol (see **Page 17-68**)
- PSR flags for the SSC protocol (see **Page 17-100**)
- PSR flags for the IIC protocol (see **Page 17-130**)
- PSR flags for the IIS protocol (see **Page 17-147**)

17.2.2.2 Mode Control

The mode control concept for system control tasks, such as suspend request for debugging, allows to program the module behavior under different device operating conditions. The behavior of a communication channel can be programmed for each of the device operating modes (normal operation, suspend mode). Therefore, each communication channel has an associated kernel state configuration register KSCFG defining its behavior in the following operating modes:

- **Normal operation**:
  This operating mode is the default operating mode when no suspend request is pending. The module clock is not switched off and the USIC registers can be read or written. The channel behavior is defined by KSCFG.NOMCFG.
- **Suspend mode**:
  This operating mode is requested when a suspend request is pending in the device. The module clock is not switched off and the USIC registers can be read or written. The channel behavior is defined by KSCFG.SUMCFG.

The four kernel modes defined by the register KSCFG are shown in **Table 17-3**.

Table 17-3 USIC Communication Channel Behavior

Kernel Mode	Channel Behavior	KSCFG. NOMCFG
Run mode 0	Channel operation as specified, no impact on data transfer	00_B
Run mode 1		01_B
Stop mode 0	Explicit stop condition as described in the protocol chapters	10_B
Stop mode 1		11_B

Generally, bit field KSCFG.NOMCFG should be configured for run mode 0 as default setting for standard operation. If a communication channel should not react to a suspend request (and to continue its operation as in normal mode), bit field KSCFG.SUMCFG has to be configured with the same value as KSCFG.NOMCFG. If the communication channel should show a different behavior and stop operation when a specific stop condition is reached, the code for stop mode 0 or stop mode 1 have to be written to KSCFG.SUMCFG.

The stop conditions are defined for the selected protocol (see mode control description in the protocol section).

Note: The stop mode selection strongly depends on the application needs and it is very unlikely that different stop modes are required in parallel in the same application. As a result, only one stop mode type (either 0 or 1) should be used in the bit fields in register KSCFG. Do not mix stop mode 0 and stop mode 1 and avoid transitions

from stop mode 0 to stop mode 1 (or vice versa) for the same communication channel.

17.2.2.3 General Channel Events and Interrupts

The general event and interrupt structure is shown in **Figure 17-6**. If a defined condition is met, an event is detected and an event indication flag becomes automatically set. The flag stays set until it is cleared by software. If enabled, an interrupt can be generated if an event is detected. The actual status of the event indication flag has no influence on the interrupt generation. As a consequence, the event indication flag does not need to be cleared to generate further interrupts.

Additionally, the service request output SRx of the USIC channel that becomes activated in case of an event condition can be selected by an interrupt node pointer. This structure allows to assign events to interrupts, e.g. depending on the application, several events can share the same interrupt routine (several events activate the same SRx output) or can be handled individually (only one event activates one SRx output).

The SRx outputs are connected to interrupt control registers to handle the CPU reaction to the service requests. This assignment is described in the implementation section on **Page 17-151**.

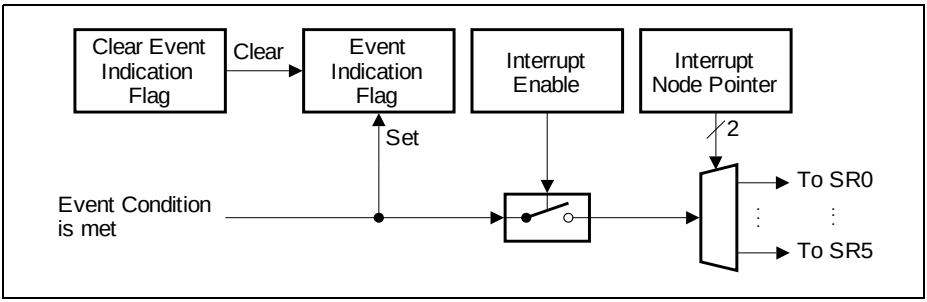


Figure 17-6 General Event and Interrupt Structure

17.2.2.4 Data Transfer Events and Interrupts

The data transfer events are based on the transmission or reception of a data word. The related indication flags are located in register PSR. All events can be individually enabled for interrupt generation.

- Receive event to indicate that a data word has been received:
  If a new received word becomes available in the receive buffer RBUF, either a receive event or an alternative receive event occurs.
  The receive event occurs if bit RBUFSR.PERR = 0. It is indicated by flag PSR.RIF and, if enabled, leads to receive interrupt.
- Receiver start event to indicate that a data word reception has started:
  When the receive clock edge that shifts in the first bit of a new data word is detected and reception is enabled, a receiver start event occurs. It is indicated by flag PSR.RSIF and, if enabled, leads to transmit buffer interrupt.
  In full duplex mode, this event follows half a shift clock cycle after the transmit buffer event and indicates when the shift control settings are internally “frozen” for the current data word reception and a new setting can be programmed.
  In SSC and IIS mode, the transmit data valid flag TCSR.TDV is cleared in single shot mode with the receiver start event.
- Alternative receive event to indicate that a specific data word has been received:
  If a new received word becomes available in the receive buffer RBUF, either a receive event or an alternative receive event occurs.
  The alternative receive event occurs if bit RBUFSR.PERR = 1. It is indicated by flag PSR.AIF and, if enabled, leads to alternative receive interrupt.
  Depending on the selected protocol, bit RBUFSR.PERR is set to indicate a parity error in ASC mode, the reception of the first byte of a new frame in IIC mode, and the WA information about right/left channel in IIS mode. In SSC mode, it is used as indication if the received word is the first data word, and is set if first and reset if not.
- Transmit shift event to indicate that a data word has been transmitted:
  A transmit shift event occurs with the last shift clock edge of a data word. It is indicated by flag PSR.TSIF and, if enabled, leads to transmit shift interrupt.
- Transmit buffer event to indicate that a data word transmission has been started:
  When a data word from the transmit buffer TBUF has been loaded to the shift register and a new data word can be written to TBUF, a transmit buffer event occurs. This happens with the transmit clock edge that shifts out the first bit of a new data word and transmission is enabled. It is indicated by flag PSR.TBIF and, if enabled, leads to transmit buffer interrupt.
  This event also indicates when the shift control settings (word length, shift direction, etc.) are internally “frozen” for the current data word transmission.
  In ASC and IIC mode, the transmit data valid flag TCSR.TDV is cleared in single shot mode with the transmit buffer event.
- Data lost event to indicate a loss of the oldest received data word:
  If the data word available in register RBUF (oldest data word from RBUF0 or RBUF1)

has not been read out before it becomes overwritten with new incoming data, this event occurs. It is indicated by flag PSR.DLIF and, if enabled, leads to a protocol interrupt.

Table 17-4 shows the registers, bits and bit fields indicating the data transfer events and controlling the interrupts of a USIC channel.

Table 17-4 Data Transfer Events and Interrupt Handling

Event	Indication Flag	Indication cleared by	Interrupt enabled by	SRx Output selected by
Standard receive event	PSR.RIF	PSCR.CRIF	CCR.RIEN	INPR.RINP
Receive start event	PSR.RSIF	PSCR.CRSIF	CCR.RSIEN	INPR.TBINP
Alternative receive event	PSR.AIF	PSCR.CAIF	CCR.AIEN	INPR.AINP
Transmit shift event	PSR.TSIF	PSCR.CTSIF	CCR.TSIEN	INPR.TSINP
Transmit buffer event	PSR.TBIF	PSCR.CTBIF	CCR.TBIEN	INPR.TBINP
Data lost event	PSR.DLIF	PSCR.CDLIF	CCR.DLIEN	INPR.PINP

Figure 17-7 shows the two transmit events and interrupts.

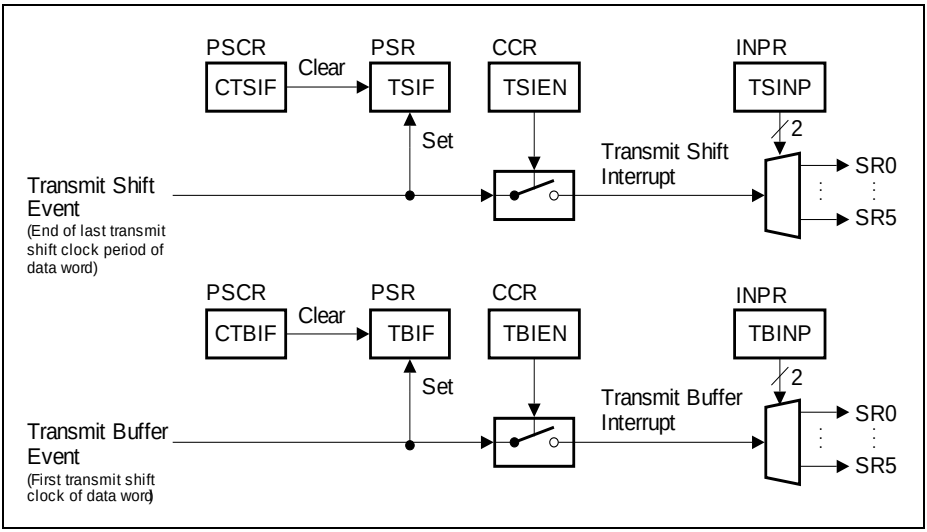


Figure 17-7 Transmit Events and Interrupts

Figure 17-8 shows the receive events and interrupts.

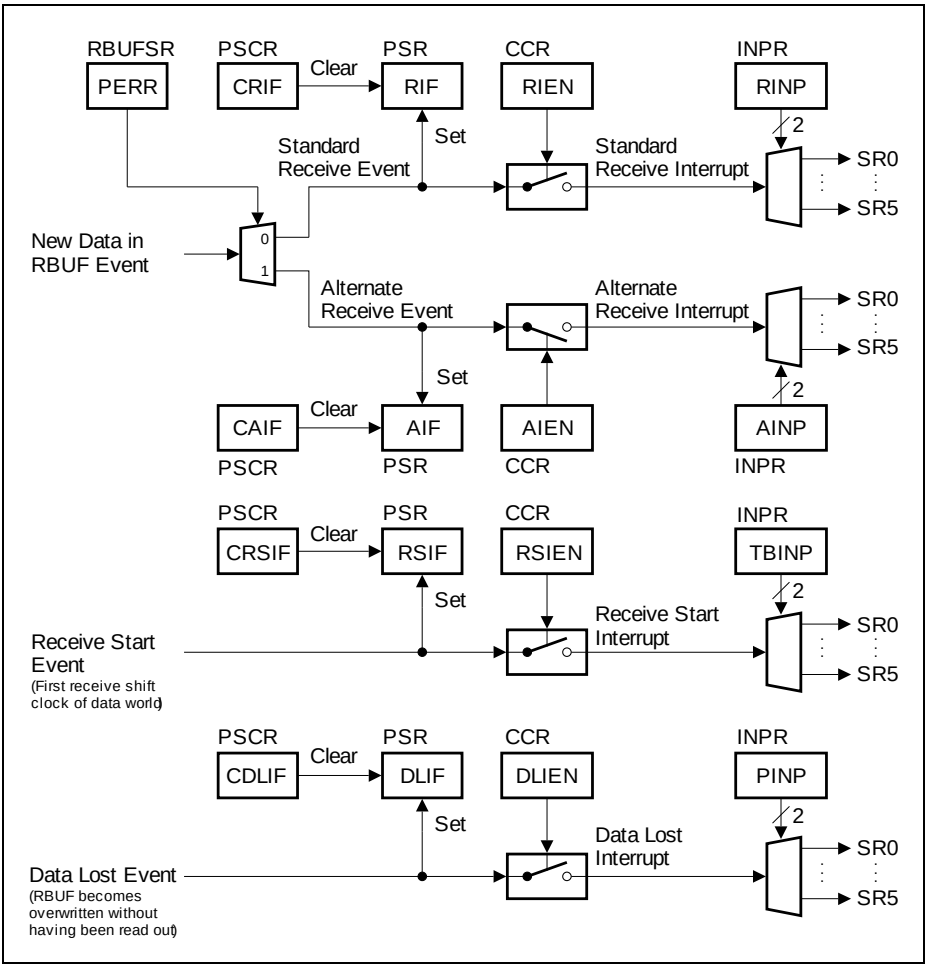


Figure 17-8 Receive Events and Interrupts

17.2.2.5 Baud Rate Generator Event and Interrupt

The baud rate generator event is based on the capture mode timer reaching its maximum value. It is indicated by flag PSR.BRGIF and, if enabled, leads to a protocol interrupt.

Table 17-5 shows the registers, bits and bit fields indicating the baud rate generator event and controlling the interrupt of a USIC channel.

Table 17-5 Baud Rate Generator Event and Interrupt Handling

Event	Indication Flag	Indication cleared by	Interrupt enabled by	SRx Output selected by
Baud rate generator event	PSR. BRGIF	PSCR. CBRGIF	CCR. BRGIEN	INPR.PINP

Figure 17-9 shows the baud rate generator event and interrupt.

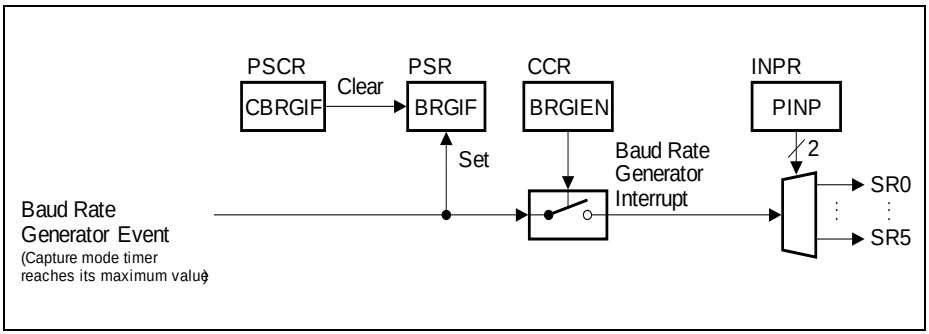


Figure 17-9 Baud Rate Generator Event and Interrupt

17.2.2.6 Protocol-specific Events and Interrupts

These events are related to protocol-specific actions that are described in the corresponding protocol chapters. The related indication flags are located in register PSR. All events can be individually enabled for the generation of the common protocol interrupt.

- Protocol-specific events in ASC mode:
  Synchronization break, data collision on the transmit line, receiver noise, format error in stop bits, receiver frame finished, transmitter frame finished
- Protocol-specific events in SSC mode:
  MSLS event (start-end of frame in master mode), DX2T event (start/end of frame in slave mode), both based on slave select signals, parity error
- Protocol-specific events in IIC mode:
  Wrong transmit code (error in frame sequence), start condition received, repeated start condition received, stop condition received, non-acknowledge received, arbitration lost, slave read request, other general errors
- Protocol-specific events in IIS mode:
  DX2T event (change on WA line), WA falling edge or rising edge detected, WA generation finished

Table 17-6 Protocol-specific Events and Interrupt Handling

Event	Indication Flag	Indication cleared by	Interrupt enabled by	SRx Output selected by
Protocol-specific events in ASC mode	PSR.ST[8:2]	PSCR.CST[8:2]	PCR.CTR[7:3]]	INPR.PINP
Protocol-specific events in SSC mode	PSR.ST[3:2]	PSCR.CST[3:2]	PCR.CTR[15:14]	INPR.PINP
Protocol-specific events in IIC mode	PSR.ST[8:1]	PSCR.CST[8:1]	PCR.CTR[24:18]	INPR.PINP
Protocol-specific events in IIS mode	PSR.ST[6:3]	PSCR.CST[6:3]	PCR.CTR[6:4], PCR.CTR[15]	INPR.PINP

17.2.3 Operating the Input Stages

All input stages offer the same feature set. They are used for all protocols, because the signal conditioning can be adapted in a very flexible way and the digital filters can be switched on and off separately.

17.2.3.1 General Input Structure

There are generally two types of input stages, one for the data input stages DX0, DX[5:3] and the other for non-data input stages DX[2:1], as shown in **Figure 17-10** and **Figure 17-11**. The difference is that for the data input stages, the input signal can be additionally selected from the port signal HWINn if hardware port control is enabled through CCR.HPCEN bit. All other enable/disable functions and selections are controlled independently for each input stage by bits in the registers DXnCR.

The desired input signal can be selected among the input lines DXnA to DXnG and a permanent 1-level by programming bit field DSEL (for the data input stages, hardware port control must be disabled for DSEL to take effect). Please refer to the interconnects section (**Section 17.12**) for the device-specific input signal assignment. Bit DPOL allows a polarity inversion of the selected input signal to adapt the input signal polarity to the internal polarity of the data shift unit and the protocol state machine. For some protocols, the input signals can be directly forwarded to the data shift unit for the data transfers (DSEN = 0, INSW = 1) without any further signal conditioning. In this case, the data path does not contain any delay due to synchronization or filtering.

In the case of noise on the input signals, there is the possibility to synchronize the input signal (signal DXnS is synchronized to f_{PB}) and additionally to enable a digital noise filter in the signal path. The synchronized input signal (and optionally filtered if DFEN = 1) is taken into account by DSEN = 1. Please note that the synchronization leads to a delay in the signal path of 2-3 times the period of f_{PB}.

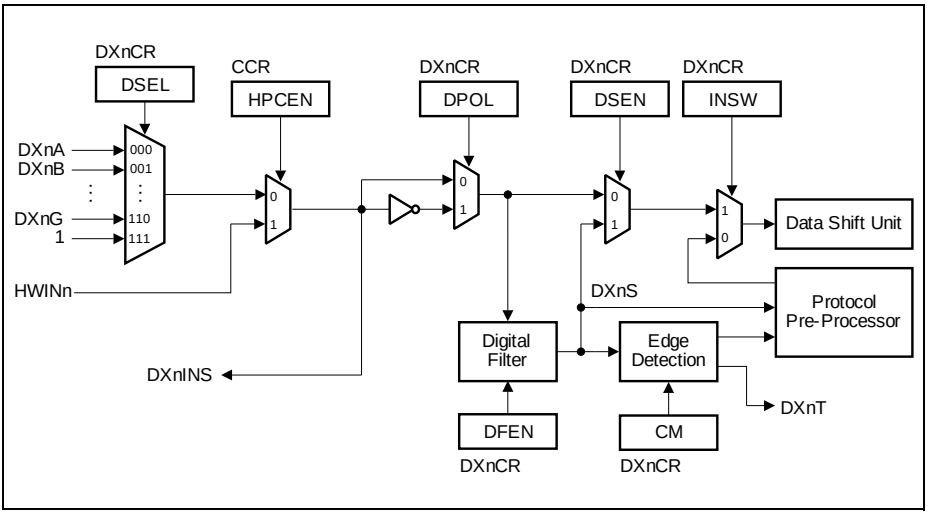


Figure 17-10 Input Conditioning for DX0 and DX[5:3]

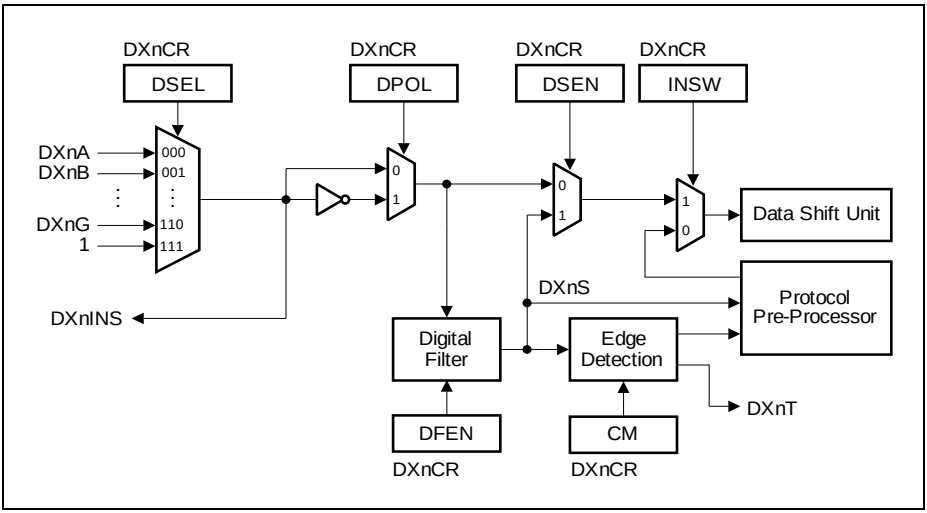


Figure 17-11 Input Conditioning for DX[2:1]

If the input signals are handled by a protocol pre-processor, the data shift unit is directly connected to the protocol pre-processor by INSW = 0. The protocol pre-processor is connected to the synchronized input signal DXnS and, depending on the selected protocol, also evaluates the edges.

To support delay compensation in SSC and IIS protocols, the DX1 input stage additionally allows the receive shift clock to be controlled independently from the transmit shift clock through the bit DCEN. When DCEN = 0, the shift clock source is selected by INSW and is the same for both receive and transmit. When DCEN = 1, the receive shift clock is derived from the selected input line as shown in **Figure 17-12**.

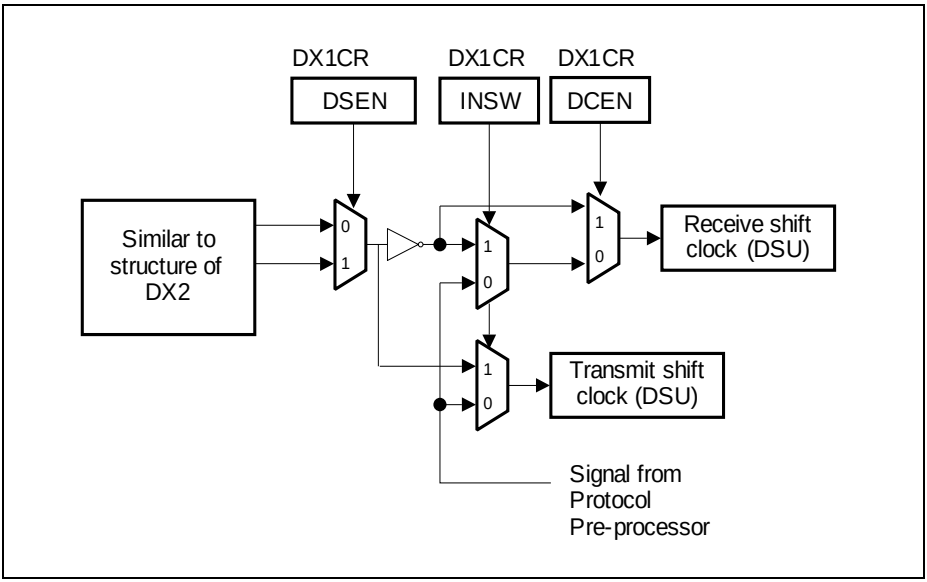


Figure 17-12 Delay Compensation Enable in DX1

17.2.3.2 Digital Filter

The digital filter can be enabled to reduce noise on the input signals. Before being filtered, the input signal becomes synchronized to f_{PB}. If the filter is disabled, signal DXnS corresponds to the synchronized input signal. If the filter is enabled, pulses shorter than one filter sampling period are suppressed in signal DXnS. After an edge of the synchronized input signal, signal DXnS changes to the new value if two consecutive samples of the new value have been detected.

In order to adapt the filter sampling period to different applications, it can be programmed. The first possibility is the system frequency f_{PB}. Longer pulses can be suppressed if the fractional divider output frequency f_{FD} is selected. This frequency is programmable in a wide range and can also be used to determine the baud rate of the data transfers.

In addition to the synchronization delay of 2-3 periods of f_{PB}, an enabled filter adds a delay of up to two filter sampling periods between the selected input and signal DXnS.

17.2.3.3 Edge Detection

The synchronized (and optionally filtered) signal DXnS can be used as input to the data shift unit and is also an input to the selected protocol pre-processor. If the protocol pre-processor does not use the DXnS signal for protocol-specific handling, DXnS can be

used for other tasks, e.g. to control data transmissions in master mode (a data word can be tagged valid for transmission, see chapter about data buffering).

A programmable edge detection indicates that the desired event has occurred by activating the trigger signal DXnT (introducing a delay of one period of f_{PB} before a reaction to this event can take place).

17.2.3.4 Selected Input Monitoring

The selected input signal of each input stage has been made available with the signals DXnINS. These signals can be used in the system to trigger other actions, e.g. to generate interrupts.

17.2.3.5 Loop Back Mode

The USIC transmitter output signals can be connected to the corresponding receiver inputs of the same communication channel in loop back mode. Therefore, the input “G” of the input stages that are needed for the selected protocol have to be selected. In this case, drivers for ASC, SSC, and IIS can be evaluated on-chip without the connections to port pins. Data transferred by the transmitter can be received by the receiver as if it would have been sent by another communication partner.

17.2.4 Operating the Baud Rate Generator

The following blocks can be configured to operate the baud rate generator, see also **Figure 17-2**.

17.2.4.1 Fractional Divider

The fractional divider generates its output frequency f_{FD} by either dividing the input frequency f_{PB} by an integer factor n or by multiplication of n/1024. It has two operating modes:

- Normal divider mode (FDR.DM = 01_B):
  In this mode, the output frequency f_{FD} is derived from the input clock f_{PB} by an integer division by a value between 1 and 1024. The division is based on a counter FDR.RESULT that is incremented by 1 with f_{PB}. After reaching the value $3FF_H$, the counter is loaded with FDR.STEP and then continues counting. In order to achieve $f_{FD} = f_{PB}$, the value of STEP has to be programmed with $3FF_H$.
  The output frequency in normal divider mode is defined by the equation:

$$f_{FD} = f_{PB} \times \frac{1}{n} \qquad \text{with } n = 1024 - \text{STEP} \tag{17.1}$$

- Fractional divider mode (FDR.DM = 10_B):
  In this mode, the output frequency f_{FD} is derived from the input clock f_{PB} by a

fractional multiplication of n/1024 for a value of n between 0 and 1023. In general, the fractional divider mode allows to program the average output clock frequency with a finer granularity than in normal divider mode. Please note that in fractional divider mode f_{FD} can have a maximum period jitter of one f_{PB} period. This jitter is not accumulated over several cycles.
The frequency f_{FD} is generated by an addition of FDR.STEP to FDR.RESULT with f_{PB}. The frequency f_{FD} is based on the overflow of the addition result over $3FF_H$.
The output frequency in fractional divider mode is defined by the equation:

$$f_{FD} = f_{PB} \times \frac{n}{1024} \qquad \text{with n = STEP} \tag{17.2}$$

The output frequency f_{FD} of the fractional divider is selected for baud rate generation by BRG.CLKSEL = 00_B.

17.2.4.2 External Frequency Input

The baud rate can be generated referring to an external frequency input (instead of to f_{PB}) if in the selected protocol the input stage DX1 is not needed (DX1CTR.INSW = 0). In this case, an external frequency input signal at the DX1 input stage can be synchronized and sampled with the system frequency f_{PB}. It can be optionally filtered by the digital filter in the input stage. This feature allows data transfers with frequencies that can not be generated by the device itself, e.g. for specific audio frequencies.

If BRG.CLKSEL = 10_B, the trigger signal DX1T determines f_{DX1}. In this mode, either the rising edge, the falling edge, or both edges of the input signal can be used for baud rate generation, depending on the configuration of the DX1T trigger event by bit field DX1CTR.CM. The signal MCLK toggles with each trigger event of DX1T.

If BRG.CLKSEL = 11_B, the rising edges of the input signal can be used for baud rate generation. The signal MCLK represents the synchronized input signal DX1S.

Both, the high time and the low time of external input signal must each have a length of minimum 2 periods of f_{PB} to be used for baud rate generation.

17.2.4.3 Divider Mode Counter

The divider mode counter is used for an integer division delivering the output frequency f_{PDIV}. Additionally, two divider stages with a fixed division by 2 provide the output signals MCLK and SCLK with 50% duty cycle. If the fractional divider mode is used, the maximum fractional jitter of 1 period of f_{PB} can also appear in these signals. The output frequencies of this divider is controlled by register BRG.

In order to define a frequency ratio between the master clock MCLK and the shift clock SCLK, the divider stage for MCLK is located in front of the divider by PDIV+1, whereas the divider stage for SCLK is located at the output of this divider.

$$f_{MCLK} = \frac{f_{PIN}}{2} \tag{17.3}$$

$$f_{SCLK} = \frac{f_{PDIV}}{2} \tag{17.4}$$

In the case that the master clock is used as reference for external devices (e.g. for IIS components) and a fixed phase relation to SCLK and other timing signals is required, it is recommended to use the MCLK signal as input for the PDIV divider. If the MCLK signal is not used or a fixed phase relation is not necessary, the faster frequency f_{PIN} can be selected as input frequency.

$$\begin{aligned} f_{PDIV} &= f_{PIN} \times \frac{1}{PDIV + 1} && \text{if PPPEN = 0} \\ f_{PDIV} &= f_{MCLK} \times \frac{1}{PDIV + 1} && \text{if PPPEN = 1} \end{aligned} \tag{17.5}$$

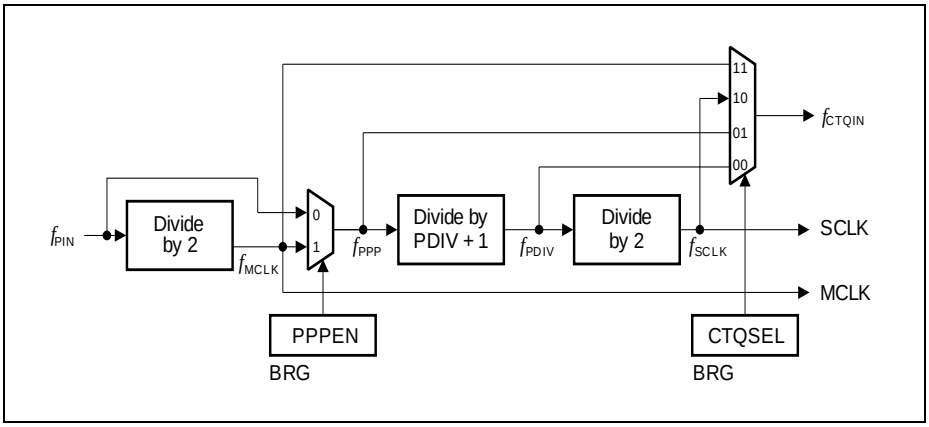


Figure 17-13 Divider Mode Counter

17.2.4.4 Capture Mode Timer

The capture mode timer is used for time interval measurement and is enabled by BRG.TMEN = 1. The timer works independently from the divider mode counter. Therefore, any serial data reception or transmission can continue while the timer is performing timing measurements. The timer counts f_{PPP} periods and stops counting

when it reaches its maximum value. Additionally, a baud rate generator interrupt event is generated (bit PSR.BRGIF becomes set).

If an event is indicated by DX0T or DX1T, the actual timer value is captured into bit field CMTR.CTV and the timer restarts from 0. Additionally, a transmit shift interrupt event is generated (bit PSR.TSIF becomes set).

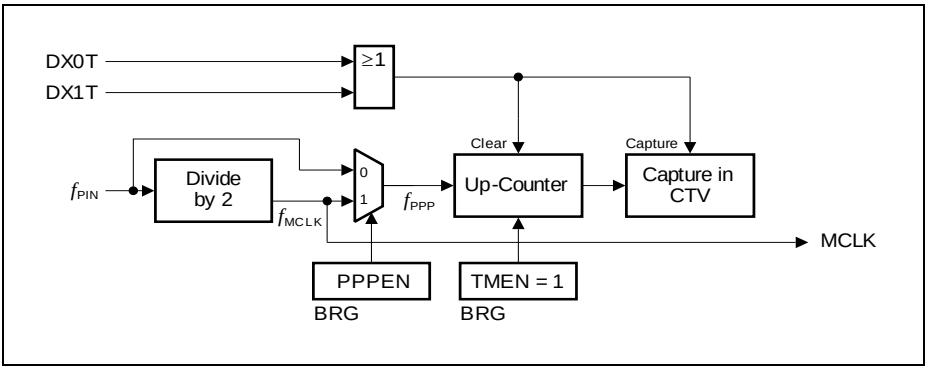


Figure 17-14 Protocol-Related Counter (Capture Mode)

The capture mode timer can be used to measure the baud rate in slave mode before starting or during data transfers, e.g. to measure the time between two edges of a data signal (by DX0T) or of a shift clock signal (by DX1T). The conditions to activate the DXnT trigger signals can be configured in each input stage.

17.2.4.5 Time Quanta Counter

The time quanta counter CTQ associated to the protocol pre-processor allows to generate time intervals for protocol-specific purposes. The length of a time quantum t_q is given by the selected input frequency f_{CTQIN} and the programmed pre-divider value.
The meaning of the time quanta depend on the selected protocol, please refer to the corresponding chapters for more protocol-specific information.

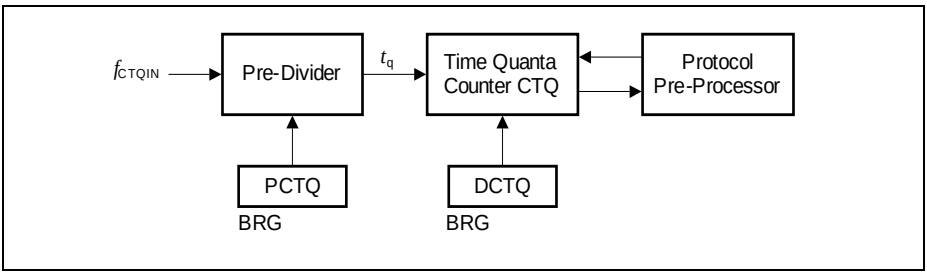


Figure 17-15 Time Quanta Counter

17.2.4.6 Master and Shift Clock Output Configuration

The master clock output signal MCLKOUT available at the corresponding output pin can be configured in polarity. The MCLK signal can be generated for each protocol in order to provide a kind of higher frequency time base compared to the shift clock.

The configuration mechanism of the master clock output signal MCLKOUT ensures that no shortened pulses can occur. Each MCLK period consists of two phases, an active phase, followed by a passive phase. The polarity of the MCLKOUT signal during the active phase is defined by the inverted level of bit BRG.MCLKCFG, evaluated at the start of the active phase. The polarity of the MCLKOUT signal during the passive phase is defined by bit BRG.MCLKCFG, evaluated at the start of the passive phase. If bit BRG.MCLKOUT is programmed with another value, the change is taken into account with the next change between the phases. This mechanism ensures that no shorter pulses than the length of a phase occur at the MCLKOUT output. In the example shown in **Figure 17-16**, the value of BRG.MCLKCFG is changed from 0 to 1 during the passive phase of MCLK period 2.

The generation of the MCLKOUT signal is enabled/disabled by the protocol pre-processor, based on bit PCR.MCLK. After this bit has become set, signal MCLKOUT is generated with the next active phase of the MCLK period. If PCR.MCLK = 0 (MCLKOUT generation disabled), the level for the passive phase is also applied for active phase.

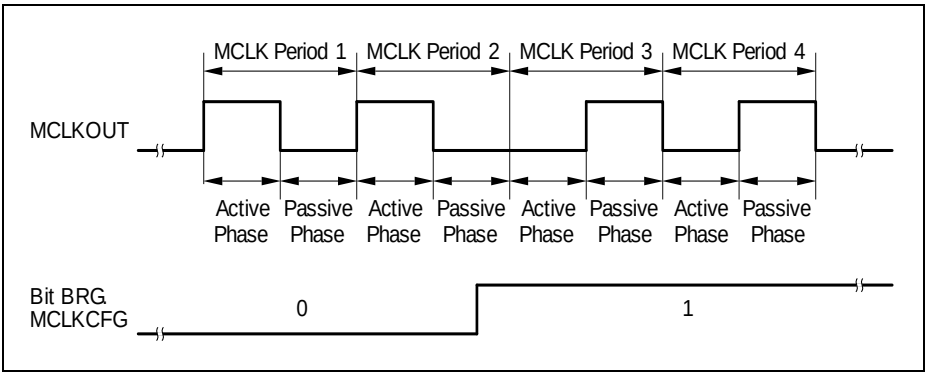


Figure 17-16 Master Clock Output Configuration

The shift clock output signal SCLKOUT available at the corresponding output pin can be configured in polarity and additionally, a delay of one period of f_{PDIV} (= half SCLK period) can be introduced. The delay allows to adapt the order of the shift clock edges to the application requirements. If the delay is used, it has to be taken into account for the calculation of the signal propagation times and loop delays.

The mechanism for the polarity control of the SCLKOUT signal is similar to the one for MCLKOUT, but based on bit field BRG.SCLKCFG. The generation of the SCLKOUT

signal is enabled/disabled by the protocol pre-processor. Depending on the selected protocol, the protocol pre-processor can control the generation of the SCLKOUT signal independently of the divider chain, e.g. for protocols without the need of a shift clock available at a pin, the SCLKOUT generation is disabled.

17.2.5 Operating the Transmit Data Path

The transmit data path is based on 16-bit wide transmit shift registers (TSR and TSR[3:0]) and a transmit buffer TBUF. The data transfer parameters like data word length, data frame length, or the shift direction are controlled commonly for transmission and reception by the shift control register SCTR. The transmit control and status register TCSR controls the transmit data handling and monitors the transmit status.

A change of the value of the data shift output signal DOUTx only happens at the corresponding edge of the shift clock input signal. The level of the last data bit of a data word/frame is held constant at DOUTx until the next data word begins with the next corresponding edge of the shift clock.

17.2.5.1 Transmit Buffering

The transmit shift registers can not be directly accessed by software, because they are automatically updated with the value stored in the transmit buffer TBUF if a currently transmitted data word is finished and new data is valid for transmission. Data words can be loaded directly into TBUF by writing to one of the transmit buffer input locations TBUFx (see **Page 17-32**) or, optionally, by a FIFO buffer stage (see **Page 17-38**).

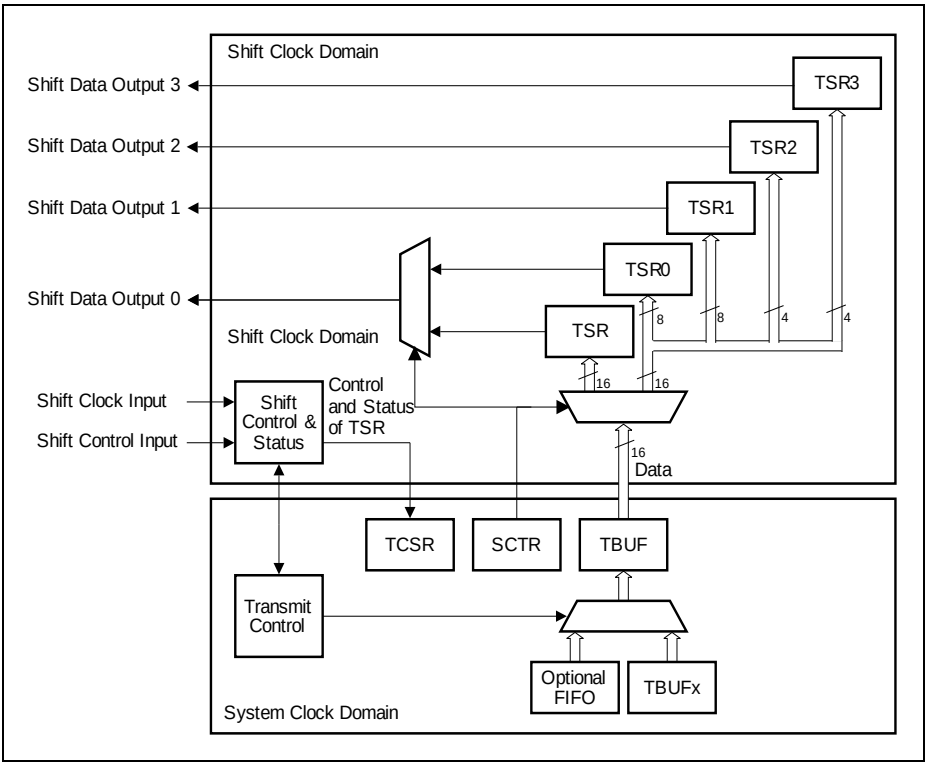


Figure 17-17 Transmit Data Path

17.2.5.2 Transmit Data Shift Mode

The transmit shift data can be selected to be shifted out one, two or four bits at a time through the corresponding number of output lines. This option allows the USIC to support protocols such as the Dual- and Quad-SSC. The selection is done through the bit field DSM in the shift control register SCTR.

Note: The bit field SCTR.DSM controls the data shift mode for both the transmit and receive paths to allow the transmission and reception of data through one to four data lines.

For the shift mode with two or four parallel data outputs, the data word and frame length must be in multiples of two or four respectively. The number of data shifts required to output a specific data word or data frame length is thus reduced by the factor of the number of parallel data output lines. For example, to transmit a 16-bit data word through four output lines, only four shifts are required.

Depending on the shift mode, different transmit shift registers with different bit composition are used as shown in **Table 17-7**. Note that the 'n' in the table denotes the shift number less one, i.e. for the first data shift n = 0, the second data shift n = 1 and continues until the total number of shifts less one is reached.

For all transmit shift registers, whether the first bit shifted out is the MSB or LSB depends on the setting of SCTR.SDIR.

Table 17-7 Transmit Shift Register Composition

Transmit Shift Registers	Single Data Output (SCTR.DSM = 00_B)	Two Data Outputs (SCTR.DSM = 10_B)	Four Data Outputs (SCTR.DSM = 11_B)
TSR	All data bits	Not used	Not used
TSR0	Not used	Bit n*2	Bit n*4
TSR1	Not used	Bit n*2 + 1	Bit n*4 + 1
TSR2	Not used	Not used	Bit n*4 + 2
TSR3	Not used	Not used	Bit n*4 + 3

17.2.5.3 Transmit Control Information

The transmit control information TCI is a 5-bit value derived from the address x of the written TBUFx or INx input location. For example, writing to TBUF31 generates a TCI of 11111_B.

The TCI can be used as an additional control parameter for data transfers to dynamically change the data word length, the data frame length, or other protocol-specific functions (for more details about this topic, please refer to the corresponding protocol chapters). The way how the TCI is used in different applications can be programmed by the bits WLEMD, FLEMD, SELMD, WAMD and HPCMD in register TCSR. Please note that not all possible settings lead to useful system behavior.

- Word length control:
  If TCSR.WLEMD = 1, bit field SCTR.WLE is updated with TCI[3:0] if a transmit buffer input location TBUFx is written. This function can be used in all protocols to dynamically change the data word length between 1 and 16 data bits per data word. Additionally, bit TCSR.EOF is updated with TCI[4]. This function can be used in SSC master mode to control the slave select generation to finish data frames. It is recommended to program TCSR.FLEMD = TCSR.SELMD = TCSR.WAMD = TCSR.HPCMD = 0.
- Frame length control:
  If TCSR.FLEMD = 1, bit field SCTR.FLE[4:0] is updated with TCI[4:0] and SCTR.FLE[5] becomes 0 if a transmit buffer input location TBUFx is written. This function can be used in all protocols to dynamically change the data frame length

between 1 and 32 data bits per data frame. It is recommended to program TCSR.SELMD = TCSR.WLEMD = TCSR.WAMD = TCSR.HPCMD = 0.

- Select output control:
  If TCSR.SELMD = 1, bit field PCR.CTR[20:16] is updated with TCI[4:0] and PCR.CTR[23:21] becomes 0 if a transmit buffer input location TBUFx is written. This function can be used in SSC master mode to define the targeted slave device(s). It is recommended to program TCSR.WLEMD = TCSR.FLEMD = TCSR.WAMD = TCSR.HPCMD = 0.
- Word address control:
  If TCSR.WAMD = 1, bit TCSR.WA is updated with TCI[4] if a transmit buffer input location TBUFx is written. This function can be used in IIS mode to define if the data word is transmitted on the right or the left channel. It is recommended to program TCSR.WLEMD = TCSR.FLEMD = TCSR.SELMD = TCSR.HPCMD = 0.
- Hardware Port control:
  If TCSR.HPCMD = 1, bit field SCTR.DSM is updated with TCI[1:0] if a transmit buffer input location TBUFx is written. This function can be used in SSC protocols to dynamically change the number of data input and output lines to set up for standard, dual and quad SSC formats.
  Additionally, bit TCSR.HPCDIR is updated with TCI[2]. This function can be used in SSC protocols to control the pin(s) direction when the hardware port control function is enabled through CCR.HPCEN = 1. It is recommended to program TCSR.FLEMD = TCSR.WLEMD = TCSR.SELMD = TCSR.WAMD = 0.

17.2.5.4 Transmit Data Validation

The data word in the transmit buffer TBUF can be tagged valid or invalid for transmission by bit TCSR.TDV (transmit data valid). A combination of data flow related and event related criteria define whether the data word is considered valid for transmission. A data validation logic checks the start conditions for each data word. Depending on the result of the check, the transmit shift register is loaded with different values, according to the following rules:

- If a USIC channel is the communication master (it defines the start of each data word transfer), a data word transfer can only be started with valid data in the transmit buffer TBUF. In this case, the transmit shift register is loaded with the content of TBUF, that is not changed due to this action.
- If a USIC channel is a communication slave (it can not define the start itself, but has to react), a data word transfer requested by the communication master has to be started independently of the status of the data word in TBUF. If a data word transfer is requested and started by the master, the transmit shift register is loaded at the first corresponding shift clock edge either with the data word in TBUF (if it is valid for transmission) or with the level defined by bit SCTR.PDL (if the content of TBUF has not been valid at the transmission start). In both cases, the content of TBUF is not changed.

The control and status bits for the data validation are located in register TCSR. The data validation is based on the logic blocks shown in **Figure 17-18**.

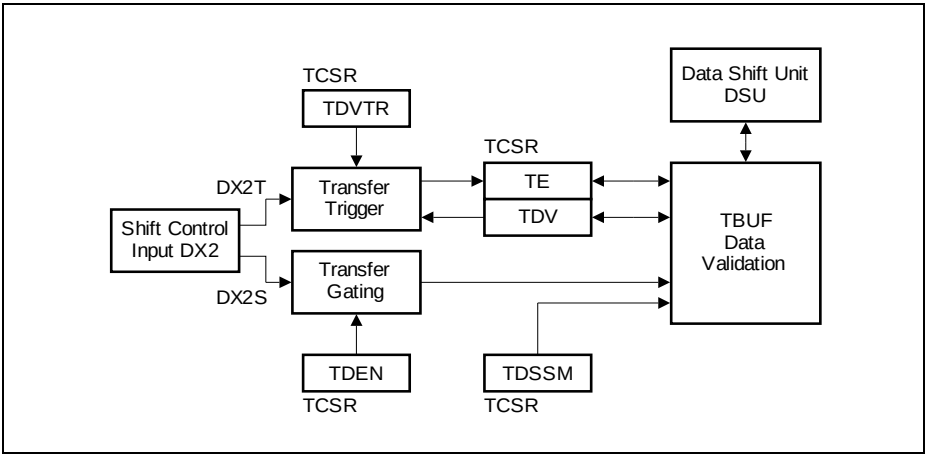


Figure 17-18 Transmit Data Validation

- A transfer gating logic enables or disables the data word transfer from TBUF under software or under hardware control. If the input stage DX2 is not needed for data shifting, signal DX2S can be used for gating purposes. The transfer gating logic is controlled by bit field TCSR.TDEN.
- A transfer trigger logic supports data word transfers related to events, e.g. timer based or related to an input pin. If the input stage DX2 is not needed for data shifting, signal DX2T can be used for trigger purposes. The transfer trigger logic is controlled by bit TCSR.TDVTR and the occurrence of a trigger event is indicated by bit TCSR.TE. For example, this can be used for triggering the data transfer upon receiving the Clear to Send (CTS) signal at DX2 in the RS-232 protocol.
- A data validation logic combining the inputs from the gating logic, the triggering logic and DSU signals. A transmission of the data word located in TBUF can only be started if the gating enables the start, bit TCSR.TDV = 1, and bit TCSR.TE = 1.
  The content of the transmit buffer TBUF should not be overwritten with new data while it is valid for transmission and a new transmission can start. If the content of TBUF has to be changed, it is recommended to clear bit TCSR.TDV by writing FMR.MTDV = 10_B before updating the data. Bit TCSR.TDV becomes automatically set when TBUF is updated with new data. Another possibility are the interrupts TBI (for ASC and IIC) or RSI (for SSC and IIS) indicating that a transmission has started. While a transmission is in progress, TBUF can be loaded with new data. In this case the user has to take care that an update of the TBUF content takes place before a new transmission starts.

With this structure, the following data transfer functionality can be achieved:

- If bit TCSR.TDSSM = 0, the content of the transmit buffer TBUF is always considered as valid for transmission. The transfer trigger mechanism can be used to start the transfer of the same data word based on the selected event (e.g. on a timer base or an edge at a pin) to realize a kind of life-sign mechanism. Furthermore, in slave mode, it is ensured that always a correct data word is transmitted instead of the passive data level.
- Bit TCSR.TDSSM = 1 has to be programmed to allow word-by-word data transmission with a kind of single-shot mechanism. After each transmission start, a new data word has to be loaded into the transmit buffer TBUF, either by software write actions to one of the transmit buffer input locations TBUFx or by an optional data buffer (e.g. FIFO buffer). To avoid that data words are sent out several times or to allow data handling with an additional data buffer (e.g. FIFO), bit TCSR.TDSSM has to be 1.
- Bit TCSR.TDV becoming automatically set when a new data word is loaded into the transmit buffer TBUF, a transmission start can be requested by a write action of the data to be transmitted to at least the low byte of one of the transmit buffer input locations TBUFx. The additional information TCI can be used to control the data word length or other parameters independently for each data word by a single write access.
- Bit field FMR.MTDV allows software driven modification (set or clear) of bit TCSR.TDV. Together with the gating control bit field TCSR.TDEN, the user can set up the transmit data word without starting the transmission. A possible program sequence could be: clear TCSR.TDEN = 00_B, write data to TBUFx, clear TCSR.TDV by writing FMR.MTDV = 10_B, re-enable the gating with TCSR.TDEN = 01_B and then set TCSR.TDV under software control by writing FMR.MTDV = 01_B.

17.2.6 Operating the Receive Data Path

The receive data path is based on two sets of 16-bit wide receive shift registers RSR0[3:0] and RSR1[3:0] and a receive buffer for each of the set (RBUF0 and RBUF1). The data transfer parameters like data word length, data frame length, or the shift direction are controlled commonly for transmission and reception by the shift control registers.

Register RBUF01SR monitors the status of RBUF0 and RBUF1.

17.2.6.1 Receive Buffering

The receive shift registers cannot be directly accessed by software, but their contents are automatically loaded into the receive buffer registers RBUF0 (or RBUF1 respectively) if a complete data word has been received or the frame is finished. The received data words in RBUF0 or RBUF1 can be read out in the correct order directly from register RBUF or, optionally, from a FIFO buffer stage (see **Page 17-38**).

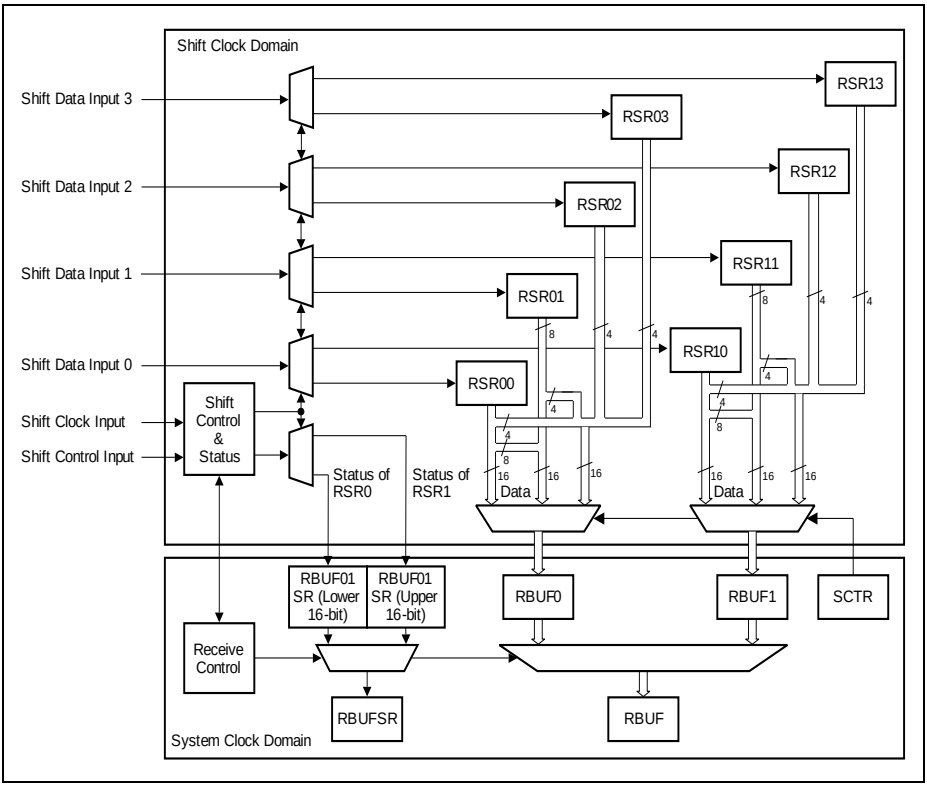


Figure 17-19 Receive Data Path

17.2.6.2 Receive Data Shift Mode

Receive data can be selected to be shifted in one, two or four bits at a time through the corresponding number of input stages and data input lines. This option allows the USIC to support protocols such as the Dual- and Quad-SSC. The selection is done through the bit field DSM in the shift control register SCTR.

Note: The bit field SCTR.DSM controls the data shift mode for both the transmit and receive paths to allow the transmission and reception of data through one to four data lines.

For the shift mode with two or four parallel data inputs, the data word and frame length must be in multiples of two or four respectively. The number of data shifts required to input a specific data word or data frame length is thus reduced by the factor of the

number of parallel data input lines. For example, to receive a 16-bit data word through four input lines, only four shifts are required.

Depending on the shift mode, different receive shift registers with different bit composition are used as shown in **Table 17-7**. Note that the 'n' in the table denotes the shift number less one, i.e. for the first data shift n = 0, the second data shift n = 1 and continues until the total number of shifts less one is reached.

For all receive shift registers, whether the first bit shifted in is the MSB or LSB depends on the setting of SCTR.SDIR.

Table 17-8 Receive Shift Register Composition

Receive Shift Registers	Input stage used	Single Data Input (SCTR.DSM = 00_B)	Two Data Inputs (SCTR.DSM = 10_B)	Four Data Inputs (SCTR.DSM = 11_B)
RSRx0	DX0	All data bits	Bit n*2	Bit n*4
RSRx1	DX3	Not used	Bit n*2 + 1	Bit n*4 + 1
RSRx2	DX4	Not used	Not used	Bit n*4 + 2
RSRx3	DX5	Not used	Not used	Bit n*4 + 3

17.2.6.3 Baud Rate Constraints

The following baud rate constraints have to be respected to ensure correct data reception and buffering. The user has to take care about these restrictions when selecting the baud rate and the data word length with respect to the module clock frequency f_{PB}.

- A received data word in a receiver shift registers RSRx[3:0] must be held constant for at least 4 periods of f_{PB} in order to ensure correct loading of the related receiver buffer register RBUFx.
- The shift control signal has to be constant inactive for at least 5 periods of f_{PB} between two consecutive frames in order to correctly detect the end of a frame.
- The shift control signal has to be constant active for at least 1 period of f_{PB} in order to correctly detect a frame (shortest frame).
- A minimum setup and hold time of the shift control signal with respect to the shift clock signal has to be ensured.

17.2.7 Hardware Port Control

Hardware port control is intended for SSC protocols with half-duplex configurations, where a single port pin is used for both input and output data functions, to control the pin direction through a dedicated hardware interface. All settings in Pn_IOCRy.PCx, except for the input pull device selection and output driver type (open drain or push-pull), are overruled by the hardware port control.

Input pull device selection is done through the Pn_IOCRy.PCx as before, while the output driver is fixed to push-pull-only in this mode.

One, two or four port pins can be selected with the hardware port control to support SSC protocols with multiple bi-directional data lines, such as dual- and quad-SSC. This selection and the enable/disable of the hardware port control is done through CCR.HPCEN. The direction of all selected pins is controlled through a single bit SCTR.HPCDIR.

SCTR.HPCDIR is automatically shadowed with the start of each data word to prevent changing of the pin direction in the middle of a data word transfer.

17.2.8 Operating the FIFO Data Buffer

The FIFO data buffers of a USIC module are built in a similar way, with transmit buffer and receive buffer capability for each channel. Depending on the device, the amount of available FIFO buffer area can vary. In the XMC4500, totally 64 buffer entries can be distributed among the transmit or receive FIFO buffers of both channels of the USIC module.

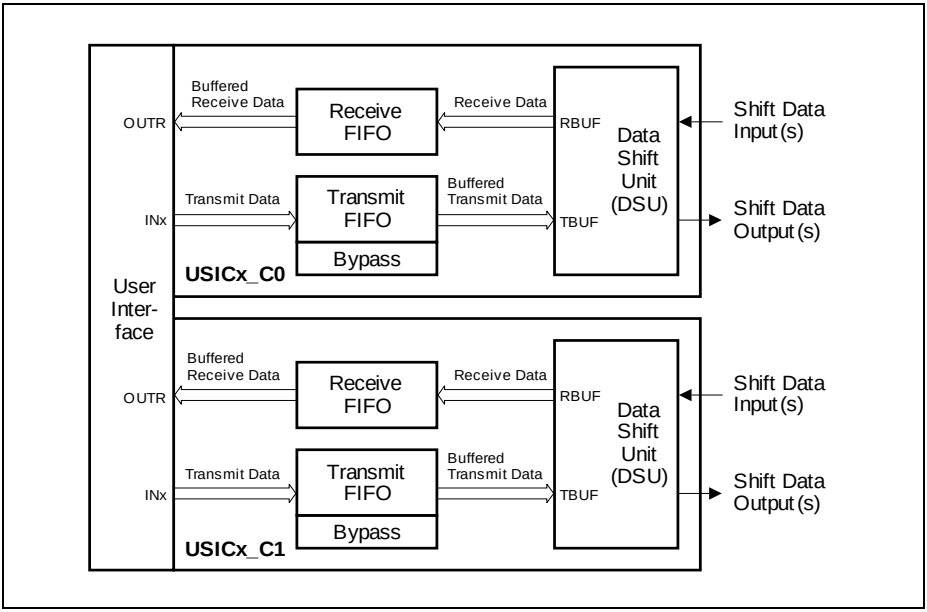


Figure 17-20 FIFO Buffer Overview

In order to operate the FIFO data buffers, the following issues have to be considered:

- FIFO buffer available and selected:
  The transmit FIFO buffer and the bypass structure are only available if CCFG.TB = 1, whereas the receive FIFO buffer is only available if CCFG.RB = 1.
  It is recommended to configure all buffer parameters while there is no data traffic for this USIC channel and the FIFO mechanism is disabled by TBCTR.SIZE = 0 (for transmit buffer) or RBCTR.SIZE = 0 (for receive buffer). The allocation of a buffer area by writing TBCTR or RBCTR has to be done while the corresponding FIFO buffer is disabled. The FIFO buffer interrupt control bits can be modified independently of data traffic.
- FIFO buffer setup:
  The total amount of available FIFO buffer entries limits the length of the transmit and receive buffers for each USIC channel.
- Bypass setup:
  In addition to the transmit FIFO buffer, a bypass can be configured as described on **Page 17-49**.

17.2.8.1 FIFO Buffer Partitioning

If available, the FIFO buffer area consists of a defined number of FIFO buffer entries, each containing a data part and the associated control information (RCI for receive data, TCI for transmit data). One FIFO buffer entry represents the finest granularity that can be allocated to a receive FIFO buffer or a transmit FIFO buffer. All available FIFO buffer entries of a USIC module are located one after the other in the FIFO buffer area. The overall counting starts with FIFO entry 0, followed by 1, 2, etc.

For each USIC module, a certain number of FIFO entries is available, that can be allocated to the channels of the same USIC module. It is not possible to assign FIFO buffer area to USIC channels that are not located within the same USIC module.
For each USIC channel, the size of the transmit and the receive FIFO buffer can be chosen independently. For example, it is possible to allocate the full amount of available FIFO entries as transmit buffer for one USIC channel. Some possible scenarios of FIFO buffer partitioning are shown in **Figure 17-21**.

Each FIFO buffer consists of a set of consecutive FIFO entries. The size of a FIFO data buffer can only be programmed as a power of 2, starting with 2 entries, then 4 entries, then 8 entries, etc. A FIFO data buffer can only start at a FIFO entry aligned to its size. For example, a FIFO buffer containing n entries can only start with FIFO entry 0, n, 2*n, 3*n, etc. and consists of the FIFO entries [x*n, (x+1)*n-1], with x being an integer number (incl. 0). It is not possible to have “holes” with unused FIFO entries within a FIFO buffer, whereas there can be unused FIFO entries between two FIFO buffers.

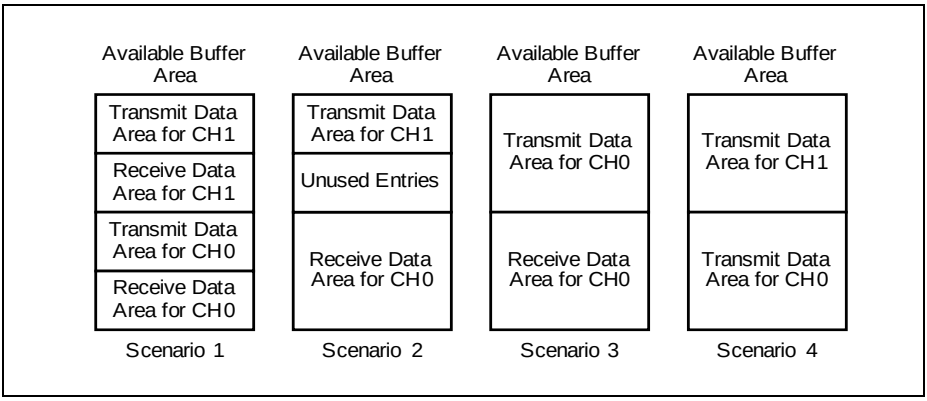


Figure 17-21 FIFO Buffer Partitioning

The data storage inside the FIFO buffers is based on pointers, that are internally updated whenever the data contents of the FIFO buffers have been modified. This happens automatically when new data is put into a FIFO buffer or the oldest data is taken from a FIFO buffer. As a consequence, the user program does not need to modify the pointers for data handling. Only during the initialization phase, the start entry of a FIFO buffer has to be defined by writing the number of the first FIFO buffer entry in the FIFO buffer to the corresponding bit field DPTR in register RBCTR (for a receive FIFO buffer) or TBCTR (for a transmit FIFO buffer) while the related bit field RBCTR.SIZE=0 (or TBCTR.SIZE = 0, respectively). The assignment of buffer entries to a FIFO buffer (regarding to size and pointers) must not be changed by software while the related USIC channel is taking part in data traffic.

17.2.8.2 Transmit Buffer Events and Interrupts

The transmit FIFO buffer mechanism detects the following events, that can lead to interrupts (if enabled):

- Standard transmit buffer event
- Transmit buffer error event

Standard Transmit Buffer Event

The standard transmit buffer event is triggered by the filling level of the transmit buffer (given by TRBSR.TBFLVL) exceeding (TBCTR.LOF = 1) or falling below (TBCTR.LOF = 0)[1)] a programmed limit (TBCTR.LIMIT).

1) If the standard transmit buffer event is used to indicate that new data has to be written to one of the INx locations, TBCTR.LOF = 0 should be programmed.

If the event trigger with TRBSR.STBT feature is disabled (TBCTR.STBTEN = 0), the trigger of the standard transmit buffer event is based on the transition of the fill level from equal to below or above the limit, not the fact of being below or above.

If TBCTR.STBTEN = 1, the transition of the fill level below or above the programmed limit additionally sets TRBSR.STBT. This bit triggers also the standard transmit buffer event whenever there is a transfer data to TBUF event or write data to INx event.

The way TRBSR.STBT is cleared depends on the trigger mode (selected by TBCTR.STBTM). If TBCTR.STBTM = 0, TRBSR.STBT is cleared by hardware when the buffer fill level equals the programmed limit again (TRBSR.TBFLVL = TBCTR.LIMIT). If TBCTR.STBTM = 1, TRBSR.STBT is cleared by hardware when the buffer fill level equals the buffer size (TRBSR.TBFLVL = TBCTR.SIZE).

Note: The flag TRBSR.STBI is set only when the transmit buffer fill level exceeds or falls below the programmed limit (depending on TBCTR.LOF setting). Standard transmit buffer events triggered by TRBSR.STBT does not set the flag.

Figure 17-22 shows examples of the standard transmit buffer event with the different TBCTR.STBTEN and TBCTR.STBTM settings. These examples are meant to illustrate the hardware behaviour and might not always represent real application use cases.

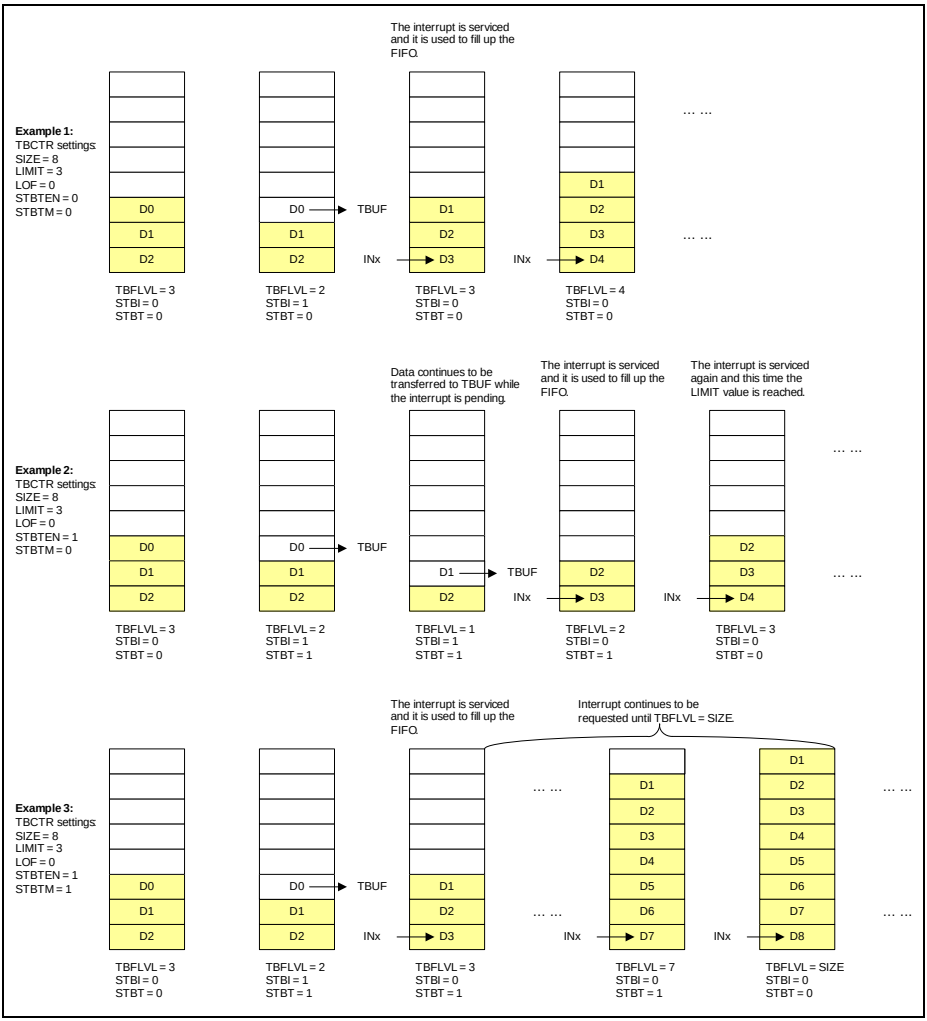


Figure 17-22 Standard Transmit Buffer Event Examples

Transmit Buffer Error Event

The transmit buffer error event is triggered when software has written to a full buffer. The written value is ignored.

Transmit Buffer Events and Interrupt Handling

Figure 17-23 shows the transmit buffer events and interrupts.

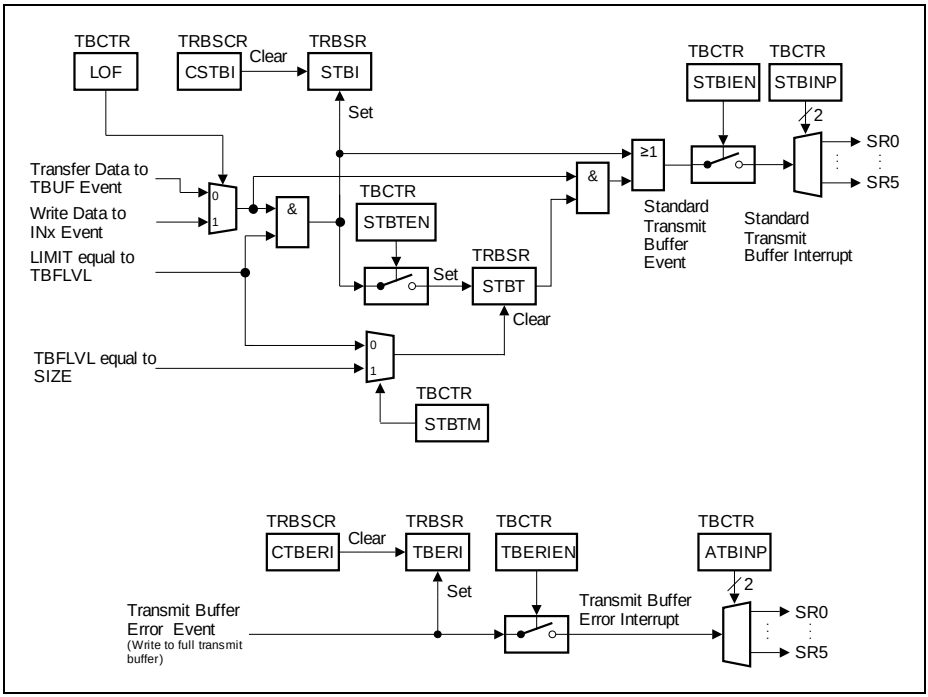


Figure 17-23 Transmit Buffer Events

Table 17-9 shows the registers, bits and bit fields to indicate the transmit buffer events and to control the interrupts related to the transmit FIFO buffers of a USIC channel.

Table 17-9 Transmit Buffer Events and Interrupt Handling

Event	Indication Flag	Indication cleared by	Interrupt enabled by	SRx Output selected by
Standard transmit buffer event	TRBSR.STBI	TRBSCR.CSTBI	TBCTR.STBIEN	TBCTR.STBINP
	TRBSR.STBT	Cleared by hardware		
Transmit buffer error event	TRBSR.TBERI	TRBSCR.CTBERI	TBCTR.TBERIEN	TBCTR.ATBINP

17.2.8.3 Receive Buffer Events and Interrupts

The receive FIFO buffer mechanism detects the following events, that can lead to an interrupt (if enabled):

- Standard receive buffer event
- Alternative receive buffer event
- Receive buffer error event

The standard receive buffer event and the alternative receive buffer event can be programmed to two different modes, one referring to the filling level of the receive buffer, the other one related to a bit position in the receive control information RCI of the data word that becomes available in OUTR.

If the interrupt generation refers to the filling level of the receive FIFO buffer, only the standard receive buffer event is used, whereas the alternative receive buffer event is not used. This mode can be selected to indicate that a certain amount of data has been received, without regarding the content of the associated RCI.

If the interrupt generation refers to RCI, the filling level is not taken into account. Each time a new data word becomes available in OUTR, an event is detected. If bit RCI[4] = 0, a standard receive buffer event is signaled, otherwise an alternative receive buffer device (RCI[4] = 1). Depending on the selected protocol and the setting of RBCTR.RCIM, the value of RCI[4] can hold different information that can be used for protocol-specific interrupt handling (see protocol sections for more details).

Standard Receive Buffer Event in Filling Level Mode

In filling level mode (RBCTR.RNM = 0), the standard receive buffer event is triggered by the filling level of the receive buffer (given by TRBSR.RBFLVL) exceeding (RBCTR.LOF = 1) or falling below (RBCTR.LOF = 0) a programmed limit (RBCTR.LIMIT).[1)]

If the event trigger with bit TRBSR.SRBT feature is disabled (RBCTR.SRBTEN = 0), the trigger of the standard receive buffer event is based on the transition of the fill level from equal to below or above the limit, not the fact of being below or above.

If RBCTR.SRBTEN = 1, the transition of the fill level below or above the programmed limit additionally sets the bit TRBSR.SRBT. This bit also triggers the standard receive buffer event each time there is a data read out event or new data received event.

The way TRBSR.SRBT is cleared depends on the trigger mode (selected by RBCTR.SRBTM). If RBCTR.SRBTM = 0, TRBSR.SRBT is cleared by hardware when the buffer fill level equals the programmed limit again (TRBSR.RBFLVL = RBCTR.LIMIT). If RBCTR.SRBTM = 1, TRBSR.SRBT is cleared by hardware when the buffer fill level equals 0 (TRBSR.RBFLVL = 0).

1) If the standard receive buffer event is used to indicate that new data has to be read from OUTR, RBCTR.LOF = 1 should be programmed.

Note: The flag TRBSR.SRBI is set only when the receive buffer fill level exceeds or falls below the programmed limit (depending on RBCTR.LOF setting). Standard receive buffer events triggered by TRBSR.SRBT does not set the flag.

Figure 17-24 shows examples of the standard receive buffer event with the different RBCTR.SRBTEN and RBCTR.SRBTM settings. These examples are meant to illustrate the hardware behaviour and might not always represent real application use cases.

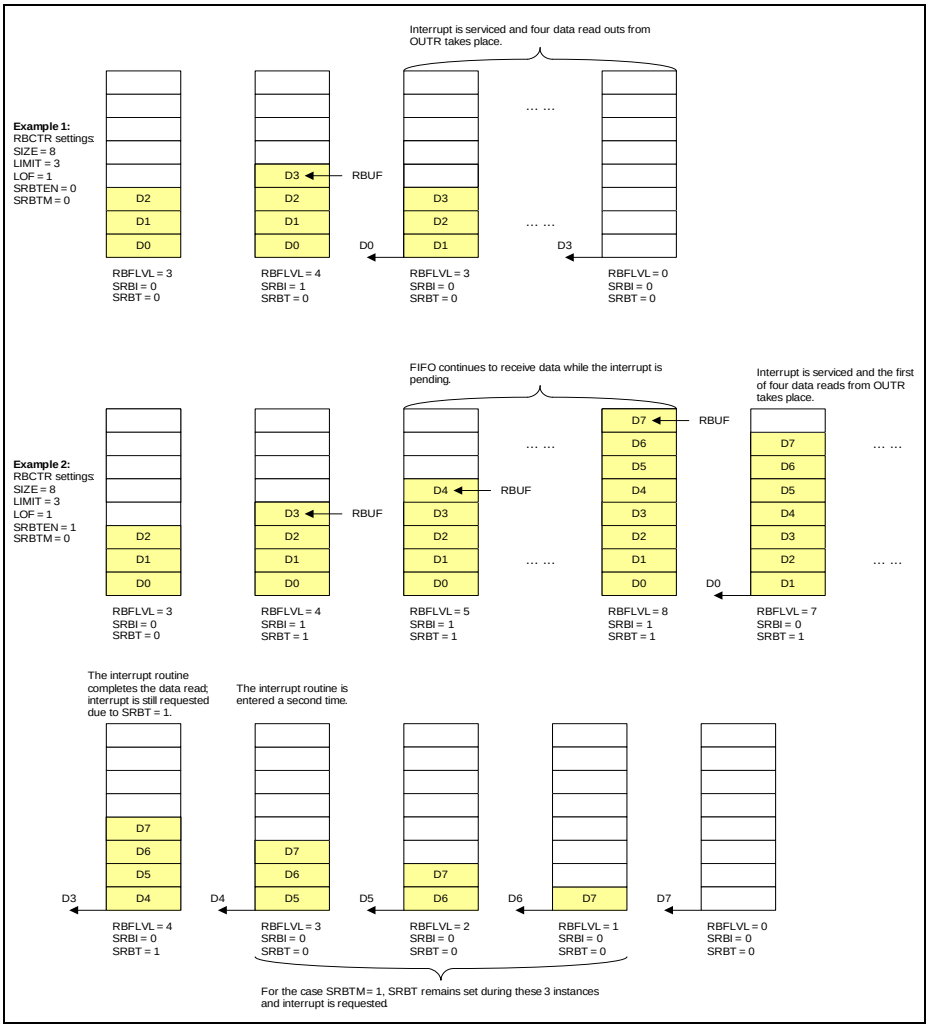


Figure 17-24 Standard Receive Buffer Event Examples

Standard and Alternate Receive Buffer Events in RCI Mode

In RCI mode (RBCTR.RNM = 1), the standard receive buffer event is triggered when the OUTR stage is updated with a new data value with RCI[4] = 0.

If the OUTR stage is updated with a new data value with RCI[4] = 1, an alternate receive buffer event is triggered instead.

Receive Buffer Error Event

The receive buffer error event is triggered if the software reads from an empty buffer, regardless of RBCTR.RNM value. The read data is invalid.

Receive Buffer Events and Interrupt Handling

Figure 17-25 shows the receiver buffer events and interrupts in filling level mode.

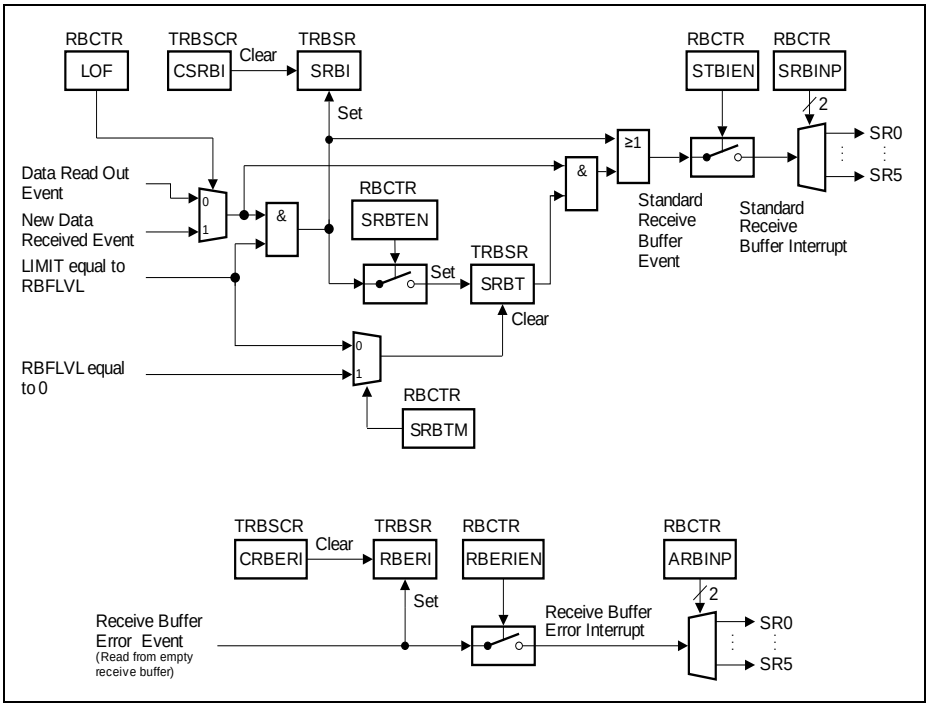


Figure 17-25 Receiver Buffer Events in Filling Level Mode

Figure 17-26 shows the receiver buffer events and interrupts in RCI mode.

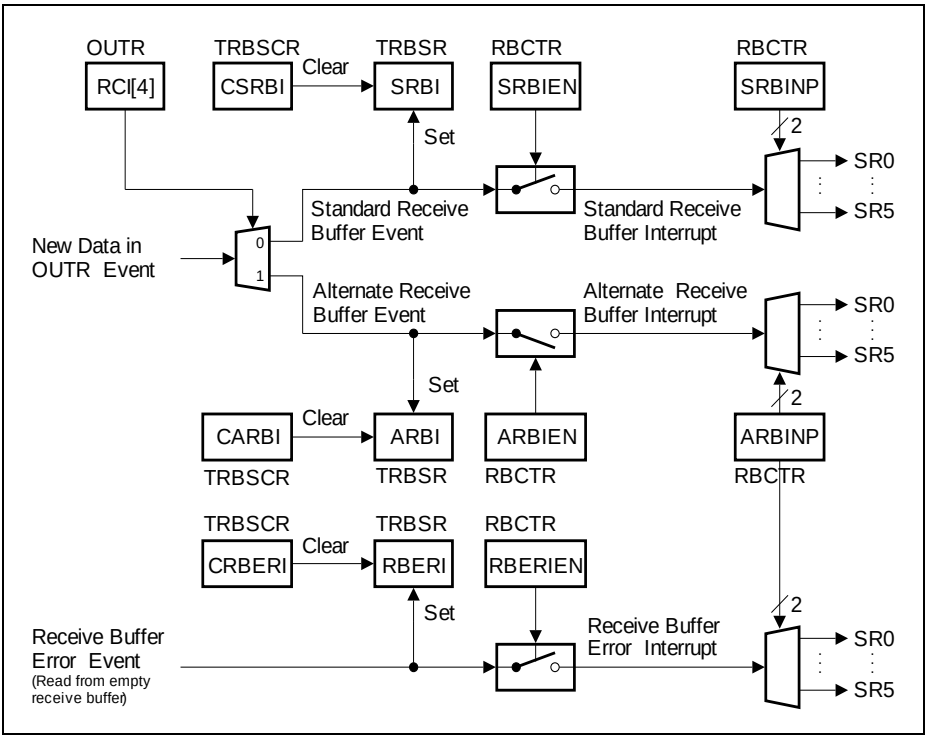


Figure 17-26 Receiver Buffer Events in RCI Mode

Table 17-10 shows the registers, bits and bit fields to indicate the receive buffer events and to control the interrupts related to the receive FIFO buffers of a USIC channel.

Table 17-10 Receive Buffer Events and Interrupt Handling

Event	Indication Flag	Indication cleared by	Interrupt enabled by	SRx Output selected by
Standard receive buffer event	TRBSR. SRBI	TRBSCR. CSRBI	RBCTR. SRBIEN	RBCTR. SRBINP
	TRBSR. SRBT	Cleared by hardware		

Table 17-10 Receive Buffer Events and Interrupt Handling (cont'd)

Event	Indication Flag	Indication cleared by	Interrupt enabled by	SRx Output selected by
Alternative receive buffer event	TRBSR. ARBI	TRBSCR. CARBI	RBCTR. ARBIEN	RBCTR. ARBINP
Receive buffer error event	TRBSR. RBERI	TRBSCR. CRBERI	RBCTR. RBERIEN	RBCTR. ARBINTXDP

17.2.8.4 FIFO Buffer Bypass

The data bypass mechanism is part of the transmit FIFO control block. It allows to introduce a data word in the data stream without modifying the transmit FIFO buffer contents, e.g. to send a high-priority message. The bypass structure consists of a bypass data word of maximum 16 bits in register BYP and some associated control information in register BYPCR. For example, these bits define the word length of the bypass data word and configure a transfer trigger and gating mechanism similar to the one for the transmit buffer TBUF.

The bypass data word can be tagged valid or invalid for transmission by bit BYRCR.BDV (bypass data valid). A combination of data flow related and event related criteria define whether the bypass data word is considered valid for transmission. A data validation logic checks the start conditions for this data word. Depending on the result of the check, the transmit buffer register TBUF is loaded with different values, according to the following rules:

- Data from the transmit FIFO buffer or the bypass data can only be transferred to TBUF if TCSR.TDV = 0 (TBUF is empty).
- Bypass data can only be transferred to TBUF if the bypass is enabled by BYPCR.BDEN or the selecting gating condition is met.
- If the bypass data is valid for transmission and has either a higher transmit priority than the FIFO data or if the transmit FIFO is empty, the bypass data is transferred to TBUF.
- If the bypass data is valid for transmission and has a lower transmit priority than the FIFO buffer that contains valid data, the oldest transmit FIFO data is transferred to TBUF.
- If the bypass data is not valid for transmission and the FIFO buffer contains valid data, the oldest FIFO data is transferred to TBUF.
- If neither the bypass data is valid for transmission nor the transmit FIFO buffer contains valid data, TBUF is unchanged.

The bypass data validation is based on the logic blocks shown in **Figure 17-27**.

- A transfer gating logic enables or disables the bypass data word transfer to TBUF under software or under hardware control. If the input stage DX2 is not needed for

data shifting, signal DX2S can be used for gating purposes. The transfer gating logic is controlled by bit field BYPCR.BDEN.

- A transfer trigger logic supports data word transfers related to events, e.g. timer based or related to an input pin. If the input stage DX2 is not needed for data shifting, signal DX2T can be used for trigger purposes. The transfer trigger logic is controlled by bit BYPCR.BDVTR.
- A bypass data validation logic combining the inputs from the gating logic, the triggering logic and TCSR.TDV.

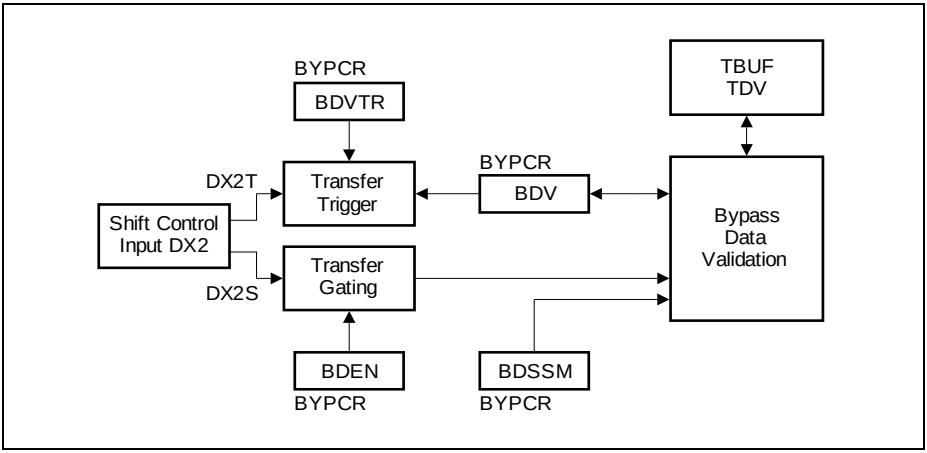


Figure 17-27 Bypass Data Validation

With this structure, the following bypass data transfer functionality can be achieved:

- Bit BYPCR.BDSSM = 1 has to be programmed for a single-shot mechanism. After each transfer of the bypass data word to TBUF, the bypass data word has to be tagged valid again. This can be achieved either by writing a new bypass data word to BYP or by DX2T if BDVTR = 1 (e.g. trigger on a timer base or an edge at a pin).
- Bit BYPCR.BDSSM = 0 has to be programmed if the bypass data is permanently valid for transmission (e.g. as alternative data if the data FIFO runs empty).

17.2.8.5 FIFO Access Constraints

The data in the shared FIFO buffer area is accessed by the hardware mechanisms for data transfer of each communication channel (for transmission and reception) and by software to read out received data or to write data to be transmitted. As a consequence, the data delivery rate can be limited by the FIFO mechanism. Each access by hardware to the FIFO buffer area has priority over a software access, that is delayed in case of an access collision.

In order to avoid data loss and stalling of the CPU due to delayed software accesses, the

baud rate, the word length and the software access mechanism have to be taken into account. Each access to the FIFO data buffer area by software or by hardware takes one period of f_{PB}. Especially a continuous flow of very short, consecutive data words can lead to an access limitation.

17.2.8.6 Handling of FIFO Transmit Control Information

In addition to the transmit data, the transmit control information TCI can be transferred from the transmit FIFO or bypass structure to the USIC channel. Depending on the selected protocol and the enabled update mechanism, some settings of the USIC channel parameters can be modified. The modifications are based on the TCI of the FIFO data word loaded to TBUF or by the bypass control information if the bypass data is loaded into TBUF.

- TCSR.SELMD = 1: update of PCR.CTR[20:16] by FIFO TCI or BYPCR.BSELO with additional clear of PCR.CTR[23:21]
- TCSR.WLEMD = 1: update of SCTR.WLE and TCSR.EOF by FIFO TCI or BYPCR.BWLE (if the WLE information is overwritten by TCI or BWLE, the user has to take care that FLE is set accordingly)
- TCSR.FLEMD = 1: update of SCTR.FLE[4:0] by FIFO TCI or BYPCR.BWLE with additional clear of SCTR.FLE[5]
- TCSR.HPCMD = 1: update of SCTR.DSM and SCTR.HPCDIR by FIFO TCI or BYPCR.BHPC
- TCSR.WAMD = 1: update of TCSR.WA by FIFO TCI[4]

See **Section 17.2.5.3** for more details on TCI.

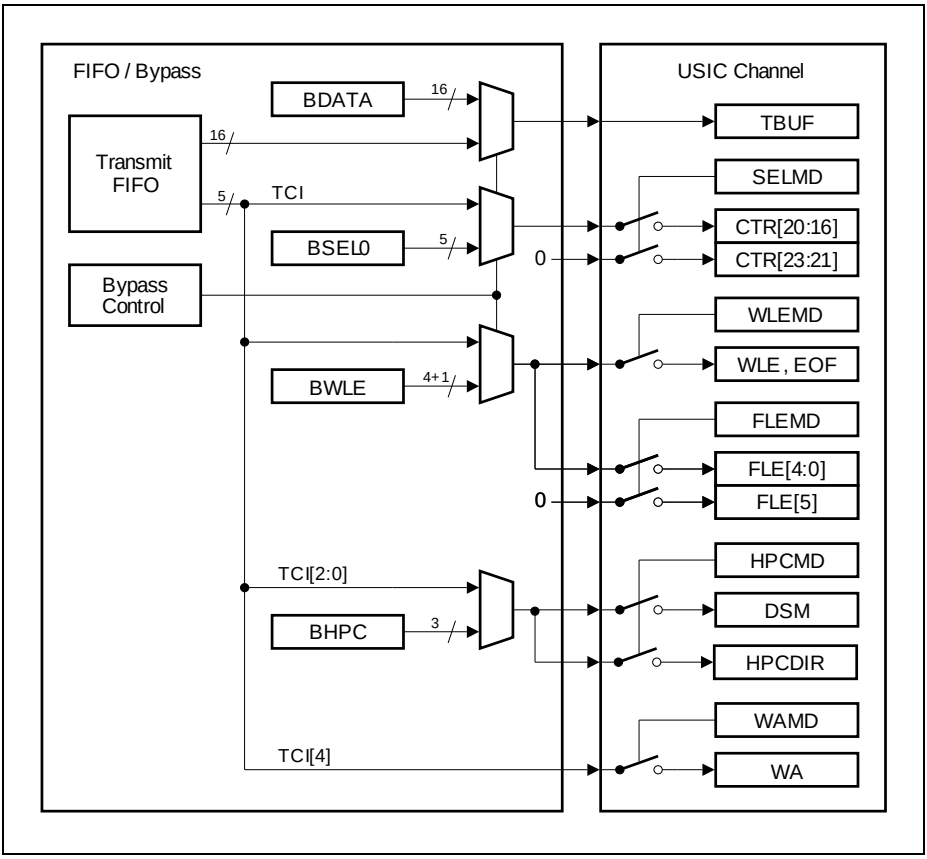


Figure 17-28 TCI Handling with FIFO / Bypass

17.3 Asynchronous Serial Channel (ASC = UART)

The asynchronous serial channel ASC covers the reception and the transmission of asynchronous data frames and provides a hardware LIN support. The receiver and transmitter being independent, frames can start at different points in time for transmission and reception. The ASC mode is selected by CCR.MODE = 0010_B with CCFG.ASC = 1 (ASC mode available).

17.3.1 Signal Description

An ASC connection is characterized by the use of a single connection line between a transmitter and a receiver. The receiver input RXD signal is handled by the input stage DX0.

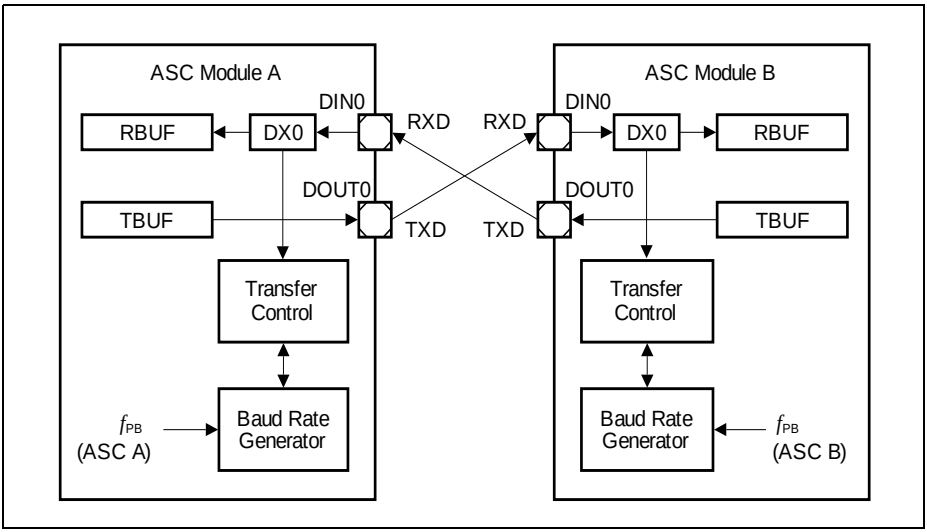


Figure 17-29 ASC Signal Connections for Full-Duplex Communication

For full-duplex communication, an independent communication line is needed for each transfer direction. **Figure 17-29** shows an example with a point-to-point full-duplex connection between two communication partners ASC A and ASC B.

For half-duplex or multi-transmitter communication, a single communication line is shared between the communication partners. **Figure 17-30** shows an example with a point-to-point half-duplex connection between ASC A and ASC B. In this case, the user has to take care that only one transmitter is active at a time. In order to support transmitter collision detection, the input stage DX1 can be used to monitor the level of the transmit line and to check if the line is in the idle state or if a collision occurred.
There are two possibilities to connect the receiver input DIN0 to the transmitter output

DOUT0. Communication partner ASC A uses an internal connection with only the transmit pin TXD, that is delivering its input value as RXD to the DX0 input stage for reception and to DX1 to check for transmitter collisions. Communication partner ASC B uses an external connection between the two pins TXD and RXD.

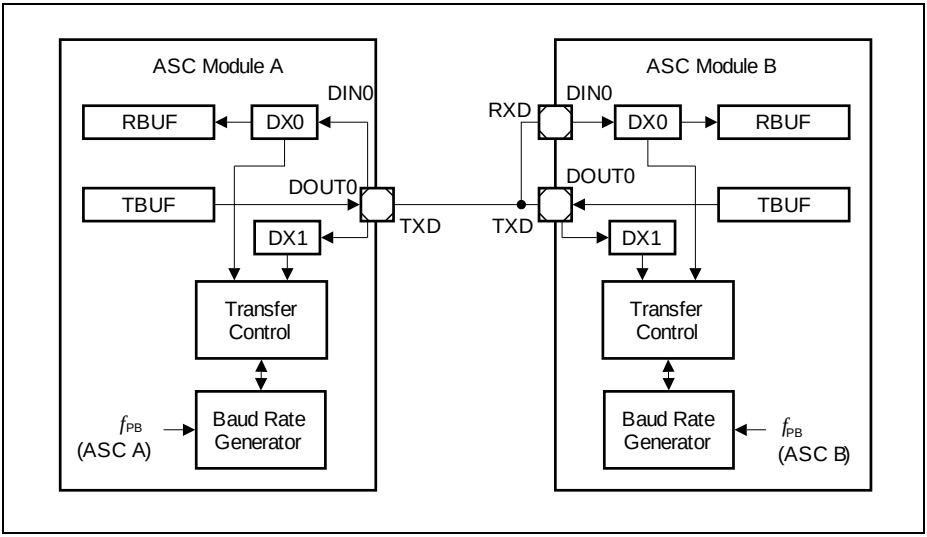


Figure 17-30 ASC Signal Connections for Half-Duplex Communication

17.3.2 Frame Format

A standard ASC frame is shown in **Figure 17-31**. It consists of:

- An idle time with the signal level 1.
- One start of frame bit (SOF) with the signal level 0.
- A data field containing a programmable number of data bits (1-63).
- A parity bit (P), programmable for either even or odd parity. It is optionally possible to handle frames without parity bit.
- One or two stop bits with the signal level 1.

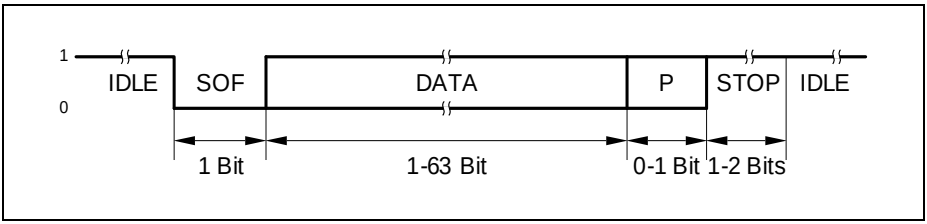


Figure 17-31 Standard ASC Frame Format

The protocol specific bits (SOF, P, STOP) are automatically handled by the ASC protocol state machine and do not appear in the data flow via the receive and transmit buffers.

17.3.2.1 Idle Time

The receiver and the transmitter independently check the respective data input lines (DX0, DX1) for being idle. The idle detection ensures that an SOF bit of a recently enabled ASC module does not collide with an already running frame of another ASC module.

In order to start the idle detection, the user software has to clear bits PSR.RXIDLE and/or PSR.TXIDLE, e.g. before selecting the ASC mode or during operation. If a bit is cleared by software while a data transfer is in progress, the currently running frame transfer is finished normally before starting the idle detection again. Frame reception is only possible if PSR.RXIDLE = 1 and frame transmission is only possible if PSR.TXIDLE = 1. The duration of the idle detection depends on the setting of bit PCR.IDM. In the case that a collision is not possible, the duration can be shortened and the bus can be declared as being idle by setting PCR.IDM = 0.

In the case that the complete idle detection is enabled by PCR.IDM = 1, the data input of DX0 is considered as idle (PSR.RXIDLE becomes set) if a certain number of consecutive passive bit times has been detected. The same scheme applies for the transmitter's data input of DX1. Here, bit PSR.TXIDLE becomes set if the idle condition of this input signal has been detected.

The duration of the complete idle detection is given by the number of programmed data bits per frame plus 2 (in the case without parity) or plus 3 (in the case with parity). The counting of consecutive bit times with 1 level restarts from the beginning each time an edge is found, after leaving a stop mode or if ASC mode becomes enabled.

If the idle detection bits PSR.RXIDLE and/or TXIDLE are cleared by software, the counting scheme is not stopped (no re-start from the beginning). As a result, the cleared bit(s) can become set immediately again if the respective input line still meets the idle criterion.

Please note that the idle time check is based on bit times, so the maximum time can be up to 1 bit time more than programmed value (but not less).

17.3.2.2 Start Bit Detection

The receiver input signal DIN0 (selected signal of input stage DX0) is checked for a falling edge. An SOF bit is detected when a falling edge occurs while the receiver is idle or after the sampling point of the last stop bit. To increase noise immunity, the SOF bit timing starts with the first falling edge that is detected. If the sampled bit value of the SOF is 1, the previous falling edge is considered to be due to noise and the receiver is considered to be idle again.

17.3.2.3 Data Field

The length of the data field (number of data bits) can be programmed by bit field SCTR.FLE. It can vary between 1 and 63 data bits, corresponding to values of SCTR.FLE = 0 to 62 (the value of 63 is reserved and must not be programmed in ASC mode).

The data field can consist of several data words, e.g. a transfer of 12 data bits can be composed of two 8-bit words, with the 12 bits being split into 8-bits of the first word and 4 bits of the second word. The user software has to take care that the transmit data is available in-time, once a frame has been started. If the transmit buffer runs empty during a running data frame, the passive data level (SCTR.PDL) is sent out.

The shift direction can be programmed by SCTR.SDIR. The standard setting for ASC frames with LSB first is achieved with the default setting SDIR = 0.

17.3.2.4 Parity Bit

The ASC allows parity generation for transmission and parity check for reception on frame base. The type of parity can be selected by bit field CCR.PM, common for transmission and reception (no parity, even or odd parity). If the parity handling is disabled, the ASC frame does not contain any parity bit. For consistency reasons, all communication partners have to be programmed to the same parity mode.

After the last data bit of the data field, the transmitter automatically sends out its calculated parity bit if parity generation has been enabled. The receiver interprets this bit as received parity and compares it to its internally calculated one. The received parity bit value and the result of the parity check are monitored in the receiver buffer status registers, RBUFSR and RBUF01SR, as receiver buffer status information. These registers contain bits to monitor a protocol-related argument (PAR) and protocol-related error indication (PERR).

17.3.2.5 Stop Bit(s)

Each ASC frame is completed by 1 or 2 of stop bits with the signal level 1 (same level as the idle level). The number of stop bits is programmable by bit PSR.STPB. A new start bit can be transferred directly after the last stop bit.

17.3.3 Operating the ASC

In order to operate the ASC protocol, the following issues have to be considered:

- Select ASC mode:
  It is recommended to configure all parameters of the ASC that do not change during run time while CCR.MODE = 0000_B. Bit field SCTR.TRM = 01_B has to be programmed. The configuration of the input stages has to be done while CCR.MODE = 0000_B to avoid unintended edges of the input signals and the ASC mode can be enabled by CCR.MODE = 0010_B afterwards.
- Pin connections:
  Establish a connection of input stage DX0 with the selected receive data input pin (signal DIN0) with DX0CR.INSW = 0 and configure a transmit data output pin (signal DOUT0). For collision or idle detection of the transmitter, the input stage DX1 has to be connected to the selected transmit output pin, also with DX1CR.INSW = 0. Additionally, program DX2CR.INSW = 0.
  Due to the handling of the input data stream by the synchronous protocol handler, the propagation delay of the synchronization in the input stage has to be considered.
- Bit timing configuration:
  The desired baud rate setting has to be selected, comprising the fractional divider, the baud rate generator and the bit timing. Please note that not all feature combinations can be supported by the application at the same time, e.g. due to propagation delays. For example, the length of a frame is limited by the frequency difference of the transmitter and the receiver device. Furthermore, in order to use the average of samples (SMD = 1), the sampling point has to be chosen to respect the signal settling and data propagation times.
- Data format configuration:
  The word length, the frame length, and the shift direction have to be set up according to the application requirements by programming the register SCTR. If required by the application, the data input and output signals can be inverted.
  Additionally, the parity mode has to be configured (CCR.PM).

17.3.3.1 Bit Timing

In ASC mode, each bit (incl. protocol bits) is divided into time quanta in order to provide granularity in the sub-bit range to adjust the sample point to the application requirements. The number of time quanta per bit is defined by bit fields BRG.DCTQ and the length of a time quantum is given by BRG.PCTQ.

In the example given in **Figure 17-32**, one bit time is composed of 16 time quanta (BRG.DCTQ = 15). It is not recommended to program less than 4 time quanta per bit time.

Bit field PCR.SP determines the position of the sampling point for the bit value. The value of PCR.SP must not be set to a value greater than BRG.DCTQ. It is possible to sample the bit value only once per bit time or to take the average of samples. Depending on bit

PCR.SMD, either the current input value is directly sampled as bit value, or a majority decision over the input values sampled at the latest three time quanta is taken into account. The standard ASC bit timing consists of 16 time quanta with sampling after 8 or 9 time quanta with majority decision.

The bit timing setup (number of time quanta and the sampling point definition) is common for the transmitter and the receiver. Due to independent bit timing blocks, the receiver and the transmitter can be in different time quanta or bit positions inside their frames. The transmission of a frame is aligned to the time quanta generation.

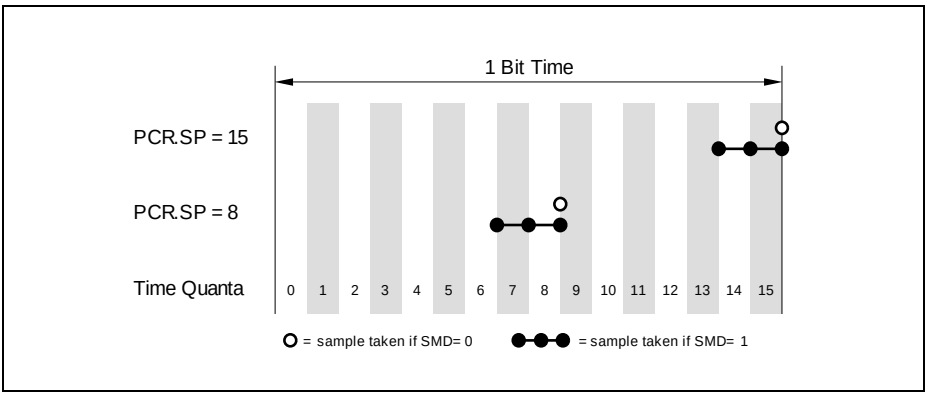


Figure 17-32 ASC Bit Timing

The sample point setting has to be adjusted carefully if collision or idle detection is enabled (via DX1 input signal), because the driver delay and some external delays have to be taken into account. The sample point for the transmit line has to be set to a value where the bit level is stable enough to be evaluated.
If the sample point is located late in the bit time, the signal itself has more time to become stable, but the robustness against differences in the clock frequency of transmitter and receiver decreases.

17.3.3.2 Baud Rate Generation

The baud rate f_{ASC} in ASC mode depends on the number of time quanta per bit time and their timing. The baud rate setting should only be changed while the transmitter and the receiver are idle. The bits in register BRG define the baud rate setting:

- BRG.CTQSEL
  to define the input frequency f_{CTQIN} for the time quanta generation
- BRG.PCTQ
  to define the length of a time quantum (division of f_{CTQIN} by 1, 2, 3, or 4)
- BRG.DCTQ
  to define the number of time quanta per bit time

The standard setting is given by CTQSEL = 00_B ($f_{CTQIN} = f_{PDIV}$) and PPPEN = 0 ($f_{PPP} = f_{PIN}$). Under these conditions, the baud rate is given by:

$$f_{ASC} = f_{PIN} \times \frac{1}{PDIV + 1} \times \frac{1}{PCTQ + 1} \times \frac{1}{DCTQ + 1} \quad (17.6)$$

In order to generate slower frequencies, two additional divide-by-2 stages can be selected by CTQSEL = 10_B ($f_{CTQIN} = f_{SCLK}$) and PPPEN = 1 ($f_{PPP} = f_{MCLK}$), leading to:

$$f_{ASC} = \frac{f_{PIN}}{2 \times 2} \times \frac{1}{PDIV + 1} \times \frac{1}{PCTQ + 1} \times \frac{1}{DCTQ + 1} \quad (17.7)$$

17.3.3.3 Noise Detection

The ASC receiver permanently checks the data input line of the DX0 stage for noise (the check is independent from the setting of bit PCR.SMD). Bit PSR.RNS (receiver noise) becomes set if the three input samples of the majority decision are not identical at the sample point for the bit value. The information about receiver noise gets accumulated over several bits in bit PSR.RNS (it has to be cleared by software) and can trigger a protocol interrupt each time noise is detected if enabled by PCR.RNIEN.

17.3.3.4 Collision Detection

In some applications, such as data transfer over a single data line shared by several sending devices (see **Figure 17-30**), several transmitters have the possibility to send on the same data output line TXD. In order to avoid collisions of transmitters being active at the same time or to allow a kind of arbitration, a collision detection has been implemented.

The data value read at the TXD input at the DX1 stage and the transmitted data bit value are compared after the sampling of each bit value. If enabled by PCR.CDEN = 1 and a bit sent is not equal to the bit read back, a collision is detected and bit PSR.COL is set. If enabled, bit PSR.COL = 1 disables the transmitter (the data output lines become 1) and generates a protocol interrupt. The content of the transmit shift register is considered as invalid, so the transmit buffer has to be programmed again.

17.3.3.5 Pulse Shaping

For some applications, the 0 level of transmitted bits with the bit value 0 is not applied at the transmit output during the complete bit time. Instead of driving the original 0 level, only a 0 pulse is generated and the remaining time quanta of the bit time are driven with 1 level. The length of a bit time is not changed by the pulse shaping, only the signalling is changed.

In the standard ASC signalling scheme, the 0 level is signalled during the complete bit time with bit value 0 (ensured by programming PCR.PL = 000_B). In the case PCR.PL > 000_B, the transmit output signal becomes 0 for the number of time quanta defined by PCR.PL. In order to support correct reception with pulse shaping by the transmitter, the sample point has to be adjusted in the receiver according to the applied pulse length.

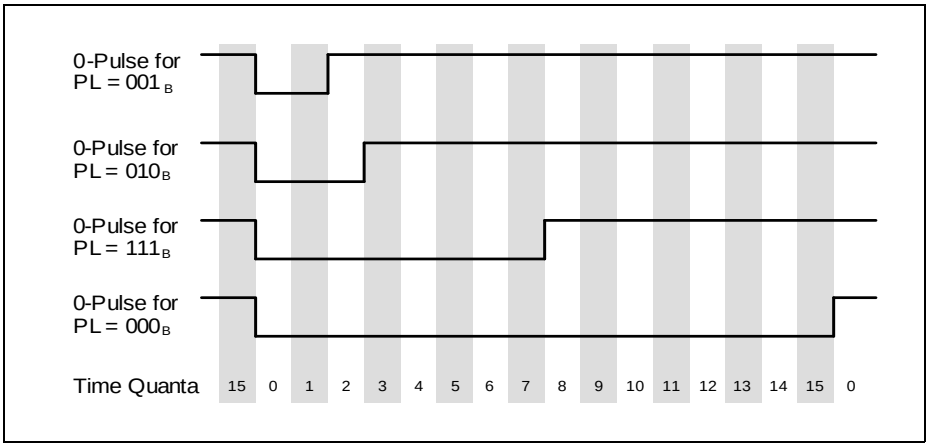


Figure 17-33 Transmitter Pulse Length Control

Figure 17-34 shows an example for the transmission of an 8-bit data word with LSB first and one stop bit (e.g. like for IrDA). The polarity of the transmit output signal has been inverted by SCTR.DOCFG = 01_B.

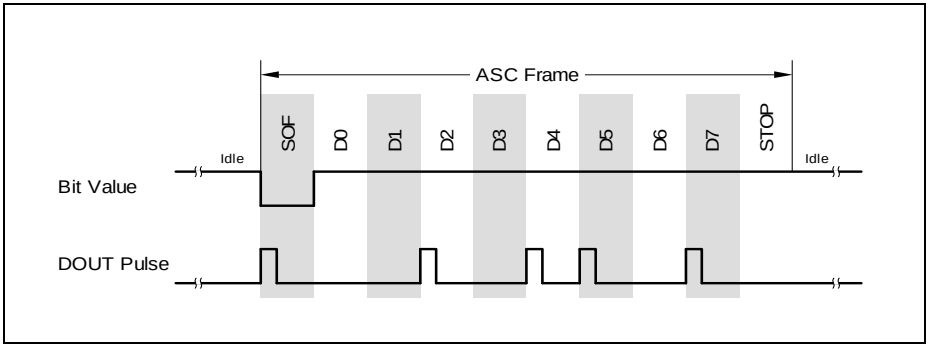


Figure 17-34 Pulse Output Example

17.3.3.6 Automatic Shadow Mechanism

The contents of the protocol control register PCR, as well as bit field SCTR.FLE are internally kept constant while a data frame is transferred by an automatic shadow mechanism (shadowing takes place with each frame start). The registers can be programmed all the time with new settings that are taken into account for the next data frame. During a data frame, the applied (shadowed) setting is not changed, although new values have been written after the start of the data frame.
Bit fields SCTR.WLE and SCTR.SDIR are shadowed automatically with the start of each data word. As a result, a data frame can consist of data words with a different length. It is recommended to change SCTR.SDIR only when no data frame is running to avoid interference between hardware and software.

Please note that the starting point of a data word can be different for a transmitter and a receiver. In order to ensure correct handling, it is recommended to modify SCTR.WLE only while transmitter and receiver are both idle. If the transmitter and the receiver are referring to the same data signal (e.g. in a LIN bus system), SCTR.WLE can be modified while a data transfer is in progress after the RSI event has been detected.

17.3.3.7 End of Frame Control

The number of bits per ASC frame is defined by bit field SCTR.FLE. In order to support different frame length settings for consecutively transmitted frames, this bit field can be modified by hardware. The automatic update mechanism is enabled by TCSR.FLEMD = 1 (in this case, bits TCSR.WLEMD, SELMD, WAMD and HPCMD have to be written with 0).

If enabled, the transmit control information TCI automatically overwrites the bit field TCSR.FLEMD when the ASC frame is started (leading to frames with 1 to 32 data bits). The TCI value represents the written address location of TBUFxx (without additional data buffer) or INxx (with additional data buffer). With this mechanism, an ASC with 8 data bits is generated by writing a data word to TBUF07 (IN07, respectively).

17.3.3.8 Mode Control Behavior

In ASC mode, the following kernel modes are supported:

- Run Mode 0/1:
  Behavior as programmed, no impact on data transfers.
- Stop Mode 0:
  Bit PSR.TXIDLE is cleared. A new transmission is not started. A current transmission is finished normally. Bit PSR.RXIDLE is not modified. Reception is still possible. When leaving stop mode 0, bit TXIDLE is set according to PCR.IDM.
- Stop Mode 1:
  Bit PSR.TXIDLE is cleared. A new transmission is not started. A current transmission is finished normally. Bit PSR.RXIDLE is cleared. A new reception is not possible. A

current reception is finished normally.
When leaving stop mode 1, bits TXIDLE and RXIDLE are set according to PCR.IDM.

17.3.3.9 Disabling ASC Mode

In order to switch off ASC mode without any data corruption, the receiver and the transmitter have to be both idle. This is ensured by requesting Stop Mode 1 in register KSCFG. After waiting for the end of the frame, the ASC mode can be disabled.

17.3.3.10 Protocol Interrupt Events

The following protocol-related events are generated in ASC mode and can lead to a protocol interrupt. The collision detection and the transmitter frame finished events are related to the transmitter, whereas the receiver events are given by the synchronization break detection, the receiver noise detection, the format error checks and the end of the received frame.

Please note that the bits in register PSR are not automatically cleared by hardware and have to be cleared by software in order to monitor new incoming events.

- Collision detection:
  This interrupt indicates that the transmitted value (DOUT0) does not match with the input value of the DX1 input stage at the sample point of a bit. For more details refer to **Page 17-59**.
- Transmitter frame finished:
  This interrupt indicates that the transmitter has completely finished a frame. Bit PSR.TFF becomes set at the end of the last stop bit. The DOUT0 signal assignment to port pins can be changed while no transmission is in progress.
- Receiver frame finished:
  This interrupt indicates that the receiver has completely finished a frame. Bit PSR.RFF becomes set at the end of the last stop bit. The DIN0 signal assignment to port pins can be changed while no reception is in progress.
- Synchronization break detection:
  This interrupt can be used in LIN networks to indicate the reception of the synchronization break symbol (at the beginning of a LIN frame).
- Receiver noise detection:
  This interrupt indicates that the input value at the sample point of a bit and at the two time quanta before are not identical.
- Format error:
  The bit value of the stop bit(s) is defined as 1 level for the ASC protocol. A format error is signalled if the sampled bit value of a stop bit is 0.

17.3.3.11 Data Transfer Interrupt Handling

The data transfer interrupts indicate events related to ASC frame handling.

- Transmit buffer interrupt TBI:
  Bit PSR.TBIF is set after the start of first data bit of a data word. This is the earliest point in time when a new data word can be written to TBUF.
  With this event, bit TCSR.TDV is cleared and new data can be loaded to the transmit buffer.
- Transmit shift interrupt TSI:
  Bit PSR.TSIF is set after the start of the last data bit of a data word.
- Receiver start interrupt RSI:
  Bit PSR.RSIF is set after the sample point of the first data bit of a data word.
- Receiver interrupt RI and alternative interrupt AI:
  Bit PSR.RIF is set after the sampling point of the last data bit of a data word if this data word is not directly followed by a parity bit (parity generation disabled or not the last word of a data frame).
  If the data word is directly followed by a parity bit (last data word of a data frame and parity generation enabled), bit PSR.RIF is set after the sampling point of the parity bit if no parity error has been detected. If a parity error has been detected, bit PSR.AIF is set instead of bit PSR.RIF.
  The first data word of a data frame is indicated by RBUFSR.SOF = 1 for the received word.
  Bit PSR.RIF is set for a receiver interrupt RI with WA = 0. Bit PSR.AIF is set for a alternative interrupt AI with WA = 1.

17.3.3.12 Baud Rate Generator Interrupt Handling

The baud rate generator interrupt indicate that the capture mode timer has reached its maximum value. With this event, the bit PSR.BRGIF is set.

17.3.3.13 Protocol-Related Argument and Error

The protocol-related argument (RBUFSR.PAR) and the protocol-related error (RBUFSR.PERR) are two flags that are assigned to each received data word in the corresponding receiver buffer status registers.

In ASC mode, the received parity bit is monitored by the protocol-related argument and the result of the parity check by the protocol-related error indication (0 = received parity bit equal to calculated parity value). This information being elaborated only for the last received data word of each data frame, both bit positions are 0 for data words that are not the last data word of a data frame or if the parity generation is disabled.

17.3.3.14 Receive Buffer Handling

If a receive FIFO buffer is available (CCFG.RB = 1) and enabled for data handling (RBCTR.SIZE > 0), it is recommended to set RBCTR.RCIM = 11_B in ASC mode. This leads to an indication that the data word has been the first data word of a new data frame

if bit OUTR.RCI[0] = 1, a parity error is indicated by OUTR.RCI[4] = 1, and the received parity bit value is given by OUTR.RCI[3].

The standard receive buffer event and the alternative receive buffer event can be used for the following operations in RCI mode (RBCTR.RNM = 1):

- A standard receive buffer event indicates that a data word can be read from OUTR that has been received without parity error.
- An alternative receive buffer event indicates that a data word can be read from OUTR that has been received with parity error.

17.3.3.15 Sync-Break Detection

The receiver permanently checks the DIN0 signal for a certain number of consecutive bit times with 0 level. The number is given by the number of programmed bits per frame (SCTR.FLE) plus 2 (in the case without parity) or plus 3 (in the case with parity). If a 0 level is detected at a sample point of a bit after this event has been found, bit PSR.SBD is set and additionally, a protocol interrupt can be generated (if enabled by PCR.SBD = 1). The counting restarts from 0 each time a falling edge is found at input DIN0. This feature can be used for the detection of a synchronization break for slave devices in a LIN bus system (the master does not check for sync break).
For example, in a configuration for 8 data bits without parity generation, bit PCR.SBD is set after at the next sample point at 0 level after 10 complete bit times have elapsed (representing the sample point of the 11th bit time since the first falling edge).

17.3.3.16 Transfer Status Indication

The receiver status can be monitored by flag PSR[9] = BUSY if bit PCR.CTR[16] (receiver status enable RSTEN) is set. In this case, bit BUSY is set during a complete frame reception from the beginning of the start of frame bit to the end of the last stop bit. The transmitter status can be monitored by flag PSR[9] = BUSY if bit PCR.CTR[17] (transmitter status enable TSTEN) is set. In this case, bit BUSY is set during a complete frame reception from the beginning of the start of frame bit to the end of the last stop bit. If both bits RSTEN and TSTEN are set, flag BUSY indicates the logical OR-combination of the receiver and the transmitter status. If both bits are cleared, flag BUSY is not modified depending on the transfer status (status changes are ignored).

17.3.4 ASC Protocol Registers

In ASC mode, the registers PCR and PSR handle ASC related information.

17.3.4.1 ASC Protocol Control Register

In ASC mode, the PCR register bits or bit fields are defined as described in this section.

PCR
Protocol Control Register [ASC Mode]

($3C_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MCLK							**0**							**TSTEN**	**RSTEN**
rw							r							rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	PL			**SP**				**FFIEN**	**FEIEN**	**RNIEN**	**CDEN**	**SBIEN**	**IDM**	**STPB**	**SMD**
	rw				rw			rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
SMD	0	rw	**Sample Mode** This bit field defines the sample mode of the ASC receiver. The selected data input signal can be sampled only once per bit time or three times (in consecutive time quanta). When sampling three times, the bit value shifted in the receiver shift register is given by a majority decision among the three sampled values. 0_B Only one sample is taken per bit time. The current input value is sampled. 1_B Three samples are taken per bit time and a majority decision is made.
STPB	1	rw	**Stop Bits** This bit defines the number of stop bits in an ASC frame. 0_B The number of stop bits is 1. 1_B The number of stop bits is 2.

Field	Bits	Type	Description
IDM	2	rw	**Idle Detection Mode** This bit defines if the idle detection is switched off or based on the frame length. 0_B The bus idle detection is switched off and bits PSR.TXIDLE and PSR.RXIDLE are set automatically to enable data transfers without checking the inputs before. 1_B The bus is considered as idle after a number of consecutive passive bit times defined by SCTR.FLE plus 2 (in the case without parity bit) or plus 3 (in the case with parity bit).
SBIEN	3	rw	**Synchronization Break Interrupt Enable** This bit enables the generation of a protocol interrupt if a synchronization break is detected. The automatic detection is always active, so bit SBD can be set independently of SBIEN. 0_B The interrupt generation is disabled. 1_B The interrupt generation is enabled.
CDEN	4	rw	**Collision Detection Enable** This bit enables the reaction of a transmitter to the collision detection. 0_B The collision detection is disabled. 1_B If a collision is detected, the transmitter stops its data transmission, outputs a 1, sets bit PSR.COL and generates a protocol interrupt. In order to allow data transmission again, PSR.COL has to be cleared by software.
RNIEN	5	rw	**Receiver Noise Detection Interrupt Enable** This bit enables the generation of a protocol interrupt if receiver noise is detected. The automatic detection is always active, so bit PSR.RNS can be set independently of PCR.RNIEN. 0_B The interrupt generation is disabled. 1_B The interrupt generation is enabled.

Field	Bits	Type	Description
FEIEN	6	rw	**Format Error Interrupt Enable** This bit enables the generation of a protocol interrupt if a format error is detected. The automatic detection is always active, so bits PSR.FER0/FER1 can be set independently of PCR.FEIEN. 0_B The interrupt generation is disabled. 1_B The interrupt generation is enabled.
FFIEN	7	rw	**Frame Finished Interrupt Enable** This bit enables the generation of a protocol interrupt if the receiver or the transmitter reach the end of a frame. The automatic detection is always active, so bits PSR.RFF or PSR.TFF can be set independently of PCR.FFIEN. 0_B The interrupt generation is disabled. 1_B The interrupt generation is enabled.
SP	[12:8]	rw	**Sample Point** This bit field defines the sample point of the bit value. The sample point must not be located outside the programmed bit timing (PCR.SP ≤ BRG.DCTQ).
PL	[15:13]	rw	**Pulse Length** This bit field defines the length of a 0 data bit, counted in time quanta, starting with the time quantum 0 of each bit time. Each bit value that is a 0 can lead to a 0 pulse that is shorter than a bit time, e.g. for IrDA applications. The length of a bit time is not changed by PL, only the length of the 0 at the output signal. The pulse length must not be longer than the programmed bit timing (PCR.PL ≤ BRG.DCTQ). This bit field is only taken into account by the transmitter and is ignored by the receiver. 000_B The pulse length is equal to the bit length (no shortened 0). 001_B The pulse length of a 0 bit is 2 time quanta. 010_B The pulse length of a 0 bit is 3 time quanta. ... 111_B The pulse length of a 0 bit is 8 time quanta.

Field	Bits	Type	Description
RSTEN	16	rw	**Receiver Status Enable** This bit enables the modification of flag PSR[9] = BUSY according to the receiver status. 0_B Flag PSR[9] is not modified depending on the receiver status. 1_B Flag PSR[9] is set during the complete reception of a frame.
TSTEN	17	rw	**Transmitter Status Enable** This bit enables the modification of flag PSR[9] = BUSY according to the transmitter status. 0_B Flag PSR[9] is not modified depending on the transmitter status. 1_B Flag PSR[9] is set during the complete transmission of a frame.
MCLK	31	rw	**Master Clock Enable** This bit enables the generation of the master clock MCLK. 0_B The MCLK generation is disabled and the MCLK signal is 0. 1_B The MCLK generation is enabled.
0	[30:18]	r	**Reserved** Returns 0 if read; should be written with 0.

17.3.4.2 ASC Protocol Status Register

In ASC mode, the PSR register bits or bit fields are defined as described in this section. The bits and bit fields in register PSR are not cleared by hardware.

The flags in the PSR register can be cleared by writing a 1 to the corresponding bit position in register PSCR. Writing a 1 to a bit position in PSR sets the corresponding flag, but does not lead to further actions (no interrupt generation). Writing a 0 has no effect. The PSR flags should be cleared by software before enabling a new protocol.

PSR
Protocol Status Register [ASC Mode] (48_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															BRGIF
r															rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AIF	RIF	TBIF	TSIF	DLIF	RSIF	BUSY	TFF	RFF	FER1	FER0	RNS	COL	SBD	RXIDLE	TXIDLE
rwh	rwh	rwh	rwh	rwh	rwh	r	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
TXIDLE	0	rwh	**Transmission Idle** This bit shows if the transmit line (DX1) has been idle. A frame transmission can only be started if TXIDLE is set. 0_B The transmitter line has not yet been idle. 1_B The transmitter line has been idle and frame transmission is possible.
RXIDLE	1	rwh	**Reception Idle** This bit shows if the receive line (DX0) has been idle. A frame reception can only be started if RXIDLE is set. 0_B The receiver line has not yet been idle. 1_B The receiver line has been idle and frame reception is possible.
SBD	2	rwh	**Synchronization Break Detected[1)]** This bit is set if a programmed number of consecutive bit values with level 0 has been detected (called synchronization break, e.g. in a LIN bus system). 0_B A synchronization break has not yet been detected. 1_B A synchronization break has been detected.
COL	3	rwh	**Collision Detected[1)]** This bit is set if a collision has been detected (with PCR.CDEN = 1). 0_B A collision has not yet been detected and frame transmission is possible. 1_B A collision has been detected and frame transmission is not possible.

Field	Bits	Type	Description
RNS	4	rwh	**Receiver Noise Detected[1)]** This bit is set if receiver noise has been detected. 0_B Receiver noise has not been detected. 1_B Receiver noise has been detected.
FER0	5	rwh	**Format Error in Stop Bit 0[1)]** This bit is set if a 0 has been sampled in the stop bit 0 (called format error 0). 0_B A format error 0 has not been detected. 1_B A format error 0 has been detected.
FER1	6	rwh	**Format Error in Stop Bit 1[1)]** This bit is set if a 0 has been sampled in the stop bit 1 (called format error 1). 0_B A format error 1 has not been detected. 1_B A format error 1 has been detected.
RFF	7	rwh	**Receive Frame Finished[1)]** This bit is set if the receiver has finished the last stop bit. 0_B The received frame is not yet finished. 1_B The received frame is finished.
TFF	8	rwh	**Transmitter Frame Finished[1)]** This bit is set if the transmitter has finished the last stop bit. 0_B The transmitter frame is not yet finished. 1_B The transmitter frame is finished.
BUSY	9	r	**Transfer Status BUSY** This bit indicates the receiver status (if PCR.RSTEN = 1) or the transmitter status (if PCR.TSTEN = 1) or the logical OR combination of both (if PCR.RSTEN = PCR.TSTEN = 1). 0_B A data transfer does not take place. 1_B A data transfer currently takes place.
RSIF	10	rwh	**Receiver Start Indication Flag** 0_B A receiver start event has not occurred. 1_B A receiver start event has occurred.
DLIF	11	rwh	**Data Lost Indication Flag** 0_B A data lost event has not occurred. 1_B A data lost event has occurred.
TSIF	12	rwh	**Transmit Shift Indication Flag** 0_B A transmit shift event has not occurred. 1_B A transmit shift event has occurred.

Field	Bits	Type	Description
TBIF	13	rwh	**Transmit Buffer Indication Flag** 0_B A transmit buffer event has not occurred. 1_B A transmit buffer event has occurred.
RIF	14	rwh	**Receive Indication Flag** 0_B A receive event has not occurred. 1_B A receive event has occurred.
AIF	15	rwh	**Alternative Receive Indication Flag** 0_B An alternative receive event has not occurred. 1_B An alternative receive event has occurred.
BRGIF	16	rwh	**Baud Rate Generator Indication Flag** 0_B A baud rate generator event has not occurred. 1_B A baud rate generator event has occurred.
0	[31:17]	r	**Reserved** Returns 0 if read; should be written with 0.

1) This status bit can generate a protocol interrupt (see **Page 17-21**). The general interrupt status flags are described in the general interrupt chapter.

17.3.5 Hardware LIN Support

In order to support the LIN protocol, bit TCSR.FLEMD = 1 should be set for the master. For slave devices, it can be cleared and the fixed number of 8 data bits has to be set (SCTR.FLE = 7_H). For both, master and slave devices, the parity generation has to be switched off (CCR.PM = 00_B) and transfers take place with LSB first (SCTR.SDIR = 0) and 1 stop bit (PCR.STPB = 0).

The Local Interconnect Network (LIN) data exchange protocol contains several symbols that can all be handled in ASC mode. Each single LIN symbol represents a complete ASC frame. The LIN bus is a master-slave bus system with a single master and multiple slaves (for the exact definition please refer to the official LIN specification).

A complete LIN frame contains the following symbols:

- Synchronization break:
  The master sends a synchronization break to signal the beginning of a new frame. It contains at least 13 consecutive bit times at 0 level, followed by at least one bit time at 1 level (corresponding to 1 stop bit). Therefore, TBUF11 if the transmit buffer is used, (or IN11 if the FIFO buffer is used) has to be written with 0 (leading to a frame with SOF followed by 12 data bits at 0 level).
  A slave device shall detect 11 consecutive bit times at 0 level, which done by the synchronization break detection. Bit PSR.SBD is set if such an event is detected and a protocol interrupt can be generated. Additionally, the received data value of 0 appears in the receive buffer and a format error is signaled.

If the baud rate of the slave has to be adapted to the master, the baud rate measurement has to be enabled for falling edges by setting BRG.TMEN = 1, DX0CR.CM = 10_H and DX1CR.CM = 00_H before the next symbol starts.

- Synchronization byte:
  The master sends this symbol after writing the data value 55_H to TBUF07 (or IN07). A slave device can either receive this symbol without any further action (and can discard it) or it can use the falling edges for baud rate measurement. Bit PSR.TSIF = 1 (with optionally the corresponding interrupt) indicates the detection of a falling edge and the capturing of the elapsed time since the last falling edge in BRG.PDIV. Valid captured values can be read out after the second, third, fourth and fifth activation of TSIF. After the fifth activation of TSIF within this symbol, the baud rate detection can be disabled (BRG.TMEN = 0) and BRG.PDIV can be programmed with the formerly captured value divided by twice the number of time quanta per bit (assuming BRG.PCTQ = 00_B).
- Other symbols:
  The other symbols of a LIN frame can be handled with ASC data frames without specific actions.

If LIN frames should be sent out on a frame base by the LIN master, the input DX2 can be connected to external timers to trigger the transmit actions (e.g. the synchronization break symbol has been prepared but is started if a trigger occurs). Please note that during the baud rate measurement of the ASC receiver, the ASC transmitter of the same USIC channel can still perform a transmission.

17.4 Synchronous Serial Channel (SSC)

The synchronous serial channel SSC covers the data transfer function of an SPI-like module. It can handle reception and transmission of synchronous data frames between a device operating in master mode and at least one device in slave mode. Besides the standard SSC protocol consisting of one input and one output data line, SSC protocols with two (Dual-SSC) or four (Quad-SSC) input/output data lines are also supported. The SSC mode is selected by CCR.MODE = 0001_B with CCFG.SSC = 1 (SSC mode is available).

17.4.1 Signal Description

A synchronous SSC data transfer is characterized by a simultaneous transfer of a shift clock signal together with the transmit and/or receive data signal(s) to determine when the data is valid (definition of transmit and sample point).

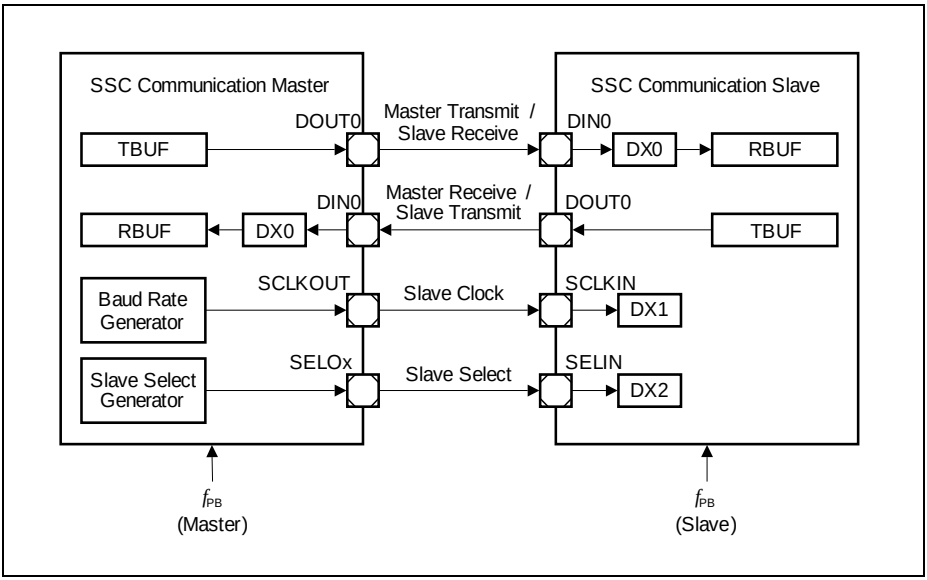


Figure 17-35 SSC Signals for Standard Full-Duplex Communication

In order to explicitly indicate the start and the end of a data transfer and to address more than one slave devices individually, the SSC module supports the handling of slave select signals. They are optional and are not necessarily needed for SSC data transfers. The SSC module supports up to 8 different slave select output signals for master mode operation (named SELOx, with x = 0-7) and 1 slave select input SELIN for slave mode. In most applications, the slave select signals are active low.

A device operating in master mode controls the start and end of a data frame, as well as the generation of the shift clock and slave select signals. This comprises the baud rate setting for the shift clock and the delays between the shift clock and the slave select output signals. If several SSC modules are connected together, there can be only one SSC master at a time, but several slaves. Slave devices receive the shift clock and optionally a slave select signal(s). For the programming of the input stages DXn please refer to **Page 17-21**.

Table 17-11 SSC Communication Signals

SSC Mode	Receive Data	Transmit Data	Shift Clock	Slave Select(s)
Standard SSC Master	MRST[1)], input DIN0, handled by DX0	MTSR[2)], Output DOUT0	Output SCLKOUT	Output(s) SELOx
Standard SSC Slave	MTSR, input DIN0, handled by DX0	MRST, Output DOUT0	Input SCLKIN, handled by DX1	input SELIN, handled by DX2
Dual-SSC Master	MRST[1:0], input DIN[1:0], handled by DX0 and DX3	MTSR[1:0], Output DOUT[1:0]	Output SCLKOUT	Output(s) SELOx
Dual-SSC Slave	MTSR[1:0], input DIN[1:0], handled by DX0 and DX3	MRST[1:0], Output DOUT[1:0]	Input SCLKIN, handled by DX1	input SELIN, handled by DX2
Quad-SSC Master	MRST[3:0], input DIN[3:0], handled by DX0, DX3, DX4 and DX5	MTSR[3:0], Output DOUT[3:0]	Output SCLKOUT	Output(s) SELOx
Quad-SSC Slave	MTSR[3:0], input DIN[3:0], handled by DX0, DX3, DX4 and DX5	MRST[3:0], Output DOUT[3:0]	Input SCLKIN, handled by DX1	input SELIN, handled by DX2

1) MRST = master receive slave transmit, also known as MISO = master in slave out

2) MTSR = master transmit slave receive, also known as MOSI = master out slave in

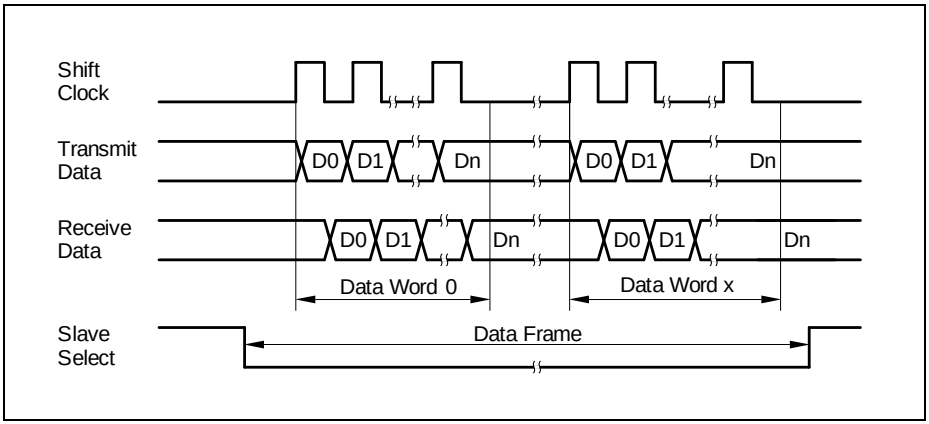


Figure 17-36 4-Wire SSC Standard Communication Signals

17.4.1.1 Transmit and Receive Data Signals

In standard SSC half-duplex mode, a single data line is used, either for data transfer from the master to a slave or from a slave to the master. In this case, MRST and MTSR are connected together, one signal as input, the other one as output, depending on the data direction. The user software has to take care about the data direction to avoid data collision (e.g. by preparing dummy data of all 1s for transmission in case of a wired AND connection with open-drain drivers, by enabling/disabling push/pull output drivers or by switching pin direction with hardware port control enabled). In full-duplex mode, data transfers take place in parallel between the master device and a slave device via two independent data signals MTSR and MRST, as shown in **Figure 17-35**.

The receive data input signal DIN0 is handled by the input stage DX0. In master mode (referring to MRST) as well as in slave mode (referring to MTSR), the data input signal DIN0 is taken from an input pin. The signal polarity of DOUT0 (data output) with respect to the data bit value can be configured in block DOCFG (data output configuration) by bit field SCTR.DOCFG.

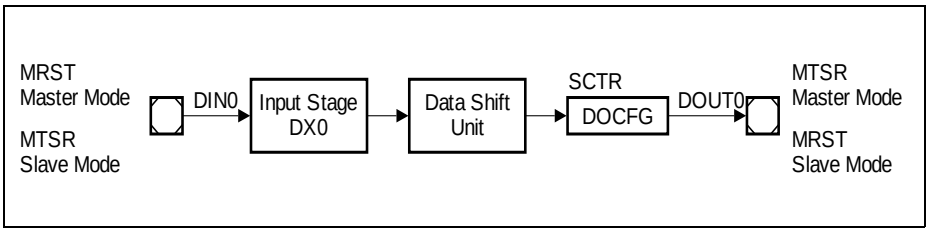


Figure 17-37 SSC Data Signals

For dual- and quad-SSC modes that require multiple input and output data lines to be used, additional input stages, DINx and DOUTx signals need to be set up.

17.4.1.2 Shift Clock Signals

The shift clock signal is handled by the input stage DX1. In slave mode, the signal SCLKIN is received from an external master, so the DX1 stage has to be connected to an input pin. The input stage can invert the received input signal to adapt to the polarity of SCLKIN to the function of the data shift unit (data transmission on rising edges, data reception on falling edges).

In master mode, the shift clock is generated by the internal baud rate generator. The output signal SCLK of the baud rate generator is taken as shift clock input for the data shift unit. The internal signal SCLK is made available for external slave devices by signal SCLKOUT. For complete closed loop delay compensation in a slave mode, SCLKOUT can also take the transmit shift clock from the input stage DX1. The selection is done through the bit BRG.SCLKOSEL. See **Section 17.4.6.3**.

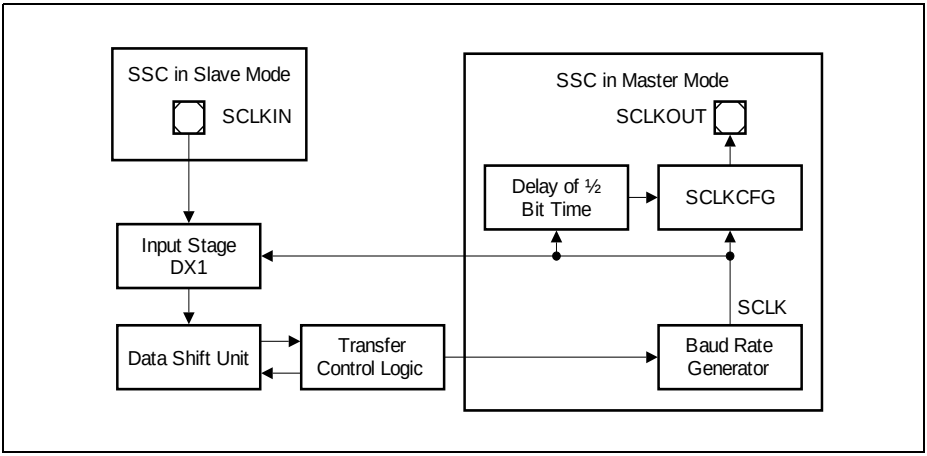


Figure 17-38 SSC Shift Clock Signals

Due to the multitude of different SSC applications, in master mode, there are different ways to configure the shift clock output signal SCLKOUT with respect to SCLK. This is done in the block SCLKCFG (shift clock configuration) by bit field BRG.SCLKCFG, allowing 4 possible settings, as shown in **Figure 17-39**.

- No delay, no polarity inversion (SCLKCFG = 00_B, SCLKOUT equals SCLK):
  The inactive level of SCLKOUT is 0, while no data frame is transferred. The first data bit of a new data frame is transmitted with the first rising edge of SCLKOUT and the first data bit is received in with the first falling edge of SCLKOUT. The last data bit of a data frame is transmitted with the last rising clock edge of SCLKOUT and the last

data bit is received in with the last falling edge of SCLKOUT. This setting can be used in master and in slave mode. It corresponds to the behavior of the internal data shift unit.

- No delay, polarity inversion (SCLKCFG = 01_B):
  The inactive level of SCLKOUT is 1, while no data frame is transferred. The first data bit of a new data frame is transmitted with the first falling clock edge of SCLKOUT and the first data bit is received with the first rising edge of SCLKOUT. The last data bit of a data frame is transmitted with the last falling edge of SCLKOUT and the last data bit is received with the last rising edge of SCLKOUT. This setting can be used in master and in slave mode.
- SCLKOUT is delayed by 1/2 shift clock period, no polarity inversion (SCLKCFG = 10_B):
  The inactive level of SCLKOUT is 0, while no data frame is transferred.
  The first data bit of a new data frame is transmitted 1/2 shift clock period before the first rising clock edge of SCLKOUT. Due to the delay, the next data bits seem to be transmitted with the falling edges of SCLKOUT. The last data bit of a data frame is transmitted 1/2 period of SCLKOUT before the last rising clock edge of SCLKOUT. The first data bit is received 1/2 shift clock period before the first falling edge of SCLKOUT. Due to the delay, the next data bits seem to be received with the rising edges of SCLKOUT. The last data bit is received 1/2 period of SCLKOUT before the last falling clock edge of SCLKOUT.
  This setting can be used only in master mode and not in slave mode (the connected slave has to provide the first data bit before the first SCLKOUT edge, e.g. as soon as it is addressed by its slave select).
- SCLKOUT is delayed by 1/2 shift clock period, polarity inversion (SCLKCFG = 11_B):
  The inactive level of SCLKOUT is 1, while no data frame is transferred.
  The first data bit of a new data frame is transmitted 1/2 shift clock period before the first falling clock edge of SCLKOUT. Due to the delay, the next data bits seem to be transmitted with the rising edges of SCLKOUT. The last data bit of a data frame is transmitted 1/2 period of SCLKOUT before the last falling clock edge of SCLKOUT. The first data bit is received 1/2 shift clock period before the first rising edge of SCLKOUT. Due to the delay, the next data bits seem to be received with the falling edges of SCLKOUT. The last data bit is received 1/2 period of SCLKOUT before the last rising clock edge of SCLKOUT.
  This setting can be used only in master mode and not in slave mode (the connected slave has to provide the first data bit before the first SCLKOUT edge, e.g. as soon as it is addressed by its slave select).

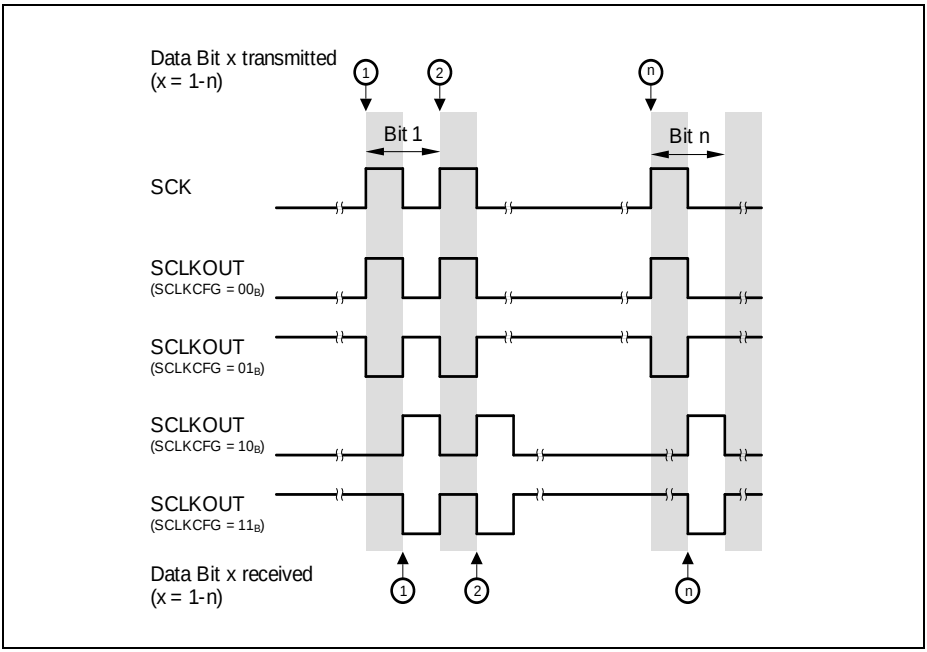


Figure 17-39 SCLKOUT Configuration in SSC Master Mode

Note: If a configuration with delay is selected and a slave select line is used, the slave select delays have to be set up accordingly.

17.4.1.3 Slave Select Signals

The slave select signal is handled by the input stage DX2. In slave mode, the input signal SELIN is received from an external master via an input pin. The input stage can invert the received input signal to adapt the polarity of signal SELIN to the function of the data shift unit (the module internal signals are considered as high active, so a data transfer is only possible while the slave select input of the data shift unit is at 1-level, otherwise, shift clock pulses are ignored and do not lead to data transfers). If an input signal SELIN is low active, it should be inverted in the DX2 input stage.

In master mode, a master slave select signal MSLS is generated by the internal slave select generator. In order to address different external slave devices independently, the internal MSLS signal is made available externally via up to 8 SELOx output signals that can be configured by the block SELCFG (select configuration).

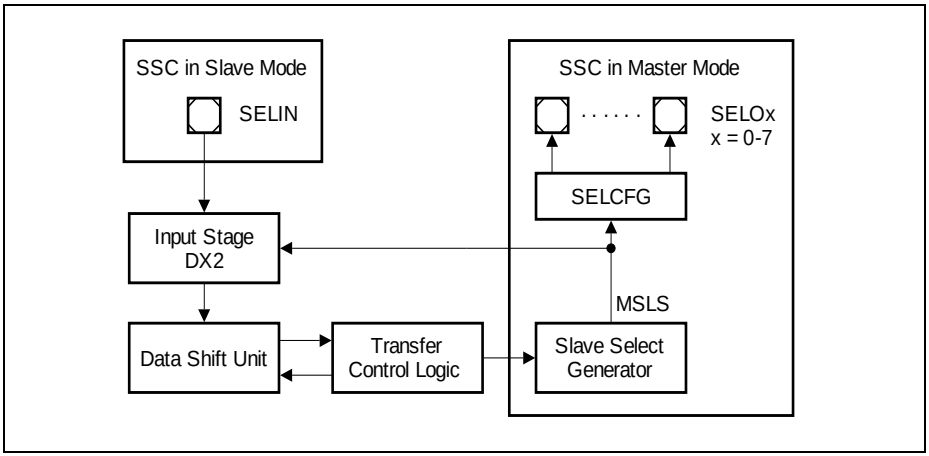


Figure 17-40 SSC Slave Select Signals

The control of the SELCFG block is based on protocol specific bits and bit fields in the protocol control register PCR. For the generation of the MSLS signal please refer to **Section 17.4.3.2**.

- PCR.SELCTR to chose between direct and coded select mode
- PCR.SELINV to invert the SELOx outputs
- PCR.SELO[7:0] as individual value for each SELOx line

The SELCFG block supports the following configurations of the SELOx output signals:

- Direct Select Mode (SELCTR = 1):
  Each SELOx line (with x = 0-7) can be directly connected to an external slave device. If bit x in bit field SELO is 0, the SELOx output is permanently inactive. A SELOx output becomes active while the internal signal MSLS is active (see **Section 17.4.3.2**) and bit x in bit field SELO is 1. Several external slave devices can be addressed in parallel if more than one bit in bit field SELO are set during a data frame. The number of external slave devices that can be addressed individually is limited to the number of available SELOx outputs.
- Coded Select Mode (SELCTR = 0):
  The SELOx lines (with x = 1-7) can be used as addresses for an external address decoder to increase the number of external slave devices. These lines only change with the start of a new data frame and have no other relation to MSLS. Signal SELO0 can be used as enable signal for the external address decoder. It is active while MSLS is active (during a data frame) and bit 0 in bit field SELO is 1. Furthermore, in coded select mode, this output line is delayed by one cycle of f_{PB} compared to MSLS to allow the other SELOx lines to stabilize before enabling the address decoder.

17.4.2 Operating the SSC

This chapter contains SSC issues, that are of general interest and not directly linked to either master mode or slave mode.

17.4.2.1 Automatic Shadow Mechanism

The contents of the baud rate control register BRG, bit fields SCTR.FLE as well as the protocol control register PCR are internally kept constant while a data frame is transferred (= while MSLS is active) by an automatic shadow mechanism. The registers can be programmed all the time with new settings that are taken into account for the next data frame. During a data frame, the applied (shadowed) setting is not changed, although new values have been written after the start of the data frame.

Bit fields SCTR.WLE, SCTR.DSM, SCTR.HPCDIR and SCTR.SDIR are shadowed automatically with the start of each data word. As a result, a data frame can consist of data words with a different length, or data words that are transmitted or received through different number of data lines. It is recommended to change SCTR.SDIR only when no data frame is running to avoid interference between hardware and software.

Please note that the starting point of a data word are different for a transmitter (first bit transmitted) and a receiver (first bit received). In order to ensure correct handling, it is recommended to refer to the receive start interrupt RSI before modifying SCTR.WLE. If TCSR.WLEMD = 1, it is recommended to update TCSR and TBUFxx after the receiver start interrupt has been generated.

17.4.2.2 Mode Control Behavior

In SSC mode, the following kernel modes are supported:

- Run Mode 0/1:
  Behavior as programmed, no impact on data transfers.
- Stop Mode 0/1:
  The content of the transmit buffer is considered as not valid for transmission. Although being considered as 0, bit TCSR.TDV it is not modified by the stop mode condition.
  In master mode, a currently running word transfer is finished normally, but no new data word is started (the stop condition is not considered as end-of-frame condition).
  In slave mode, a currently running word transfer is finished normally. Passive data will be sent out instead of a valid data word if a data word transfer is started by the external master while the slave device is in stop mode. In order to avoid passive slave transmit data, it is recommended not to program stop mode for an SSC slave device if the master device does not respect the slave device's stop mode.

17.4.2.3 Disabling SSC Mode

In order to disable SSC mode without any data corruption, the receiver and the transmitter have to be both idle. This is ensured by requesting Stop Mode 1 in register KSCFG. After Stop Mode 1 has been acknowledged by KSCFG.2 = 1, the SSC mode can be disabled.

17.4.2.4 Data Frame Control

An SSC data frame can consist of several consecutive data words that may be separated by an inter-word delay. Without inter-word delay, the data words seem to form a longer data word, being equivalent to a data frame. The length of the data words are most commonly identical within a data frame, but may also differ from one word to another. The data word length information (defined by SCTR.WLE) is evaluated for each new data word, whereas the frame length information (defined by SCTR.FLE) is evaluated at the beginning at each start of a new frame.
The length of an SSC data frame can be defined in two different ways:

- By the number of bits per frame:
  If the number of bits per data frame is defined (frame length FLE), a slave select signal is not necessarily required to indicate the start and the end of a data frame.
  If the programmed number of bits per frame is reached within a data word, the frame is considered as finished and remaining data bits in the last data word are ignored and are not transferred.
  This method can be applied for data frames with up to 63 data bits.
- By the slave select signal:
  If the number of bits per data frame is not known, the start/end information of a data frame is given by a slave select signal. If a deactivation of the slave select signal is detected within a data word, the frame is considered as finished and remaining data bits in the last data word are ignored and are not transferred.
  This method has to be applied for frames with more than 63 data bits (programming limit of FLE). The advantage of slave select signals is the clearly defined start and end condition of data frames in a data stream. Furthermore, slave select signals allow to address slave devices individually.

17.4.2.5 Parity Mode

The SSC allows parity generation for transmission and parity check for reception on frame base. The type of parity can be selected by bit field CCR.PM, common for transmission and reception (no parity, even or odd parity). If the parity handling is disabled, the SSC frame does not contain any parity bit. For consistency reasons, all communication partners have to be programmed to the same parity mode.

If parity generation has been enabled, the transmitter automatically extends the clock by one cycle after the last data word of the data frame, and sends out its calculated parity bit in this cycle.

Figure 17-41 shows how a parity bit is added to the transmitted data bits of a frame. The number of the transmitted bits of a complete frame with parity is always one more than that without parity. The parity bit is transmitted as the last bit of a frame, following the data bits, independent of the shift direction (SCTR.SDIR).

Note: For dual and quad SSC protocols, the parity bit will be transmitted and received only on DOUT0 and DX0 respectively in the extended clock cycle.

Figure 17-41 Data Frames without/with Parity

Similarly, after the receiver receives the last word of a data frame as defined by FLE, it expects an additional one clock cycle, which will contain the parity bit. The receiver interprets this bit as received parity and separates it from the received data. The received parity bit value is instead monitored in the protocol-related argument (PAR) of the receiver buffer status registers as receiver buffer status information. The receiver compares the bit to its internally calculated parity and the result of the parity check is indicated by the flag PSR.PARERR. The parity error event generates a protocol interrupt if PCR.PARIEN = 1.

Parity bit generation and detection is not supported for the following cases:

- When frame length is 64 data bits or greater, i.e. FLE = 63_H;
- When in slave mode, the end of frame occurs before the number of data bits defined by FLE is reached.

17.4.2.6 Transfer Mode

In SSC mode, bit field SCTR.TRM = 01_B has to be programmed to allow data transfers. Setting SCTR.TRM = 00_B disables and stops the data transfer immediately.

17.4.2.7 Data Transfer Interrupt Handling

The data transfer interrupts indicate events related to SSC frame handling.

- Transmit buffer interrupt TBI:
  Bit PSR.TBIF is set after the start of first data bit of a data word.
- Transmit shift interrupt TSI:
  Bit PSR.TSIF is set after the start of the last data bit of a data word.
- Receiver start interrupt RSI:
  Bit PSR.RSIF is set after the reception of the first data bit of a data word.
  With this event, bit TCSR.TDV is cleared and new data can be loaded to the transmit buffer.
- Receiver interrupt RI:
  The reception of the second, third, and all subsequent words in a multi-word frame is always indicated by RBUFSR.SOF = 0. Bit PSR.RIF is set after the reception of the last data bit of a data word if RBUFSR.SOF = 0.
  Bit RBUFSR.SOF indicates whether the received data word has been the first data word of a multi-word frame or some subsequent word. In SSC mode, it decides if alternative interrupt or receive interrupt is generated.
- Alternative interrupt AI:
  The reception of the first word in a frame is always indicated by RBUFSR.SOF = 1. This is true both in case of reception of multi-word frames and single-word frames. In SSC mode, this results in setting PSR.AIF.

17.4.2.8 Baud Rate Generator Interrupt Handling

The baud rate generator interrupt indicate that the capture mode timer has reached its maximum value. With this event, the bit PSR.BRGIF is set.

17.4.2.9 Protocol-Related Argument and Error

The protocol-related argument (RBUFSR.PAR) and the protocol-related error (RBUFSR.PERR) are two flags that are assigned to each received data word in the corresponding receiver buffer status registers.

In SSC mode, the received parity bit is monitored by the protocol-related argument. The received start of frame indication is monitored by the protocol-related error indication (0 = received word is not the first word of a frame, 1 = received word is the first word of a new frame).

Note: For SSC, the parity error event indication bit is located in the PSR register.

17.4.2.10 Receive Buffer Handling

If a receive FIFO buffer is available (CCFG.RB = 1) and enabled for data handling (RBCTR.SIZE > 0), it is recommended to set RBCTR.RCIM = 01_B in SSC mode. This leads to an indication that the data word has been the first data word of a new data frame if bit OUTR.RCI[4] = 1, and the word length of the received data is given by OUTR.RCI[3:0].

The standard receive buffer event and the alternative receive buffer event can be used for the following operation in RCI mode (RBCTR.RNM = 1):

- A standard receive buffer event indicates that a data word can be read from OUTR that has not been the first word of a data frame.
- An alternative receive buffer event indicates that the first data word of a new data frame can be read from OUTR.

17.4.2.11 Multi-IO SSC Protocols

The SSC implements the following three features to support multiple data input/output SSC protocols, such as the dual- and quad-SSC:

1. Data Shift Mode (**Section 17.2.5.2**)
   Configures the data for transmission and reception using one, two or four data lines in parallel, through the bit field SCTR.DSM.
2. Hardware Port Control (**Section 17.2.7**)
   Sets up a dedicated hardware interface to control the direction of the pins overlaid with both DINx and DOUTx functions, through the bit SCTR.HPCDIR.
3. Transmit Control Information (**Section 17.2.5.3**)
   Allows the dynamic control of both the shift mode and pin direction during data transfers by writing to SCTR.DSM and SCTR.HPCDIR with TCI.

Figure 17-42 shows an example of a Quad-SSC protocol, which requires the master SSC to first transmit a command byte (to request a quad output read from the slave) and a dummy byte through a single data line. At the end of the dummy byte, both master and slave SSC switches to quad data lines, and with the roles of transmitter and receiver reversed. The master SSC then receives the data four bits per shift clock from the slave through the MRST[3:0] lines.

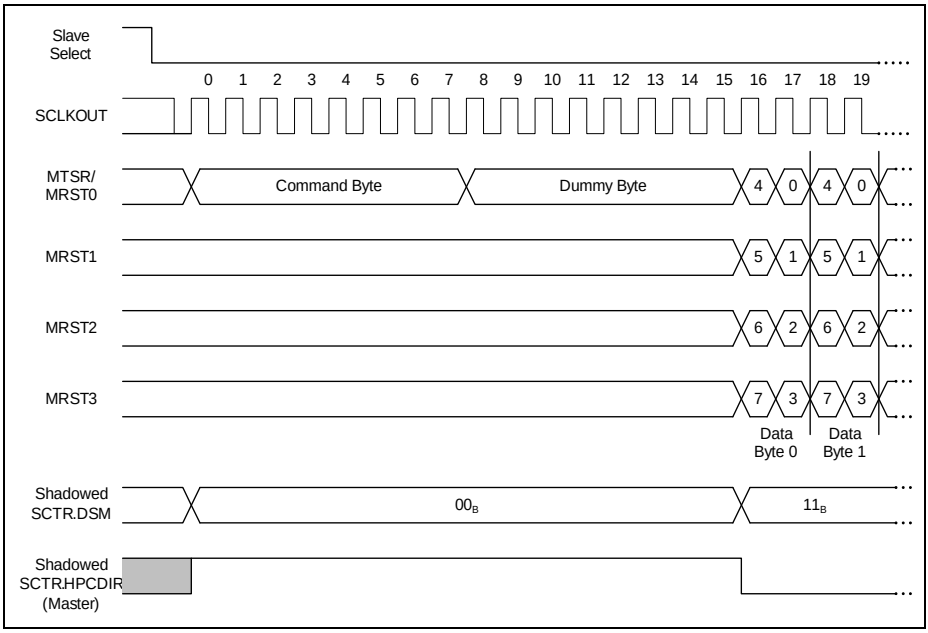


Figure 17-42 Quad-SSC Example

To work with the quad-SSC protocol in the given example, the following issues have to be additionally considered on top of those defined in **Section 17.4.3** and **Section 17.4.4**:

- During the initialization phase:
  - Set CCR.HPCEN to 11_B to enable the dedicated hardware interface to the DX0/DOUT0, DX3/DOUT1, DX4/DOUT2 and DX5/DOUT3 pins.
  - Set TCSR.[4:0] to 10_H to enable hardware port control in TCI
- To start the data transfer:
  - For the master SSC, write the command and dummy bytes into TBUF04 to select a single data line in output mode and initiate the data transfer.
  - For the slave SSC, dummy data can be preloaded into TBUF00 to select a single data line in input mode.
- To switch to quad data lines and pin direction:

– For the master SSC, write subsequent dummy data to TBUF03 to select dual data lines in input mode to read in valid slave data.
– For the slave SSC, write valid data to TBUF07 for transmission through dual data lines in output mode.

Figure 17-43 shows the connections for the Quad-SSC example.

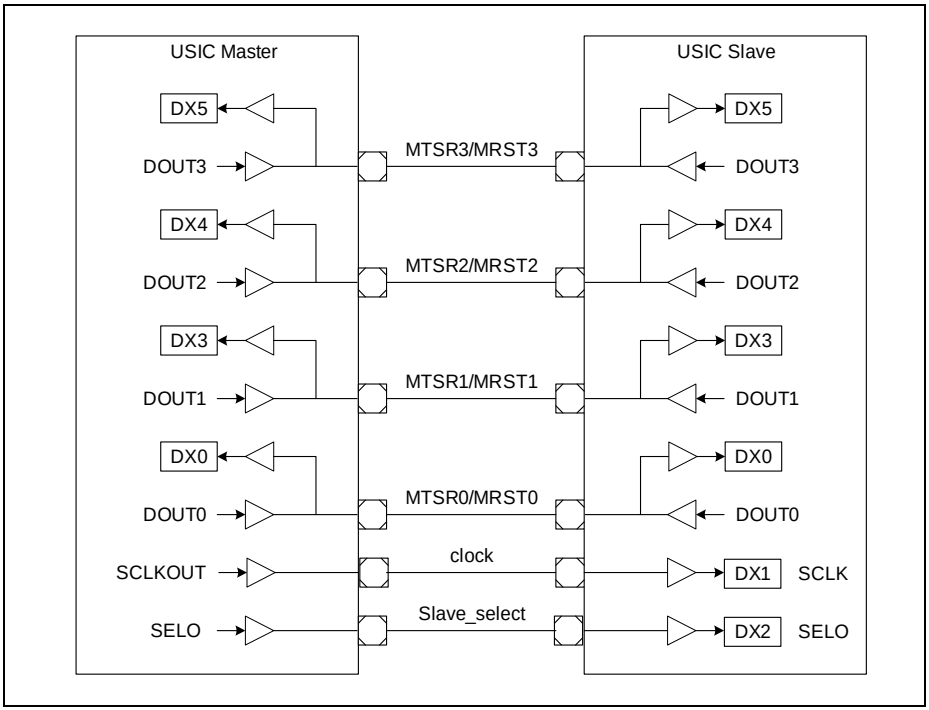


Figure 17-43 Connections for Quad-SSC Example

17.4.3 Operating the SSC in Master Mode

In order to operate the SSC in master mode, the following issues have to be considered:

- Select SSC mode:
  It is recommended to configure all parameters of the SSC that do not change during run time while CCR.MODE = 0000_B. Bit field SCTR.TRM = 01_B has to be programmed. The configuration of the input stages has to be done while CCR.MODE = 0000_B to avoid unintended edges of the input signals and the SSC mode can be enabled by CCR.MODE = 0001_B afterwards.
- Pin connections:
  Establish a connection of the input stage (DX0, DX3, DX4, DX5) with the selected

receive data input pin (DIN[3:0]) with DXnCR.INSW = 1 and configure the transmit data output pin (DOUT[3:0]). One, two or four such connections may be needed depending on the protocol. For half-duplex configurations, hardware port control can be also used to establish the required connections.

- Baud rate generation:
  The desired baud rate setting has to be selected, comprising the fractional divider and the baud rate generator. Bit DX1CR.INSW = 0 has to be programmed to use the baud rate generator output SCLK directly as input for the data shift unit. Configure a shift clock output pin (signal SCLKOUT).
- Slave select generation:
  The slave select delay generation has to be enabled by setting PCR.MSLSEN = 1 and the programming of the time quanta counter setting. Bit DX2CR.INSW = 0 has to be programmed to use the slave select generator output MSLS as input for the data shift unit. Configure slave select output pins (signals SELOx) if needed.
- Data format configuration:
  The word length, the frame length, the shift direction and shift mode have to be set up according to the application requirements by programming the register SCTR.

Note: The USIC can only receive in master mode if it is transmitting, because the master frame handling refers to bit TDV of the transmitter part.

17.4.3.1 Baud Rate Generation

The baud rate (determining the length of one data bit) of the SSC is defined by the frequency of the SCLK signal (one period of f_{SCLK} represents one data bit). The SSC baud rate generation does not imply any time quanta counter.

In a standard SSC application, the phase relation between the optional MCLK output signal and SCLK is not relevant and can be disabled (BRG.PPPEN = 0). In this case, the SCLK signal directly derives from the protocol input frequency f_{PIN}. In the exceptional case that a fixed phase relation between the MCLK signal and SCLK is required (e.g. when using MCLK as clock reference for external devices), the additional divider by 2 stage has to be taken into account (BRG.PPPEN = 1).

The adjustable divider factor is defined by bit field BRG.PDIV.

$$f_{SCLK} = \frac{f_{PIN}}{2} \times \frac{1}{PDIV + 1} \quad \text{if PPPEN = 0}$$
$$f_{SCLK} = \frac{f_{PIN}}{2 \times 2} \times \frac{1}{PDIV + 1} \quad \text{if PPPEN = 1} \qquad (17.8)$$

17.4.3.2 MSLS Generation

The slave select signals indicate the start and the end of a data frame and are also used by the communication master to individually select the desired slave device. A slave

select output of the communication master becomes active a programmable time before a data part of the frame is started (leading delay T_{ld}), necessary to prepare the slave device for the following communication. After the transfer of a data part of the frame, it becomes inactive again a programmable time after the end of the last bit (trailing delay T_{td}) to respect the slave hold time requirements. If data frames are transferred back-to-back one after the other, the minimum time between the deactivation of the slave select and the next activation of a slave select is programmable (next-frame delay T_{nf}). If a data frame consists of more than one data word, an optional delay between the data words can also be programmed (inter-word delay T_{iw}).

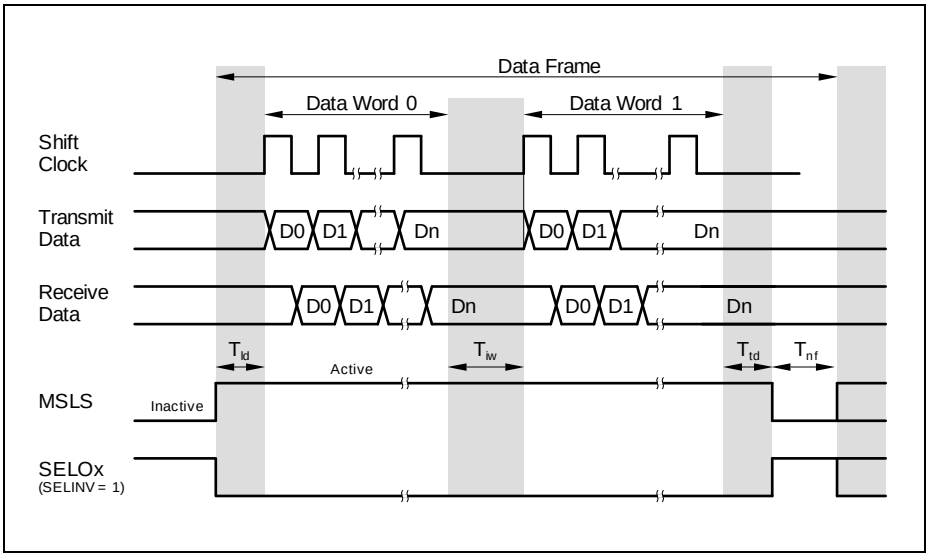


Figure 17-44 MSLS Generation in SSC Master Mode

In SSC master mode, the slave select delays are defined as follows:

- Leading delay T_{ld}:
  The leading delay starts if valid data is available for transmission. The internal signal MSLS becomes active with the start of the leading delay. The first shift clock edge (rising edge) of SCLK is generated by the baud rate generator after the leading delay has elapsed.
- Trailing delay T_{td}
  The trailing delay starts at the end of the last SCLK cycle of a data frame. The internal signal MSLS becomes inactive with the end of the trailing delay.
- Inter-word delay T_{iw}:
  This delay is optional and can be enabled/disabled by PCR.TIWEN. If the inter-word delay is disabled (TIWEN = 0), the last data bit of a data word is directly followed by

the first data bit of the next data word of the same data frame. If enabled (TIWEN = 1), the inter-word delay starts at the end of the last SCLK cycle of a data word. The first SCLK cycle of the following data word of the same data frame is started when the inter-word delay has elapsed. During this time, no shift clock pulses are generated and signal MSLS stays active. The communication partner has time to “digest” the previous data word or to prepare for the next one.

- Next-frame delay T_{nf}:
  The next-frame delay starts at the end of the trailing delay. During this time, no shift clock pulses are generated and signal MSLS stays inactive. A frame is considered as finished after the next-frame delay has elapsed.

17.4.3.3 Automatic Slave Select Update

If the number of bits per SSC frame and the word length are defined by bit fields SCTR.FLE and SCTR.WLE, the transmit control information TCI can be used to update the slave select setting PCR.CTR[23:16] to control the SELOx select outputs. The automatic update mechanism is enabled by TCSR.SELMD = 1 (bits TCSR.WLEMD, FLEMD, and WAMD have to be cleared). In this case, the TCI of the first data word of a frame defines the slave select setting of the complete frame due to the automatic shadow mechanism (see **Page 17-61**).

17.4.3.4 Slave Select Delay Generation

The slave select delay generation is based on time quanta. The length of a time quantum (defined by the period of the f_{CTQIN}) and the number of time quanta per delay can be programmed.

In standard SSC applications, the leading delay T_{ld} and the trailing delay T_{td} are mainly used to ensure stability on the input and output lines as well as to respect setup and hold times of the input stages. These two delays have the same length (in most cases shorter than a bit time) and can be programmed with the same set of bit fields.

- BRG.CTQSEL
  to define the input frequency f_{CTQIN} for the time quanta generation for T_{ld} and T_{td}
- BRG.PCTQ
  to define the length of a time quantum (division of f_{CTQIN} by 1, 2, 3, or 4) for T_{ld} and T_{td}
- BRG.DCTQ
  to define the number of time quanta for the delay generation for T_{ld} and T_{td}

The inter-word delay T_{iw} and the next-frame delay T_{nf} are used to handle received data or to prepare data for the next word or frame. These two delays have the same length (in most cases in the bit time range) and can be programmed with a second, independent set of bit fields.

- PCR.CTQSEL1
  to define the input frequency f_{CTQIN} for the time quanta generation for T_{nf} and T_{iw}
- PCR.PCTQ1
  to define the length of a time quantum (division of f_{CTQIN} by 1, 2, 3, or 4) for T_{nf} and T_{iw}
- PCR.DCTQ1
  to define the number of time quanta for the delay generation for T_{nf} and T_{iw}
- PCR.TIWEN
  to enable/disable the inter-word delay T_{iw}

Each delay depends on the length of a time quantum and the programmed number of time quanta given by the bit fields CTQSEL/CTQSEL1, PCTQ/DCTQ and PCTQ1/DCTQ1 (the coding of CTQSEL1 is similar to CTQSEL, etc.). To provide a high flexibility in programming the delay length, the input frequencies can be selected between several possibilities (e.g. based on bit times or on the faster inputs of the protocol-related divider). The delay times are defined as follows:

$$T_{ld} = T_{td} = \frac{(PCTQ + 1) \times (DCTQ + 1)}{f_{CTQIN}}$$
$$T_{iw} = T_{nf} = \frac{(PCTQ1 + 1) \times (DCTQ1 + 1)}{f_{CTQIN}} \quad (17.9)$$

17.4.3.5 Protocol Interrupt Events

The following protocol-related events generated in SSC mode and can lead to a protocol interrupt. They are related to the start and the end of a data frame. After the start of a data frame a new setting could be programmed for the next data frame and after the end of a data frame the SSC connections to pins can be changed.

Please note that the bits in register PSR are not all automatically cleared by hardware and have to be cleared by software in order to monitor new incoming events.

- MSLS Interrupt:
  This interrupt indicates in master mode (MSLS generation enabled) that a data frame has started (activation of MSLS) and has been finished (deactivation of MSLS). Any change of the internal MSLS signal sets bit PSR.MSLSEV and additionally, a protocol interrupt can be generated if PCR.MSLSIEN = 1. The actual state of the internal MSLS signal can be read out at PSR.MSLS to take appropriate actions when this interrupt has been detected.
- DX2T Interrupt:
  This interrupt monitors edges of the input signal of the DX2 stage (although this signal is not used as slave select input for data transfers).
  A programmable edge detection for the DX2 input signal sets bit PSR.DX2TEV and additionally, a protocol interrupt can be generated if PCR.DX2TIEN = 1. The actual state of the selected input signal can be read out at PSR.DX2S to take appropriate actions when this interrupt has been detected.
- Parity Error Interrupt:
  This interrupt indicates that there is a mismatch in the received parity bit (in RBUFSR.PAR) with the calculated parity bit of the last received word of a data frame.

17.4.3.6 End-of-Frame Control

The information about the frame length is required for the MSLS generator of the master device. In addition to the mechanism based on the number of bits per frame (selected with SCTR.FLE < 63), the following alternative mechanisms for end of frame handling are supported. It is recommended to set SCTRFLE = 63 (if several end of frame mechanisms are activated in parallel, the first end condition being found finishes the frame).

- Software-based start of frame indication TCSR.SOF:
  This mechanism can be used if software handles the TBUF data without data FIFO. If bit SOF is set, a valid content of TBUF is considered as first word of a new frame. Bit SOF has to be set before the content of TBUF is transferred to the transmit shift register, so it is recommended to write it before writing data to TBUF. A current data word transfer is finished completely and the slave select delays T_{td} and T_{nf} are applied before starting a new data frame with T_{ld} and the content of TBUF.
  For software-handling of bit SOF, bit TCSR.WLEMD = 0 has to be programmed. In this case, all TBUF[31:0] address locations show an identical behavior (TCI not taken into account for data handling).
- Software-based end of frame indication TCSR.EOF:
  This mechanism can be used if software handles the TBUF data without data FIFO. If bit EOF is set, a valid content of TBUF is considered as last word of a new frame. Bit EOF has to be set before the content of TBUF is transferred to the transmit shift register, so it is recommended to write it before writing data to TBUF. The data word in TBUF is sent out completely and the slave select delays T_{td} and T_{nf} are applied. A new data frame can start with T_{ld} with the next valid TBUF value.
  For software-handling of bit EOF, bit TCSR.WLEMD = 0 has to be programmed. In this case, all TBUF[31:0] address locations show an identical behavior (TCI not taken into account for data handling).
- Software-based address related end of frame handling:
  This mechanism can be used if software handles the TBUF data without data FIFO. If bit TCSR.WLEMD = 1, the address of the written TBUF[31:0] is used as transmit control information TCI[4:0] to update SCTR.WLE (= TCI[3:0]) and TCSR.EOF (= TCI[4]) for each data word. The written TBUF[31:0] address location defines the word length and the end of a frame (locations TBUF[31:16] lead to a frame end).
  For example, writing transmit data to TBUF[07] results in a data word of 8-bit length without finishing the frame, whereas writing transmit data to TBUF[31] leads to a data word length of 16 bits, followed by T_{td}, the deactivation of MSLS and T_{nf}.
  If TCSR.WLEMD = 1, bits TCSR.EOF and SOF, as well as SCTR.WLE must not be written by software after writing data to a TBUF location. Furthermore, it is recommended to clear bits TCSR.SELMD, FLEMD and WAMD.
- FIFO-based address related end of frame handling:
  This mechanism can be used if a data FIFO is used to store the transmit data. The general behavior is similar to the software-based address related end of frame

handling, except that transmit data is not written to the locations TBUF[31:0], but to the FIFO input locations IN[31:0] instead. In this case, software must not write to any of the TBUF locations.

- TBUF related end of frame handling:
  If bit PCR.FEM = 0, an end of frame is assumed if the transmit buffer TBUF does not contain valid transmit data at the end of a data word transmission (TCSR.TDV = 0 or in Stop Mode). In this case, the software has to take care that TBUF does not run empty during a data frame in Run Mode. If bit PCR.FEM = 1, signal MSLS stays active while the transmit buffer is waiting for new data (TCSR.TDV = 1 again) or until Stop Mode is left.
- Explicit end of frame by software:
  The software can explicitly stop a frame by clearing bit PSR.MSLS by writing a 1 to the related bit position in register PSCR. This write action immediately clears bit PSR.MSLS, whereas the internal MSLS signal becomes inactive after finishing a currently running word transfer and respecting the slave select delays T_{td} and T_{nf}.

17.4.4 Operating the SSC in Slave Mode

In order to operate the SSC in slave mode, the following issues have to be considered:

- Select SSC mode:
  It is recommended to configure all parameters of the SSC that do not change during run time while CCR.MODE = 0000_B. Bit field SCTR.TRM = 01_B has to be programmed. The configuration of the input stages has to be done while CCR.MODE = 0000_B to avoid unintended edges of the input signals and the SSC mode can be enabled afterwards by CCR.MODE = 0001_B.
- Pin connections:
  Establish the connection of the input stage (DX0, DX3, DX4, DX5) with the selected receive data input pin (DIN[3:0]) with DXnCR.INSW = 1 and configure the transmit data output pin (DOUT[3:0]). One, two or four such connections may be needed depending on the protocol. For half-duplex configurations, hardware port control can be also used to establish the required connections.
  Establish a connection of input stage DX1 with the selected shift clock input pin (signal SCLKIN) with DX1CR.INSW = 1.
  Establish a connection of input stage DX2 with the selected slave select input pin (signal SELIN) with DX2CR.INSW = 1. If no slave select input signal is used, the DX2 stage has to deliver a 1-level to the data shift unit to allow data reception and transmission. If a slave device is not selected (DX2 stage delivers a 0 to the data shift unit) and a shift clock pulse are received, the incoming data is not received and the DOUTx signal outputs the passive data level defined by SCTR.PDL.
- Baud rate generation:
  The baud rate generator is not needed and can be switched off by the fractional divider.
- Data format configuration:
  If required, the shift mode can be set up for reception and/or transmission of two or four data bits at one time by programming the register SCTR.
- Slave select generation:
  The slave select delay generation is not needed and can be switched off. The bits and bit fields MSLSEN, SELCTR, SELINV, CTQSEL1, PCTQ1, DCTQ1, MSLSIEN, SELO[7:0], and TIWEN in register PCR are not necessary and can be programmed to 0.

17.4.4.1 Protocol Interrupts

The following protocol-related events generated in SSC mode and can lead to a protocol interrupt. They are related to the start and the end of a data frame. After the start of a data frame a new setting could be programmed for the next data frame and after the end of a data frame the SSC connections to pins can be changed.

Please note that the bits in register PSR are not all automatically cleared by hardware and have to be cleared by software in order to monitor new incoming events.

- MSLS event:
  The MSLS generation being switched off, this event is not available.
- DX2T event:
  The slave select input signal SELIN is handled by the DX2 stage and the edges of the selected signal can generate a protocol interrupt. This interrupt allows to indicate that a data frame has started and/or that a data frame has been completely finished. A programmable edge detection for the DX2 input signal activates DX2T, sets bit PSR.DX2TEV and additionally, a protocol interrupt can be generated if PCR.DX2TIEN = 1. The actual state of the selected input signal can be read out at PSR.DX2S to take appropriate actions when this interrupt has been detected.
- Parity Error Interrupt:
  This interrupt indicates that there is a mismatch in the received parity bit (in RBUFSR.PAR) with the calculated parity bit of the last received word of a data frame.

17.4.4.2 End-of-Frame Control

In slave mode, the following possibilities exist to determine the frame length. The slave device either has to refer to an external slave select signal, or to the number of received data bits.

- Frame length known in advance by the slave device, no slave select:
  In this case bit field SCTR.FLE can be programmed to the known value (if it does not exceed 63 bits). A currently running data word transfer is considered as finished if the programmed frame length is reached.
- Frame length not known by the slave, no slave select:
  In this case, the slave device's software has to decide on data word base if a frame is finished. Bit field SCTR.FLE can be either programmed to the word length SCTR.WLE, or to its maximum value to disable the slave internal frame length evaluation by counting received bits.
- Slave device addressed via slave select signal SELIN:
  If the slave device is addressed by a slave select signal delivered by the communication master, the frame start and end information are given by this signal. In this case, bit field SCTR.FLE should be programmed to its maximum value to disable the slave internal frame length evaluation.

17.4.5 SSC Protocol Registers

In SSC mode, the registers PCR and PSR handle SSC related information.

17.4.5.1 SSC Protocol Control Registers

In SSC mode, the PCR register bits or bit fields are defined as described in this section.

PCR
Protocol Control Register [SSC Mode]

($3C_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MCLK	0						TIWEN	SELO							
rw	rw						rw	rw							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DX2TIEN	MSLSIEN	PARIEN	DCTQ1					PCTQ1		CTQSEL1		FEM	SELINV	SELCTR	MSLSEN
rw	rw	rw	rw					rw		rw		rw	rw	rw	rw

Field	Bits	Type	Description
MSLSEN	0	rw	**MSLS Enable** This bit enables/disables the generation of the master slave select signal MSLS. If the SSC is a transfer slave, the SLS information is read from a pin and the internal generation is not needed. If the SSC is a transfer master, it has to provide the MSLS signal. 0_B The MSLS generation is disabled (MSLS = 0). This is the setting for SSC slave mode. 1_B The MSLS generation is enabled. This is the setting for SSC master mode.
SELCTR	1	rw	**Select Control** This bit selects the operating mode for the SELO[7:0] outputs. 0_B The coded select mode is enabled. 1_B The direct select mode is enabled.

Field	Bits	Type	Description
SELINV	2	rw	**Select Inversion** This bit defines if the polarity of the SELO[7:0] outputs in relation to the master slave select signal MSLS. 0_B The SELO outputs have the same polarity as the MSLS signal (active high). 1_B The SELO outputs have the inverted polarity to the MSLS signal (active low).
FEM	3	rw	**Frame End Mode** This bit defines if a transmit buffer content that is not valid for transmission is considered as an end of frame condition for the slave select generation. 0_B The current data frame is considered as finished when the last bit of a data word has been sent out and the transmit buffer TBUF does not contain new data (TDV = 0). 1_B The MSLS signal is kept active also while no new data is available and no other end of frame condition is reached. In this case, the software can accept delays in delivering the data without automatic deactivation of MSLS in multi-word data frames.
CTQSEL1	[5:4]	rw	**Input Frequency Selection** This bit field defines the input frequency f_{CTQIN} for the generation of the slave select delays T_{iw} and T_{nf}. 00_B $f_{CTQIN} = f_{PDIV}$ 01_B $f_{CTQIN} = f_{PPP}$ 10_B $f_{CTQIN} = f_{SCLK}$ 11_B $f_{CTQIN} = f_{MCLK}$
PCTQ1	[7:6]	rw	**Divider Factor PCTQ1 for T_{iw} and T_{nf}** This bit field represents the divider factor PCTQ1 (range = 0 - 3) for the generation of the inter-word delay and the next-frame delay. $T_{iw} = T_{nf} = 1/f_{CTQIN}$ x (PCTQ1 + 1) x (DCTQ1 + 1)
DCTQ1	[12:8]	rw	**Divider Factor DCTQ1 for T_{iw} and T_{nf}** This bit field represents the divider factor DCTQ1 (range = 0 - 31) for the generation of the inter-word delay and the next-frame delay. $T_{iw} = T_{nf} = 1/f_{CTQIN}$ x (PCTQ1 + 1) x (DCTQ1 + 1)

Field	Bits	Type	Description
PARIEN	13	rw	**Parity Error Interrupt Enable** This bit enables/disables the generation of a protocol interrupt with the detection of a parity error. 0_B A protocol interrupt is not generated with the detection of a parity error. 1_B A protocol interrupt is generated with the detection of a parity error.
MSLSIEN	14	rw	**MSLS Interrupt Enable** This bit enables/disables the generation of a protocol interrupt if the state of the MSLS signal changes (indicated by PSR.MSLSEV = 1). 0_B A protocol interrupt is not generated if a change of signal MSLS is detected. 1_B A protocol interrupt is generated if a change of signal MSLS is detected.
DX2TIEN	15	rw	**DX2T Interrupt Enable** This bit enables/disables the generation of a protocol interrupt if the DX2T signal becomes activated (indicated by PSR.DX2TEV = 1). 0_B A protocol interrupt is not generated if DX2T is activated. 1_B A protocol interrupt is generated if DX2T is activated.
SELO	[23:16]	rw	**Select Output** This bit field defines the setting of the SELO[7:0] output lines. 0_B The corresponding SELOx line cannot be activated. 1_B The corresponding SELOx line can be activated (according to the mode selected by SELCTR).
TIWEN	24	rw	**Enable Inter-Word Delay T_{iw}** This bit enables/disables the inter-word delay T_{iw} after the transmission of a data word. 0_B No delay between data words of the same frame. 1_B The inter-word delay T_{iw} is enabled and introduced between data words of the same frame.

Field	Bits	Type	Description
MCLK	31	rw	**Master Clock Enable** This bit enables/disables the generation of the master clock output signal MCLK, independent from master or slave mode. 0_B The MCLK generation is disabled and output MCLK = 0. 1_B The MCLK generation is enabled.
0	[30:25]	rw	**Reserved** Returns 0 if read; should be written with 0.

17.4.5.2 SSC Protocol Status Register

In SSC mode, the PSR register bits or bit fields are defined as described in this section. The bits and bit fields in register PSR are not cleared by hardware.

The flags in the PSR register can be cleared by writing a 1 to the corresponding bit position in register PSCR. Writing a 1 to a bit position in PSR sets the corresponding flag, but does not lead to further actions (no interrupt generation). Writing a 0 has no effect. The PSR flags should be cleared by software before enabling a new protocol.

PSR
Protocol Status Register [SSC Mode] (48_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															BRGIF
r															rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AIF	RIF	TBIF	TSIF	DLIF	RSIF	0					PARERR	DX2TEV	MSLSEV	DX2S	MSLS
rwh	rwh	rwh	rwh	rwh	rwh	r					rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
MSLS	0	rwh	**MSLS Status** This bit indicates the current status of the MSLS signal. It must be cleared by software to stop a running frame. 0_B The internal signal MSLS is inactive (0). 1_B The internal signal MSLS is active (1).
DX2S	1	rwh	**DX2S Status** This bit indicates the current status of the DX2S signal that can be used as slave select input SELIN. 0_B DX2S is 0. 1_B DX2S is 1.
MSLSEV	2	rwh	**MSLS Event Detected**[1)] This bit indicates that the MSLS signal has changed its state since MSLSEV has been cleared. Together with the MSLS status bit, the activation/deactivation of the MSLS signal can be monitored. 0_B The MSLS signal has not changed its state. 1_B The MSLS signal has changed its state.

Field	Bits	Type	Description
DX2TEV	3	rwh	**DX2T Event Detected**[1)] This bit indicates that the DX2T trigger signal has been activated since DX2TEV has been cleared. 0_B The DX2T signal has not been activated. 1_B The DX2T signal has been activated.
PARERR	4	rwh	**Parity Error Event Detected**[1)] This bit indicates that there is a mismatch in the received parity bit (in RBUFSR.PAR) with the calculated parity bit of the last received word of the data frame. 0_B A parity error event has not been activated. 1_B A parity error event has been activated.
RSIF	10	rwh	**Receiver Start Indication Flag** 0_B A receiver start event has not occurred. 1_B A receiver start event has occurred.
DLIF	11	rwh	**Data Lost Indication Flag** 0_B A data lost event has not occurred. 1_B A data lost event has occurred.
TSIF	12	rwh	**Transmit Shift Indication Flag** 0_B A transmit shift event has not occurred. 1_B A transmit shift event has occurred.
TBIF	13	rwh	**Transmit Buffer Indication Flag** 0_B A transmit buffer event has not occurred. 1_B A transmit buffer event has occurred.
RIF	14	rwh	**Receive Indication Flag** 0_B A receive event has not occurred. 1_B A receive event has occurred.
AIF	15	rwh	**Alternative Receive Indication Flag** 0_B An alternative receive event has not occurred. 1_B An alternative receive event has occurred.
BRGIF	16	rwh	**Baud Rate Generator Indication Flag** 0_B A baud rate generator event has not occurred. 1_B A baud rate generator event has occurred.
0	[9:5], [31:17]	r	**Reserved** Returns 0 if read; not modified in SSC mode.

1) This status bit can generate a protocol interrupt in SSC mode (see **Page 17-21**). The general interrupt status flags are described in the general interrupt chapter.

17.4.6 SSC Timing Considerations

The input and output signals have to respect certain timings in order to ensure correct data reception and transmission. In addition to module internal timings (due to input filters, reaction times on events, etc.), also the timings from the input pin via the input stage (T_{in}) to the module and from the module via the output driver stage to the pin (T_{out}), as well as the signal propagation on the wires (T_{prop}) have to be taken into account.

Please note that there might be additional delays in the DXn input stages, because the digital filter and the synchronization stages lead to systematic delays, that have to be considered if these functions are used.

17.4.6.1 Closed-loop Delay

A system-inherent limiting factor for the baud rate of an SSC connection is the closed-loop delay. In a typical application setup, a communication master device is connected to a slave device in full-duplex mode with independent lines for transmit and receive data. In a general case, all transmitters refer to one shift clock edge for transmission and all receivers refer to the other shift clock edge for reception. The master device's SSC module sends out the transmit data, the shift clock and optionally the slave select signal. Therefore, the baud rate generation (BRG) and slave select generation (SSG) are part of the master device. The frame control is similar for SSC modules in master and slave mode, the main difference is the fact which module generates the shift clock and optionally, the slave select signals.

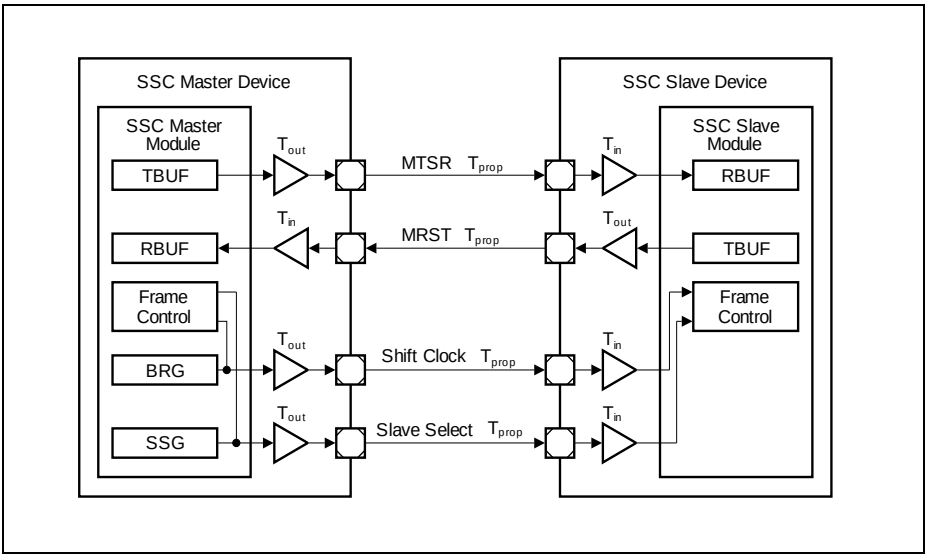


Figure 17-45 SSC Closed-loop Delay

The signal path between the SSC modules of the master and the slave device includes the master's output driver, the wiring to the slave device and the slave device's input stage. With the received shift clock edges, the slave device receives the master's transmit data and transmits its own data back to the master device, passing by a similar signal path in the other direction. The master module receives the slave's transmit data related to its internal shift clock edges. In order to ensure correct data reception in the master device, the slave's transmit data has to be stable (respecting setup and hold times) as master receive data with the next shift clock edge of the master (generally 1/2 shift clock period). To avoid data corruption, the accumulated delays of the input and output stages, the signal propagation on the wiring and the reaction times of the transmitter/receiver have to be carefully considered, especially at high baud rates.
In the given example, the time between the generation of the shift clock signal and the evaluation of the receive data by the master SSC module is given by the sum of T_{out_master} + 2 x T_{prop} + T_{in_slave} + T_{out_slave} + T_{in_master} + module reaction times + input setup times.
The input path is characterized by an input delay depending mainly on the input stage characteristics of the pads. The output path delay is determined by the output driver delay and its slew rate, the external load and current capability of the driver. The device specific values for the input/output driver are given in the Data Sheet.

Figure 17-46 describes graphically the closed-loop delay and the effect of two delay compensation options discussed in **Section 17.4.6.2** and **Section 17.4.6.3**.

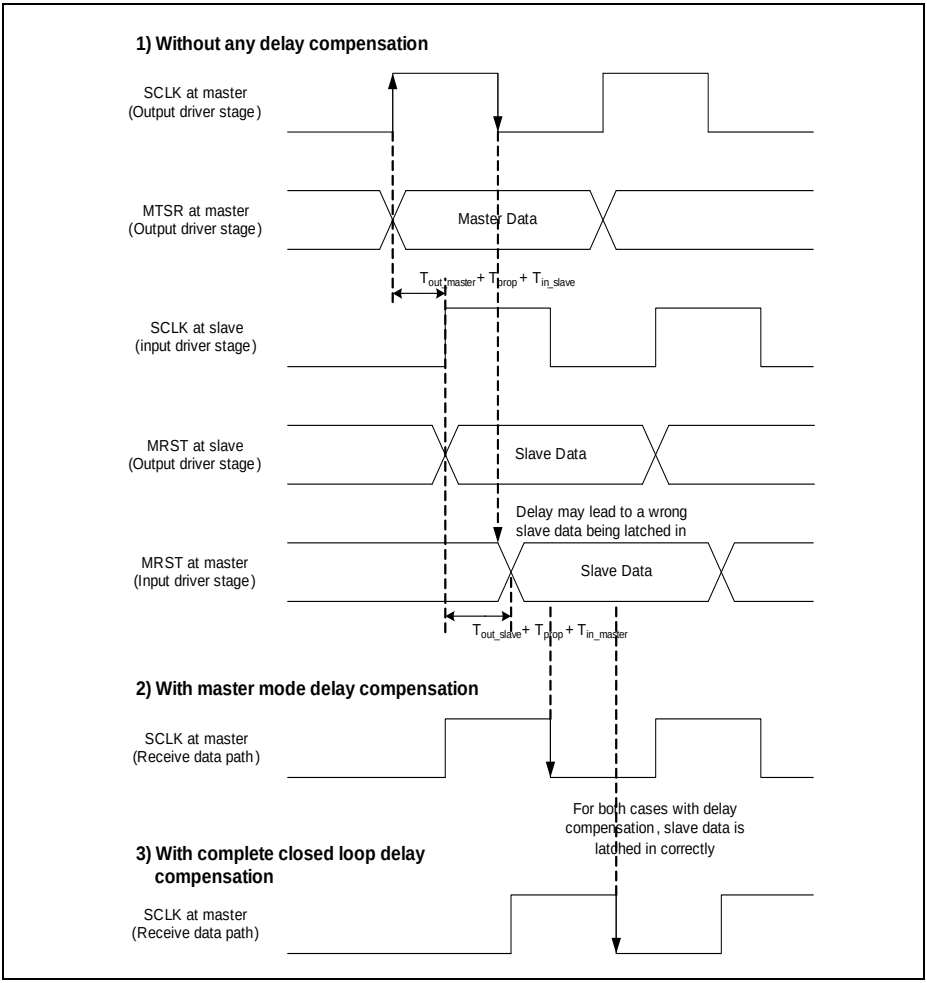


Figure 17-46 SSC Closed-loop Delay Timing Waveform

17.4.6.2 Delay Compensation in Master Mode

A higher baud rate can be reached by delay compensation in master mode. This compensation is possible if (at least) the shift clock pin is bidirectional.

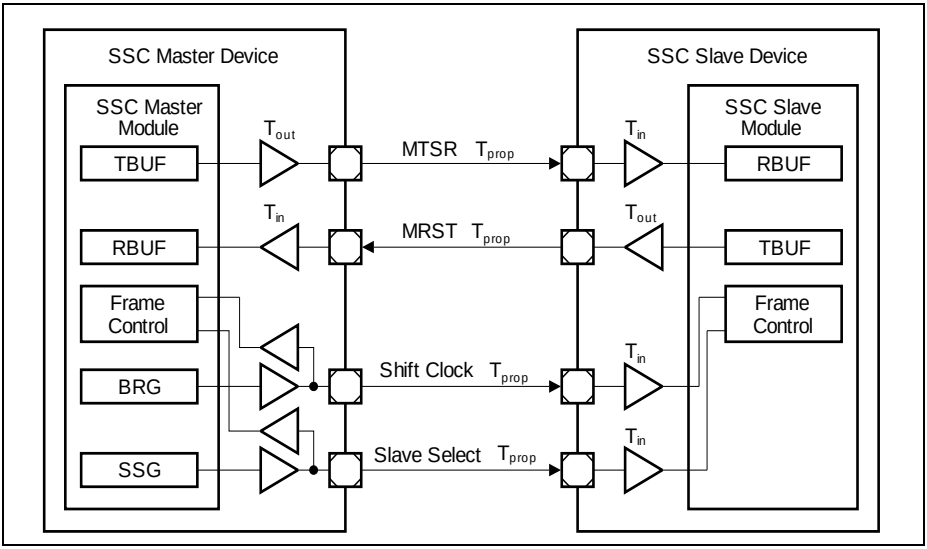


Figure 17-47 SSC Master Mode with Delay Compensation

If the receive shift clock signal in master mode is directly taken from the input function in parallel to the output signal, the output delay of the master device's shift clock output is compensated and only the difference between the input delays of the master and the slave devices have to be taken into account instead of the complete master's output delay and the slave's input delay of the shift clock path. The delay compensation is enabled with DX1CR.DCEN = 1 while DX1CR.INSW = 0 (transmit shift clock is taken from the baud-rate generator).

In the given example, the time between the evaluation of the shift clock signal and the receive data by the master SSC module is reduced by $T_{in\_master} + T_{out\_master}$.

Although being a master mode, the shift clock input and optionally the slave select signal are not directly connected internally to the data shift unit, but are taken as external signals from input pins. The delay compensation does not lead to additional pins for the SSC communication if the shift clock output pin (slave select output pin, respectively) is/are bidirectional. In this case, the input signal is decoupled from other internal signals, because it is related to the signal level at the pin itself.

17.4.6.3 Complete Closed-loop Delay Compensation

Alternatively, the complete closed-loop delay can be compensated by using one additional pin on both the SSC master and slave devices for the SSC communication.

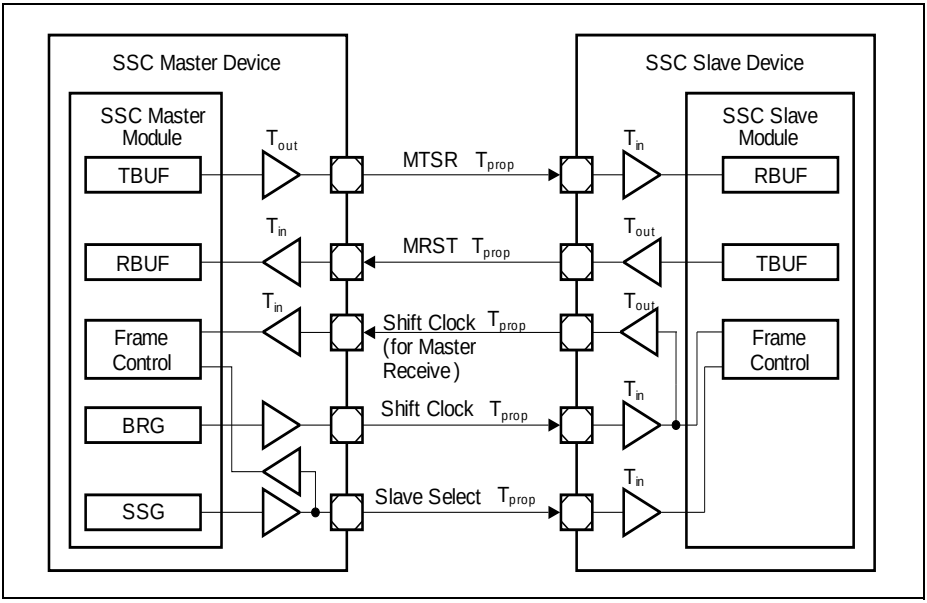


Figure 17-48 SSC Complete Closed-loop Delay Compensation

The principle behind this delay compensation method is to have the slave feedback the shift clock back to the master, which uses it as the receive shift clock. By going through a complete closed-loop signal path, the receive shift clock is thus fully compensated.

The slave has to setup the SCLKOUT pin function to output the shift clock by setting the bit BRG.SCLKOSEL to 1, while the master has to setup the DX1 pin function to receive the shift clock from the slave and enable the delay compensation with DX1CR.DCEN = 1 and DX1CR.INSW = 0.

17.5 Inter-IC Bus Protocol (IIC)

The IIC protocol of the USIC refers to the IIC bus specification **[17]**. Contrary to that specification, the USIC device assumes rise/fall times of the bus signals of max. 300 ns in all modes. Please refer to the pad characteristics in the AC/DC chapter for the driver capability. CBUS mode and HS mode are not supported.
The IIC mode is selected by CCR.MODE = 0100_B with CCFG.IIC = 1 (IIC mode available).

17.5.1 Introduction

USIC IIC Features:

- Two-wire interface, with one line for shift clock transfer and synchronization (shift clock SCL), the other one for the data transfer (shift data SDA)
- Communication in standard mode (100 kBit/s) or in fast mode (up to 400 kBit/s)
- Support of 7-bit addressing, as well as 10-bit addressing
- Master mode operation,
  where the IIC controls the bus transactions and provides the clock signal.
- Slave mode operation,
  where an external master controls the bus transactions and provides the clock signal.
- Multi-master mode operation,
  where several masters can be connected to the bus and bus arbitration can take place, i.e. the IIC module can be master or slave. The master/slave operation of an IIC bus participant can change from frame to frame.
- Efficient frame handling (low software effort), also allowing DMA transfers
- Powerful interrupt handling due to multitude of indication flags
- Compensation support for input delays

17.5.1.1 Signal Description

An IIC connection is characterized by two wires (SDA and SCL). The output drivers for these signals must have open-drain characteristics to allow the wired-AND connection of all SDA lines together and all SCL lines together to form the IIC bus system. Due to this structure, a high level driven by an output stage does not necessarily lead immediately to a high level at the corresponding input. Therefore, each SDA or SCL connection has to be input and output at the same time, because the input function always monitors the level of the signal, also while sending.

- Shift data SDA: input handled by DX0 stage, output signal DOUT0
- Shift clock SCL: input handled by DX1 stage, output signal SCLKOUT

Figure 17-29 shows a connection of two IIC bus participants (modules IIC A and IIC B) using the USIC. In this example, the pin assignment of module IIC A shows separate pins for the input and output signals for SDA and SCL. This assignment can be used if the application does not provide pins having DOUT0 and a DX0 stage input for the same pin

(similar for SCLKOUT and DX1). The pin assignment of module IIC B shows the connection of DOUT0 and a DX0 input at the same pin, also for SCLKOUT and a DX1 input.

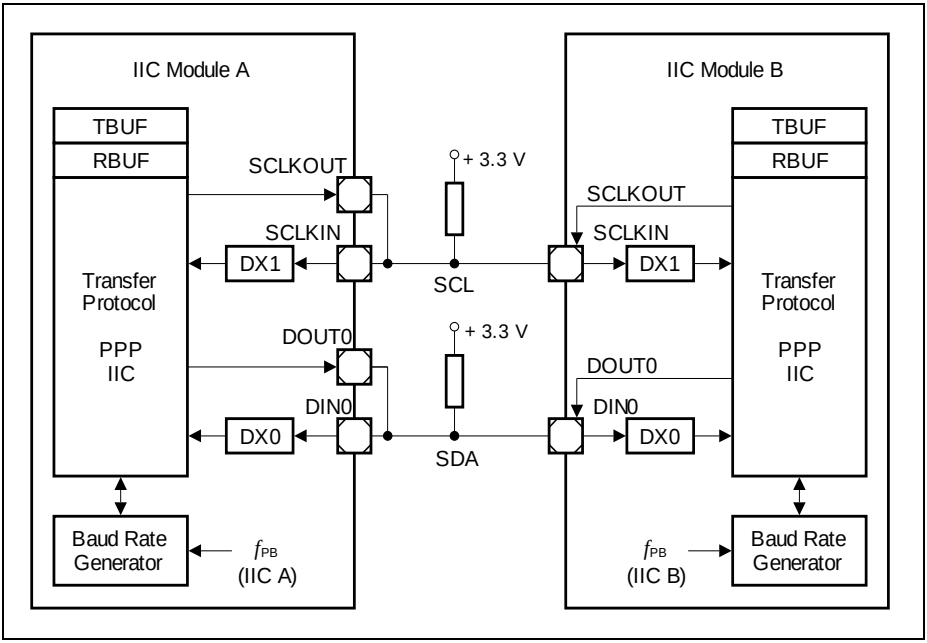


Figure 17-49 IIC Signal Connections

17.5.1.2 Symbols

A symbol is a sequence of edges on the lines SDA and SCL. Symbols contain 10 or 25 time quanta t_q, depending on the selected baud rate. The baud rate generator determines the length of the time quanta t_q, the sequence of edges in a symbol is handled by the IIC protocol pre-processor, and the sequence of symbols can be programmed by the user according to the application needs.

The following symbols are defined:

- Bus idle:
  SDA and SCL are high. No data transfer takes place currently.
- Data bit symbol:
  SDA stable during the high phase of SCL. SDA then represents the transferred bit value. There is one clock pulse on SCL for each transferred bit of data. During data transfers SDA may only change while SCL is low.

- Start symbol:
  Signal SDA being high followed by a falling edge of SDA while SCL is high indicates a start condition. This start condition initiates a data transfer over the IIC bus after the bus has been idle.
- Repeated start symbol:
  This start condition initiates a data transfer over the bus after a data symbol when the bus has not been idle. Therefore, SDA is set high and SCL low, followed by a start symbol.
- Stop symbol:
  A rising edge on SDA while SCL is high indicates a stop condition. This stop condition terminates a data transfer to release the bus to idle state. Between a start condition and a stop condition an arbitrary number of bytes may be transferred.

17.5.1.3 Frame Format

Data is transferred by the 2-line IIC bus (SDA, SCL) using a protocol that ensures reliable and efficient transfers. The sender of a (data) byte receives and checks the value of the following acknowledge field. The IIC being a wired-AND bus system, a 0 of at least one device leads to a 0 on the bus, which is received by all devices.

A data word consists of 8 data bit symbols for the data value, followed by another data bit symbol for the acknowledge bit. The data word can be interpreted as address information (after a start symbol) or as transferred data (after the address).

In order to be able to receive an acknowledge signal, the sender of the data bits has to release the SDA line by sending a 1 as acknowledge value. Depending on the internal state of the receiver, the acknowledge bit is either sent active or passive.

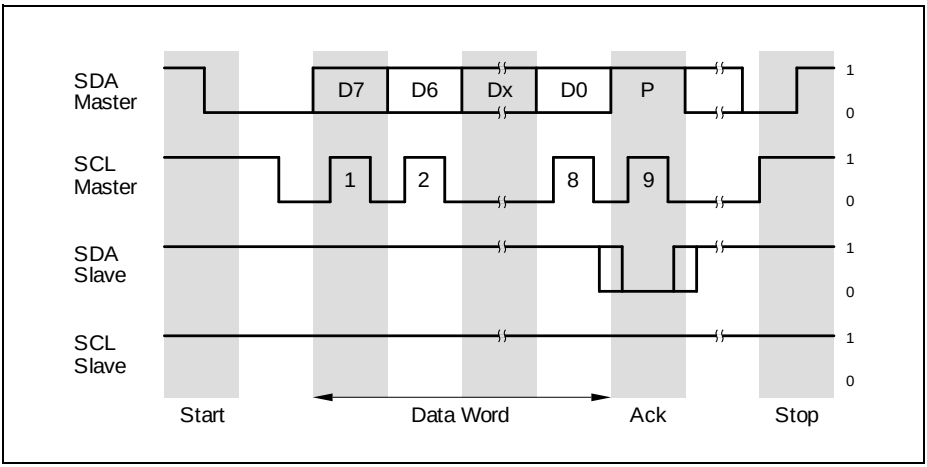


Figure 17-50 IIC Frame Example (simplified)

17.5.2 Operating the IIC

In order to operate the IIC protocol, the following issues have to be considered:

- Select IIC mode:
  It is recommended to configure all parameters of the IIC that do not change during run time while CCR.MODE = 0000_B. Bit field SCTR.TRM = 11_B should be programmed. The configuration of the input stages has to be done while CCR.MODE = 0000_B to avoid unintended edges of the input signals and the IIC mode can be enabled by CCR.MODE = 0100_B afterwards.
- Pin connections:
  Establish a connection of input stage DX0 (with DX0CR.DPOL = 0) to the selected shift data pin SDA (signal DIN0) with DX0CR.INSW = 0 and configure the transmit data output signal DOUT0 (with SCTR.DOCFG = 00_B) to the same pin. If available, this can be the same pin for input and output, or connect the selected input pin and the output pin to form the SDA line.
  The same mechanism applies for the shift clock line SCL. Here, signal SCLKOUT (with BRG.SCLKCFG = 00_B) and an input of the DX1 stage have to be connected (with DX1CR.DPOL = 0).
  The input stage DX2 is not used for the IIC protocol.
  If the digital input filters are enabled in the DX0/1 stages, their delays have to be taken into account for correct calculation of the signal timings.
  The pins used for SDA and SCL have to be set to open-drain mode to support the wired-AND structure of the IIC bus lines.
  Note that the basic I/O port configuration for the IIC I/O pins must also setup correctly before the IIC mode becomes enabled by CCR.MODE = 0100_B.
- Bit timing configuration:
  In standard mode (100 kBit/s) a minimum module frequency of 2 MHz is necessary, whereas in fast mode (400 kBit/s) a minimum of 10 MHz is required. Additionally, if the digital filter stage should be used to eliminate spikes up to 50 ns, a filter frequency of 20 MHz is necessary.
  There could be an uncertainty in the SCL high phase timing of maximum $1/f_{PPP}$ if another IIC participant lengthens the SCL low phase on the bus.
  More details are given in **Section 17.5.3**.
- Data format configuration:
  The data format has to be configured for 8 data bits (SCTR.WLE = 7), unlimited data flow (SCTR.FLE = $3FF_H$), and MSB shifted first (SCTR.SDIR = 1). The parity generation has to be disabled (CCR.PM = 00_B).
- General hints:
  The IIC slave module becomes active (for reception or transmission) if it is selected by the address sent by the master. In the case that the slave sends data to the master, it uses the transmit path. So a master must not request to read data from the slave address defined for its own channel in order to avoid collisions.
  The built-in error detection mechanisms are only activated while the IIC module is

taking part in IIC bus traffic.
If the slave can not deal with too high frequencies, it can lengthen the low phase of the SCL signal.
For data transfers according to the IIC specification, the shift data line SDA shall only change while SCL = 0 (defined by IIC bus specification).

17.5.2.1 Transmission Chain

The IIC bus protocol requiring a kind of in-bit-response during the arbitration phase and while a slave is transmitting, the resulting loop delay of the transmission chain can limit the reachable maximal baud rate, strongly depending on the bus characteristics (bus load, module frequency, etc.).

Figure 17-49 shows the general signal path and the delays in the case of a slave transmission. The shift clock SCL is generated by the master device, output on the wire, then it passes through the input stage and the input filter. Now, the edges can be detected and the SDA data signal can be generated accordingly. The SDA signal passes through the output stage and the wire to the master receiver part. There, it passes through the input stage and the input filter before it is sampled.

This complete loop has to be finished (including all settling times to obtain stable signal levels) before the SCL signal changes again. The delays in this path have to be taken into account for the calculation of the baud rate as a function of f_{PB} and f_{PPP}.

17.5.2.2 Byte Stretching

If a device is selected as transceiver and should transmit a data byte but the transmit buffer TBUF does not contain valid data to be transmitted, the device ties down SCL = 0 at the end of the previous acknowledge bit. The waiting period is finished if new valid data has been detected in TBUF.

17.5.2.3 Master Arbitration

During the address and data transmission, the master transmitter checks at the rising edge of SCL for each data bit if the value it is sending is equal to the value read on the SDA line. If yes, the next data bit values can be 0. If this is not the case (transmitted value = 1, value read = 0), the master has lost the transmit arbitration. This is indicated by status flag PSR.ARL and can generate a protocol interrupt if enabled by PCR.ARLIEN.
When the transmit arbitration has been lost, the software has to initialize the complete frame again, starting with the first address byte together with the start condition for a new master transmit attempt. Arbitration also takes place for the ACK bit.

17.5.2.4 Release of TBUF

In case of a non-acknowledge or an error, the content of TBUF becomes invalid. In both cases, the software has to flush the transmit buffer and to set it up again with appropriate values to react on the previous event.

17.5.2.5 Mode Control Behavior

In multi-master mode, only run mode 0 and stop mode 0 are supported, the other modes must not be programmed.

- Run Mode 0:
  Behavior as programmed. If TCSR.TDV = 0 (no new valid TBUF entry found) when a new TBUF entry needs to be processed, the IIC module waits for TDV becoming set to continue operation.
- Run Mode 1:
  Behavior as programmed. If in master mode, TCSR.TDV = 0 (no new valid TBUF entry found) when a new TBUF entry needs to be processed, the IIC module sends a stop condition to finish the frame. In slave mode, no difference to run mode 0.
- Stop Mode 0:
  Bit TCSR.TDV is internally considered as 0 (the bit itself is not modified by the stop mode). A currently running word is finished normally, but no new word is started in case of master mode (wait for TDV active).
  Bit TDV being considered as 0 for master and slave, the slave will force a wait state on the bus if read by an external master, too.
  Additionally, it is not possible to force the generation of a STOP condition out of the wait state. The reason is, that a master read transfer must be finished with a not-acknowledged followed by a STOP condition to allow the slave to release his SDA line. Otherwise the slave may force the SDA line to 0 (first data bit of next byte) making it impossible to generate the STOP condition (rising edge on SDA).
  To continue operation, the mode must be switched to run mode 0
- Stop Mode 1:
  Same as stop mode 0, but additionally, a master sends a STOP condition to finish the frame.
  If stop mode 1 is requested for a master device after the first byte of a 10 bit address, a stop condition will be sent out. In this case, a slave device will issue an error interrupt.

17.5.2.6 Data Transfer Interrupt Handling

The data transfer interrupts indicate events related to IIC frame handling. As the data input and output pins are the same in IIC protocol, a IIC transmitter also receives the output data at its input pin. However, no receive related interrupts will be generated in this case.

- Transmit buffer event:
  The transmit buffer event indication flag PSR.TBIF is set when the content of the transmit buffer TBUF has been loaded to the transmit shift register, indicating that the action requested by the TBUF entry has started.
  With this event, bit TCSR.TDV is cleared. This interrupt can be used to write the next TBUF entry while the last one is in progress (handled by the transmitter part).
- Receive event:
  This receive event indication flag PSR.RIF indicates that a new data byte has been written to the receive buffer RBUF0/1 (except for the first data byte of a new frame, that is indicated by an alternative receive interrupt). The flag becomes set when the data byte is received (after the falling edge of SCL). This interrupt can be used to read out the received data while a new data byte can be in progress (handled by the receiver part).
- Alternate receive event:
  The alternative receive event indication flag AIF is based on bit RBUFSR[9] (same as RBUF[9]), indicating that the received data word has been the first data word of a new data frame.
- Transmit shift event:
  The transmit shift event indication flag TSIF is set after the start of the last data bit of a data byte.
- Receive start event:
  The receive start event indication flag RSIF is set after the sample point of the first data bit of a data byte.

Note: The transmit shift and receive start events can be ignored if the application does not require them during the IIC data transfer.

17.5.2.7 IIC Protocol Interrupt Events

The following protocol-related events are generated in IIC mode and can lead to a protocol interrupt.

Please note that the bits in register PSR are not all automatically cleared by hardware and have to be cleared by software in order to monitor new incoming events.

- start condition received at a correct position in a frame (PSR.SCR)
- repeated start condition received at a correct position in a frame (PSR.RSCR)
- stop condition transferred at a correct position in a frame (PSR.PCR)
- master arbitration lost (PSR.ARL)
- slave read requested (PSR.SRR)
- acknowledge received (PSR.ACK)
- non-acknowledge received (PSR.NACK)
- start condition not at the expected position in a frame (PSR.ERR)
- stop condition not at the expected position in a frame (PSR.ERR)

- as slave, 10-bit address interrupted by a stop condition after the first address byte (PSR.ERR)
- TDF slave code in master mode (PSR.WTDF)
- TDF master code in slave mode (PSR.WTDF)
- Reserved TDF code found (PSR.WDTF)
- Start condition code during a running frame in master mode (PSR.WTDF)
- Data byte transmission code after transfer direction has been changed to reception (master read) in master mode (PSR.WTDF)

If a wrong TDF code is found in TBUF, the error event is active until the TDF value is either corrected or invalidated. If the related interrupt is enabled, the interrupt handler should check PSR.WDTF first and correct or invalidate TBUF, before dealing with the other possible interrupt events.

17.5.2.8 Baud Rate Generator Interrupt Handling

The baud rate generator interrupt indicate that the capture mode timer has reached its maximum value. With this event, the bit PSR.BRGIF is set.

17.5.2.9 Receiver Address Acknowledge

After a (repeated) start condition, the master sends a slave address to identify the target device of the communication. The start address can comprise one or two address bytes (for 7 bit or for 10 bit addressing schemes). After an address byte, a slave sensitive to the transmitted address has to acknowledge the reception.

Therefore, the slave's address can be programmed in the device, where it is compared to the received address. In case of a match, the slave answers with an acknowledge (SDA = 0). Slaves that are not targeted answer with an non-acknowledge (SDA = 1).
In addition to the match of the programmed address, another address byte value has to be answered with an acknowledge if the slave is capable to handle the corresponding requests. The address byte 00_H indicates a general call address, that can be acknowledged. The value 01_H stands for a start byte generation, that is not acknowledged

In order to allow selective acknowledges for the different values of the address byte(s), the following control mechanism is implemented:

- The address byte 00_H is acknowledged if bit PCR.ACK00 is set.
- The address byte 01_H is not acknowledged.
- The first 7 bits of a received first address byte are compared to the programmed slave address (PCR.SLAD[15:9]). If these bits match, the slave sends an acknowledge. In addition to this, if the slave address is programmed to 1111 $0XX_B$, the slave device waits for a second address byte and compares it also to PCR.SLAD[7:0] and sends an acknowledge accordingly to cover the 10 bit addressing mode. The user has to

take care about reserved addresses (refer to IIC specification for more detailed description). Only the address 1111 $0XX_B$ is supported.

Under each of these conditions, bit PSR.SLSEL will be set when the addressing delivered a match. This bit is cleared automatically by a (repeated) start condition.

17.5.2.10 Receiver Handling

A selected slave receiver always acknowledges a received data byte. If the receive buffers RBUF0/1 are already full and can not accept more data, the respective register is overwritten (PSR.DLI becomes set in this case and a protocol interrupt can be generated).

An address reception also uses the registers RBUF0/1 to store the address before checking if the device is selected. The received addresses do not set RDV0/1, so the addresses are not handled like received data.

17.5.2.11 Receiver Status Information

In addition to the received data byte, some IIC protocol related information is stored in the 16-bit data word of the receive buffer. The received data byte is available at the bit positions RBUF[7:0], whereas the additional information is monitored at the bit positions RBUF[12:8]. This structure allows to identify the meaning of each received data byte without reading additional registers, also when using a FIFO data buffer.

- RBUF[8]:
  Value of the received acknowledge bit. This information is also available in RBUFSR[8] as protocol argument.
- RBUF[9]:
  A 1 at this bit position indicates that after a (repeated) start condition followed by the address reception the first data byte of a new frame has been received. A 0 at this bit position indicates further data bytes. This information is also available in RBUFSR[9], allowing different interrupt routines for the address and data handling.
- RBUF[10]:
  A 1 at this bit position indicates that the data byte has been received when the device has been in slave mode, whereas a 0 indicates a reception in master mode.
- RBUF[11]:
  A 1 at this bit position indicates an incomplete/erroneous data byte in the receive buffer caused by a wrong position of a START or STOP condition in the frame. The bit is not identical to the frame error status bit in PSR, because the bit in the PSR has to be cleared by software (“sticky” bit), whereas RBUF[11] is evaluated data byte by data byte. If RBUF[11] = 0, the received data byte has been correct, independent of former errors.
- RBUF[12]:
  A 0 at this bit position indicates that the programmed address has been received. A 1 indicates a general call address.

17.5.3 Symbol Timing

The symbol timing of the IIC is determined by the master stimulating the shift clock line SCL. It is different for standard and fast IIC mode.

- 100 kBaud standard mode (PCR.STIM = 0):
  The symbol timing is based on 10 time quanta t_q per symbol. A minimum module clock frequency f_{PB} = 2 MHz is required.
- 400 kBaud standard mode (PCR.STIM = 1):
  The symbol timing is based on 25 time quanta t_q per symbol. A minimum module clock frequency f_{PB} = 10 MHz is required.

The baud rate setting should only be changed while the transmitter and the receiver are idle or CCR.MODE = 0. The bits in register BRG define the length of a time quantum t_q that is given by one period of f_{PCTQ}.

- BRG.CTQSEL
  to define the input frequency f_{CTQIN} for the time quanta generation
- BRG.PCTQ
  to define the length of a time quantum (division of f_{CTQIN} by 1, 2, 3, or 4)
- BRG.DCTQ
  to define the number of time quanta per symbol (number of t_q = DCTQ + 1)

The standard setting is given by CTQSEL = 00_B ($f_{CTQIN} = f_{PDIV}$) and PPPEN = 0 ($f_{PPP} = f_{IN}$). Under these conditions, the frequency f_{PCTQ} is given by:

$$f_{PCTQ} = f_{PIN} \times \frac{1}{PDIV + 1} \times \frac{1}{PCTQ + 1} \quad (17.10)$$

To respect the specified SDA hold time of 300 ns after a falling edge of signal SCL, a hold delay t_{HDEL} has been introduced. It also prevents an erroneous detection of a start or a stop condition. The length of this delay can be programmed by bit field PCR.HDEL. Taking into account the input sampling and output update, bit field HDEL can be programmed according to:

$$HDEL \geq 300\ ns \times f_{PPP} - \left(3 \times \frac{f_{PPP}}{f_{PB}}\right) + 1 \quad \text{with digital filter and } HDEL_{min} = 2$$

$$HDEL \geq 300\ ns \times f_{PPP} - \left(3 \times \frac{f_{PPP}}{f_{PB}}\right) + 2 \quad \text{without digital filter and } HDEL_{min} = 1 \quad (17.11)$$

If the digital input filter is used, HDEL compensates the filter delay of 2 filter periods (f_{PPP} should be used) in case of a spike on the input signal. This ensures that a data bit on the SDA line changing just before the rising edge or behind the falling edge of SCL will not be treated as a start or stop condition.

17.5.3.1 Start Symbol

Figure 17-51 shows the general start symbol timing.

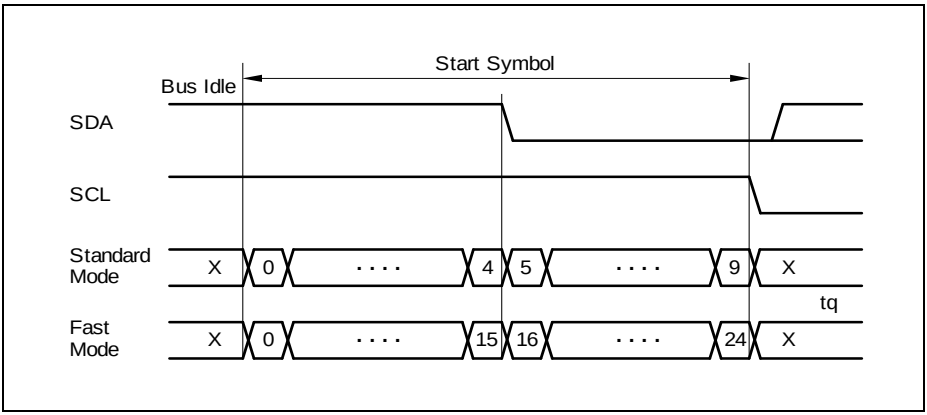


Figure 17-51 Start Symbol Timing

17.5.3.2 Repeated Start Symbol

During the first part of a repeated start symbol, an SCL low value is driven for the specified number of time quanta. Then a high value is output. After the detection of a rising edge at the SCL input, a normal start symbol is generated, as shown in **Figure 17-52**.

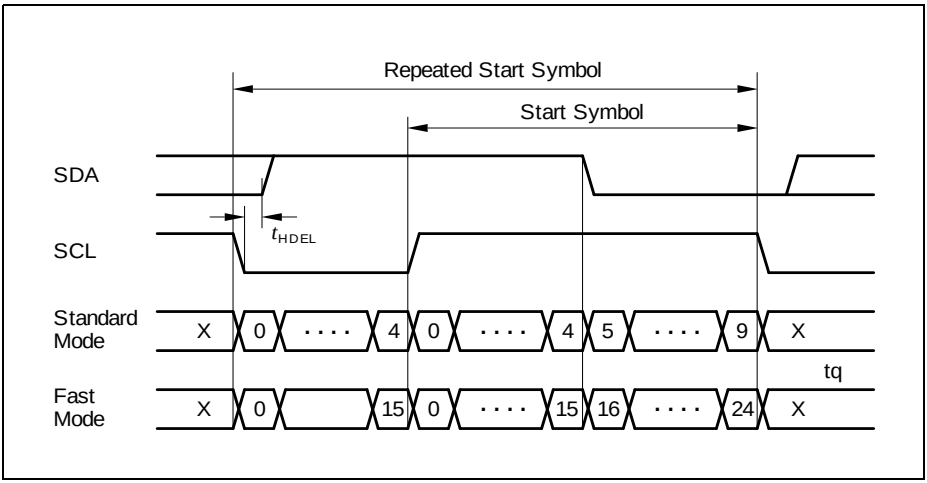


Figure 17-52 Repeated Start Symbol Timing

17.5.3.3 Stop Symbol

Figure 17-53 shows the stop symbol timing.

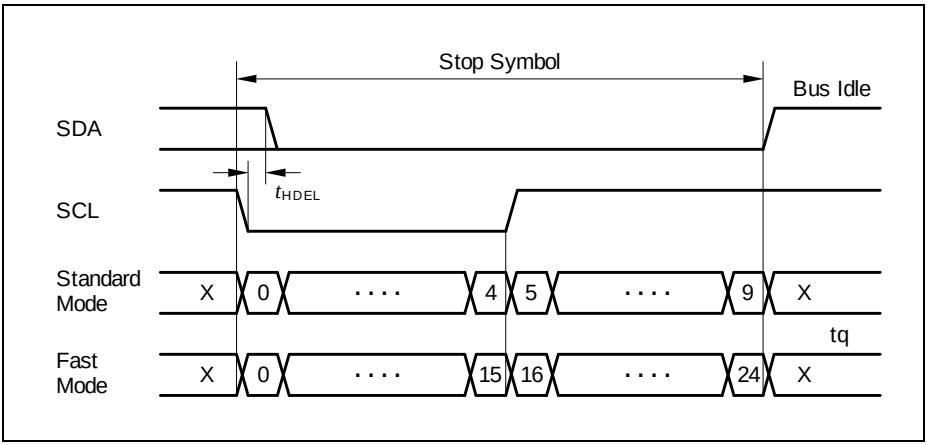


Figure 17-53 Stop Symbol Timing

17.5.3.4 Data Bit Symbol

Figure 17-54 shows the general data bit symbol timing.

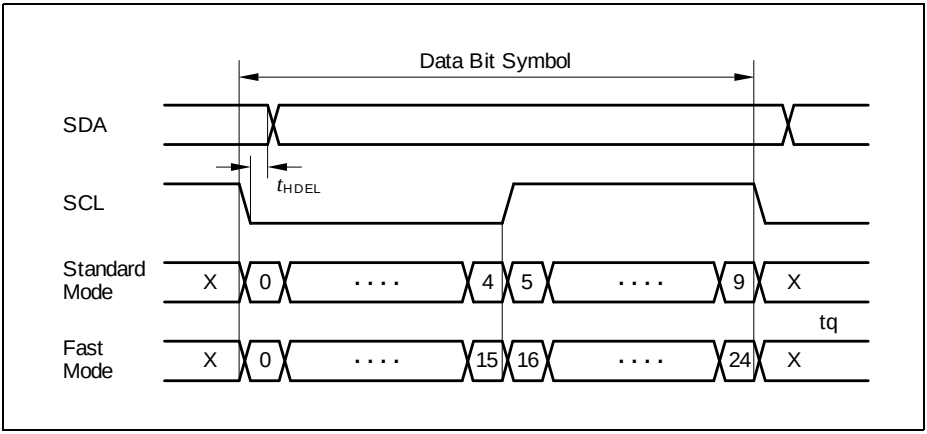


Figure 17-54 Data Bit Symbol

Output SDA changes after the time t_{HDEL} defined by PCR.HDEL has elapsed if a falling edge is detected at the SCL input to respect the SDA hold time. The value of PCR.HDEL allows compensation of the delay of the SCL input path (sampling, filtering).

In the case of an acknowledge transmission, the USIC IIC waits for the receiver indicating that a complete byte has been received. This adds an additional delay of 3 periods of f_{PB} to the path. The minimum module input frequency has to be selected properly to ensure the SDA setup time to SCL rising edge.

17.5.4 Data Flow Handling

The handling of the data flow and the sequence of the symbols in an IIC frame is controlled by the IIC transmitter part of the USIC communication channel. The IIC bus protocol is byte-oriented, whereas a USIC data buffer word can contain up to 16 data bits. In addition to the data byte to be transmitted (located at TBUF[7:0]), bit field TDF (transmit data format) to control the IIC sequence is located at the bit positions TBUF[10:8]. The TDF code defines for each data byte how it should be transmitted (IIC master or IIC slave), and controls the transmission of (repeated) start and stop symbols. This structure allows the definition of a complete IIC frame for an IIC master device only by writing to TBUFx or by using a FIFO data buffer mechanism, because no other control registers have to be accessed. Alternatively, polling of the ACK and NACK bits in PSR register can be performed, and the next data byte is transmitted only after an ACK is received.

If a wrong or unexpected TDF code is encountered (e.g. due to a software error during setup of the transmit buffer), a stop condition will be sent out by the master. This leads to an abort of the currently running frame. A slave module waits for a valid TDF code and sets SCL = 0. The software then has to invalidate the unexpected TDF code and write a valid one.

Please note that during an arbitration phase in multi-master bus systems an unpredictable bus behavior may occur due to an unexpected stop condition.

17.5.4.1 Transmit Data Formats

The following transmit data formats are available in master mode:

Table 17-12 Master Transmit Data Formats

TDF Code	Description
000_B	**Send data byte as master** This format is used to transmit a data byte from the master to a slave. The transmitter sends its data byte (TBUF[7:0]), receives and checks the acknowledge bit sent by the slave.
010_B	**Receive data byte and send acknowledge** This format is used by the master to read a data byte from a slave. The master acknowledges the transfer with a 0-level to continue the transfer. The content of TBUF[7:0] is ignored.

Table 17-12 Master Transmit Data Formats (cont'd)

TDF Code	Description
011_B	**Receive data byte and send not-acknowledge** This format is used by the master to read a data byte from a slave. The master does not acknowledge the transfer with a 1-level to finish the transfer. The content of TBUF[7:0] is ignored.
100_B	**Send start condition** If TBUF contains this entry while the bus is idle, a start condition will be generated. The content of TBUF[7:0] is taken as first address byte for the transmission (bits TBUF[7:1] are the address, the LSB is the read/write control).
101_B	**Send repeated start condition** If TBUF contains this entry and SCL = 0 and a byte transfer is not in progress, a repeated start condition will be sent out if the device is the current master. The current master is defined as the device that has set the start condition (and also won the master arbitration) for the current message. The content of TBUF[7:0] is taken as first address byte for the transmission (bits TBUF[7:1] are the address, the LSB is the read/write control).
110_B	**Send stop condition** If the current master has finished its last byte transfer (including acknowledge), it sends a stop condition if this format is in TBUF. The content of TBUF[7:0] is ignored.
111_B	**Reserved** This code must not be programmed. No additional action except releasing the TBUF entry and setting the error bit in PSR (that can lead to a protocol interrupt).

The following transmit data format is available in slave mode (the symbols in a frame are controlled by the master and the slave only has to send data if it has been "asked" by the master):

Table 17-13 Slave Transmit Data Format

TDF Code	Description
001_B	**Send data byte as slave** This format is used to transmit a data byte from a slave to the master. The transmitter sends its data byte (TBUF[7:0]) plus the acknowledge bit as a 1.

17.5.4.2 Valid Master Transmit Data Formats

Due to the IIC frame format definitions, only some specific sequences of TDF codes are possible and valid. If the USIC IIC module detects a wrong TDF code in a running frame, the transfer is aborted and flag PCR.WTDF is set. Additionally, an interrupt can be generated if enabled by the user. In case of a wrong TDF code, the frame will be aborted immediately with a STOP condition if the USIC IIC master still owns the SDA line. But if the accessed slave owns the SDA line (read transfer), the master must perform a dummy read with a non-acknowledge so that the slave releases the SDA line before a STOP condition can be sent. The received data byte of the dummy read will be stored in RBUF0/1, but RDV0/1 will not be set. Therefore the dummy read will not generate a receive interrupt and the data byte will not be stored into the receive FIFO.

If the transfer direction has changed in the current frame (master read access), the transmit data request (TDF = 000_B) is not possible and won't be accepted (leading to a wrong TDF Code indication).

Table 17-14 Valid TDF Codes Overview

Frame Position	Valid TDF Codes
First TDF code (master idle)	Start (100_B)
Read transfer: second TDF code (after start or repeated start)	Receive with acknowledge (010_B) or receive with not-acknowledge (011_B)
Write transfer: second TDF code (after start or repeated start)	Transmit (000_B), repeated start (101_B), or stop (110_B)
Read transfer: third and subsequent TDF code after acknowledge	Receive with acknowledge (010_B) or receive with not-acknowledge (011_B)
Read transfer: third and subsequent TDF code after not-acknowledge	Repeated start (101_B) or stop (110_B)
Write transfer: third and subsequent TDF code	Transmit (000_B), repeated start (101_B), or stop (110_B)

- First TDF code:
  A master transfer starts with the TDF start code (100_B). All other codes are ignored, but no WTDF error will be indicated.
- TDF code after a start (100_B) or repeated start code (101_B) in case of a read access:
  If a master-read transfer is started (determined by the LSB of the address byte = 1), the transfer direction of SDA changes and the slave will actively drive the data line. In this case, only the codes 010_B and 011_B are valid. To abort the transfer in case of a wrong code, a dummy read must be performed by the master before the STOP condition can be generated.

- TDF code after a start (100_B) or repeated start code (101_B) in case of a write access:
  If a master-write transfer is started (determined by the LSB of the address byte = 0), the master still owns the SDA line. In this case, the transmit (000_B), repeated start (101_B) and stop (110_B) codes are valid. The other codes are considered as wrong. To abort the transfer in case of a wrong code, the STOP condition is generated immediately.
- TDF code of the third and subsequent command in case of a read access with acknowledged previous data byte:
  If a master-read transfer is started (determined by the LSB of the address byte), the transfer direction of SDA changes and the slave will actively drive the data line. To force the slave to release the SDA line, the master has to not-acknowledge a byte transfer. In this case, only the receive codes 010_B and 011_B are valid. To abort the transfer in case of a wrong code, a dummy read must be performed by the master before the STOP condition can be generated.
- TDF code of the third and subsequent command in case of a read access with a not-acknowledged previous data byte:
  If a master-read transfer is started (determined by the LSB of the address byte), the transfer direction of SDA changes and the slave will actively drive the data line. To force the slave to release the SDA line, the master has to not-acknowledge a byte transfer. In this case, only the restart (101_B) and stop code (110_B) are valid. To abort the transfer in case of a wrong code, the STOP condition is generated immediately.
- TDF code of the third and subsequent command in case of a write access:
  If a master-write transfer is started (determined by the LSB of the address byte), the master still owns the SDA line. In this case, the transmit (000_B), repeated start (101_B) and stop (110_B) codes are valid. The other codes are considered as wrong. To abort the transfer in case of a wrong code, the STOP condition is generated immediately.
- After a master device has received a non-acknowledge from a slave device, a stop condition will be sent out automatically, except if the following TDF code requests a repeated start condition. In this case, the TDF code is taken into account, whereas all other TDF codes are ignored.

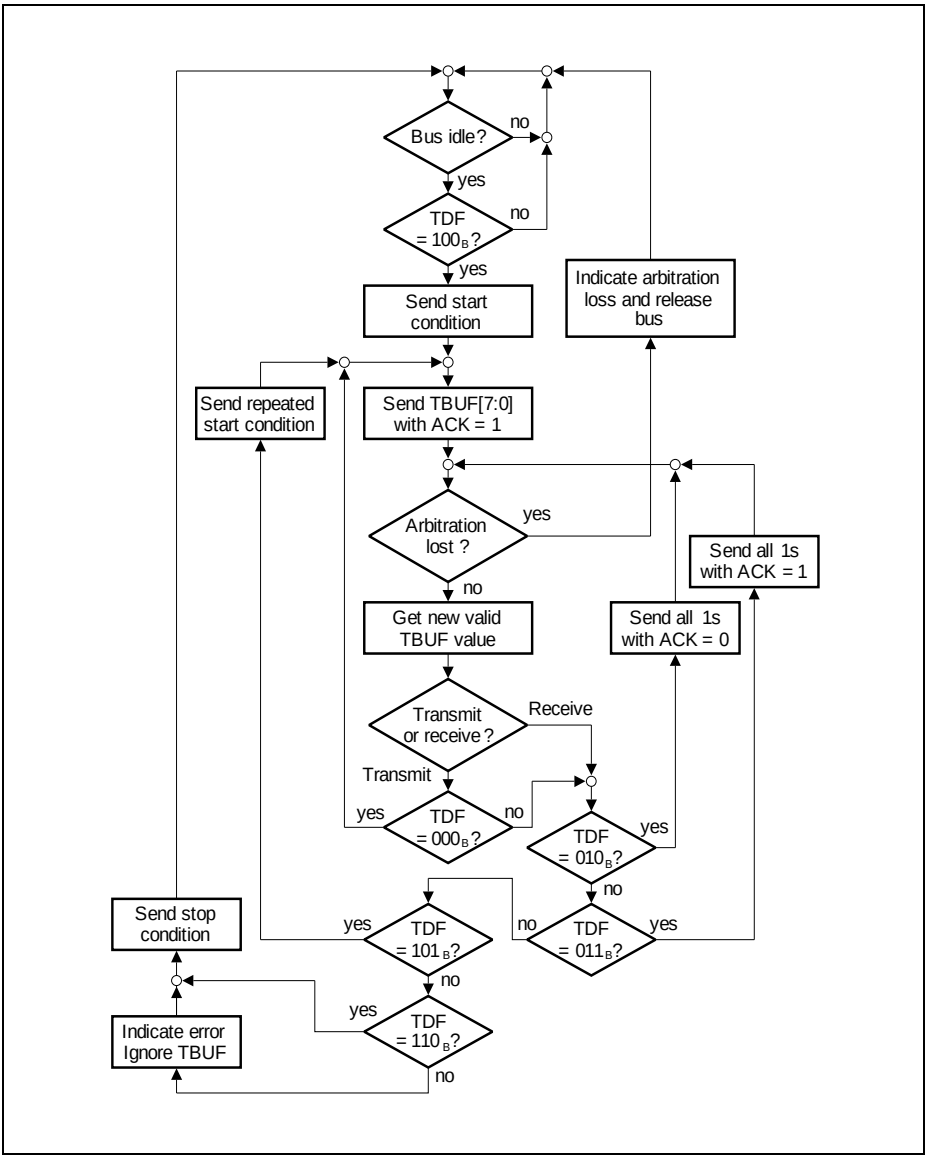


Figure 17-55 IIC Master Transmission

17.5.4.3 Master Transmit/Receive Modes

In master transmit mode, the IIC sends a number of data bytes to a slave receiver. The TDF code sequence for the master transmit mode is shown in **Table 17-15**.

Table 17-15 TDF Code Sequence for Master Transmit

TDF Code Sequence	TBUF[10:8] (TDF Code)	TBUF[7:0]	IIC Response	Interrupt Events
1st code	100_B	Slave address + write bit	Send START condition, slave address and write bit	SCR: Indicates a START condition is detected TBIF: Next word can be written to TBUF
2nd code	000_B	Data or 2nd slave address byte	Send data or 2nd slave address byte	TBIF: Next word can be written to TBUF
Subsequent codes for data transmit	000_B	Data	Send data	TBIF: Next word can be written to TBUF
Last code	110_B	Don't care	Send STOP condition	PCR: Indicates a STOP condition is detected

In master receive mode, the IIC receives a number of data bytes from a slave transmitter. The TDF code sequence for the master receive 7-bit and 10-bit addressing modes are shown in **Table 17-16** and **Table 17-17**.

Table 17-16 TDF Code Sequence for Master Receive (7-bit Addressing Mode)

TDF Code Sequence	TBUF[10:8] (TDF Code)	TBUF[7:0]	IIC Response	Interrupt Events
1st code	100_B	Slave address + read bit	Send START condition, slave address and read bit	SCR: Indicates a START condition is detected TBIF: Next word can be written to TBUF
2nd code	010_B	Don't care	Receive data and send ACK bit	TBIF: Next word can be written to TBUF AIF: First data received can be read

Table 17-16 TDF Code Sequence for Master Receive (7-bit Addressing Mode)

TDF Code Sequence	TBUF[10:8] (TDF Code)	TBUF[7:0]	IIC Response	Interrupt Events
Subsequent codes for data receive	010_B	Don't care	Receive data and send ACK bit	TBIF: Next word can be written to TBUF RIF: Subsequent data received can be read
Code for last data to be received	011_B	Don't care	Receive data and send NACK bit	TBIF: Next word can be written to TBUF RIF: Last data received can be read
Last code	110_B	Don't care	Send STOP condition	PCR: Indicates a STOP condition is detected

Table 17-17 TDF Code Sequence for Master Receive (10-bit Addressing Mode)

TDF Code Sequence	TBUF[10:8] (TDF Code)	TBUF[7:0]	IIC Response	Interrupt Events
1st code	100_B	Slave address (1st byte) + write bit	Send START condition, slave address (1st byte) and write bit	SCR: Indicates a START condition is detected TBIF: Next word can be written to TBUF
2nd code	000_B	Slave address (2nd byte)	Send address (2nd byte)	TBIF: Next word can be written to TBUF
3rd code	101_B	1st slave address + read bit	Send repeated START condition, slave address (1st byte) and read bit	RSCR: Indicates a repeated START condition is detected TBIF: Next word can be written to TBUF
4th code	010_B	Don't care	Receive data and send ACK bit	TBIF: Next word can be written to TBUF AIF: First data received can be read
Subsequent codes for data receive	010_B	Don't care	Receive data and send ACK bit	TBIF: Next word can be written to TBUF RIF: Subsequent data received can be read

Table 17-17 TDF Code Sequence for Master Receive (10-bit Addressing Mode)

TDF Code Sequence	TBUF[10:8] (TDF Code)	TBUF[7:0]	IIC Response	Interrupt Events
Code for last data to be received	011_B	Don't care	Receive data and send NACK bit	TBIF: Next word can be written to TBUF RIF: Last data received from slave can be read
Last code	110_B	Don't care	Send STOP condition	PCR: Indicates a STOP condition is detected

Figure 17-56 shows the interrupt events during the master transmit-slave receive and master receive/slave transmit sequences.

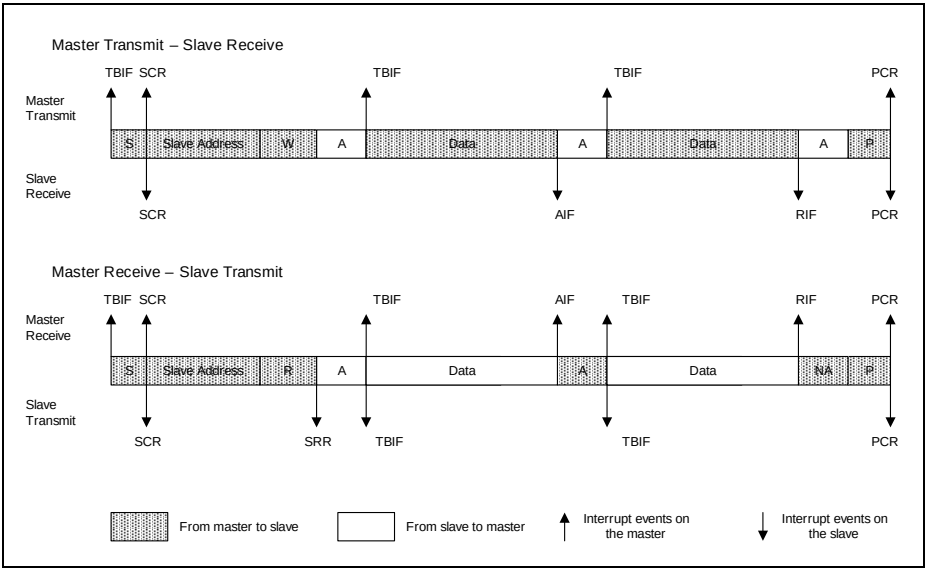


Figure 17-56 Interrupt Events on Data Transfers

17.5.4.4 Slave Transmit/Receive Modes

In slave receive mode, no TDF code needs to be written and data reception is indicated by the alternate receive (AIF) or receive (RIF) events.

In slave transmit mode, upon receiving its own slave address or general call address if this option is enabled, a slave read request event (SRR) will be triggered. The slave IIC then writes the TDF code 001_B and the requested data to TBUF to transmit the data to

the master. The slave does not check if the master reply with an ACK or NACK to the transmitted data.

In both cases, the data transfer is terminated by the master sending a STOP condition, which is indicated by a PCR event. See also **Figure 17-56**.

17.5.5 IIC Protocol Registers

In IIC mode, the registers PCR and PSR handle IIC related information.

17.5.5.1 IIC Protocol Control Registers

In IIC mode, the PCR register bits or bit fields are defined as described in this section.

PCR
Protocol Control Register [IIC Mode]

($3C_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MCLK	**ACKIEN**	**HDEL**				**SACKDIS**	**ERRIEN**	**SRRIEN**	**ARLIEN**	**NACKIEN**	**PCRIEN**	**RSCRIEN**	**SCRIEN**	**STIM**	**ACK00**
rw	rw	rw				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SLAD															
rw															

Field	Bits	Type	Description
SLAD	[15:0]	rw	**Slave Address** This bit field contains the programmed slave address. The corresponding bits in the first received address byte are compared to the bits SLAD[15:9] to check for address match. If SLAD[15:11] = 11110_B, then the second address byte is also compared to SLAD[7:0].
ACK00	16	rw	**Acknowledge 00_H** This bit defines if a slave device should be sensitive to the slave address 00_H. 0_B The slave device is not sensitive to this address. 1_B The slave device is sensitive to this address.

Field	Bits	Type	Description
STIM	17	rw	**Symbol Timing** This bit defines how many time quanta are used in a symbol. 0_B A symbol contains 10 time quanta. The timing is adapted for standard mode (100 kBaud). 1_B A symbol contains 25 time quanta. The timing is adapted for fast mode (400 kBaud).
SCRIEN	18	rw	**Start Condition Received Interrupt Enable** This bit enables the generation of a protocol interrupt if a start condition is detected. 0_B The start condition interrupt is disabled. 1_B The start condition interrupt is enabled.
RSCRIEN	19	rw	**Repeated Start Condition Received Interrupt Enable** This bit enables the generation of a protocol interrupt if a repeated start condition is detected. 0_B The repeated start condition interrupt is disabled. 1_B The repeated start condition interrupt is enabled.
PCRIEN	20	rw	**Stop Condition Received Interrupt Enable** This bit enables the generation of a protocol interrupt if a stop condition is detected. 0_B The stop condition interrupt is disabled. 1_B The stop condition interrupt is enabled.
NACKIEN	21	rw	**Non-Acknowledge Interrupt Enable** This bit enables the generation of a protocol interrupt if a non-acknowledge is detected by a master. 0_B The non-acknowledge interrupt is disabled. 1_B The non-acknowledge interrupt is enabled.
ARLIEN	22	rw	**Arbitration Lost Interrupt Enable** This bit enables the generation of a protocol interrupt if an arbitration lost event is detected. 0_B The arbitration lost interrupt is disabled. 1_B The arbitration lost interrupt is enabled.
SRRIEN	23	rw	**Slave Read Request Interrupt Enable** This bit enables the generation of a protocol interrupt if a slave read request is detected. 0_B The slave read request interrupt is disabled. 1_B The slave read request interrupt is enabled.

Field	Bits	Type	Description
ERRIEN	24	rw	**Error Interrupt Enable** This bit enables the generation of a protocol interrupt if an IIC error condition is detected (indicated by PSR.ERR or PSR.WTDF). 0_B The error interrupt is disabled. 1_B The error interrupt is enabled.
SACKDIS	25	rw	**Slave Acknowledge Disable** This bit disables the generation of an active acknowledge signal for a slave device (active acknowledge = 0 level). Once set by software, it is automatically cleared with each (repeated) start condition. If this bit is set after a byte has been received (indicated by an interrupt) but before the next acknowledge bit has started, the next acknowledge bit will be sent with passive level. This would indicate that the receiver does not accept more bytes. As a result, a minimum of 2 bytes will be received if the first receive interrupt is used to set this bit. 0_B The generation of an active slave acknowledge is enabled (slave acknowledge with 0 level = more bytes can be received). 1_B The generation of an active slave acknowledge is disabled (slave acknowledge with 1 level = reception stopped).
HDEL	[29:26]	rw	**Hardware Delay** This bit field defines the delay used to compensate the internal treatment of the SCL signal (see **Page 17-116**) in order to respect the SDA hold time specified for the IIC protocol.
ACKIEN	30	rw	**Acknowledge Interrupt Enable** This bit enables the generation of a protocol interrupt if an acknowledge is detected by a master. 0_B The acknowledge interrupt is disabled. 1_B The acknowledge interrupt is enabled.
MCLK	31	rw	**Master Clock Enable** This bit enables generation of the master clock MCLK (not directly used for IIC protocol, can be used as general frequency output). 0_B The MCLK generation is disabled and MCLK is 0. 1_B The MCLK generation is enabled.

17.5.5.2 IIC Protocol Status Register

The following PSR status bits or bit fields are available in IIC mode. Please note that the bits in register PSR are not cleared by hardware.

The flags in the PSR register can be cleared by writing a 1 to the corresponding bit position in register PSCR. Writing a 1 to a bit position in PSR sets the corresponding flag, but does not lead to further actions (no interrupt generation). Writing a 0 has no effect. These flags should be cleared by software before enabling a new protocol.

PSR
Protocol Status Register [IIC Mode] (48_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															BRGIF
r															rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AIF	RIF	TBIF	TSIF	DLIF	RSIF	ACK	ERR	SRR	ARL	NACK	PCR	RSCR	SCR	WTDF	SLSEL
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
SLSEL	0	rwh	**Slave Select** This bit indicates that this device has been selected as slave. 0_B The device is not selected as slave. 1_B The device is selected as slave.
WTDF	1	rwh	**Wrong TDF Code Found**[1)] This bit indicates that an unexpected/wrong TDF code has been found. A protocol interrupt can be generated if PCR.ERRIEN = 1. 0_B A wrong TDF code has not been found. 1_B A wrong TDF code has been found.
SCR	2	rwh	**Start Condition Received**[1)] This bit indicates that a start condition has been detected on the IIC bus lines.A protocol interrupt can be generated if PCR.SCRIEN = 1. 0_B A start condition has not yet been detected. 1_B A start condition has been detected.

Field	Bits	Type	Description
RSCR	3	rwh	**Repeated Start Condition Received[1)]** This bit indicates that a repeated start condition has been detected on the IIC bus lines. A protocol interrupt can be generated if PCR.RSCRIEN = 1. 0_B A repeated start condition has not yet been detected. 1_B A repeated start condition has been detected.
PCR	4	rwh	**Stop Condition Received[1)]** This bit indicates that a stop condition has been detected on the IIC bus lines. A protocol interrupt can be generated if PCR.PCRIEN = 1. 0_B A stop condition has not yet been detected. 1_B A stop condition has been detected.
NACK	5	rwh	**Non-Acknowledge Received[1)]** This bit indicates that a non-acknowledge has been received in master mode. This bit is not set in slave mode. A protocol interrupt can be generated if PCR.NACKIEN = 1. 0_B A non-acknowledge has not been received. 1_B A non-acknowledge has been received.
ARL	6	rwh	**Arbitration Lost[1)]** This bit indicates that an arbitration has been lost. A protocol interrupt can be generated if PCR.ARLIEN = 1. 0_B An arbitration has not been lost. 1_B An arbitration has been lost.
SRR	7	rwh	**Slave Read Request[1)]** This bit indicates that a slave read request has been detected. It becomes active to request the first data byte to be made available in the transmit buffer. For further consecutive data bytes, the transmit buffer issues more interrupts. For the end of the transfer, the master transmitter sends a stop condition. A protocol interrupt can be generated if PCR.SRRIEN = 1. 0_B A slave read request has not been detected. 1_B A slave read request has been detected.

Field	Bits	Type	Description
ERR	8	rwh	**Error**[1)] This bit indicates that an IIC error (frame format or TDF code) has been detected. A protocol interrupt can be generated if PCR.ERRIEN = 1. 0_B An IIC error has not been detected. 1_B An IIC error has been detected.
ACK	9	rwh	**Acknowledge Received**[1)] This bit indicates that an acknowledge has been received in master mode. This bit is not set in slave mode. A protocol interrupt can be generated if PCR.ACKIEN = 1. 0_B An acknowledge has not been received. 1_B An acknowledge has been received.
RSIF	10	rwh	**Receiver Start Indication Flag** 0_B A receiver start event has not occurred. 1_B A receiver start event has occurred.
DLIF	11	rwh	**Data Lost Indication Flag** 0_B A data lost event has not occurred. 1_B A data lost event has occurred.
TSIF	12	rwh	**Transmit Shift Indication Flag** 0_B A transmit shift event has not occurred. 1_B A transmit shift event has occurred.
TBIF	13	rwh	**Transmit Buffer Indication Flag** 0_B A transmit buffer event has not occurred. 1_B A transmit buffer event has occurred.
RIF	14	rwh	**Receive Indication Flag** 0_B A receive event has not occurred. 1_B A receive event has occurred.
AIF	15	rwh	**Alternative Receive Indication Flag** 0_B An alternative receive event has not occurred. 1_B An alternative receive event has occurred.
BRGIF	16	rwh	**Baud Rate Generator Indication Flag** 0_B A baud rate generator event has not occurred. 1_B A baud rate generator event has occurred.
0	[31:17]	r	**Reserved** Returns 0 if read; not modified in IIC mode.

1) This status bit can generate a protocol interrupt (see **Page 17-21**). The general interrupt status flags are described in the general interrupt chapter.

17.6 Inter-IC Sound Bus Protocol (IIS)

This chapter describes how the USIC module handles the IIS protocol. This serial protocol can handle reception and transmission of synchronous data frames between a device operating in master mode and a device in slave mode. An IIS connection based on a USIC communication channel supports half-duplex and full-duplex data transfers. The IIS mode is selected by CCR.MODE = 0011_B with CCFG.IIS = 1 (IIS mode is available).

17.6.1 Introduction

The IIS protocol is a synchronous serial communication protocol mainly for audio and infotainment applications **[18]**.

17.6.1.1 Signal Description

A connection between an IIS master and an IIS slave is based on the following signals:

- A shift clock signal SCK, generated by the transfer master. It is permanently generated while an IIS connection is established, also while no valid data bits are transferred.
- A word address signal WA (also named WS), generated by the transfer master. It indicates the beginning of a new data word and the targeted audio channel (e.g. left/right). The word address output signal WA is available on all SELOx outputs if the WA generation is enabled (by PCR.WAGEN = 1 for the transfer master). The WA signal changes synchronously to the falling edges of the shift clock.
- If the transmitter is the IIS master device, it generates a master transmit slave receive data signal. The data changes synchronously to the falling edges of the shift clock.
- If the transmitter is the IIS slave device, it generates a master receive slave transmit data signal. The data changes synchronously to the falling edges of the shift clock.

The transmitter part and the receiver part of the USIC communication channel can be used together to establish a full-duplex data connection between an IIS master and a slave device.

Table 17-18 IIS IO Signals

IIS Mode	Receive Data	Transmit Data	Shift Clock	Word Address
Master	Input DIN0, handled by DX0	Output DOUT0	Output SCLKOUT	Output(s) SELOx
Slave	Input DIN0, handled by DX0	Output DOUT0	Input SCLKIN, handled by DX1	Input SELIN, handled by DX2

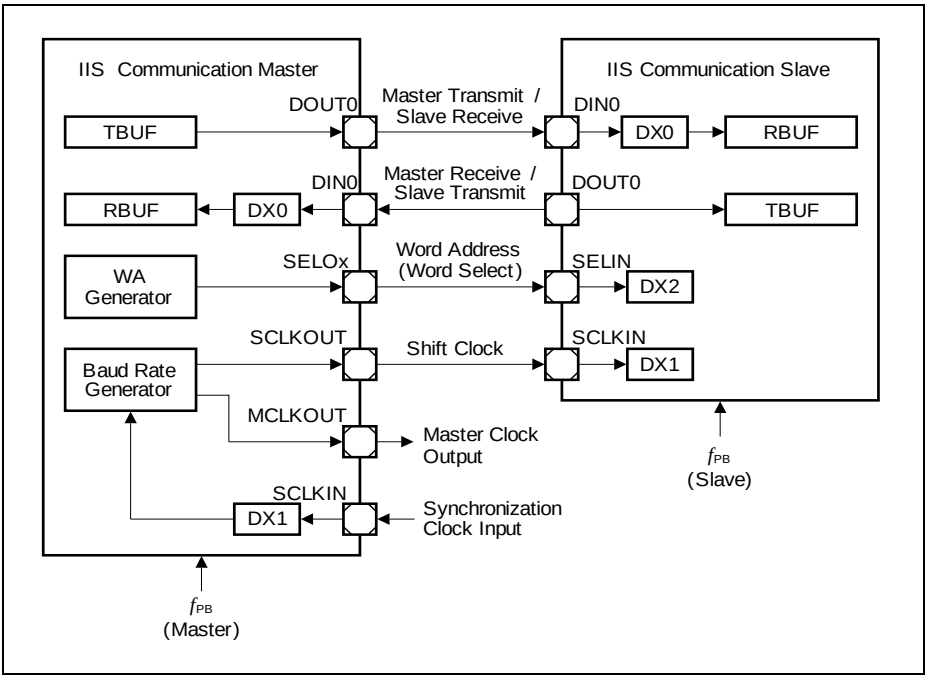


Figure 17-57 IIS Signals

Two additional signals are available for the USIC IIS communication master:

- A master clock output signal MCLKOUT with a fixed phase relation to the shift clock to support oversampling for audio components. It can also be used as master clock output of a communication network with synchronized IIS connections.
- A synchronization clock input SCLKIN for synchronization of the shift clock generation to an external frequency to support audio frequencies that can not be directly derived from the system clock f_{PB} of the communication master. It can be used as master clock input of a communication network with synchronized IIS connections.

17.6.1.2 Protocol Overview

An IIS connection supports transfers for two different data frames via the same data line, e.g. a data frames for the left audio channel and a data frame for the right audio channel. The word address signal WA is used to distinguish between the different data frames. Each data frame can consist of several data words.

In a USIC communication channel, data words are tagged for being transmitted for the left or for the right channel. Also the received data words contain a tag identifying the WA state when the data has been received.

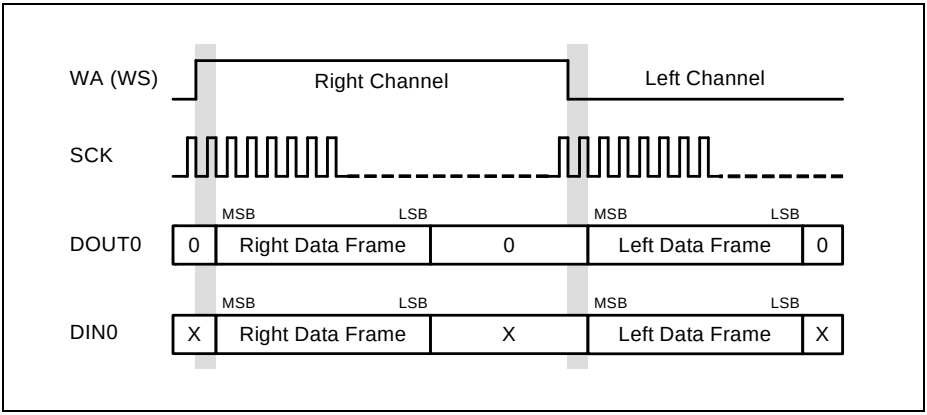


Figure 17-58 Protocol Overview

17.6.1.3 Transfer Delay

The transfer delay feature allows the transfer of data (transmission and reception) with a programmable delay (counted in shift clock periods).

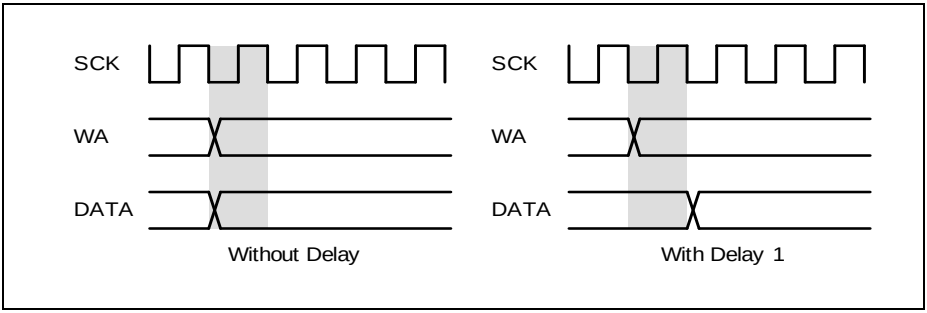


Figure 17-59 Transfer Delay for IIS

17.6.1.4 Connection of External Audio Components

The IIS signals can be used to communicate with external audio devices (such as Codecs) or other audio data sources/destinations.

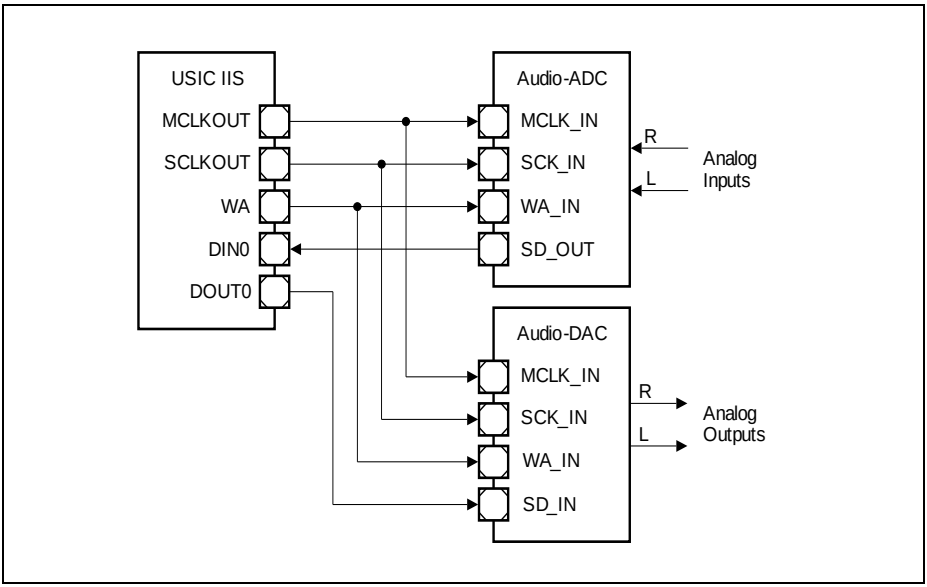


Figure 17-60 Connection of External Audio Devices

In some applications, especially for Audio-ADCs or Audio-DACs, a master clock signal is required with a fixed phase relation to the shift clock signal. The frequency of MCLKOUT is a multiple of the shift frequency SCLKOUT. This factor defines the oversampling factor of the external device (commonly used values: 256 or 384).

17.6.2 Operating the IIS

This chapter contains IIS issues, that are of general interest and not directly linked to master mode or slave mode.

17.6.2.1 Frame Length and Word Length Configuration

After each change of the WA signal, a complete data frame is intended to be transferred (frame length ≤ system word length). The number of data bits transferred after a change of signal WA is defined by SCTR.FLE. A data frame can consist of several data words with a data word length defined by SCTR.WLE. The changes of signal WA define the system word length as the number of SCLK cycles between two changes of WA (number of bits available for the right channel and same number available for the left channel).

If the system word length is longer than the frame length defined by SCTR.FLE, the additional bits are transmitted with passive data level (SCTR.PDL). If the system word

length is smaller than the device frame length, not all LSBs of the transmit data can be transferred.

It is recommended to program bits WLEMD, FLEMD and SELMD in register TCSR to 0.

17.6.2.2 Automatic Shadow Mechanism

The baud rate and shift control setting are internally kept constant while a data frame is transferred by an automatic shadow mechanism. The registers can be programmed all the time with new settings that are taken into account for the next data frame. During a data frame, the applied (shadowed) setting is not changed, although new values have been written after the start of the data frame. The setting is internally “frozen” with the start of each data frame.

Although this shadow mechanism being implemented, it is recommended to change the baud rate and shift control setting only while the IIS protocol is switched off.

17.6.2.3 Mode Control Behavior

In IIS mode, the following kernel modes are supported:

- Run Mode 0/1:
  Behavior as programmed, no impact on data transfers.
- Stop Mode 0/1:
  Bit PCR.WAGEN is internally considered as 0 (the bit itself is not changed). If WAGEN = 1, then the current system word cycle is finished and then the WA generation is stopped, but PSR.END is not set. The complete data frame is finished before entering stop mode, including a possible delay due to PCR.TDEL.
  When leaving a stop mode with WAGEN = 1, the WA generation starts from the beginning.

17.6.2.4 Transfer Delay

The transfer delay can be used to synchronize a data transfer to an event (e.g. a change of the WA signal). This event has to be synchronously generated to the falling edge of the shift clock SCK (like the change of the transmit data), because the input signal for the event is directly sampled in the receiver (as a result, the transmitter can use the detection information with its next edge).

Event signals that are asynchronous to the shift clock while the shift clock is running must not be used. In the example in **Figure 17-59**, the event (change of signal WA) is generated by the transfer master and as a result, is synchronous to the shift clock SCK. With the rising edge of SCK, signal WA is sampled and checked for a change. If a change is detected, a transfer delay counter TDC is automatically loaded with its programmable reload value (PCR.TDEL), otherwise it is decremented with each rising edge of SCK until it reaches 0, where it stops. The transfer itself is started if the value of TDC has become 0. This can happen under two conditions:

- TDC is reloaded with a PCR.TDEL = 0 when the event is detected
- TDC has reached 0 while counting down

The transfer delay counter is internal to the IIS protocol pre-processor and can not be observed by software. The transfer delay in SCK cycles is given by PCR.TDEL+1.

In the example in **Figure 17-61**, the reload value PCR.TDEL for TDC is 0. When the samples taken on receiver side show the change of the WA signal, the counter TDC is reloaded. If the reload value is 0, the data transfer starts with 1 shift clock cycle delay compared to the change of WA.

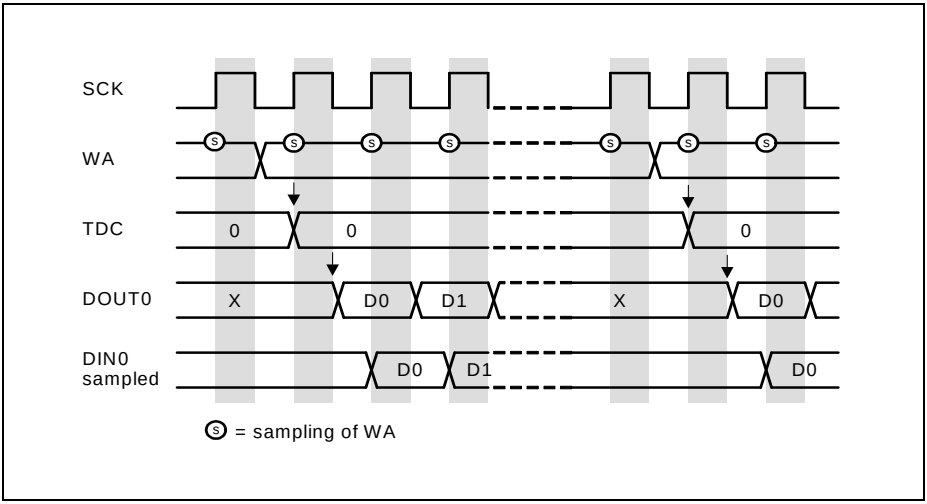


Figure 17-61 Transfer Delay with Delay 1

The ideal case without any transfer delay is shown in **Figure 17-62**. The WA signal changes and the data output value become valid at the same time. This implies that the transmitter “knows” in advance that the event signal will change with the next rising edge of TCLK. This is achieved by delaying the data transmission after the previously detected WA change the system word length minus 1.

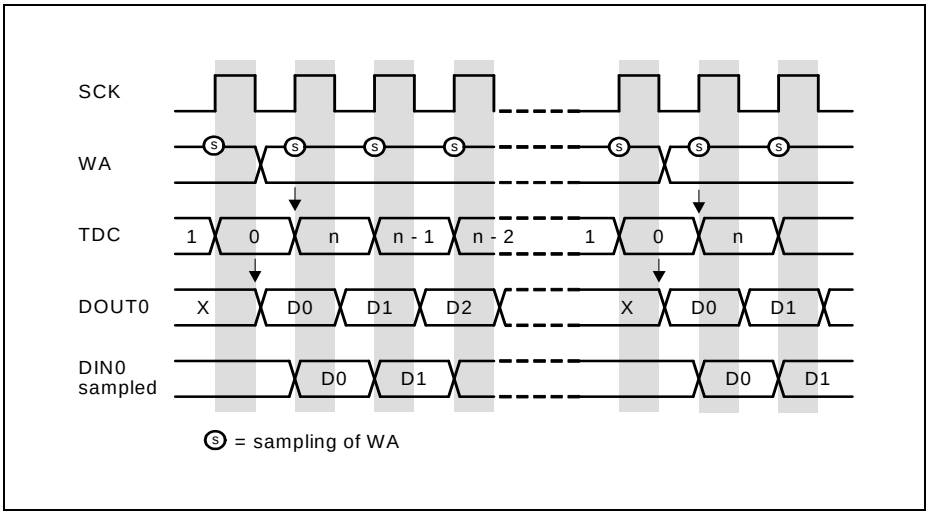


Figure 17-62 No Transfer Delay

If the end of the transfer delay is detected simultaneously to change of WA, the transfer is started and the delay counter is reloaded with PCR.TDEL. This allows to run the USIC as IIS device without any delay. In this case, internally the delay from the previous event elapses just at the moment when a new event occurs. If PCR.TDEL is set to a value bigger than the system word length, no transfer takes place.

17.6.2.5 Parity Mode

Parity generation is not supported in IIS mode and bit field CCR.PM = 00_B has to be programmed.

17.6.2.6 Transfer Mode

In IIS mode, bit field SCTR.TRM = 11_B has to be programmed to allow data transfers. Setting SCTR.TRM = 00_B disables and stops the data transfer immediately.

17.6.2.7 Data Transfer Interrupt Handling

The data transfer interrupts indicate events related to IIS frame handling.

- Transmit buffer interrupt TBI:
  Bit PSR.TBIF is set after the start of first data bit of a data word.
- Transmit shift interrupt TSI:
  Bit PSR.TSIF is set after the start of the last data bit of a data word.

- Receiver start interrupt RSI:
  Bit PSR.RSIF is set after the reception of the first data bit of a data word.
  With this event, bit TCSR.TDV is cleared and new data can be loaded to the transmit buffer.
- Receiver interrupt RI and alternative interrupt AI:
  Bit PSR.RIF is set at after the reception of the last data bit of a data word with WA = 0.
  Bit RBUFSR.SOF indicates whether the received data word has been the first data word of a new data frame.
  Bit PSR.AIF is set at after the reception of the last data bit of a data word with WA = 1.
  Bit RBUFSR.SOF indicates whether the received data word has been the first data word of a new data frame.

17.6.2.8 Baud Rate Generator Interrupt Handling

The baud rate generator interrupt indicate that the capture mode timer has reached its maximum value. With this event, the bit PSR.BRGIF is set.

17.6.2.9 Protocol-Related Argument and Error

In order to distinguish between data words received for the left or the right channel, the IIS protocol pre-processor samples the level of the WA input (just after the WA transition) and propagates it as protocol-related error (although it is not an error, but an indication) to the receive buffer status register at the bit position RBUFSR[9]. This bit position defines if either a standard receive interrupt (if RBUFSR[9] = 0) or an alternative receive interrupt (if RBUFSR[9] = 1) becomes activated when a new data word has been received. Incoming data can be handled by different interrupts or DMA mechanisms for the left and the right channel if the corresponding events are directed to different interrupt nodes. Flag PAR is always 0.

17.6.2.10 Transmit Data Handling

The IIS protocol pre-processor allows to distinguish between the left and the right channel for data transmission. Therefore, bit TCSR.WA indicates on which channel the data in the buffer will be transmitted. If TCSR.WA = 0, the data will be transmitted after a falling edge of WA. If TCSR.WA = 1, the data will be transmitted after a rising edge of WA. The WA value sampled after the WA transition is considered to distinguish between both channels (referring to PSR.WA).

Bit TCSR.WA can be automatically updated by the transmit control information TCI[4] for each data word if TCSR.WAMD = 1. In this case, data written to TBUF[15:0] (or IN[15:0] if a FIFO data buffer is used) is considered as left channel data, whereas data written to TBUF[31:16] (or IN[31:16] if a FIFO data buffer is used) is considered as right channel data.

17.6.2.11 Receive Buffer Handling

If a receive FIFO buffer is available (CCFG.RB = 1) and enabled for data handling (RBCTR.SIZE > 0), it is recommended to set RBCTR.RCIM = 11_B in IIS mode. This leads to an indication that the data word has been the first data word of a new data frame if bit OUTR.RCI[0] = 1, and the channel indication by the sampled WA value is given by OUTR.RCI[4].

The standard receive buffer event and the alternative receive buffer event can be used for the following operation in RCI mode (RBCTR.RNM = 1):

- A standard receive buffer event indicates that a data word can be read from OUTR that belongs to a data frame started when WA = 0.
- An alternative receive buffer event indicates that a data word can be read from OUTR that belongs to a data frame started when WA = 1.

17.6.2.12 Loop-Delay Compensation

The synchronous signaling mechanism of the IIS protocol being similar to the one of the SSC protocol, the closed-loop delay has to be taken into account for the application setup. In IIS mode, loop-delay compensation in master mode is also possible to achieve higher baud rates.

Please refer to the more detailed description in the SSC chapter.

17.6.3 Operating the IIS in Master Mode

In order to operate the IIS in master mode, the following issues have to be considered:

- Select IIS mode:
  It is recommended to configure all parameters of the IIS that do not change during run time while CCR.MODE = 0000_B. Bit field SCTR.TRM = 11_B has to be programmed. The configuration of the input stages has to be done while CCR.MODE = 0000_B to avoid unintended edges of the input signals and the IIS mode can be enabled by CCR.MODE = 0011_B afterwards.
- Pin connection for data transfer:
  Establish a connection of input stage DX0 with the selected receive data input pin (DIN0) with DX0CR.INSW = 1. Configure a transmit data output pin (DOUT0) for a transmitter.
  The data shift unit allowing full-duplex data transfers based on the same WA signal, the values delivered by the DX0 stage are considered as data bits (receive function can not be disabled independently from the transmitter). To receive IIS data, the transmitter does not necessarily need to be configured (no assignment of DOUT0 signal to a pin).
- Baud rate generation:
  The desired baud rate setting has to be selected, comprising the fractional divider and the baud rate generator. Bit DX1CR.INSW = 0 has to be programmed to use the

baud rate generator output SCLK directly as input for the data shift unit. Configure a shift clock output pin with the inverted signal SCLKOUT without additional delay (BRG.SCLKCFG = 01_B).

- Word address WA generation:
  The WA generation has to be enabled by setting PCR.WAGEN = 1 and the programming of the number of shift clock cycles between the changes of WA. Bit DX2CR.INSW = 0 has to be programmed to use the WA generator as input for the data shift unit. Configure WA output pin for signal SELOx if needed.
- Data format configuration:
  The word length, the frame length, and the shift direction have to be set up according to the application requirements by programming the register SCTR. Generally, the MSB is shifted first (SCTR.SDIR = 1).
  Bit TCSR.WAMD can be set to use the transmit control information TCI[4] to distinguish the data words for transmission while WA = 0 or while WA = 1.

17.6.3.1 Baud Rate Generation

The baud rate is defined by the frequency of the SCLK signal (one period of f_{SCLK} represents one data bit).

If the fractional divider mode is used to generate f_{PIN}, there can be an uncertainty of one period of f_{PB} for f_{PIN}. This uncertainty does not accumulate over several SCLK cycles. As a consequence, the average frequency is reached, whereas the duty cycle of 50% of the SCLK and MCLK signals can vary by one period of f_{PB}.

In IIS applications, where the phase relation between the optional MCLK output signal and SCLK is not relevant, SCLK can be based on the frequency f_{PIN} (BRG.PPPEN = 0). In the case that a fixed phase relation between the MCLK signal and SCLK is required (e.g. when using MCLK as clock reference for external devices), the additional divider by 2 stage has to be taken into account (BRG.PPPEN = 1). This division is due to the fact that signal MCLK toggles with each cycle of f_{PIN}. Signal SCLK is then based on signal MCLK, see **Figure 17-63**.

The adjustable integer divider factor is defined by bit field BRG.PDIV.

$$f_{SCLK} = \frac{f_{PIN}}{2} \times \frac{1}{PDIV + 1} \quad \text{if PPPEN = 0}$$
$$f_{SCLK} = \frac{f_{PIN}}{2 \times 2} \times \frac{1}{PDIV + 1} \quad \text{if PPPEN = 1} \qquad (17.12)$$

Note: In the IIS protocol, the master (unit generating the shift clock and the WA signal) changes the status of its data and WA output line with the falling edge of SCK. The slave transmitter also has to transmit on falling edges. The sampling of the received data is done with the rising edges of SCLK. The input stage DX1 and the

SCLKOUT have to be programmed to invert the shift clock signal to fit to the internal signals.

17.6.3.2 WA Generation

The word address (or word select) line WA regularly toggles after N cycles of signal SCLK. The time between the changes of WA is called system word length and can be programmed by using the following bit fields.

In IIS master mode, the system word length is defined by:

- BRG.CTQSEL = 10_B
  to base the WA toggling on SCLK
- BRG.PCTQ
  to define the number N of SCLK cycles per system word length
- BRG.DCTQ
  to define the number N of SCLK cycles per system word length

$$N = (PCTQ + 1) \times (DCTQ + 1) \quad (17.13)$$

17.6.3.3 Master Clock Output

The master clock signal MCLK can be generated by the master of the IIS transfer (BRG.PPPEN = 1). It is used especially to connect external Codec devices. It can be configured by bit BRG.MCLKCFG in its polarity to become the output signal MCLKOUT.

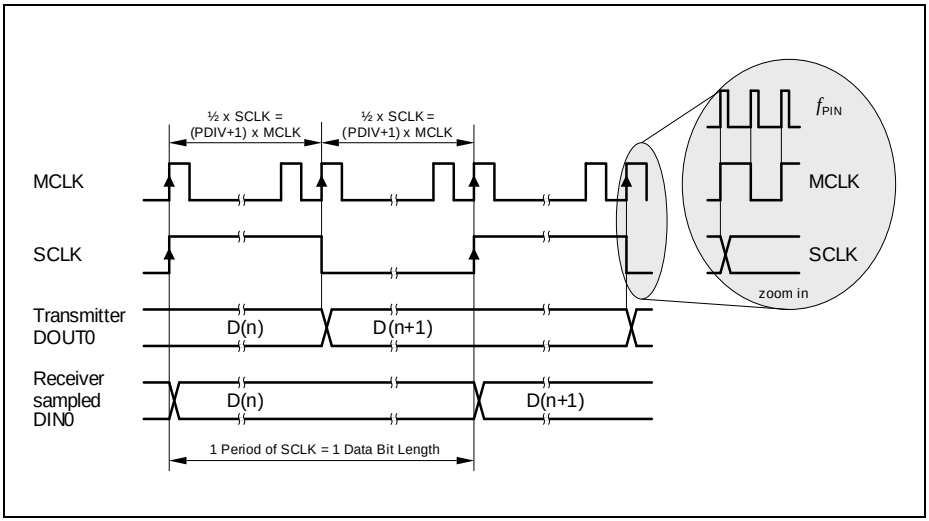


Figure 17-63 MCLK and SCLK for IIS

17.6.3.4 Protocol Interrupt Events

The following protocol-related events are generated in IIS mode and can lead to a protocol interrupt.

Please note that the bits in register PSR are not all automatically cleared by hardware and have to be cleared by software in order to monitor new incoming events.

- WA rising/falling edge events:
  The WA generation block indicates two events that are monitored in register PSR. Flag PSR.WAFE is set with the falling edge, flag PSR.WARE with the rising edge of the WA signal. A protocol interrupt can be generated if PCR.WAFEIEN = 1 for the falling edge, similar for PCR.WAREIEN = 1 for a rising edge.
- WA end event:
  The WA generation block also indicates when it has stopped the WA generation after it has been disabled by writing PCR.WAGEN = 0. A protocol interrupt can be generated if PCR.ENDIEN = 1.
- DX2T event:
  An activation of the trigger signal DX2T is indicated by PSR.DX2TEV = 1 and can generate a protocol interrupt if PCR.DX2TIEN = 1. This event can be evaluated instead of the WA rising/falling events if a delay compensation like in SSC mode (for details, refer to corresponding SSC section) is used.

17.6.4 Operating the IIS in Slave Mode

In order to operate the IIS in slave mode, the following issues have to be considered:

- Select IIS mode:
  It is recommended to configure all parameters of the IIS that do not change during run time while CCR.MODE = 0000_B. Bit field SCTR.TRM = 11_B has to be programmed. The configuration of the input stages has to be done while CCR.MODE = 0000_B to avoid unintended edges of the input signals and the IIS mode can be enabled by CCR.MODE = 0011_B afterwards.
- Pin connection for data transfer:
  Establish a connection of input stage DX0 with the selected receive data input pin (DIN0) with DX0CR.INSW = 1. Configure a transmit data output pin (DOUT0) for a transmitter.
  The data shift unit allowing full-duplex data transfers based on the same WA signal, the values delivered by the DX0 stage are considered as data bits (receive function can not be disabled independently from the transmitter). To receive IIS data, the transmitter does not necessarily need to be configured (no assignment of DOUT0 signal to a pin).
- Pin connection for shift clock:
  Establish a connection of input stage DX1 with the selected shift clock input pin (SCLKIN) with DX1CR.INSW = 1 and with inverted polarity (DX1CR.DPOL = 1).

- Pin connection for WA input:
  Establish a connection of input stage DX2 with the WA input pin (SELIN) with DX2CR.INSW = 1.
- Baud rate generation:
  The baud rate generator is not needed and can be switched off by the fractional divider.
- WA generation:
  The WA generation is not needed and can be switched off (PCR.WAGEN = 0).

17.6.4.1 Protocol Events and Interrupts

The following protocol-related event is generated in IIS mode and can lead to a protocol interrupt.

Please note that the bits in register PSR are not all automatically cleared by hardware and have to be cleared by software in order to monitor new incoming events.

- WA rising/falling/end events:
  The WA generation being switched off, these events are not available.
- DX2T event:
  An activation of the trigger signal DX2T is indicated by PSR.DX2TEV = 1 and can generate a protocol interrupt if PCR.DX2TIEN = 1.

17.6.5 IIS Protocol Registers

In IIS mode, the registers PCR and PSR handle IIS related information.

17.6.5.1 IIS Protocol Control Registers

In IIS mode, the PCR register bits or bit fields are defined as described in this section.

PCR
Protocol Control Register [IIS Mode]

($3C_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MCLK	0									TDEL					
rw	rw									rw					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DX2TIEN	0								ENDIEN	WAREIEN	WAFEIEN	0	SELINV	DTEN	WAGEN
rw	rw								rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
WAGEN	0	rw	**WA Generation Enable** This bit enables/disables the generation of word address control output signal WA. 0_B The IIS can be used as slave. The generation of the word address signal is disabled. The output signal WA is 0. The MCLKO signal generation depends on PCR.MCLK. 1_B The IIS can be used as master. The generation of the word address signal is enabled. The signal starts with a 0 after being enabled. The generation of MCLK is enabled, independent of PCR.MCLK. After clearing WAGEN, the USIC module stops the generation of the WA signal within the next 4 WA periods.
DTEN	1	rw	**Data Transfers Enable** This bit enables/disables the transfer of IIS frames as a reaction to changes of the input word address control line WA. 0_B The changes of the WA input signal are ignored and no transfers take place. 1_B Transfers are enabled.
SELINV	2	rw	**Select Inversion** This bit defines if the polarity of the SELOx outputs in relation to the internally generated word address signal WA. 0_B The SELOx outputs have the same polarity as the WA signal. 1_B The SELOx outputs have the inverted polarity to the WA signal.
WAFEIEN	4	rw	**WA Falling Edge Interrupt Enable** This bit enables/disables the activation of a protocol interrupt when a falling edge of WA has been generated. 0_B A protocol interrupt is not activated if a falling edge of WA is generated. 1_B A protocol interrupt is activated if a falling edge of WA is generated.

Field	Bits	Type	Description
WAREIEN	5	rw	**WA Rising Edge Interrupt Enable** This bit enables/disables the activation of a protocol interrupt when a rising edge of WA has been generated. 0_B A protocol interrupt is not activated if a rising edge of WA is generated. 1_B A protocol interrupt is activated if a rising edge of WA is generated.
ENDIEN	6	rw	**END Interrupt Enable** This bit enables/disables the activation of a protocol interrupt when the WA generation stops after clearing PCR.WAGEN (complete system word length is processed before stopping). 0_B A protocol interrupt is not activated. 1_B A protocol interrupt is activated.
DX2TIEN	15	rw	**DX2T Interrupt Enable** This bit enables/disables the generation of a protocol interrupt if the DX2T signal becomes activated (indicated by PSR.DX2TEV = 1). 0_B A protocol interrupt is not generated if DX2T is active. 1_B A protocol interrupt is generated if DX2T is active.
TDEL	[21:16]	rw	**Transfer Delay** This bit field defines the transfer delay when an event is detected. If bit field TDEL = 0, the additional delay functionality is switched off and a delay of one shift clock cycle is introduced.
MCLK	31	rw	**Master Clock Enable** This bit enables generation of the master clock MCLK (not directly used for IIC protocol, can be used as general frequency output). 0_B The MCLK generation is disabled and MCLK is 0. 1_B The MCLK generation is enabled.
0	3, [14:7], [30:22]	rw	**Reserved** Returns 0 if read; should be written with 0;

17.6.5.2 IIS Protocol Status Register

The following PSR status bits or bit fields are available in IIS mode. Please note that the bits in register PSR are not cleared by hardware.

The flags in the PSR register can be cleared by writing a 1 to the corresponding bit position in register PSCR. Writing a 1 to a bit position in PSR sets the corresponding flag, but does not lead to further actions (no interrupt generation). Writing a 0 has no effect. These flags should be cleared by software before enabling a new protocol.

PSR
Protocol Status Register [IIS Mode] (48_H) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
							0								BRGIF
							r								rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AIF	RIF	TBIF	TSIF	DLIF	RSIF		0		END	WARE	WAFE	DX2TEV	0	DX2S	WA
rwh	rwh	rwh	rwh	rwh	rwh		r		rwh	rwh	rwh	rwh	r	rwh	rwh

Field	Bits	Type	Description
WA	0	rwh	**Word Address** This bit indicates the status of the WA input signal, sampled after a transition of WA has been detected. This information is forwarded to the corresponding bit position RBUFSR[9] to distinguish between data received for the right and the left channel. 0_B WA has been sampled 0. 1_B WA has been sampled 1.
DX2S	1	rwh	**DX2S Status** This bit indicates the current status of the DX2S signal, which is used as word address signal WA. 0_B DX2S is 0. 1_B DX2S is 1.
DX2TEV	3	rwh	**DX2T Event Detected**[1)] This bit indicates that the DX2T signal has been activated. In IIS slave mode, an activation of DX2T generates a protocol interrupt if PCR.DX2TIEN = 1. 0_B The DX2T signal has not been activated. 1_B The DX2T signal has been activated.

Field	Bits	Type	Description
WAFE	4	rwh	**WA Falling Edge Event[1)]** This bit indicates that a falling edge of the WA output signal has been generated. This event generates a protocol interrupt if PCR.WAFEIEN = 1. 0_B A WA falling edge has not been generated. 1_B A WA falling edge has been generated.
WARE	5	rwh	**WA Rising Edge Event[1)]** This bit indicates that a rising edge of the WA output signal has been generated. This event generates a protocol interrupt if PCR.WAREIEN = 1. 0_B A WA rising edge has not been generated. 1_B A WA rising edge has been generated.
END	6	rwh	**WA Generation End[1)]** This bit indicates that the WA generation has ended after clearing PCR.WAGEN. This bit should be cleared by software before clearing WAGEN. 0_B The WA generation has not yet ended (if it is running and WAGEN has been cleared). 1_B The WA generation has ended (if it has been running).
RSIF	10	rwh	**Receiver Start Indication Flag** 0_B A receiver start event has not occurred. 1_B A receiver start event has occurred.
DLIF	11	rwh	**Data Lost Indication Flag** 0_B A data lost event has not occurred. 1_B A data lost event has occurred.
TSIF	12	rwh	**Transmit Shift Indication Flag** 0_B A transmit shift event has not occurred. 1_B A transmit shift event has occurred.
TBIF	13	rwh	**Transmit Buffer Indication Flag** 0_B A transmit buffer event has not occurred. 1_B A transmit buffer event has occurred.
RIF	14	rwh	**Receive Indication Flag** 0_B A receive event has not occurred. 1_B A receive event has occurred.
AIF	15	rwh	**Alternative Receive Indication Flag** 0_B An alternative receive event has not occurred. 1_B An alternative receive event has occurred.

Field	Bits	Type	Description
BRGIF	16	rwh	**Baud Rate Generator Indication Flag** 0_B A baud rate generator event has not occurred. 1_B A baud rate generator event has occurred.
0	2, [9:7], [31:17]	r	**Reserved** Returns 0 if read; not modified in IIS mode.

1) This status bit can generate a protocol interrupt (see **Page 17-21**). The general interrupt status flags are described in the general interrupt chapter.

17.7 Service Request Generation

The USIC module provides 6 service request outputs SR[5:0] to be shared between two channels. The service request outputs SR[5:0] are connected to interrupt nodes in the Nested Vectored Interrupt Controller (NVIC). Additionally, the first 4 outputs, SR[3:0], are also connected to the DMA Line Router (DLR). For details of the interrupt node and DMA line assignments, refer to the respective chapter in the specification.

Each USIC communication channel can be connected to up to 6 service request handlers (connected to USICx_CHy.SR[5:0], though 3 or 4 are normally used, e.g. one for transmission, one for reception, one or two for protocol or error handling, or for the alternative receive events).

17.8 Debug Behaviour

Each USIC communication channel can be pre-configured to enter one of four kernel modes, when the program execution of the CPU is halted by the debugger.

Refer to **Section 17.2.2.2** for details.

17.9 Power, Reset and Clock

The USIC module is located in the core power domain. The module, including all registers other than the bit field KSCFG.SUMCFG, can be reset to its default state by a system reset or a software reset triggered through the setting of corresponding bits in PRSETx registers. The bit field KSCFG.SUMCFG is reset to its default value only by a debug reset.

The USIC module is clocked by the Peripheral Bus clock (f_{PB}). If the module clock is disabled by KSCFG.MODEN = 0, the module cannot be accessed by read or write operations (except register KSCFG that can always be accessed).

17.10 Initialization and System Dependencies

The USIC module is held in reset after a start-up from a system or software reset. Therefore, the application has to apply the following initialization sequence before operating the USIC module:

- Release reset of USIC module by writing a 1 to the USICxRS bit in SCU_PRCLR0 or SCU_PRCLR1 registers
- Enable the module by writing 1s to the MODEN and BPMODEN bits in KSCFG register.

17.11 Registers

Table 17-19 shows all registers which are required for programming a USIC channel, as well as the FIFO buffer. It summarizes the USIC communication channel registers and defines the relative addresses and the reset values.

Please note that all registers can be accessed with any access width (8-bit, 16-bit, 32-bit), independent of the described width.

All USIC registers (except bit field KSCFG.SUMCFG) are always reset by a system reset. Bit field KSCFG.SUMCFG is reset by a debug reset.

Note: The register bits marked "w" always deliver 0 when read. They are used to modify flip-flops in other registers or to trigger internal actions.

Figure 17-64 shows the register types of the USIC module registers and channel registers. In a specific microcontroller, module registers of USIC module "x" are marked by the module prefix "USICx_". Channel registers of USIC module "x" are marked by the channel prefix "USICx_CH0_" and "USICx_CH1_".

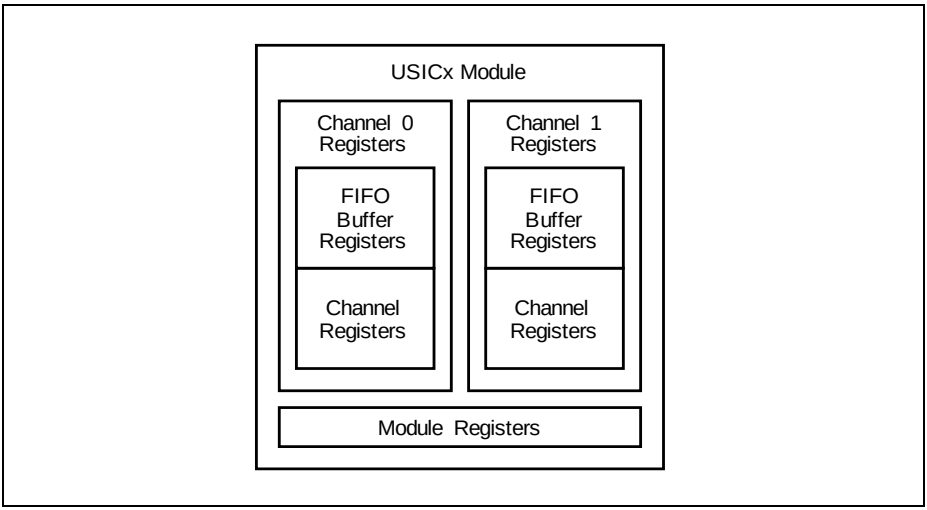


Figure 17-64 USIC Module and Channel Registers

Table 17-19 USIC Kernel-Related and Kernel Registers

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Description see
			Read	Write	
Module Registers[1)]					
ID	Module Identification Register	008_H	U, PV	U, PV	**Page 17-157**
Channel Registers					
–	reserved	000_H	nBE	nBE	–
CCFG	Channel Configuration Register	004_H	U, PV	U, PV	**Page 17-162**

Table 17-19 USIC Kernel-Related and Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Description see
			Read	Write	
KSCFG	Kernel State Configuration Register	$00C_H$	U, PV	U, PV	Page 17-163
FDR	Fractional Divider Register	010_H	U, PV	PV	Page 17-176
BRG	Baud Rate Generator Register	014_H	U, PV	PV	Page 17-177
INPR	Interrupt Node Pointer Register	018_H	U, PV	U, PV	Page 17-166
DX0CR	Input Control Register 0	$01C_H$	U, PV	U, PV	Page 17-171
DX1CR	Input Control Register 1	020_H	U, PV	U, PV	Page 17-173
DX2CR	Input Control Register 2	024_H	U, PV	U, PV	Page 17-171
DX3CR	Input Control Register 3	028_H	U, PV	U, PV	
DX4CR	Input Control Register 4	$02C_H$	U, PV	U, PV	
DX5CR	Input Control Register 5	030_H	U, PV	U, PV	
SCTR	Shift Control Register	034_H	U, PV	U, PV	Page 17-181
TCSR	Transmit Control/Status Register	038_H	U, PV	U, PV	Page 17-184
PCR	Protocol Control Register	$03C_H$	U, PV	U, PV	Page 17-168 2)
			U, PV	U, PV	Page 17-65 3)
			U, PV	U, PV	Page 17-96 4)
			U, PV	U, PV	Page 17-127 5)
			U, PV	U, PV	Page 17-145 6)
CCR	Channel Control Register	040_H	U, PV	PV	Page 17-158
CMTR	Capture Mode Timer Register	044_H	U, PV	U, PV	Page 17-180

Table 17-19 USIC Kernel-Related and Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Description see
			Read	Write	
PSR	Protocol Status Register	048_H	U, PV	U, PV	**Page 17-168** 2)
			U, PV	U, PV	**Page 17-69** 3)
			U, PV	U, PV	**Page 17-100** 4)
			U, PV	U, PV	**Page 17-130** 5)
			U, PV	U, PV	**Page 17-148** 6)
PSCR	Protocol Status Clear Register	$04C_H$	U, PV	U, PV	**Page 17-170**
RBUFSR	Receiver Buffer Status Register	050_H	U, PV	U, PV	**Page 17-202**
RBUF	Receiver Buffer Register	054_H	U, PV	U, PV	**Page 17-199**
RBUFD	Receiver Buffer Register for Debugger	058_H	U, PV	U, PV	**Page 17-201**
RBUF0	Receiver Buffer Register 0	$05C_H$	U, PV	U, PV	**Page 17-193**
RBUF1	Receiver Buffer Register 1	060_H	U, PV	U, PV	**Page 17-194**
RBUF01SR	Receiver Buffer 01 Status Register	064_H	U, PV	U, PV	**Page 17-195**
FMR	Flag Modification Register	068_H	U, PV	U, PV	**Page 17-191**
–	reserved; do not access this location	$06C_H$	U, PV	nBE	–
–	reserved	070_H - $07C_H$	nBE	nBE	–
TBUFx	Transmit Buffer Input Location x (x = 00-31)	080_H + x*4	U, PV	U, PV	**Page 17-193**
FIFO Buffer Registers					
BYP	Bypass Data Register	100_H	U, PV	U, PV	**Page 17-203**
BYPCR	Bypass Control Register	104_H	U, PV	U, PV	**Page 17-204**
TBCTR	Transmit Buffer Control Register	108_H	U, PV	U, PV	**Page 17-212**
RBCTR	Receive Buffer Control Register	$10C_H$	U, PV	U, PV	**Page 17-216**
TRBPTR	Transmit/Receive Buffer Pointer Register	110_H	U, PV	U, PV	**Page 17-224**

Table 17-19 USIC Kernel-Related and Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Description see
			Read	Write	
TRBSR	Transmit/Receive Buffer Status Register	114_H	U, PV	U, PV	Page 17-207
TRBSCR	Transmit/Receive Buffer Status Clear Register	118_H	U, PV	U, PV	Page 17-211
OUTR	Receive Buffer Output Register	$11C_H$	U, PV	U, PV	Page 17-222
OUTDR	Receive Buffer Output Register for Debugger	120_H	U, PV	U, PV	Page 17-223
–	reserved	124_H - $17C_H$	nBE	nBE	–
INx	Transmit FIFO Buffer Input Location x (x = 00-31)	180_H + x*4	U, PV	U, PV	Page 17-221

1) Details of the module identification registers are described in the implementation section (see Page 17-157).
2) This page shows the general register layout.
3) This page shows the register layout in ASC mode.
4) This page shows the register layout in SSC mode.
5) This page shows the register layout in IIC mode.
6) This page shows the register layout in IIS mode.

17.11.1 Address Map

The registers of the USIC communication channel are available at the following base addresses. The exact register address is given by the relative address of the register (given in Table 17-19) plus the channel base address (given in Table 17-20).

Table 17-20 Registers Address Space

Module	Base Address	End Address	Note
USIC0_CH0	40030000_H	$400301FF_H$	–
USIC0_CH1	40030200_H	$400303FF_H$	–
USIC1_CH0	48020000_H	$480201FF_H$	–
USIC1_CH1	48020200_H	$480203FF_H$	–
USIC2_CH0	48024000_H	$480241FF_H$	–
USIC2_CH1	48024200_H	$480243FF_H$	–

Table 17-21 FIFO and Reserved Address Space

Module	Base Address	End Address	Note
USIC0	40030400_H	$400307FF_H$	USIC0 RAM area, shared between USIC0_CH0 and USIC0_CH1
reserved	40030800_H	$40033FFF_H$	This address range is reserved
USIC1	48020400_H	$480207FF_H$	USIC1 RAM area, shared between USIC1_CH0 and USIC1_CH1
reserved	48020800_H	$48023FFF_H$	This address range is reserved
USIC2	48024400_H	$480247FF_H$	USIC2 RAM area, shared between USIC2_CH0 and USIC2_CH1
reserved	48024800_H	$48027FFF_H$	This address range is reserved

17.11.2 Module Identification Registers

The module identification registers indicate the function and the design step of the USIC modules.

USIC0_ID
Module Identification Register

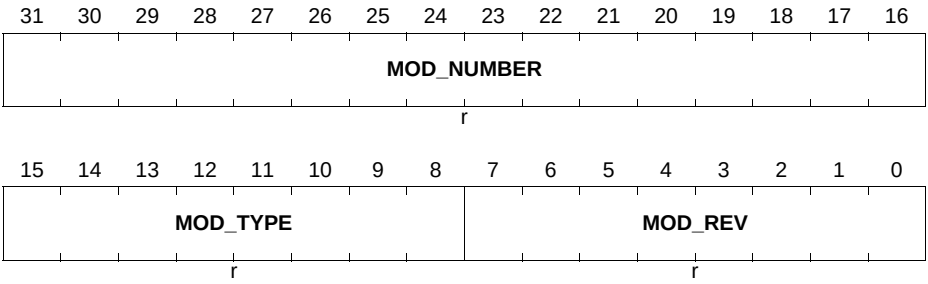
(4003 0008$_H$) **Reset Value: 00AA C0XX$_H$**

USIC1_ID
Module Identification Register

(4802 0008$_H$) **Reset Value: 00AA C0XX$_H$**

USIC2_ID
Module Identification Register

(4802 4008$_H$) **Reset Value: 00AA C0XX$_H$**



Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** MOD_REV defines the revision number. The value of a module revision starts with 01$_H$ (first revision).
MOD_TYPE	[15:8]	r	**Module Type** This bit field is C0$_H$. It defines the module as a 32-bit module.
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines the USIC module identification number (00AA$_H$ = USIC).

17.11.3 Channel Control and Configuration Registers

17.11.3.1 Channel Control Register

The channel control register contains the enable/disable bits for hardware port control and interrupt generation on channel events, the control of the parity generation and the protocol selection of a USIC channel.

FDR can be written only with a privilege mode access.

CCR
Channel Control Register **(40_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															BRGIEN
r															rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AIEN	RIEN	TBIEN	TSIEN	DLIEN	RSIEN	PM		HPCEN		0		MODE			
rw	rw	rw	rw	rw	rw	rw		rw		r		rw			

Field	Bits	Type	Description
MODE	[3:0]	rw	**Operating Mode** This bit field selects the protocol for this USIC channel. Selecting a protocol that is not available (see register CCFG) or a reserved combination disables the USIC channel. When switching between two protocols, the USIC channel has to be disabled before selecting a new protocol. In this case, registers PCR and PSR have to be cleared or updated by software. 0_H The USIC channel is disabled. All protocol-related state machines are set to an idle state. 1_H The SSC (SPI) protocol is selected. 2_H The ASC (SCI, UART) protocol is selected. 3_H The IIS protocol is selected. 4_H The IIC protocol is selected. Other bit combinations are reserved.

Field	Bits	Type	Description
HPCEN	[7:6]	rw	**Hardware Port Control Enable** This bit enables the hardware port control for the specified set of DX[3:0] and DOUT[3:0] pins. 00_B The hardware port control is disabled. 01_B The hardware port control is enabled for DX0 and DOUT0. 10_B The hardware port control is enabled for DX3, DX0 and DOUT[1:0]. 11_B The hardware port control is enabled for DX0, DX[5:3] and DOUT[3:0]. *Note: The hardware port control feature is useful only for SSC protocols in half-duplex configurations, such as dual- and quad-SSC. For all other protocols HPCEN must always be written with 00_B.*
PM	[9:8]	rw	**Parity Mode** This bit field defines the parity generation of the sampled input values. 00_B The parity generation is disabled. 01_B Reserved 10_B Even parity is selected (parity bit = 1 on odd number of 1s in data, parity bit = 0 on even number of 1s in data). 11_B Odd parity is selected (parity bit = 0 on odd number of 1s in data, parity bit = 1 on even number of 1s in data).
RSIEN	10	rw	**Receiver Start Interrupt Enable** This bit enables the interrupt generation in case of a receiver start event. 0_B The receiver start interrupt is disabled. 1_B The receiver start interrupt is enabled. In case of a receiver start event, the service request output SRx indicated by INPR.TBINP is activated.

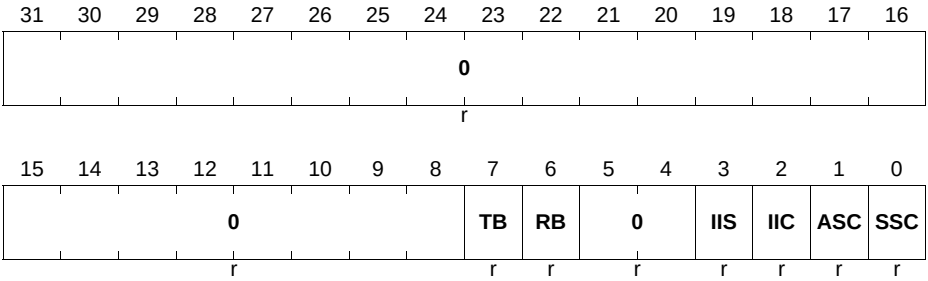
Field	Bits	Type	Description
DLIEN	11	rw	**Data Lost Interrupt Enable** This bit enables the interrupt generation in case of a data lost event (data received in RBUFx while RDVx = 1). 0_B The data lost interrupt is disabled. 1_B The data lost interrupt is enabled. In case of a data lost event, the service request output SRx indicated by INPR.PINP is activated.
TSIEN	12	rw	**Transmit Shift Interrupt Enable** This bit enables the interrupt generation in case of a transmit shift event. 0_B The transmit shift interrupt is disabled. 1_B The transmit shift interrupt is enabled. In case of a transmit shift interrupt event, the service request output SRx indicated by INPR.TSINP is activated.
TBIEN	13	rw	**Transmit Buffer Interrupt Enable** This bit enables the interrupt generation in case of a transmit buffer event. 0_B The transmit buffer interrupt is disabled. 1_B The transmit buffer interrupt is enabled. In case of a transmit buffer event, the service request output SRx indicated by INPR.TBINP is activated.
RIEN	14	rw	**Receive Interrupt Enable** This bit enables the interrupt generation in case of a receive event. 0_B The receive interrupt is disabled. 1_B The receive interrupt is enabled. In case of a receive event, the service request output SRx indicated by INPR.RINP is activated.
AIEN	15	rw	**Alternative Receive Interrupt Enable** This bit enables the interrupt generation in case of a alternative receive event. 0_B The alternative receive interrupt is disabled. 1_B The alternative receive interrupt is enabled. In case of an alternative receive event, the service request output SRx indicated by INPR.AINP is activated.

Field	Bits	Type	Description
BRGIEN	16	rw	**Baud Rate Generator Interrupt Enable** This bit enables the interrupt generation in case of a baud rate generator event. 0_B The baud rate generator interrupt is disabled. 1_B The baud rate generator interrupt is enabled. In case of a baud rate generator event, the service request output SRx indicated by INPR.PINP is activated.
0	[5:4], [31:17]	r	**Reserved** Read as 0; should be written with 0.

17.11.3.2 Channel Configuration Register

The channel configuration register contains indicates the functionality that is available in the USIC channel.

CCFG
Channel Configuration Register (04_H) Reset Value: 0000 00CF$_H$



Field	Bits	Type	Description
SSC	0	r	**SSC Protocol Available** This bit indicates if the SSC protocol is available. 0_B The SSC protocol is not available. 1_B The SSC protocol is available.
ASC	1	r	**ASC Protocol Available** This bit indicates if the ASC protocol is available. 0_B The ASC protocol is not available. 1_B The ASC protocol is available.
IIC	2	r	**IIC Protocol Available** This bit indicates if the IIC functionality is available. 0_B The IIC protocol is not available. 1_B The IIC protocol is available.
IIS	3	r	**IIS Protocol Available** This bit indicates if the IIS protocol is available. 0_B The IIS protocol is not available. 1_B The IIS protocol is available.
RB	6	r	**Receive FIFO Buffer Available** This bit indicates if an additional receive FIFO buffer is available. 0_B A receive FIFO buffer is not available. 1_B A receive FIFO buffer is available.

Field	Bits	Type	Description
TB	7	r	**Transmit FIFO Buffer Available** This bit indicates if an additional transmit FIFO buffer is available. 0_B A transmit FIFO buffer is not available. 1_B A transmit FIFO buffer is available.
0	[5:4], [15:8], [31:16]	r	**Reserved** Read as 0; should be written with 0.

17.11.3.3 Kernel State Configuration Register

The kernel state configuration register KSCFG allows the selection of the desired kernel modes for the different device operating modes.

KSCFG
Kernel State Configuration Register ($0C_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				BPSUM	0	SUMCFG		BPNOM	0	NOMCFG		0		BPMODEN	MODEN
r				w	r	rw		w	r	rw		r		w	rw

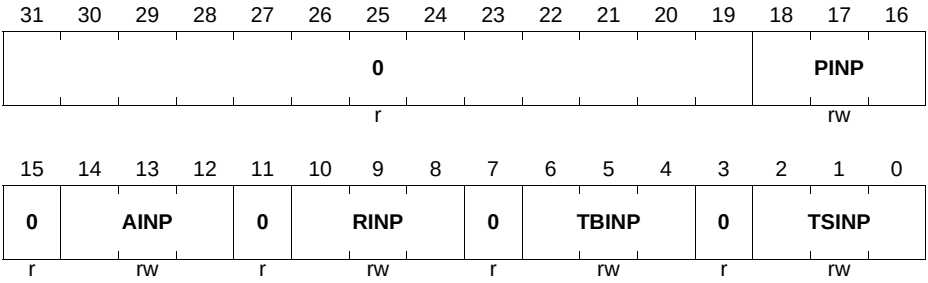
Field	Bits	Type	Description
MODEN	0	rw	**Module Enable** This bit enables the module kernel clock and the module functionality. 0_B The module is switched off immediately (without respecting a stop condition). It does not react on mode control actions and the module clock is switched off. The module does not react on read accesses and ignores write accesses (except to KSCFG). 1_B The module is switched on and can operate. After writing 1 to MODEN, it is recommended to read register KSCFG to avoid pipeline effects in the control block before accessing other USIC registers.
BPMODEN	1	w	**Bit Protection for MODEN** This bit enables the write access to the bit MODEN. It always reads 0. 0_B MODEN is not changed. 1_B MODEN is updated with the written value.
NOMCFG	[5:4]	rw	**Normal Operation Mode Configuration** This bit field defines the kernel mode applied in normal operation mode. 00_B Run mode 0 is selected. 01_B Run mode 1 is selected. 10_B Stop mode 0 is selected. 11_B Stop mode 1 is selected.
BPNOM	7	w	**Bit Protection for NOMCFG** This bit enables the write access to the bit field NOMCFG. It always reads 0. 0_B NOMCFG is not changed. 1_B NOMCFG is updated with the written value.
SUMCFG	[9:8]	rw	**Suspend Mode Configuration** This bit field defines the kernel mode applied in suspend mode. Coding like NOMCFG.

Field	Bits	Type	Description
BPSUM	11	w	**Bit Protection for SUMCFG** This bit enables the write access to the bit field SUMCFG. It always reads 0. 0_B SUMCFG is not changed. 1_B SUMCFG is updated with the written value.
0	[3:2], 6, 10, [31:12]	r	**Reserved** Read as 0; should be written with 0. Bit 2 can read as 1 after BootROM exit (but can be ignored).

17.11.3.4 Interrupt Node Pointer Register

The interrupt node pointer register defines the service request output SRx that is activated if the corresponding event occurs and interrupt generation is enabled.

INPR
Interrupt Node Pointer Register (18_H) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
TSINP	[2:0]	rw	**Transmit Shift Interrupt Node Pointer** This bit field defines which service request output SRx becomes activated in case of a transmit shift interrupt. 000_B Output SR0 becomes activated. 001_B Output SR1 becomes activated. 010_B Output SR2 becomes activated. 011_B Output SR3 becomes activated. 100_B Output SR4 becomes activated. 101_B Output SR5 becomes activated. *Note: All other settings of the bit field are reserved.*
TBINP	[6:4]	rw	**Transmit Buffer Interrupt Node Pointer** This bit field defines which service request output SRx will be activated in case of a transmit buffer interrupt or a receive start interrupt. Coding like TSINP.
RINP	[10:8]	rw	**Receive Interrupt Node Pointer** This bit field defines which service request output SRx will be activated in case of a receive interrupt. Coding like TSINP.

Field	Bits	Type	Description
AINP	[14:12]	rw	**Alternative Receive Interrupt Node Pointer** This bit field defines which service request output SRx will be activated in case of a alternative receive interrupt. Coding like TSINP.
PINP	[18:16]	rw	**Transmit Shift Interrupt Node Pointer** This bit field defines which service request output SRx becomes activated in case of a protocol interrupt. 000_B Output SR0 becomes activated. 001_B Output SR1 becomes activated. 010_B Output SR2 becomes activated. 011_B Output SR3 becomes activated. 100_B Output SR4 becomes activated. 101_B Output SR5 becomes activated. *Note: All other settings of the bit field are reserved.*
0	3, 7, 11, 15, [31:19]	r	**Reserved** Read as 0; should be written with 0.

17.11.4 Protocol Related Registers

17.11.4.1 Protocol Control Registers

The bits in the protocol control register define protocol-specific functions. They have to be configured by software before enabling a new protocol. Only the bits used for the selected protocol are taken into account, whereas the other bit positions always read as 0. The protocol-specific meaning is described in the related protocol section.

PCR
Protocol Control Register **($3C_H$)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CTR 31	**CTR 30**	**CTR 29**	**CTR 28**	**CTR 27**	**CTR 26**	**CTR 25**	**CTR 24**	**CTR 23**	**CTR 22**	**CTR 21**	**CTR 20**	**CTR 19**	**CTR 18**	**CTR 17**	**CTR 16**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTR 15	**CTR 14**	**CTR 13**	**CTR 12**	**CTR 11**	**CTR 10**	**CTR 9**	**CTR 8**	**CTR 7**	**CTR 6**	**CTR 5**	**CTR 4**	**CTR 3**	**CTR 2**	**CTR 1**	**CTR 0**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
CTRx (x = 0-31)	x	rw	**Protocol Control Bit x** This bit is a protocol control bit.

17.11.4.2 Protocol Status Register

The flags in the protocol status register can be cleared by writing a 1 to the corresponding bit position in register PSCR. Writing a 1 to a bit position in PSR sets the corresponding flag, but does not lead to further actions (no interrupt generation). Writing a 0 has no effect. These flags should be cleared by software before enabling a new protocol. The protocol-specific meaning is described in the related protocol section.

PSR
Protocol Status Register **(48_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
							0								**BRG IF**
							r								rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AIF	**RIF**	**TBIF**	**TSIF**	**DLIF**	**RSIF**	**ST9**	**ST8**	**ST7**	**ST6**	**ST5**	**ST4**	**ST3**	**ST2**	**ST1**	**ST0**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
STx (x = 0-9)	x	rwh	**Protocol Status Flag x** See protocol specific description.
RSIF	10	rwh	**Receiver Start Indication Flag** 0_B A receiver start event has not occurred. 1_B A receiver start event has occurred.
DLIF	11	rwh	**Data Lost Indication Flag** 0_B A data lost event has not occurred. 1_B A data lost event has occurred.
TSIF	12	rwh	**Transmit Shift Indication Flag** 0_B A transmit shift event has not occurred. 1_B A transmit shift event has occurred.
TBIF	13	rwh	**Transmit Buffer Indication Flag** 0_B A transmit buffer event has not occurred. 1_B A transmit buffer event has occurred.
RIF	14	rwh	**Receive Indication Flag** 0_B A receive event has not occurred. 1_B A receive event has occurred.
AIF	15	rwh	**Alternative Receive Indication Flag** 0_B An alternative receive event has not occurred. 1_B An alternative receive event has occurred.
BRGIF	16	rwh	**Baud Rate Generator Indication Flag** 0_B A baud rate generator event has not occurred. 1_B A baud rate generator event has occurred.
0	[31:17]	r	**Reserved;** read as 0; should be written with 0;

17.11.4.3 Protocol Status Clear Register

Read accesses to this register always deliver 0 at all bit positions.

PSCR
Protocol Status Clear Register **($4C_H$)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															CBRGIF
r															w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAIF	CRIF	CTBIF	CTSIF	CDLIF	CRSIF	CST9	CST8	CST7	CST6	CST5	CST4	CST3	CST2	CST1	CST0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
CSTx (x = 0-9)	x	w	**Clear Status Flag x in PSR** 0_B No action 1_B Flag PSR.STx is cleared.
CRSIF	10	w	**Clear Receiver Start Indication Flag** 0_B No action 1_B Flag PSR.RSIF is cleared.
CDLIF	11	w	**Clear Data Lost Indication Flag** 0_B No action 1_B Flag PSR.DLIF is cleared.
CTSIF	12	w	**Clear Transmit Shift Indication Flag** 0_B No action 1_B Flag PSR.TSIF is cleared.
CTBIF	13	w	**Clear Transmit Buffer Indication Flag** 0_B No action 1_B Flag PSR.TBIF is cleared.
CRIF	14	w	**Clear Receive Indication Flag** 0_B No action 1_B Flag PSR.RIF is cleared.
CAIF	15	w	**Clear Alternative Receive Indication Flag** 0_B No action 1_B Flag PSR.AIF is cleared.
CBRGIF	16	w	**Clear Baud Rate Generator Indication Flag** 0_B No action 1_B Flag PSR.BRGIF is cleared.

Field	Bits	Type	Description
0	[31:17]	r	**Reserved;** read as 0; should be written with 0;

17.11.5 Input Stage Register

17.11.5.1 Input Control Registers

The input control registers contain the bits to define the characteristics of the input stages (input stage DX0 is controlled by register DX0CR, etc.).

DX0CR
Input Control Register 0 **($1C_H$)** **Reset Value: 0000 0000$_H$**
DX2CR
Input Control Register 2 **(24_H)** **Reset Value: 0000 0000$_H$**
DX3CR
Input Control Register 3 **(28_H)** **Reset Value: 0000 0000$_H$**
DX4CR
Input Control Register 4 **($2C_H$)** **Reset Value: 0000 0000$_H$**
DX5CR
Input Control Register 5 **(30_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DXS	**0**			**CM**		**SFSEL**	**DPOL**	**0**	**DSEN**	**DFEN**	**INSW**	**0**	**DSEL**		
rh	r			rw		rw	rw	r	rw	rw	rw	r	rw		

Field	Bits	Type	Description
DSEL	[2:0]	rw	**Data Selection for Input Signal** This bit field defines the input data signal for the corresponding input line for protocol pre-processor. The selection can be made from the input vector DXn[G:A]. 000_B The data input DXnA is selected. 001_B The data input DXnB is selected. 010_B The data input DXnC is selected. 011_B The data input DXnD is selected. 100_B The data input DXnE is selected. 101_B The data input DXnF is selected. 110_B The data input DXnG is selected. 111_B The data input is always 1.
INSW	4	rw	**Input Switch** This bit defines if the data shift unit input is derived from the input data path DXn or from the selected protocol pre-processors. 0_B The input of the data shift unit is controlled by the protocol pre-processor. 1_B The input of the data shift unit is connected to the selected data input line. This setting is used if the signals are directly derived from an input pin without treatment by the protocol pre-processor.
DFEN	5	rw	**Digital Filter Enable** This bit enables/disables the digital filter for signal DXnS. 0_B The input signal is not digitally filtered. 1_B The input signal is digitally filtered.
DSEN	6	rw	**Data Synchronization Enable** This bit selects if the asynchronous input signal or the synchronized (and optionally filtered) signal DXnS can be used as input for the data shift unit. 0_B The un-synchronized signal can be taken as input for the data shift unit. 1_B The synchronized signal can be taken as input for the data shift unit.

Field	Bits	Type	Description
DPOL	8	rw	**Data Polarity for DXn** This bit defines the signal polarity of the input signal. 0_B The input signal is not inverted. 1_B The input signal is inverted.
SFSEL	9	rw	**Sampling Frequency Selection** This bit defines the sampling frequency of the digital filter for the synchronized signal DXnS. 0_B The sampling frequency is f_{PB}. 1_B The sampling frequency is f_{FD}.
CM	[11:10]	rw	**Combination Mode** This bit field selects which edge of the synchronized (and optionally filtered) signal DXnS actives the trigger output DXnT of the input stage. 00_B The trigger activation is disabled. 01_B A rising edge activates DXnT. 10_B A falling edge activates DXnT. 11_B Both edges activate DXnT.
DXS	15	rh	**Synchronized Data Value** This bit indicates the value of the synchronized (and optionally filtered) input signal. 0_B The current value of DXnS is 0. 1_B The current value of DXnS is 1.
0	3, 7, [14:12], [31:16]	r	**Reserved** Read as 0; should be written with 0.

DX1CR
Input Control Register 1 **(20_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DXS	**0**			**CM**		**SFSEL**	**DPOL**	**0**	**DSEN**	**DFEN**	**INSW**	**DCEN**	**DSEL**		
rh	r			rw		rw	rw	r	rw	rw	rw	rw	rw		

Field	Bits	Type	Description
DSEL	[2:0]	rw	**Data Selection for Input Signal** This bit field defines the input data signal for the corresponding input line for protocol pre-processor. The selection can be made from the input vector DX1[G:A]. 000_B The data input DX1A is selected. 001_B The data input DX1B is selected. 010_B The data input DX1C is selected. 011_B The data input DX1D is selected. 100_B The data input DX1E is selected. 101_B The data input DX1F is selected. 110_B The data input DX1G is selected. 111_B The data input is always 1.
DCEN	3	rw	**Delay Compensation Enable** This bit selects if the receive shift clock is controlled by INSW or derived from the input data path DX1. 0_B The receive shift clock is dependent on INSW selection. 1_B The receive shift clock is connected to the selected data input line. This setting is used if delay compensation is required in SSC and IIS protocols, else DCEN should always be 0.
INSW	4	rw	**Input Switch** This bit defines if the data shift unit input is derived from the input data path DX1 or from the selected protocol pre-processors. 0_B The input of the data shift unit is controlled by the protocol pre-processor. 1_B The input of the data shift unit is connected to the selected data input line. This setting is used if the signals are directly derived from an input pin without treatment by the protocol pre-processor.
DFEN	5	rw	**Digital Filter Enable** This bit enables/disables the digital filter for signal DX1S. 0_B The input signal is not digitally filtered. 1_B The input signal is digitally filtered.

Field	Bits	Type	Description
DSEN	6	rw	**Data Synchronization Enable** This bit selects if the asynchronous input signal or the synchronized (and optionally filtered) signal DX1S can be used as input for the data shift unit. 0_B The un-synchronized signal can be taken as input for the data shift unit. 1_B The synchronized signal can be taken as input for the data shift unit.
DPOL	8	rw	**Data Polarity for DXn** This bit defines the signal polarity of the input signal. 0_B The input signal is not inverted. 1_B The input signal is inverted.
SFSEL	9	rw	**Sampling Frequency Selection** This bit defines the sampling frequency of the digital filter for the synchronized signal DX1S. 0_B The sampling frequency is f_{PB}. 1_B The sampling frequency is f_{FD}.
CM	[11:10]	rw	**Combination Mode** This bit field selects which edge of the synchronized (and optionally filtered) signal DX1S actives the trigger output DX1T of the input stage. 00_B The trigger activation is disabled. 01_B A rising edge activates DX1T. 10_B A falling edge activates DX1T. 11_B Both edges activate DX1T.
DXS	15	rh	**Synchronized Data Value** This bit indicates the value of the synchronized (and optionally filtered) input signal. 0_B The current value of DX1S is 0. 1_B The current value of DX1S is 1.
0	7, [14:12], [31:16]	r	**Reserved** Read as 0; should be written with 0.

17.11.6 Baud Rate Generator Registers

17.11.6.1 Fractional Divider Register

The fractional divider register FDR allows the generation of the internal frequency f_{FD}, that is derived from the system clock f_{PB}.

FDR can be written only with a privilege mode access.

FDR
Fractional Divider Register **(10_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		0				RESULT									
rw		r				rh									

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DM		0				STEP									
rw		r				rw									

Field	Bits	Type	Description
STEP	[9:0]	rw	**Step Value** In normal divider mode STEP contains the reload value for RESULT after RESULT has reached 3FF$_H$. In fractional divider mode STEP defines the value added to RESULT with each input clock cycle.
DM	[15:14]	rw	**Divider Mode** This bit fields defines the functionality of the fractional divider block. 00_B The divider is switched off, f_{FD} = 0. 01_B Normal divider mode selected. 10_B Fractional divider mode selected. 11_B The divider is switched off, f_{FD} = 0.

Field	Bits	Type	Description
RESULT	[25:16]	rh	**Result Value** In normal divider mode this bit field is updated with f_{PB} according to: RESULT = RESULT + 1 In fractional divider mode this bit field is updated with f_{PB} according to: RESULT = RESULT + STEP If bit field DM is written with 01_B or 10_B, RESULT is loaded with a start value of $3FF_H$.
0	[31:30]	rw	**Reserved for Future Use** Must be written with 0 to allow correct fractional divider operation.
0	[13:10], [29:26]	r	**Reserved** Read as 0; should be written with 0.

17.11.6.2 Baud Rate Generator Register

The protocol-related counters for baud rate generation and timing measurement are controlled by the register BRG.

FDR can be written only with a privilege mode access.

BRG
Baud Rate Generator Register (14_H) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SCLKCFG		MCLKCFG	SCLKOSEL	0		PDIV									
rw		rw	rw	r		rw									

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	DCTQ					PCTQ		CTQSEL		0	PPPEN	TMEN	0	CLKSEL	
r	rw					rw		rw		r	rw	rw	r	rw	

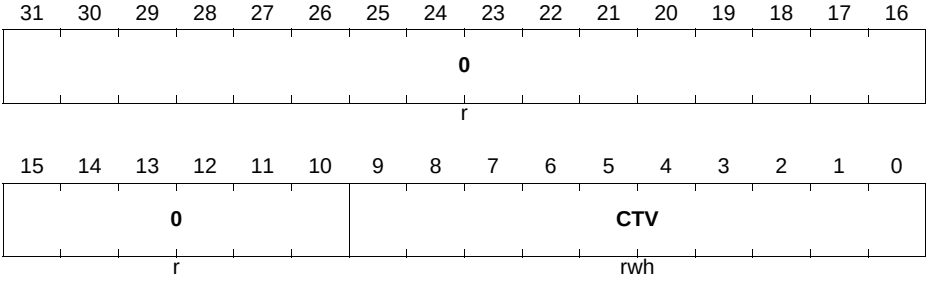
Field	Bits	Type	Description
CLKSEL	[1:0]	rw	**Clock Selection** This bit field defines the input frequency f_{PIN} 00_B The fractional divider frequency f_{FD} is selected. 01_B Reserved, no action 10_B The trigger signal DX1T defines f_{PIN}. Signal MCLK toggles with f_{PIN}. 11_B Signal MCLK corresponds to the DX1S signal and the frequency f_{PIN} is derived from the rising edges of DX1S.
TMEN	3	rw	**Timing Measurement Enable** This bit enables the timing measurement of the capture mode timer. 0_B Timing measurement is disabled: The trigger signals DX0T and DX1T are ignored. 1_B Timing measurement is enabled: The 10-bit counter is incremented by 1 with f_{PPP} and stops counting when reaching its maximum value. If one of the trigger signals DX0T or DX1T become active, the counter value is captured into bit field CTV, the counter is cleared and a transmit shift event is generated.
PPPEN	4	rw	**Enable 2:1 Divider for f_{PPP}** This bit defines the input frequency f_{PPP}. 0_B The 2:1 divider for f_{PPP} is disabled. $f_{PPP} = f_{PIN}$ 1_B The 2:1 divider for f_{PPP} is enabled. $f_{PPP} = f_{MCLK} = f_{PIN} / 2$.
CTQSEL	[7:6]	rw	**Input Selection for CTQ** This bit defines the length of a time quantum for the protocol pre-processor. 00_B $f_{CTQIN} = f_{PDIV}$ 01_B $f_{CTQIN} = f_{PPP}$ 10_B $f_{CTQIN} = f_{SCLK}$ 11_B $f_{CTQIN} = f_{MCLK}$

Field	Bits	Type	Description
PCTQ	[9:8]	rw	**Pre-Divider for Time Quanta Counter** This bit field defines length of a time quantum tq for the time quanta counter in the protocol pre-processor. t_Q = (PCTQ + 1) / f_{CTQIN}
DCTQ	[14:10]	rw	**Denominator for Time Quanta Counter** This bit field defines the number of time quanta t_q taken into account by the time quanta counter in the protocol pre-processor.
PDIV	[25:16]	rw	**Divider Mode: Divider Factor to Generate f_{PDIV}** This bit field defines the ratio between the input frequency f_{PPP} and the divider frequency f_{PDIV}.
SCLKOSEL	28	rw	**Shift Clock Output Select** This bit field selects the input source for the SCLKOUT signal. 0_B SCLK from the baud rate generator is selected as the SCLKOUT input source. 1_B The transmit shift clock from DX1 input stage is selected as the SCLKOUT input source. *Note: The setting SCLKOSEL = 1 is used only when complete closed loop delay compensation is required for a slave SSC/IIS. The default setting of SCLKOSEL = 0 should be always used for all other cases.*
MCLKCFG	29	rw	**Master Clock Configuration** This bit field defines the level of the passive phase of the MCLKOUT signal. 0_B The passive level is 0. 1_B The passive level is 1.
SCLKCFG	[31:30]	rw	**Shift Clock Output Configuration** This bit field defines the level of the passive phase of the SCLKOUT signal and enables/disables a delay of half of a SCLK period. 00_B The passive level is 0 and the delay is disabled. 01_B The passive level is 1 and the delay is disabled. 10_B The passive level is 0 and the delay is enabled. 11_B The passive level is 1 and the delay is enabled.
0	2, 5, 15, [27:26]	r	**Reserved** Read as 0; should be written with 0.

17.11.6.3 Capture Mode Timer Register

The captured timer value is provided by the register CMTR.

CMTR
Capture Mode Timer Register **(44_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
CTV	[9:0]	rwh	**Captured Timer Value** The value of the counter is captured into this bit field if one of the trigger signals DX0T or DX1T are activated by the corresponding input stage.
0	[31:10]	r	**Reserved** Read as 0; should be written with 0.

17.11.7 Transfer Control and Status Registers

17.11.7.1 Shift Control Register

The data shift unit is controlled by the register SCTR. The values in this register are applied for data transmission and reception.

Please note that the shift control settings SDIR, WLE, FLE, DSM and HPCDIR are shared between transmitter and receiver. They are internally “frozen” for a each data word transfer in the transmitter with the first transmit shift clock edge and with the first receive shift clock edge in the receiver. The software has to take care that updates of these bit fields by software are done coherently (e.g. refer to the receiver start event indication PSR.RSIF).

SCTR
Shift Control Register **(34_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0				WLE				0		FLE					
r				rwh				r		rwh					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						TRM		DOCFG		0	HPC DIR	DSM		PDL	SDIR
r						rw		rw		r	rw	rw		rw	rw

Field	Bits	Type	Description
SDIR	0	rw	**Shift Direction** This bit defines the shift direction of the data words for transmission and reception. 0_B Shift LSB first. The first data bit of a data word is located at bit position 0. 1_B Shift MSB first. The first data bit of a data word is located at the bit position given by bit field SCTR.WLE.
PDL	1	rw	**Passive Data Level** This bit defines the output level at the shift data output signal when no data is available for transmission. The PDL level is output with the first relevant transmit shift clock edge of a data word. 0_B The passive data level is 0. 1_B The passive data level is 1.

Field	Bits	Type	Description
DSM	[3:2]	rw	**Data Shift Mode** This bit field describes how the receive and transmit data is shifted in and out. 00_B Receive and transmit data is shifted in and out one bit at a time through DX0 and DOUT0. 01_B Reserved. 10_B Receive and transmit data is shifted in and out two bits at a time through two input stages (DX0 and DX3) and DOUT[1:0] respectively. 11_B Receive and transmit data is shifted in and out four bits at a time through four input stages (DX0, DX[5:3]) and DOUT[3:0] respectively. *Note: Dual- and Quad-output modes are used only by the SSC protocol. For all other protocols DSM must always be written with 00_B.*
HPCDIR	4	rw	**Port Control Direction** This bit defines the direction of the port pin(s) which allows hardware pin control (CCR.PCEN = 1). 0_B The pin(s) with hardware pin control enabled are selected to be in input mode. 1_B The pin(s) with hardware pin control enabled are selected to be in output mode.
DOCFG	[7:6]	rw	**Data Output Configuration** This bit defines the relation between the internal shift data value and the data output signal DOUTx. $X0_B$ DOUTx = shift data value $X1_B$ DOUTx = inverted shift data value

Field	Bits	Type	Description
TRM	[9:8]	rw	**Transmission Mode** This bit field describes how the shift control signal is interpreted by the DSU. Data transfers are only possible while the shift control signal is active. 00_B The shift control signal is considered as inactive and data frame transfers are not possible. 01_B The shift control signal is considered active if it is at 1-level. This is the setting to be programmed to allow data transfers. 10_B The shift control signal is considered active if it is at 0-level. It is recommended to avoid this setting and to use the inversion in the DX2 stage in case of a low-active signal. 11_B The shift control signal is considered active without referring to the actual signal level. Data frame transfer is possible after each edge of the signal.
FLE	[21:16]	rwh	**Frame Length** This bit field defines how many bits are transferred within a data frame. A data frame can consist of several concatenated data words. If TCSR.FLEMD = 1, the value can be updated automatically by the data handler.
WLE	[27:24]	rwh	**Word Length** This bit field defines the data word length (amount of bits that are transferred in each data word) for reception and transmission. The data word is always right-aligned in the data buffer at the bit positions [WLE down to 0]. If TCSR.WLEMD = 1, the value can be updated automatically by the data handler. 0_H The data word contains 1 data bit located at bit position 0. 1_H The data word contains 2 data bits located at bit positions [1:0]. ... E_H The data word contains 15 data bits located at bit positions [14:0]. F_H The data word contains 16 data bits located at bit positions [15:0].

Field	Bits	Type	Description
0	5, [15:10], [23:22], [31:28]	r	**Reserved** Read as 0; should be written with 0.

17.11.7.2 Transmission Control and Status Register

The data transmission is controlled and monitored by register TCSR.

TCSR
Transmit Control/Status Register (38$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0			**TE**	**TVC**	**TV**	**0**	**TSOF**	**0**							
r			rh	rh	rh	r	rh	r							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		**WA**	**TDVTR**	**TDEN**		**0**	**TDSSM**	**TDV**	**EOF**	**SOF**	**HPCMD**	**WAMD**	**FLEMD**	**SELMD**	**WLEMD**
r		rwh	rw	rw		r	rw	rh	rwh	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
WLEMD	0	rw	**WLE Mode** This bit enables the data handler to automatically update the bit field SCTR.WLE by the transmit control information TCI[3:0] and bit TCSR.EOF by TCI[4] (see **Page 17-32**). If enabled, an automatic update takes place when new data is loaded to register TBUF, either by writing to one of the transmit buffer input locations TBUFx or by an optional data buffer. 0_B The automatic update of SCTR.WLE and TCSR.EOF is disabled. 1_B The automatic update of SCTR.WLE and TCSR.EOF is enabled.

Field	Bits	Type	Description
SELMD	1	rw	**Select Mode** This bit can be used mainly for the SSC protocol. It enables the data handler to automatically update bit field PCR.CTR[20:16] by the transmit control information TCI[4:0] and clear bit field PCR.CTR[23:21] (see **Page 17-32**). If enabled, an automatic update takes place when new data is loaded to register TBUF, either by writing to one of the transmit buffer input locations TBUFx or by an optional data buffer. 0_B The automatic update of PCR.CTR[23:16] is disabled. 1_B The automatic update of PCR.CTR[23:16] is disabled.
FLEMD	2	rw	**FLE Mode** This bit enables the data handler to automatically update bits SCTR.FLE[4:0] by the transmit control information TCI[4:0] and to clear bit SCTR.FLE[5] (see **Page 17-32**). If enabled, an automatic update takes place when new data is loaded to register TBUF, either by writing to one of the transmit buffer input locations TBUFx or by an optional data buffer. 0_B The automatic update of FLE is disabled. 1_B The automatic update of FLE is enabled.
WAMD	3	rw	**WA Mode** This bit can be used mainly for the IIS protocol. It enables the data handler to automatically update bit TCSR.WA by the transmit control information TCI[4] (see **Page 17-32**). If enabled, an automatic update takes place when new data is loaded to register TBUF, either by writing to one of the transmit buffer input locations TBUFx or by an optional data buffer. 0_B The automatic update of bit WA is disabled. 1_B The automatic update of bit WA is enabled.

Field	Bits	Type	Description
HPCMD	4	rw	**Hardware Port Control Mode** This bit can be used mainly for the dual and quad SSC protocol. It enables the data handler to automatically update bit SCTR.DSM by the transmit control information TCI[1:0] and bit SCTR.HPCDIR by TCI[2] (see **Page 17-32**). If enabled, an automatic update takes place when new data is loaded to register TBUF, either by writing to one of the transmit buffer input locations TBUFx or by an optional data buffer. 0_B The automatic update of bits SCTR.DSM and SCTR.HPCDIR is disabled. 1_B The automatic update of bits SCTR.DSM and SCTR.HPCDIR is enabled.
SOF	5	rw	**Start Of Frame** This bit is only taken into account for the SSC protocol, otherwise it is ignored. It indicates that the data word in TBUF is considered as the first word of a new SSC frame if it is valid for transmission (TCSR.TDV = 1). This bit becomes cleared when the TBUF data word is transferred to the transmit shift register. 0_B The data word in TBUF is not considered as first word of a frame. 1_B The data word in TBUF is considered as first word of a frame. A currently running frame is finished and MSLS becomes deactivated (respecting the programmed delays).

Field	Bits	Type	Description
EOF	6	rwh	**End Of Frame** This bit is only taken into account for the SSC protocol, otherwise it is ignored. It can be modified automatically by the data handler if bit WLEMD = 1. It indicates that the data word in TBUF is considered as the last word of an SSC frame. If it is the last word, the MSLS signal becomes inactive after the transfer, respecting the programmed delays. This bit becomes cleared when the TBUF data word is transferred to the transmit shift register. 0_B The data word in TBUF is not considered as last word of an SSC frame. 1_B The data word in TBUF is considered as last word of an SSC frame.
TDV	7	rh	**Transmit Data Valid** This bit indicates that the data word in the transmit buffer TBUF can be considered as valid for transmission. The TBUF data word can only be sent out if TDV = 1. It is automatically set when data is moved to TBUF (by writing to one of the transmit buffer input locations TBUFx, or optionally, by the bypass or FIFO mechanism). 0_B The data word in TBUF is not valid for transmission. 1_B The data word in TBUF is valid for transmission and a transmission start is possible. New data should not be written to a TBUFx input location while TDV = 1.

Field	Bits	Type	Description
TDSSM	8	rw	**TBUF Data Single Shot Mode** This bit defines if the data word TBUF data is considered as permanently valid or if the data should only be transferred once. 0_B The data word in TBUF is not considered as invalid after it has been loaded into the transmit shift register. The loading of the TBUF data into the shift register does not clear TDV. 1_B The data word in TBUF is considered as invalid after it has been loaded into the shift register. In ASC and IIC mode, TDV is cleared with the TBI event, whereas in SSC and IIS mode, it is cleared with the RSI event. TDSSM = 1 has to be programmed if an optional data buffer is used.
TDEN	[11:10]	rw	**TBUF Data Enable** This bit field controls the gating of the transmission start of the data word in the transmit buffer TBUF. 00_B A transmission start of the data word in TBUF is disabled. If a transmission is started, the passive data level is sent out. 01_B A transmission of the data word in TBUF can be started if TDV = 1. 10_B A transmission of the data word in TBUF can be started if TDV = 1 while DX2S = 0. 11_B A transmission of the data word in TBUF can be started if TDV = 1 while DX2S = 1.
TDVTR	12	rw	**TBUF Data Valid Trigger** This bit enables the transfer trigger unit to set bit TCSR.TE if the trigger signal DX2T becomes active for event driven transfer starts, e.g. timer-based or depending on an event at an input pin. Bit TDVTR has to be 0 for protocols where the input stage DX2 is used for data shifting. 0_B Bit TCSR.TE is permanently set. 1_B Bit TCSR.TE is set if DX2T becomes active while TDV = 1.

Field	Bits	Type	Description
WA	13	rwh	**Word Address** This bit is only taken into account for the IIS protocol, otherwise it is ignored. It can be modified automatically by the data handler if bit WAMD = 1. Bit WA defines for which channel the data stored in TBUF will be transmitted. 0_B The data word in TBUF will be transmitted after a falling edge of WA has been detected (referring to PSR.WA). 1_B The data word in TBUF will be transmitted after a rising edge of WA has been detected (referring to PSR.WA).
TSOF	24	rh	**Transmitted Start Of Frame** This bit indicates if the latest start of a data word transmission has taken place for the first data word of a new data frame. This bit is updated with the transmission start of each data word. 0_B The latest data word transmission has not been started for the first word of a data frame. 1_B The latest data word transmission has been started for the first word of a data frame.
TV	26	rh	**Transmission Valid** This bit represents the transmit buffer underflow and indicates if the latest start of a data word transmission has taken place with a valid data word from the transmit buffer TBUF. This bit is updated with the transmission start of each data word. 0_B The latest start of a data word transmission has taken place while no valid data was available. As a result, the transmission of a data words with passive level (SCTR.PDL) has been started. 1_B The latest start of a data word transmission has taken place with valid data from TBUF.

Field	Bits	Type	Description
TVC	27	rh	**Transmission Valid Cumulated** This bit cumulates the transmit buffer underflow indication TV. It is cleared automatically together with bit TV and has to be set by writing FMR.ATVC = 1. 0_B Since TVC has been set, at least one data buffer underflow condition has occurred. 1_B Since TVC has been set, no data buffer underflow condition has occurred.
TE	28	rh	**Trigger Event** If the transfer trigger mechanism is enabled, this bit indicates that a trigger event has been detected (DX2T = 1) while TCSR.TDV = 1. If the event trigger mechanism is disabled, the bit TE is permanently set. It is cleared by writing FMR.MTDV = 10_B or when the data word located in TBUF is loaded into the shift register. 0_B The trigger event has not yet been detected. A transmission of the data word in TBUF can not be started. 1_B The trigger event has been detected (or the trigger mechanism is switched off) and a transmission of the data word in TBUF can not be started.
0	9, [23:14], 25, [31:29]	r	**Reserved** Read as 0; should be written with 0.

17.11.7.3 Flag Modification Registers

The flag modification register FMR allows the modification of control and status flags related to data handling by using only write accesses. Read accesses to FMR always deliver 0 at all bit positions.

Additionally, the service request outputs of this USIC channel can be activated by software (the activation is triggered by the write access and is deactivated automatically).

FMR
Flag Modification Register **(68_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0										SIO5	SIO4	SIO3	SIO2	SIO1	SIO0
r										w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CRDV1	CRDV0	0									ATVC	0		MTDV	
w	w	r									w	r		w	

Field	Bits	Type	Description
MTDV	[1:0]	w	**Modify Transmit Data Valid** Writing to this bit field can modify bits TCSR.TDV and TCSR.TE to control the start of a data word transmission by software. 00_B No action. 01_B Bit TDV is set, TE is unchanged. 10_B Bits TDV and TE are cleared. 11_B Reserved
ATVC	4	w	**Activate Bit TVC** Writing to this bit can set bit TCSR.TVC to start a new cumulation of the transmit buffer underflow condition. 0_B No action. 1_B Bit TCSR.TVC is set.
CRDV0	14	w	**Clear Bits RDV for RBUF0** Writing 1 to this bit clears bits RBUF01SR.RDV00 and RBUF01SR.RDV10 to declare the received data in RBUF0 as no longer valid (to emulate a read action). 0_B No action. 1_B Bits RBUF01SR.RDV00 and RBUF01SR.RDV10 are cleared.

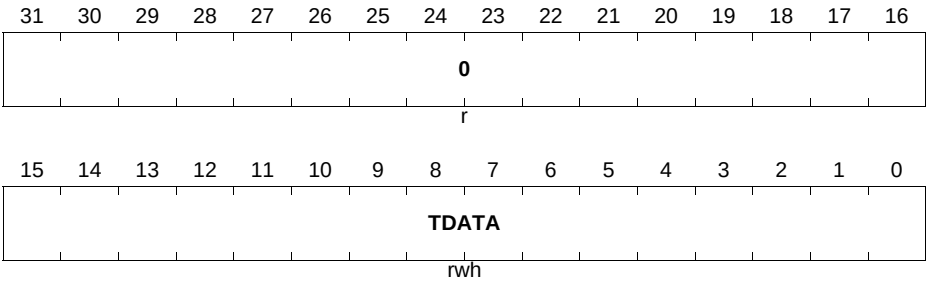
Field	Bits	Type	Description
CRDV1	15	w	**Clear Bit RDV for RBUF1** Writing 1 to this bit clears bits RBUF01SR.RDV01 and RBUF01SR.RDV11 to declare the received data in RBUF1 as no longer valid (to emulate a read action). 0_B No action. 1_B Bits RBUF01SR.RDV01 and RBUF01SR.RDV11 are cleared.
SIO0, **SIO1,** **SIO2,** **SIO3,** **SIO4,** **SIO5**	16, 17, 18, 19, 20, 21	w	**Set Interrupt Output SRx** Writing a 1 to this bit field activates the service request output SRx of this USIC channel. It has no impact on service request outputs of other USIC channels. 0_B No action. 1_B The service request output SRx is activated.
0	[3:2], [13:5], [31:22]	r	**Reserved** Read as 0; should be written with 0.

17.11.8 Data Buffer Registers

17.11.8.1 Transmit Buffer Locations

The 32 independent data input locations TBUF00 to TBUF31 are address locations that can be used as data entry locations for the transmit buffer. Data written to one of these locations will appear in a common register TBUF. Additionally, the 5 bit coding of the number [31:0] of the addressed data input location represents the transmit control information TCI (please refer to the protocol sections for more details).
The internal transmit buffer register TBUF contains the data that will be loaded to the transmit shift register for the next transmission of a data word. It can be read out at all TBUF00 to TBUF31 addresses.

TBUFx (x = 00-31)
Transmit Buffer Input Location x (80_H + x*4) Reset Value: 0000 0000_H

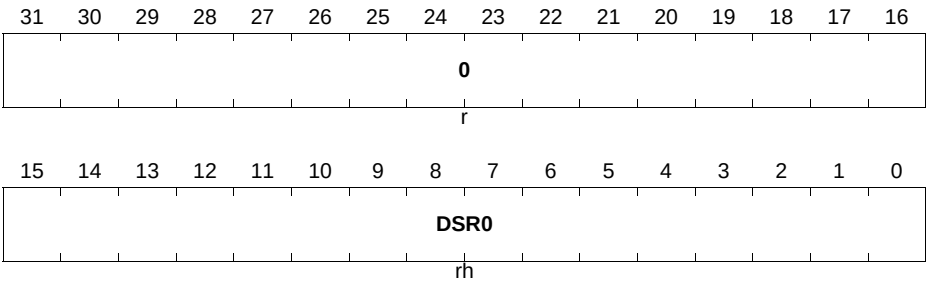


Field	Bits	Type	Description
TDATA	[15:0]	rwh	**Transmit Data** This bit field contains the data to be transmitted (read view). A data write action to at least the low byte of TDATA sets TCSR.TDV.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

17.11.8.2 Receive Buffer Registers RBUF0, RBUF1

The receive buffer register RBUF0 contains the data received from RSR0[3:0]. A read action does not change the status of the receive data from “not yet read = valid” to “already read = not valid”.

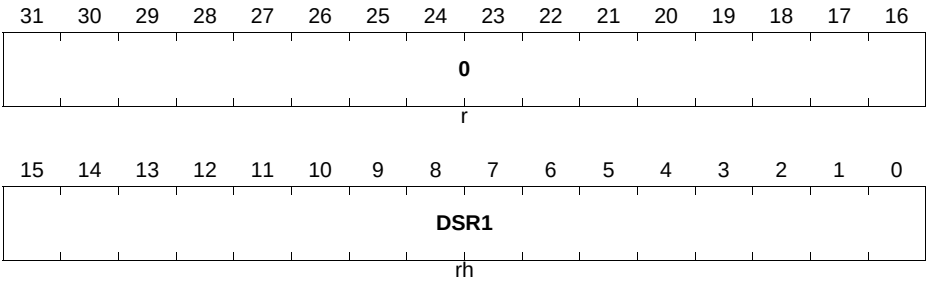
RBUF0
Receiver Buffer Register 0 ($5C_H$) Reset Value: 0000 0000_H



Field	Bits	Type	Description
DSR0	[15:0]	rh	**Data of Shift Registers 0[3:0]**
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

The receive buffer register RBUF1 contains the data received from RSR1[3:0]. A read action does not change the status of the receive data from “not yet read = valid” to “already read = not valid”.

RBUF1
Receiver Buffer Register 1 **(60_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
DSR1	[15:0]	rh	**Data of Shift Registers 1[3:0]**
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

The receive buffer status register RBUF01SR provides the status of the data in receive buffers RBUF0 and RBUF1.

RBUF01SR

Receiver Buffer 01 Status Register (64_H) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DS1	RDV 11	RDV 10	0			PER R1	PAR 1	0	SOF 1	0		WLEN1			
rh	rh	rh	r			rh	rh	r	rh	r		rh			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DS0	RDV 01	RDV 00	0			PER R0	PAR 0	0	SOF 0	0		WLEN0			
rh	rh	rh	r			rh	rh	r	rh	r		rh			

Field	Bits	Type	Description
WLEN0	[3:0]	rh	**Received Data Word Length in RBUF0** This bit field indicates how many bits have been received within the last data word stored in RBUF0. This number indicates how many data bits have to be considered as receive data, whereas the other bits in RBUF0 have been cleared automatically. The received bits are always right-aligned. 0_H One bit has been received. ... F_H Sixteen bits have been received.
SOF0	6	rh	**Start of Frame in RBUF0** This bit indicates whether the data word in RBUF0 has been the first data word of a data frame. 0_B The data in RBUF0 has not been the first data word of a data frame. 1_B The data in RBUF0 has been the first data word of a data frame.
PAR0	8	rh	**Protocol-Related Argument in RBUF0** This bit indicates the value of the protocol-related argument. This value is elaborated depending on the selected protocol and adds additional information to the data word in RBUF0. The meaning of this bit is described in the corresponding protocol chapter.

Field	Bits	Type	Description
PERR0	9	rh	**Protocol-related Error in RBUF0** This bit indicates if the value of the protocol-related argument meets an expected value. This value is elaborated depending on the selected protocol and adds additional information to the data word in RBUF0. The meaning of this bit is described in the corresponding protocol chapter. 0_B The received protocol-related argument PAR matches the expected value. The reception of the data word sets bit PSR.RIF and can generate a receive interrupt. 1_B The received protocol-related argument PAR does not match the expected value. The reception of the data word sets bit PSR.AIF and can generate an alternative receive interrupt.
RDV00	13	rh	**Receive Data Valid in RBUF0** This bit indicates the status of the data content of register RBUF0. This bit is identical to bit RBUF01SR.RDV10 and allows consisting reading of information for the receive buffer registers. It is set when a new data word is stored in RBUF0 and automatically cleared if it is read out via RBUF. 0_B Register RBUF0 does not contain data that has not yet been read out. 1_B Register RBUF0 contains data that has not yet been read out.
RDV01	14	rh	**Receive Data Valid in RBUF1** This bit indicates the status of the data content of register RBUF1. This bit is identical to bit RBUF01SR.RDV11 and allows consisting reading of information for the receive buffer registers. It is set when a new data word is stored in RBUF1 and automatically cleared if it is read out via RBUF. 0_B Register RBUF1 does not contain data that has not yet been read out. 1_B Register RBUF1 contains data that has not yet been read out.

Field	Bits	Type	Description
DS0	15	rh	**Data Source** This bit indicates which receive buffer register (RBUF0 or RBUF1) is currently visible in registers RBUF(D) and in RBUFSR for the associated status information. It indicates which buffer contains the oldest data (the data that has been received first). This bit is identical to bit RBUF01SR.DS1 and allows consisting reading of information for the receive buffer registers. 0_B The register RBUF contains the data of RBUF0 (same for associated status information). 1_B The register RBUF contains the data of RBUF1 (same for associated status information).
WLEN1	[19:16]	rh	**Received Data Word Length in RBUF1** This bit field indicates how many bits have been received within the last data word stored in RBUF1. This number indicates how many data bits have to be considered as receive data, whereas the other bits in RBUF1 have been cleared automatically. The received bits are always right-aligned. 0_H One bit has been received. ... F_H Sixteen bits have been received.
SOF1	22	rh	**Start of Frame in RBUF1** This bit indicates whether the data word in RBUF1 has been the first data word of a data frame. 0_B The data in RBUF1 has not been the first data word of a data frame. 1_B The data in RBUF1 has been the first data word of a data frame.
PAR1	24	rh	**Protocol-Related Argument in RBUF1** This bit indicates the value of the protocol-related argument. This value is elaborated depending on the selected protocol and adds additional information to the data word in RBUF1. The meaning of this bit is described in the corresponding protocol chapter.

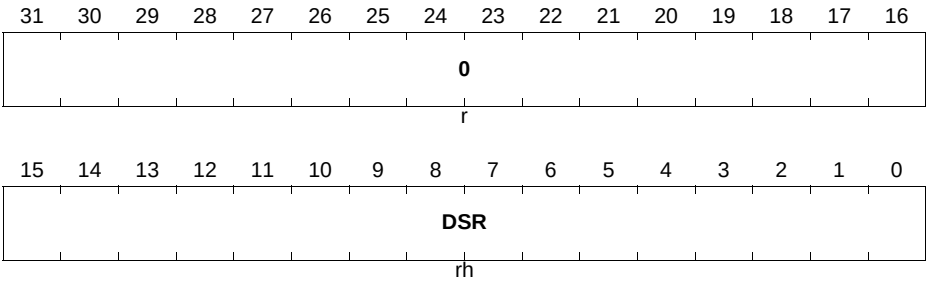
Field	Bits	Type	Description
PERR1	25	rh	**Protocol-related Error in RBUF1** This bit indicates if the value of the protocol-related argument meets an expected value. This value is elaborated depending on the selected protocol and adds additional information to the data word in RBUF1. The meaning of this bit is described in the corresponding protocol chapter. 0_B The received protocol-related argument PAR matches the expected value. The reception of the data word sets bit PSR.RIF and can generate a receive interrupt. 1_B The received protocol-related argument PAR does not match the expected value. The reception of the data word sets bit PSR.AIF and can generate an alternative receive interrupt.
RDV10	29	rh	**Receive Data Valid in RBUF0** This bit indicates the status of the data content of register RBUF0. This bit is identical to bit RBUF01SR.RDV00 and allows consisting reading of information for the receive buffer registers. 0_B Register RBUF0 does not contain data that has not yet been read out. 1_B Register RBUF0 contains data that has not yet been read out.
RDV11	30	rh	**Receive Data Valid in RBUF1** This bit indicates the status of the data content of register RBUF1. This bit is identical to bit RBUF01SR.RDV01 and allows consisting reading of information for the receive buffer registers. 0_B Register RBUF1 does not contain data that has not yet been read out. 1_B Register RBUF1 contains data that has not yet been read out.

Field	Bits	Type	Description
DS1	31	rh	**Data Source** This bit indicates which receive buffer register (RBUF0 or RBUF1) is currently visible in registers RBUF(D) and in RBUFSR for the associated status information. It indicates which buffer contains the oldest data (the data that has been received first). This bit is identical to bit RBUF01SR.DS0 and allows consisting reading of information for the receive buffer registers. 0_B The register RBUF contains the data of RBUF0 (same for associated status information). 1_B The register RBUF contains the data of RBUF1 (same for associated status information).
0	[5:4], 7, [12:10], [21:20], 23, [28:26]	r	**Reserved** Read as 0; should be written with 0.

17.11.8.3 Receive Buffer Registers RBUF, RBUFD, RBUFSR

The receiver buffer register RBUF shows the content of the either RBUF0 or RBUF1, depending on the order of reception. Always the oldest data (the data word that has been received first) from both receive buffers can be read from RBUF. It is recommended to read out the received data from RBUF instead of RBUF0/1. With a read access of at least the low byte of RBUF, the status of the receive data is automatically changed from “not yet read = valid” to “already read = not valid”, the content of RBUF becomes updated, and the next received data word becomes visible in RBUF.

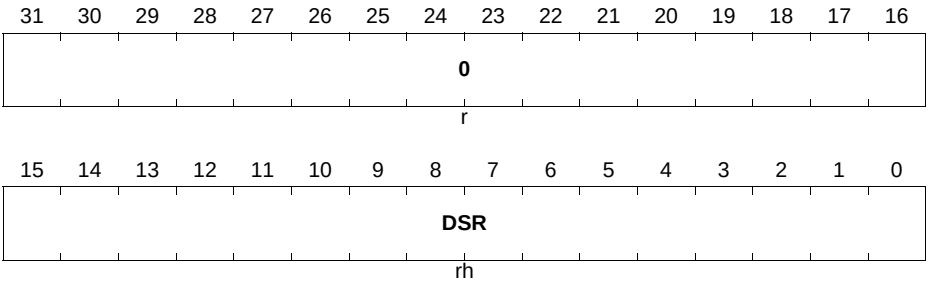
RBUF
Receiver Buffer Register **(54_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
DSR	[15:0]	rh	**Received Data** This bit field monitors the content of either RBUF0 or RBUF1, depending on the reception sequence.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

If a debugger should be used to monitor the received data, the automatic update mechanism has to be de-activated to guaranty data consistency. Therefore, the receiver buffer register for debugging RBUFD is available. It is similar to RBUF, but without the automatic update mechanism by a read action. So a debugger (or other monitoring function) can read RBUFD without disturbing the receive sequence.

RBUFD
Receiver Buffer Register for Debugger(58_H) **Reset Value: $0000\ 0000_H$**

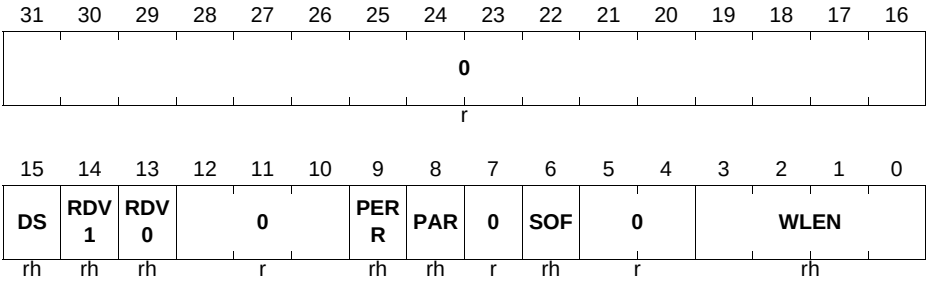


Field	Bits	Type	Description
DSR	[15:0]	rh	**Data from Shift Register** Same as RBUF.DSR, but without releasing the buffer after a read action.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

The receive buffer status register RBUFSR provides the status of the data in receive buffers RBUF and RBUFD. If bits RBUF01SR.DS0 (or RBUF01SR.DS1) are 0, the lower 16-bit content of RBUF01SR is monitored in RBUFSR, otherwise the upper 16-bit content of RBUF01SR is shown.

RBUFSR

Receiver Buffer Status Register **(50_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
WLEN	[3:0]	rh	**Received Data Word Length in RBUF or RBUFD** Description see RBUF01SR.WLEN0 or RBUF01SR.WLEN1.
SOF	6	rh	**Start of Frame in RBUF or RBUFD** Description see RBUF01SR.SOF0 or RBUF01SR.SOF1.
PAR	8	rh	**Protocol-Related Argument in RBUF or RBUFD** Description see RBUF01SR.PAR0 or RBUF01SR.PAR1.
PERR	9	rh	**Protocol-related Error in RBUF or RBUFD** Description see RBUF01SR.PERR0 or RBUF01SR.PERR1.
RDV0	13	rh	**Receive Data Valid in RBUF or RBUFD** Description see RBUF01SR.RDV00 or RBUF01SR.RDV10.
RDV1	14	rh	**Receive Data Valid in RBUF or RBUFD** Description see RBUF01SR.RDV01 or RBUF01SR.RDV11.

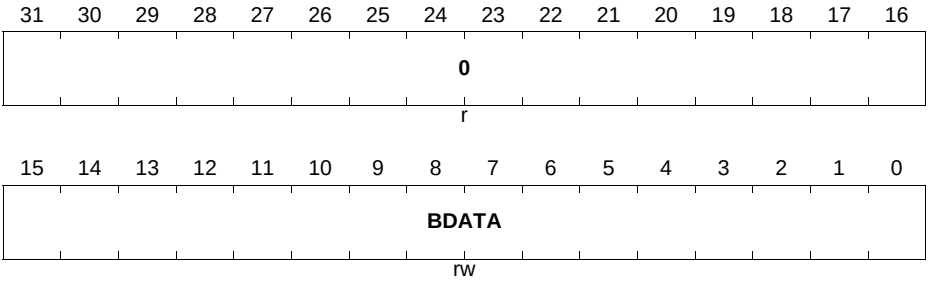
Field	Bits	Type	Description
DS	15	rh	**Data Source of RBUF or RBUFD** Description see RBUF01SR.DS0 or RBUF01SR.DS1.
0	[5:4], 7, [12:10], [31:16]	r	**Reserved** Read as 0; should be written with 0.

17.11.9 FIFO Buffer and Bypass Registers

17.11.9.1 Bypass Registers

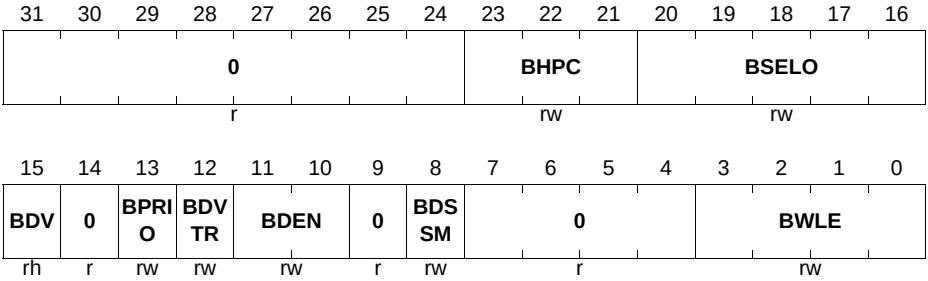
A write action to at least the low byte of the bypass data register sets BYPCR.BDV = 1 (bypass data tagged valid).

BYP
Bypass Data Register **(100_H)** **Reset Value: 0000 0000_H**



Bit (Field)	Width	Type	Description
BDATA	[15:0]	rw	**Bypass Data** This bit field contains the bypass data.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

BYPCR
Bypass Control Register **(104_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
BWLE	[3:0]	rw	**Bypass Word Length** This bit field defines the word length of the bypass data. The word length is given by BWLE + 1 with the data word being right-aligned in the data buffer at the bit positions [BWLE down to 0]. The bypass data word is always considered as an own frame with the length of BWLE. Same coding as SCTR.WLE.
BDSSM	8	rw	**Bypass Data Single Shot Mode** This bit defines if the bypass data is considered as permanently valid or if the bypass data is only transferred once (single shot mode). 0_B The bypass data is still considered as valid after it has been loaded into TBUF. The loading of the data into TBUF does not clear BDV. 1_B The bypass data is considered as invalid after it has been loaded into TBUF. The loading of the data into TBUF clears BDV.

Field	Bits	Type	Description
BDEN	[11:10]	rw	**Bypass Data Enable** This bit field defines if and how the transfer of bypass data to TBUF is enabled. 00_B The transfer of bypass data is disabled. 01_B The transfer of bypass data to TBUF is possible. Bypass data will be transferred to TBUF according to its priority if BDV = 1. 10_B Gated bypass data transfer is enabled. Bypass data will be transferred to TBUF according to its priority if BDV = 1 and while DX2S = 0. 11_B Gated bypass data transfer is enabled. Bypass data will be transferred to TBUF according to its priority if BDV = 1 and while DX2S = 1.
BDVTR	12	rw	**Bypass Data Valid Trigger** This bit enables the bypass data for being tagged valid when DX2T is active (for time framing or time-out purposes). 0_B Bit BDV is not influenced by DX2T. 1_B Bit BDV is set if DX2T is active.
BPRIO	13	rw	**Bypass Priority** This bit defines the priority between the bypass data and the transmit FIFO data. 0_B The transmit FIFO data has a higher priority than the bypass data. 1_B The bypass data has a higher priority than the transmit FIFO data.
BDV	15	rh	**Bypass Data Valid** This bit defines if the bypass data is valid for a transfer to TBUF. This bit is set automatically by a write access to at least the low-byte of register BYP. It can be cleared by software by writing TRBSCR.CBDV. 0_B The bypass data is not valid. 1_B The bypass data is valid.
BSELO	[20:16]	rw	**Bypass Select Outputs** This bit field contains the value that is written to PCR.CTR[20:16] if bypass data is transferred to TBUF while TCSR.SELMD = 1. In the SSC protocol, this bit field can be used to define which SELOx output line will be activated when bypass data is transmitted.

Field	Bits	Type	Description
BHPC	[23:21]	rw	**Bypass Hardware Port Control** This bit field contains the value that is written to SCTR[4:2] if bypass data is transferred to TBUF while TCSR.HPCMD = 1. In the SSC protocol, this bit field can be used to define the data shift mode and if hardware port control is enabled through CCR.HPCEN = 1, the pin direction when bypass data is transmitted.
0	[7:4], 9, 14, [31:24]	r	**Reserved** Read as 0; should be written with 0.

17.11.9.2 General FIFO Buffer Control Registers

The transmit and receive FIFO status information of USICx_CHy is given in registers USICx_CHy.TRBSR.

The bits related to the transmitter buffer in this register can only by written if the transmit buffer functionality is enabled by CCFG.TB = 1, otherwise write accesses are ignored. A similar behavior applies for the bits related to the receive buffer referring to CCFG.RB = 1.

The interrupt flags (event flags) in the transmit and receive FIFO status register TRBSR can be cleared by writing a 1 to the corresponding bit position in register TRBSCR, whereas writing a 0 has to effect on these bits. Writing a 1 by software to SRBI, RBERI, ARBI, STBI, or TBERI sets the corresponding bit to simulate the detection of a transmit/receive buffer event, but without activating any service request output (therefore, see FMR.SIOx).

Bits TBUS and RBUS have been implemented for testing purposes. They can be ignored by data handling software. Please note that a read action can deliver either a 0 or a 1 for these bits. It is recommended to treat them as “don’t care”.

TRBSR
Transmit/Receive Buffer Status Register

(114$_H$) Reset Value: 0000 0808$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	TBFLVL							0	RBFLVL						
r	rh							r	rh						

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	STBT	TBUS	TFULL	TEMPTY	0	TBERI	STBI	0	SRBT	RBUS	RFULL	REMPTY	ARBI	RBERI	SRBI
r	rh	rh	rh	rh	r	rwh	rwh	r	rh	rh	rh	rh	rwh	rwh	rwh

Field	Bits	Type	Description
SRBI	0	rwh	**Standard Receive Buffer Event** This bit indicates that a standard receive buffer event has been detected. It is cleared by writing TRBSCR.CSRBI = 1. If enabled by RBCTR.SRBIEN, the service request output SRx selected by RBCTR.SRBINP becomes activated if a standard receive buffer event is detected. 0_B A standard receive buffer event has not been detected. 1_B A standard receive buffer event has been detected.
RBERI	1	rwh	**Receive Buffer Error Event** This bit indicates that a receive buffer error event has been detected. It is cleared by writing TRBSCR.CRBERI = 1. If enabled by RBCTR.RBERIEN, the service request output SRx selected by RBCTR.ARBINP becomes activated if a receive buffer error event is detected. 0_B A receive buffer error event has not been detected. 1_B A receive buffer error event has been detected.

Field	Bits	Type	Description
ARBI	2	rwh	**Alternative Receive Buffer Event** This bit indicates that an alternative receive buffer event has been detected. It is cleared by writing TRBSCR.CARBI = 1. If enabled by RBCTR.ARBIEN, the service request output SRx selected by RBCTR.ARBINP becomes activated if an alternative receive buffer event is detected. 0_B An alternative receive buffer event has not been detected. 1_B An alternative receive buffer event has been detected.
REMPTY	3	rh	**Receive Buffer Empty** This bit indicates whether the receive buffer is empty. 0_B The receive buffer is not empty. 1_B The receive buffer is empty.
RFULL	4	rh	**Receive Buffer Full** This bit indicates whether the receive buffer is full. 0_B The receive buffer is not full. 1_B The receive buffer is full.
RBUS	5	rh	**Receive Buffer Busy** This bit indicates whether the receive buffer is currently updated by the FIFO handler. 0_B The receive buffer information has been completely updated. 1_B The OUTR update from the FIFO memory is ongoing. A read from OUTR will be delayed. FIFO pointers from the previous read are not yet updated.
SRBT	6	rh	**Standard Receive Buffer Event Trigger** This bit triggers a standard receive buffer event when set. If enabled by RBCTR.SRBIEN, the service request output SRx selected by RBCTR.SRBINP becomes activated until the bit is cleared. 0_B A standard receive buffer event is not triggered using this bit. 1_B A standard receive buffer event is triggered using this bit.

Field	Bits	Type	Description
STBI	8	rwh	**Standard Transmit Buffer Event** This bit indicates that a standard transmit buffer event has been detected. It is cleared by writing TRBSCR.CSTBI = 1. If enabled by TBCTR.STBIEN, the service request output SRx selected by TBCTR.STBINP becomes activated if a standard transmit buffer event is detected. 0_B A standard transmit buffer event has not been detected. 1_B A standard transmit buffer event has been detected.
TBERI	9	rwh	**Transmit Buffer Error Event** This bit indicates that a transmit buffer error event has been detected. It is cleared by writing TRBSCR.CTBERI = 1. If enabled by TBCTR.TBERIEN, the service request output SRx selected by TBCTR.ATBINP becomes activated if a transmit buffer error event is detected. 0_B A transmit buffer error event has not been detected. 1_B A transmit buffer error event has been detected.
TEMPTY	11	rh	**Transmit Buffer Empty** This bit indicates whether the transmit buffer is empty. 0_B The transmit buffer is not empty. 1_B The transmit buffer is empty.
TFULL	12	rh	**Transmit Buffer Full** This bit indicates whether the transmit buffer is full. 0_B The transmit buffer is not full. 1_B The transmit buffer is full.

Field	Bits	Type	Description
TBUS	13	rh	**Transmit Buffer Busy** This bit indicates whether the transmit buffer is currently updated by the FIFO handler. 0_B The transmit buffer information has been completely updated. 1_B The FIFO memory update after write to INx is ongoing. A write to INx will be delayed. FIFO pointers from the previous INx write are not yet updated.
STBT	14	rh	**Standard Transmit Buffer Event Trigger** This bit triggers a standard transmit buffer event when set. If enabled by TBCTR.STBIEN, the service request output SRx selected by TBCTR.STBINP becomes activated until the bit is cleared. 0_B A standard transmit buffer event is not triggered using this bit. 1_B A standard transmit buffer event is triggered using this bit.
RBFLVL	[22:16]	rh	**Receive Buffer Filling Level** This bit field indicates the filling level of the receive buffer, starting with 0 for an empty buffer.
TBFLVL	[30:24]	rh	**Transmit Buffer Filling Level** This bit field indicates the filling level of the transmit buffer, starting with 0 for an empty buffer.
0	7, 10, 15, 23, 31	r	**Reserved** Read as 0; should be written with 0.

The bits in register TRBSCR are used to clear the notification bits in register TRBSR or to clear the FIFO mechanism for the transmit or receive buffer. A read action always delivers 0.

TRBSCR
Transmit/Receive Buffer Status Clear Register
(118_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FLUSHTB	FLUSHRB	0			CBDV	CTBERI	CSTBI	0					CARBI	CRBERI	CSRBI
w	w	r			w	w	w	r					w	w	w

Field	Bits	Type	Description
CSRBI	0	w	**Clear Standard Receive Buffer Event** 0_B No effect. 1_B Clear TRBSR.SRBI.
CRBERI	1	w	**Clear Receive Buffer Error Event** 0_B No effect. 1_B Clear TRBSR.RBERI.
CARBI	2	w	**Clear Alternative Receive Buffer Event** 0_B No effect. 1_B Clear TRBSR.ARBI.
CSTBI	8	w	**Clear Standard Transmit Buffer Event** 0_B No effect. 1_B Clear TRBSR.STBI.
CTBERI	9	w	**Clear Transmit Buffer Error Event** 0_B No effect. 1_B Clear TRBSR.TBERI.
CBDV	10	w	**Clear Bypass Data Valid** 0_B No effect. 1_B Clear BYPCR.BDV.

Field	Bits	Type	Description
FLUSHRB	14	w	**Flush Receive Buffer** 0_B No effect. 1_B The receive FIFO buffer is cleared (filling level is cleared and output pointer is set to input pointer value). Should only be used while the FIFO buffer is not taking part in data traffic.
FLUSHTB	15	w	**Flush Transmit Buffer** 0_B No effect. 1_B The transmit FIFO buffer is cleared (filling level is cleared and output pointer is set to input pointer value). Should only be used while the FIFO buffer is not taking part in data traffic.
0	[7:3], [13:11], [31:16]	r	**Reserved** Read as 0; should be written with 0.

17.11.9.3 Transmit FIFO Buffer Control Registers

The transmit FIFO buffer is controlled by register TBCTR. TBCTR can only by written if the transmit buffer functionality is enabled by CCFG.TB = 1, otherwise write accesses are ignored.

TBCTR
Transmitter Buffer Control Register (108_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TBERIEN	**STBIEN**	**0**	**LOF**	**0**	**SIZE**			**0**		**ATBINP**			**STBINP**		
rw	rw	r	rw	r	rw			r		rw			rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STBTEN	**STBTM**	**LIMIT**						**0**		**DPTR**					
rw	rw	rw						r		w					

Field	Bits	Type	Description
DPTR	[5:0]	w	**Data Pointer** This bit field defines the start value for the transmit buffer pointers when assigning the FIFO entries to the transmit FIFO buffer. A read always delivers 0. When writing DPTR while SIZE = 0, both transmitter pointers TDIPTR and RTDOPTR in register TRBPTR are updated with the written value and the buffer is considered as empty. A write access to DPTR while SIZE > 0 is ignored and does not modify the pointers.
LIMIT	[13:8]	rw	**Limit For Interrupt Generation** This bit field defines the target filling level of the transmit FIFO buffer that is used for the standard transmit buffer event detection.
STBTM	14	rw	**Standard Transmit Buffer Trigger Mode** This bit selects the standard transmit buffer event trigger mode. 0_B Trigger mode 0: If enabled by TBCTR.STBTEN = 1, the transition of the transmit buffer fill level away from the programmed limit, sets the bit TRBSR.STBT. 1_B Trigger mode 1: If enabled by TBCTR.STBTEN = 1, the condition whereby the transmit buffer fill level is below the buffer size, sets the bit TRBSR.STBT.
STBTEN	15	rw	**Standard Transmit Buffer Trigger Enable** This bit enables/disables triggering of the standard transmit buffer event through bit TRBSR.STBT. 0_B The standard transmit buffer event trigger through bit TRBSR.STBT is disabled. 1_B The standard transmit buffer event trigger through bit TRBSR.STBT is enabled.

Field	Bits	Type	Description
STBINP	[18:16]	rw	**Standard Transmit Buffer Interrupt Node Pointer** This bit field defines which service request output SRx becomes activated in case of a standard transmit buffer event. 000_B Output SR0 becomes activated. 001_B Output SR1 becomes activated. 010_B Output SR2 becomes activated. 011_B Output SR3 becomes activated. 100_B Output SR4 becomes activated. 101_B Output SR5 becomes activated. *Note: All other settings of the bit field are reserved.*
ATBINP	[21:19]	rw	**Alternative Transmit Buffer Interrupt Node Pointer** This bit field define which service request output SRx will be activated in case of a transmit buffer error event. 000_B Output SR0 becomes activated. 001_B Output SR1 becomes activated. 010_B Output SR2 becomes activated. 011_B Output SR3 becomes activated. 100_B Output SR4 becomes activated. 101_B Output SR5 becomes activated. *Note: All other settings of the bit field are reserved.*
SIZE	[26:24]	rw	**Buffer Size** This bit field defines the number of FIFO entries assigned to the transmit FIFO buffer. 000_B The FIFO mechanism is disabled. The buffer does not accept any request for data. 001_B The FIFO buffer contains 2 entries. 010_B The FIFO buffer contains 4 entries. 011_B The FIFO buffer contains 8 entries. 100_B The FIFO buffer contains 16 entries. 101_B The FIFO buffer contains 32 entries. 110_B The FIFO buffer contains 64 entries. 111_B Reserved

Field	Bits	Type	Description
LOF	28	rw	**Buffer Event on Limit Overflow** This bit defines which relation between filling level and programmed limit leads to a standard transmit buffer event. 0_B A standard transmit buffer event occurs when the filling level equals the limit value and gets lower due to transmission of a data word. 1_B A standard transmit buffer interrupt event occurs when the filling level equals the limit value and gets bigger due to a write access to a data input location INx.
STBIEN	30	rw	**Standard Transmit Buffer Interrupt Enable** This bit enables/disables the generation of a standard transmit buffer interrupt in case of a standard transmit buffer event. 0_B The standard transmit buffer interrupt generation is disabled. 1_B The standard transmit buffer interrupt generation is enabled.
TBERIEN	31	rw	**Transmit Buffer Error Interrupt Enable** This bit enables/disables the generation of a transmit buffer error interrupt in case of a transmit buffer error event (software writes to a full transmit buffer). 0_B The transmit buffer error interrupt generation is disabled. 1_B The transmit buffer error interrupt generation is enabled.
0	[7:6], [23:22], 27, 29	r	**Reserved** Read as 0; should be written with 0.

17.11.9.4 Receive FIFO Buffer Control Registers

The receive FIFO buffer is controlled by register RBCTR. This register can only be written if the receive buffer functionality is enabled by CCFG.RB = 1, otherwise write accesses are ignored.

RBCTR
Receiver Buffer Control Register ($10C_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RBERIEN	SRBIEN	ARBIEN	LOF	RNM	SIZE			RCIM		ARBINP			SRBINP		
rw	rw	rw	rw	rw	rw			rw		rw			rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SRBTEN	SRBTM	LIMIT						0		DPTR					
rw	rw	rw						r		w					

Field	Bits	Type	Description
DPTR	[5:0]	w	**Data Pointer** This bit field defines the start value for the receive buffer pointers when assigning the FIFO entries to the receive FIFO buffer. A read always delivers 0. When writing DPTR while SIZE = 0, both receiver pointers RDIPTR and RDOPTR in register TRBPTR are updated with the written value and the buffer is considered as empty. A write access to DPTR while SIZE > 0 is ignored and does not modify the pointers.
LIMIT	[13:8]	rw	**Limit For Interrupt Generation** This bit field defines the target filling level of the receive FIFO buffer that is used for the standard receive buffer event detection.

Field	Bits	Type	Description
SRBTM	14	rw	**Standard Receive Buffer Trigger Mode** This bit selects the standard receive buffer event trigger mode. 0_B Trigger mode 0: If enabled by RBCTR.SRBTEN = 1, the transition of the transmit buffer fill level away from the programmed limit, sets the bit TRBSR.SRBT. 1_B Trigger mode 1: If enabled by RBCTR.SRBTEN = 1, the condition whereby the transmit buffer fill level is above 0, sets the bit TRBSR.SRBT.
SRBTEN	15	rw	**Standard Receive Buffer Trigger Enable** This bit enables/disables triggering of the standard receive buffer event through bit TRBSR.SRBT. 0_B The standard receive buffer event trigger through bit TRBSR.SRBT is disabled. 1_B The standard receive buffer event trigger through bit TRBSR.SRBT is enabled.
SRBINP	[18:16]	rw	**Standard Receive Buffer Interrupt Node Pointer** This bit field defines which service request output SRx becomes activated in case of a standard receive buffer event. 000_B Output SR0 becomes activated. 001_B Output SR1 becomes activated. 010_B Output SR2 becomes activated. 011_B Output SR3 becomes activated. 100_B Output SR4 becomes activated. 101_B Output SR5 becomes activated. *Note: All other settings of the bit field are reserved.*

Field	Bits	Type	Description
ARBINP	[21:19]	rw	**Alternative Receive Buffer Interrupt Node Pointer** This bit field defines which service request output SRx becomes activated in case of an alternative receive buffer event or a receive buffer error event. 000_B Output SR0 becomes activated. 001_B Output SR1 becomes activated. 010_B Output SR2 becomes activated. 011_B Output SR3 becomes activated. 100_B Output SR4 becomes activated. 101_B Output SR5 becomes activated. *Note: All other settings of the bit field are reserved.*
RCIM	[23:22]	rw	**Receiver Control Information Mode** This bit field defines which information from the receiver status register RBUFSR is propagated as 5 bit receiver control information RCI[4:0] to the receive FIFO buffer and can be read out in registers OUT(D)R. 00_B RCI[4] = PERR, RCI[3:0] = WLEN 01_B RCI[4] = SOF, RCI[3:0] = WLEN 10_B RCI[4] = 0, RCI[3:0] = WLEN 11_B RCI[4] = PERR, RCI[3] = PAR, RCI[2:1] = 00_B, RCI[0] = SOF
SIZE	[26:24]	rw	**Buffer Size** This bit field defines the number of FIFO entries assigned to the receive FIFO buffer. 000_B The FIFO mechanism is disabled. The buffer does not accept any request for data. 001_B The FIFO buffer contains 2 entries. 010_B The FIFO buffer contains 4 entries. 011_B The FIFO buffer contains 8 entries. 100_B The FIFO buffer contains 16 entries. 101_B The FIFO buffer contains 32 entries. 110_B The FIFO buffer contains 64 entries. 111_B Reserved

Field	Bits	Type	Description
RNM	27	rw	**Receiver Notification Mode** This bit defines the receive buffer event mode. The receive buffer error event is not affected by RNM. 0_B Filling level mode: A standard receive buffer event occurs when the filling level equals the limit value and changes, either due to a read access from OUTR (LOF = 0) or due to a new received data word (LOF = 1). 1_B RCI mode: A standard receive buffer event occurs when register OUTR is updated with a new value if the corresponding value in OUTR.RCI[4] = 0. If OUTR.RCI[4] = 1, an alternative receive buffer event occurs instead of the standard receive buffer event.
LOF	28	rw	**Buffer Event on Limit Overflow** This bit defines which relation between filling level and programmed limit leads to a standard receive buffer event in filling level mode (RNM = 0). In RCI mode (RNM = 1), bit fields LIMIT and LOF are ignored. 0_B A standard receive buffer event occurs when the filling level equals the limit value and gets lower due to a read access from OUTR. 1_B A standard receive buffer event occurs when the filling level equals the limit value and gets bigger due to the reception of a new data word.
ARBIEN	29	rw	**Alternative Receive Buffer Interrupt Enable** This bit enables/disables the generation of an alternative receive buffer interrupt in case of an alternative receive buffer event. 0_B The alternative receive buffer interrupt generation is disabled. 1_B The alternative receive buffer interrupt generation is enabled.

Field	Bits	Type	Description
SRBIEN	30	rw	**Standard Receive Buffer Interrupt Enable** This bit enables/disables the generation of a standard receive buffer interrupt in case of a standard receive buffer event. 0_B The standard receive buffer interrupt generation is disabled. 1_B The standard receive buffer interrupt generation is enabled.
RBERIEN	31	rw	**Receive Buffer Error Interrupt Enable** This bit enables/disables the generation of a receive buffer error interrupt in case of a receive buffer error event (the software reads from an empty receive buffer). 0_B The receive buffer error interrupt generation is disabled. 1_B The receive buffer error interrupt generation is enabled.
0	[7:6]	r	**Reserved** Read as 0; should be written with 0.

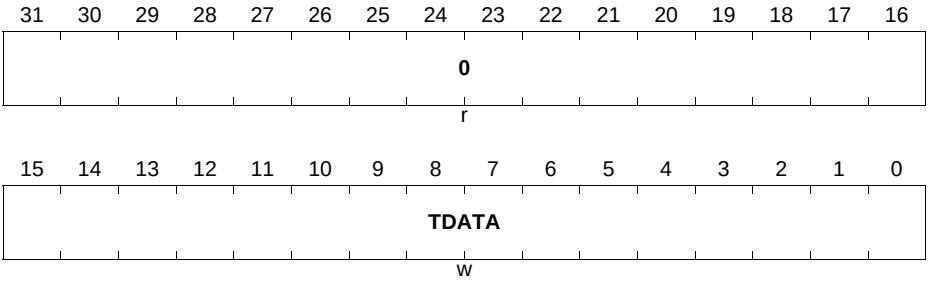
17.11.9.5 FIFO Buffer Data Registers

The 32 independent data input locations IN00 to IN31 are addresses that can be used as data entry locations for the transmit FIFO buffer. Data written to one of these locations will be stored in the transmit buffer FIFO. Additionally, the 5-bit coding of the number [31:0] of the addressed data input location represents the transmit control information TCI.

If the FIFO is already full and new data is written to it, the write access is ignored and a transmit buffer error event is signaled.

INx (x = 00-31)
Transmit FIFO Buffer Input Location x

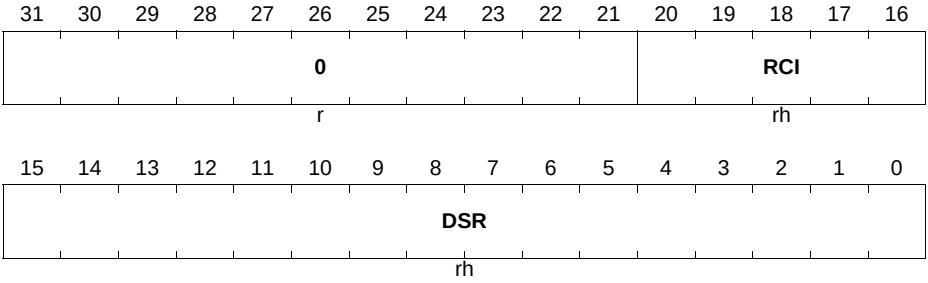
(180_H + x *4) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
TDATA	[15:0]	w	**Transmit Data** This bit field contains the data to be transmitted (write view), read actions deliver 0. A write action to at least the low byte of TDATA triggers the data storage in the FIFO.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

The receiver FIFO buffer output register OUTR shows the oldest received data word in the FIFO buffer and contains the receiver control information RCI containing the information selected by RBCTR.RCIM. A read action from this address location delivers the received data. With a read access of at least the low byte, the data is declared to be read and the next entry becomes visible. Write accesses to OUTR are ignored.

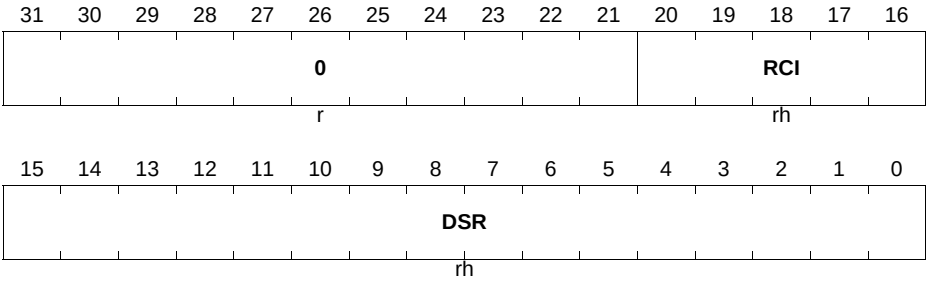
OUTR
Receiver Buffer Output Register (11C$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
DSR	[15:0]	rh	**Received Data** This bit field monitors the content of the oldest data word in the receive FIFO. Reading at least the low byte releases the buffer entry currently shown in DSR.
RCI	[20:16]	rh	**Receiver Control Information** This bit field monitors the receiver control information associated to DSR. The bit structure of RCI depends on bit field RBCTR.RCIM.
0	[31:21]	r	**Reserved** Read as 0; should be written with 0.

If a debugger should be used to monitor the received data in the FIFO buffer, the FIFO mechanism must not be activated in order to guaranty data consistency. Therefore, a second address set is available, named OUTDR (D like debugger), having the same bit fields like the original buffer output register OUTR, but without the FIFO mechanism. A debugger can read here (in order to monitor the receive data flow) without the risk of data corruption. Write accesses to OUTDR are ignored.

OUTDR
Receiver Buffer Output Register L for Debugger
(120_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
DSR	[15:0]	rh	**Data from Shift Register** Same as OUTR.DSR, but without releasing the buffer after a read action.
RCI	[20:16]	rh	**Receive Control Information from Shift Register** Same as OUTR.RCI.
0	[31:21]	r	**Reserved** Read as 0; should be written with 0.

17.11.9.6 FIFO Buffer Pointer Registers

The pointers for FIFO handling of the transmit and receive FIFO buffers are located in register TRBPTR. The pointers are automatically handled by the FIFO buffer mechanism and do not need to be modified by software. As a consequence, these registers can only be read by software (e.g. for verification purposes), whereas write accesses are ignored.

TRBPTR
Transmit/Receive Buffer Pointer Register

(110_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		RDOPTR						0		RDIPTR					
r		rh						r		rh					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		TDOPTR						0		TDIPTR					
r		rh						r		rh					

Field	Bits	Type	Description
TDIPTR	[5:0]	rh	**Transmitter Data Input Pointer** This bit field indicates the buffer entry that will be used for the next transmit data coming from the INx addresses.
TDOPTR	[13:8]	rh	**Transmitter Data Output Pointer** This bit field indicates the buffer entry that will be used for the next transmit data to be output to TBUF.
RDIPTR	[21:16]	rh	**Receiver Data Input Pointer** This bit field indicates the buffer entry that will be used for the next receive data coming from RBUF.
RDOPTR	[29:24]	rh	**Receiver Data Output Pointer** This bit field indicates the buffer entry that will be used for the next receive data to be output at the OUT(D)R addresses.
0	[7:6], [15:14], [23:22], [31:30]	r	**Reserved** Read as 0; should be written with 0.

17.12 Interconnects

The XMC4500 device contains three USIC modules (USIC0, USIC1 and USIC2) with 2 communication channels each.

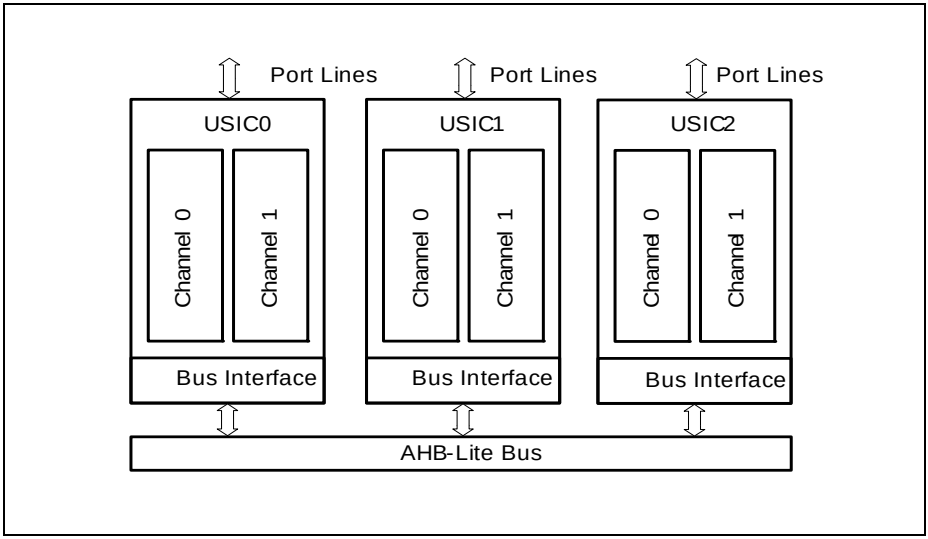


Figure 17-65 USIC Module Structure in XMC4500

Figure 17-66 shows the I/O lines of one USIC channel. The tables in this section define the pin assignments and internal connections of the USIC channels I/O lines in the XMC4500 device. Naming convention: USICx_CHy refers to USIC module x channel y.

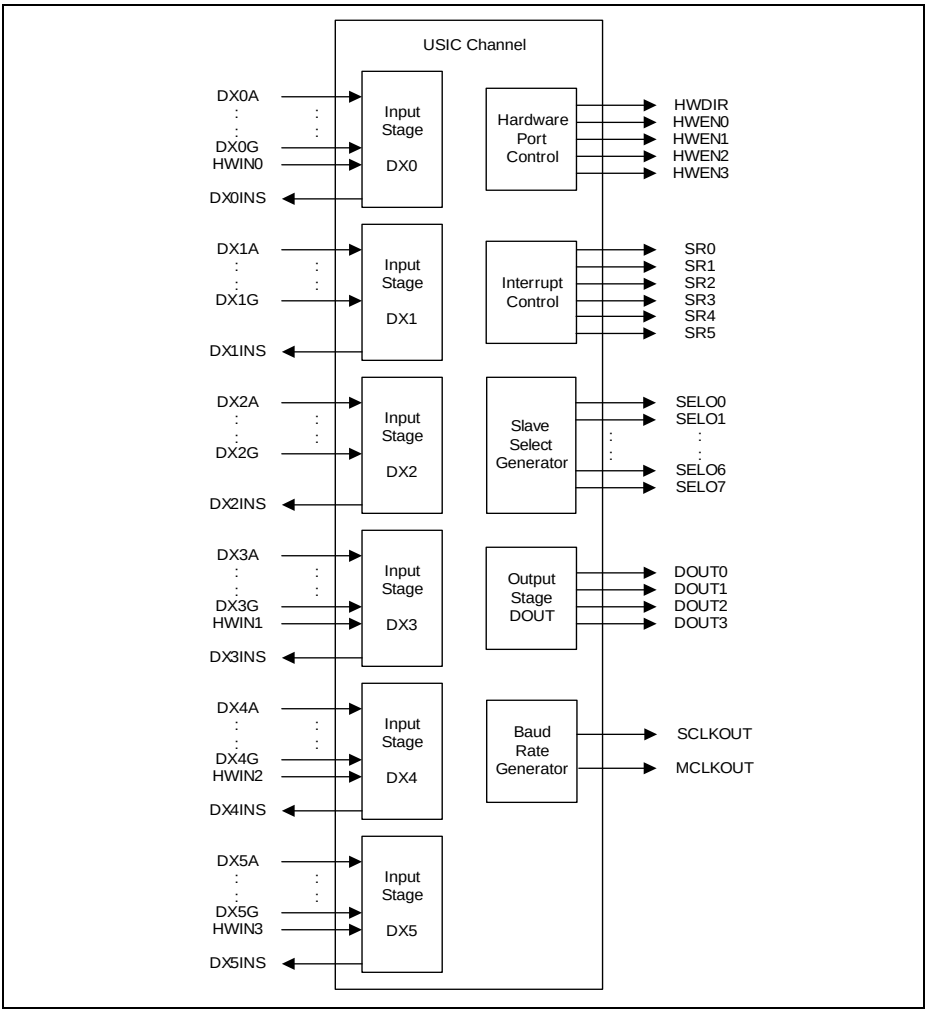


Figure 17-66 USIC Channel I/O Lines

The connections of the service request outputs SR[5:0] are described in **Section 17.7**.

17.12.1 USIC Module 0 Interconnects

The interconnects of USIC module 0 is grouped into the following categories:

- **USIC Module 0 Channel 0 Interconnects** (**Table 17-22**)

- **USIC Module 0 Channel 1 Interconnects** (**Table 17-23**)

Table 17-22 USIC Module 0 Channel 0 Interconnects

Input/Output	I/O	Connected To	Description
Data Inputs (DX0)			
USIC0_CH0.DX0A	I	P1.5	Shift data input
USIC0_CH0.DX0B	I	P1.4	Shift data input
USIC0_CH0.DX0C	I	P4.7	Shift data input
USIC0_CH0.DX0D	I	P5.0	Shift data input
USIC0_CH0.DX0E	I	0	Shift data input
USIC0_CH0.DX0F	I	XTAL1	Shift data input
USIC0_CH0.DX0G	I	USIC0_CH0.DOUT0	Loop back shift data input
USIC0_CH0.HWIN0	I	P1.5	HW controlled shift data input
Clock Inputs			
USIC0_CH0.DX1A	I	P1.1	Shift clock input
USIC0_CH0.DX1B	I	P0.8	Shift clock input
USIC0_CH0.DX1C	I	0	Shift clock input
USIC0_CH0.DX1D	I	0	Shift clock input
USIC0_CH0.DX1E	I	0	Shift clock input
USIC0_CH0.DX1F	I	USIC0_CH0.DX0INS	Shift clock input
USIC0_CH0.DX1G	I	USIC0_CH0.SCLKOUT	Loop back shift clock input
Control Inputs			
USIC0_CH0.DX2A	I	P1.0	Shift control input
USIC0_CH0.DX2B	I	P0.7	Shift control input
USIC0_CH0.DX2C	I	0	Shift control input
USIC0_CH0.DX2D	I	0	Shift control input
USIC0_CH0.DX2E	I	CCU40.SR1	Shift control input
USIC0_CH0.DX2F	I	CCU80.SR1	Shift control input
USIC0_CH0.DX2G	I	USIC0_CH0.SELO0	Loop back shift control input
Data Inputs (DX3)			
USIC0_CH0.DX3A	I	0	Shift data input
USIC0_CH0.DX3B	I	0	Shift data input

Table 17-22 USIC Module 0 Channel 0 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC0_CH0.DX3C	I	0	Shift data input
USIC0_CH0.DX3D	I	0	Shift data input
USIC0_CH0.DX3E	I	0	Shift data input
USIC0_CH0.DX3F	I	0	Shift data input
USIC0_CH0.DX3G	I	USIC0_CH0.DOUT1	Loop back shift data input
USIC0_CH0.HWIN1	I	P1.4	HW controlled shift data input
Data Inputs (DX4)			
USIC0_CH0.DX4A	I	0	Shift data input
USIC0_CH0.DX4B	I	0	Shift data input
USIC0_CH0.DX4C	I	0	Shift data input
USIC0_CH0.DX4D	I	0	Shift data input
USIC0_CH0.DX4E	I	0	Shift data input
USIC0_CH0.DX4F	I	0	Shift data input
USIC0_CH0.DX4G	I	USIC0_CH0.DOUT2	Loop back shift data input
USIC0_CH0.HWIN2	I	P1.3	HW controlled shift data input
Data Inputs (DX5)			
USIC0_CH0.DX5A	I	0	Shift data input
USIC0_CH0.DX5B	I	0	Shift data input
USIC0_CH0.DX5C	I	0	Shift data input
USIC0_CH0.DX5D	I	0	Shift data input
USIC0_CH0.DX5E	I	0	Shift data input
USIC0_CH0.DX5F	I	0	Shift data input
USIC0_CH0.DX5G	I	USIC0_CH0.DOUT3	Loop back shift data input
USIC0_CH0.HWIN3	I	P1.2	HW controlled shift data input
Data Outputs			
USIC0_CH0.DOUT0	O	P1.5 P1.7 P5.1 P1.5.HW0_OUT	Shift data output
USIC0_CH0.DOUT1	O	P1.4.HW0_OUT	Shift data output
USIC0_CH0.DOUT2	O	P1.3.HW0_OUT	Shift data output

Table 17-22 USIC Module 0 Channel 0 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC0_CH0.DOUT3	O	P1.2.HW0_OUT	Shift data output
Clock Outputs			
USIC0_CH0.MCLKOUT	O	P1.3	Master clock output
USIC0_CH0.SCLKOUT	O	P0.8 P1.1 P1.6 P1.10	Shift clock output
Control Outputs			
USIC0_CH0.SELO0	O	P0.7 P1.0 P1.11	Shift control output
USIC0_CH0.SELO1	O	P1.8	Shift control output
USIC0_CH0.SELO2	O	P4.6	Shift control output
USIC0_CH0.SELO3	O	P4.5	Shift control output
USIC0_CH0.SELO4	O	P4.4	Shift control output
USIC0_CH0.SELO5	O	P4.3	Shift control output
USIC0_CH0.SELO6	O	not connected	Shift control output
USIC0_CH0.SELO7	O	not connected	Shift control output
System Related Outputs			
USIC0_CH0.DX0INS	O	USIC0_CH0.DX1F	Selected DX0 input signal
USIC0_CH0.DX1INS	O	DAC.TRIGGER[6]	Selected DX1 input signal
USIC0_CH0.DX2INS	O	CCU40.IN0L CCU42.IN0L CCU43.IN0L	Selected DX2 input signal
USIC0_CH0.DX3INS	O	not connected	Selected DX3 input signal
USIC0_CH0.DX4INS	O	not connected	Selected DX4 input signal
USIC0_CH0.DX5INS	O	not connected	Selected DX5 input signal

Table 17-23 USIC Module 0 Channel 1 Interconnects

Input/Output	I/O	Connected To	Description
Data Inputs (DX0)			
USIC0_CH1.DX0A	I	P2.2	Shift data input
USIC0_CH1.DX0B	I	P2.5	Shift data input
USIC0_CH1.DX0C	I	P6.3	Shift data input
USIC0_CH1.DX0D	I	P3.13	Shift data input
USIC0_CH1.DX0E	I	P4.0	Shift data input
USIC0_CH1.DX0F	I	XTAL1	Shift data input
USIC0_CH1.DX0G	I	USIC0_CH1.DOUT0	Loop back shift data input
USIC0_CH1.HWIN0	I	P3.13	HW controlled shift data input
Clock Inputs			
USIC0_CH1.DX1A	I	P2.4	Shift clock input
USIC0_CH1.DX1B	I	P3.0	Shift clock input
USIC0_CH1.DX1C	I	P6.2	Shift clock input
USIC0_CH1.DX1D	I	0	Shift clock input
USIC0_CH1.DX1E	I	0	Shift clock input
USIC0_CH1.DX1F	I	USIC0_CH1.DX0INS	Shift clock input
USIC0_CH1.DX1G	I	USIC0_CH1.SCLKOUT	Loop back shift clock input
Control Inputs			
USIC0_CH1.DX2A	I	P2.3	Shift control input
USIC0_CH1.DX2B	I	P3.1	Shift control input
USIC0_CH1.DX2C	I	P6.1	Shift control input
USIC0_CH1.DX2D	I	0	Shift control input
USIC0_CH1.DX2E	I	CCU42.SR1	Shift control input
USIC0_CH1.DX2F	I	CCU80.SR1	Shift control input
USIC0_CH1.DX2G	I	USIC0_CH1.SELO0	Loop back shift control input
Data Inputs (DX3)			
USIC0_CH1.DX3A	I	0	Shift data input
USIC0_CH1.DX3B	I	0	Shift data input
USIC0_CH1.DX3C	I	0	Shift data input

Table 17-23 USIC Module 0 Channel 1 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC0_CH1.DX3D	I	0	Shift data input
USIC0_CH1.DX3E	I	0	Shift data input
USIC0_CH1.DX3F	I	0	Shift data input
USIC0_CH1.DX3G	I	USIC0_CH1.DOUT1	Loop back shift data input
USIC0_CH1.HWIN1	I	P3.12	HW controlled shift data input
Data Inputs (DX4)			
USIC0_CH1.DX4A	I	0	Shift data input
USIC0_CH1.DX4B	I	0	Shift data input
USIC0_CH1.DX4C	I	0	Shift data input
USIC0_CH1.DX4D	I	0	Shift data input
USIC0_CH1.DX4E	I	0	Shift data input
USIC0_CH1.DX4F	I	0	Shift data input
USIC0_CH1.DX4G	I	USIC0_CH1.DOUT2	Loop back shift data input
USIC0_CH1.HWIN2	I	P3.11	HW controlled shift data input
Data Inputs (DX5)			
USIC0_CH1.DX5A	I	0	Shift data input
USIC0_CH1.DX5B	I	0	Shift data input
USIC0_CH1.DX5C	I	0	Shift data input
USIC0_CH1.DX5D	I	0	Shift data input
USIC0_CH1.DX5E	I	0	Shift data input
USIC0_CH1.DX5F	I	0	Shift data input
USIC0_CH1.DX5G	I	USIC0_CH1.DOUT3	Loop back shift data input
USIC0_CH1.HWIN3	I	P3.10	HW controlled shift data input
Data Outputs			
USIC0_CH1.DOUT0	O	P2.5 P3.5 P3.13 P6.4 P3.13.HW0_OUT	Shift data output
USIC0_CH1.DOUT1	O	P3.12.HW0_OUT	Shift data output
USIC0_CH1.DOUT2	O	P3.11.HW0_OUT	Shift data output

Table 17-23 USIC Module 0 Channel 1 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC0_CH1.DOUT3	O	P3.10.HW0_OUT	Shift data output
Clock Outputs			
USIC0_CH1.MCLKOUT	O	P6.5	Master clock output
USIC0_CH1.SCLKOUT	O	P2.4 P3.0 P3.6 P6.2	Shift clock output
Control Outputs			
USIC0_CH1.SELO0	O	P2.3 P3.1 P4.1 P6.1	Shift control output
USIC0_CH1.SELO1	O	P3.12 P6.0	Shift control output
USIC0_CH1.SELO2	O	P1.14 P3.11	Shift control output
USIC0_CH1.SELO3	O	P1.13 P3.8	Shift control output
USIC0_CH1.SELO4	O	not connected	Shift control output
USIC0_CH1.SELO5	O	not connected	Shift control output
USIC0_CH1.SELO6	O	not connected	Shift control output
USIC0_CH1.SELO7	O	not connected	Shift control output
System Related Outputs			
USIC0_CH1.DX0INS	O	USIC0_CH1.DX1F	Selected DX0 input signal
USIC0_CH1.DX1INS	O	not connected	Selected DX1 input signal
USIC0_CH1.DX2INS	O	CCU40.IN2L CCU42.IN1L CCU43.IN1L	Selected DX2 input signal
USIC0_CH1.DX3INS	O	not connected	Selected DX3 input signal
USIC0_CH1.DX4INS	O	not connected	Selected DX4 input signal
USIC0_CH1.DX5INS	O	not connected	Selected DX5 input signal

17.12.2 USIC Module 1 Interconnects

The interconnects of USIC module 1 is grouped into the following three categories:

- **USIC Module 1 Channel 0 Interconnects** (**Table 17-24**)
- **USIC Module 1 Channel 1 Interconnects** (**Table 17-25**)

Table 17-24 USIC Module 1 Channel 0 Interconnects

Input/Output	I/O	Connected To	Description
Data Inputs (DX0)			
USIC1_CH0.DX0A	I	P0.4	Shift data input
USIC1_CH0.DX0B	I	P0.5	Shift data input
USIC1_CH0.DX0C	I	P2.15	Shift data input
USIC1_CH0.DX0D	I	P2.14	Shift data input
USIC1_CH0.DX0E	I	0	Shift data input
USIC1_CH0.DX0F	I	XTAL1	Shift data input
USIC1_CH0.DX0G	I	USIC1_CH0.DOUT0	Loop back shift data input
USIC1_CH0.HWIN0	I	P0.5	HW controlled shift data input
Clock Inputs			
USIC1_CH0.DX1A	I	P0.11	Shift clock input
USIC1_CH0.DX1B	I	P5.8	Shift clock input
USIC1_CH0.DX1C	I	0	Shift clock input
USIC1_CH0.DX1D	I	0	Shift clock input
USIC1_CH0.DX1E	I	0	Shift clock input
USIC1_CH0.DX1F	I	USIC1_CH0.DX0INS	Shift clock input
USIC1_CH0.DX1G	I	USIC1_CH0.SCLKOUT	Loop back shift clock input
Control Inputs			
USIC1_CH0.DX2A	I	P0.6	Shift control input
USIC1_CH0.DX2B	I	P5.9	Shift control input
USIC1_CH0.DX2C	I	0	Shift control input
USIC1_CH0.DX2D	I	0	Shift control input
USIC1_CH0.DX2E	I	CCU41.SR1	Shift control input
USIC1_CH0.DX2F	I	CCU81.SR1	Shift control input
USIC1_CH0.DX2G	I	USIC1_CH0.SELO0	Loop back shift control input

Table 17-24 USIC Module 1 Channel 0 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
Data Inputs (DX3)			
USIC1_CH0.DX3A	I	0	Shift data input
USIC1_CH0.DX3B	I	0	Shift data input
USIC1_CH0.DX3C	I	0	Shift data input
USIC1_CH0.DX3D	I	0	Shift data input
USIC1_CH0.DX3E	I	0	Shift data input
USIC1_CH0.DX3F	I	0	Shift data input
USIC1_CH0.DX3G	I	USIC1_CH0.DOUT1	Loop back shift data input
USIC1_CH0.HWIN1	I	P0.4	HW controlled shift data input
Data Inputs (DX4)			
USIC1_CH0.DX4A	I	0	Shift data input
USIC1_CH0.DX4B	I	0	Shift data input
USIC1_CH0.DX4C	I	0	Shift data input
USIC1_CH0.DX4D	I	0	Shift data input
USIC1_CH0.DX4E	I	0	Shift data input
USIC1_CH0.DX4F	I	0	Shift data input
USIC1_CH0.DX4G	I	USIC1_CH0.DOUT2	Loop back shift data input
USIC1_CH0.HWIN2	I	P0.3	HW controlled shift data input
Data Inputs (DX5)			
USIC1_CH0.DX5A	I	0	Shift data input
USIC1_CH0.DX5B	I	0	Shift data input
USIC1_CH0.DX5C	I	0	Shift data input
USIC1_CH0.DX5D	I	0	Shift data input
USIC1_CH0.DX5E	I	0	Shift data input
USIC1_CH0.DX5F	I	0	Shift data input
USIC1_CH0.DX5G	I	USIC1_CH0.DOUT3	Loop back shift data input
USIC1_CH0.HWIN3	I	P0.2	HW controlled shift data input
Data Outputs			
USIC1_CH0.DOUT0	O	P0.5 P2.14 P0.5.HW0_OUT	Shift data output

Table 17-24 USIC Module 1 Channel 0 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC1_CH0.DOUT1	O	P0.4.HW0_OUT	Shift data output
USIC1_CH0.DOUT2	O	P0.3.HW0_OUT	Shift data output
USIC1_CH0.DOUT3	O	P0.2.HW0_OUT	Shift data output
Clock Outputs			
USIC1_CH0.MCLKOUT	O	P5.10	Master clock output
USIC1_CH0.SCLKOUT	O	P0.11 P5.8	Shift clock output
Control Outputs			
USIC1_CH0.SELO0	O	P0.6 P5.9	Shift control output
USIC1_CH0.SELO1	O	P0.14 P5.11	Shift control output
USIC1_CH0.SELO2	O	P0.15	Shift control output
USIC1_CH0.SELO3	O	P3.14	Shift control output
USIC1_CH0.SELO4	O	not connected	Shift control output
USIC1_CH0.SELO5	O	not connected	Shift control output
USIC1_CH0.SELO6	O	not connected	Shift control output
USIC1_CH0.SELO7	O	not connected	Shift control output
System Related Outputs			
USIC1_CH0.DX0INS	O	USIC1_CH0.DX1F	Selected DX0 input signal
USIC1_CH0.DX1INS	O	DAC.TRIGGER[7]	Selected DX1 input signal
USIC1_CH0.DX2INS	O	CCU40.IN3L CCU42.IN2L CCU43.IN2L	Selected DX2 input signal
USIC1_CH0.DX3INS	O	not connected	Selected DX3 input signal
USIC1_CH0.DX4INS	O	not connected	Selected DX4 input signal
USIC1_CH0.DX5INS	O	not connected	Selected DX5 input signal

Table 17-25 USIC Module 1 Channel 1 Interconnects

Input/Output	I/O	Connected To	Description
Data Inputs (DX0)			
USIC1_CH1.DX0A	I	P3.15	Shift data input
USIC1_CH1.DX0B	I	P3.14	Shift data input
USIC1_CH1.DX0C	I	P4.2	Shift data input
USIC1_CH1.DX0D	I	P0.0	Shift data input
USIC1_CH1.DX0E	I	CAN1INS	Shift data input
USIC1_CH1.DX0F	I	XTAL1	Shift data input
USIC1_CH1.DX0G	I	USIC1_CH1.DOUT0	Loop back shift data input
USIC1_CH1.HWIN0	I	P3.15	HW controlled shift data input
Clock Inputs			
USIC1_CH1.DX1A	I	P0.10	Shift clock input
USIC1_CH1.DX1B	I	P0.13	Shift clock input
USIC1_CH1.DX1C	I	P4.0	Shift clock input
USIC1_CH1.DX1D	I	0	Shift clock input
USIC1_CH1.DX1E	I	0	Shift clock input
USIC1_CH1.DX1F	I	USIC1_CH1.DX0INS	Shift clock input
USIC1_CH1.DX1G	I	USIC1_CH1.SCLKOUT	Loop back shift clock input
Control Inputs			
USIC1_CH1.DX2A	I	P0.9	Shift control input
USIC1_CH1.DX2B	I	P0.12	Shift control input
USIC1_CH1.DX2C	I	0	Shift control input
USIC1_CH1.DX2D	I	0	Shift control input
USIC1_CH1.DX2E	I	CCU43.SR1	Shift control input
USIC1_CH1.DX2F	I	CCU81.SR1	Shift control input
USIC1_CH1.DX2G	I	USIC1_CH1.SELO0	Loop back shift control input
Data Inputs (DX3)			
USIC1_CH1.DX3A	I	0	Shift data input
USIC1_CH1.DX3B	I	0	Shift data input
USIC1_CH1.DX3C	I	0	Shift data input

Table 17-25 USIC Module 1 Channel 1 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC1_CH1.DX3D	I	0	Shift data input
USIC1_CH1.DX3E	I	0	Shift data input
USIC1_CH1.DX3F	I	0	Shift data input
USIC1_CH1.DX3G	I	USIC1_CH1.DOUT1	Loop back shift data input
USIC1_CH1.HWIN1	I	P3.14	HW controlled shift data input
Data Inputs (DX4)			
USIC1_CH1.DX4A	I	0	Shift data input
USIC1_CH1.DX4B	I	0	Shift data input
USIC1_CH1.DX4C	I	0	Shift data input
USIC1_CH1.DX4D	I	0	Shift data input
USIC1_CH1.DX4E	I	0	Shift data input
USIC1_CH1.DX4F	I	0	Shift data input
USIC1_CH1.DX4G	I	USIC1_CH1.DOUT2	Loop back shift data input
USIC1_CH1.HWIN2	I	P0.15	HW controlled shift data input
Data Inputs (DX5)			
USIC1_CH1.DX5A	I	0	Shift data input
USIC1_CH1.DX5B	I	0	Shift data input
USIC1_CH1.DX5C	I	0	Shift data input
USIC1_CH1.DX5D	I	0	Shift data input
USIC1_CH1.DX5E	I	0	Shift data input
USIC1_CH1.DX5F	I	0	Shift data input
USIC1_CH1.DX5G	I	USIC1_CH1.DOUT3	Loop back shift data input
USIC1_CH1.HWIN3	I	P0.14	HW controlled shift data input
Data Outputs			
USIC1_CH1.DOUT0	O	P0.1 P3.15 P4.2 P3.15.HW0_OUT	Shift data output
USIC1_CH1.DOUT1	O	P3.14.HW0_OUT	Shift data output
USIC1_CH1.DOUT2	O	P0.15.HW0_OUT	Shift data output
USIC1_CH1.DOUT3	O	P0.14.HW0_OUT	Shift data output

Table 17-25 USIC Module 1 Channel 1 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
Clock Outputs			
USIC1_CH1.MCLKOUT	O	P4.1	Master clock output
USIC1_CH1.SCLKOUT	O	P0.10 P0.13	Shift clock output
Control Outputs			
USIC1_CH1.SELO0	O	P0.9 P0.12	Shift control output
USIC1_CH1.SELO1	O	P0.2 P3.3	Shift control output
USIC1_CH1.SELO2	O	P3.4	Shift control output
USIC1_CH1.SELO3	O	P3.5	Shift control output
USIC1_CH1.SELO4	O	P3.6	Shift control output
USIC1_CH1.SELO5	O	not connected	Shift control output
USIC1_CH1.SELO6	O	not connected	Shift control output
USIC1_CH1.SELO7	O	not connected	Shift control output
System Related Outputs			
USIC1_CH1.DX0INS	O	USIC1_CH1.DX1F	Selected DX0 input signal
USIC1_CH1.DX1INS	O	not connected	Selected DX1 input signal
USIC1_CH1.DX2INS	O	CCU42.IN3L CCU43.IN3L	Selected DX2 input signal
USIC1_CH1.DX3INS	O	not connected	Selected DX3 input signal
USIC1_CH1.DX4INS	O	not connected	Selected DX4 input signal
USIC1_CH1.DX5INS	O	not connected	Selected DX5 input signal

17.12.3 USIC Module 2 Interconnects

The interconnects of USIC module 2 is grouped into the following three categories:

- **USIC Module 2 Channel 0 Interconnects** (**Table 17-26**)
- **USIC Module 2 Channel 1 Interconnects** (**Table 17-27**)

Table 17-26 USIC Module 2 Channel 0 Interconnects

Input/Output	I/O	Connected To	Description
Data Inputs (DX0)			
USIC2_CH0.DX0A	I	P5.1	Shift data input
USIC2_CH0.DX0B	I	P5.0	Shift data input
USIC2_CH0.DX0C	I	P3.7	Shift data input
USIC2_CH0.DX0D	I	0	Shift data input
USIC2_CH0.DX0E	I	0	Shift data input
USIC2_CH0.DX0F	I	XTAL1	Shift data input
USIC2_CH0.DX0G	I	USIC2_CH0.DOUT0	Loop back shift data input
USIC2_CH0.HWIN0	I	P5.0	HW controlled shift data input
Clock Inputs			
USIC2_CH0.DX1A	I	P5.2	Shift clock input
USIC2_CH0.DX1B	I	0	Shift clock input
USIC2_CH0.DX1C	I	0	Shift clock input
USIC2_CH0.DX1D	I	0	Shift clock input
USIC2_CH0.DX1E	I	0	Shift clock input
USIC2_CH0.DX1F	I	USIC2_CH0.DX0INS	Shift clock input
USIC2_CH0.DX1G	I	USIC2_CH0.SCLKOUT	Loop back shift clock input
Control Inputs			
USIC2_CH0.DX2A	I	P5.3	Shift control input
USIC2_CH0.DX2B	I	0	Shift control input
USIC2_CH0.DX2C	I	0	Shift control input
USIC2_CH0.DX2D	I	0	Shift control input
USIC2_CH0.DX2E	I	CCU41.SR1	Shift control input
USIC2_CH0.DX2F	I	CCU81.SR1	Shift control input
USIC2_CH0.DX2G	I	USIC2_CH0.SELO0	Loop back shift control input
Data Inputs (DX3)			
USIC2_CH0.DX3A	I	0	Shift data input
USIC2_CH0.DX3B	I	0	Shift data input
USIC2_CH0.DX3C	I	0	Shift data input

Table 17-26 USIC Module 2 Channel 0 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC2_CH0.DX3D	I	0	Shift data input
USIC2_CH0.DX3E	I	0	Shift data input
USIC2_CH0.DX3F	I	0	Shift data input
USIC2_CH0.DX3G	I	USIC2_CH0.DOUT1	Loop back shift data input
USIC2_CH0.HWIN1	I	P5.1	HW controlled shift data input
Data Inputs (DX4)			
USIC2_CH0.DX4A	I	0	Shift data input
USIC2_CH0.DX4B	I	0	Shift data input
USIC2_CH0.DX4C	I	0	Shift data input
USIC2_CH0.DX4D	I	0	Shift data input
USIC2_CH0.DX4E	I	0	Shift data input
USIC2_CH0.DX4F	I	0	Shift data input
USIC2_CH0.DX4G	I	USIC2_CH0.DOUT2	Loop back shift data input
USIC2_CH0.HWIN2	I	P5.7	HW controlled shift data input
Data Inputs (DX5)			
USIC2_CH0.DX5A	I	0	Shift data input
USIC2_CH0.DX5B	I	0	Shift data input
USIC2_CH0.DX5C	I	0	Shift data input
USIC2_CH0.DX5D	I	0	Shift data input
USIC2_CH0.DX5E	I	0	Shift data input
USIC2_CH0.DX5F	I	0	Shift data input
USIC2_CH0.DX5G	I	USIC2_CH0.DOUT3	Loop back shift data input
USIC2_CH0.HWIN3	I	P2.6	HW controlled shift data input
Data Outputs			
USIC2_CH0.DOUT0	O	P3.8 P5.0 P5.0.HW0_OUT	Shift data output
USIC2_CH0.DOUT1	O	P5.1.HW0_OUT	Shift data output
USIC2_CH0.DOUT2	O	P5.7.HW0_OUT	Shift data output
USIC2_CH0.DOUT3	O	P2.6.HW0_OUT	Shift data output
Clock Outputs			

Table 17-26 USIC Module 2 Channel 0 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC2_CH0.MCLKOUT	O	not connected	Master clock output
USIC2_CH0.SCLKOUT	O	P3.9 P5.2	Shift clock output
Control Outputs			
USIC2_CH0.SELO0	O	P3.10 P5.3	Shift control output
USIC2_CH0.SELO1	O	P5.4	Shift control output
USIC2_CH0.SELO2	O	P5.5	Shift control output
USIC2_CH0.SELO3	O	P5.6	Shift control output
USIC2_CH0.SELO4	O	P2.6	Shift control output
USIC2_CH0.SELO5	O	not connected	Shift control output
USIC2_CH0.SELO6	O	not connected	Shift control output
USIC2_CH0.SELO7	O	not connected	Shift control output
System Related Outputs			
USIC2_CH0.DX0INS	O	USIC2_CH0.DX1F	Selected DX0 input signal
USIC2_CH0.DX1INS	O	not connected	Selected DX1 input signal
USIC2_CH0.DX2INS	O	not connected	Selected DX2 input signal
USIC2_CH0.DX3INS	O	not connected	Selected DX3 input signal
USIC2_CH0.DX4INS	O	not connected	Selected DX4 input signal
USIC2_CH0.DX5INS	O	not connected	Selected DX5 input signal

Table 17-27 USIC Module 2 Channel 1 Interconnects

Input/Output	I/O	Connected To	Description
Data Inputs (DX0)			
USIC2_CH1.DX0A	I	P3.5	Shift data input
USIC2_CH1.DX0B	I	P3.4	Shift data input
USIC2_CH1.DX0C	I	P4.0	Shift data input
USIC2_CH1.DX0D	I	P3.12	Shift data input
USIC2_CH1.DX0E	I	CAN1INS	Shift data input
USIC2_CH1.DX0F	I	XTAL1	Shift data input

Table 17-27 USIC Module 2 Channel 1 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC2_CH1.DX0G	I	USIC2_CH1.DOUT0	Loop back shift data input
USIC2_CH1.HWIN0	I	P4.7	HW controlled shift data input
Clock Inputs			
USIC2_CH1.DX1A	I	P4.2	Shift clock input
USIC2_CH1.DX1B	I	P3.6	Shift clock input
USIC2_CH1.DX1C	I	0	Shift clock input
USIC2_CH1.DX1D	I	0	Shift clock input
USIC2_CH1.DX1E	I	0	Shift clock input
USIC2_CH1.DX1F	I	USIC2_CH1.DX0INS	Shift clock input
USIC2_CH1.DX1G	I	USIC2_CH1.SCLKOUT	Loop back shift clock input
Control Inputs			
USIC2_CH1.DX2A	I	P4.1	Shift control input
USIC2_CH1.DX2B	I	P4.1	Shift control input
USIC2_CH1.DX2C	I	0	Shift control input
USIC2_CH1.DX2D	I	0	Shift control input
USIC2_CH1.DX2E	I	CCU43.SR1	Shift control input
USIC2_CH1.DX2F	I	CCU81.SR1	Shift control input
USIC2_CH1.DX2G	I	USIC2_CH1.SELO0	Loop back shift control input
Data Inputs (DX3)			
USIC2_CH1.DX3A	I	0	Shift data input
USIC2_CH1.DX3B	I	0	Shift data input
USIC2_CH1.DX3C	I	0	Shift data input
USIC2_CH1.DX3D	I	0	Shift data input
USIC2_CH1.DX3E	I	0	Shift data input
USIC2_CH1.DX3F	I	0	Shift data input
USIC2_CH1.DX3G	I	USIC2_CH1.DOUT1	Loop back shift data input
USIC2_CH1.HWIN1	I	P4.6	HW controlled shift data input
Data Inputs (DX4)			
USIC2_CH1.DX4A	I	0	Shift data input
USIC2_CH1.DX4B	I	0	Shift data input

Table 17-27 USIC Module 2 Channel 1 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC2_CH1.DX4C	I	0	Shift data input
USIC2_CH1.DX4D	I	0	Shift data input
USIC2_CH1.DX4E	I	0	Shift data input
USIC2_CH1.DX4F	I	0	Shift data input
USIC2_CH1.DX4G	I	USIC2_CH1.DOUT2	Loop back shift data input
USIC2_CH1.HWIN2	I	P4.5	HW controlled shift data input
Data Inputs (DX5)			
USIC2_CH1.DX5A	I	0	Shift data input
USIC2_CH1.DX5B	I	0	Shift data input
USIC2_CH1.DX5C	I	0	Shift data input
USIC2_CH1.DX5D	I	0	Shift data input
USIC2_CH1.DX5E	I	0	Shift data input
USIC2_CH1.DX5F	I	0	Shift data input
USIC2_CH1.DX5G	I	USIC2_CH1.DOUT3	Loop back shift data input
USIC2_CH1.HWIN3	I	P4.4	HW controlled shift data input
Data Outputs			
USIC2_CH1.DOUT0	O	P3.5 P3.11 P4.7.HW0_OUT	Shift data output
USIC2_CH1.DOUT1	O	P4.6.HW0_OUT	Shift data output
USIC2_CH1.DOUT2	O	P4.5.HW0_OUT	Shift data output
USIC2_CH1.DOUT3	O	P4.4.HW0_OUT	Shift data output
Clock Outputs			
USIC2_CH1.MCLKOUT	O	P3.4	Master clock output
USIC2_CH1.SCLKOUT	O	P3.6 P3.13 P4.2	Shift clock output
Control Outputs			
USIC2_CH1.SELO0	O	P3.0 P4.1	Shift control output
USIC2_CH1.SELO1	O	P4.2	Shift control output

Table 17-27 USIC Module 2 Channel 1 Interconnects (cont'd)

Input/Output	I/O	Connected To	Description
USIC2_CH1.SELO2	O	P4.3	Shift control output
USIC2_CH1.SELO3	O	not connected	Shift control output
USIC2_CH1.SELO4	O	not connected	Shift control output
USIC2_CH1.SELO5	O	not connected	Shift control output
USIC2_CH1.SELO6	O	not connected	Shift control output
USIC2_CH1.SELO7	O	not connected	Shift control output
System Related Outputs			
USIC2_CH1.DX0INS	O	USIC2_CH1.DX1F	Selected DX0 input signal
USIC2_CH1.DX1INS	O	not connected	Selected DX1 input signal
USIC2_CH1.DX2INS	O	not connected	Selected DX2 input signal
USIC2_CH1.DX3INS	O	not connected	Selected DX3 input signal
USIC2_CH1.DX4INS	O	not connected	Selected DX4 input signal
USIC2_CH1.DX5INS	O	not connected	Selected DX5 input signal

18 Controller Area Network Controller (MultiCAN)

This chapter describes the MultiCAN controller of the XMC4500. It contains the following sections:

- CAN basics (see **Page 18-5**)
- Overview of the CAN Module in the XMC4500 (see **Page 18-4**)
- Functional description of the MultiCAN Kernel (see **Page 18-14**)
- MultiCAN Kernel register description (see **Page 18-59**)
- XMC4500 implementation-specific details are listed below,
  - Service Request Generation (see **Page 18-53**)
  - Debug Behavior (see **Page 18-55**)
  - Power, Reset and Clock (see **Page 18-56**)
  - Interconnects (see **Page 18-120**)

Note: The MultiCAN register names described in this chapter are referenced in the XMC4500 Reference Manual by the module name prefix “CAN_”.

18.1 Overview

The MultiCAN module contains independently operating CAN nodes with Full-CAN functionality that are able to exchange Data and Remote Frames via a gateway function. Transmission and reception of CAN frames is handled in accordance with CAN specification V2.0 B (active). Each CAN node can receive and transmit standard frames with 11-bit identifiers as well as extended frames with 29-bit identifiers.

All CAN nodes share a common set of message objects. Each message object can be individually allocated to one of the CAN nodes. Besides serving as a storage container for incoming and outgoing frames, message objects can be combined to build gateways between the CAN nodes or to setup a FIFO buffer.

The message objects are organized in double-chained linked lists, where each CAN node has its own list of message objects. A CAN node stores frames only into message objects that are allocated to the message object list of the CAN node, and it transmits only messages belonging to this message object list. A powerful, command-driven list controller performs all message object list operations.

The bit timings for the CAN nodes are derived from the module timer clock (f_{CAN}), and are programmable up to a data rate of 1 Mbit/s. External bus transceivers are connected to a CAN node via a pair of receive and transmit pins.

18.1.1 Features

The MultiCAN module provides the following functionality:

- 3 independent CAN nodes and 64 message objects available.
- Compliant with ISO 11898
- CAN functionality according to CAN specification V2.0 B active
- Dedicated control registers for each CAN node
- Data transfer rates up to 1 Mbit/s
- Flexible and powerful message transfer control and error handling capabilities
- Advanced CAN bus bit timing analysis and baud rate detection for each CAN node via a frame counter
- Full-CAN functionality: A set of 64 message objects can be individually
  - Allocated (assigned) to any CAN node
  - Configured as transmit or receive object
  - Set up to handle frames with 11-bit or 29-bit identifier
  - Identified by a timestamp via a frame counter
  - Configured to remote monitoring mode
- Advanced acceptance filtering
  - Each message object provides an individual acceptance mask to filter incoming frames
  - A message object can be configured to accept standard or extended frames or to accept both standard and extended frames

  - Message objects can be grouped into four priority classes for transmission and reception
  - The selection of the message to be transmitted first can be based on frame identifier, IDE bit and RTR bit according to CAN arbitration rules, or according to its order in the list
- Advanced message object functionality
  - Message objects can be combined to build FIFO message buffers of arbitrary size, limited only by the total number of message objects
  - Message objects can be linked to form a gateway that automatically transfers frames between two different CAN buses. A single gateway can link any two CAN nodes. An arbitrary number of gateways can be defined.
- Advanced data management
  - The message objects are organized in double-chained lists
  - List reorganizations can be performed at any time, even during full operation of the CAN nodes
  - A powerful, command-driven list controller manages the organization of the list structure and ensures consistency of the list
  - Message FIFOs are based on the list structure and can easily be scaled in size during CAN operation
  - Static allocation commands offer compatibility with TwinCAN applications that are not list-based
- Advanced interrupt handling
  - Up to 8 interrupt output lines are available. Interrupt requests can be routed individually to one of the 8 interrupt output lines
  - Message post-processing notifications can be mapped flexibly using dedicated registers consisting of notification bits

18.1.2 Block Diagram

This section describes the serial communication module called MultiCAN (CAN = Controller Area Network) of the XMC4500. A MultiCAN module can contain between two and eight independent CAN nodes, depending on the device, each representing one serial communication interface.

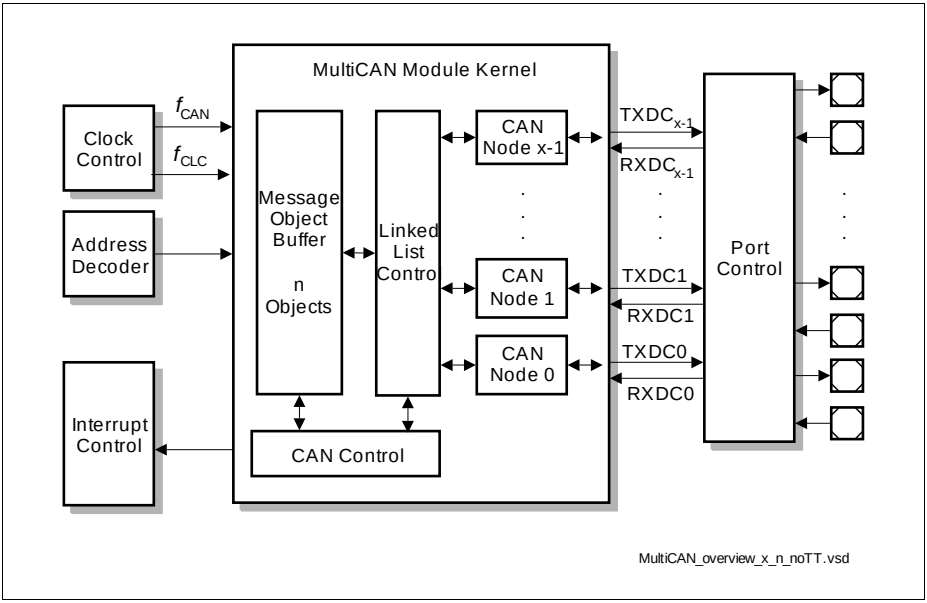


Figure 18-1 Overview of the MultiCAN Module

18.2 CAN Basics

CAN is an asynchronous serial bus system with one logical bus line. It has an open, linear bus structure with equal bus participants called nodes. A CAN bus consists of two or more nodes.

The bus logic corresponds to a “wired-AND” mechanism. Recessive bits (equivalent to the logic 1 level) are overwritten by dominant bits (logic 0 level). As long as no bus node is sending a dominant bit, the bus is in the recessive state. In this state, a dominant bit from any bus node generates a dominant bus state. The maximum CAN bus speed is, by definition, 1 Mbit/s. This speed limits the CAN bus to a length of up to 40 m. For bus lengths longer than 40 m, the bus speed must be reduced.

The binary data of a CAN frame is coded in NRZ code (Non-Return-to-Zero). To ensure re-synchronization of all bus nodes, bit stuffing is used. This means that during the transmission of a message, a maximum of five consecutive bits can have the same polarity. Whenever five consecutive bits of the same polarity have been transmitted, the transmitter will insert one additional bit (stuff bit) of the opposite polarity into the bit stream before transmitting further bits. The receiver also checks the number of bits with the same polarity and removes the stuff bits from the bit stream (= destuffing).

18.2.1 Addressing and Bus Arbitration

In the CAN protocol, address information is defined in the identifier field of a message. The identifier indicates the contents of the message and its priority. The lower the binary value of the identifier, the higher is the priority of the message.

For bus arbitration, CSMA/CD with NDA (Carrier Sense Multiple Access/Collision Detection with Non-Destructive Arbitration) is used. If bus node A attempts to transmit a message across the network, it first checks that the bus is in the idle state (“Carrier Sense”) i.e. no node is currently transmitting. If this is the case (and no other node wishes to start a transmission at the same moment), node A becomes the bus master and sends its message. All other nodes switch to receive mode during the first transmitted bit (Start-Of-Frame bit). After correct reception of the message (acknowledged by each node), each bus node checks the message identifier and stores the message, if required. Otherwise, the message is discarded.

If two or more bus nodes start their transmission at the same time (“Multiple Access”), bus collision of the messages is avoided by bit-wise arbitration (“Collision Detection / Non-Destructive Arbitration” together with the “Wired-AND” mechanism, dominant bits override recessive bits). Each node that sends also reads back the bus level. When a recessive bit is sent but a dominant one is read back, bus arbitration is lost and the transmitting node switches to receive mode. This condition occurs for example when the message identifier of a competing node has a lower binary value and therefore sends a message with a higher priority. In this way, the bus node with the highest priority message wins arbitration without losing time by having to repeat the message. Other nodes that lost arbitration will automatically try to repeat their transmission once the bus

returns to idle state. Therefore, the same identifier can be sent in a Data Frame only by one node in the system. There must not be more than one node programmed to send Data Frames with the same identifier.

Standard message identifier has a length of 11 bits. CAN specification 2.0B extends the message identifier lengths to 29 bits, i.e. the extended identifier.

18.2.2 CAN Frame Formats

There are three types of CAN frames:

- Data Frames
- Remote Frames
- Error Frames

A Data Frame contains a Data Field of 0 to 8 bytes in length. A Remote Frame contains no Data Field and is typically generated as a request for data (e.g. from a sensor). Data and Remote Frames can use an 11-bit “Standard” identifier or a 29-bit “Extended” identifier. An Error Frame can be generated by any node that detects a CAN bus error.

18.2.2.1 Data Frames

There are two types of Data Frames defined (see **Figure 18-2**):

- Standard Data Frame
- Extended Data Frame

Standard Data Frame

A Data Frame begins with the Start-Of-Frame bit (SOF = dominant level) for hard synchronization of all nodes. The SOF is followed by the Arbitration Field consisting of 12 bits, the 11-bit identifier (reflecting the contents and priority of the message), and the RTR (Remote Transmission Request) bit. With RTR at dominant level, the frame is marked as Data Frame. With RTR at recessive level, the frame is defined as a Remote Frame.

The next field is the Control Field consisting of 6 bits. The first bit of this field is the IDE (Identifier Extension) bit and is at dominant level for the Standard Data Frame. The following bit is reserved and defined as a dominant bit. The remaining 4 bits of the Control Field are the Data Length Code (DLC) that specifies the number of bytes in the Data Field. The Data Field can be 0 to 8 bytes wide. The Cyclic Redundancy (CRC) Field that follows the data bytes is used to detect possible transmission errors. It consists of a 15-bit CRC sequence, completed by a recessive CRC delimiter bit.

The final field is the Acknowledge Field. During the ACK Slot, the transmitting node sends out a recessive bit. Any node that has received an error free frame acknowledges the correct reception of the frame by sending back a dominant bit, regardless of whether or not the node is configured to accept that specific message. This behavior assigns the

CAN protocol to the “in-bit-response” group of protocols. The recessive ACK delimiter bit, which must not be overwritten by a dominant bit, completes the Acknowledge Field.

Seven recessive End-of-Frame (EOF) bits finish the Data Frame. Between any two consecutive frames, the bus must remain in the recessive state for at least 3 bit times (called Inter Frame Space). If after the Inter Frame Space, no other nodes attempt to transmit the bus remains in idle state with a recessive level.

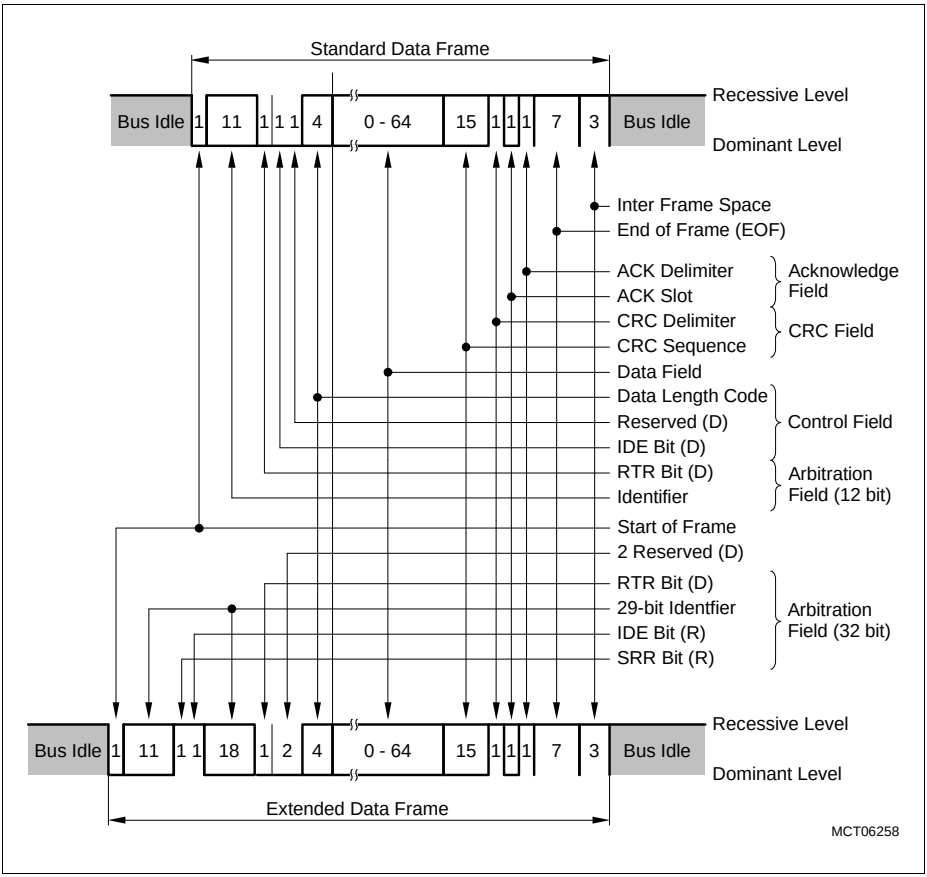


Figure 18-2 CAN Data Frame

Extended Data Frame

In the Extended CAN Data Frame, the message identifier of the standard frame has been extended to 29-bit. A split of the extended identifier into two parts, an 11-bit least

significant section (as in standard CAN frame) and an 18-bit most significant section, ensures that the Identifier Extension bit (IDE) can remain at the same bit position in both standard and extended frames.

In the Extended CAN Data Frame, the SOF bit is followed by the 32-bit Arbitration Field. The first 11 bits are the least significant bits of the 29-bit Identifier (“Base-ID”). These 11 bits are followed by the recessive Substitute Remote Request (SRR) bit. The SRR is further followed by the recessive IDE bit, which indicates the frame to be an Extended CAN frame. If arbitration remains unresolved after transmission of the first 11 bits of the identifier, and if one of the nodes involved in arbitration is sending a Standard CAN frame, then the Standard CAN frame will win arbitration due to the assertion of its dominant IDE bit. Therefore, the SRR bit in an Extended CAN frame is recessive to allow the assertion of a dominant RTR bit by a node that is sending a Standard CAN Remote Frame. The SRR and IDE bits are followed by the remaining 18 bits of the extended identifier and the RTR bit.

Control field and frame termination is identical to the Standard Data Frame.

18.2.2.2 Remote Frames

Normally, data transmission is performed on an autonomous basis with the data source node (e.g. a sensor) sending out a Data Frame. It is also possible, however, for a destination node (or nodes) to request the data from the source. For this purpose, the destination node sends a Remote Frame with an identifier that matches the identifier of the required Data Frame. The appropriate data source node will then send a Data Frame as a response to this remote request.

There are 2 differences between a Remote Frame and a Data Frame.

- The RTR bit is in the recessive state in a Remote Frame.
- There is no Data Field in a Remote Frame.

If a Data Frame and a Remote Frame with the same identifier are transmitted at the same time, the Data Frame wins arbitration due to the dominant RTR bit following the identifier. In this way, the node that transmitted the Remote Frame receives the requested data immediately. The format of a Standard and Extended Remote Frames is shown in **Figure 18-3**.

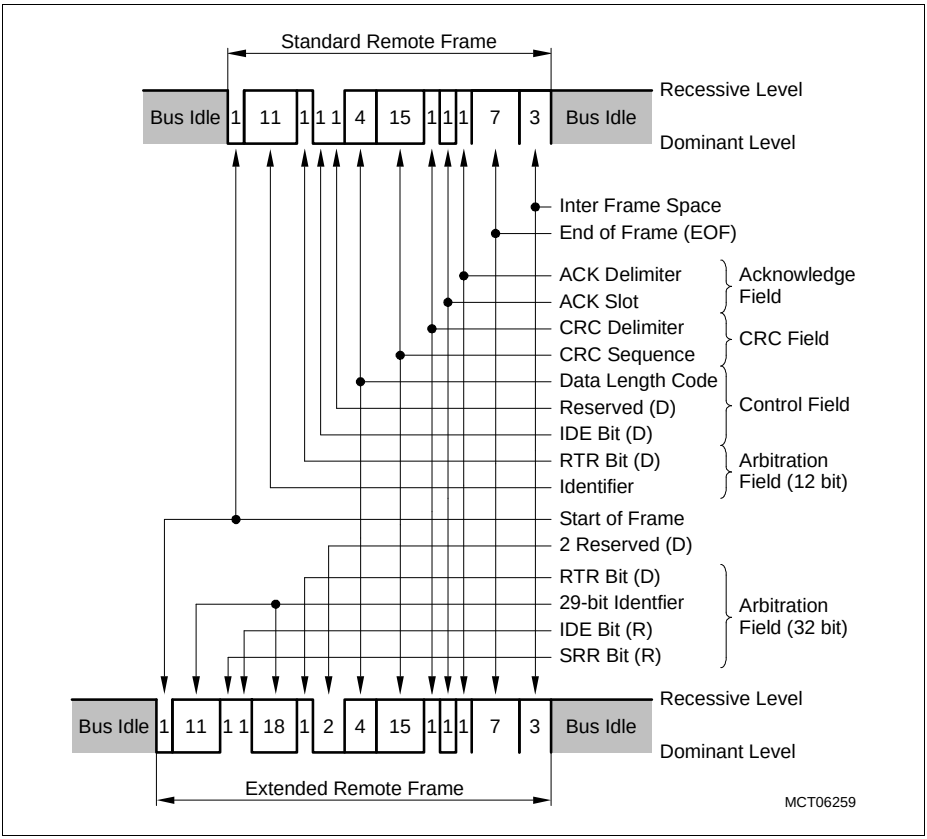


Figure 18-3 CAN Remote Frame

18.2.2.3 Error Frames

An Error Frame is generated by any node that detects a bus error. An Error Frame consists of two fields, an Error Flag field followed by an Error Delimiter field. The Error Delimiter Field consists of 8 recessive bits and allows the bus nodes to restart bus communications after an error. There are, however, two forms of Error Flag fields. The form of the Error Flag field depends on the error status of the node that detects the error.

When an error-active node detects a bus error, the node generates an Error Frame with an active-error flag. The error-active flag is composed of six consecutive dominant bits that actively violate the bit-stuffing rule. All other stations recognize a bit-stuffing error and generate Error Frames themselves. The resulting Error Flag field on the CAN bus therefore consists of six to twelve consecutive dominant bits (generated by one or more nodes). The Error Delimiter field completes the Error Frame. After completion of the Error Frame, bus activity returns to normal and the interrupted node attempts to re-send the aborted message.

If an error-passive node detects a bus error, the node transmits an error-passive flag followed, again, by the Error Delimiter field. The error-passive flag consists of six consecutive recessive bits, and therefore the Error Frame (for an error-passive node) consists of 14 recessive bits (i.e. no dominant bits). Therefore, the transmission of an Error Frame by an error-passive node will not affect any other node on the network, unless the bus error is detected by the node that is actually transmitting (i.e. the bus master). If the bus master node generates an error-passive flag, this may cause other nodes to generate Error Frames due to the resulting bit-stuffing violation. After transmission of an Error Frame an error-passive node must wait for 6 consecutive recessive bits on the bus before attempting to rejoin bus communications.

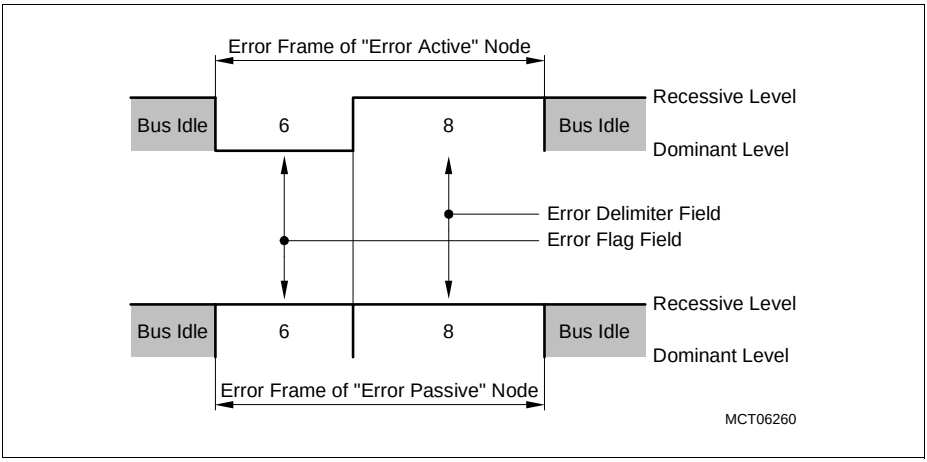


Figure 18-4 CAN Error Frames

18.2.3 The Nominal Bit Time

One bit cell (this means one high or low pulse of the NRZ code) is composed by four segments. Each segment is an integer multiple of Time Quanta t_Q. The Time Quanta is the smallest discrete timing resolution used by a CAN node. The nominal bit time definition with its segments is shown in **Figure 18-5**.

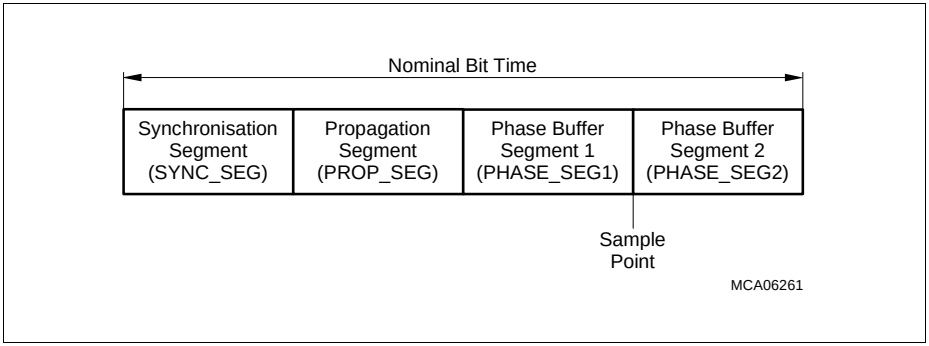


Figure 18-5 Partition of Nominal Bit Time

The Synchronization Segment (SYNC_SEG) is used to synchronize the various bus nodes. If there is a bit state change between the previous bit and the current bit, then the bus state change is expected to occur within this segment. The length of this segment is always 1 t_Q.

The Propagation Segment (PROP_SEG) is used to compensate for signal delays across the network. These delays are caused by signal propagation delay on the bus line and through the electronic interface circuits of the bus nodes.

The Phase Segments 1 and 2 (PHASE_SEG1, PHASE_SEG2) are used to compensate for edge phase errors. These segments can be lengthened or shortened by re-synchronization. PHASE_SEG2 is reserved for calculation of the subsequent bit level, and is ≥ 2 t_Q. At the sample point, the bus level is read and interpreted as the value of the bit cell. It occurs at the end of PHASE_SEG1.

The total number of t_Q in a bit time is between 8 and 25.

As a result of re-synchronization, PHASE_SEG1 can be lengthened or PHASE_SEG2 can be shortened. The amount of lengthening or shortening the phase buffer segments has an upper limit given by the re-synchronization jump width. The re-synchronization jump width may be between 1 and 4 t_Q, but it may not be longer than PHASE_SEG1.

18.2.4 Error Detection and Error Handling

The CAN protocol has sophisticated error detection mechanisms. The following errors can be detected:

- **Cyclic Redundancy Check (CRC) Error**
  With the Cyclic Redundancy Check, the transmitter calculates special check bits for the bit sequence from the start of a frame until the end of the Data Field. This CRC sequence is transmitted in the CRC Field. The receiving node also calculates the CRC sequence using the same formula, and performs a comparison to the received sequence. If a mismatch is detected, a CRC error has occurred and an Error Frame is generated. The message is repeated.
- **Acknowledge Error**
  In the Acknowledge Field of a message, the transmitter checks whether a dominant bit is read during the Acknowledge Slot (that is sent out as a recessive bit). If not, no other node has received the frame correctly, an Acknowledge Error has occurred, and the message must be repeated. No Error Frame is generated.
- **Form Error**
  If a transmitter detects a dominant bit in one of the four segments End of Frame, Interframe Space, Acknowledge Delimiter, or CRC Delimiter, a Form Error has occurred, and an Error Frame is generated. The message is repeated.
- **Bit Error**
  A Bit Error occurs if a) a transmitter sends a dominant bit and detects a recessive bit or b) if the transmitter sends a recessive bit and detects a dominant bit when monitoring the actual bus level and comparing it to the just transmitted bit. In case b), no error occurs during the Arbitration Field (ID, RTR, IDE) and the Acknowledge Slot.
- **Stuff Error**
  If between Start of Frame and CRC Delimiter, six consecutive bits with the same polarity are detected, the bit-stuffing rule has been violated. A stuff error occurs and an Error Frame is generated. The message is repeated.

Detected errors are made public to all other nodes via Error Frames (except Acknowledge Errors). The transmission of the erroneous message is aborted and the frame is repeated as soon as possible. Furthermore, each CAN node is in one of the three error states (error-active, error-passive or bus-off) according to the value of the internal error counters. The error-active state is the usual state where the bus node can transmit messages and active-error frames (made of dominant bits) without any restrictions. In the error-passive state, messages and passive-error frames (made of recessive bits) may be transmitted. The bus-off state makes it temporarily impossible for the node to participate in the bus communication. During this state, messages can be neither received nor transmitted.

Basic CAN, Full CAN

There is one more CAN characteristic that is related to the interface of a CAN module (controller) and the host CPU: Basic-CAN and Full-CAN functionality.

In Basic-CAN devices, only basic functions of the protocol are implemented in hardware, such as the generation and the check of the bit stream. The decision, whether a received message has to be stored or not (acceptance filtering), and the complete message management must be done by software. Normally, the CAN device also provides only one transmit buffer and one or two receive buffers. Therefore, the host CPU load is quite high when using Basic-CAN modules. The main advantage of Basic-CAN is a reduced chip size leading to low costs of these devices.

Full-CAN devices (this is the case for the MultiCAN controller as implemented in XMC4500) manage the whole bus protocol in hardware, including the acceptance filtering and message management. Full-CAN devices contain message objects that handle autonomously the identifier, the data, the direction (receive or transmit) and the information of Standard CAN/Extended CAN operation. During the initialization of the device, the host CPU determines which messages are to be sent and which are to be received. The host CPU is informed by interrupt if the identifier of a received message matches with one of the programmed (receive-) message objects. The CPU load of Full-CAN devices is greatly reduced. When using Full-CAN devices, high baud rates and high bus loads with many messages can be handled.

18.3 MultiCAN Kernel Functional Description

This section describes the functionality of the MultiCAN module.

18.3.1 Module Structure

Figure 18-6 shows the general structure of the MultiCAN module.

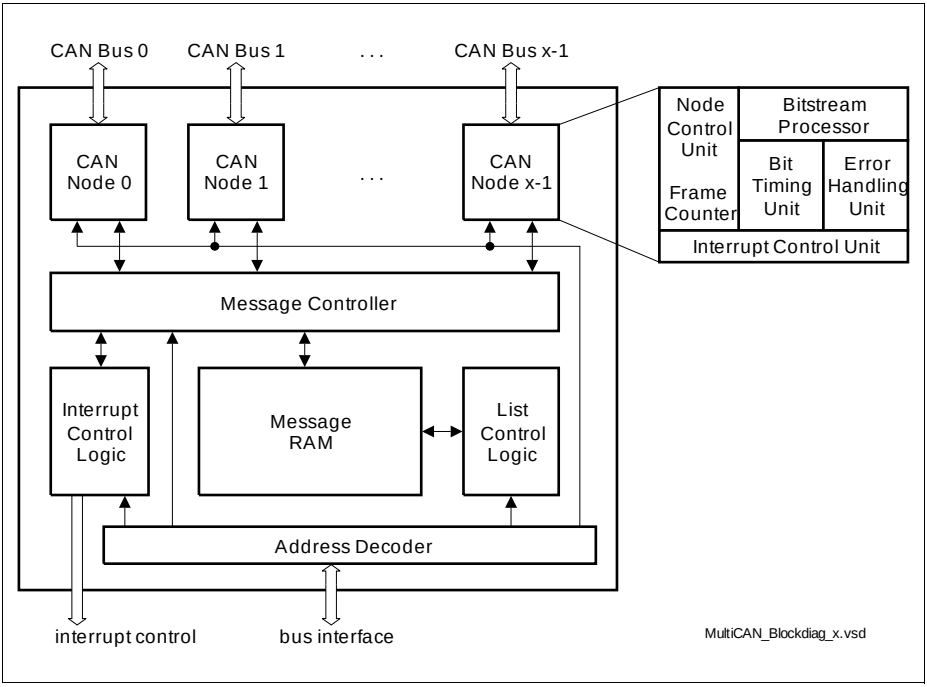


Figure 18-6 MultiCAN Block Diagram

CAN Nodes

Each CAN node consists of several sub-units.

- **Bitstream Processor**
  The Bitstream Processor performs data, remote, error and overload frame processing according to the ISO 11898 standard. This includes conversion between the serial data stream and the input/output registers.
- **Bit Timing Unit**
  The Bit Timing Unit determines the length of a bit time and the location of the sample point according to the user settings, taking into account propagation delays and phase shift errors. The Bit Timing Unit also performs resynchronization.

- **Error Handling Unit**
  The Error Handling Unit manages the receive and transmit error counter. Depending on the contents of both counters, the CAN node is set into an error-active, error passive or bus-off state.
- **Node Control Unit**
  The Node Control Unit coordinates the operation of the CAN node:
  – Enable/disable CAN transfer of the node
  – Enable/disable and generate node-specific events that lead to an interrupt request (CAN bus errors, successful frame transfers etc.)
  – Administration of the Frame Counter
- **Interrupt Control Unit**
  The Interrupt Control Unit in the CAN node controls the interrupt generation for the different conditions that can occur in the CAN node.

Message Controller

The Message Controller handles the exchange of CAN frames between the CAN nodes and the message objects that are stored in the Message RAM. The Message Controller performs several functions:

- Receive acceptance filtering to determine the correct message object for storing of a received CAN frame
- Transmit acceptance filtering to determine the message object to be transmitted first, individually for each CAN node
- Transfer contents between message objects and the CAN nodes, taking into account the status/control bits of the message objects
- Handling of the FIFO buffering and gateway functionality
- Aggregation of message-pending notification bits

List Controller

The List Controller performs all operations that lead to a modification of the double-chained message object lists. Only the list controller is allowed to modify the list structure. The allocation/deallocation or reallocation of a message object can be requested via a user command interface (command panel). The list controller state machine then performs the requested command autonomously.

Interrupt Control

The general interrupt structure is shown in **Figure 18-7**. The interrupt event can trigger the interrupt generation. The interrupt pulse is generated independently of the interrupt flag in the interrupt status register. The interrupt flag can be reset by software by writing a 0 to it.

If enabled by the related interrupt enable bit in the interrupt enable register, an interrupt pulse can be generated at one of the 16 interrupt output lines INT_Om of the MultiCAN

module. If more than one interrupt source is connected to the same interrupt node pointer (in the interrupt node pointer register), the requests are combined to one common line.

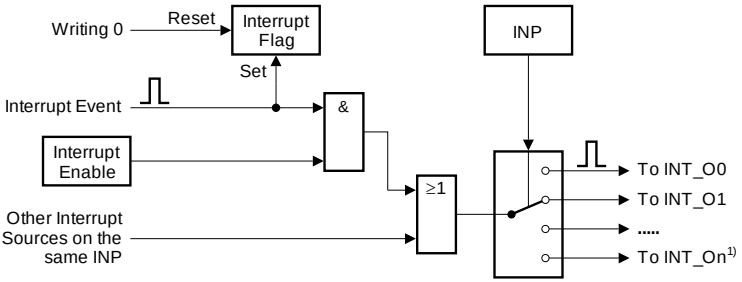


Figure 18-7 General Interrupt Structure

18.3.2 Port Input Control

It is possible to select the input lines for the RXDCx inputs for the CAN nodes. The selected input is connected to the CAN node and is also available to wake-up the system. More details are defined in **Section 18.8.2.1** on **Page 18-121**.

18.3.3 CAN Node Control

Each CAN node may be configured and run independently of the other CAN node. Each CAN node is equipped with its own node control logic to configure the global behavior and to provide status information.

Note: In the following descriptions, index “x” stands for the node number and index “n” represents the message object number.

Configuration Mode is activated when bit NCRx.CCE is set to 1. This mode allows CAN bit timing parameters and the error counter registers to be modified.

CAN Analyzer Mode is activated when bit NCRx.CALM is set to 1. In this operation mode, Data And Remote Frames are monitored without active participation in any CAN transfer (CAN transmit pin is held on recessive level). Incoming Remote Frames are stored in a corresponding transmit message object, while arriving data frames are saved in a matching receive message object.

In CAN Analyzer Mode, the entire configuration information of the received frame is stored in the corresponding message object, and can be evaluated by the CPU to determine their identifier, XTD bit information and data length code (ID and DLC optionally if the Remote Monitoring Mode is active, bit MOFCRn.RMM = 1). Incoming frames are not acknowledged, and no Error Frames are generated. If CAN Analyzer Mode is enabled, Remote Frames are not responded to by the corresponding Data Frame, and Data Frames cannot be transmitted by setting the transmit request bit MOSTATn.TXRQ. Receive interrupts are generated in CAN Analyzer Mode (if enabled) for all error free received frames.

The node-specific interrupt configuration is also defined by the Node Control Logic via the NCRx register bits TRIE, ALIE and LECIE:

- If control bit TRIE is set to 1, a transfer interrupt is generated when the NSRx register has been updated (after each successfully completed message transfer).
- If control bit ALIE is set to 1, an error interrupt is generated when a “bus-off” condition has been recognized or the Error Warning Level has been exceeded or under-run. Additionally, list or object errors lead to this type of interrupt.
- If control bit LECIE is set to 1, a last error code interrupt is generated when an error code > 0 is written into bit field NSRx.LEC by hardware.

The Node x Status Register NSRx provides an overview about the current state of the respective CAN node x, comprising information about CAN transfers, CAN node status, and error conditions.

The CAN frame counter can be used to check the transfer sequence of message objects or to obtain information about the instant a frame has been transmitted or received from the associated CAN bus. CAN frame counting is performed by a 16-bit counter, controlled by register NFCRx. Bit fields NFCRx.CFMOD and NFCRx.CFSEL determine the operation mode and the trigger event incrementing the frame counter.

18.3.3.1 Bit Timing Unit

According to the ISO 11898 standard, a CAN bit time is subdivided into different segments (**Figure 18-8**). Each segment consists of multiples of a time quantum t_q. The magnitude of t_q is adjusted by Node x Bit Timing Register bit fields NBTRx.BRP and NBTRx.DIV8, both controlling the baud rate prescaler (register NBTRx is described on **Page 18-85**). The baud rate prescaler is driven by the module timer clock f_{CAN} (generation and control of f_{CAN} is described on **Page 18-58**).

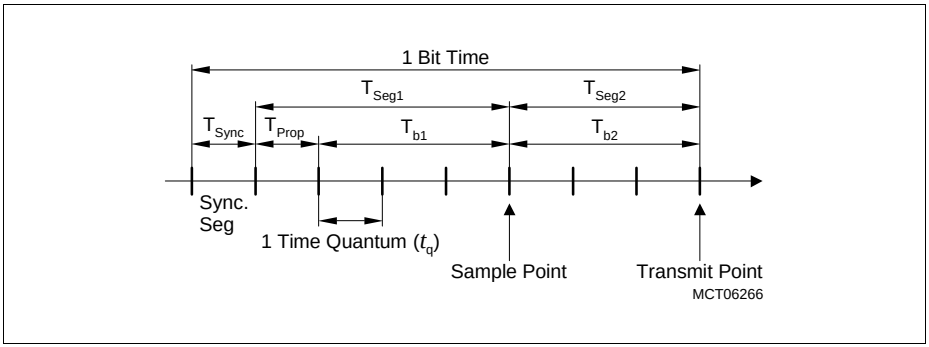


Figure 18-8 CAN Bus Bit Timing Standard

The Synchronization Segment (T_{Sync}) allows a phase synchronization between transmitter and receiver time base. The Synchronization Segment length is always one t_q. The Propagation Time Segment (T_{Prop}) takes into account the physical propagation delay in the transmitter output driver on the CAN bus line and in the transceiver circuit. For a working collision detection mechanism, T_{Prop} must be two times the sum of all propagation delay quantities rounded up to a multiple of t_q. The phase buffer segments 1 and 2 (T_{b1}, T_{b2}) before and after the signal sample point are used to compensate for a mismatch between transmitter and receiver clock phases detected in the synchronization segment.

The maximum number of time quanta allowed for re-synchronization is defined by bit field NBTRx.SJW. The Propagation Time Segment and the Phase Buffer Segment 1 are combined to parameter T_{Seg1}, which is defined by the value NBTRx.TSEG1. A minimum of 3 time quanta is demanded by the ISO standard. Parameter T_{Seg2}, which is defined by the value of NBTRx.TSEG2, covers the Phase Buffer Segment 2. A minimum of 2 time quanta is demanded by the ISO standard. According to ISO standard, a CAN bit time, calculated as the sum of T_{Sync}, T_{Seg1} and T_{Seg2}, must not fall below 8 time quanta.

Calculation of the bit time:

t_q	= (BRP + 1) / f_{CAN}	if DIV8 = 0
	= 8 × (BRP + 1) / f_{CAN}	if DIV8 = 1
T_{Sync}	= 1 × t_q	
T_{Seg1}	= (TSEG1 + 1) × t_q	(min. 3 t_q)
T_{Seg2}	= (TSEG2 + 1) × t_q	(min. 2 t_q)
bit time	= T_{Sync} + T_{Seg1} + T_{Seg2}	(min. 8 t_q)

To compensate phase shifts between clocks of different CAN controllers, the CAN controller must synchronize on any edge from the recessive to the dominant bus level. If the hard synchronization is enabled (at the start of frame), the bit time is restarted at the synchronization segment. Otherwise, the re-synchronization jump width T_{SJW} defines the maximum number of time quanta, a bit time may be shortened or lengthened by one re-synchronization. The value of SJW is defined by bit field NBTRx.SJW.

T_{SJW}	= (SJW + 1) × t_q
T_{Seg1}	≥ T_{SJW} + T_{prop}
T_{Seg2}	≥ T_{SJW}

The maximum relative tolerance for f_{CAN} depends on the Phase Buffer Segments and the re-synchronization jump width.

df_{CAN}	≤ min (T_{b1}, T_{b2}) / 2 × (13 × bit time - T_{b2})	AND
df_{CAN}	≤ T_{SJW} / 20 × bit time	

A valid CAN bit timing must be written to the CAN Node Bit Timing Register NBTR before resetting the INIT bit in the Node Control Register, i.e. before enabling the operation of the CAN node.

The Node Bit Timing Register may be written only if bit CCE (Configuration Change Enable) is set in the corresponding Node Control Register.

18.3.3.2 Bitstream Processor

Based on the message objects in the message buffer, the Bitstream Processor generates the remote and Data Frames to be transmitted via the CAN bus. It controls the CRC generator and adds the checksum information to the new remote or Data Frame. After including the SOF bit and the EOF field, the Bitstream Processor starts the CAN

bus arbitration procedure and continues with the frame transmission when the bus was found in idle state. While the data transmission is running, the Bitstream Processor continuously monitors the I/O line. If (outside the CAN bus arbitration phase or the acknowledge slot) a mismatch is detected between the voltage level on the I/O line and the logic state of the bit currently sent out by the transmit shift register, a CAN error interrupt request is generated, and the error code is indicated by the Node x Status Register bit field NSRx.LEC.

The data consistency of an incoming frame is verified by checking the associated CRC field. When an error has been detected, a CAN error interrupt request is generated and the associated error code is presented in the Node x Status Register NSRx. Furthermore, an Error Frame is generated and transmitted on the CAN bus. After decomposing a faultless frame into identifier and data portion, the received information is transferred to the message buffer executing remote and Data Frame handling, interrupt generation and status processing.

18.3.3.3 Error Handling Unit

The Error Handling Unit of a CAN node x is responsible for the fault confinement of the CAN device. Its two counters, the Receive Error Counter REC and the Transmit Error Counter TEC (bit fields of the Node x Error Counter Register NECNTx, see **Page 18-87**) are incremented and decremented by commands from the Bitstream Processor. If the Bitstream Processor itself detects an error while a transmit operation is running, the Transmit Error Counter is incremented by 8. An increment of 1 is used when the error condition was reported by an external CAN node via an Error Frame generation. For error analysis, the transfer direction of the disturbed message and the node that recognizes the transfer error are indicated for the respective CAN node x in register NECNTx. Depending on the values of the error counters, the CAN node is set into error-active, error-passive, or bus-off state.

The CAN node is in error-active state if both error counters are below the error-passive limit of 128. The CAN node is in error-passive state, if at least one of the error counters is equal to or greater than 128.

The bus-off state is activated if the Transmit Error Counter is equal to or greater than the bus-off limit of 256. This state is reported for CAN node x by the Node x Status Register flag NSRx.BOFF. The device remains in this state, until the “bus-off” recovery sequence is finished. Additionally, Node x Status Register flag NSRx.EWRN is set when at least one of the error counters is equal to or greater than the error warning limit defined by the Node x Error Count Register bit field NECNTx.EWRNLVL. Bit NSRx.EWRN is reset if both error counters fall below the error warning limit again (see **Page 18-78**).

18.3.3.4 CAN Frame Counter

Each CAN node is equipped with a frame counter that counts transmitted/received CAN frames or obtains information about the time when a frame has been started to transmit or be received by the CAN node. CAN frame counting/bit time counting is performed by a 16-bit counter that is controlled by Node x Frame Counter Register NFCRx (see **Page 18-89**). Bit field NFCRx.CFSEL determines the operation mode of the frame counter:

- **Frame Count Mode:**
  After the successful transmission and/or reception of a CAN frame, the frame counter is copied into the CFCVAL bit field of the MOIPRn register of the message object involved in the transfer. Afterwards, the frame counter is incremented.
- **Time Stamp Mode:**
  The frame counter is incremented with the beginning of a new bit time. When the transmission/reception of a frame starts, the value of the frame counter is captured and stored to the CFC bit field of the NFCRx register. After the successful transfer of the frame the captured value is copied to the CFCVAL bit field of the MOIPRn register of the message object involved in the transfer.
- **Bit Timing Mode:**
  Used for baud rate detection and analysis of the bit timing (**Chapter 18.3.5.3**).

18.3.3.5 CAN Node Interrupts

Each CAN node has four hardware triggered interrupt request types that are able to generate an interrupt request upon:

- The successful transmission or reception of a frame
- A CAN protocol error with a last error code
- An alert condition: Transmit/receive error counters reach the warning limit, bus-off state changes, a List Length Error occurs, or a List Object Error occurs
- An overflow of the frame counter

Besides the hardware generated interrupts, software initiated interrupts can be generated using the Module Interrupt Trigger Register MITR. Writing a 1 to bit n of bit field MITR.IT generates an interrupt request signal on the corresponding interrupt output line INT_On. When writing MITR.IT more than one bit can be set resulting in activation of multiple INT_On interrupt output lines at the same time. See also **“Service Request Generation” on Page 18-53** for further processing of the CAN node interrupts.

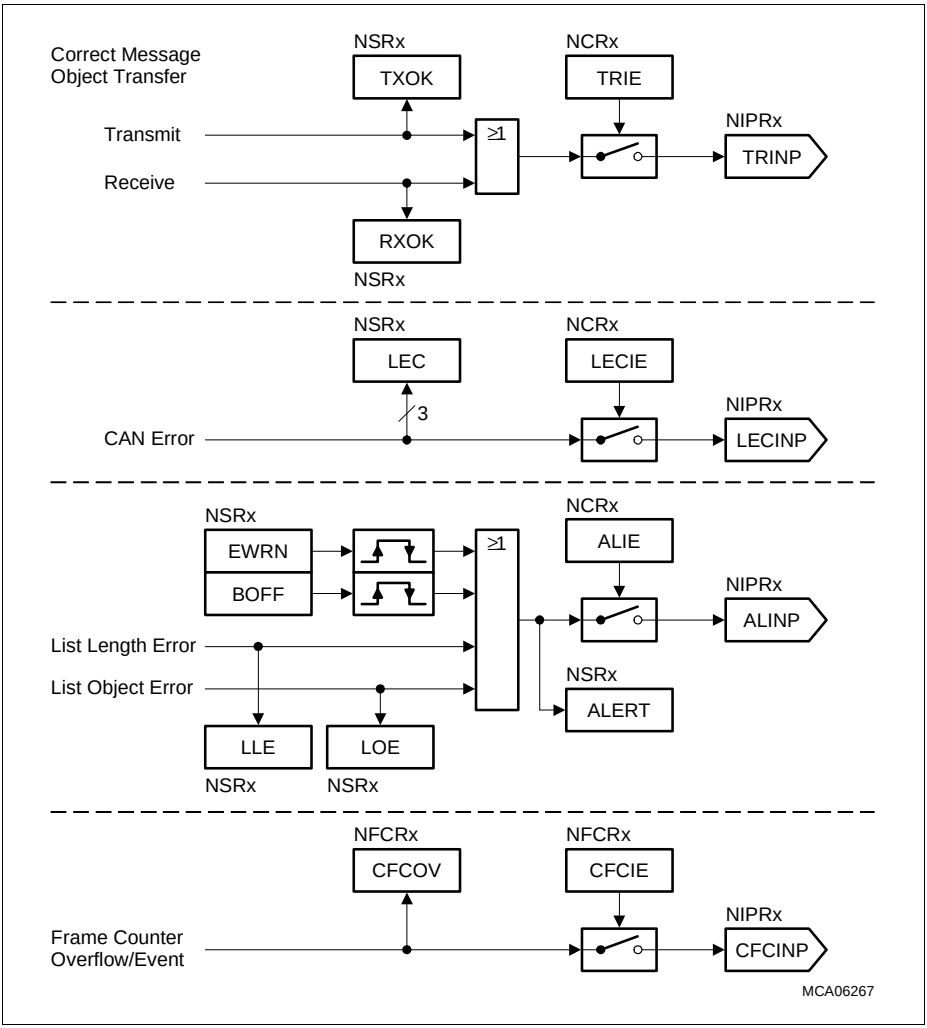


Figure 18-9 CAN Node Interrupts

18.3.4 Message Object List Structure

This section describes the structure of the message object lists in the MultiCAN module.

18.3.4.1 Basics

The message objects of the MultiCAN module are organized in double-chained lists, where each message object has a pointer to the previous message object in the list as well as a pointer to the next message object in the list. The MultiCAN module provides 8 lists. Each message object is allocated to one of these lists. In the example in **Figure 18-10**, the three message objects (3, 5, and 16) are allocated to the list with index 2 (List Register LIST2).

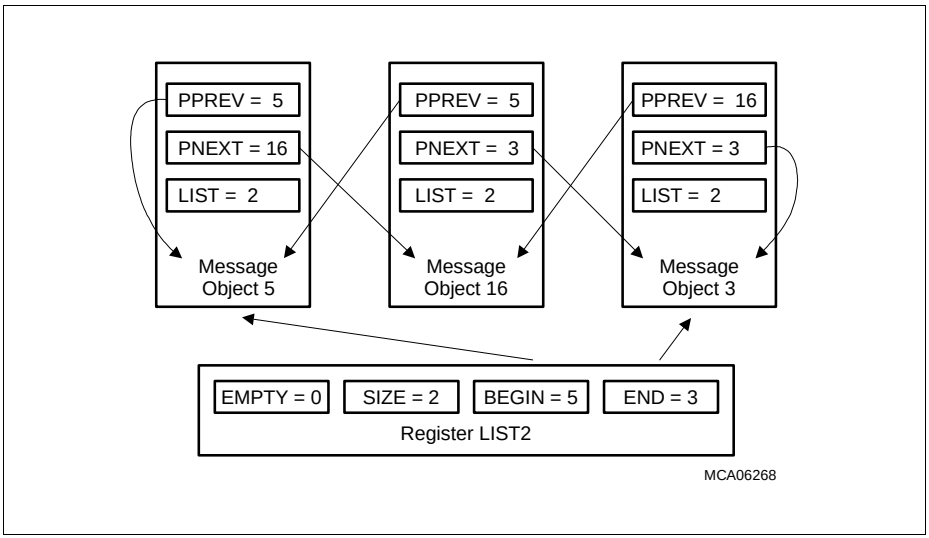


Figure 18-10 Example Allocation of Message Objects to a List

Bit field BEGIN in the List Register (for definition, see **Page 18-69**) points to the first element in the list (object 5 in the example), and bit field END points to the last element in the list (object 3 in the example). The number of elements in the list is indicated by bit field SIZE of the List Register (SIZE = number of list elements - 1, thus SIZE = 2 for the 3 elements in the example). The EMPTY bit of the List Register indicates whether or not a list is empty (EMPTY = 0 in the example, because list 2 is not empty).

Each message object n has a pointer PNEXT in its Message Object n Control Register MOCTRn (see **Page 18-93**) that points to the next message object in the list, and a pointer PPREV that points to the previous message object in the list. PPREV of the first message object points to the message object itself because the first message object has no predecessor (in the example message object 5 is the first message object in the list,

indicated by PPREV = 5). PNEXT of the last message object also points to the message object itself because the last message object has no successor (in the example, object 3 is the last message object in the list, indicated by PNEXT = 3).

Bit field MOCTRn.LIST indicates the list index number to which the message object is currently allocated. The message object of the example are allocated to list 2. Therefore, all LIST bit fields for the message objects assigned to list 2 are set to LIST = 2.

18.3.4.2 List of Unallocated Elements

The list with list index 0 has a special meaning: it is the list of all unallocated elements. An element is called unallocated if it belongs to list 0 (MOCTRn.LIST = 0). It is called allocated if it belongs to a list with an index not equal to 0 (MOCTRn.LIST > 0).

After reset, all message objects are unallocated. This means that they are assigned to the list of unallocated elements with MOCTRn.LIST = 0. After this initial allocation of the message objects caused by reset, the list of all unallocated message objects is ordered by message number (predecessor of message object n is object n-1, successor of object n is object n+1).

18.3.4.3 Connection to the CAN Nodes

Each CAN node is linked to one unique list of message objects. A CAN node performs message transfer only with the message objects that are allocated to the list of the CAN node. This is illustrated in **Figure 18-11**. Frames that are received on a CAN node may only be stored in one of the message objects that belongs to the CAN node; frames to be transmitted on a CAN node are selected only from the message objects that are allocated to that node, as indicated by the vertical arrows.

There are more lists (8) than CAN nodes (3). This means that some lists are not linked to one of the CAN nodes. A message object that is allocated to one of these unlinked lists cannot receive messages directly from a CAN node and it may not transmit messages.

FIFO and gateway mechanisms refer to message numbers and not directly to a specific list. The user must take care that the message objects targeted by FIFO/gateway belong to the desired list. The mechanisms make it possible to work with lists that do not belong to the CAN node.

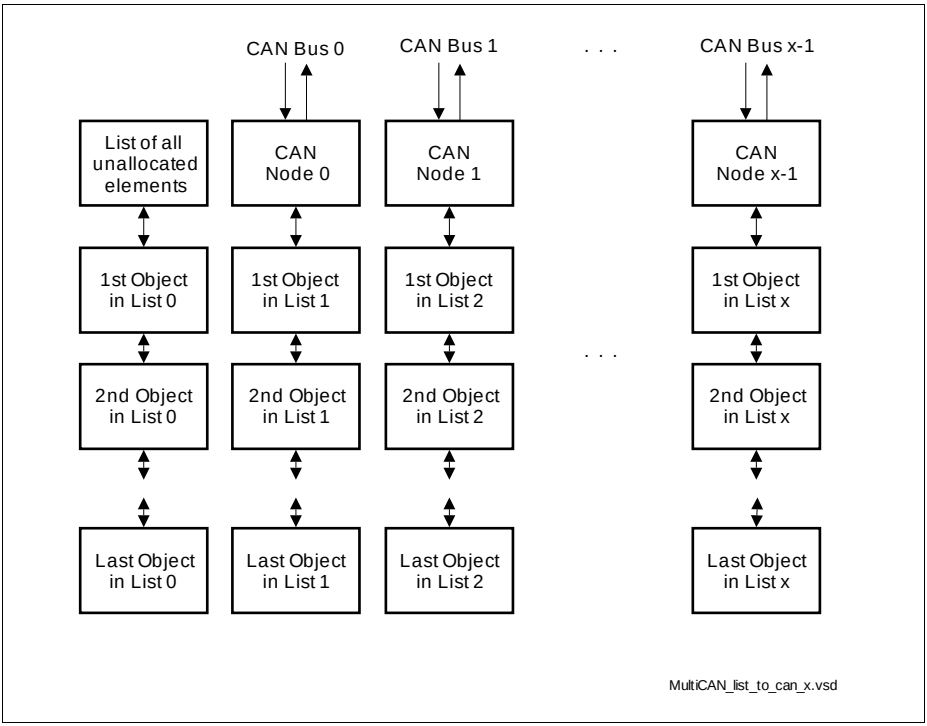


Figure 18-11 Message Objects Linked to CAN Nodes

18.3.4.4 List Command Panel

The list structure cannot be modified directly by write accesses to the LIST registers and the PPREV, PNEXT and LIST bit fields in the Message Object Control Registers, as they are read only. The list structure is managed by and limited to the list controller inside the MultiCAN module. The list controller is controlled via a command panel allowing the user to issue list allocation commands to the list controller. The list controller has two main purposes:

1. Ensure that all operations that modify the list structure result in a consistent list structure.
2. Present maximum ease of use and flexibility to the user.

The list controller and the associated command panel allows the programmer to concentrate on the final properties of the list, which are characterized by the allocation of message objects to a CAN node, and the ordering relation between objects that are allocated to the same list. The process of list (re-)building is done in the list controller.

Table 18-1 gives an overview on the available panel commands while Table 18-6 on Page 18-64 describes the panel commands in more detail.

Table 18-1 Panel Commands Overview

Command Name	Description
No Operation	No new command is started.
Initialize Lists	Run the initialization sequence to reset the CTRL and LIST field of all message objects.
Static Allocate	Allocate message object to a list.
Dynamic Allocate	Allocate the first message object of the list of unallocated objects to the selected list.
Static Insert Before	Remove a message object (source object) from the list that it currently belongs to, and insert it before a given destination object into the list structure of the destination object.
Dynamic Insert Before	Insert a new message object before a given destination object.
Static Insert Behind	Remove a message object (source object) from the list that it currently belongs to, and insert it behind a given destination object into the list structure of the destination object.
Dynamic Insert Behind	Insert a new message object behind a given destination object.

A panel command is started by writing the respective command code into the Panel Control Register bit field PANCTR.PANCMD (see Page 18-63). The corresponding command arguments must be written into bit fields PANCTR.PANAR1 and PANCTR.PANAR2 before writing the command code, or latest along with the command code in a single 32-bit write access to the Panel Control Register.

With the write operation of a valid command code, the PANCTR.BUSY flag is set and further write accesses to the Panel Control Register are ignored. The BUSY flag remains active and the control panel remains locked until the execution of the requested command has been completed. After a reset, the list controller builds up list 0. During this operation, BUSY is set and other accesses to the CAN RAM are forbidden. The CAN RAM can be accessed again when BUSY becomes inactive.

Note: The CAN RAM is automatically initialized after reset by the list controller in order to ensure correct list pointers in each message object. The end of this CAN RAM initialization is indicated by bit PANCTR.BUSY becoming inactive.

In case of a dynamic allocation command that takes an element from the list of unallocated objects, the PANCTR.RBUSY bit is also set along with the BUSY bit

(RBUSY = BUSY = 1). This indicates that bit fields PANAR1 and PANAR2 are going to be updated by the list controller in the following way:

1. The message number of the message object taken from the list of unallocated elements is written to PANAR1.
2. If ERR (bit 7 of PANAR2) is set to 1, the list of unallocated elements was empty and the command is aborted. If ERR is 0, the list was not empty and the command will be performed successfully.

The results of a dynamic allocation command are written before the list controller starts the actual allocation process. As soon as the results are available, RBUSY becomes inactive (RBUSY = 0) again, while BUSY still remains active until completion of the command. This allows the user to set up the new message object while it is still in the process of list allocation. The access to message objects is not limited during ongoing list operations. However, any access to a register resource located inside the RAM delays the ongoing allocation process by one access cycle.

As soon as the command is finished, the BUSY flag becomes inactive (BUSY = 0) and write accesses to the Panel Control Register are enabled again. Also, the “No Operation” command code is automatically written to the PANCTR.PANCMD field. A new command may be started any time when BUSY = 0.

All fields of the Panel Control Register PANCTR except BUSY and RBUSY may be written by the user. This makes it possible to save and restore the Panel Control Register if the Command Panel is used within independent (mutually interruptible) interrupt service routines. If this is the case, any task that uses the Command Panel and that may interrupt another task that also uses the Command Panel should poll the BUSY flag until it becomes inactive and save the whole PANCTR register to a memory location before issuing a command. At the end of the interrupt service routine, the task should restore PANCTR from the memory location.

Before a message object that is allocated to the list of an active CAN node shall be moved to another list or to another position within the same list, bit MOCTRn.MSGVAL (“Message Valid”) of message object n must be cleared.

18.3.5 CAN Node Analysis Features

The chapter describes the CAN node analysis capabilities of the MultiCAN module.

18.3.5.1 Analyzer Mode

The CAN Analyzer Mode makes it possible to monitor the CAN traffic for each CAN node individually without affecting the logical state of the CAN bus. The CAN Analyzer Mode for CAN node x is selected by setting Node x Control Register bit NCRx.CALM.

In CAN Analyzer Mode, the transmit pin of a CAN node is held at a recessive level permanently. The CAN node may receive frames (Data, Remote, and Error Frames) but is not allowed to transmit. Received Data/Remote Frames are not acknowledged (i.e. acknowledge slot is sent recessive) but will be received and stored in matching message objects as long as there is any other node that acknowledges the frame. The complete message object functionality is available, but no transmit request will be executed.

18.3.5.2 Loop-Back Mode

The MultiCAN module provides a Loop-Back Mode to enable an in-system test of the MultiCAN module as well as the development of CAN driver software without access to an external CAN bus.

The loop-back feature consists of an internal CAN bus (inside the MultiCAN module) and a bus select switch for each CAN node (see **Figure 18-12**). With the switch, each CAN node can be connected either to the internal CAN bus (Loop-Back Mode activated) or the external CAN bus, respectively to transmit and receive pins (normal operation). The CAN bus that is not currently selected is driven recessive; this means the transmit pin is held at 1, and the receive pin is ignored by the CAN nodes that are in Loop-Back Mode.

The Loop-Back Mode is selected for CAN node x by setting the Node x Port Control Register bit NPCRx.LBM. All CAN nodes that are in Loop-Back Mode may communicate together via the internal CAN bus without affecting the normal operation of the other CAN nodes that are not in Loop-Back Mode.

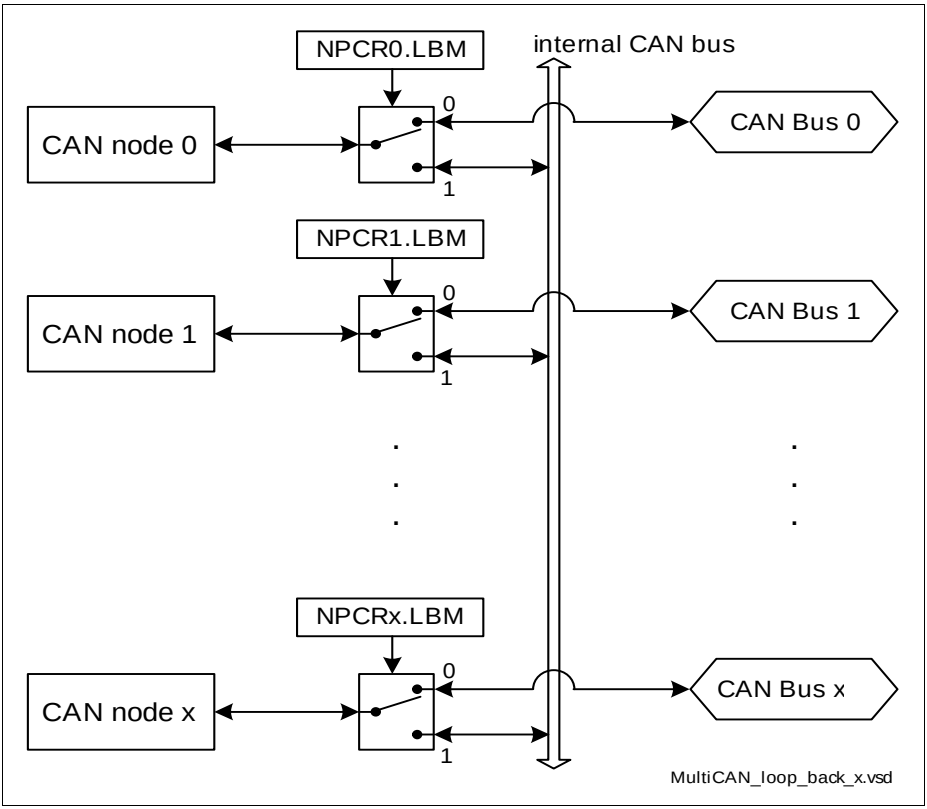


Figure 18-12 Loop-Back Mode

18.3.5.3 Bit Timing Analysis

Detailed analysis of the bit timing can be performed for each CAN node using the analysis modes of the CAN frame counter. The bit timing analysis functionality of the frame counter may be used for automatic detection of the CAN baud rate, as well as to analyze the timing of the CAN network.

Bit timing analysis for CAN node x is selected when bit field NFCRx.CFMOD = 10_B. Bit timing analysis does not affect the operation of the CAN node.

The bit timing measurement results are written into the NFCRx.CFC bit field. Whenever NFCRx.CFC is updated in bit timing analysis mode, bit NFCRx.CFCOV is also set to indicate the CFC update event. If NFCRx.CFCIE is set, an interrupt request can be generated (see **Figure 18-9**).

Automatic Baud Rate Detection

For automatic baud rate detection, the time between the observation of subsequent dominant edges on the CAN bus must be measured. This measurement is automatically performed if bit field NFCRx.CFSEL = 000_B. With each dominant edge monitored on the CAN receive input line, the time (measured in f_{CAN} clock cycles) between this edge and the most recent dominant edge is stored in the NFCRx.CFC bit field.

Synchronization Analysis

The bit time synchronization is monitored if NFCRx.CFSEL = 010_B. The time between the first dominant edge and the sample point is measured and stored in the NFCRx.CFC bit field. The bit timing synchronization offset may be derived from this time as the first edge after the sample point triggers synchronization and there is only one synchronization between consecutive sample points.

Synchronization analysis can be used, for example, for fine tuning of the baud rate during reception of the first CAN frame with the measured baud rate.

Driver Delay Measurement

The delay between a transmitted edge and the corresponding received edge is measured when NFCRx.CFSEL = 011_B (dominant to dominant) and NFCRx.CFSEL = 100_B (recessive to recessive). These delays indicate the time needed to represent a new bit value on the physical implementation of the CAN bus.

18.3.6 Message Acceptance Filtering

The chapter describes the Message Acceptance Filtering capabilities of the MultiCAN module.

18.3.6.1 Receive Acceptance Filtering

When a CAN frame is received by a CAN node, a unique message object is determined in which the received frame is stored after successful frame reception. A message object is qualified for reception of a frame if the following six conditions are met.

- The message object is allocated to the message object list of the CAN node by which the frame is received.
- Bit MOSTATn.MSGVAL in the Message Status Register (see **Page 18-96**) is set.
- Bit MOSTATn.RXEN is set.
- Bit MOSTATn.DIR is equal to bit RTR of the received frame.
  If bit MOSTATn.DIR = 1 (transmit object), the message object accepts only Remote Frames. If bit MOSTATn.DIR = 0 (receive object), the message object accepts only Data Frames.
- If bit MOAMRn.MIDE = 1, the IDE bit of the received frame becomes evaluated in the following way: If MOARn.IDE = 1, the IDE bit of the received frame must be set (indicates extended identifier). If MOARn.IDE = 0, the IDE bit of the received frame must be cleared (indicates standard identifier).
  If bit MOAMRn.MIDE = 0, the IDE bit of the received frame is “don’t care”. In this case, message objects with standard and extended frames are accepted.
- The identifier of the received frame matches the identifier stored in the Arbitration Register of the message object as qualified by the acceptance mask in the MOAMRn register. This means that each bit of the received message object identifier is equal to the bit field MOARn.ID, except those bits for which the corresponding acceptance mask bits in bit field MOAMRn.AM are cleared. These identifier bits are “don’t care” for reception. **Figure 18-13** illustrates this receive message identifier check.

Among all messages that fulfill all six qualifying criteria the message object with the highest receive priority wins receive acceptance filtering and becomes selected to store the received frame. All other message objects lose receive acceptance filtering.

The following priority scheme is defined for the message objects:

A message object a (MOa) has higher receive priority than a message object b (MOb) if the following two conditions are fulfilled (see **Page 18-110**):

1. MOa has a higher priority class than MOb. This means, the 2-bit priority bit field MOARa.PRI must be equal or less than bit field MOARb.PRI.
2. If both message objects have the same priority class (MOARa.PRI = MOARb.PRI), MOb is a list successor of MOa. This means that MOb can be reached by means of successively stepping forward in the list, starting from a.

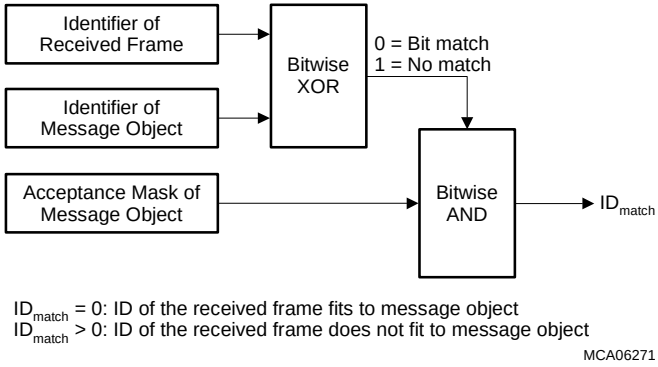


Figure 18-13 Received Message Identifier Acceptance Check

18.3.6.2 Transmit Acceptance Filtering

A message is requested for transmission by setting a transmit request in the message object that holds the message. If more than one message object have a valid transmit request for the same CAN node, one of these message objects is chosen for transmission, because only a single message object can be transmitted at one time on a CAN bus.

A message object is qualified for transmission on a CAN node if the following four conditions are met (see also **Figure 18-14**).

1. The message object is allocated to the message object list of the CAN node.
2. Bit MOSTATn.MSGVAL is set.
3. Bit MOSTATn.TXRQ is set.
4. Bit MOSTATn.TXEN0 and MOSTATn.TXEN1 are set.

A priority scheme determines which one of all qualifying message objects is transmitted first. It is assumed that message object a (MOa) and message object b (MOb) are two message objects qualified for transmission. MOb is a list successor of MOa. For both message objects, CAN messages CANa and CANb are defined (identifier, IDE, and RTR are taken from the message-specific bit fields and bits MOARn.ID, MOARn.IDE and MOCTRn.DIR).

If both message objects belong to the same priority class (identical PRI bit field in register MOARn), MOa has a higher transmit priority than MOb if one of the following conditions is fulfilled.

- PRI = 10_B and CAN message MOa has higher or equal priority than CAN message MOb with respect to CAN arbitration rules (see **Table 18-12** on **Page 18-111**).
- PRI = 01_B or PRI = 11_B (priority by list order).

The message object that is qualified for transmission and has highest transmit priority wins the transmit acceptance filtering, and will be transmitted first. All other message objects lose the current transmit acceptance filtering round. They get a new chance in subsequent acceptance filtering rounds.

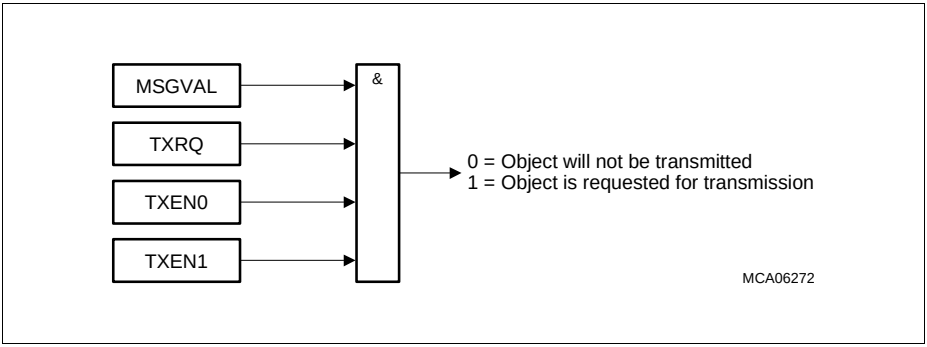


Figure 18-14 Effective Transmit Request of Message Object

18.3.7 Message Postprocessing

After a message object has successfully received or transmitted a frame, the CPU can be notified to perform a postprocessing on the message object. The postprocessing of the MultiCAN module consists of two elements:

1. Message interrupts to trigger postprocessing.
2. Message pending registers to collect pending message interrupts into a common structure for postprocessing.

18.3.7.1 Message Object Interrupts

When the storage of a received frame into a message object or the successful transmission of a frame is completed, a message interrupt can be issued. For each message object, a transmit and a receive interrupt can be generated and routed to one of the sixteen CAN interrupt output lines (see **Figure 18-15**). A receive interrupt occurs also after a frame storage event that has been induced by a FIFO or a gateway action. The status bits TXPND and RXPND in the Message Object n Status Register are always set after a successful transmission/reception, whether or not the respective message interrupt is enabled.

A third FIFO full interrupt condition of a message object is provided. If bit field MOFCRn.OVIE (Overflow Interrupt Enable) is set, the FIFO full interrupt will be activated depending on the actual message object type.

In case of a Receive FIFO Base Object (MOFCRn.MMC = 0001_B), the FIFO full interrupt is routed to the interrupt output line INT_Om as defined by the transmit interrupt node pointer MOIPRn.TXINP.

In case of a Transmit FIFO Base Object (MOFCRn.MMC = 0010_B), the FIFO full interrupt becomes routed to the interrupt output line INT_Om as defined by the receive interrupt node pointer MOIPRn.RXINP.

See also **“Service Request Generation” on Page 18-53** for further processing of the message object interrupts.

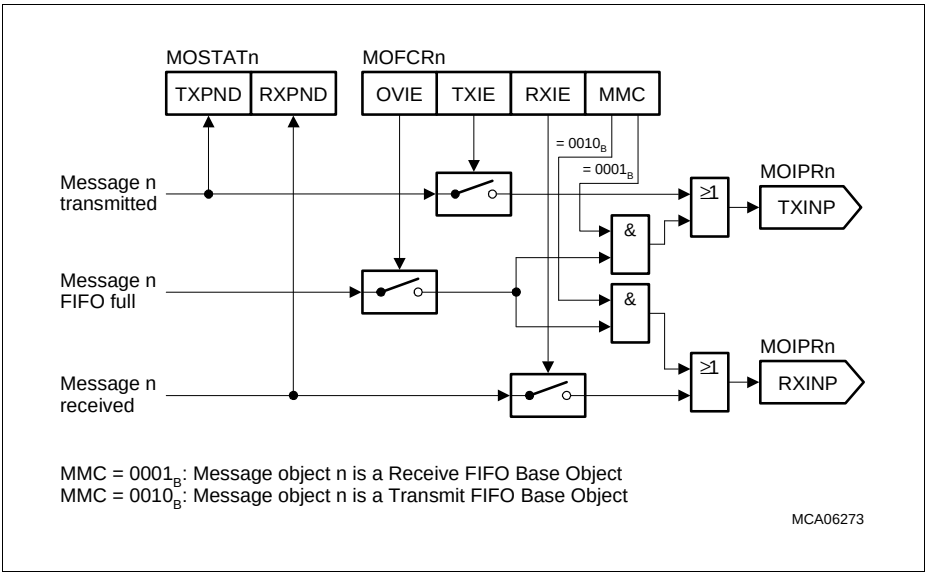


Figure 18-15 Message Interrupt Request Routing

18.3.7.2 Pending Messages

When a message interrupt request is generated, a message pending bit is set in one of the Message Pending Registers. There are 8 Message Pending Registers, MSPNDk (k = 0-7) with 32 pending bits available each. The general **Figure 18-16** shows the allocation of the message pending bits in case that the maximum possible number of eight Message Pending Registers are implemented and available on the chip.

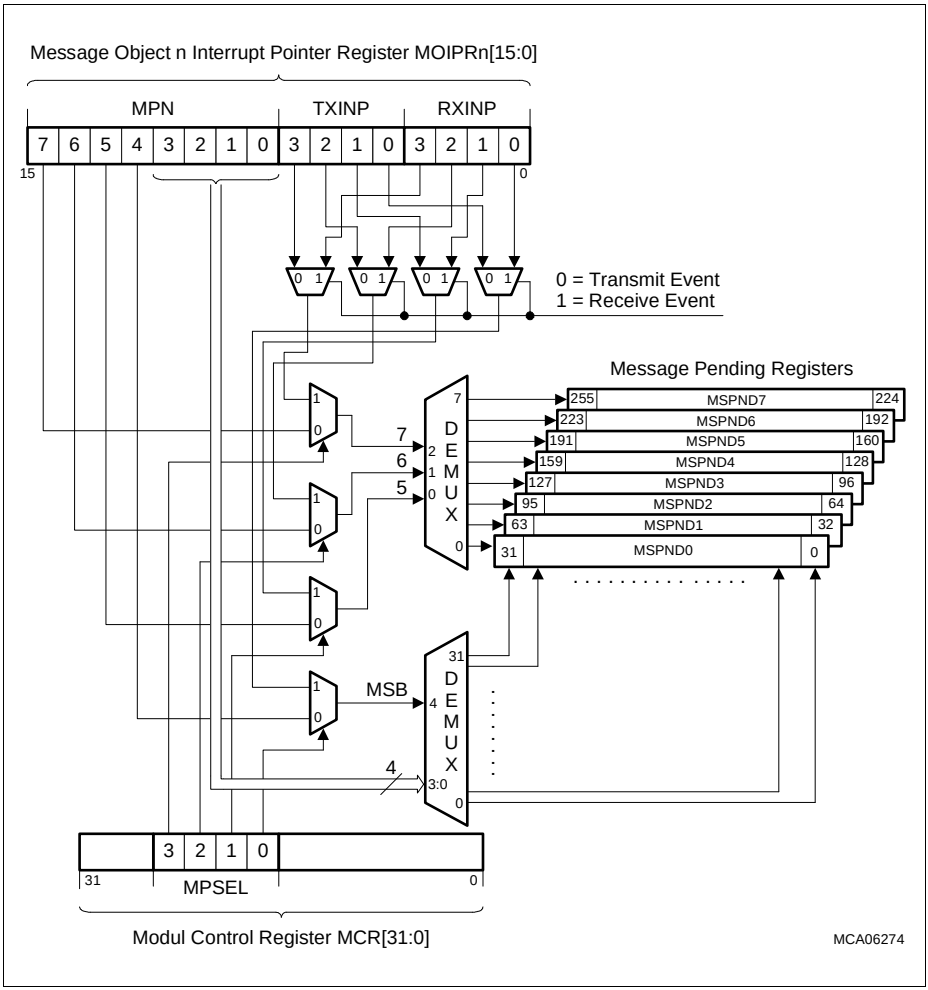


Figure 18-16 Message Pending Bit Allocation

The location of a pending bit is defined by two demultiplexers selecting the number k of the MSPNDk registers (3-bit demux), and the bit location within the corresponding MSPNDk register (5-bit demux).

Allocation Case 1

In this allocation case, bit field MCR.MPSEL = 0000_B (see **Page 18-67**). The location selection consists of 2 parts:

- The upper three bits of MOIPRn.MPN (MPN[7:5]) select the number k of a Message Pending Register MSPNDk in which the pending bit will be set.
- The lower five bits of MOIPRn.MPN (MPN[4:0]) select the bit position (0-31) in MSPNDk for the pending bit to be set.

Allocation Case 2

In this allocation case, bit field MCR.MPSEL is taken into account for pending bit allocation. Bit field MCR.MPSEL makes it possible to include the interrupt request node pointer for reception (MOIPRn.RXINP) or transmission (MOIPRn.TXINP) for pending bit allocation in such a way that different target locations for the pending bits are used in receive and transmit case. If MPSEL = 1111_B, the location selection operates in the following way:

- At a transmit event, the upper 3 bits of TXINP determine the number k of a Message Pending Register MSPNDk in which the pending bit will be set. At a receive event, the upper 3 bits of RXINP determine the number k.
- The bit position (0-31) in MSPNDk for the pending bit to be set is selected by the lowest bit of TXINP or RXINP (selects between low and high half-word of MSPNDk) and the four least significant bits of MPN.

General Hints

The Message Pending Registers MSPNDk can be written by software. Bits that are written with 1 are left unchanged, and bits which are written with 0 are cleared. This makes it possible to clear individual MSPNDk bits with a single register write access. Therefore, access conflicts are avoided when the MultiCAN module (hardware) sets another pending bit at the same time when software writes to the register.

Each Message Pending Register MSPNDk is associated with a Message Index Register MSIDk (see **Page 18-72**) which indicates the lowest bit position of all set (1) bits in Message Pending Register k. The MSIDk register is a read-only register that is updated immediately when a value in the corresponding Message Pending Register k is changed by software or hardware.

18.3.8 Message Object Data Handling

This chapter describes the handling capabilities for the Message Object Data of the MultiCAN module.

18.3.8.1 Frame Reception

After the reception of a message, it is stored in a message object according to the scheme shown in **Figure 18-17**. The MultiCAN module not only copies the received data into the message object, and it provides advanced features to enable consistent data exchange between MultiCAN and CPU.

MSGVAL

Bit MSGVAL (Message Valid) in the Message Object n Status Register MOSTATn is the main switch of the message object. During the frame reception, information is stored in the message object only when MSGVAL = 1. If bit MSGVAL is reset by the CPU, the MultiCAN module stops all ongoing write accesses to the message object. Now the message object can be re-configured by the CPU with subsequent write accesses to it without being disturbed by the MultiCAN.

RTSEL

When the CPU re-configures a message object during CAN operation (for example, clears MSGVAL, modifies the message object and sets MSGVAL again), the following scenario can occur:

1. The message object wins receive acceptance filtering.
2. The CPU clears MSGVAL to re-configure the message object.
3. The CPU sets MSGVAL again after re-configuration.
4. The end of the received frame is reached. As MSGVAL is set, the received data is stored in the message object, a message interrupt request is generated, gateway and FIFO actions are processed, etc.

After the re-configuration of the message object (after step 3 above) the storage of further received data may be undesirable. This can be achieved through bit MOCTRn.RTSEL (Receive/Transmit Selected) that makes it possible to disconnect a message object from an ongoing frame reception.

When a message object wins the receive acceptance filtering, its RTSEL bit is set by the MultiCAN module to indicate an upcoming frame delivery. The MultiCAN module checks RTSEL whether it is set on successful frame reception to verify that the object is still ready for receiving the frame. The received frame is then stored in the message object (along with all subsequent actions such as message interrupts, FIFO & gateway actions, flag updates) only if RTSEL = 1.

When a message object is invalidated during CAN operation (resetting bit MSGVAL), RTSEL should be cleared before setting MSGVAL again (latest with the same write

access that sets MSGVAL) to prevent the storage of a frame that belongs to the old context of the message object. Therefore, a message object re-configuration should consist of the following steps:

1. Clear MSGVAL bit
2. Re-configure the message object while MSGVAL = 0
3. Clear RTSEL bit and set MSGVAL again

RXEN

Bit MOSTATn.RXEN enables a message object for frame reception. A message object can receive CAN messages from the CAN bus only if RXEN = 1. The MultiCAN module evaluates RXEN only during receive acceptance filtering. After receive acceptance filtering, RXEN is ignored and has no further influence on the actual storage of a received message in a message object.

Bit RXEN enables the “soft phase out” of a message object: after clearing RXEN, a currently received CAN message for which the message object has won acceptance filtering is still stored in the message object but for subsequent messages the message object no longer wins receive acceptance filtering.

RXUPD, NEWDAT and MSGLST

An ongoing frame storage process is indicated by the RXUPD (Receive Updating) flag in the MOSTATn register. RXUPD is set with the start and cleared with the end of a message object update, which consists of frame storage as well as flag updates.

After storing the received frame (identifier, IDE bit, DLC; including the Data Field for Data Frames), the NEWDAT (New Data) bit of the message object is set. If NEWDAT was already set before it becomes set again, bit MSGLST (Message Lost) is set to indicate a data loss condition.

The RXUPD and NEWDAT flags can help to read consistent frame data from the message object during an ongoing CAN operation. The following steps are recommended to be executed:

1. Clear NEWDAT bit.
2. Read message content (identifier, data etc.) from the message object.
3. Check that both, NEWDAT and RXUPD, are cleared. If this is not the case, go back to step 1.
4. When step 3 was successful, the message object contents are consistent and has not been updated by the MultiCAN module while reading.

Bits RXUPD, NEWDAT and MSGLST have the same behavior for the reception of Data as well as Remote Frames.

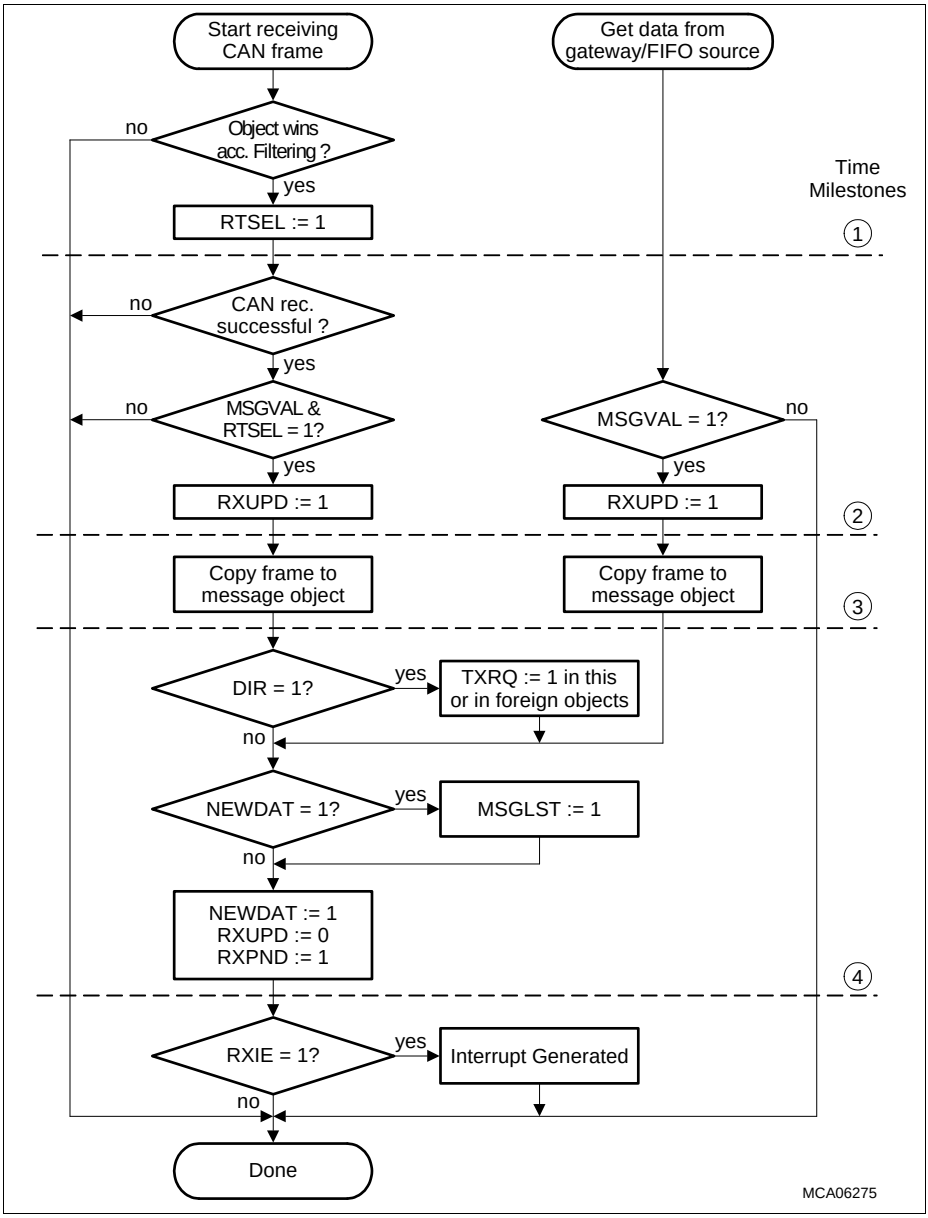


Figure 18-17 Reception of a Message Object

18.3.8.2 Frame Transmission

The process of a message object transmission is shown in **Figure 18-18**. Along with the copy of the message object content to be transmitted (identifier, IDE bit, RTR = DIR bit, DLC, including the Data Field for Data Frames) into the internal transmit buffer of the assigned CAN node, several status flags are also served and monitored to control consistent data handling.

The transmission process of a message object starting after the transmit acceptance filtering is identical for Remote and Data Frames.

MSGVAL, TXRQ, TXEN0, TXEN1

A message can only be transmitted if all four bits in registers MOSTATn, MSGVAL (Message Valid), TXRQ (Transmit Request), TXEN0 (Transmit Enable 0), TXEN1 (Transmit Enable 1) are set as shown in **Figure 18-14**. Although these bits are equivalent with respect to the transmission process, they have different semantics:

Table 18-2 Message Transmission Bit Definitions

Bit	Description
MSGVAL	**Message Valid** This is the main switch bit of the message object.
TXRQ	**Transmit Request** This is the standard transmit request bit. This bit must be set whenever a message object should be transmitted. TXRQ is cleared by hardware at the end of a successful transmission, except when there is new data (indicated by NEWDAT = 1) to be transmitted. When bit MOFCRn.STT (“Single Transmit Trial”) is set, TXRQ becomes already cleared when the contents of the message object are copied into the transmit frame buffer of the CAN node. A received remote request (after a Remote Frame reception) sets bit TXRQ to request the transmission of the requested data frame.
TXEN0	**Transmit Enable 0** This bit can be temporarily cleared by software to suppress the transmission of this message object when it writes new content to the Data Field. This avoids transmission of inconsistent frames that consist of a mixture of old and new data. Remote requests are still accepted when TXEN0 = 0, but transmission of the Data Frame is suspended until transmission is re-enabled by software (setting TXEN0).

Table 18-2 Message Transmission Bit Definitions (cont'd)

Bit	Description
TXEN1	**Transmit Enable 1** This bit is used in transmit FIFOs to select the message object that is transmit active within the FIFO structure. For message objects that are not transmit FIFO elements, TXEN1 can either be set permanently to 1 or can be used as a second independent transmission enable bit.

RTSEL

When a message object has been identified to be transmitted next after transmission acceptance filtering, bit MOCTRn.RTSEL (Receive/Transmit Selected) is set.

When the message object is copied into the internal transmit buffer, bit RTSEL is checked, and the message is transmitted only if RTSEL = 1. After the successful transmission of the message, bit RTSEL is checked again and the message postprocessing is only executed if RTSEL = 1.

For a complete re-configuration of a valid message object, the following steps should be executed:

1. Clear MSGVAL bit
2. Re-configure the message object while MSGVAL = 0
3. Clear RTSEL and set MSGVAL

Clearing of RTSEL ensures that the message object is disconnected from an ongoing/scheduled transmission and no message object processing (copying message to transmit buffer including clearing NEWDAT, clearing TXRQ, time stamp update, message interrupt, etc.) within the old context of the object can occur after the message object becomes valid again, but within a new context.

NEWDAT

When the contents of a message object have been transferred to the internal transmit buffer of the CAN node, bit MOSTATn.NEWDAT (New Data) is cleared by hardware to indicate that the transmit message object data is no longer new.

When the transmission of the frame is successful and NEWDAT is still cleared (if no new data has been copied into the message object meanwhile), TXRQ (Transmit Request) is cleared automatically by hardware.

If, however, the NEWDAT bit has been set again by the software (because a new frame should be transmitted), TXRQ is not cleared to enable the transmission of the new data.

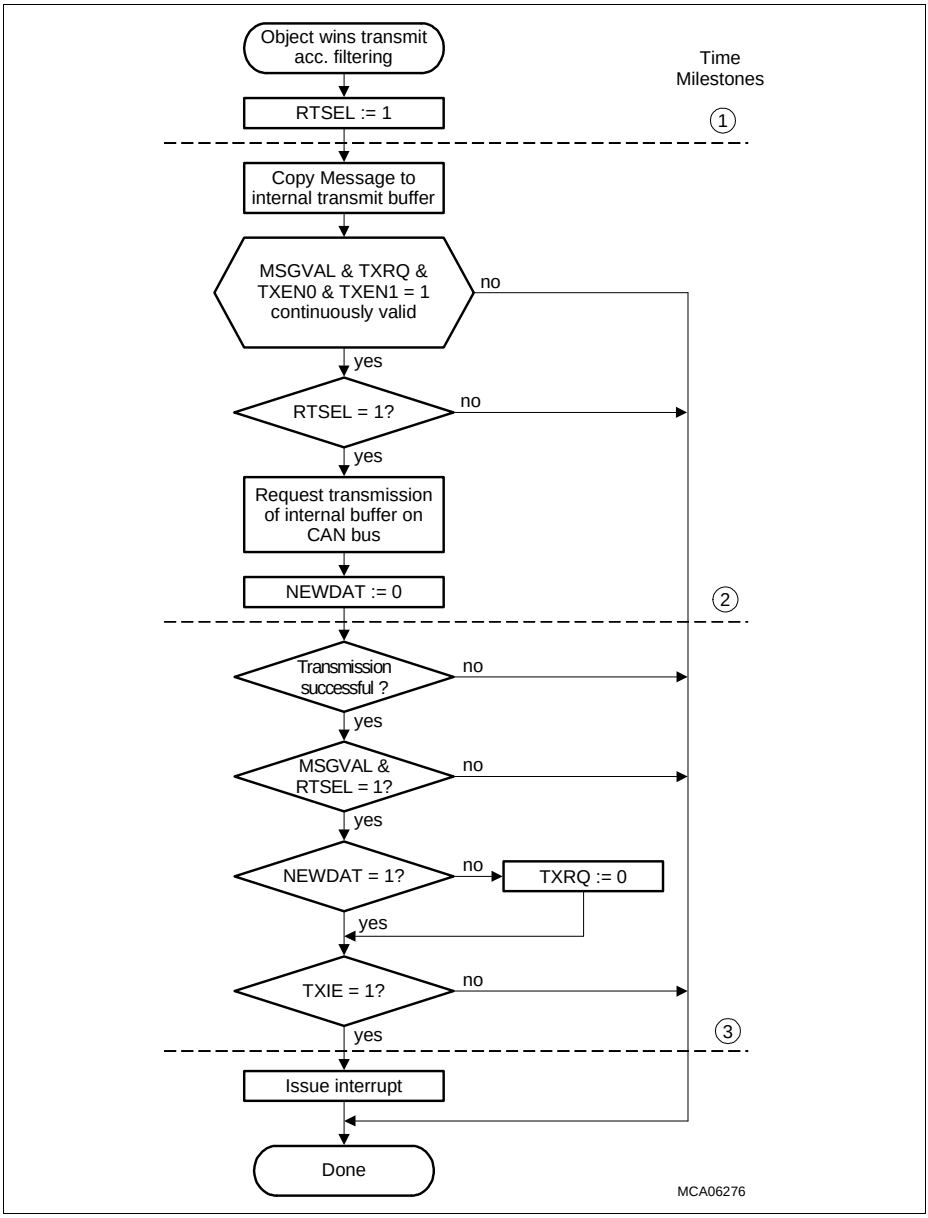


Figure 18-18 Transmission of a Message Object

18.3.9 Message Object Functionality

This chapter describes the functionality of the Message Objects in the MultiCAN module.

18.3.9.1 Standard Message Object

A message object is selected as standard message object when bit field MOFCRn.MMC = 0000_B (see **Page 18-103**). The standard message object can transmit and receive CAN frames according to the basic rules described in the previous sections. Additional services such as Single Data Transfer Mode or Single Transmit Trial (see following sections) are available and can be individually selected.

18.3.9.2 Single Data Transfer Mode

Single Data Transfer Mode is a useful feature in order to broadcast data over the CAN bus without unintended duplication of information. Single Data Transfer Mode is selected via bit MOFCRn.SDT.

Message Reception

When a received message stored in a message object is overwritten by a new received message, the contents of the first message are lost and replaced with the contents of the new received message (indicated by MSGLST = 1).

If SDT is set (Single Data Transfer Mode activated), bit MSGVAL of the message object is automatically cleared by hardware after the storage of a received Data Frame. This prevents the reception of further messages.

After the reception of a Remote Frame, bit MSGVAL is not automatically cleared.

Message Transmission

When a message object receives a series of multiple remote requests, it transmits several Data Frames in response to the remote requests. If the data within the message object has not been updated in the time between the transmissions, the same data can be sent more than once on the CAN bus.

In Single Data Transfer Mode (SDT = 1), this is avoided because MSGVAL is automatically cleared after the successful transmission of a Data Frame.

After the transmission of a Remote Frame, bit MSGVAL is not automatically cleared.

18.3.9.3 Single Transmit Trial

If the bit STT in the message object function register is set (STT = 1), the transmission request is cleared (TXRQ = 0) when the frame contents of the message object have been copied to the internal transmit buffer of the CAN node. Thus, the transmission of the message object is not tried again when it fails due to CAN bus errors.

18.3.9.4 Message Object FIFO Structure

In case of high CPU load it may be difficult to process a series of CAN frames in time. This may happen if multiple messages are received or must be transmitted in short time.

Therefore, a FIFO buffer structure is available to avoid loss of incoming messages and to minimize the setup time for outgoing messages. The FIFO structure can also be used to automate the reception or transmission of a series of CAN messages and to generate a single message interrupt when the whole CAN frame series is done.

There can be several FIFOs in parallel. The number of FIFOs and their size are limited only by the number of available message objects. A FIFO can be installed, resized and de-installed at any time, even during CAN operation.

The basic structure of a FIFO is shown in **Figure 18-19**. A FIFO consists of one base object and n slave objects. The slave objects are chained together in a list structure (similar as in message object lists). The base object may be allocated to any list. Although **Figure 18-19** shows the base object as a separate part beside the slave objects, it is also possible to integrate the base object at any place into the chain of slave objects. This means that the base object is slave object, too (not possible for gateways). The absolute object numbers of the message objects have no impact on the operation of the FIFO.

The base object does not need to be allocated to the same list as the slave objects. Only the slave object must be allocated to a common list (as they are chained together). Several pointers (BOT, CUR and TOP) that are located in the Message Object n FIFO/Gateway Pointer Register MOFGPRn link the base object to the slave objects, regardless whether the base object is allocated to the same or to another **list** than the slave objects.

The smallest FIFO would be a single message object which is both, FIFO base and FIFO slave (not very useful). The biggest possible FIFO structure would include all message objects of the MultiCAN module. Any FIFO sizes between these limits are possible.

In the FIFO base object, the FIFO boundaries are defined. Bit field MOFGPRn.BOT of the base object points to (includes the number of) the bottom slave object in the FIFO structure. The MOFGPRn.TOP bit field points to (includes the number of) the top slave object in the FIFO structure. The MOFGPRn.CUR bit field points to (includes the number of) the slave object that is actually selected by the MultiCAN module for message transfer. When a message transfer takes place with this object, CUR is set to the next message object in the list structure of the slave objects (CUR = PNEXT of current object). If CUR was equal to TOP (top of the FIFO reached), the next update of CUR will result in CUR = BOT (wrap-around from the top to the bottom of the FIFO). This scheme represents a circular FIFO structure where the bit fields BOT and TOP establish the link from the last to the first element.

Bit field MOFGPRn.SEL of the base object can be used for monitoring purposes. It makes it possible to define a slave object within the list at which a message interrupt is

generated whenever the CUR pointer reaches the value of the SEL pointer. Thus SEL makes it possible to detect the end of a predefined message transfer series or to issue a warning interrupt when the FIFO becomes full.

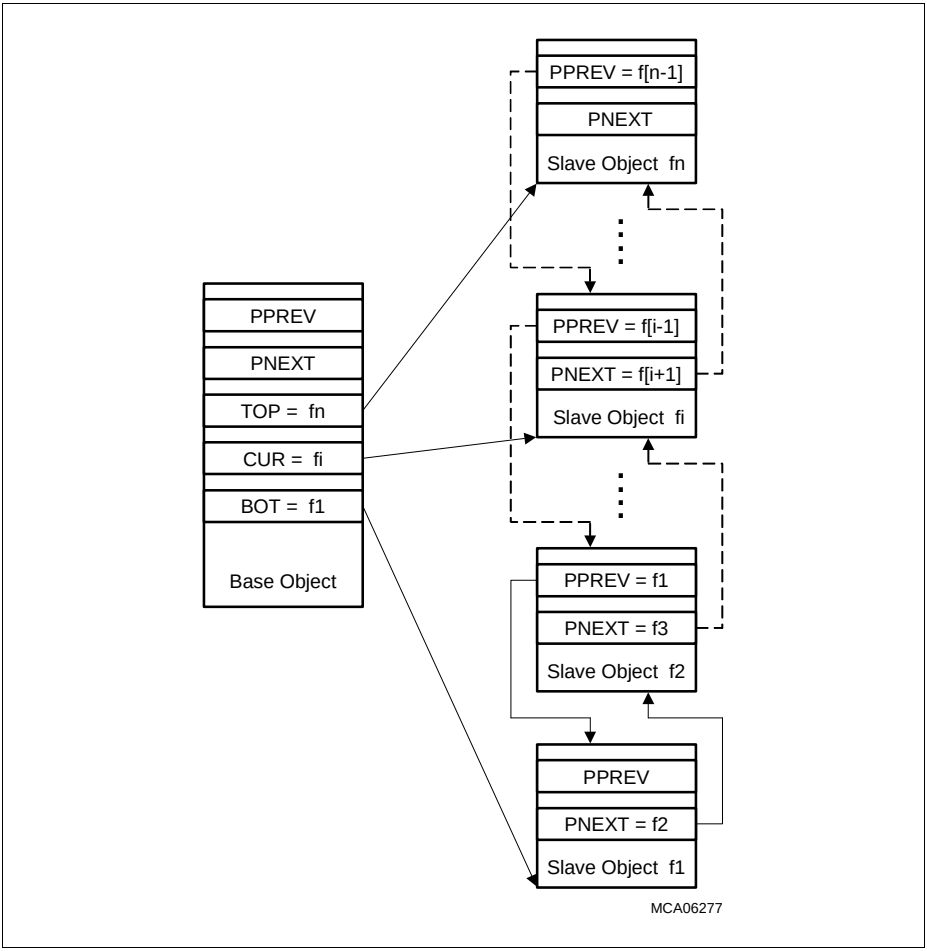


Figure 18-19 FIFO Structure with FIFO Base Object and n FIFO Slave Objects

18.3.9.5 Receive FIFO

The Receive FIFO structure is used to buffer incoming (received) Remote or Data Frames.

A Receive FIFO is selected by setting MOFCRn.MMC = 0001_B in the FIFO base object. This MMC code automatically designates a message object as FIFO base object. The message modes of the FIFO slave objects are not relevant for the operation of the Receive FIFO.

When the FIFO base object receives a frame from the CAN node it belongs to, the frame is not stored in the base object itself but in the message object that is selected by the base object's MOFGPRn.CUR pointer. This message object receives the CAN message as if it is the direct receiver of the message. However, MOFCRn.MMC = 0000_B is implicitly assumed for the FIFO slave object, and a standard message delivery is performed. The actual message mode (MMC setting) of the FIFO slave object is ignored. For the slave object, no acceptance filtering takes place that checks the received frame for a match with the identifier, IDE bit, and DIR bit.

With the reception of a CAN frame, the current pointer CUR of the base object is set to the number of the next message object in the FIFO structure. This message object will then be used to store the next incoming message.

If bit field MOFCRn.OVIE ("Overflow Interrupt Enable") of the FIFO base object is set and the current pointer MOFGPRn.CUR becomes equal to MOFGPRn.SEL, a FIFO overflow interrupt request is generated. This interrupt request is generated on interrupt node TXINP of the base object immediately after the storage of the received frame in the slave object. Transmit interrupts are still generated if TXIE is set.

A CAN message is stored in a FIFO slave only if MSGVAL = 1 in both FIFO base and slave object.

In order to avoid direct reception of a message by a slave message object, as if it was an independent message object and not a part of a FIFO, the bit RXEN of each slave object must be cleared. The setting of the bit RXEN is "don't care" only if the slave object is located in a list not assigned to a CAN node.

18.3.9.6 Transmit FIFO

The Transmit FIFO structure is used to buffer a series of Data or Remote Frames that must be transmitted. A transmit FIFO consists of one base message object and one or more slave message objects.

A Transmit FIFO is selected by setting MOFCRn.MMC = 0010_B in the FIFO base object. Unlike the Receive FIFO, slave objects assigned to the Transmit FIFO must explicitly set their bit fields MOFCRn.MMC = 0011_B. The CUR pointer in all slave objects must point back to the Transmit FIFO Base Object (to be initialized by software).

The MOSTATn.TXEN1 bits (Transmit Enable 1) of all message objects except the one which is selected by the CUR pointer of the base object must be cleared by software. TXEN1 of the message (slave) object selected by CUR must be set. CUR (of the base object) may be initialized to any FIFO slave object.

When tagging the message objects of the FIFO as valid to start the operation of the FIFO, then the base object must be tagged valid (MSGVAL = 1) first.

Before a Transmit FIFO becomes de-installed during operation, its slave objects must be tagged invalid (MSGVAL = 0).

The Transmit FIFO uses the TXEN1 bit in the Message Object Control Register of all FIFO elements to select the actual message for transmission. Transmit acceptance filtering evaluates TXEN1 for each message object and a message object can win transmit acceptance filtering only if its TXEN1 bit is set. When a FIFO object has transmitted a message, the hardware clears its TXEN1 bit in addition to standard transmit postprocessing (clear TXRQ, transmit interrupt etc.), and moves the CUR pointer in the next FIFO base object to be transmitted. TXEN1 is set automatically (by hardware) in the next message object. Thus, TXEN1 moves along the Transmit FIFO structure as a token that selects the active element.

If bit field MOFCRn.OVIE (“Overflow Interrupt Enable”) of the FIFO base object is set and the current pointer CUR becomes equal to MOFGPRn.SEL, a FIFO overflow interrupt request is generated. The interrupt request is generated on interrupt node RXINP of the base object after postprocessing of the received frame. Receive interrupts are still generated for the Transmit FIFO base object if bit RXIE is set.

18.3.9.7 Gateway Mode

The Gateway Mode makes it possible to establish an automatic information transfer between two independent CAN buses without CPU interaction.

The Gateway Mode operates on message object level. In Gateway mode, information is transferred between two message objects, resulting in an information transfer between the two CAN nodes to which the message objects are allocated. A gateway may be established with any pair of CAN nodes, and there can be as many gateways as there are message objects available to build the gateway structure.

Gateway Mode is selected by setting MOFCRs.MMC = 0100_B for the gateway source object s. The gateway destination object d is selected by the MOFGPRd.CUR pointer of the source object. The gateway destination object only needs to be valid (its MSGVAL = 1). All other settings are not relevant for the information transfer from the source object to the destination object.

Gateway source object behaves as a standard message object with the difference that some additional actions are performed by the MultiCAN module when a CAN frame has been received and stored in the source object (see **Figure 18-20**):

1. If bit MOFCRs.DLCC is set, the data length code MOFCRs.DLC is copied from the gateway source object to the gateway destination object.
2. If bit MOFCRs.IDC is set, the identifier MOARs.ID and the identifier extension MOARs.IDE are copied from the gateway source object to the gateway destination object.
3. If bit MOFCRs.DATC is set, the data bytes stored in the two data registers MODATALs and MODATAHs are copied from the gateway source object to the gateway destination object. All 8 data bytes are copied, even if MOFCRs.DLC indicates less than 8 data bytes.
4. If bit MOFCRs.GDFS is set, the transmit request flag MOSTATd.TXRQ is set in the gateway destination object.
5. The receive pending bit MOSTATd.RXPND and the new data bit MOSTATd.NEWDAT are set in the gateway destination object.
6. A message interrupt request is generated for the gateway destination object if its MOSTATd.RXIE is set.
7. The current object pointer MOFGPRs.CUR of the gateway source object is moved to the next destination object according to the FIFO rules as described on **Page 18-46**. A gateway with a single (static) destination object is obtained by setting MOFGPRs.TOP = MOFGPRs.BOT = MOFGPRs.CUR = destination object.

The link from the gateway source object to the gateway destination object works in the same way as the link from a FIFO base to a FIFO slave. This means that a gateway with an integrated destination FIFO may be created; in **Figure 18-19**, the object on the left is the gateway source object and the message object on the right side is the gateway destination objects.

The gateway operates equivalent for the reception of data frames (source object is receive object, i.e. DIR = 0) as well as for the reception of Remote Frames (source object is transmit object).

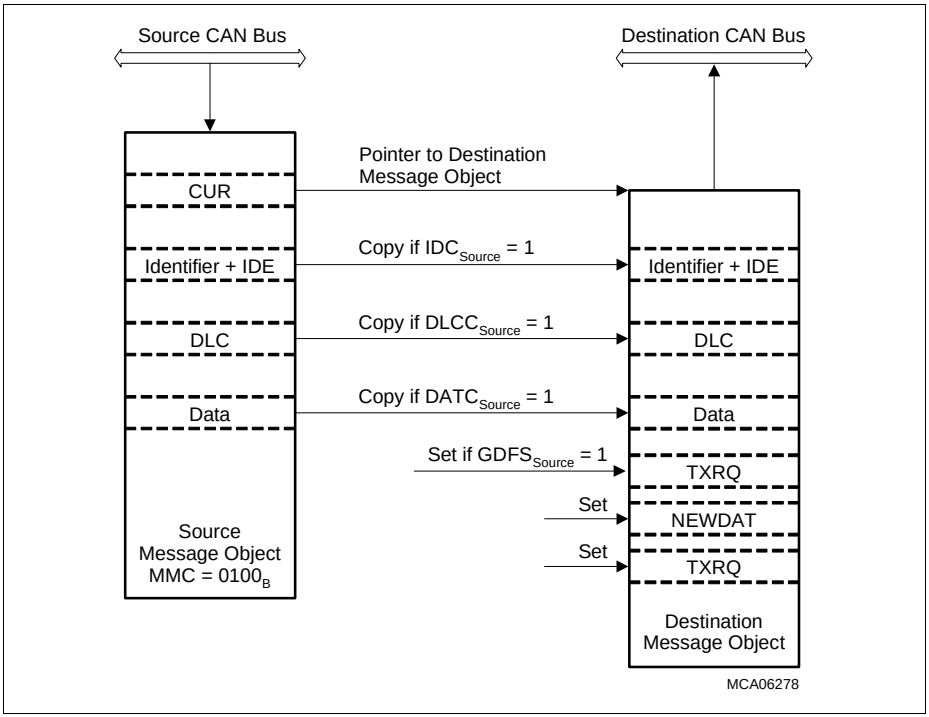


Figure 18-20 Gateway Transfer from Source to Destination

18.3.9.8 Foreign Remote Requests

When a Remote Frame has been received on a CAN node and is stored in a message object, a transmit request is set to trigger the answer (transmission of a Data Frame) to the request or to automatically issue a secondary request. If the Foreign Remote Request Enable bit MOFCRn.FRREN is cleared in the message object in which the remote request is stored, MOSTATn.TXRQ is set in the same message object.

If bit FRREN is set (FRREN = 1: foreign remote request enabled), TXRQ is set in the message object that is referenced by pointer MOFGPRn.CUR. The value of CUR is, however, not changed by this feature.

Although the foreign remote request feature works independently of the selected message mode, it is especially useful for gateways to issue a remote request on the source bus of a gateway after the reception of a remote request on the gateway destination bus. According to the setting of FRREN in the gateway destination object, there are two capabilities to handle remote requests that appear on the destination side (assuming that the source object is a receive object and the destination is a transmit object, i.e. DIR_{source} = 0 and $DIR_{destination}$ = 1):

FRREN = 0 in the Gateway Destination Object

1. A Remote Frame is received by gateway destination object.
2. TXRQ is set automatically in the gateway destination object.
3. A Data Frame with the current data stored in the destination object is transmitted on the destination bus.

FRREN = 1 in the Gateway Destination Object

1. A Remote Frame is received by gateway destination object.
2. TXRQ is set automatically in the gateway source object (must be referenced by CUR pointer of the destination object).
3. A remote request is transmitted by the source object (which is a receive object) on the source CAN bus.
4. The receiver of the remote request responds with a Data Frame on the source bus.
5. The Data Frame is stored in the source object.
6. The Data Frame is copied to the destination object (gateway action).
7. TXRQ is set in the destination object (assuming $GDFS_{source}$ = 1).
8. The new data stored in the destination object is transmitted on the destination bus, in response to the initial remote request on the destination bus.

18.4 Service Request Generation

The interrupt control logic in the MultiCAN module uses an interrupt compressing scheme that allows high flexibility in interrupt processing. There are 140 hardware interrupt sources and one software interrupt source available:

- CAN node interrupts:
  - Four different interrupt sources for each of the three CAN nodes = 12 interrupt sources
- Message object interrupts:
  - Two interrupt source for each message object = 128 interrupt sources
- One software initiated interrupt (register MITR)

Each of the 140 hardware initiated interrupt sources is controlled by a 4-bit interrupt pointer that directs the interrupt source to one of the 8 interrupt outputs INT_Om (m = 0-7). This makes it possible to connect more than one interrupt source (between one and all) to one interrupt output line. The interrupt wiring matrix shown in **Figure 18-21** is built up according to the following rules:

- Each output of the 4-bit interrupt pointer demultiplexer is connected to exactly one OR-gate input of the INT_Om line. The number “m” of the corresponding selected INT_Om interrupt output line is defined by the interrupt pointer value.
- Each INT_Om output line has an input OR gate which is connected to all interrupt pointer demultiplexer outputs which are selected by an identical 4-bit pointer value.

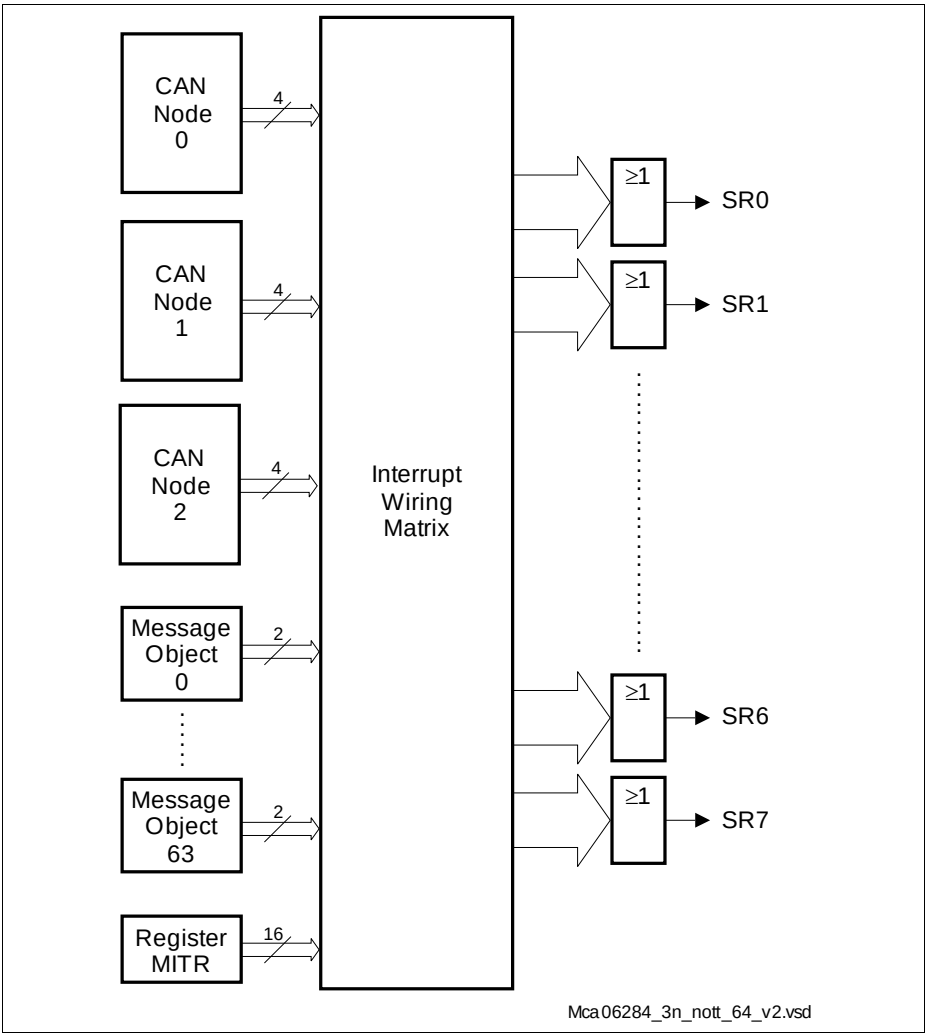


Figure 18-21 Interrupt Compressor

18.5 Debug behavior

The Suspend Mode can be triggered by the OCDS in order to freeze the state of the module and to permit access to the registers (at least for read actions). The MultiCAN module provides the following Suspend Modes:

- The current action is finished (Soft Suspend Mode):
  The module clock f_{CLC} keeps running. Module functions are stopped automatically after internal actions have been finished (for example, after a CAN frame has been sent out). The end of the internal actions is indicated to the fractional divider by a suspend mode acknowledged signal. Due to this behavior, the communication network is not blocked. Furthermore, all registers are accessible for read and write actions. As a result, the debugger can stop the module actions and modify registers. These modifications are taken into account after the Suspend Mode is left.

The Soft Suspend Mode can be individually enabled for each CAN node.

A CAN node that is not active can always be suspended. Refer to **CAN_CLC** and **CAN_FDR** registers for the corresponding OCDS control.

18.6 Power, Reset and Clock

The MultiCAN module is located in the core power domain. The module and all registers are reset to their default state by a system reset as shown on **Table 18-4** and **Table 18-13**.

18.6.1 Clock Control

The CAN module timer clock f_{CAN} of the functional blocks of the MultiCAN module is derived from the module control clock f_{CLC}. The Fractional Divider is used to generate f_{CAN} used for bit timing calculation, The frequency of f_{CAN} is identical for all CAN nodes. The register file operate with the module control clock f_{CLC}. See also **"Module Clock Generation" on Page 18-58**.

The output clock f_{CAN} of the Fractional Divider is based on the system clock f_{CLC}, but only every n-th clock pulse is taken. The suspend signal (coming as acknowledge from the MultiCAN module in response to a OCDS suspend request) freezes or resets the Fractional Divider.

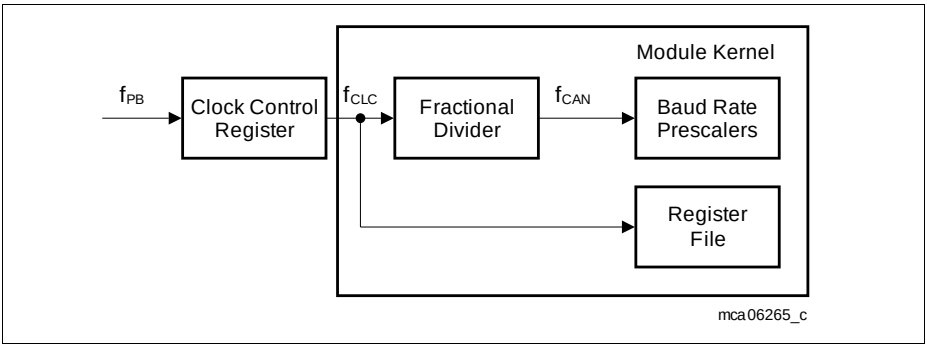


Figure 18-22 MultiCAN Clock Generation

Table 18-3 indicates the minimum operating frequencies in MHz for f_{CLC} that are required for a baud rate of 1 Mbit/s for the active CAN nodes. If a lower baud rate is desired, the values can be scaled linearly (e.g. for a maximum of 500 kbit/s, 50% of the indicated value are required).

The values imply that the CPU (or DMA) executes maximum accesses to the MultiCAN module. The values may contain rounding effects.

Table 18-3 Minimum Operating Frequencies [MHz]

Number of allocated message objects MO[1)]	1 CAN node active	2 CAN nodes active	3 CAN nodes active
16 MO	12	19	26
32 MO	15	23	30
64 MO	21	28	37

1) Only those message objects have to be taken into account that are allocated to a CAN node. The unallocated message objects have no influence on the minimum operating frequency.

The baud rate generation of the MultiCAN being based on f_{PB}, this frequency has to be chosen carefully to allow correct CAN bit timing. The required value of f_{PB} is given by an integer multiple (n) of the CAN baud rate multiplied by the number of time quanta per CAN bit time. For example, to reach 1 Mbit/s with 20 tq per bit time, possible values of f_{PB} are given by formula [n × 20] MHz, with n being an integer value, starting at 1.
In order to minimize jitter, it is not recommended to use the fractional divider mode for high baud rates.

18.6.2 Module Clock Generation

As shown in **Figure 18-23**, the clock signals for the MultiCAN module are generated and controlled by a clock control unit. This clock generation unit is responsible for the enable/disable control, the clock frequency adjustment, and the debug clock control. This unit includes two registers:

- CAN_CLC: generation of the module control clock f_{CLC}
- CAN_FDR: frequency control of the module timer clock f_{CAN}

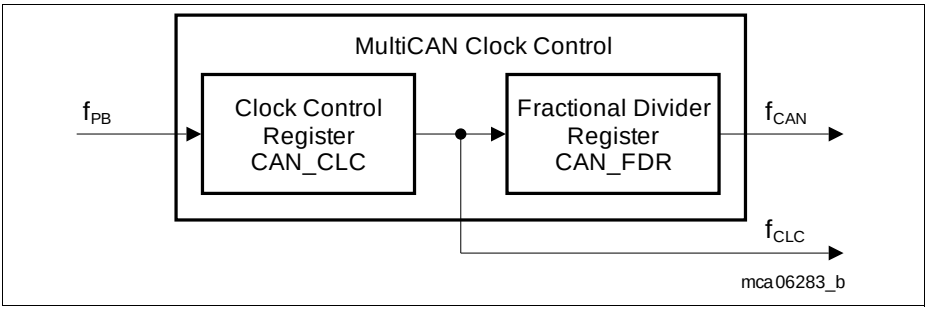


Figure 18-23 MultiCAN Module Clock Generation

The module control clock f_{CLC} is used inside the MultiCAN module for control purposes such as clocking of control logic and register operations. The frequency of f_{CLC} is identical to the system clock frequency f_{PB}. The clock control register CAN_CLC makes it possible to enable/disable f_{CLC} under certain conditions.

The module timer clock f_{CAN} is used inside the MultiCAN module as input clock for all timing relevant operations (e.g. bit timing). The settings in the CAN_FDR register determine the frequency of the module timer clock f_{CAN} according the following two formulas:

$$f_{CAN} = f_{PB} \times \frac{1}{n} \quad \text{with n = 1024 - CAN\_FDR.STEP} \tag{18.1}$$

$$f_{CAN} = f_{PB} \times \frac{n}{1024} \quad \text{with n = 0-1023} \tag{18.2}$$

Equation (18.1) applies to normal divider mode (CAN_FDR.DM = 01_B) of the fractional divider. **Equation (18.2)** applies to fractional divider mode (CAN_FDR.DM = 10_B).

Note: The CAN module is disabled after reset. In general, after reset, the module control clock f_{CLC} must be switched on (writing to register CAN_CLC) before the frequency of the module timer clock f_{CAN} is defined (writing to register CAN_FDR).

18.7 Register Description

This section describes the kernel registers of the MultiCAN module. All MultiCAN kernel register names described in this section are also referenced in other parts of the XMC4500 Reference Manual by the module name prefix “CAN_”.

MultiCAN Kernel Register Overview

The MultiCAN Kernel include three blocks of registers:

- Global Module Registers
- Node Registers, for each CAN node x
- Message Object Registers, for each message object n

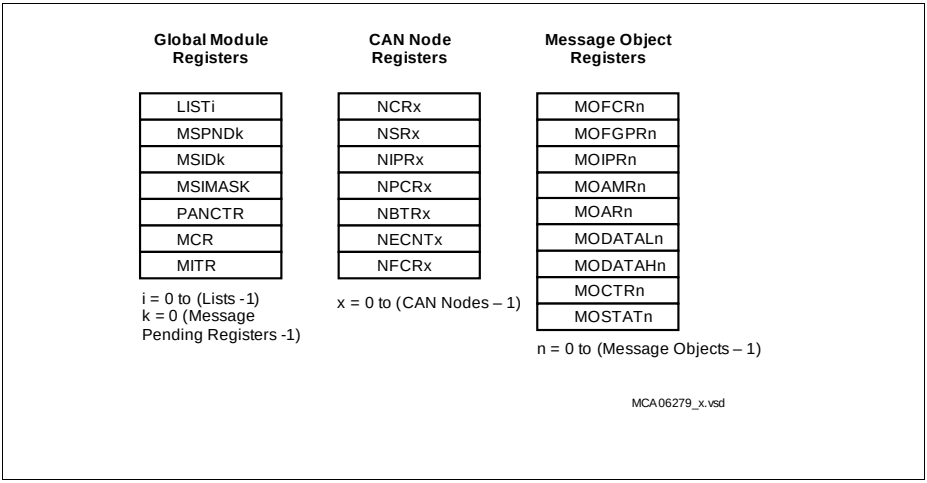


Figure 18-24 MultiCAN Kernel Registers

The registers of the MultiCAN module kernel are listed below.

Table 18-4 Registers Address Space - MultiCAN Kernel Registers

Module	Base Address	End Address	Note
CAN	$4801\ 4000_H$	$4801\ 7FFF_H$	-

Table 18-5 Registers Overview - MultiCAN Kernel Registers

Register Short Name	Register Long Name	Offset Address [1]	Access Mode [2]		Description see
			Read	Write	
Global Module Registers					
LISTi	List Register i	$0100_H + i \times 4_H$	U, PV	U, PV	Page 18-69
MSPNDk	Message Pending Register k	$0140_H + k \times 4_H$	U, PV	U, PV	Page 18-71
MSIDk	Message Index Register k	$0180_H + k \times 4_H$	U, PV	U, PV	Page 18-72
MSIMASK	Message Index Mask Register	$01C0_H$	U, PV	U, PV	Page 18-73
PANCTR	Panel Control Register	$01C4_H$	U, PV	U, PV	Page 18-63
MCR	Module Control Register	$01C8_H$	U, PV	U, PV	Page 18-67
MITR	Module Interrupt Trigger Reg.	$01CC_H$	U, PV	U, PV	Page 18-68
CAN Node Registers					
NCRx	Node x Control Register	$0200_H + x \times 100_H$	U, PV	U, PV	Page 18-74
NSRx	Node x Status Register	$0204_H + x \times 100_H$	U, PV	U, PV	Page 18-78
NIPRx	Node x Interrupt Pointer Reg.	$0208_H + x \times 100_H$	U, PV	U, PV	Page 18-82
NPCRx	Node x Port Control Register	$020C_H + x \times 100_H$	U, PV	U, PV	Page 18-84
NBTRx	Node x Bit Timing Register	$0210_H + x \times 100_H$	U, PV	U, PV	Page 18-85
NECNTx	Node x Error Counter Register	$0214_H + x \times 100_H$	U, PV	U, PV	Page 18-87
NFCRx	Node x Frame Counter Register	$0218_H + x \times 100_H$	U, PV	U, PV	Page 18-89
Message Object Registers					
MOFCRn	Message Object n Function Control Register	$1000_H + n \times 20_H$	U, PV	U, PV	Page 18-103

Table 18-5 Registers Overview - MultiCAN Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Address [1]	Access Mode [2]		Description see
			Read	Write	
MOFGPRn	Message Object n FIFO/Gateway Pointer Register	$1004_H + n \times 20_H$	U, PV	U, PV	Page 18-107
MOIPRn	Message Object n Interrupt Pointer Register	$1008_H + n \times 20_H$	U, PV	U, PV	Page 18-101
MOAMRn	Message Object n Acceptance Mask Register	$100C_H + n \times 20_H$	U, PV	U, PV	Page 18-108
MODATALn	Message Object n Data Register Low	$1010_H + n \times 20_H$	U, PV	U, PV	Page 18-112
MODATAHn	Message Object n Data Register High	$1014_H + n \times 20_H$	U, PV	U, PV	Page 18-113
MOARn	Message Object n Arbitration Register	$1018_H + n \times 20_H$	U, PV	U, PV	Page 18-109
MOCTRn MOSTATn	Message Object n Control Reg. Message Object n Status Reg.	$101C_H + n \times 20_H$	U, PV	U, PV	Page 18-93 Page 18-96

1) The absolute register address is calculated as follows:
Module Base Address (**Table 18-4**) + Offset Address (shown in this column)
Further, the following ranges for parameters i, k, x, and n are valid: i = 0-7, k = 0-7, x = 0-2, n = 0-63.

2) Accesses to empty addresses: nBE

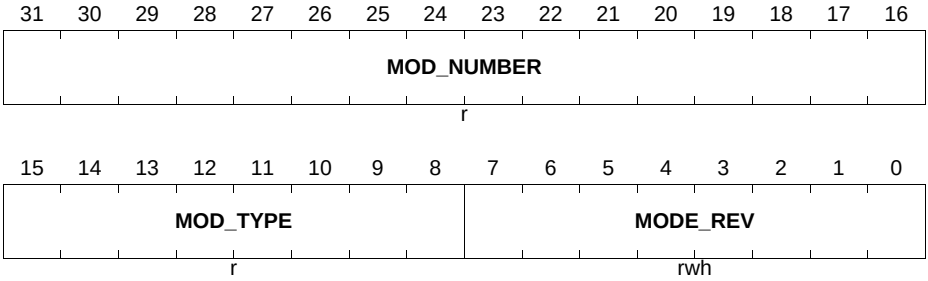
18.7.1 Global Module Registers

All list operations such as allocation, de-allocation and relocation of message objects within the list structure are performed via the Command Panel. It is not possible to modify the list structure directly by software by writing to the message objects and the LIST registers.

ID

The ID (Module Identification Register) defines the MultiCAN module identification number, module type and module revision number.

ID
Module Identification Register (008$_H$) Reset Value: 002B C0XX$_H$



Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** MOD_REV defines the revision number. The value of a module revision starts with 01H (first revision).
MOD_TYPE	[15:8]	r	**Module Type** C0$_H$ Define the module as a 32-bit module.
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines the MultiCAN module identification number (=002BH)

PANCTR

The Panel Control Register PANCTR is used to start a new command by writing the command arguments and the command code into its bit fields.

PANCTR
Panel Control Register **($1C4_H$)** **Reset Value: $0000\ 0301_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PANAR2								PANAR1							
rwh								rwh							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						RBUSY	BUSY	PANCMD							
r						rh	rh	rwh							

Field	Bits	Type	Description
PANCMD	[7:0]	rwh	**Panel Command** This bit field is used to start a new command by writing a panel command code into it. At the end of a panel command, the NOP (no operation) command code is automatically written into PANCMD. The coding of PANCMD is defined in **Table 18-6**.
BUSY	8	rh	**Panel Busy Flag** 0_B Panel has finished command and is ready to accept a new command. 1_B Panel operation is in progress.
RBUSY	9	rh	**Result Busy Flag** 0_B No update of PANAR1 and PANAR2 is scheduled by the list controller. 1_B A list command is running (BUSY = 1) that will write results to PANAR1 and PANAR2, but the results are not yet available.
PANAR1	[23:16]	rwh	**Panel Argument 1** See **Table 18-6**.
PANAR2	[31:24]	rwh	**Panel Argument 2** See **Table 18-6**.

Field	Bits	Type	Description
0	[15:10]	r	**Reserved** Read as 0; should be written with 0.

Panel Commands

A panel operation consists of a command code (PANCMD) and up to two panel arguments (PANAR1, PANAR2). Commands that have a return value deliver it to the PANAR1 bit field. Commands that return an error flag deliver it to bit 31 of the Panel Control Register, this means bit 7 of PANAR2.

Table 18-6 Panel Commands

PANCMD	PANAR2	PANAR1	Command Description
00_H	–	–	**No Operation** Writing 00_H to PANCMD has no effect. No new command is started.
01_H	**Result:** Bit 7: ERR Bit 6-0: undefined	–	**Initialize Lists** Run the initialization sequence to reset the CTRL and LIST fields of all message objects. List registers LIST[7:0] are set to their reset values. This results in the de-allocation of all message objects. The initialization command requires that bits NCRx.INIT and NCRx.CCE are set for all CAN nodes. Bit 7 of PANAR2 (ERR) reports the success of the operation: 0_B Initialization was successful 1_B Not all NCRx.INIT and NCRx.CCE bits are set. Therefore, no initialization is performed. The initialize lists command is automatically performed with each reset of the MultiCAN module, but with the exception that all message object registers are reset, too.

Table 18-6 Panel Commands (cont'd)

PANCMD	PANAR2	PANAR1	Command Description
02_H	**Argument:** List Index	**Argument:** Message Object Number	**Static Allocate** Allocate message object to a list. The message object is removed from the list that it currently belongs to, and appended to the end of the list, given by PANAR2. This command is also used to deallocate a message object. In this case, the target list is the list of unallocated elements (PANAR2 = 0).
03_H	**Argument:** List Index **Result:** Bit 7: ERR Bit 6-0: undefined	**Result:** Message Object Number	**Dynamic Allocate** Allocate the first message object of the list of unallocated objects to the selected list. The message object is appended to the end of the list. The message number of the message object is returned in PANAR1. An ERR bit (bit 7 of PANAR2) reports the success of the operation: 0_B Success. 1_B The operation has not been performed because the list of unallocated elements was empty.
04_H	**Argument:** Destination Object Number	**Argument:** Source Object Number	**Static Insert Before** Remove a message object (source object) from the list that it currently belongs to, and insert it before a given destination object into the list structure of the destination object. The source object thus becomes the predecessor of the destination object.

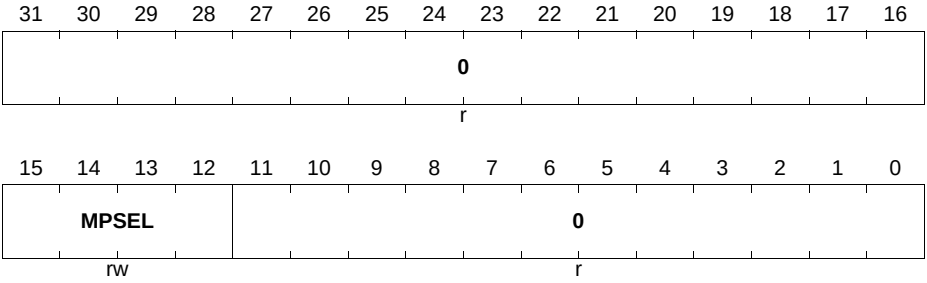
Table 18-6 Panel Commands (cont'd)

PANCMD	PANAR2	PANAR1	Command Description
05_H	**Argument:** Destination Object Number **Result:** Bit 7: ERR Bit 6-0: undefined	**Result:** Object Number of inserted object	**Dynamic Insert Before** Insert a new message object before a given destination object. The new object is taken from the list of unallocated elements (the first element is chosen). The number of the new object is delivered as a result to PANAR1. An ERR bit (bit 7 of PANAR2) reports the success of the operation: 0_B Success. 1_B The operation has not been performed because the list of unallocated elements was empty.
06_H	**Argument:** Destination Object Number	**Argument:** Source Object Number	**Static Insert Behind** Remove a message object (source object) from the list that it currently belongs to, and insert it behind a given destination object into the list structure of the destination object. The source object thus becomes the successor of the destination object.
07_H	**Argument:** Destination Object Number **Result:** Bit 7: ERR Bit 6-0: undefined	**Result:** Object Number of inserted object	**Dynamic Insert Behind** Insert a new message object behind a given destination object. The new object is taken from the list of unallocated elements (the first element is chosen). The number of the new object is delivered as result to PANAR1. An ERR bit (bit 7 of PANAR2) reports the success of the operation: 0_B Success. 1_B The operation has not been performed because the list of unallocated elements was empty.
08_H - FF_H	–	–	**Reserved**

MCR

The Module Control Register MCR contains basic settings that determine the operation of the MultiCAN module.

MCR
Module Control Register **($1C8_H$)** **Reset Value: $0000\ 0000_H$**

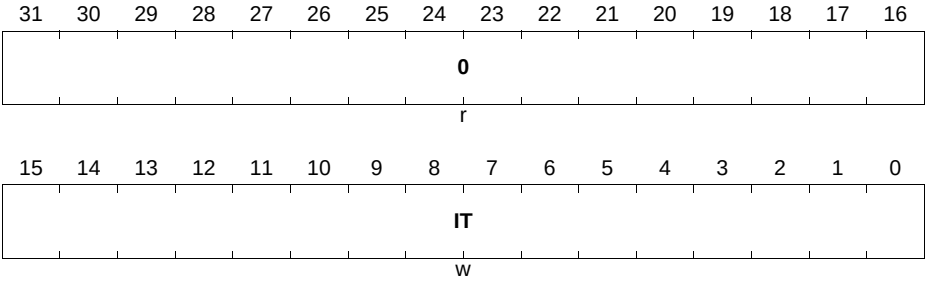


Field	Bits	Type	Description
MPSEL	[15:12]	rw	**Message Pending Selector** Bit field MPSEL makes it possible to select the bit position of the message pending bit after a message reception/transmission by a mixture of the MOIPRn register bit fields RXINP, TXINP, and MPN. Selection details are given in **Figure 18-16** on **Page 18-37**.
0	[31:16], [11:0]	r	**Reserved** Read as 0; should be written with 0.

MITR

The Interrupt Trigger Register ITR is used to trigger interrupt requests on each interrupt output line by software.

MITR
Module Interrupt Trigger Register (1CC$_H$) **Reset Value: 0000 0000$_H$**



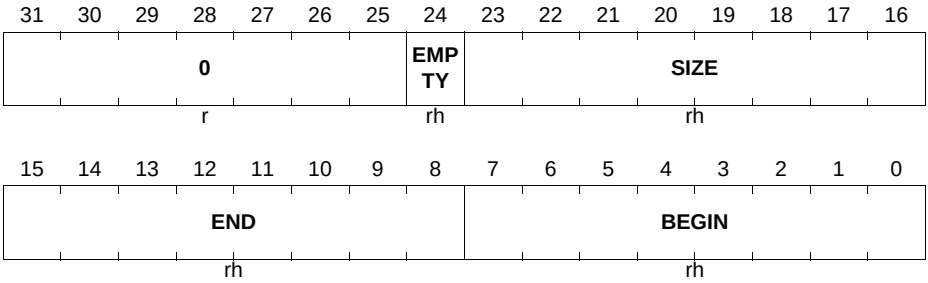
Field	Bits	Type	Description
IT	[7:0]	w	**Interrupt Trigger** Writing a 1 to IT[n] (n = 0-7) generates an interrupt request on interrupt output line INT_O[n]. Writing a 0 to IT[n] has no effect. Bit field IT is always read as 0. Multiple interrupt requests can be generated with a single write operation to MITR by writing a 1 to several bit positions of IT.
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

LIST

Each CAN node has a list that determines the allocated message objects. Additionally, a list of all unallocated objects is available. Furthermore, general purpose lists are available which are not associated to a CAN node. The List Registers are assigned in the following way:

- LIST0 provides the list of all unallocated objects
- LIST1 provides the list for CAN node 0
- LIST2 provides the list for CAN node 1
- LIST3 provides the list for CAN node 2
- LIST[7:4] are not associated to a CAN node (free lists)

LIST0
List Register 0 **(100_H)** **Reset Value: 003F 3F00$_H$**
LISTx (x = 1-7)
List Register x **(100_H+x*4_H)** **Reset Value: 0100 0000$_H$**



Field	Bits	Type	Description
BEGIN	[7:0]	rh	**List Begin** BEGIN indicates the number of the first message object in list i.
END	[15:8]	rh	**List End** END indicates the number of the last message object in list i.
SIZE	[23:16]	rh	**List Size** SIZE indicates the number of elements in the list i. SIZE = number of list elements - 1 SIZE = 0 indicates that list x is empty.

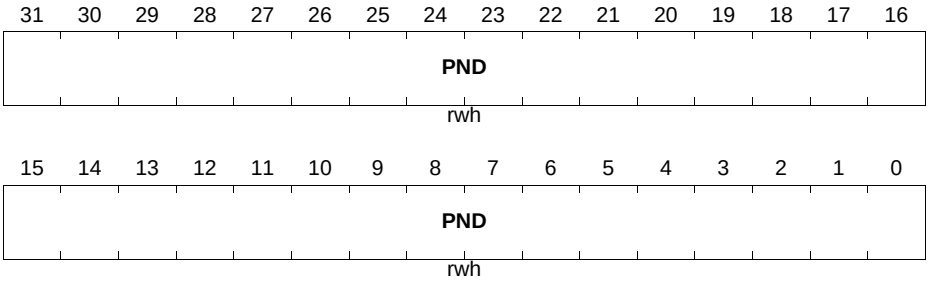
Field	Bits	Type	Description
EMPTY	24	rh	**List Empty Indication** 0_B At least one message object is allocated to list i. 1_B No message object is allocated to the list x. List x is empty.
0	[31:25]	r	**Reserved** Read as 0.

MSPNDk

When a message object n generates an interrupt request upon the transmission or reception of a message, then the request is routed to the interrupt output line selected by the bit field MOIPRn.TXINP or MOIPRn.RXINP of the message object n. As there are more message objects than interrupt output lines, an interrupt routine typically processes requests from more than one message object. Therefore, a priority selection mechanism is implemented in the MultiCAN module to select the highest priority object within a collection of message objects.

The Message Pending Register MSPNDk contains the pending interrupt notification of list i.

MSPNDk (k = 0-7)
Message Pending Register k **(140_H+k*4_H)** **Reset Value: 0000 0000_H**

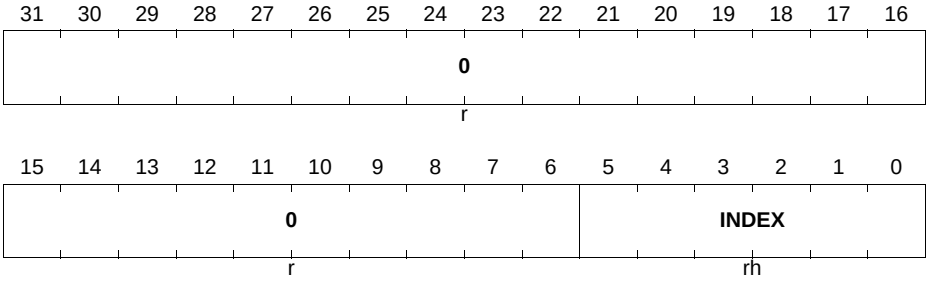


Field	Bits	Type	Description
PND	[31:0]	rwh	**Message Pending** When a message interrupt occurs, the message object sets a bit in one of the MSPND register, where the bit position is given by the MPN[4:0] field of the IPR register of the message object. The register selection n is given by the higher bits of MPN. The register bits can be cleared by software (write 0). Writing a 1 has no effect.

MSIDk

Each Message Pending Register has a Message Index Register MSIDk associated with it. The Message Index Register shows the active (set) pending bit with lowest bit position within groups of pending bits.

MSIDk (k = 0-7)
Message Index Register k **(180_H+k*4_H)** **Reset Value: 0000 0020$_H$**

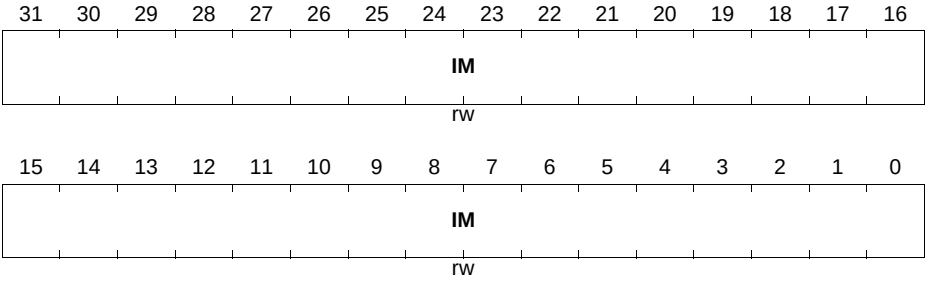


Field	Bits	Type	Description
INDEX	[5:0]	rh	**Message Pending Index** The value of INDEX is given by the bit position i of the pending bit of MSPNDk with the following properties: 1. MSPNDk[i] & IM[i] = 1 2. i = 0 or MSPNDk[i-1:0] & IM[i-1:0] = 0 If no bit of MSPNDk satisfies these conditions then INDEX reads 100000_B. Thus INDEX shows the position of the first pending bit of MSPNDk, in which only those bits of MSPNDk that are selected in the Message Index Mask Register are taken into account.
0	[31:6]	r	**Reserved** Read as 0; should be written with 0.

MSIMASK

The Message Index Mask Register MSIMASK selects individual bits for the calculation of the Message Pending Index. The Message Index Mask Register is used commonly for all Message Pending registers and their associated Message Index registers.

MSIMASK
Message Index Mask Register **($1C0_H$)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
IM	[31:0]	rw	**Message Index Mask** Only those bits in MSPNDk for which the corresponding Index Mask bits are set contribute to the calculation of the Message Index.

18.7.2 CAN Node Registers

The CAN node registers are built in for each CAN node of the MultiCAN module. They contain information that is directly related to the operation of the CAN nodes and are shared among the nodes.

NCR

The Node Control Register contains basic settings that determine the operation of the CAN node.

NCRx (x = 0-2)
Node x Control Register **(200_H+x*100_H)** **Reset Value: 0000 0001_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							SUSEN	CALM	CCE	0	CANDIS	ALIE	LECIE	TRIE	INIT
r							rw	rw	rw	r	rw	rw	rw	rw	rwh

Field	Bits	Type	Description
INIT	0	rwh	**Node Initialization** 0_B Resetting bit INIT enables the participation of the node in the CAN traffic. If the CAN node is in the bus-off state, the ongoing bus-off recovery (which does not depend on the INIT bit) is continued. With the end of the bus-off recovery sequence the CAN node is allowed to take part in the CAN traffic. If the CAN node is not in the bus-off state, a sequence of 11 consecutive recessive bits must be detected before the node is allowed to take part in the CAN traffic. 1_B Setting this bit terminates the participation of this node in the CAN traffic. Any ongoing frame transfer is cancelled and the transmit line goes recessive. If the CAN node is in the bus-off state, then the running bus-off recovery sequence is continued. If the INIT bit is still set after the successful completion of the bus-off recovery sequence, i.e. after detecting 128 sequences of 11 consecutive recessive bits (11×1), then the CAN node leaves the bus-off state but remains inactive as long as INIT remains set. Bit INIT is automatically set when the CAN node enters the bus-off state (see **Page 18-20**).
TRIE	1	rw	**Transfer Interrupt Enable** TRIE enables the transfer interrupt of CAN node x. This interrupt is generated after the successful reception or transmission of a CAN frame in node x. 0_B Transfer interrupt is disabled. 1_B Transfer interrupt is enabled. Bit field NIPRx.TRINP selects the interrupt output line which becomes activated at this type of interrupt.

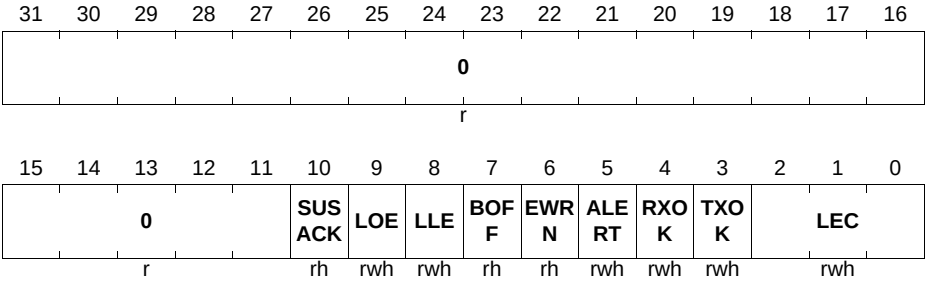
Field	Bits	Type	Description
LECIE	2	rw	**LEC Indicated Error Interrupt Enable** LECIE enables the last error code interrupt of CAN node x. This interrupt is generated with each update of bit field NSRx.LEC with LEC > 0 (CAN protocol error). 0_B Last error code interrupt is disabled. 1_B Last error code interrupt is enabled. Bit field NIPRx.LECINP selects the interrupt output line which becomes activated at this type of interrupt.
ALIE	3	rw	**Alert Interrupt Enable** ALIE enables the alert interrupt of CAN node x. This interrupt is generated by any one of the following events: • A change of bit NSRx.BOFF • A change of bit NSRx.EWRN • A List Length Error, which also sets bit NSRx.LLE • A List Object Error, which also sets bit NSRx.LOE • A Bit INIT is set by hardware 0_B Alert interrupt is disabled. 1_B Alert interrupt is enabled. Bit field NIPRx.ALINP selects the interrupt output line which becomes activated at this type of interrupt.
CANDIS	4	rw	**CAN Disable** Setting this bit disables the CAN node. The CAN node first waits until it is bus-idle or bus-off. Then bit INIT is automatically set, and an alert interrupt is generated if bit ALIE is set.
CCE	6	rw	**Configuration Change Enable** 0_B The Bit Timing Register, the Port Control Register, and the Error Counter Register may only be read. All attempts to modify them are ignored. 1_B The Bit Timing Register, the Port Control Register, and the Error Counter Register may be read and written.
CALM	7	rw	**CAN Analyzer Mode** If this bit is set, then the CAN node operates in Analyzer Mode. This means that messages may be received, but not transmitted. No acknowledge is sent on the CAN bus upon frame reception. Active-error flags are sent recessive instead of dominant. The transmit line is continuously held at recessive (1) level. Bit CALM can be written only while bit INIT is set.

Field	Bits	Type	Description
SUSEN	8	rw	**Suspend Enable** This bit makes it possible to set the CAN node into Suspend Mode via OCDS (on chip debug support): 0_B An OCDS suspend trigger is ignored by the CAN node. 1_B An OCDS suspend trigger disables the CAN node: As soon as the CAN node becomes bus-idle or bus-off, bit INIT is internally forced to 1 to disable the CAN node. The actual value of bit INIT remains unchanged. Bit SUSEN is reset via OCDS Reset.
0	[31:9], 5	r	**Reserved** Read as 0; should be written with 0.

NSR

The Node Status Register NSRx reports errors as well as successfully transferred CAN frames.

NSRx (x = 0-2)
Node x Status Register **(204_H+x*100_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
LEC	[2:0]	rwh	**Last Error Code** This bit field indicates the type of the last (most recent) CAN error. The encoding of this bit field is described in **Table 18-7**.
TXOK	3	rwh	**Message Transmitted Successfully** 0_B No successful transmission since last (most recent) flag reset. 1_B A message has been transmitted successfully (error-free and acknowledged by at least another node). TXOK must be reset by software (write 0). Writing 1 has no effect.
RXOK	4	rwh	**Message Received Successfully** 0_B No successful reception since last (most recent) flag reset. 1_B A message has been received successfully. RXOK must be reset by software (write 0). Writing 1 has no effect.

Field	Bits	Type	Description
ALERT	5	rwh	**Alert Warning** The ALERT bit is set upon the occurrence of one of the following events (the same events which also trigger an alert interrupt if ALIE is set): • A change of bit NSRx.BOFF • A change of bit NSRx.EWRN • A List Length Error, which also sets bit NSRx.LLE • A List Object Error, which also sets bit NSRx.LOE • Bit INIT has been set by hardware ALERT must be reset by software (write 0). Writing 1 has no effect.
EWRN	6	rh	**Error Warning Status** 0_B No warning limit exceeded. 1_B One of the error counters REC or TEC reached the warning limit EWRNLVL.
BOFF	7	rh	**Bus-off Status** 0_B CAN controller is not in the bus-off state. 1_B CAN controller is in the bus-off state.
LLE	8	rwh	**List Length Error** 0_B No List Length Error since last (most recent) flag reset. 1_B A List Length Error has been detected during message acceptance filtering. The number of elements in the list that belongs to this CAN node differs from the list SIZE given in the list termination pointer. LLE must be reset by software (write 0). Writing 1 has no effect.
LOE	9	rwh	**List Object Error** 0_B No List Object Error since last (most recent) flag reset. 1_B A List Object Error has been detected during message acceptance filtering. A message object with wrong LIST index entry in the Message Object Control Register has been detected. LOE must be reset by software (write 0). Writing 1 has no effect.

Field	Bits	Type	Description
SUSACK	10	rh	**Suspend Acknowledge** 0_B The CAN node is not in Suspend Mode or a suspend request is pending, but the CAN node has not yet reached bus-idle or bus-off. 1_B The CAN node is in Suspend Mode: The CAN node is inactive (bit NCR.INIT internally forced to 1) due to an OCDS suspend request.
0	[31:11]	r	**Reserved** Read as 0; should be written with 0.

Encoding of the LEC Bit Field

Table 18-7 Encoding of the LEC Bit Field

LEC Value	Signification
000_B	**No Error:** No error was detected for the last (most recent) message on the CAN bus.
001_B	**Stuff Error:** More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.
010_B	**Form Error:** A fixed format part of a received frame has the wrong format.
011_B	**Ack Error:** The transmitted message was not acknowledged by another node.
100_B	**Bit1 Error:** During a message transmission, the CAN node tried to send a recessive level (1) outside the arbitration field and the acknowledge slot, but the monitored bus value was dominant.
101_B	**Bit0 Error:** Two different conditions are signaled by this code: 1. During transmission of a message (or acknowledge bit, active-error flag, overload flag), the CAN node tried to send a dominant level (0), but the monitored bus value was recessive. 2. During bus-off recovery, this code is set each time a sequence of 11 recessive bits has been monitored. The CPU may use this code as indication that the bus is not continuously disturbed.

Table 18-7 Encoding of the LEC Bit Field (cont'd)

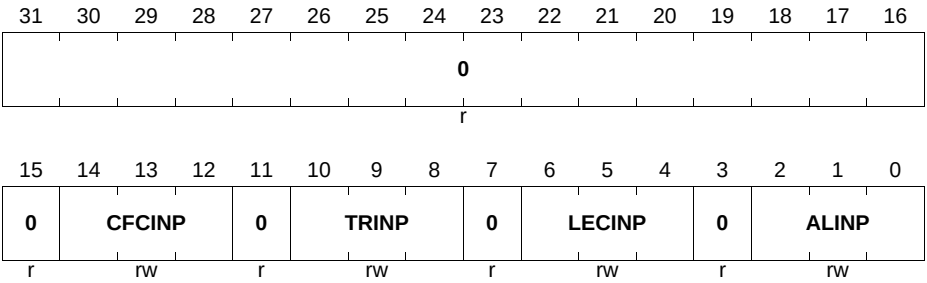
LEC Value	Signification
110_B	**CRC Error:** The CRC checksum of the received message was incorrect.
111_B	**CPU write to LEC:** Whenever the the CPU writes the value 111 to LEC, it takes the value 111. Whenever the CPU writes another value to LEC, the written LEC value is ignored.

NIPR

The four interrupt pointers in the Node Interrupt Pointer Register NIPRx select one out of the sixteen interrupt outputs individually for each type of CAN node interrupt. See also **Page 18-21** for more CAN node interrupt details.

NIPRx (x = 0-2)
Node x Interrupt Pointer Register

(208_H+x*100_H) **Reset Value: 0000 0000_H**



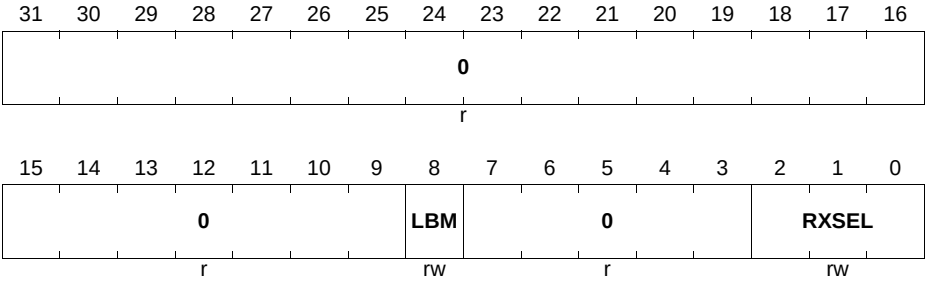
Field	Bits	Type	Description
ALINP	[2:0]	rw	**Alert Interrupt Node Pointer** ALINP selects the interrupt output line INT_Om (m = 0-7) for an alert interrupt of CAN Node x. 000_B Interrupt output line INT_O0 is selected. 001_B Interrupt output line INT_O1 is selected. $…_B$ … 111_B Interrupt output line INT_O7 is selected.
LECINP	[6:4]	rw	**Last Error Code Interrupt Node Pointer** LECINP selects the interrupt output line INT_Om (m = 0-7) for an LEC interrupt of CAN Node x. 000_B Interrupt output line INT_O0 is selected. 001_B Interrupt output line INT_O1 is selected. $…_B$ … 111_B Interrupt output line INT_O7 is selected.

Field	Bits	Type	Description
TRINP	[10:8]	rw	**Transfer OK Interrupt Node Pointer** TRINP selects the interrupt output line INT_Om (m = 0-7) for a transfer OK interrupt of CAN Node x. 000_B Interrupt output line INT_O0 is selected. 001_B Interrupt output line INT_O1 is selected. $\ldots_B$ … 111_B Interrupt output line INT_O7 is selected.
CFCINP	[14:12]	rw	**Frame Counter Interrupt Node Pointer** CFCINP selects the interrupt output line INT_Om (m = 0-7) for a transfer OK interrupt of CAN Node x. 000_B Interrupt output line INT_O0 is selected. 001_B Interrupt output line INT_O1 is selected. $\ldots_B$ … 111_B Interrupt output line INT_O7 is selected.
0	[31:15], 11, 7, 3	r	**Reserved** Read as 0; should be written with 0.

NPCR

The Node Port Control Register NPCRx configures the CAN bus transmit/receive ports. NPCRx can be written only if bit NCRx.CCE is set.

NPCRx (x = 0-2)
Node x Port Control Register ($20C_H+x*100_H$) Reset Value: 0000 0000$_H$

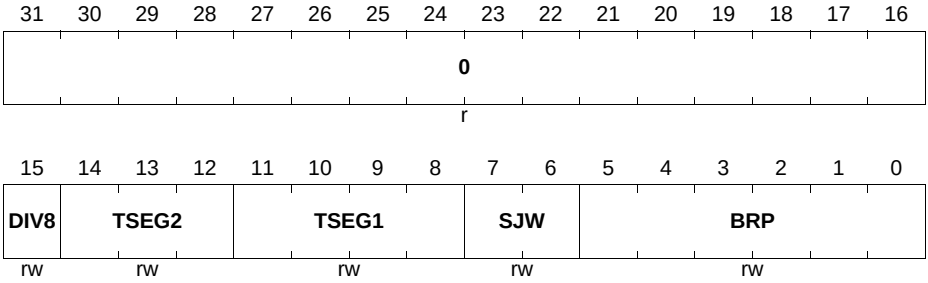


Field	Bits	Type	Description
RXSEL	[2:0]	rw	**Receive Select** RXSEL selects one out of 8 possible receive inputs. The CAN receive signal is performed only through the selected input. *Note: In XMC4500, only specific combinations of RXSEL are available (see also **"MultiCAN I/O Control Selection and Setup" on Page 18-122**).*
LBM	8	rw	**Loop-Back Mode** 0_B Loop-Back Mode is disabled. 1_B Loop-Back Mode is enabled. This node is connected to an internal (virtual) loop-back CAN bus. All CAN nodes which are in Loop-Back Mode are connected to this virtual CAN bus so that they can communicate with each other internally. The external transmit line is forced recessive in Loop-Back Mode.
0	[7:3], [31:9]	r	**Reserved** Read as 0; should be written with 0.

NBTR

The Node Bit Timing Register NBTRx contains all parameters to set up the bit timing for the CAN transfer. NBTRx can be written only if bit NCRx.CCE is set.

NBTRx (x = 0-2)
Node x Bit Timing Register **(210_H+x*100_H)** **Reset Value: 0000 0000_H**



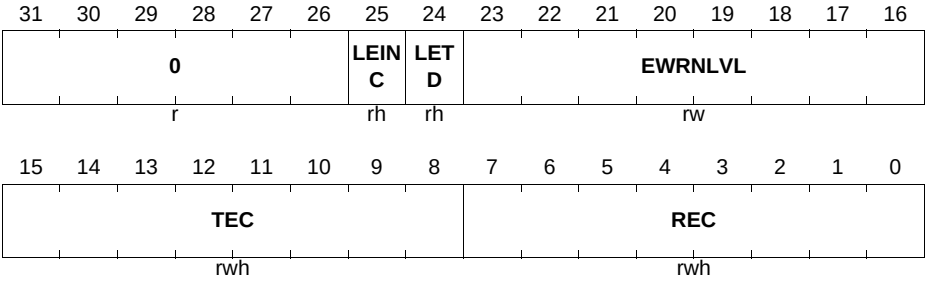
Field	Bits	Type	Description
BRP	[5:0]	rw	**Baud Rate Prescaler** The duration of one time quantum is given by (BRP + 1) clock cycles if DIV8 = 0. The duration of one time quantum is given by 8 × (BRP + 1) clock cycles if DIV8 = 1.
SJW	[7:6]	rw	**(Re) Synchronization Jump Width** (SJW + 1) time quanta are allowed for re-synchronization.
TSEG1	[11:8]	rw	**Time Segment Before Sample Point** (TSEG1 + 1) time quanta is the user-defined nominal time between the end of the synchronization segment and the sample point. It includes the propagation segment, which takes into account signal propagation delays. The time segment may be lengthened due to re-synchronization. Valid values for TSEG1 are 2 to 15.
TSEG2	[14:12]	rw	**Time Segment After Sample Point** (TSEG2 + 1) time quanta is the user-defined nominal time between the sample point and the start of the next synchronization segment. It may be shortened due to re-synchronization. Valid values for TSEG2 are 1 to 7.

Field	Bits	Type	Description
DIV8	15	rw	**Divide Prescaler Clock by 8** 0_B A time quantum lasts (BRP+1) clock cycles. 1_B A time quantum lasts 8 × (BRP+1) clock cycles.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

NECNT

The Node Error Counter Register NECNTx contains the CAN receive and transmit error counter as well as some additional bits to ease error analysis. NECNTx can be written only if bit NCRx.CCE is set.

NECNTx (x = 0-2)
Node x Error Counter Register (214_H+x*100_H) Reset Value: 0060 0000_H



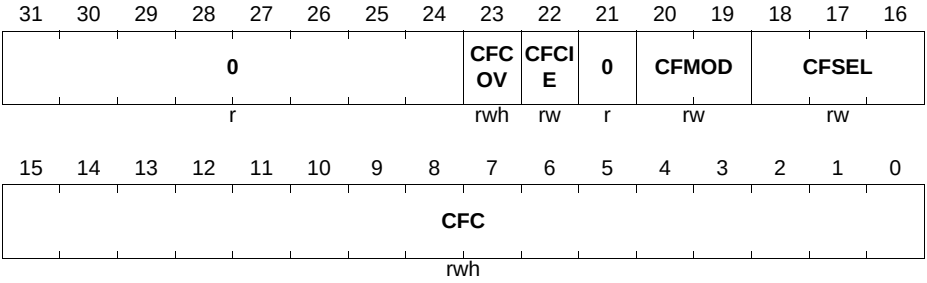
Field	Bits	Type	Description
REC	[7:0]	rwh	**Receive Error Counter** Bit field REC contains the value of the receive error counter of CAN node x.
TEC	[15:8]	rwh	**Transmit Error Counter** Bit field TEC contains the value of the transmit error counter of CAN node x.
EWRNLVL	[23:16]	rw	**Error Warning Level** Bit field EWRNLVL determines the threshold value (warning level, default 96) to be reached in order to set the corresponding error warning bit EWRN.
LETD	24	rh	**Last Error Transfer Direction** 0_B The last error occurred while the CAN node x was receiver (REC has been incremented). 1_B The last error occurred while the CAN node x was transmitter (TEC has been incremented).
LEINC	25	rh	**Last Error Increment** 0_B The last error led to an error counter increment of 1. 1_B The last error led to an error counter increment of 8.

Field	Bits	Type	Description
0	[31:26]	r	**Reserved** Read as 0; should be written with 0.

NFCR

The Node Frame Counter Register NFCRx contains the actual value of the frame counter as well as control and status bits of the frame counter.

NFCRx (x = 0-2)
Node x Frame Counter Register (218_H+x*100_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
CFC	[15:0]	rwh	**CAN Frame Counter** In Frame Count Mode (CFMOD = 00_B), this bit field contains the frame count value. In Time Stamp Mode (CFMOD = 01_B), this bit field contains the captured bit time count value, captured with the start of a new frame. In all Bit Timing Analysis Modes (CFMOD = 10_B), CFC always displays the number of f_{CLC} clock cycles (measurement result) minus 1. Example: a CFC value of 34 in measurement mode CFSEL = 000_B means that 35 f_{CLC} clock cycles have been elapsed between the most recent two dominant edges on the receive input.

Field	Bits	Type	Description
CFSEL	[18:16]	rw	**CAN Frame Count Selection** This bit field selects the function of the frame counter for the chosen frame count mode. **Frame Count Mode** Bit 0 If Bit 0 of CFSEL is set, then CFC is incremented each time a foreign frame (i.e. a frame not matching to a message object) has been received on the CAN bus. Bit 1 If Bit 1 of CFSEL is set, then CFC is incremented each time a frame matching to a message object has been received on the CAN bus. Bit 2 If Bit 2 of CFSEL is set, then CFC is incremented each time a frame has been transmitted successfully by the node. **Time Stamp Mode** 000_B The frame counter is incremented (internally) at the beginning of a new bit time. The value is sampled during the SOF bit of a new frame. The sampled value is visible in the CFC field. **Bit Timing Mode** The available bit timing measurement modes are shown in **Table 18-8**. If CFCIE is set, then an interrupt on request node x (where x is the CAN node number) is generated with a CFC update.
CFMOD	[20:19]	rw	**CAN Frame Counter Mode** This bit field determines the operation mode of the frame counter. 00_B Frame Count Mode: The frame counter is incremented upon the reception and transmission of frames. 01_B Time Stamp Mode: The frame counter is used to count bit times. 10_B Bit Timing Mode: The frame counter is used for analysis of the bit timing.

Field	Bits	Type	Description
CFCIE	22	rw	**CAN Frame Count Interrupt Enable** CFCIE enables the CAN frame counter overflow interrupt of CAN node x. 0_B CAN frame counter overflow interrupt is disabled. 1_B CAN frame counter overflow interrupt is enabled. Bit field NIPRx.CFCINP selects the interrupt output line that is activated at this type of interrupt.
CFCOV	23	rwh	**CAN Frame Counter Overflow Flag** Flag CFCOV is set upon a frame counter overflow (transition from $FFFF_H$ to 0000_H). In bit timing analysis mode, CFCOV is set upon an update of CFC. An interrupt request is generated if CFCIE = 1. 0_B No overflow has occurred since last flag reset. 1_B An overflow has occurred since last flag reset. CFCOV must be reset by software.
0	21, [31:24]	r	**Reserved** Read as 0; should be written with 0.

Bit Timing Analysis Modes

Table 18-8 Bit Timing Analysis Modes (CFMOD = 10)

CFSEL	Measurement
000_B	Whenever a dominant edge (transition from 1 to 0) is monitored on the receive input, the time (measured in clock cycles) between this edge and the most recent dominant edge is stored in CFC.
001_B	Whenever a recessive edge (transition from 0 to 1) is monitored on the receive input the time (measured in clock cycles) between this edge and the most recent dominant edge is stored in CFC.
010_B	Whenever a dominant edge is received as a result of a transmitted dominant edge, the time (clock cycles) between both edges is stored in CFC.
011_B	Whenever a recessive edge is received as a result of a transmitted recessive edge, the time (clock cycles) between both edges is stored in CFC.
100_B	Whenever a dominant edge that qualifies for synchronization is monitored on the receive input, the time (measured in clock cycles) between this edge and the most recent sample point is stored in CFC.

Table 18-8 Bit Timing Analysis Modes (CFMOD = 10) (cont'd)

CFSEL	Measurement
101_B	With each sample point, the time (measured in clock cycles) between the start of the new bit time and the start of the previous bit time is stored in CFC[11:0]. Additional information is written to CFC[15:12] at each sample point: CFC[15]: Transmit value of actual bit time CFC[14]: Receive sample value of actual bit time CFC[13:12]: CAN bus information (see **Table 18-9**)
110_B	Reserved, do not use this combination.
111_B	Reserved, do not use this combination.

Table 18-9 CAN Bus State Information

CFC[13:12]	CAN Bus State
00_B	**NoBit** The CAN bus is idle, performs bit (de-) stuffing or is in one of the following frame segments: SOF, SRR, CRC, delimiters, first 6 EOF bits, IFS.
01_B	**NewBit** This code represents the first bit of a new frame segment. The current bit is the first bit in one of the following frame segments: Bit 10 (MSB) of standard ID (transmit only), RTR, reserved bits, IDE, DLC(MSB), bit 7 (MSB) in each data byte and the first bit of the ID extension.
10_B	**Bit** This code represents a bit inside a frame segment with a length of more than one bit (not the first bit of those frame segments that is indicated by NewBit). The current bit is processed within one of the following frame segments: ID bits (except first bit of standard ID for transmission and first bit of ID extension), DLC (3 LSB) and bits 6-0 in each data byte.
11_B	**Done** The current bit is in one of the following frame segments: Acknowledge slot, last bit of EOF, active/passive-error frame, overload frame. Two or more directly consecutive Done codes signal an Error Frame.

18.7.3 Message Object Registers

MOCTR

The Message Object Control Register MOCTRn and the Message Object Status Register MOSTATn are located at the same address offset within a message object address block (offset address $1C_H$). The MOCTRn is a write-only register that makes it possible to set/reset CAN transfer related control bits through software.

MOCTR0
Message Object 0 Control Register ($101C_H$) **Reset Value: $0100\ 0000_H$**
MOCTRn (n = 1-62)
Message Object n Control Register
($101C_H+n*20_H$)
Reset Value: $((n+1)*01000000_H)+((n-1)*00010000_H)$

MOCTR63
Message Object 63 Control Register ($17FC_H$) **Reset Value: $3F3E\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0				SETDIR	SETTXEN1	SETTXEN0	SETTXRQ	SETRXEN	SETRTSEL	SETMSGVAL	SETMSGLST	SETNEWDAT	SETRXUPD	SETTXPND	SETRXPND
w				w	w	w	w	w	w	w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				RESDIR	RESTXEN1	RESTXEN0	RESTXRQ	RESRXEN	RESRTSEL	RESMSGVAL	RESMSGLST	RESNEWDAT	RESRXUPD	RESTXPND	RESRXPND
w				w	w	w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
RESRXPND, SETRXPND	0, 16	w	**Reset/Set Receive Pending** These bits control the set/reset condition for RXPND (see **Table 18-10**).
RESTXPND, SETTXPND	1, 17	w	**Reset/Set Transmit Pending** These bits control the set/reset condition for TXPND (see **Table 18-10**).
RESRXUPD, SETRXUPD	2, 18	w	**Reset/Set Receive Updating** These bits control the set/reset condition for RXUPD (see **Table 18-10**).

Field	Bits	Type	Description
RESNEWDAT, SETNEWDAT	3, 19	w	**Reset/Set New Data** These bits control the set/reset condition for NEWDAT (see **Table 18-10**).
RESMSGLST, SETMSGLST	4, 20	w	**Reset/Set Message Lost** These bits control the set/reset condition for MSGLST (see **Table 18-10**).
RESMSGVAL, SETMSGVAL	5, 21	w	**Reset/Set Message Valid** These bits control the set/reset condition for MSGVAL (see **Table 18-10**).
RESRTSEL, SETRTSEL	6, 22	w	**Reset/Set Receive/Transmit Selected** These bits control the set/reset condition for RTSEL (see **Table 18-10**).
RESRXEN, SETRXEN	7, 23	w	**Reset/Set Receive Enable** These bits control the set/reset condition for RXEN (see **Table 18-10**).
RESTXRQ, SETTXRQ	8, 24	w	**Reset/Set Transmit Request** These bits control the set/reset condition for TXRQ (see **Table 18-10**).
RESTXEN0, SETTXEN0	9, 25	w	**Reset/Set Transmit Enable 0** These bits control the set/reset condition for TXEN0 (see **Table 18-10**).
RESTXEN1, SETTXEN1	10, 26	w	**Reset/Set Transmit Enable 1** These bits control the set/reset condition for TXEN1 (see **Table 18-10**).
RESDIR, SETDIR	11, 27	w	**Reset/Set Message Direction** These bits control the set/reset condition for DIR (see **Table 18-10**).
0	[15:12], [31:28]	w	**Reserved** Should be written with 0.

Table 18-10 Reset/Set Conditions for Bits in Register MOCTRn

RESy Bit[1)]	SETy Bit	Action on Write
Write 0	Write 0	Leave element unchanged
	No write	
No write	Write 0	
Write 1	Write 1	
Write 1	Write 0	Reset element
	No write	
Write 0	Write 1	Set element
No write		

1) The parameter “y” stands for the second part of the bit name (“RXPND”, “TXPND”, … up to “DIR”).

MOSTAT

The MOSTATn is a read-only register that indicates message object list status information such as the number of the current message object predecessor and successor message object, as well as the list number to which the message object is assigned.

MOSTAT0
Message Object 0 Status Register ($101C_H$) Reset Value: 0100 0000_H
MOSTATn (n = 1-62)
Message Object n Status Register
($101C_H+n*20_H$)
Rest Value: $((n+1)*01000000_H)+((n-1)*00010000_H)$
MOSTAT63
Message Object 63 Status Register ($17FC_H$) Reset Value: 3F3E 0000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PNEXT								PPREV							
rh								rh							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LIST				DIR	TX EN1	TX EN0	TX RQ	RX EN	RTS EL	MSG VAL	MSG LST	NEW DAT	RX UPD	TX PND	RX PND
rh				rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
RXPND	0	rh	**Receive Pending** 0_B No CAN message has been received. 1_B A CAN message has been received by the message object n, either directly or via gateway copy action. RXPND is set by hardware and must be reset by software.
TXPND	1	rh	**Transmit Pending** 0_B No CAN message has been transmitted. 1_B A CAN message from message object n has been transmitted successfully over the CAN bus. TXPND is set by hardware and must be reset by software.

Field	Bits	Type	Description
RXUPD	2	rh	**Receive Updating** 0_B No receive update ongoing. 1_B Message identifier, DLC, and data of the message object are currently updated.
NEWDAT	3	rh	**New Data** 0_B No update of the message object n since last flag reset. 1_B Message object n has been updated. NEWDAT is set by hardware after a received CAN frame has been stored in message object n. NEWDAT is cleared by hardware when a CAN transmission of message object n has been started. NEWDAT should be set by software after the new transmit data has been stored in message object n to prevent the automatic reset of TXRQ at the end of an ongoing transmission.
MSGLST	4	rh	**Message Lost** 0_B No CAN message is lost. 1_B A CAN message is lost because NEWDAT has become set again when it has already been set.
MSGVAL	5	rh	**Message Valid** 0_B Message object n is not valid. 1_B Message object n is valid. Only a valid message object takes part in CAN transfers.

Field	Bits	Type	Description
RTSEL	6	rh	**Receive/Transmit Selected** 0_B Message object n is not selected for receive or transmit operation. 1_B Message object n is selected for receive or transmit operation. **Frame Reception:** RTSEL is set by hardware when message object n has been identified for storage of a CAN frame that is currently received. Before a received frame becomes finally stored in message object n, a check is performed to determine if RTSEL is set. Thus the CPU can suppress a scheduled frame delivery to this message object n by clearing RTSEL by software. **Frame Transmission:** RTSEL is set by hardware when message object n has been identified to be transmitted next. A check is performed to determine if RTSEL is still set before message object n is actually set up for transmission and bit NEWDAT is cleared. It is also checked that RTSEL is still set before its message object n is verified due to the successful transmission of a frame. RTSEL needs to be checked only when the context of message object n changes, and a conflict with an ongoing frame transfer shall be avoided. In all other cases, RTSEL can be ignored. RTSEL has no impact on message acceptance filtering. RTSEL is not cleared by hardware.
RXEN	7	rh	**Receive Enable** 0_B Message object n is not enabled for frame reception. 1_B Message object n is enabled for frame reception. RXEN is evaluated for receive acceptance filtering only.

Field	Bits	Type	Description
TXRQ	8	rh	**Transmit Request** 0_B No transmission of message object n is requested. 1_B Transmission of message object n on the CAN bus is requested. The transmit request becomes valid only if TXRQ, TXEN0, TXEN1 and MSGVAL are set. TXRQ is set by hardware if a matching Remote Frame has been received correctly. TXRQ is reset by hardware if message object n has been transmitted successfully and NEWDAT is not set again by software.
TXEN0	9	rh	**Transmit Enable 0** 0_B Message object n is not enabled for frame transmission. 1_B Message object n is enabled for frame transmission. Message object n can be transmitted only if both bits, TXEN0 and TXEN1, are set. The user may clear TXEN0 in order to inhibit the transmission of a message that is currently updated, or to disable automatic response of Remote Frames.
TXEN1	10	rh	**Transmit Enable 1** 0_B Message object n is not enabled for frame transmission. 1_B Message object n is enabled for frame transmission. Message object n can be transmitted only if both bits, TXEN0 and TXEN1, are set. TXEN1 is used by the MultiCAN module for selecting the active message object in the Transmit FIFOs.

Field	Bits	Type	Description
DIR	11	rh	**Message Direction** 0_B Receive Object selected: With TXRQ = 1, a Remote Frame with the identifier of message object n is scheduled for transmission. On reception of a Data Frame with matching identifier, the message is stored in message object n. 1_B Transmit Object selected: If TXRQ = 1, message object n is scheduled for transmission of a Data Frame. On reception of a Remote Frame with matching identifier, bit TXRQ is set.
LIST	[15:12]	rh	**List Allocation** LIST indicates the number of the message list to which message object n is allocated. LIST is updated by hardware when the list allocation of the object is modified by a panel command.
PPREV	[23:16]	rh	**Pointer to Previous Message Object** PPREV holds the message object number of the previous message object in a message list structure.
PNEXT	[31:24]	rh	**Pointer to Next Message Object** PNEXT holds the message object number of the next message object in a message list structure.

Table 18-11 MOSTATn Reset Values

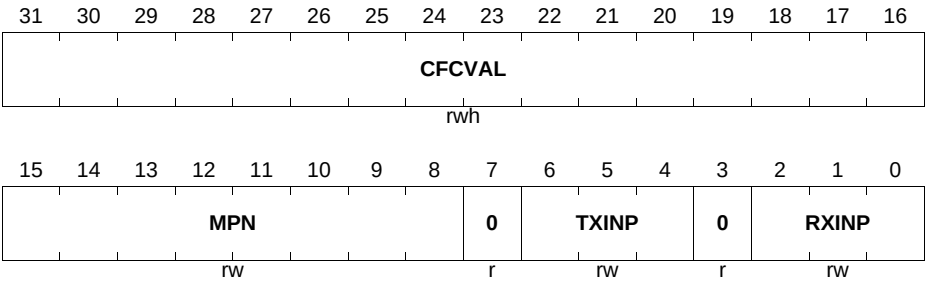
Message Object	PNEXT	PPREV	Reset Value
0	1	0	$0100\ 0000_H$
1	2	0	$0200\ 0000_H$
2	3	1	$0301\ 0000_H$
3	4	2	$0402\ 0000_H$
…	…	…	…
60	61	59	$3D3B\ 0000_H$
61	62	60	$3E3C\ 0000_H$
62	63	61	$3F3D\ 0000_H$
63	63	62	$3F3E\ 0000_H$

MOIPR

The Message Object Interrupt Pointer Register MOIPRn holds the message interrupt pointers, the message pending number, and the frame counter value of message object n.

MOIPRn (n = 0-63)
Message Object n Interrupt Pointer Register

(1008_H+n*20_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
RXINP	[2:0]	rw	**Receive Interrupt Node Pointer** RXINP selects the interrupt output line INT_Om (m = 0-7) for a receive interrupt event of message object n. RXINP can also be taken for message pending bit selection (see **Page 18-37**). 000_B Interrupt output line INT_O0 is selected. 001_B Interrupt output line INT_O1 is selected. $\ldots_B$ … 111_B Interrupt output line INT_O7 is selected.
TXINP	[6:4]	rw	**Transmit Interrupt Node Pointer** TXINP selects the interrupt output line INT_Om (m = 0-7) for a transmit interrupt event of message object n. TXINP can also be taken for message pending bit selection (see **Page 18-37**). 000_B Interrupt output line INT_O0 is selected. 001_B Interrupt output line INT_O1 is selected. $\ldots_B$ … 111_B Interrupt output line INT_O7 is selected.

Field	Bits	Type	Description
MPN	[15:8]	rw	**Message Pending Number** This bit field selects the bit position of the bit in the Message Pending Register that is set upon a message object n receive/transmit interrupt.
CFCVAL	[31:16]	rwh	**CAN Frame Counter Value** When a message is stored in message object n or message object n has been successfully transmitted, the CAN frame counter value NFCRx.CFC is then copied to CFCVAL.
0	7, 3	r	**Reserved** Read as 0; should be written with 0.

MOFCR

The Message Object Function Control Register MOFCRn contains bits that select and configure the function of the message object. It also holds the CAN data length code.

MOFCRn (n = 0-63)
Message Object n Function Control Register
(1000_H+n*20_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0				DLC				STT	SDT	RMM	FRR EN	0	OVIE	TXIE	RXIE
rw				rwh				rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				DATC	DLCC	IDC	GDFS	0				MMC			
rw				rw	rw	rw	rw	rw				rw			

Field	Bits	Type	Description
MMC	[3:0]	rw	**Message Mode Control** MMC controls the message mode of message object n. 0000_B Standard Message Object 0001_B Receive FIFO Base Object 0010_B Transmit FIFO Base Object 0011_B Transmit FIFO Slave Object 0100_B Gateway Source Object $..._B$ Reserved
GDFS	8	rw	**Gateway Data Frame Send** 0_B TXRQ is unchanged in the destination object. 1_B TXRQ is set in the gateway destination object after the internal transfer from the gateway source to the gateway destination object. Applicable only to a gateway source object; ignored in other nodes.

Field	Bits	Type	Description
IDC	9	rw	**Identifier Copy** 0_B The identifier of the gateway source object is not copied. 1_B The identifier of the gateway source object (after storing the received frame in the source) is copied to the gateway destination object. Applicable only to a gateway source object; ignored in other nodes.
DLCC	10	rw	**Data Length Code Copy** 0_B Data length code is not copied. 1_B Data length code of the gateway source object (after storing the received frame in the source) is copied to the gateway destination object. Applicable only to a gateway source object; ignored in other nodes.
DATC	11	rw	**Data Copy** 0_B Data fields are not copied. 1_B Data fields in registers MODATALn and MODATAHn of the gateway source object (after storing the received frame in the source) are copied to the gateway destination. Applicable only to a gateway source object; ignored in other nodes.
RXIE	16	rw	**Receive Interrupt Enable** RXIE enables the message receive interrupt of message object n. This interrupt is generated after reception of a CAN message (independent of whether the CAN message is received directly or indirectly via a gateway action). 0_B Message receive interrupt is disabled. 1_B Message receive interrupt is enabled. Bit field MOIPRn.RXINP selects the interrupt output line which becomes activated at this type of interrupt.

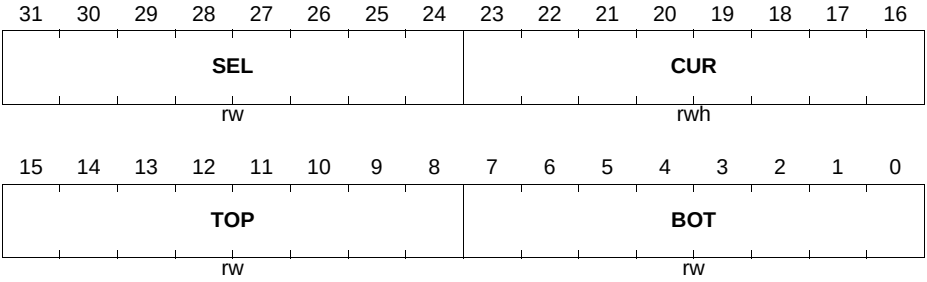
Field	Bits	Type	Description
TXIE	17	rw	**Transmit Interrupt Enable** TXIE enables the message transmit interrupt of message object n. This interrupt is generated after the transmission of a CAN message. 0_B Message transmit interrupt is disabled. 1_B Message transmit interrupt is enabled. Bit field MOIPRn.TXINP selects the interrupt output line which becomes activated at this type of interrupt.
OVIE	18	rw	**Overflow Interrupt Enable** OVIE enables the FIFO full interrupt of message object n. This interrupt is generated when the pointer to the current message object (CUR) reaches the value of SEL in the FIFO/Gateway Pointer Register. 0_B FIFO full interrupt is disabled. 1_B FIFO full interrupt is enabled. If message object n is a Receive FIFO base object, bit field MOIPRn.TXINP selects the interrupt output line which becomes activated at this type of interrupt. If message object n is a Transmit FIFO base object, bit field MOIPRn.RXINP selects the interrupt output line which becomes activated at this type of interrupt. For all other message object modes, bit OVIE has no effect.
FRREN	20	rw	**Foreign Remote Request Enable** Specifies whether the TXRQ bit is set in message object n or in a foreign message object referenced by the pointer CUR. 0_B TXRQ of message object n is set on reception of a matching Remote Frame. 1_B TXRQ of the message object referenced by the pointer CUR is set on reception of a matching Remote Frame.

Field	Bits	Type	Description
RMM	21	rw	**Transmit Object Remote Monitoring** 0_B Remote monitoring is disabled: Identifier, IDE bit, and DLC of message object n remain unchanged upon the reception of a matching Remote Frame. 1_B Remote monitoring is enabled: Identifier, IDE bit, and DLC of a matching Remote Frame are copied to transmit object n in order to monitor incoming Remote Frames. Bit RMM applies only to transmit objects and has no effect on receive objects.
SDT	22	rw	**Single Data Transfer** If SDT = 1 and message object n is not a FIFO base object, then MSGVAL is reset when this object has taken part in a successful data transfer (receive or transmit). If SDT = 1 and message object n is a FIFO base object, then MSGVAL is reset when the pointer to the current object CUR reaches the value of SEL in the FIFO/Gateway Pointer Register. With SDT = 0, bit MSGVAL is not affected.
STT	23	rw	**Single Transmit Trial** If this bit is set, then TXRQ is cleared on transmission start of message object n. Thus, no transmission retry is performed in case of transmission failure.
DLC	[27:24]	rwh	**Data Length Code** Bit field determines the number of data bytes for message object n. Valid values for DLC are 0 to 8. A value of DLC > 8 results in a data length of 8 data bytes. If a frame with DLC > 8 is received, the received value is stored in the message object.
0	[7:4], [15:12], 19	rw	**Reserved** Read as 0 after reset; value last written is read back; should be written with 0.

MOFGPR

The Message Object FIFO/Gateway Pointer register MOFGPRn contains a set of message object link pointers that are used for FIFO and gateway operations.

MOFGPRn (n = 0-63)
Message Object n FIFO/Gateway Pointer Register
(1004_H+n*20_H) **Reset Value: 0000 0000_H**

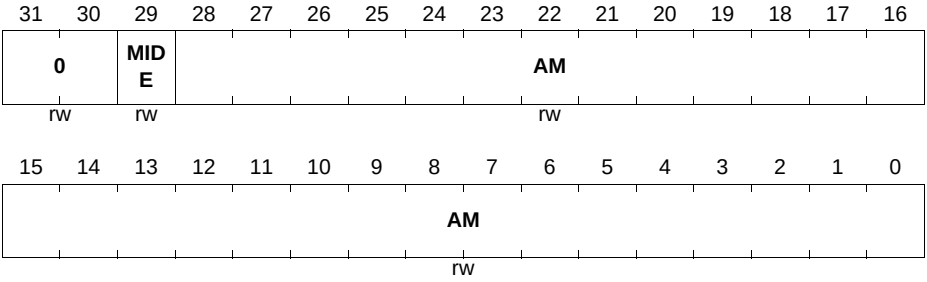


Field	Bits	Type	Description
BOT	[7:0]	rw	**Bottom Pointer** Bit field BOT points to the first element in a FIFO structure.
TOP	[15:8]	rw	**Top Pointer** Bit field TOP points to the last element in a FIFO structure.
CUR	[23:16]	rwh	**Current Object Pointer** Bit field CUR points to the actual target object within a FIFO/Gateway structure. After a FIFO/gateway operation CUR is updated with the message number of the next message object in the list structure (given by PNEXT of the message control register) until it reaches the FIFO top element (given by TOP) when it is reset to the bottom element (given by BOT).
SEL	[31:24]	rw	**Object Select Pointer** Bit field SEL is the second (software) pointer to complement the hardware pointer CUR in the FIFO structure. SEL is used for monitoring purposes (FIFO interrupt generation).

MOAMR

Message Object n Acceptance Mask Register MOAMRn contains the mask bits for the acceptance filtering of the message object n.

MOAMRn (n = 0-63)
Message Object n Acceptance Mask Register
($100C_H$+n*20_H) **Reset Value: 3FFF FFFF$_H$**



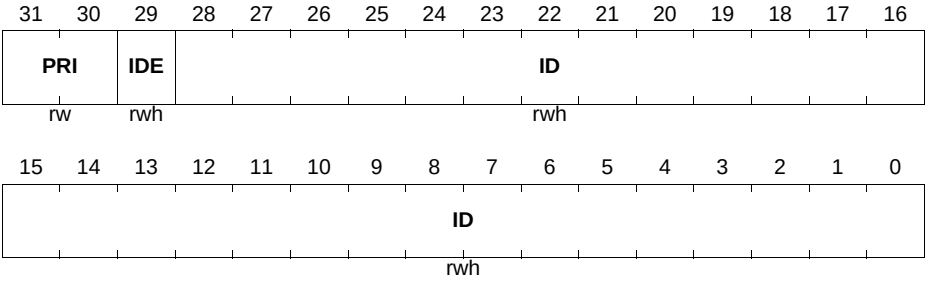
Field	Bits	Type	Description
AM	[28:0]	rw	**Acceptance Mask for Message Identifier** Bit field AM is the 29-bit mask for filtering incoming messages with standard identifiers (AM[28:18]) or extended identifiers (AM[28:0]). For standard identifiers, bits AM[17:0] are “don’t care”.
MIDE	29	rw	**Acceptance Mask Bit for Message IDE Bit** 0_B Message object n accepts the reception of both, standard and extended frames. 1_B Message object n receives frames only with matching IDE bit.
0	[31:30]	rw	**Reserved** Read as 0 after reset; value last written is read back; should be written with 0.

MOAR

Message Object n Arbitration Register MOARn contains the CAN identifier of the message object.

MOARn (n = 0-63)
Message Object n Arbitration Register

(1018_H+n*20_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
ID	[28:0]	rwh	**CAN Identifier of Message Object n** Identifier of a standard message (ID[28:18]) or an extended message (ID[28:0]). For standard identifiers, bits ID[17:0] are “don’t care”.
IDE	29	rwh	**Identifier Extension Bit of Message Object n** 0_B Message object n handles standard frames with 11-bit identifier. 1_B Message object n handles extended frames with 29-bit identifier.

Field	Bits	Type	Description
PRI	[31:30]	rw	**Priority Class** PRI assigns one of the four priority classes 0, 1, 2, 3 to message object n. A lower PRI number defines a higher priority. Message objects with lower PRI value always win acceptance filtering for frame reception and transmission over message objects with higher PRI value. Acceptance filtering based on identifier/mask and list position is performed only between message objects of the same priority class. PRI also determines the acceptance filtering method for transmission: 00_B Applicable only if TTCAN is available. 01_B Transmit acceptance filtering is based on the list order. This means that message object n is considered for transmission only if there is no other message object with valid transmit request (MSGVAL & TXEN0 & TXEN1 = 1) somewhere before this object in the list. 10_B Transmit acceptance filtering is based on the CAN identifier. This means, message object n is considered for transmission only if there is no other message object with higher priority identifier + IDE + DIR (with respect to CAN arbitration rules) somewhere in the list (see **Table 18-12**). 11_B Transmit acceptance filtering is based on the list order (as PRI = 01_B).

Transmit Priority of Msg. Objects based on CAN Arbitration Rules

Table 18-12 Transmit Priority of Msg. Objects Based on CAN Arbitration Rules

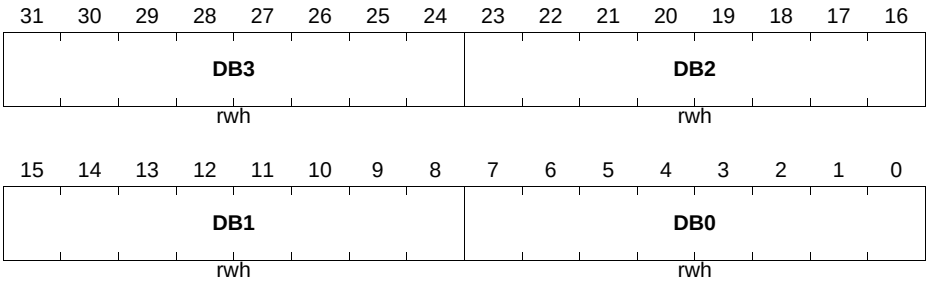
Settings of Arbitrarily Chosen Message Objects A and B, (A has higher transmit priority than B)	Comment
A.MOAR[28:18] < B.MOAR[28:18] (11-bit standard identifier of A less than 11-bit standard identifier of B)	Messages with lower standard identifier have higher priority than messages with higher standard identifier. MOAR[28] is the most significant bit (MSB) of the standard identifier. MOAR[18] is the least significant bit of the standard identifier.
A.MOAR[28:18] = B.MOAR[28:18] A.MOAR.IDE = 0 (send Standard Frame) B.MOAR.IDE = 1 (send Extended Frame)	Standard Frames have higher transmit priority than Extended Frames with equal standard identifier.
A.MOAR[28:18] = B.MOAR[28:18] A.MOAR.IDE = B.MOAR.IDE = 0 A.MOCTR.DIR = 1 (send Data Frame) B.MOCTR.DIR = 0 (send Remote Fame)	Standard Data Frames have higher transmit priority than standard Remote Frames with equal identifier.
A.MOAR[28:0] = B.MOAR[28:0] A.MOAR.IDE = B.MOAR.IDE = 1 A.MOCTR.DIR = 1 (send Data Frame) B.MOCTR.DIR = 0 (send Remote Frame)	Extended Data Frames have higher transmit priority than Extended Remote Frames with equal identifier.
A.MOAR[28:0] < B.MOAR[28:0] A.MOAR.IDE = B.MOAR.IDE = 1 (29-bit identifier)	Extended Frames with lower identifier have higher transmit priority than Extended Frames with higher identifier. MOAR[28] is the most significant bit (MSB) of the overall identifier (standard identifier MOAR[28:18] and identifier extension MOAR[17:0]). MOAR[0] is the least significant bit (LSB) of the overall identifier.

MODATAL

Message Object n Data Register Low MODATALn contains the lowest four data bytes of message object n. Unused data bytes are set to zero upon reception and ignored for transmission.

MODATALn (n = 0-63)
Message Object n Data Register Low

(1010_H+n*20_H) **Reset Value: 0000 0000_H**



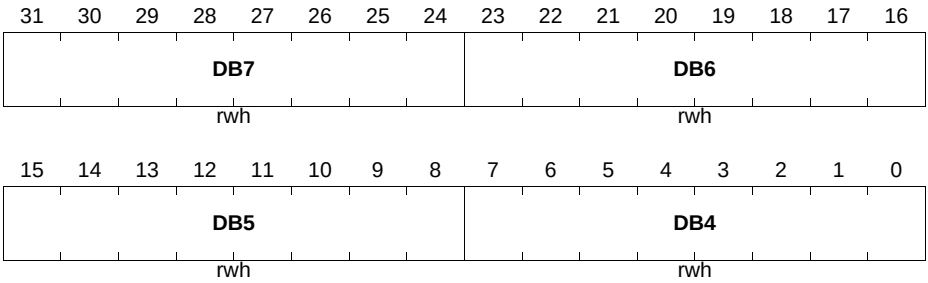
Field	Bits	Type	Description
DB0	[7:0]	rwh	**Data Byte 0 of Message Object n**
DB1	[15:8]	rwh	**Data Byte 1 of Message Object n**
DB2	[23:16]	rwh	**Data Byte 2 of Message Object n**
DB3	[31:24]	rwh	**Data Byte 3 of Message Object n**

MODATAH

Message Object n Data Register High MODATAH contains the highest four data bytes of message object n. Unused data bytes are set to zero upon reception and ignored for transmission.

MODATAHn (n = 0-63)
Message Object n Data Register High

(1014_H+n*20_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
DB4	[7:0]	rwh	**Data Byte 4 of Message Object n**
DB5	[15:8]	rwh	**Data Byte 5 of Message Object n**
DB6	[23:16]	rwh	**Data Byte 6 of Message Object n**
DB7	[31:24]	rwh	**Data Byte 7 of Message Object n**

18.7.4 MultiCAN Module External Registers

The registers listed in **Figure 18-25** must be programmed for proper operation of the MultiCAN module.

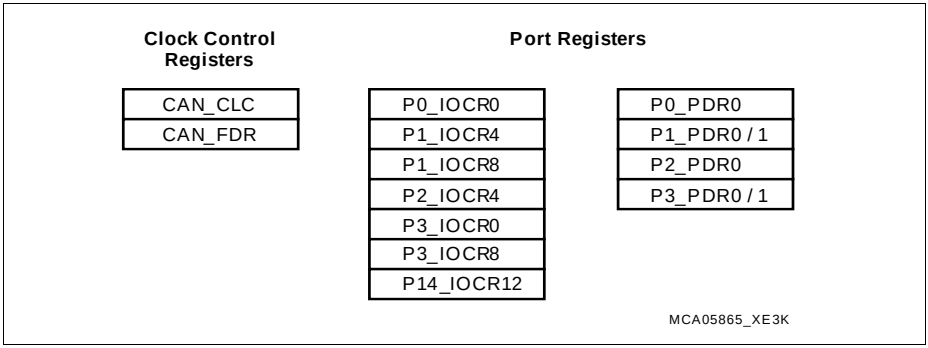


Figure 18-25 CAN Implementation-specific Special Function Registers

Table 18-13 MultiCAN Module External Registers

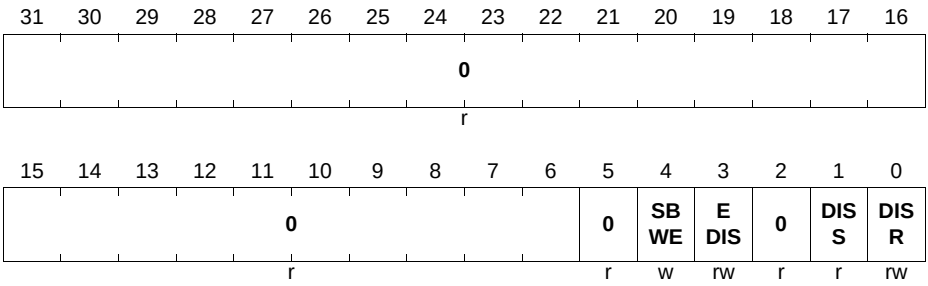
Short Name	Description	Offset Addr	Access Mode[1)]		Description see
			Read	Write	
Module Identification Registers					
ID	Module Identification Register	008_H	U, PV	U, PV	Page 18-62
Clock Control Registers					
CLC	Clock Control Register	000_H	U, PV	PV, E	Page 18-115
FDR	Fractional Divider Register	$00C_H$	U, PV	PV, E	Page 18-116

1) Accesses to empty addresses: nBE

CAN_CLC

The clock control registers makes it possible to control (enable/disable) the module control clock f_{CLC}.

CAN_CLC
CAN Clock Control Register (000_H) Reset Value: 0000 0003$_H$



Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module.
DISS	1	r	**Module Disable Status Bit** Bit indicates the current status of the module.
EDIS	3	rw	**Sleep Mode Enable Control** Used to control module's sleep mode.
SBWE	4	w	**Module Suspend Bit Write Enable for OCDS** Determines whether SPEN and FSOE are write-protected.
0	2, 5, [31:6]	r	**Reserved** Read as 0; should be written with 0.

Note: In disabled state, no registers of CAN module can be read or written except the CAN_CLC register.

CAN_FDR

The fractional divider register allows the programmer to control the clock rate of the module timer clock f_{CAN}.

CAN_FDR
CAN Fractional Divider Register (00C$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DISCLK	ENHW	SUSREQ	SUSACK	0		RESULT									
rwh	rw	rh	rh	r		rh									

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DM		SC		SM	0	STEP									
rw		rw		rw	r	rw									

Field	Bits	Type	Description
STEP	[9:0]	rw	**Step Value** Reload or addition value for RESULT.
SM	11	rw	**Suspend Mode** SM selects between granted or immediate Suspend Mode.
SC	[13:12]	rw	**Suspend Control** This bit field determines the behavior of the fractional divider in Suspend Mode.
DM	[15:14]	rw	**Divider Mode** This bit field selects normal divider mode, fractional divider mode, and off-state.
RESULT	[25:16]	rh	**Result Value** Bit field for the addition result.
SUSACK	28	rh	**Suspend Mode Acknowledge** Indicates state of $\overline{\text{SPNDACK}}$ signal.
SUSREQ	29	rh	**Suspend Mode Request** Indicates state of SPND signal.
ENHW	30	rw	**Enable Hardware Clock Control** Controls operation of ECEN input and DISCLK bit.

Field	Bits	Type	Description
DISCLK	31	rwh	**Disable Clock** Hardware controlled disable for f_{OUT} signal.
0	10, [27:26]	r	**Reserved** Read as 0; should be written with 0.

MultiCAN Module Register Address Map

The complete MultiCAN module register address map of **Figure 18-26** shows the general implementation-specific registers for clock control, module identification, and interrupt service request control and adds the absolute address information.

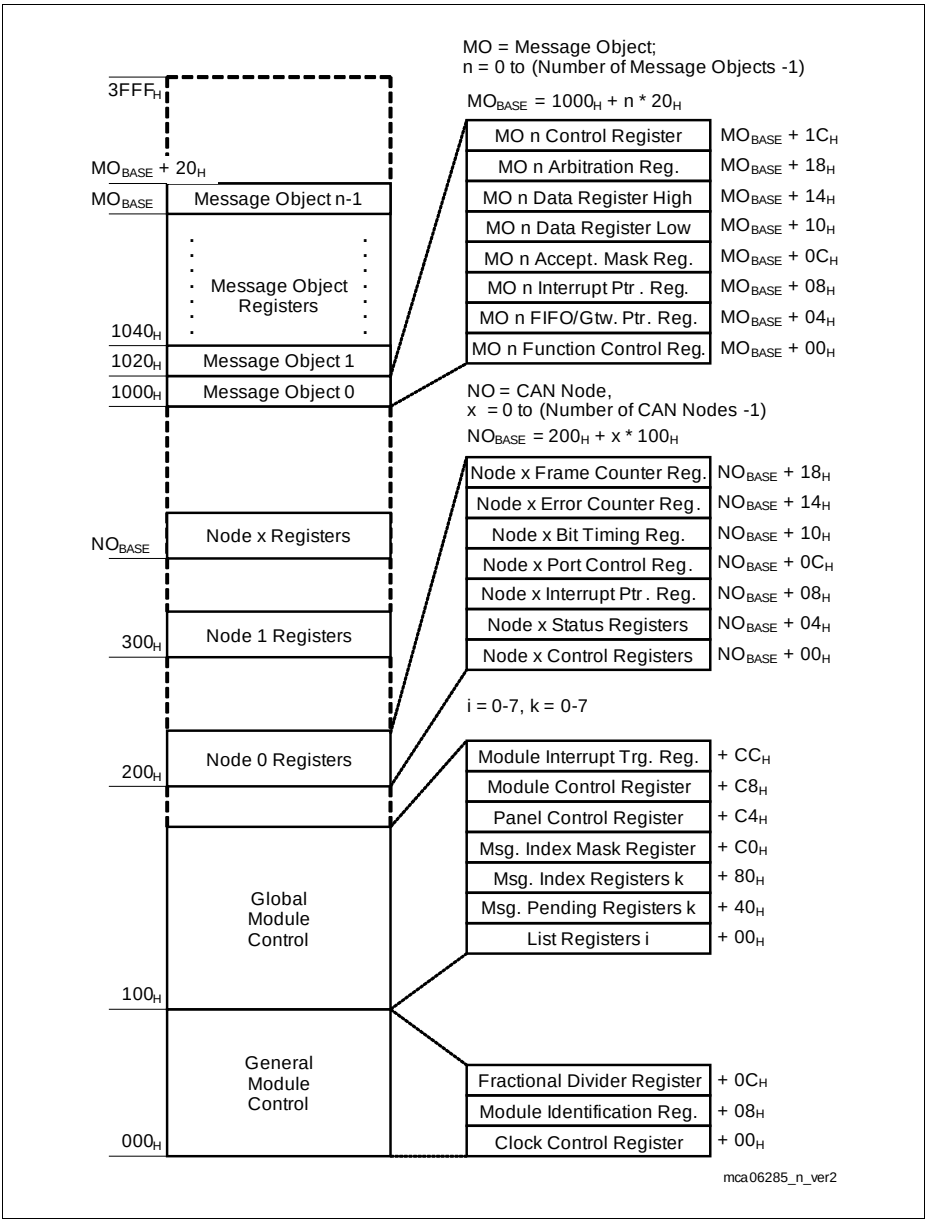


Figure 18-26 MultiCAN Module Register Map

18.8 Interconnects

This section describes CAN module interfaces with the clock control, port connections, and address decoding.

18.8.1 Interfaces of the MultiCAN Module

Figure 18-27 shows the XMC4500 specific implementation details and interconnections of the MultiCAN module. The six I/O lines of the MultiCAN module (two I/O lines of each CAN node) are connected to I/O lines of Port 0,1,2,3,4 and 14. The MultiCAN module is also supplied by clock control, interrupt control, and address decoding logic. MultiCAN interrupts can be directed to the GPDMA controller and the CCU4 modules. CAN interrupts are able to trigger DMA transfers and CCU4 operations.

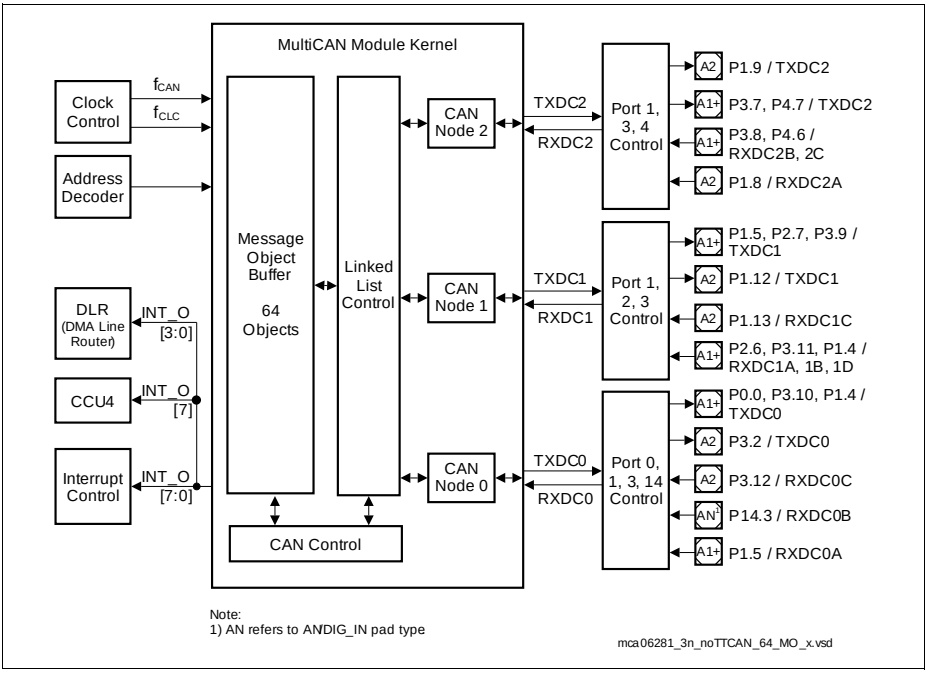


Figure 18-27 CAN module Implementation and Interconnections

18.8.2 Port and I/O Line Control

The interconnections between the MultiCAN module and the port I/O lines are controlled in the port logic. Additionally to the port input selection, the following port control operations must be executed:

- Input/output function selection (IOCR registers)
- Pad driver characteristics selection for the outputs (PDR registers)

18.8.2.1 Input/Output Function Selection in Ports

The port input/output control registers contain the bit fields that select the digital output and input driver characteristics such as pull-up/down devices, port direction (input/output), open-drain, and alternate output selections. The I/O lines for the MultiCAN module are controlled by the port input/output control registers Pn_IOCRy PCx defined in the GPIO chapter.

Additionally to the I/O control selection, as defined in **Table 18-14**, the selection of a CAN node's receive input line requires that bit field RXSEL in its node port control register NPCRx must be set.

The selected input signal (selected by bit field NPCRx.RXSEL) for each CAN node is made available by internal signal CANxINS (CAN node x input signal, with x = 0 - 2) as shown in **Figure 18-28**. The default setting after reset of a node's NPCRx.RXSEL bit field connect node x with RXDCx I/O line (x = 0-2).

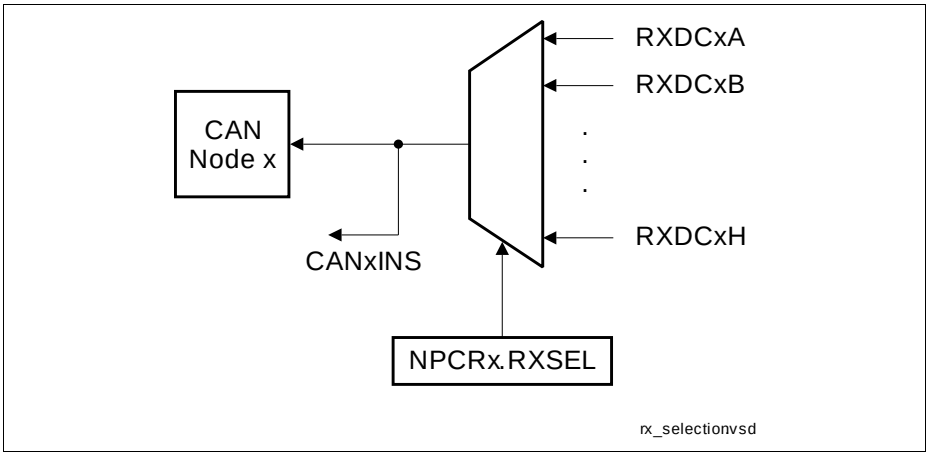


Figure 18-28 CAN Module Receive Input Selection

Table 18-14 shows how bits and bit fields must be programmed for the required I/O functionality of the CAN I/O lines.

Table 18-14 MultiCAN I/O Control Selection and Setup

Input/Output	I/O	Connected To	Description
Receive Inputs (Node 0)			
CAN.CAN0_RXDC0A	I	P1.5	CAN Receive Input
CAN.CAN0_RXDC0B	I	P14.3	CAN Receive Input
CAN.CAN0_RXDC0C	I	P3.12	CAN Receive Input
Receive Inputs (Node 1)			
CAN.CAN1_RXDC1A	I	P2.6	CAN Receive Input
CAN.CAN1_RXDC1B	I	P3.11	CAN Receive Input
CAN.CAN1_RXDC1C	I	P1.13	CAN Receive Input
CAN.CAN1_RXDC1D	I	P1.4	CAN Receive Input
CAN.CAN1_RXDC1F	I	CAN0INS	CAN Receive Input
Receive Inputs (Node 2)			
CAN.CAN2_RXDC2A	I	P1.8	CAN Receive Input
CAN.CAN2_RXDC2B	I	P3.8	CAN Receive Input
CAN.CAN2_RXDC2C	I	P4.6	CAN Receive Input
CAN.CAN2_RXDC2F	I	CAN1INS	CAN Receive Input
Transmit Outputs (Node 0)			
CAN.CAN0_TXDC0	O	P0.0	CAN Transmit Output
	O	P1.4	CAN Transmit Output
	O	P3.2	CAN Transmit Output
	O	P3.10	CAN Transmit Output
Transmit Outputs (Node 1)			
CAN.CAN1_TXDC1	O	P3.9	CAN Transmit Output
	O	P2.7	CAN Transmit Output
	O	P1.12	CAN Transmit Output
	O	P1.5	CAN Transmit Output
Transmit Outputs (Node 2)			

Table 18-14 MultiCAN I/O Control Selection and Setup (cont'd)

Input/Output	I/O	Connected To	Description
CAN.CAN2_TXDC2	O	P3.7	CAN Transmit Output
	O	P1.9	CAN Transmit Output
	O	P4.7	CAN Transmit Output

18.8.2.2 MultiCAN Interrupt Output Connections

The interrupt outputs of the MultiCAN module are connected as shown in **Table 18-15**.

Table 18-15 CAN Interrupt Output Connections

Input/Output	I/O	Connected To	Description
System Related Outputs			
SR0	O	NVIC	Interrrupt Request
	O	DLR	DMA Request
SR1	O	CPU	CAN Interrrupt Output
	O	DLR	DMA Request
SR2	O	CPU	CAN Interrrupt Output
	O	DLR	DMA Request
SR3	O	CPU	CAN Interrrupt Output
	O	DLR	DMA Request
SR4	O	NVIC	Interrrupt Request
SR5	O	NVIC	Interrrupt Request
SR6	O	NVIC	Interrrupt Request
SR7	O	NVIC	Interrrupt Request
	O	CCU4	CCU4 Trigger

18.8.2.3 Connections to USIC Inputs

The internal signal CAN1INS is connected to the USIC module, see **Table 18-16**.

Table 18-16 CAN-to-USIC Connections

Input/Output	I/O	Connected To	Description
System Related Outputs			
CAN.CAN1INS	O	U1C1_DX0E	CAN Receive Multiplexer Output
	O	U2C2_DX0E	CAN Receive Multiplexer Output

Analog Frontend Peripherals

19 Versatile Analog-to-Digital Converter (VADC)

The XMC4500 provides a series of analog input channels connected to a cluster of Analog/Digital Converters using the Successive Approximation Register (SAR) principle to convert analog input values (voltages) to discrete digital values.
The number of analog input channels and ADCs depends on the chosen product type (please refer to **"Product-Specific Configuration" on Page 19-127**).

Table 19-1 Abbreviations used in ADC chapter

ADC	Analog to Digital Converter
DMA	Direct Memory Access (controller)
DNL	Differential Non-Linearity (error)
INL	Integral Non-Linearity (error)
LSB_n	Least Significant Bit: finest granularity of the analog value in digital format, represented by one least significant bit of the conversion result with n bits resolution (measurement range divided in 2^n equally distributed steps)
SCU	System Control Unit of the device
TUE	Total Unadjusted Error

19.1 Overview

Each converter of the ADC cluster can operate independent of the others, controlled by a dedicated set of registers and triggered by a dedicated group request source. The results of each channel can be stored in a dedicated channel-specific result register or in a group-specific result register.

A background request source can access all analog input channels that are not assigned to any group request source. These conversions are executed with low priority. The background request source can, therefore, be regarded as an additional background converter.

The Versatile Analog to Digital Converter module (VADC) of the XMC4500 comprises a set of converter blocks that can be operated either independently or via a common request source that emulates a background converter. Each converter block is equipped with a dedicated input multiplexer and dedicated request sources, which together build separate groups.

This basic structure supports application-oriented programming and operating while still providing general access to all resources. The almost identical converter groups allow a flexible assignment of functions to channels.

The basic module clock f_{ADC} is connected to the system clock signal f_{PB}.

Feature List

The following features describe the functionality of the ADC cluster:

- Nominal analog supply voltage 3.3 V
- Input voltage range from 0 V up to analog supply voltage
- Standard (V_{AREF}) and alternate (CH0) reference voltage source selectable for each channel to support ratiometric measurements and different signal scales
- Up to 4 independent converters with up to 8 analog input channels
- External analog multiplexer control, including adjusted sample time and scan support
- Conversion speed and sample time adjustable to adapt to sensors and reference
- Conversion time below 1 µs (depending on result width and sample time)
- Flexible source selection and arbitration
  - Programmable arbitrary conversion sequence (single or repeated)
  - Configurable auto scan conversion (single or repeated) on each converter
  - Configurable auto scan conversion (single or repeated) in the background (all converters)
  - Conversions triggered by software, timer events, or external events
  - Cancel-inject-restart mode for reduced conversion delay on priority channels
- Powerful result handling
  - Selectable result width of 8/10/12 bits
  - Fast Compare Mode
  - Independent result registers
  - Configurable limit checking against programmable border values
  - Data rate reduction through adding a selectable number of conversion results
  - FIR/IIR filter with selectable coefficients
- Flexible service request generation based on selectable events
- Built-in safety features
  - Broken wire detection with programmable default levels
  - Multiplexer test mode to verify signal path integrity
- Support of suspend and power saving modes

Note: Additional functions are available from the out of range comparator (see description in the SCU).

Table 19-2 VADC Applications

Use Case VADC	Application
Automatic scheduling of complex conversion sequences, including priorization of time-critical conversions	Motor control, Power conversion
Effective result handling for bursts of high-speed conversions	Highly dynamic input signals
Synchronous sampling of up to 4 input signals	Multi-phase current measurement

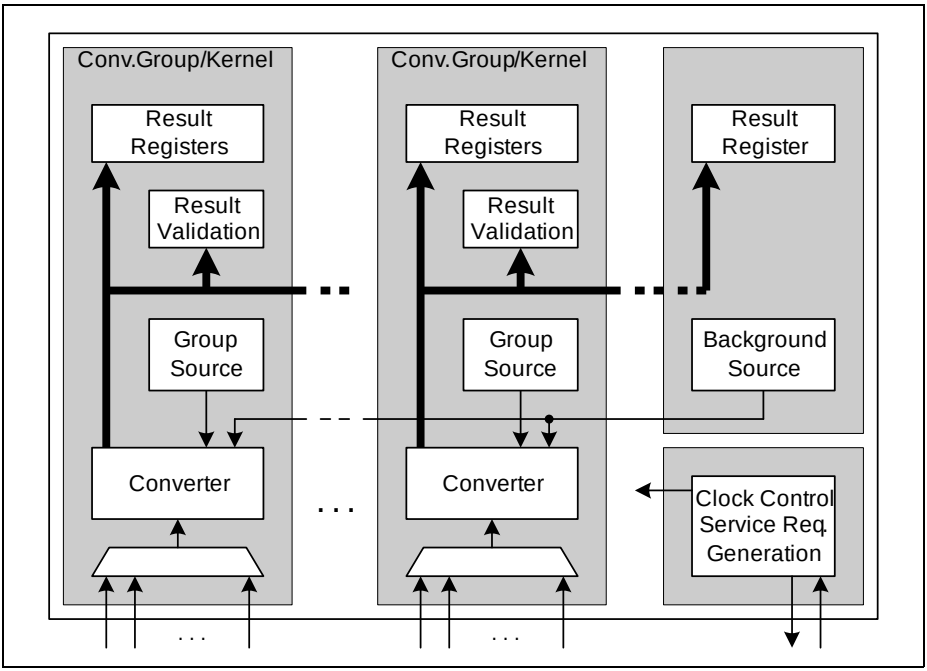


Figure 19-1 ADC Structure Overview

19.2 Introduction and Basic Structure

The Versatile Analog to Digital Converter module (VADC) of the XMC4500 comprises a set of converter blocks that can be operated either independently or via a common request source that emulates a background converter. Each converter block is equipped with a dedicated input multiplexer and dedicated request sources, which together build separate groups.

This basic structure supports application-oriented programming and operating while still providing general access to all resources. The almost identical converter groups allow a flexible assignment of functions to channels.

A set of functional units can be configured according to the requirements of a given application. These units build a path from the input signals to the digital results.

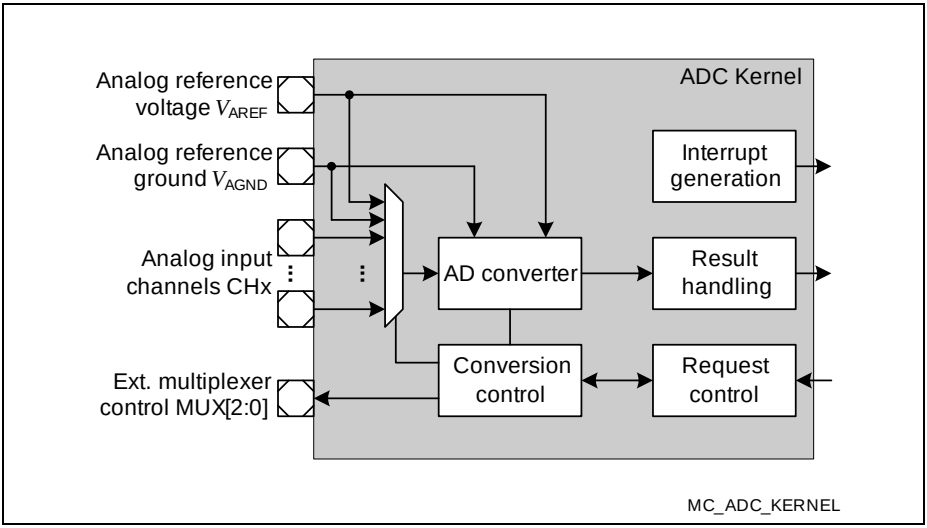


Figure 19-2 ADC Kernel Block Diagram

Conversion Modes and Request Sources

Analog/Digital conversions can be requested by several request sources (2 group request sources and the background request source) and can be executed in several conversion modes. The request sources can be enabled concurrently with configurable priorities.

- **Fixed Channel Conversion (single or continuous)**
  A specific channel source requests conversions of one selectable channel (once or repeatedly)
- **Auto Scan Conversion (single or continuous)**
  A channel scan source (request source 1 or 2) requests auto scan conversions of a configurable linear sequence of all available channels (once or repeatedly)
- **Channel Sequence Conversion (single or continuous)**
  A queued source (request source 0) requests a sequence of conversions of up to 8 arbitrarily selectable channels (once or repeatedly)

The conversion modes can be used concurrently by the available request sources, i.e. conversions in different modes can be enabled at the same time. Each source can be enabled separately and can be triggered by external events, such as edges of PWM or timer signals, or pin transitions.

Request Source Control

Because all request sources can be enabled at the same time, an arbiter resolves concurrent conversion requests from different sources. Each source can be triggered by external signals, by on-chip signals, or by software.

Requests with higher priority can either cancel a running lower-priority conversion (cancel-inject-repeat mode) or be converted immediately after the currently running conversion (wait-for-start mode). If the target result register has not been read, a conversion can be deferred (wait-for-read mode).

Certain channels can also be synchronized with other ADC kernels, so several signals can be converted in parallel.

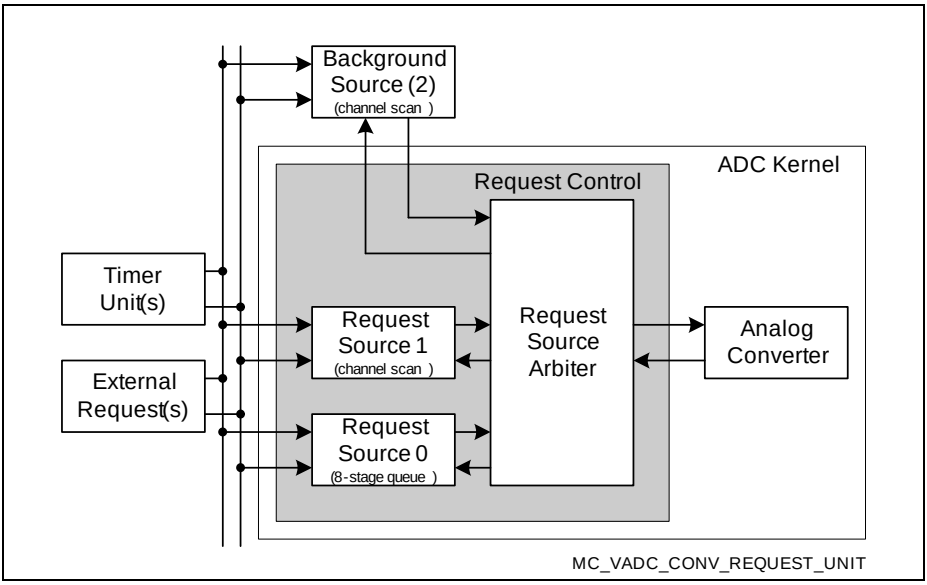


Figure 19-3 Conversion Request Unit

Input Channel Selection

The analog input multiplexer selects one of the available analog inputs (CH0 - CHx[1)]) to be converted. Three sources can select a linear sequence, an arbitrary sequence, or a specific channel. The priorities of these sources can be configured.

Additional external analog multiplexers can be controlled automatically, if more separate input channels are required than are built in.

*Note: Not all analog input channels are necessarily available in all packages, due to pin limitations. Please refer to the implementation description in **Section 19.14**.*

Conversion Control

Conversion parameters, such as sample phase duration, reference voltage, or result resolution can be configured for 4 input classes (2 group-specific classes, 2 global classes). Each channel can be individually assigned to one of these input classes.

The input channels can, thus, be adjusted to the type of sensor (or other analog sources) connected to the ADC.

1) The availablity of input channels depends on the package of the used product type. A summary can be found in **Section 19.14.2**.

This unit also controls the built-in multiplexer and external analog multiplexers, if selected.

Analog/Digital Converter

The selected input channel is converted to a digital value by first sampling the voltage on the selected input and then generating the selected number of result bits.

For 12-bit conversions, post-calibration is executed after converting the channel.

For broken wire detection (see **Section 19.10.1**), the converter network can be preloaded before sampling the selected input channel.

Result Handling

The conversion results of each analog input channel can be directed to one of 16 group-specific result registers and one global result register to be stored there. A result register can be used by a group of channels or by a single channel.

The wait-for-read mode avoids data loss due to result overwrite by blocking a conversion until the previous result has been read.

Data reduction (e.g. for digital anti-aliasing filtering) can automatically add up to 4 conversion results before issuing a service request.

Alternatively, an FIR or IIR filter can be enabled that preprocesses the conversion results before sending them to the result register.

Also, result registers can be concatenated to build FIFO structures that store a number of conversion results without overwriting previous data. This increases the allowed CPU latency for retrieving conversion data from the ADC.

Service Request Generation

Several ADC events can issue service requests to CPU or DMA:

- **Source events** indicate the completion of a conversion sequence in the corresponding request source. This event can be used to trigger the setup of a new sequence.
- **Channel events** indicate the completion of a conversion for a certain channel. This can be combined with limit checking, so interrupt are generated only if the result is within a defined range of values.
- **Result events** indicate the availability of new result data in the corresponding result register. If data reduction mode is active, events are generated only after a complete accumulation sequence.

Each event can be assigned to one of eight service request nodes. This allows grouping the requests according to the requirements of the application.

Safety Features

Safety-aware applications are supported with mechanisms that help to ensure the integrity of a signal path.

Broken-wire-detection (BWD) preloads the converter network with a selectable level before sampling the input channel. The result will then reflect the preload value if the input signal is no more connected. If buffer capacitors are used, a certain number of conversions may be required to reach the failure indication level.

Pull Down Diagnostics (PDD) connects an additional strong pull-down device to an input channel. A subsequent conversion can then confirm the expected modified signal level. This allows to check the proper connection of a signal source (sensor) to the multiplexer.

Multiplexer Diagnostics (MD) connects a weak pull-up or pull-down device to an input channel. A subsequent conversion can then confirm the expected modified signal level. This allows to check the proper operation of the multiplexer.

Converter Diagnostics (CD) connects an alternate signal to the converter. A subsequent conversion can then confirm the proper operation of the converter.

19.3 Configuration of General Functions

While many parameters can be selected individually for each channel, source, or group, some adjustments are valid for the whole ADC cluster:

- Clock control
- Kernel synchronization
- External multiplexer control
- Test functions

19.3.1 General Clocking Scheme and Control

The A/D Converters of the XMC4500 are supplied with a global clock signal from the system f_{ADC}. Two clock signals are derived from this input and are distributed to all converters. The global configuration register defines common clock bases for all converters of the cluster. This ensures deterministic behavior of converters that shall operate in parallel.

The analog converter clock f_{ADCI} determines the performance of the converters and must be selected to comply with the specification given in the Data Sheet.

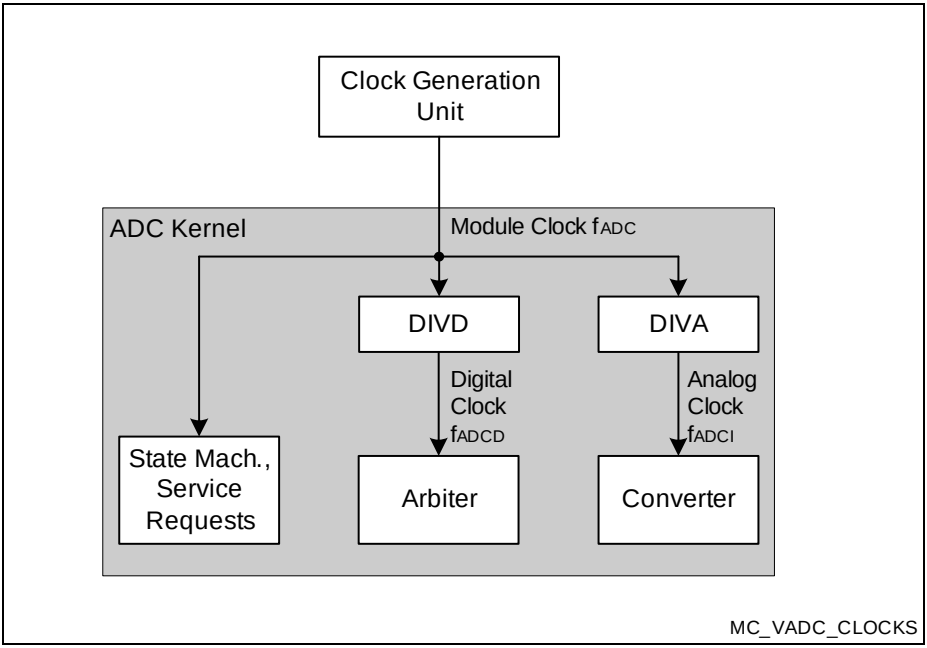


Figure 19-4 Clock Signal Summary

19.3.2 Priority Channel Assignment

Each channel of a group can be assigned to this group's request sources and is then regarded as a priority channel. An assigned priority channel can only be converted by its own group's request sources. A not assigned channel can also be converted by the background request source.

19.4 Module Activation and Power Saving

The analog converter of the ADC draws a permanent current during its operation. It can be deactivated between conversions to reduce the consumed overall energy.

The operating mode is determined by bitfield **GxARBCFG (x = 0 - 3)**.ANONS:

- ANONS = 11_B: **Normal Operation**
  The converter is active, conversions are started immediately.
  Requires no wakeup time.
- ANONS = 10_B or 01_B: **Reserved**
- ANONS = 00_B: **Converter switched Off** (default after reset)
  The converter is switched off. Furthermore, digital logic blocks are set to their initial state. If the arbiter is currently running, it completes the actual arbitration round and then stops.
  Before starting a conversion, select the active mode for ANONS.
  Requires the wakeup time (see below).

Wakeup Time from Analog Powerdown

When the converter is activated, it needs a certain wakeup time to settle before a conversion can be properly executed. This wakeup time can be established by waiting the required period before starting a conversion, or by adding it to the intended sample time.

The wakeup time is approximately 15 µs.
Exact numbers can be found in the respective Data Sheets.

Note: The wakeup time is also required after initially enabling the converter.

Calibration

Calibration automatically compensates deviations caused by process, temperature, and voltage variations. This ensures precise results throughout the operation time.

An initial start-up calibration is required once after a reset for all calibrated converters and is triggered globally. All calibrated converters must be enabled (ANONS = 11_B) before initiating the start-up calibration. Conversions may be started after the initial calibration sequence. This is indicated by bit CAL = 0_B.

After that, postcalibration cycles will compensate the effects of drifting parameters.

19.5 Conversion Request Generation

The conversion request unit of a group autonomously handles the generation of conversion requests. Three request sources (2 group-specific sources and the background source) can generate requests for the conversion of an analog channel. The arbiter resolves concurrent requests and selects the channel to be converted next.

Upon a trigger event, the request source requests the conversion of a certain analog input channel or a sequence of channels.

- **Software triggers**
  directly activate the respective request source.
- **External triggers**
  synchronize the request source activation with external events, such as a trigger pulse from a timer generating a PWM signal or from a port pin.

Application software selects the trigger, the channel(s) to be converted, and the request source priority. A request source can also be activated directly by software without requiring an external trigger.

The arbiter regularly scans the request sources for pending conversion requests and selects the conversion request with the highest priority. This conversion request is then forwarded to the converter to start the conversion of the requested channel.

Each request source can operate in single-shot or in continuous mode:

- **In single-shot mode**,
  the programmed conversion (sequence) is requested once after being triggered. A subsequent conversion (sequence) must be triggered again.
- **In continuous mode**,
  the programmed conversion (sequence) is automatically requested repeatedly after being triggered once.

For each request source, external triggers are generated from one of 16 selectable trigger inputs (REQTRx[P:A]) and from one of 16 selectable gating inputs (REQGTx[P:A]). The available trigger signals for the XMC4500 are listed in **Section 19.14.3**.

*Note: **Figure 19-3 “Conversion Request Unit” on Page 19-6** summarizes the request sources.*

Two types of requests sources are available:

- **A queued source** can issue conversion requests for an arbitrary sequence of input channels. The channel numbers for this sequence can be freely programmed[1]. This supports application-specific conversion sequences that cannot be covered by a channel scan source. Also, multiple conversions of the same channel within a sequence are supported.
  A queued source converts a series of input channels permanently or on a regular time base. For example, if programmed with medium priority, some input channels can be converted upon a specified event (e.g. synchronized to a PWM). Conversions of lower priority sources are suspended in the meantime.
  Request source 0 is a group-specific 8-stage queued source.
- **A channel scan source** can issue conversion requests for a coherent sequence of input channels. This sequence begins with the highest enabled channel number and continues towards lower channel numbers. All available channels[1] can be enabled for the scan sequence. Each channel is converted once per sequence.
  A scan source converts a series of input channels permanently or on a regular time base. For example, if programmed with low priority, some input channels can be scanned in a background task to update information that is not time-critical.
  – Request source 1 is a group-specific channel scan source.
  – Request source 2 is a global channel scan source (background source).
  The background source can request all channels of all groups.

1) The availablity of input channels depends on the package of the used product type. A summary can be found in **Section 19.14.2**.
The background source can only request non-priority channels, i.e. channels that are not selected in registers GxCHASS. Priority channels are reserved for the group-specific request sources 0 and 1.

19.5.1 Queued Request Source Handling

A queued request source supports short conversion sequences (up to 8) of arbitrary channels (contrary to a scan request source with a fixed conversion order for the enabled channels). The programmed sequence is stored in a queue buffer (based on a FIFO mechanism). The requested channel numbers are entered via the queue input, while queue stage 0 defines the channel to be converted next.

A conversion request is only issued to the request source arbiter if a valid entry is stored in queue stage 0.

If the arbiter aborts a conversion triggered by a queued request source due to higher priority requests, the corresponding conversion parameters are automatically saved in the backup stage. This ensures that an aborted conversion is not lost but takes part in the next arbitration round (before stage 0).

The trigger and gating unit generates trigger events from the selected external (outside the ADC) trigger and gating signals. For example, a timer unit can issue a request signal to synchronize conversions to PWM events.

Trigger events start a queued sequence and can be generated either via software or via the selected hardware triggers. The occurrence of a trigger event is indicated by bit QSRx.EV. This flag is cleared when the corresponding conversion is started or by writing to bit QMRx.CEV.

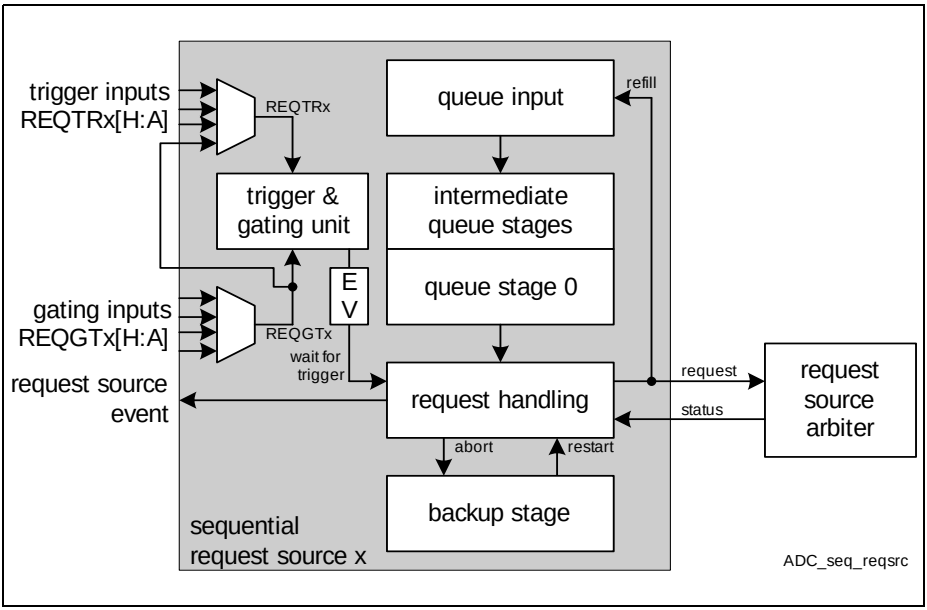


Figure 19-5 Queued Request Source

A sequence is defined by entering conversion requests into the queue input register (**GxQINR0 (x = 0 - 3)**). Each entry selects the channel to be converted and can enable an external trigger, generation of an interrupt, and an automatic refill (i.e. copy this entry to the top of the queue after conversion). The entries are stored in the queue buffer stages.

The content of stage 0 (**GxQ0R0 (x = 0 - 3)**) selects the channel to be converted next. When the requested conversion is started, the contents of this queue stage is invalidated and copied to the backup stage. Then the next queue entry can be handled (if available).

Note: The contents of the queue stages cannot be modified directly, but only by writing to the queue input or by flushing the queue.
*The current status of the queue is shown in register **GxQSR0 (x = 0 - 3)**.*
If all queue entries have automatic refill selected, the defined conversion sequence can be repeated without re-programming.

Properties of the Queued Request Source

Queued request source 0 provides 8 buffer stages and can handle sequences of up to 8 input channel entries. It supports short application-specific conversion sequences, especially for timing-critical sequences containing also multiple conversions of the same channel.

Queued Source Operation

Configure the queued request source by executing the following actions:

- Define the sequence by writing the entries to the queue input **GxQINR0 (x = 0 - 3)**. Initialize the complete sequence before enabling the request source, because with enabled refill feature, software writes to QINRx are not allowed.
- If hardware trigger or gating is desired, select the appropriate trigger and gating inputs and the proper transitions by programming **GxQCTRL0 (x = 0 - 3)**.
  Enable the trigger and select the gating mode by programming bitfield ENGT in register **GxQMR0 (x = 0 - 3)**.[1)]
- Enable the corresponding arbitration slot (0) to accept conversion requests from the queued source (see register **GxARBPR (x = 0 - 3)**).

Start a queued sequence by generating a trigger event:

- If a hardware trigger is selected and enabled, generate the configured transition at the selected input signal, e.g. from a timer or an input pin.
- Generate a software trigger event by setting GxQMR0.TREV = 1.

1) If PDOUT signals from the ERU are used, initialize the ERU accordingly before enabling the gate inputs to avoid un expected signal transitions.

- Write a new entry to the queue input of an empty queue. This leads to a (new) valid queue entry that is forwarded to queue stage 0 and starts a conversion request (if enabled by GxQMR0.ENGT and without waiting for an external trigger).

Note: If the refill mechanism is activated, a processed entry is automatically reloaded into the queue. This permanently repeats the respective sequence (autoscan). In this case, do not write to the queue input while the queued source is running. Write operations to a completely filled queue are ignored.

Stop or abort an ongoing queued sequence by executing the following actions:

- If external gating is enabled, switch the gating signal to the defined inactive level. This does not modify the queue entries, but only prevents issuing conversion requests to the arbiter.
- Disable the corresponding arbitration slot (0) in the arbiter. This does not modify the queue entries, but only prevents the arbiter from accepting requests from the request handling block.
- Disable the queued source by clearing bitfield ENGT = 00_B.
  - Invalidate the next pending queue entry by setting bit GxQMR0.CLRV = 1. If the backup stage contains a valid entry, this one is invalidated, otherwise stage 0 is invalidated.
  - Remove all entries from the queue by setting bit GxQMR0.FLUSH = 1.

Queue Request Source Events and Service Requests

A request source event of a queued source occurs when a conversion is finished. A source event service request can be generated based on a request source event according to the structure shown in **Figure 19-6**. If a request source event is detected, it sets the corresponding indication flag in register **GxSEFLAG (x = 0 - 3)**. These flags can also be set by writing a 1 to the corresponding bit position, whereas writing 0 has no effect. The indication flags can be cleared by SW by writing a 1 to the corresponding bit position in register **GxSEFCLR (x = 0 - 3)**.
The interrupt enable bit is taken from stage 0 for a normal sequential conversion, or from the backup stage for a repeated conversion after an abort.

The service request output line SRx that is selected by the request source event interrupt node pointer bitfields in register **GxSEVNP (x = 0 - 3)** becomes activated each time the related request source event is detected (and enabled by GxQ0R0.ENSI, or GxQBUR0.ENSI respectively) or the related bit position in register **GxSEFLAG (x = 0 - 3)** is written with a 1 (this write action simulates a request source event).

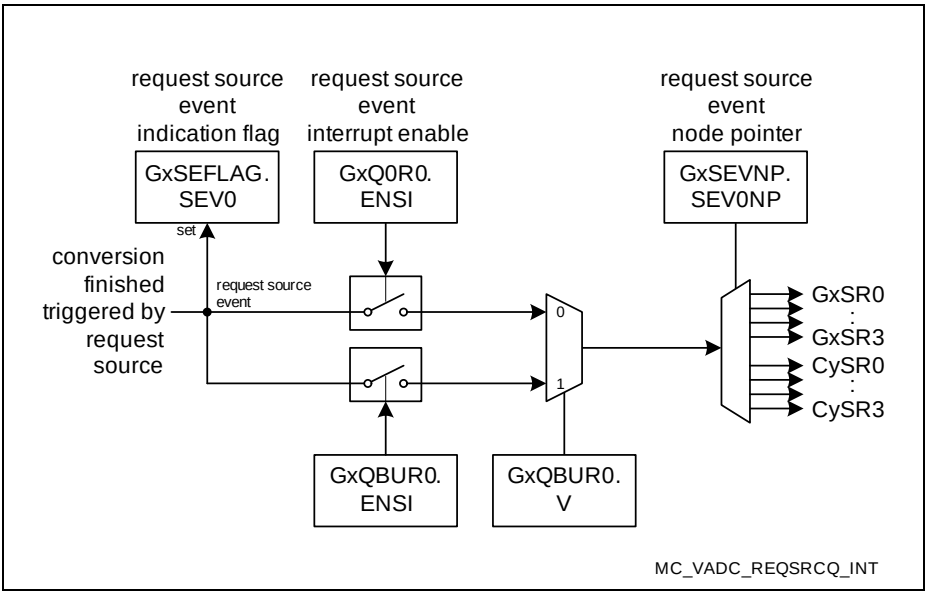


Figure 19-6 Interrupt Generation of a Queued Request Source

19.5.2 Channel Scan Request Source Handling

The VADC provides two types of channel scan sources:

- Source 1: Group scan source
  This scan source can request all channels of the corresponding group.
- Source 2: Background scan source
  This scan source can request all channels of all groups.
  Priority channels selected in registers **GxCHASS (x = 0 - 3)** cannot take part in background conversion sequences.

Both sources operate in the same way and provide the same register interface. The background source provides more request/pending bits because it can request all channels of all groups.

Each analog input channel can be included in or excluded from the scan sequence by setting or clearing the corresponding channel select bit in register **GxASSEL (x = 0 - 3)** or **BRSSELx (x = 0 - 3)**. The programmed register value remains unchanged by an ongoing scan sequence. The scan sequence starts with the highest enabled channel number and continues towards lower channel numbers.

Upon a load event, the request pattern is transferred to the pending bits in register **GxASPND (x = 0 - 3)** or **BRSPNDx (x = 0 - 3)**. The pending conversion requests indicate

which input channels are to be converted in an ongoing scan sequence. Each conversion start that was triggered by the scan request source, automatically clears the corresponding pending bit. If the last conversion triggered by the scan source is finished and all pending bits are cleared, the current scan sequence is considered finished and a request source event is generated.

A conversion request is only issued to the request source arbiter if at least one pending bit is set.

If the arbiter aborts a conversion triggered by the scan request source due to higher priority requests, the corresponding pending bit is automatically set. This ensures that an aborted conversion is not lost but takes part in the next arbitration round.

The trigger and gating unit generates load events from the selected external (outside the ADC) trigger and gating signals. For example, a timer unit can issue a request signal to synchronize conversions to PWM events.

Load events start a scan sequence and can be generated either via software or via the selected hardware triggers. The request source event can also generate an automatic load event, so the programmed sequence is automatically repeated.

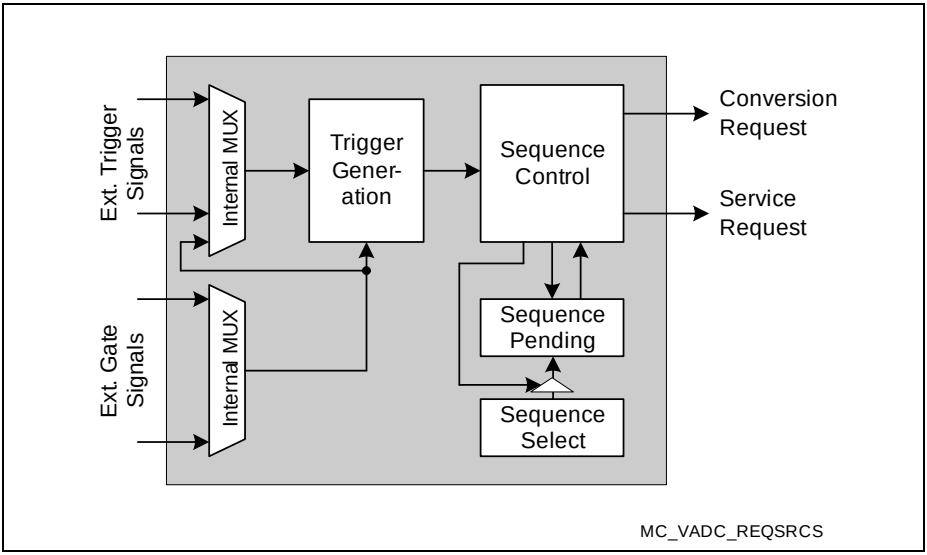


Figure 19-7 Scan Request Source

Scan Source Operation

Configure the scan request source by executing the following actions:

- Select the input channels for the sequence by programming **GxASSEL (x = 0 - 3)** or **BRSSELx (x = 0 - 3)**
- If hardware trigger or gating is desired, select the appropriate trigger and gating inputs and the proper signal transitions by programming **GxASCTRL (x = 0 - 3)** or **BRSCTRL**. Enable the trigger and select the gating mode by programming **GxASMR (x = 0 - 3)** or **BRSMR**.[1)]
- Define the load event operation (handling of pending bits, autoscan mode) by programming **GxASMR (x = 0 - 3)** or **BRSMR**.
  A load event with bit LDM = 0 copies the content of **GxASSEL (x = 0 - 3)** or **BRSSELx (x = 0 - 3)** to **GxASPND (x = 0 - 3)** or **BRSPNDx (x = 0 - 3)** (overwrite mode). This starts a new scan sequence and aborts any pending conversions from a previous scan sequence.
  A load event with bit LDM = 1 OR-combines the content of **GxASSEL (x = 0 - 3)** or **BRSSELx (x = 0 - 3)** to **GxASPND (x = 0 - 3)** or **BRSPNDx (x = 0 - 3)** (combine mode). This starts a scan sequence that includes pending conversions from a prvious scan sequence.
- Enable the corresponding arbitration slot (1) to accept conversion requests from the channel scan source (see register **GxARBPR (x = 0 - 3)**).

Start a channel scan sequence by generating a load event:

- If a hardware trigger is selected and enabled, generate the configured transition at the selected input signal, e.g. from a timer or an input pin.
- Generate a software load event by setting LDEV = 1 (**GxASMR (x = 0 - 3)** or **BRSMR**).
- Generate a load event by writing the scan pattern directly to the pending bits in **GxASPND (x = 0 - 3)** or **BRSPNDx (x = 0 - 3)**. The pattern is copied to **GxASSEL (x = 0 - 3)** or **BRSSELx (x = 0 - 3)** and a load event is generated automatically.
  In this case, a scan sequence can be defined and started with a single data write action, e.g. under PEC control (provided that the pattern fits into one register).

Note: If autoscan is enabled, a load event is generated automatically each time a request source event occurs when the scan sequence has finished. This permanently repeats the defined scan sequence (autoscan).

Stop or abort an ongoing scan sequence by executing the following actions:

- If external gating is enabled, switch the gating signal to the defined inactive level. This does not modify the conversion pending bits, but only prevents issuing conversion requests to the arbiter.

1) If PDOUT signals from the ERU are used, initialize the ERU accordingly before enabling the gate inputs to avoid un expected signal transitions.

- Disable the corresponding arbitration slot (1 or 2) in the arbiter. This does not modify the contents of the conversion pending bits, but only prevents the arbiter from accepting requests from the request handling block.
- Disable the channel scan source by clearing bitfield ENGT = 00_B. Clear the pending request bits by setting bit CLRPND = 1 (**GxASMR (x = 0 - 3)** or **BRSMR**).

Scan Request Source Events and Service Requests

A request source event of a scan source occurs if the last conversion of a scan sequence is finished (all pending bits = 0). A request source event interrupt can be generated based on a request source event. If a request source event is detected, it sets the corresponding indication flag in register **GxSEFLAG (x = 0 - 3)**. These flags can also be set by writing a 1 to the corresponding bit position, whereas writing 0 has no effect.

The service request output SRx that is selected by the request source event interrupt node pointer bitfields in register **GxSEVNP (x = 0 - 3)** becomes activated each time the related request source event is detected (and enabled by ENSI) or the related bit position in register **GxSEFLAG (x = 0 - 3)** is written with a 1 (this write action simulates a request source event).

The indication flags can be cleared by SW by writing a 1 to the corresponding bit position in register **GxSEFCLR (x = 0 - 3)**.[1)]

1) Please refer to **"Service Request Generation" on Page 19-55**.

19.6 Request Source Arbitration

The request source arbiter regularly polls the request sources, one after the other, for pending conversion requests. Each request source is assigned to a certain time slot within an arbitration round, called arbitration slot. The duration of an arbitration slot is user-configurable via register **GLOBCFG**.

The priority of each request source is user-configurable via register **GxARBPR (x = 0 - 3)**, so the arbiter can select the next channel to be converted, in the case of concurrent requests from multiple sources, according to the application requirements.

An unused arbitration slot is considered empty and does not take part in the arbitration. After reset, all slots are disabled and must be enabled (register **GxARBPR (x = 0 - 3)**) to take part in the arbitration process.

Figure 19-8 summarizes the arbitration sequence. An arbitration round consists of one arbitration slot for each available request source. The synchronization source is always evaluated in the last slot and has a higher priority than all other sources. At the end of each arbitration round, the arbiter has determined the highest priority conversion request.

If a conversion is started in an arbitration round, this arbitration round does not deliver an arbitration winner. In the XMC4500, the following request sources are available:

- Arbitration slot 0: **Group Queued source**, 8-stage sequences in arbitrary order
- Arbitration slot 1: **Group Scan source**, sequences in defined order within group
- Arbitration slot 2: **Background Scan source**, sequences in defined order, all groups
- Last arbitration slot: **Synchronization source**, synchronized conversion requests from another ADC kernel (always handled with the highest priority in a synchronization slave kernel).

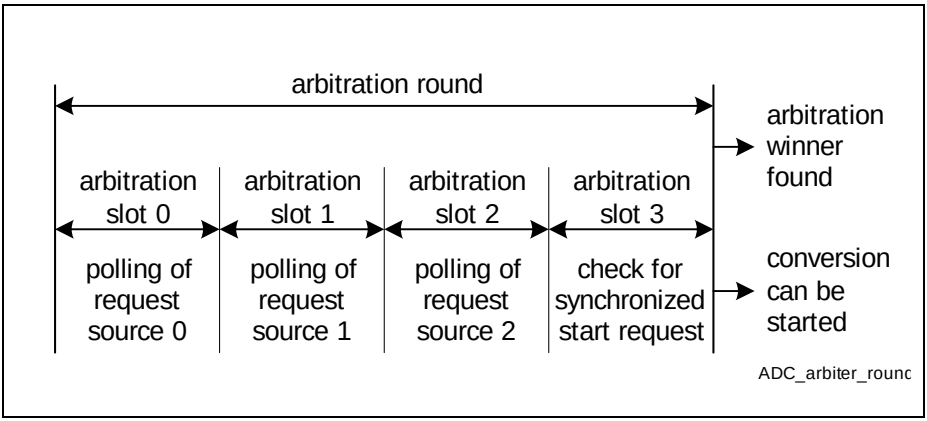


Figure 19-8 Arbitration Round with 4 Arbitration Slots

19.6.1 Arbiter Operation and Configuration

The timing of the arbiter (i.e. of an arbitration round) is determined by the number of arbitration slots within an arbitration round and by the duration of an arbitration slot.

An arbitration round consist of 4…20 arbitration slots (defined by bitfield **GxARBCFG (x = 0 - 3)**.ARBRND). 4 slots are sufficient for the XMC4500, more can be programmed to obtain the same arbiter timing for different products.

The duration of an arbitration slot is configurable t_{Slot} = (DIVD+1) / f_{ADC}.
The duration of an arbitration round, therefore, is t_{ARB} = 4 × t_{Slot}.

The period of the arbitration round introduces a timing granularity to detect an incoming conversion request signal and the earliest point to start the related conversion. This granularity can introduce a jitter of maximum one arbitration round. The jitter can be reduced by minimizing the period of an arbitration round.

To achieve a reproducible reaction time (constant delay without jitter) between the trigger event of a conversion request (e.g. by a timer unit or due to an external event) and the start of the related conversion, mainly the following two options exist. For both options, the converter has to be idle and other conversion requests must not be pending for at least one arbiter round before the trigger event occurs:

- If bit **GxARBCFG (x = 0 - 3)**.ARBM = 0, the **arbiter runs permanently**. In this mode, synchronized conversions of more than one ADC kernel are possible.[1)]
  The trigger for a conversion request has to be generated synchronously to the arbiter timing. Incoming triggers should have exactly n-times the granularity of the arbiter (n = 1, 2, 3,...). In order to allow some flexibility, the duration of an arbitration slot can be programmed in cycles of f_{ADC}.
- If bit **GxARBCFG (x = 0 - 3)**.ARBM = 1, the **arbiter stops after an arbitration round** when no conversion request have been found pending any more. The arbiter is started again if at least one enabled request source indicates a pending conversion request. The trigger for a conversion request does not need to be synchronous to the arbiter timing.
  In this mode, parallel conversions are not possible for synchronization slave kernels.

Each request source has a configurable priority, so the arbiter can resolve concurrent conversion requests from different sources. The request with the highest priority is selected for conversion. These priorities can be adapted to the requirements of a given application (see register **GxARBPR (x = 0 - 3)**).

The **Conversion Start Mode** determines the handling of the conversion request that has won the arbitration.

1) For more information, please refer to **"Synchronization of Conversions" on Page 19-46**.

19.6.2 Conversion Start Mode

When the arbiter has selected the request to be converted next, the handling of this channel depends on the current activity of the converter:

- Converter is currently idle: the conversion of the arbitration winner is started immediately.
- Current conversion has same or higher priority: the current conversion is completed, the conversion of the arbitration winner is started after that.
- Current conversion has lower priority: the action is user-configurable:
  - **Wait-for-start mode**: the current conversion is completed, the conversion of the arbitration winner is started after that. This mode provides maximum throughput, but can produce a jitter for the higher priority conversion.
    Example in **Figure 19-9**:
    Conversion A is requested (t1) and started (t2). Conversion B is then requested (t3), but started only after completion of conversion A (t4).
  - **Cancel-inject-repeat mode**: the current conversion is aborted, the conversion of the arbitration winner is started after the abortion (3 f_{ADC} cycles).
    The aborted conversion request is restored in the corresponding request source and takes part again in the next arbitration round. This mode provides minimum jitter for the higher priority conversions, but reduces the overall throughput.
    Example in **Figure 19-9**:
    Conversion A is requested (t6) and started (t7). Conversion B is then requested (t8) and started (t9), while conversion A is aborted but requested again. When conversion B is complete (t10), conversion A is restarted.
    Exception: If both requests target the same result register with wait-for-read mode active (see **Section 19.8.3**), the current conversion cannot be aborted.

Note: A cancelled conversion can be repeated automatically in each case, or it can be discarded if it was cancelled. This is selected for each source by bit RPTDIS in the corresponding source's mode register.

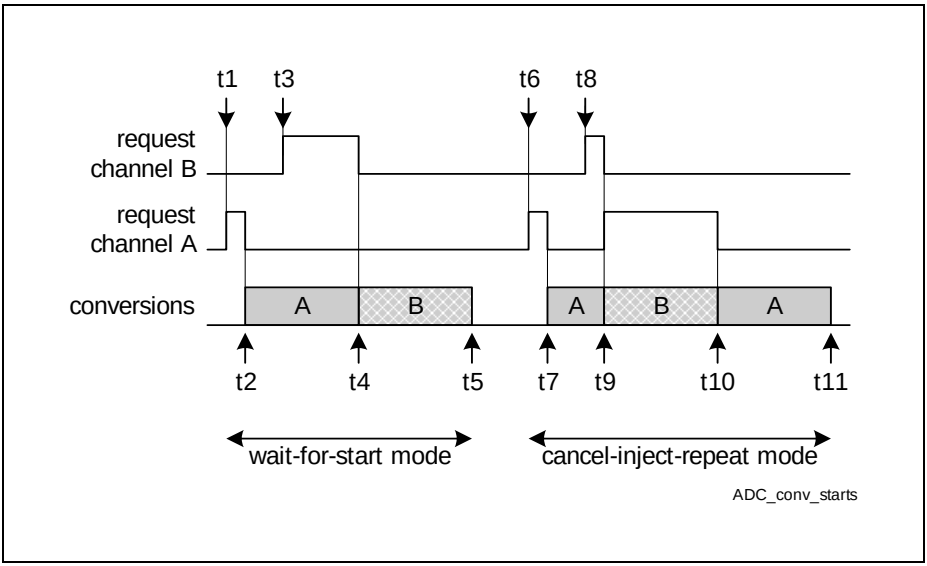


Figure 19-9 Conversion Start Modes

The conversion start mode can be individually programmed for each request source by bits in register **GxARBPR (x = 0 - 3)** and is applied to all channels requested by the source. In this example, channel A is issued by a request source with a lower priority than the request source requesting the conversion of channel B.

19.7 Analog Input Channel Configuration

For each analog input channel a number of parameters can be configured that control the conversion of this channel. The channel control registers define the following parameters:

- **Channel Parameters**: The sample time for this channel and the data width of the result are defined via input classes. Each channel can select one of two classes of its own group or one of two global classes.
- **Reference selection**: an alternate reference voltage can be selected for most channels (exceptions are marked in **Section 19.14.2**)
- **Result target**: The conversion result values are stored either in a group-specific result register or in the global result register. The group-specific result registers are selected channel-specific, selected by bitfield RESREG in register **G0CHCTRy (y = 0 - 7)** etc.
- **Result position**: The result values can be stored left-aligned or right-aligned. The exact position depends also on the configured result width and on the data accumulation mode.
  See also **Figure 19-15 “Result Storage Options” on Page 19-35**.
- **Compare with Standard Conversions (Limit Checking)**: Channel events can be generated whenever a new result value becomes available. Channel event generation can be restricted to values that lie inside or outside a user-configurable band.
  In Fast Compare Mode, channel events can be generated depending on the transitions of the (1-bit) result.
- **Broken Wire Detection**: This safety feature can detect a missing connection to an analog signal source (sensor).
- **Synchronization of Conversions**: Synchronized conversions are executed at the same time on several converters.

The **Alias Feature** redirects conversion requests for channels CH0 and/or CH1 to other channels.

19.7.1 Channel Parameters

Each analog input channel is configured by its associated channel control register.

*Note: For the safety feature “Broken Wire Detection”, refer to **Section 19.10.1**.*

The following features can be defined for each channel:

- The conversion class defines the result width and the sample time
- Generation of channel events and the result value band, if used
- Target of the result defining the target register and the position within the register

The group-specific input class registers define the sample time and data conversion mode for each channel of the respective group that selects them via bitfield ICLSEL in its channel control register GxCHCTRx.

The global input class registers define the sample time and data conversion mode for each channel of any group that selects them via bitfield ICLSEL in its channel control register GxCHCTRx.

19.7.2 Conversion Timing

The total time required for a conversion depends on several user-definable factors:

- The ADC conversion clock frequency, where $f_{ADCI} = f_{ADC} / (DIVA+1)$[1)]
- The selected sample time, where $t_S = (2 + STC) \times t_{ADCI}$
  (STC = additional sample time, see also **Table 19-9**)
- The selected operating mode (normal conversion / fast compare mode)
- The result width N (8/10/12 bits) for normal conversions
- The post-calibration time PC, if selected (PC = 2, otherwise 0)
- The selected duration of the MSB conversion (DM = 0 or 1)
- Synchronization steps done at module clock speed

The conversion time is the sum of sample time, conversion steps, and synchronization. It can be computed with the following formulas:

Standard conversions: $t_{CN} = (2 + STC + N + DM + PC) \times t_{ADCI} + 2 \times t_{ADC}$

Fast compare mode: $t_{CN} = (2 + STC + 2) \times t_{ADCI} + 2 \times t_{ADC}$

The frequency at which conversions are triggered also depends on several configurable factors:

- The selected conversion time, according to the input class definitions. For conversions using an external multiplexer, also the extended sample times count.
- Delays induced by cancelled conversions that must be repeated.
- Delays due to equidistant sampling of other channels.
- The configured arbitration cycle time.
- The frequency of external trigger signals, if enabled.

Timing Examples

System assumptions:
f_{ADC} = 120 MHz i.e. t_{ADC} = 8.3 ns, DIVA = 3, f_{ADCI} = 30 MHz i.e. t_{ADCI} = 33.3 ns

According to the given formulas the following minimum conversion times can be achieved:

12-bit calibrated conversion:
$t_{CN12C} = (2 + 12 + 2) \times t_{ADCI} + 2 \times t_{ADC} = 16 \times 33.3\ ns + 2 \times 8.3\ ns = 550\ ns$

10-bit uncalibrated conversion:
$t_{CN10} = (2 + 10) \times t_{ADCI} + 2 \times t_{ADC} = 12 \times 33.3\ ns + 2 \times 8.3\ ns = 417\ ns$

Fast comparison:
$t_{FCM} = (2 + 2) \times t_{ADCI} + 2 \times t_{ADC} = 4 \times 33.3\ ns + 2 \times 8.3\ ns = 150\ ns$

1) The minimum prescaler factor for calibrated converters is 2.

19.7.3 Alias Feature

The Alias Feature redirects conversion requests for channels CH0 and/or CH1 to other channel numbers. This feature can be used to trigger conversions of the same input channel by independent events and to store the conversion results in different result registers.

- The same signal can be measured twice without the need to read out the conversion result to avoid data loss. This allows triggering both conversions quickly one after the other and being independent from CPU/DMA service request latency.
- The sensor signal is connected to only one analog input (instead of two analog inputs). This saves input pins in low-cost applications and only the leakage of one input has to be considered in the error calculation.
- Even if the analog input CH0 is used as alternative reference (see **Figure 19-10**), the internal trigger and data handling features for channel CH0 can be used.
- The channel settings for both conversions can be different (boundary values, service requests, etc.).

In typical low-cost AC-drive applications, only one common current sensor is used to determine the phase currents. Depending on the applied PWM pattern, the measured value has different meanings and the sample points have to be precisely located in the PWM period. **Figure 19-10** shows an example where the sensor signal is connected to one input channel (CHx) but two conversions are triggered for two different channels (CHx and CH0). With the alias feature, a conversion request for CH0 leads to a conversion of the analog input CHx instead of CH0, but taking into account the settings for CH0. Although the same analog input (CHx) has been measured, the conversion results can be stored and read out from the result registers RESx (conversion triggered for CHx) and RESy (conversion triggered for CH0). Additionally, different interrupts or limit boundaries can be selected, enabled or disabled.

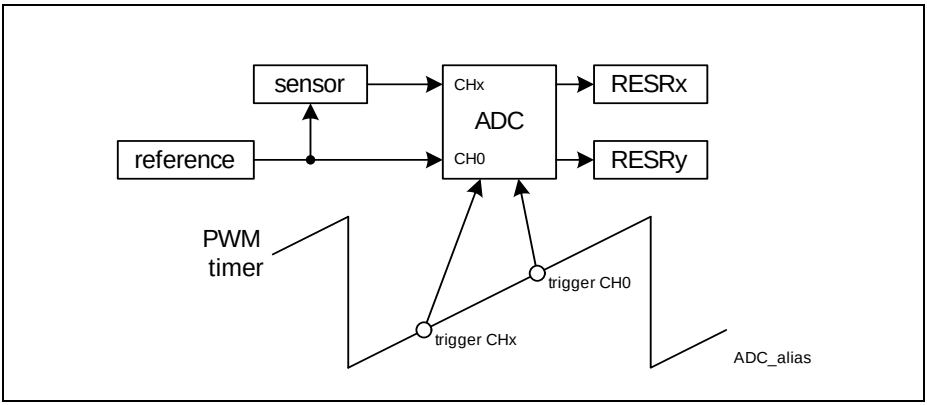


Figure 19-10 Alias Feature

19.7.4 Conversion Modes

A conversion can be executed in several ways. The conversion mode is selected according to the requested resolution of the digital result and according to the acceptable conversion time (**Section 19.7.2**).
Use bitfield CMS/CME in register **GxICLASS0 (x = 0 - 3)** etc. to select a mode.

Standard Conversions

A standard conversion returns a result value with a predefined resolution. 8-bit, 10-bit, and 12-bit resolution can be selected.

These result values can be accumulated, filtered, or used for digital limit checking.

Note: The calibrated converters can operate with and without post-calibration.

Fast Compare Mode

In Fast Compare Mode, the selected input voltage is directly compared with a digital value that is stored in the corresponding result register. This compare operation returns a binary result indicating if the compared input voltage is above or below the given reference value. This result is generated quickly and thus supports monitoring of boundary values.

Fast Compare Mode uses a 10-bit compare value stored left-aligned at bit position 11. Separate positive and negative delta values define an arbitrary hysteresis band.

Selecting Compare Values

Values for digital or analog compare operations can be selected from several sources. The separate GxBOUND registers provide software-defined compare values.

In Fast Compare Mode, the result registers provide the compare value while the bitfields of local GxBOUND registers define positive and negative delta values.

19.7.5 Compare with Standard Conversions (Limit Checking)

The limit checking mechanism can automatically compare each digital conversion result to an upper and a lower boundary value. A channel event can then be generated when the result of a conversion/comparison is inside or outside a user-defined band (see bitfield CHEVMODE and **Figure 19-11**).

This feature supports automatic range monitoring and minimizes the CPU load by issuing service requests only under certain predefined conditions.

Note: Channel events can also be generated for each result value (ignoring the band) or they can be suppressed completely.

The boundary values to which results are compared can be selected from several sources (see register GxCHCTRy).

Bitfields BNDSELU and BNDSELL select the valid upper/lower boundary value either from the group-specific boundary register **GxBOUND (x = 0 - 3)** or from the global boundary register **GLOBBOUND**. The group boundary register can be selected for each channel of the respective group, the global boundary register can be selected by each available channel.

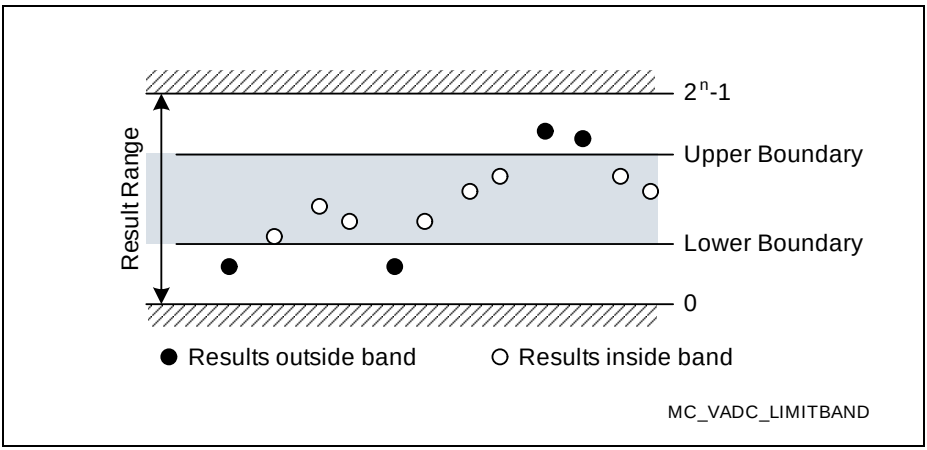


Figure 19-11 Result Monitoring through Limit Checking

A result value is considered inside the defined band when both of the following conditions are true:

- the value is less than or equal to the selected upper boundary
- the value is greater than or equal to the selected lower boundary

The result range can also be divided into two areas:
To select the lower part as valid band, set the lower boundary to the minimum value (000_H) and set the upper boundary to the highest intended value.
To select the upper part as valid band, set the upper boundary to the maximum value (FFF_H) and set the lower boundary to the lowest intended value.

19.7.6 Utilizing Fast Compare Mode

In Fast Compare Mode, the input signal is directly compared to a value in the associated result register. This comparison just provides a binary result (above/below). If the exact result value is not required, this saves conversion time. A channel event can then be generated when the input signal becomes higher (or lower) than the compare value (see bitfield CHEVMODE and **Figure 19-12**).

The compare value in Fast Compare Mode is taken from the result register. Bitfields BOUNDARY1 and BOUNDARY0 in register **GxBOUND (x = 0 - 3)** define delta limits in this case. These deltas are added to (or subtracted from) the original compare value and allow defining an arbitrary hysteresis band.

The actual used compare value depends on the Fast Compare Result FCR (see registers **G0RESy (y = 0 - 15)**, etc.):

GxRESy.FCR = 0: reference value + upper delta
(GxRESy.RESULT + GxBOUND.BOUNDARY0)
GxRESy.FCR = 1: reference value - lower delta
(GxRESy.RESULT - GxBOUND.BOUNDARY1)

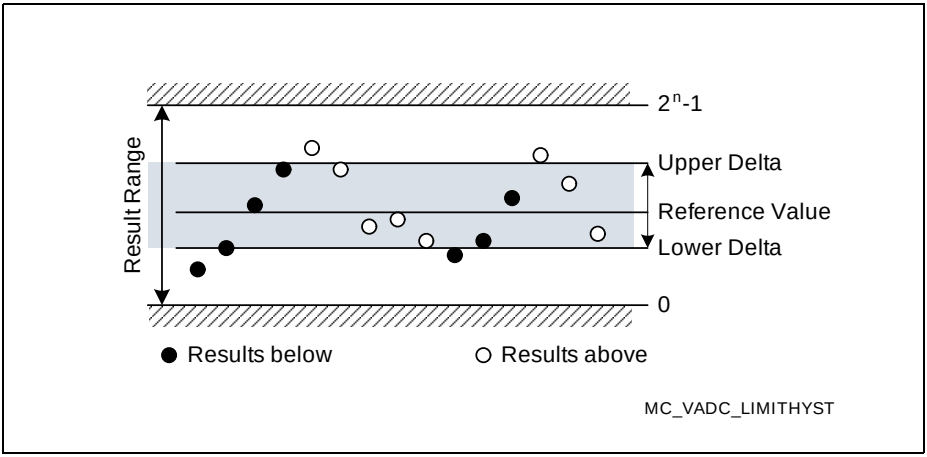


Figure 19-12 Result Monitoring through Compare with Hysteresis

19.7.7 Boundary Flag Control

Both limit checking mechanisms can be configured to automatically control the boundary flags. These boundary flags are also available as control signals for other modules. The flags can be set or cleared when the defined level is exceeded.

For standard conversions, a boundary flag will be set when the conversion result is above the defined band, and will be cleared when the conversion result is below the defined band.

The band between the two boundary values defines a hysteresis for setting/clearing the boundary flags.

Using this feature on three channels that monitor linear hall elements can produce signals to feed the three hall position inputs of a CCU6x unit.

In Fast Compare Mode, a boundary flag reflects the result of the comparisons, i.e. it will be set when the compared signal level is above the compare value, and will be cleared when the signal level is below the compare value.

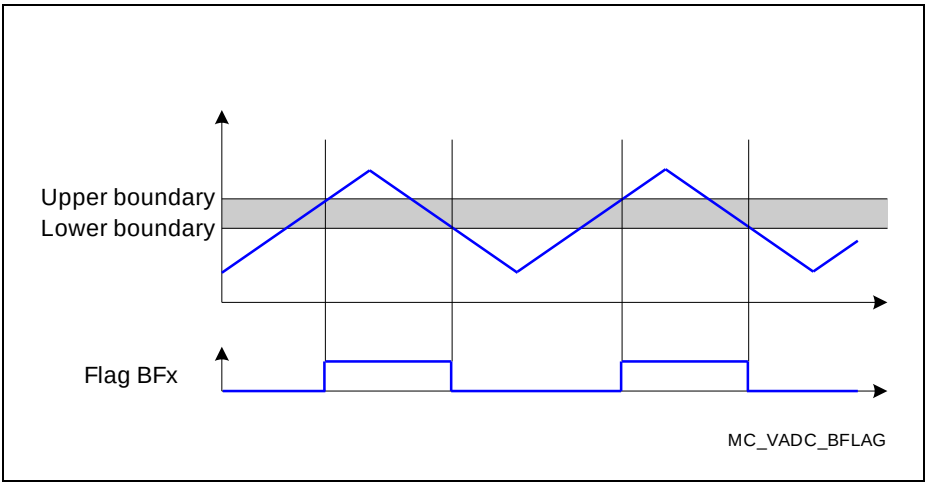


Figure 19-13 Boundary Flag Switching

19.8 Conversion Result Handling

The A/D converters can preprocess the conversions result data to a certain extent before storing them for retrieval by the CPU or a DMA channel. This supports the subsequent handling of result data by the application software.

Conversion result handling comprises the following functions:

- **Storage of Conversion Results** to user-configurable registers
- **Data Alignment** according to result width and endianess
- **Wait-for-Read Mode** to avoid loss of data
- **Result Event Generation**
- Data reduction or anti-aliasing filtering (see **Section 19.8.6**)

19.8.1 Storage of Conversion Results

The conversion result values of a certain group can be stored in one of the 16 associated group result registers or in the common global result register (can be used, for example, for the channels of the background source (see **Selecting a Result Register**).

This structure provides different locations for the conversion results of different sets of channels. Depending on the application needs (data reduction, auto-scan, alias feature, etc.), the user can distribute the conversion results to minimize CPU load and/or optimize the performance of DMA transfers.

Each result register has an individual data valid flag (VF) associated with it. This flag indicates when “new” valid data has been stored in the corresponding result register and can be read out.

For standard conversions, result values are available in bitfield RESULT. Conversions in Fast Compare Mode use bitfield RESULT for the reference value, so the result of the operation is stored in bit FCR.

Result registers can be read via two different views. These views use different addresses but access the same register data:

- When a result register is read via the **application view**, the corresponding valid flag is automatically cleared when the result is read. This provides an easy handshake between result generation and retrieval. This also supports wait-for-read mode.
- When a result register is read via the **debug view**, the corresponding valid flag remains unchanged when the result is read. This supports debugging by delivering the result value without disturbing the handshake with the application.

The application can retrieve conversion results through several result registers:

- Group result register:
  Returns the result value and the channel number
- Global result register:
  Returns the result value and the channel number and the group number

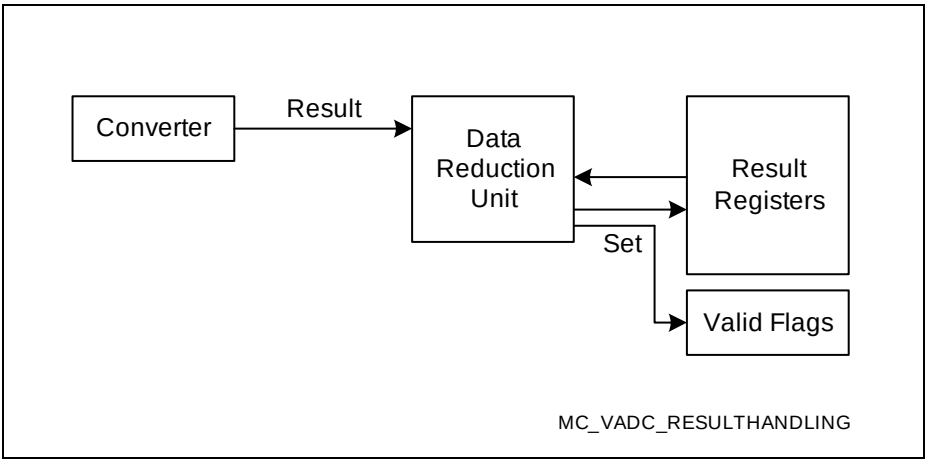


Figure 19-14 Conversion Result Storage

Selecting a Result Register

Conversion results are stored in result registers that can be assigned by the user according to the requirements of the application. The following bitfields direct the results to a register:

- RESTGT in register **G0CHCTRy (y = 0 - 7)** etc.
  Selects the global result register
- RESREG in register **G0CHCTRy (y = 0 - 7)** etc.
  Selects the group-specific result register GxRES0 … GxRES15 when channel-specific result registers are used

19.8.2 Data Alignment

The position of a conversion result value within the selected result register depends on 3 configurations (summary in **Figure 19-15**):

- The selected result width (12/10/8 bits, selected by the conversion mode)
- The selected result position (Left/Right-aligned
- The selected data accumulation mode (data reduction)

These options provide the conversion results in a way that minimizes data handling for the application software.

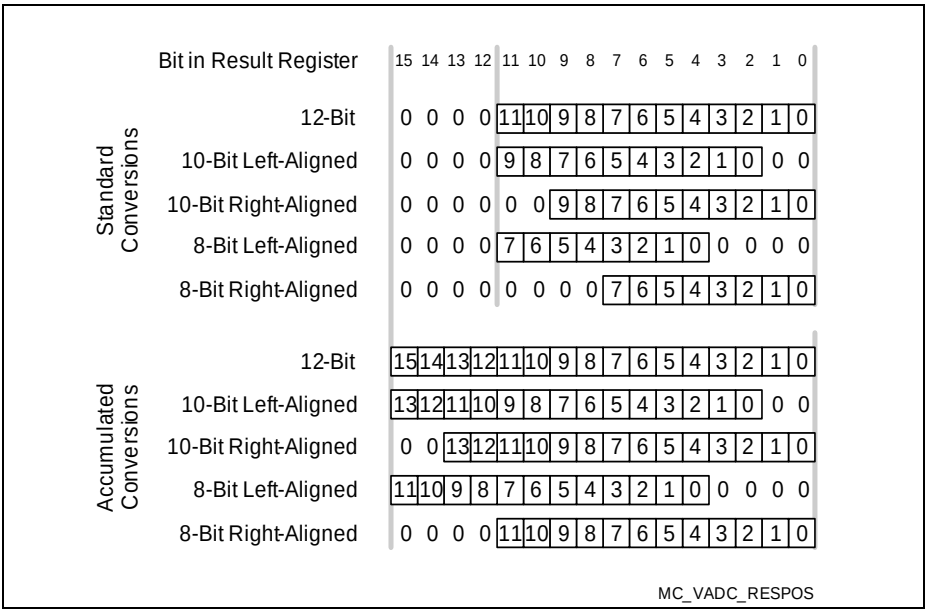


Figure 19-15 Result Storage Options

Bitfield RESULT can be written by software to provide the reference value for Fast Compare Mode. In this mode, bits 11-2 are evaluated, the other bits are ignored.

19.8.3 Wait-for-Read Mode

The wait-for-read mode prevents data loss due to overwriting a result register with a new conversion result before the CPU (or DMA) has read the previous data. For example, auto-scan conversion sequences or other sequences with “relaxed” timing requirements may use a common result register. However, the results come from different input channels, so an overwrite would destroy the result from the previous conversion[1)].

Wait-for-read mode automatically suspends the start of a conversion for this channel from this source until the current result has been read. So a conversion or a conversion sequence can be requested by a hardware or software trigger, while each conversion is only started after the result of the previous one has been read. This automatically aligns the conversion sequence with the CPU/DMA capability to read the formerly converted result (latency).

If wait-for-read mode is enabled for a result register (bit GxRCRy.WFR = 1), a request source does not generate a conversion request while the targeted result register contains valid data (indicated by the valid flag VF = 1) or if a currently running conversion targets the same result register.

If two request sources target the same result register with wait-for-read mode selected, a higher priority source cannot interrupt a lower priority conversion request started before the higher priority source has requested its conversion. Cancel-inject-repeat mode does not work in this case. In particular, this must be regarded if one of the involved sources is the background source (which usually has lowest priority). If the higher priority request targets a different result register, the lower priority conversion can be cancelled and repeated afterwards.

*Note: Wait-for-read mode is ignored for synchronized conversions of synchronization slaves (see **Section 19.9**).*
Due to its specific structure, wait-for-read mode does not work on the global result register GLOBRES.

1) Repeated conversions of a single channel that use a separate result register will not destroy other results, but rather update their own previous result value. This way, always the actual signal data is available in the result register.

19.8.4 Result FIFO Buffer

Result registers can either be used as direct target for conversion results or they can be concatenated with other result registers of the same ADC group to form a result FIFO buffer (first-in-first-out buffer mechanism). A result FIFO stores several measurement results that can be read out later with a “relaxed” CPU response timing. It is possible to set up more than one FIFO buffer structure with the available result registers.

Result FIFO structures of two or more registers are built by concatenating result registers to their following “neighbor” result register (with next higher index, see **Figure 19-16**). This is enabled by setting bitfield GxRCRy.FEN = 01_B.
Conversion results are stored to the register with the highest index of a FIFO structure. Software reads the values from the FIFO register with the lowest index.

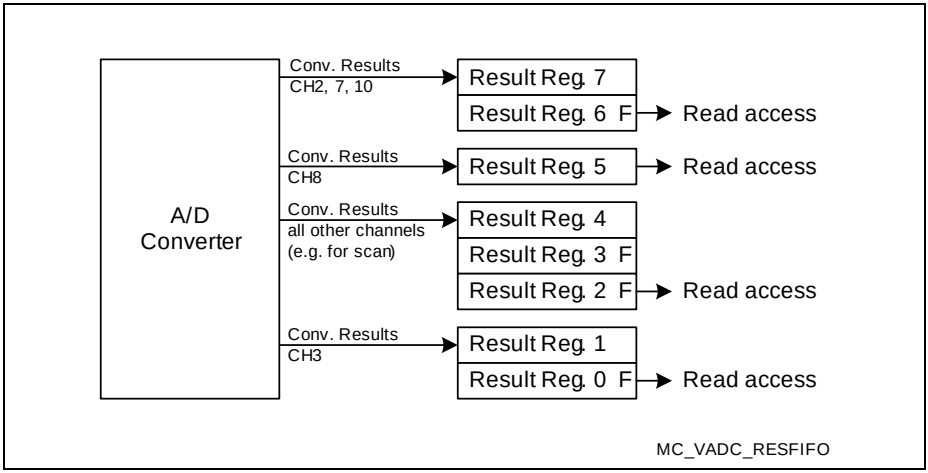


Figure 19-16 Result FIFO Buffers

In the example shown the result registers have been configured in the following way:

- 2-stage buffer consisting of result registers 7-6
- dedicated result register 5
- 3-stage buffer consisting of result registers 4-3-2
- 2-stage buffer consisting of result registers 1-0

Table 19-3 summarizes the required configuration of result registers if they are combined to build result FIFO buffers.

Table 19-3 Properties of Result FIFO Registers

Function	Input Stage	Intermed. Stage	Output Stage
Result target	YES	no	no
Application read	no	no	YES
Data reduction mode	YES	no	no
Wait-for-read mode	YES	no	no
Result event interrupt	no	no	YES
FIFO enable (FEN)	00_B	01_B	01_B
Registers in example	7, 4, 1	3	6, 2, 0

Note: If enabled, a result interrupt is generated for each data word in the FIFO.

19.8.5 Result Event Generation

A result event can be generated when a new value is stored to a result register. Result events can be restricted due to data accumulation and be generated only if the accumulation is complete.

Result events can also be suppressed completely.

19.8.6 Data Modification

The data resulting from conversions can be automatically modified before being used by an application. Several options can be selected (bitfield DMM in register **G0RCRy (y = 0 - 15)** etc.) which reduce the CPU/DMA load required to unload and/or process the conversion data.

- **Standard Data Reduction Mode (for GxRES0 … GxRES15):**
  Accumulates 2, 3, or 4 result values within each result register before generating a result interrupt. This can remove some noise from the input signal.
- **Result Filtering Mode (FIR, for GxRES7, GxRES15):**
  Applies a 3rd order Finite Impulse Response Filter (FIR) with selectable coefficients to the conversion results for the selected result register.
- **Result Filtering Mode (IIR, for GxRES7, GxRES15):**
  Applies a 1st order Infinite Impulse Response Filter (IIR) with selectable coefficients to the conversion results for the selected result register.
- **Difference Mode (for GxRES1 … GxRES15):**
  Subtracts the contents of result register GxRES0 from the conversion results for the selected result register. Bitfield DRCTR is not used in this mode.

Table 19-4 Function of Bitfield DRCTR

DRCTR	Standard Data Reduction Mode (DMM = 00_B)	DRCTR	Result Filtering Mode (DMM = 01_B)[1)]
0000_B	Data Reduction disabled	0000_B	FIR filter: a=2, b=1, c=0
0001_B	Accumulate 2 result values	0001_B	FIR filter: a=1, b=2, c=0
0010_B	Accumulate 3 result values	0010_B	FIR filter: a=2, b=0, c=1
0011_B	Accumulate 4 result values	0011_B	FIR filter: a=1, b=1, c=1
0100_B	Reserved	0100_B	FIR filter: a=1, b=0, c=2
0101_B	Reserved	0101_B	FIR filter: a=3, b=1, c=0
0110_B	Reserved	0110_B	FIR filter: a=2, b=2, c=0
0111_B	Reserved	0111_B	FIR filter: a=1, b=3, c=0
1000_B	Reserved	1000_B	FIR filter: a=3, b=0, c=1
1001_B	Reserved	1001_B	FIR filter: a=2, b=1, c=1
1010_B	Reserved	1010_B	FIR filter: a=1, b=2, c=1
1011_B	Reserved	1011_B	FIR filter: a=2, b=0, c=2
1100_B	Reserved	1100_B	FIR filter: a=1, b=1, c=2
1101_B	Reserved	1101_B	FIR filter: a=1, b=0, c=3

Table 19-4 Function of Bitfield DRCTR (cont'd)

DRCTR	Standard Data Reduction Mode (DMM = 00_B)	DRCTR	Result Filtering Mode (DMM = 01_B)[1]
1110_B	Reserved	1110_B	IIR filter: a=2, b=2
1111_B	Reserved	1111_B	IIR filter: a=3, b=4

1) The filter registers are cleared while bitfield DMM ≠ 01_B.

Standard Data Reduction Mode

The data reduction mode can be used as digital filter for anti-aliasing or decimation purposes. It accumulates a maximum of 4 conversion values to generate a final result.

Each result register can be individually enabled for data reduction, controlled by bitfield DRCTR in registers **G0CHCTRy (y = 0 - 7)**. The data reduction counter DRC indicates the actual status of the accumulation.

Note: Conversions for other result registers can be inserted between conversions to be accumulated.

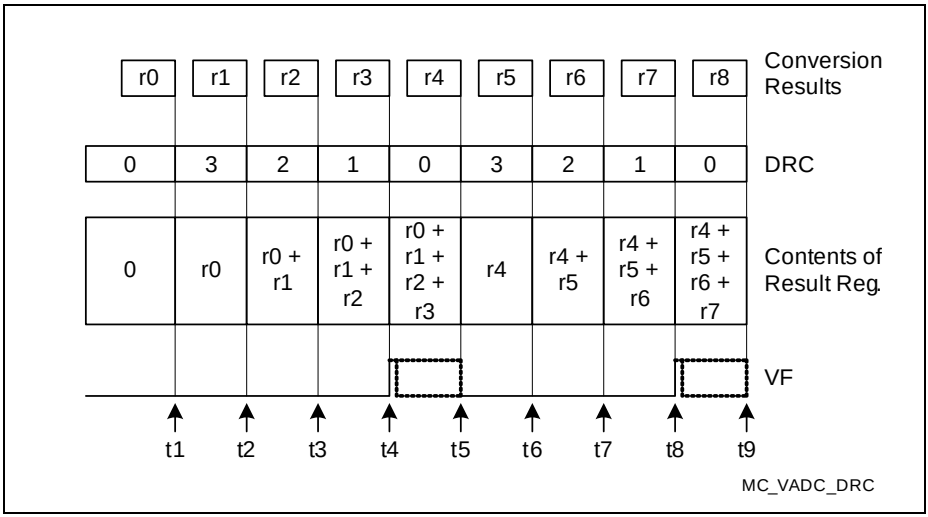


Figure 19-17 Standard Data Reduction Filter

This example shows a data reduction sequence of 4 accumulated conversion results. Eight conversion results (r0 … r7) are accumulated and produce 2 final results.

When a conversion is complete and stores data to a result register that has data reduction mode enabled, the data handling is controlled by the data reduction counter DRC:

- If DRC = 0 (t1, t5, t9 in the example), the conversion result is stored to the register. DRC is loaded with the contents of bitfield DRCTR (i.e. the accumulation begins).
- If DRC > 0 (t2, t3, t4 and t6, t7, t8 in the example), the conversion result is added to the value in the result register.
  DRC is decremented by 1.
- If DRC becomes 0, either decremented from 1 (t4 and t8 in the example) or loaded from DRCTR, the valid bit for the respective result register is set and a result register event occurs.

The final result must be read before the next data reduction sequence starts (before t5 or t9 in the example). This automatically clears the valid flag.

*Note: Software can clear the data reduction counter DRC by clearing the corresponding valid Flag (via **GxVFR (x = 0 - 3)**).*

The response time to read the final data reduction results can be increased by associating the adjacent result register to build a result FIFO (see **Figure 19-18**). In this case, the final result of a data reduction sequence is loaded to the adjacent register. The value can be read from this register until the next data reduction sequence is finished (t8 in the 2nd example).

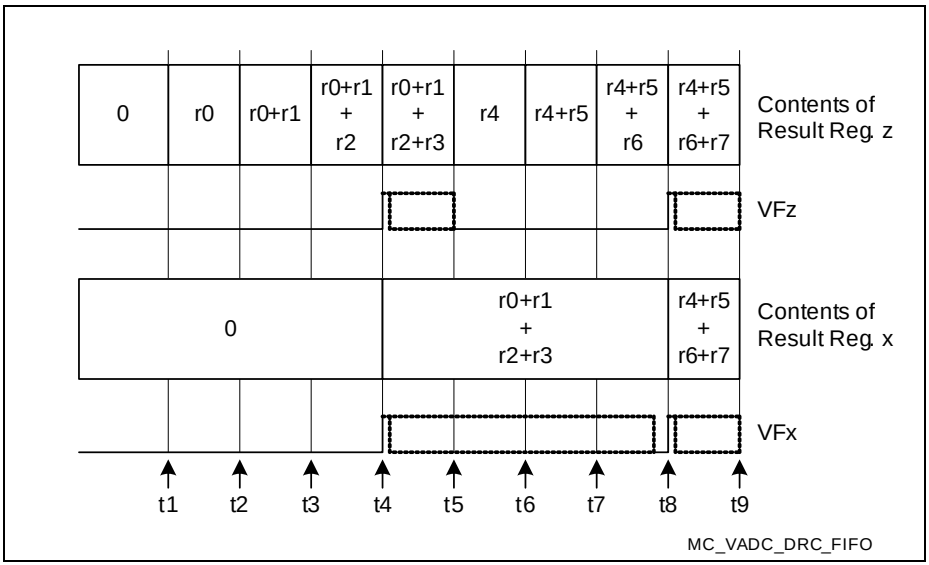


Figure 19-18 Standard Data Reduction Filter with FIFO Enabled

Finite Impulse Response Filter Mode (FIR)

The FIR filter (see **Figure 19-19**) provides 2 result buffers for intermediate results (RB1, RB2) and 3 configurable tap coefficients (a, b, c).

The conversion result and the intermediate result buffer values are added weighted with their respective coefficients to form the final value for the result register. Several predefined sets of coefficients can be selected via bitfield DRCTR (coding listed in **Table 19-4**) in registers **G0RESy (y = 0 - 15)** and **GLOBRES**. These coefficients lead to a gain of 3 or 4 to the ADC result producing a 14-bit value. The valid flag (VF) is activated for each sample after activation, i.e. for each sample generates a valid result.

Note: Conversions for other result registers can be inserted between conversions to be filtered.

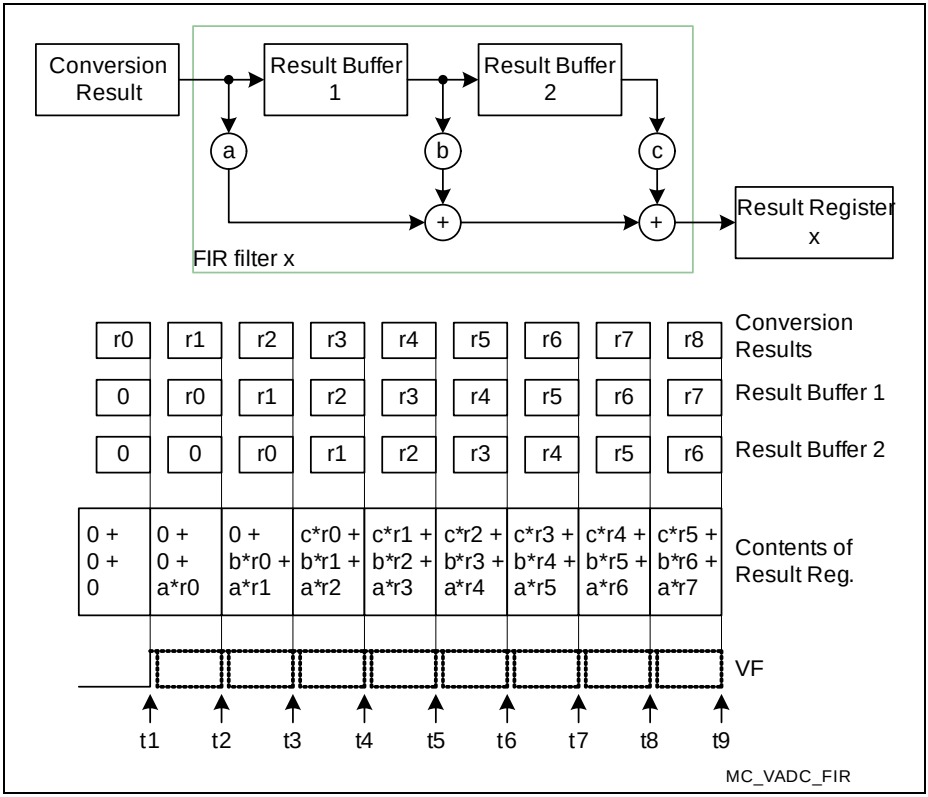


Figure 19-19 FIR Filter Structure

Note: The filter registers are cleared while bitfield DMM ≠ 01_B.

Infinite Impulse Response Filter Mode (IIR)

The IIR filter (see **Figure 19-20**) provides a result buffer (RB) and 2 configurable coefficients (a, b). It represents a first order low-pass filter.

The conversion result, weighted with the respective coefficient, and a fraction of the previous result are added to form the final value for the result register. Several predefined sets of coefficients can be selected via bitfield DRCTR (coding listed in **Table 19-4**) in registers **G0RESy (y = 0 - 15)** and **GLOBRES**. These coefficients lead to a gain of 4 to the ADC result producing a 14-bit value. The valid flag (VF) is activated for each sample after activation, i.e. for each sample generates a valid result.

Note: Conversions for other result registers can be inserted between conversions to be filtered.

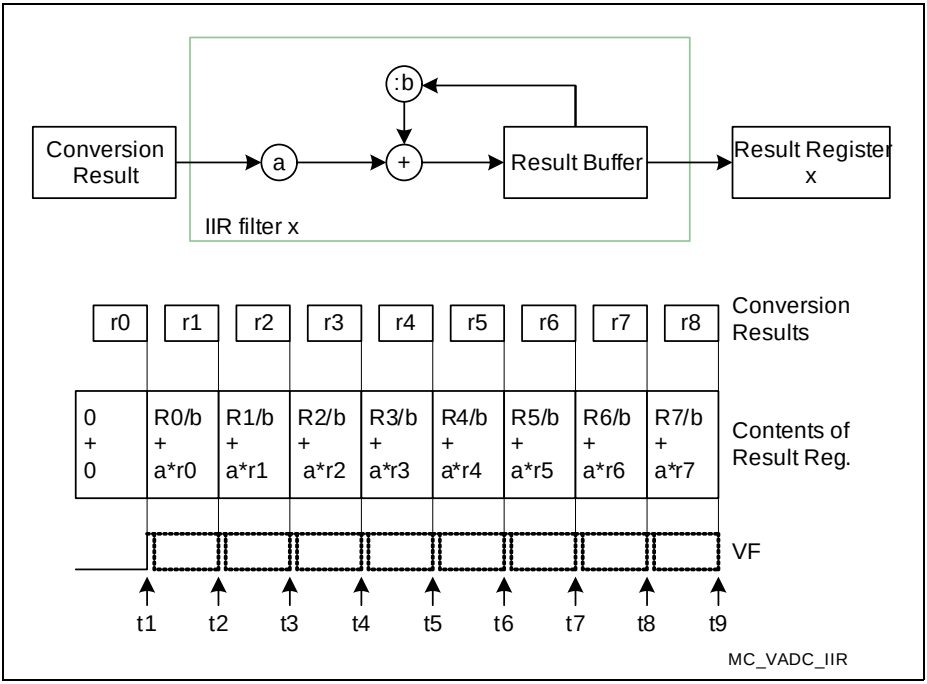


Figure 19-20 IIR Filter Structure

Note: The filter registers are cleared while bitfield DMM ≠ 01_B.

Difference Mode

Subtracting the contents of result register 0 from the actual result puts the results of the respective channel in relation to another signal. No software action is required.

The reference channel must store its result(s) into result register 0. The reference value can be determined once and then be used for a series of conversions, or it can be converted before each related conversion.

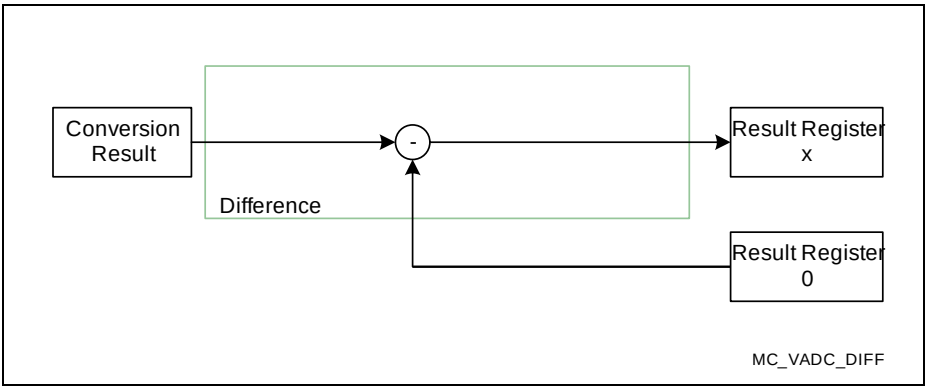


Figure 19-21 Result Difference

19.9 Synchronization of Conversions

The conversions of an ADC kernel can be scheduled either self-timed according to the kernel's configuration or triggered by external (outside the ADC) signals:
Synchronized conversions support parallel conversion of channels within a synchronization group[1]. This optimizes e.g. the control of electrical drives.
Equidistant sampling supports conversions in a fixed raster with minimum jitter. This optimizes e.g. filter algorithms or audio applications.

19.9.1 Synchronized Conversions for Parallel Sampling

Several independent ADC kernels[1] implemented in the XMC4500 can be synchronized for simultaneous measurements of analog input channels. While no parallel conversion is requested, the kernels can work independently.

The synchronization mechanism for parallel conversions ensures that the sample phases of the related channels start simultaneously. Synchronized kernels convert the same channel that is requested by the master. Different values for the resolution and the sample phase length of each kernel for a parallel conversion are supported.

A parallel conversion can be requested individually for each input channel (one or more). In the example shown in **Figure 19-22**, input channels CH3 of the ADC kernels 0 and 1 are converted synchronously, whereas other input channels do not lead to parallel conversions.

One kernel operates as synchronization master, the other kernel(s) operate(s) as synchronization slave. Each kernel can play either role. Master and slave kernels form a "conversion group" to control parallel sampling:

- **The synchronization master** ADC kernel can request a synchronized conversion of a certain channel (SYNC = 1 in the corresponding channel control register **G0CHCTRy (y = 0 - 7)** etc.), which is also requested in the connected slave ADC kernel.
  Wait-for-read mode is supported for the master.
- **The synchronization slave** ADC kernel reacts to incoming synchronized conversion requests from the master. While no synchronized conversions are requested, the slave kernel can execute "local" conversions.
  – The slave timing must be configured according to the master timing (ARBRND in register **GxARBPR (x = 0 - 3)**) to enable parallel conversions.
  – A parallel conversion request is always handled with highest priority and cancel-inject-repeat mode.
  – Wait-for-read mode is ignored in the slave. Previous results may be overwritten, in particular, if the same result register is used by other conversions.

1) For a summary, please refer to **"Synchronization Groups in the XMC4500" on Page 19-128**.

- The arbiter must run permanently (bit **GxARBPR (x = 0 - 3)**.ARBM = 0) for the synchronization slave.
  Initialize the slave before the master to have the arbiters run synchronously.
- Once started, a parallel conversion cannot be aborted.

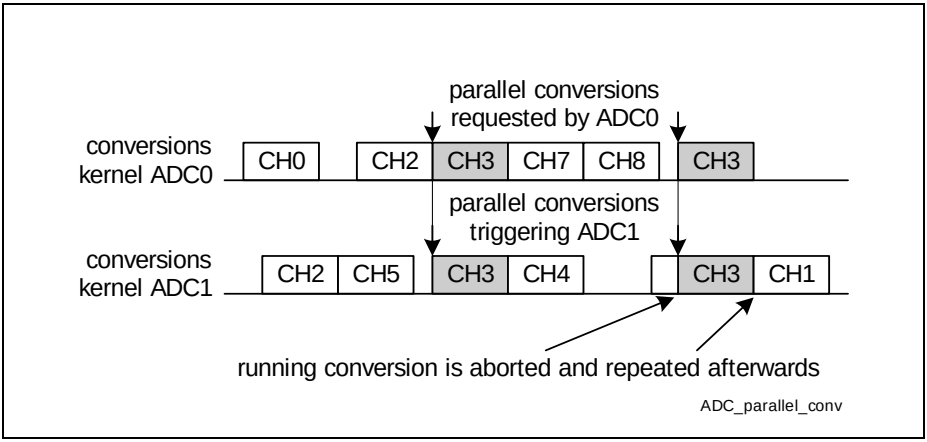


Figure 19-22 Parallel Conversions

The shown example uses synchronized conversions for channel CH3. The other channel conversions are controlled by their own kernels. ADC0 is the master, ADC1 is the slave.

The synchronization master controls the slave by providing the control information **GxARBCFG (x = 0 - 3)**.ANONS (see **Figure 19-23**) and the requested channel number. Bitfields **GxSYNCTR (x = 0 - 3)**.STSEL select the source of the ANON information for the master (00_B) and the slave(s) (01_B/10_B/11_B).

STSEL = 00_B always selects the own ANON information, and is, therefore, meant for the synchronization master or for stand-alone operation. The other control inputs (STSEL = 01_B/10_B/11_B) are connected to the other kernels of a synchronization group in ascending order (see also **Table 19-11 “Synchronization Groups in the XMC4500” on Page 19-128**).

The ready signals indicate when a slave kernel is ready to start the sample phase of a parallel conversion. Bit **GxSYNCTR (x = 0 - 3)**.EVALR1 = 1 enables the control by the ready signal.

*Note: Synchronized conversions request the same channel number, defined by the master. Using the alias feature (see **Section 19.7.3**), analog signals from different input channels can be converted. This is advantageous if e.g. CH0 is used as alternate reference.*

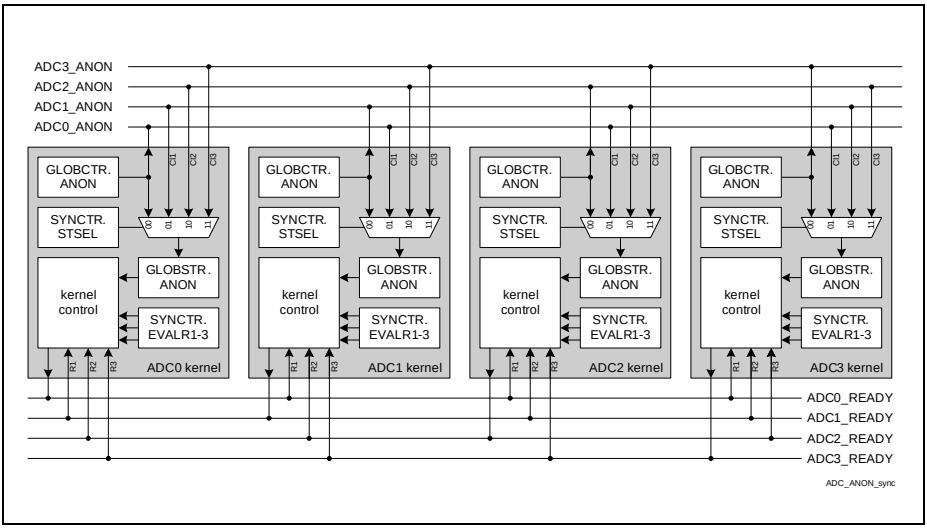


Figure 19-23 Synchronization via ANON and Ready Signals

19.9.2 Equidistant Sampling

To optimize the input data e.g. for filter or audio applications, conversions can be executed in a fixed timing raster. Conversions for equidistant sampling are triggered by an external signal (e.g. a timer). To generate the trigger signal synchronous to the arbiter, the ADC provides an output signal (ARBCNT) that is activated once per arbitration round and serves as timing base for the trigger timer. In this case, the arbiter must run permanently (**GxARBPR (x = 0 - 3)**.ARBM = 0). If the timer has an independent time base, the arbiter can be stopped while no requests are pending. The preface time (see **Figure 19-24**) must be longer than one arbitration round and the highest possible conversion time.

Select timer mode (TMEN = 1 in register **GxQCTRL0 (x = 0 - 3)** or **GxASCTRL (x = 0 - 3)**) for the intended source of equidistant conversions. In timer mode, a request of this source is triggered and arbitrated, but only started when the trigger signal is removed (see **Figure 19-24**) and the converter is idle.

To ensure that the converter is idle and the start of conversion can be controlled by the trigger signal, the equidistant conversion requests must receive highest priority. The preface time between request trigger and conversion start must be long enough for a currently active conversion to finish.

The frequency of signal REQTRx defines the sampling rate and its high time defines the preface time interval where the corresponding request source takes part in the arbitration.

Depending on the used request source, equidistant sampling is also supported for a sequence of channels. It is also possible to do equidistant sampling for more than one request source in parallel if the preface times and the equidistant conversions do not overlap.

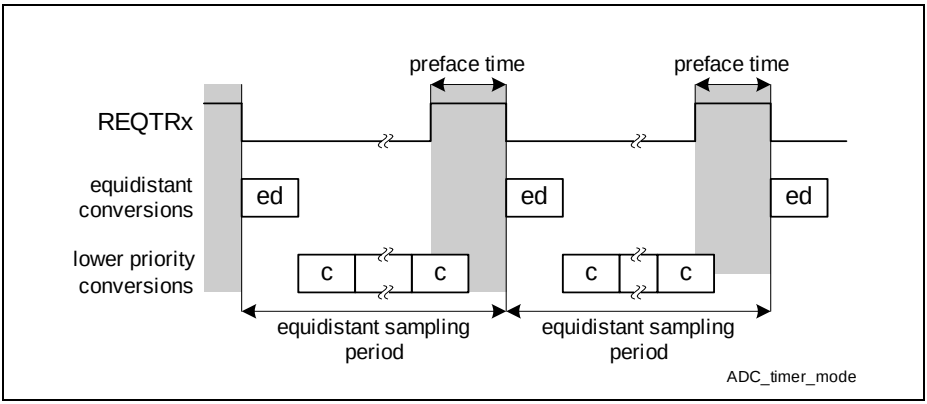


Figure 19-24 Timer Mode for Equidistant Sampling

19.10 Safety Features

Several test features can be enabled to verify the validity of the analog input signals of an application. These test features aim at different sections of the signal flow:

- **Broken Wire Detection** validates the connection from the sensor to the input pin,
- **Multiplexer Diagnostics** validates the operation of the internal analog input multiplexer,
- **Converter Diagnostics** validates the operation of the Analog/Digital converter itself.

19.10.1 Broken Wire Detection

To test the proper connection of an external analog sensor to its input pin, the converter's capacitor can be precharged to a selectable value before the regular sample phase. If the connection to the sensor is interrupted the subsequent conversion value will rather represent the precharged value than the expected sensor result. By using a precharge voltage outside the expected result range (broken wire detection preferably uses V_{AGND} and/or V_{AREF}) a valid measurement (sensor connected) can be distinguished from a failure (sensor detached).

While broken wire detection is disabled, the converter's capacitor is precharged to $V_{AREF}/2$.

Note: The duration of the complete conversion is increased by the preparation phase (same as the sample phase) if the broken wire detection is enabled. This influences the timing of conversion sequences.

Broken wire detection can be enabled for each channel separately by bitfield BWDEN in the corresponding channel control register (**G0CHCTRy (y = 0 - 7)**). This bitfield also selects the level for the preparation phase.

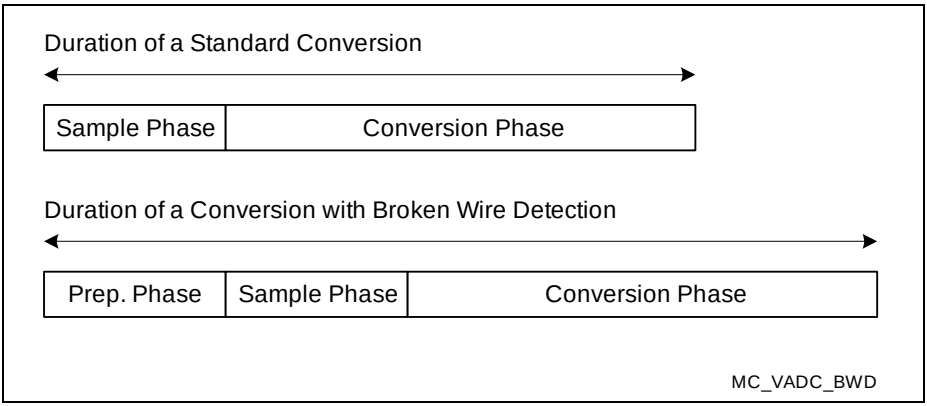


Figure 19-25 Broken Wire Detection

19.10.2 Signal Path Test Modes

Additional test structures can be activated to test the signal path from the sensor to the input pin and the internal signal path from the input pin through the multiplexer to the converter. These test structures apply additional loads to the signal path (see summary in **Figure 19-26**).

Multiplexer Diagnostics

To test the proper operation of the internal analog input multiplexer, additional test signals can be connected to channels CH1 and CH2. In combination with a known external input signal this test function shows if the multiplexer connects any pin to the converter input and if this is the correct pin.

Pull-Down Diagnostics

One single input channel provides a further strong pull-down (R_{PDD}) that can be activated to verify the external connection to a sensor.

Converter Diagnostics

To test the proper operation of the converter itself, several signals can be connected to the converter input. The test signals can be connected to the converter input either instead of the standard input signal or in parallel to the standard input signal.

The test signal can be selected from four different signals as shown in **Figure 19-26**.

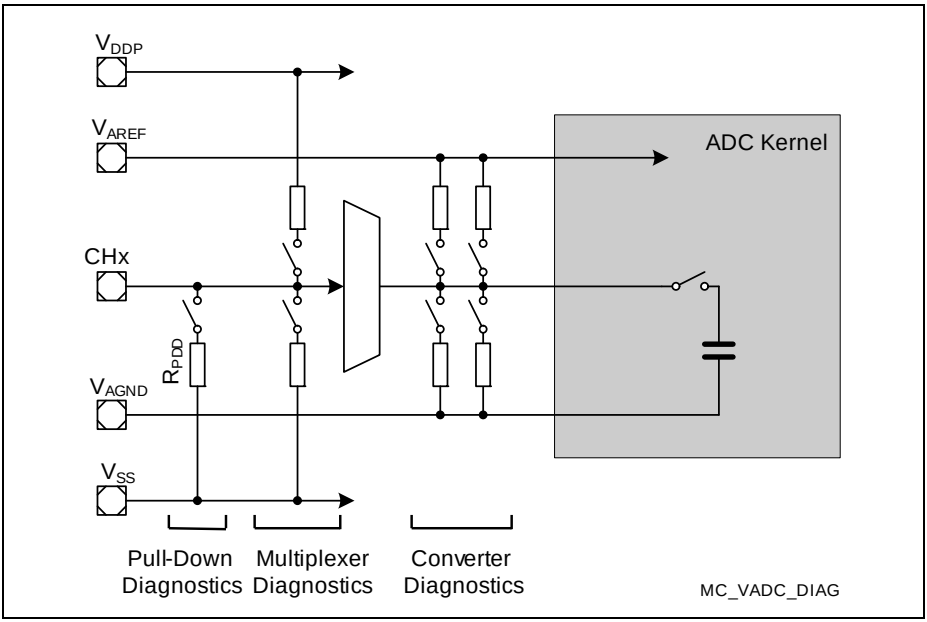


Figure 19-26 Signal Path Test

19.10.3 Configuration of Test Functions

The pull-up and pull-down devices for the test functions can be enabled individually under software control. Various test levels can be applied controlling the devices in an adequate way. Because these test functions interfere with the normal operation of the A/D Converters, they are controlled by a separate register set.

Not all test options are available for each channel. Selecting an unavailable function has no effect.

19.11 External Multiplexer Control

The number of analog input channels can be increased by connecting external analog multiplexers to an input channel. The ADC can be configured to control these external multiplexers automatically.

For each available EMUX interface (see register **EMUXSEL**) one channel can be selected for this operating mode. The ADC supports 1-out-of-8 multiplexers with several control options:

- **Sequence mode** automatically converts all configured external channels when the selected channel is encountered. In the example in **Figure 19-27** the following conversions are done: --4-32-31-30-2-1-0--4-32-31-30-2-1-0--…
- **Single-step mode** converts one external channel of the configured sequence when the selected channel is encountered. In the example in **Figure 19-27** the following conversions are done: --4-32-2-1-0--4-31-2-1-0--4-30-2-1-0--4-32-… (Single-step mode works best with one channel)
- **Steady mode** converts the configured external channel when the selected channel is encountered. In the example in **Figure 19-27** the following conversions are done: --4-32-2-1-0--4-32-2-1-0--4-32-2-1-0--…

*Note: The example in **Figure 19-27** has an external multiplexer connected to channel CH3. The start selection value EMUXSET is assumed as 2.*

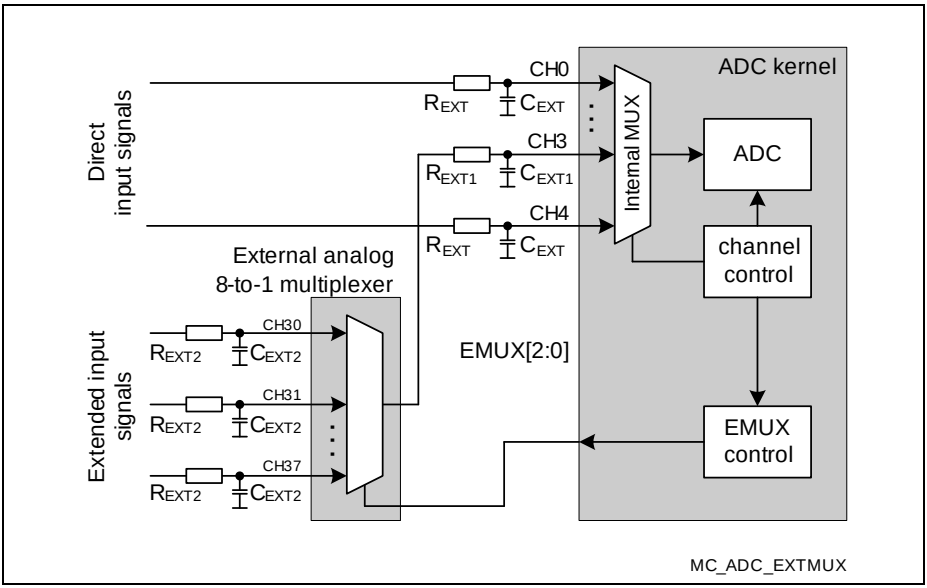


Figure 19-27 External Analog Multiplexer Example

Bitfield EMUXACT determines the control information sent to the external multiplexer. In single-step mode, EMUXACT is updated after each conversion of an enabled channel. If EMUXACT = 000_B it is reloaded from bitfield EMUXSET, otherwise it is decremented by 1.

Additional external channels may have different properties due to the modified signal path. Local filters may be used at the additional inputs (R_{EXT2}-C_{EXT2} on CH3x in **Figure 19-27**). For applications where the external multiplexer is located far from the ADC analog input, it is recommended to add an RC filter directly at the analog input of the ADC (R_{EXT1}-C_{EXT1} on CH3 in **Figure 19-27**).

Note: Each RC filter limits the bandwidth of the analog input signal.

Conversions for external channels, therefore, use the alternate conversion mode setting CME. This automatically selects a different conversion mode if required.

Switching the external multiplexer usually requires an additional settling time for the input signal. Therefore, the alternate sample time setting STCE is applied each time the external channel is changed. This automatically fulfills the different sampling time requirements in this case.

In each group an arbitrary channel can be assigned to external multiplexer control (register **GxEMUXCTR (x = 0 - 3)**). Each available port interface selects the group whose control lines are output (register **EMUXSEL**).

Control Signals

The external channel number that controls the external multiplexer can be output in standard binary format or Gray-coded. Gray code avoids intermediate multiplexer switching when selecting a sequence of channels, because only one bit changes at a time. **Table 19-5** indicates the resulting codes.

Table 19-5 EMUX Control Signal Coding

Channel	0	1	2	3	4	5	6	7
Binary	000_B	001_B	010_B	011_B	100_B	101_B	110_B	111_B
Gray	000	001	011	010	110	111	101	100

Operation Without External Multiplexer

If no external multiplexers are used in an application, the reset values of the control registers provide the appropriate setup.

EMUXMODE = 00_B disables the automatic EMUX control.

Since the control output signals are alternate port output signals, they are only visible at the respective pins if explicitly selected.

19.12 Service Request Generation

Each A/D Converter can activate up to 4 group-specific service request output signals and up to 4 shared service request output signals to issue an interrupt or to trigger a DMA channel. Two common service request groups are available, see **Table 19-10 "General Converter Configuration in the XMC4500" on Page 19-127**.

Several events can be assigned to each service request output. Service requests can be generated by three types of events:

- **Request source events:** indicate that a request source completed the requested conversion sequence and the application software can initiate further actions.
  For a scan source (group or background), the event is generated when the complete defined set of channels (pending bits) has been converted.
  For a group queue source, the event is generated according to the programming, i.e. when a channel with enabled source interrupt has been converted or when an invalid entry is encountered.
- **Channel events:** indicate that a conversion is finished. Optionally, channel events can be restricted to result values within a programmable value range. This offloads the CPU/DMA from background tasks, i.e. a service request is only activated if the specified conversion result range is met or exceeded.
- **Result events:** indicate a new valid result in a result register. Usually, this triggers a read action by the CPU (or DMA). Optionally, result events can be generated only at a reduced rate if data reduction is active.
  For example, a single DMA channel can read the results for a complete auto-scan sequence, if all channels of the sequence target the same result register and the transfers are triggered by result events.

Each ADC event is indicated by a dedicated flag that can be cleared by software. If a service request is enabled for a certain event, the service request is generated for each event, independent of the status of the corresponding event indication flag. This ensures efficient DMA handling of ADC events (the ADC event can generate a service request without the need to clear the indication flag).

Event flag registers indicate all types of events that occur during the ADC's operation. Software can set each flag by writing a 1 to the respective position in register GxCEFLAG/GxRFLAG to trigger an event. Software can clear each flag by writing a 1 to the respective position in register GxCEFCLR/GxREFCLR. If enabled, service requests are generated for each occurrance of an event, even if the associated flag remains set.

Node Pointer Registers

Requests from each event source can be directed to a set of service request nodes via associated node pointers. Requests from several sources can be directed to the same node; in this case, they are ORed to the service request output signal.

Software Service Request Activation

Each service request can be activated via software by setting the corresponding bit in register **GxSRACT (x = 0 - 3)**. This can be used for evaluation and testing purposes.

*Note: For shared service request lines see common groups in **Table 19-10**.*

19.13 Registers

The Versatile ADC is built from a series of converter blocks that are controlled in an identical way. This makes programming versatile and scalable. The corresponding registers, therefore, have an individual offset assigned (see **Table 19-7**). The exact register location is obtained by adding the respective register offset to the base address (see **Table 19-6**) of the corresponding group.

Due to the regular group structure, several registers appear within each group. Other registers are provided for each channel. This is indicated in the register overview table by placeholders:

- X###$_H$ means: x × 0400$_H$ + 0###$_H$, for x = 0 - 3
- ###Y$_H$ means: ###0$_H$ + y × 0004$_H$, for y = 0 - N (depends on register type)

Table 19-6 Registers Address Space

Module	Base Address	End Address	Note
VADC	4000 4000$_H$	4000 7FFF$_H$	

Table 19-7 Registers Overview

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Page Num.
			Read	Write	
ID	Module Identification Register	0008$_H$	U, PV	BE	19-60
CLC	Clock Control Register	0000$_H$	U, PV	PV	19-61
OCS	OCDS Control and Status Register	0028$_H$	U, PV	PV	19-62
GLOBCFG	Global Configuration Register	0080$_H$	U, PV	U, PV	19-64
GxARBCFG	Arbitration Configuration Register	X480$_H$	U, PV	U, PV	19-66
GxARBPR	Arbitration Priority Register	X484$_H$	U, PV	U, PV	19-68
GxCHASS	Channel Assignment Register, Group x	X488$_H$	U, PV	U, PV	19-65
GxQCTRL0	Queue 0 Source Control Register, Group x	X500$_H$	U, PV	U, PV	19-70
GxQMR0	Queue 0 Mode Register, Group x	X504$_H$	U, PV	U, PV	19-72
GxQSR0	Queue 0 Status Register, Group x	X508$_H$	U, PV	U, PV	19-74
GxQINR0	Queue 0 Input Register, Group x	X510$_H$	U, PV	U, PV	19-76
GxQ0R0	Queue 0 Register 0, Group x	X50C$_H$	U, PV	U, PV	19-78
GxQBUR0	Queue 0 Backup Register, Group x	X510$_H$	U, PV	U, PV	19-80

Table 19-7 Registers Overview (cont'd)

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Page Num.
			Read	Write	
GxASCTRL	Autoscan Source Control Register, Group x	$X520_H$	U, PV	U, PV	19-82
GxASMR	Autoscan Source Mode Register, Group x	$X524_H$	U, PV	U, PV	19-84
GxASSEL	Autoscan Source Channel Select Register, Group x	$X528_H$	U, PV	U, PV	19-86
GxASPND	Autoscan Source Pending Register, Group x	$X52C_H$	U, PV	U, PV	19-87
BRSCTRL	Background Request Source Control Register	0200_H	U, PV	U, PV	19-88
BRSMR	Background Request Source Mode Register	0204_H	U, PV	U, PV	19-90
BRSSELx	Background Request Source Channel Select Register, Group x	$018Y_H$	U, PV	U, PV	19-92
BRSPNDx	Background Request Source Channel Pending Register, Group x	$01CY_H$	U, PV	U, PV	19-93
GxCHCTRy	Channel x Control Register	$X60Y_H$	U, PV	U, PV	19-94
GxICLASS0	Input Class Register 0, Group x	$X4A0_H$	U, PV	U, PV	19-96
GxICLASS1	Input Class Register 1, Group x	$X4A4_H$	U, PV	U, PV	19-96
GLOBICLASS0	Input Class Register 0, Global	$00A0_H$	U, PV	U, PV	19-96
GLOBICLASS1	Input Class Register 1, Global	$00A4_H$	U, PV	U, PV	19-96
GxALIAS	Alias Register	$X4B0_H$	U, PV	U, PV	19-108
GxBOUND	Boundary Select Register, Group x	$X4B8_H$	U, PV	U, PV	19-109
GLOBBOUND	Global Boundary Select Register	$00B8_H$	U, PV	U, PV	19-109
GxBFL	Boundary Flag Register, Group x	$X4C8_H$	U, PV	U, PV	19-110
GxRCRy	Group x Result Control Register y	$X68Y_H$	U, PV	U, PV	19-99
GxRESy	Group x Result Register y	$X70Y_H$	U, PV	U, PV	19-101

Table 19-7 Registers Overview (cont'd)

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Page Num.
			Read	Write	
GxRESDy	Group x Result Register y (debug view)	$X78Y_H$	U, PV	U, PV	19-103
GLOBRCR	Global Result Control Register	0280_H	U, PV	U, PV	19-104
GLOBRES	Global Result Register	0300_H	U, PV	U, PV	19-105
GLOBRESD	Global Result Register (debug view)	0380_H	U, PV	U, PV	19-105
GxVFR	Valid Flag Register, Group x	$X5F8_H$	U, PV	U, PV	19-107
GxSYNCTR	Synchronization Control Register	$X4C0_H$	U, PV	U, PV	19-111
GLOBTF	Global Test Functions Register	0160_H	U, PV	U, PV	19-112
GxEMUXCTR	External Multiplexer Control Register, Group x	$X5F0_H$	U, PV	U, PV	19-113
EMUXSEL	External Multiplexer Select Register	$03F0_H$	U, PV	U, PV	19-115
GxSEFLAG	Source Event Flag Register, Group x	$X588_H$	U, PV	U, PV	19-115
GxCEFLAG	Channel Event Flag Register, Group x	$X580_H$	U, PV	U, PV	19-116
GxREFLAG	Result Event Flag Register, Group x	$X584_H$	U, PV	U, PV	19-117
GxSEFCLR	Source Event Flag Clear Register, Group x	$X598_H$	U, PV	U, PV	19-117
GxCEFCLR	Channel Event Flag Clear Register, Group x	$X590_H$	U, PV	U, PV	19-118
GxREFCLR	Result Event Flag Clear Register, Group x	$X594_H$	U, PV	U, PV	19-119
GLOBEFLAG	Global Event Flag Register	$00E0_H$	U, PV	U, PV	19-119

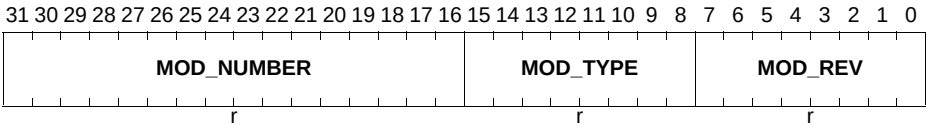
Table 19-7 Registers Overview (cont'd)

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Page Num.
			Read	Write	
GxSEVNP	Source Event Node Pointer Register, Group x	$X5C0_H$	U, PV	U, PV	19-120
GxCEVNP0	Channel Event Node Pointer Register 0, Group x	$X5A0_H$	U, PV	U, PV	19-121
GxREVNP0	Result Event Node Pointer Register 0, Group x	$X5B0_H$	U, PV	U, PV	19-122
GxREVNP1	Result Event Node Pointer Register 1, Group x	$X5B4_H$	U, PV	U, PV	19-123
GLOBEVNP	Global Event Node Pointer Register	0140_H	U, PV	U, PV	19-124
GxSRACT	Service Request Software Activation Trigger, Group x	$X5C8_H$	U, PV	U, PV	19-126

19.13.1 Module Identification

The module identification register indicates the version of the ADC module that is used in the XMC4500.

ID
Module Identification Register (0008_H) Reset Value: 00C5 C0XX$_H$



Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision** Indicates the revision number of the implementation. This information depends on the design step.
MOD_TYPE	[15:8]	r	**Module Type** This internal marker is fixed to $C0_H$.
MOD_NUMBER	[31:16]	r	**Module Number** Indicates the module identification number ($00C5_H$ = SARADC).

19.13.2 System Registers

A set of standardized registers provides general access to the module and controls basic system functions.

The Clock Control Register **CLC** allows the programmer to adapt the functionality and power consumption of the module to the requirements of the application. Register **CLC** controls the module clock signal and the reactivity to the sleep mode signal.

CLC
Clock Control Register **(0000_H)** **Reset Value: $0000\ 0003_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**EDIS**	**0**	**DISS**	**DISR**
r	r	r	r	r	r	r	r	r	r	r	r	rw	r	r	rw

Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module. Also the analog section is disabled by clearing ANONS. 0_B On request: enable the module clock 1_B Off request: stop the module clock
DISS	1	r	**Module Disable Status Bit** 0_B Module clock is enabled 1_B Off: module is not clocked
0	2	r	**Reserved, write 0, read as 0**
EDIS	3	rw	**Sleep Mode Enable Control** Used to control module's reaction to sleep mode. 0_B Sleep mode request is enabled and functional 1_B Module disregards the sleep mode control signal
0	[31:4]	r	**Reserved, write 0, read as 0**

The OCDS control and status register OCS controls the module's behavior in suspend mode (used for debugging) and includes the module-related control bits for the OCDS Trigger Bus (OTGB).

The OCDS Control and Status (OCS) register is cleared by Debug Reset.
The OCS register can only be written when the OCDS is enabled.
If OCDS is being disabled, the OCS register value will not change.
When OCDS is disabled the OCS suspend control is ineffective.
Write access is 32 bit wide only and requires Supervisor Mode.

OCS

OCDS Control and Status Register (0028_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**SUSSTA**	**SUS_P**	**SUS**				**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	rh	w	rw				r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**TG_P**	**TGB**	**TGS**	
r	r	r	r	r	r	r	r	r	r	r	r	w	rw	rw	

Field	Bits	Type	Description
TGS	[1:0]	rw	**Trigger Set for OTGB0/1** 00_B No Trigger Set output 01_B Trigger Set 1: TS16_SSIG, input sample signals 10_B Reserved 11_B Reserved
TGB	2	rw	**OTGB0/1 Bus Select** 0_B Trigger Set is output on OTGB0 1_B Trigger Set is output on OTGB1
TG_P	3	w	**TGS, TGB Write Protection** TGS and TGB are only written when TG_P is 1, otherwise unchanged. Read as 0.
0	[23:4]	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
SUS	[27:24]	rw	**OCDS Suspend Control** Controls the sensitivity to the suspend signal coming from the OCDS Trigger Switch (OTGS) 0000_B Will not suspend 0001_B Hard suspend: Clock is switched off immediately. 0010_B Soft suspend mode 0: Stop conversions after the currently running one is completed and its result has been stored. No change for the arbiter. 0011_B Soft suspend mode 1: Stop conversions after the currently running one is completed and its result has been stored. Stop arbiter after the current arbitration round. others: Reserved
SUS_P	28	w	**SUS Write Protection** SUS is only written when SUS_P is 1, otherwise unchanged. Read as 0.
SUSSTA	29	rh	**Suspend State** 0_B Module is not (yet) suspended 1_B Module is suspended
0	[31:30]	r	**Reserved, write 0, read as 0**

Table 19-8 TS16_SSIG Trigger Set VADC

Bits	Name	Description
[3:0]	GxSAMPLE	Input signal sample phase of converter group x (x = 3-0)
[15:4]	0	Reserved

19.13.3 General Registers

The global configuration register provides global control and configuration options that are valid for all converters of the cluster.

GLOBCFG
Global Configuration Register (0080_H) Reset Value: $0000\ 000F_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SUCAL	0	0	0	0	0	0	0	0	0	0	0	DPCAL3	DPCAL2	DPCAL1	DPCAL0
w	r	r	r	r	r	r	r	r	r	r	r	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIVWC	0	0	0	0	0	DIVD		DCMSB	0	0	DIVA				
w	r	r	r	r	r	rw		rw	r	r	rw				

Field	Bits	Type	Description
DIVA	[4:0]	rw	**Divider Factor for the Analog Internal Clock** Defines the frequency of the basic converter clock f_{ADCI} (base clock for conversion and sample phase). 00_H $f_{ADCI} = f_{ADC} / 2$ 01_H $f_{ADCI} = f_{ADC} / 2$ 02_H $f_{ADCI} = f_{ADC} / 3$ ... $1F_H$ $f_{ADCI} = f_{ADC} / 32$
0	[6:5]	r	**Reserved, write 0, read as 0**
DCMSB	7	rw	**Double Clock for the MSB Conversion** Selects an additional clock cycle for the conversion step of the MSB.[1)] 0_B 1 clock cycles for the MSB (standard) 1_B 2 clock cycles for the MSB (f_{ADCI} > 20 MHz)
DIVD	[9:8]	rw	**Divider Factor for the Arbiter Clock** Defines the frequency of the arbiter clock f_{ADCD}. 00_B $f_{ADCD} = f_{ADC}$ 01_B $f_{ADCD} = f_{ADC} / 2$ 10_B $f_{ADCD} = f_{ADC} / 3$ 11_B $f_{ADCD} = f_{ADC} / 4$
0	[14:10]	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
DIVWC	15	w	**Write Control for Divider Parameters** 0_B No write access to divider parameters 1_B Bitfields DIVA, DCMSB, DIVD can be written
DPCALx (x = 0 - 3)	x+16	rw	**Disable Post-Calibration** 0_B Automatic post-calibration after each conversion of group x 1_B No post-calibration *Note: This bit is only valid for the calibrated converters within the given product type.*
0	[30:20]	r	**Reserved, write 0, read as 0**
SUCAL	31	w	**Start-Up Calibration** The 0-1 transition of bit SUCAL initiates the start-up calibration phase of all calibrated analog converters. 0_B No action 1_B Initiate the start-up calibration phase (indication in bit GxARBCFG.CAL)

1) Please also refer to section **"Conversion Timing" on Page 19-26**.

GxCHASS (x = 0 - 3)
Channel Assignment Register, Group x

(x * 0400_H + 0488_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**ASSCH7**	**ASSCH6**	**ASSCH5**	**ASSCH4**	**ASSCH3**	**ASSCH2**	**ASSCH1**	**ASSCH0**
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
ASSCHy (y = 0 - 7)	y	rw	**Assignment for Channel y** 0_B Channel y can be a background channel converted with lowest priority 1_B Channel y is a priority channel within group x

Field	Bits	Type	Description
0	[31:8]	r	**Reserved, write 0, read as 0**

19.13.4 Arbitration and Source Registers

The Arbitration Configuration Register selects the timing and the behavior of the arbiter.

GxARBCFG (x = 0 - 3)
Arbitration Configuration Register, Group x

(x * 0400_H + 0480_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SAMPLE	BUSY	0	CAL	0	0	0	0	0	0	0	0	0	0	ANONS	
rh	rh	r	rh	r	r	r	r	r	r	r	r	r	r	rh	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	ARBM	0	ARBRND		0	0	ANONC	
r	r	r	r	r	r	r	r	rw	r	rw		r	r	rw	

Field	Bits	Type	Description
ANONC	[1:0]	rw	**Analog Converter Control** Defines the value of bitfield ANONS in a stand-alone converter or a converter in master mode. Coding see ANONS or **Section 19.4**.
0	[3:2]	r	**Reserved, write 0, read as 0**
ARBRND	[5:4]	rw	**Arbitration Round Length** Defines the number of arbitration slots per arb. round (arbitration round length = t_{ARB}).[1)] 00_B 4 arbitration slots per round (t_{ARB} = 4 / f_{ADCD}) 01_B 8 arbitration slots per round (t_{ARB} = 8 / f_{ADCD}) 10_B 16 arbitration slots per round (t_{ARB} = 16 / f_{ADCD}) 11_B 20 arbitration slots per round (t_{ARB} = 20 / f_{ADCD})
0	6	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
ARBM	7	rw	**Arbitration Mode** 0_B The arbiter runs permanently. This setting is required for a synchronization slave (see **Section 19.9.1**) and for equidistant sampling using the signal ARBCNT (see **Section 19.9.2**). 1_B The arbiter only runs if at least one conversion request of an enabled request source is pending. This setting ensures a reproducible latency from an incoming request to the conversion start, if the converter is idle. Synchronized conversions are not supported.
0	[15:8]	r	**Reserved, write 0, read as 0**
ANONS	[17:16]	rh	**Analog Converter Control Status** Defined by bitfield ANONC in a stand-alone kernel or a kernel in master mode. In slave mode, this bitfield is defined by bitfield ANONC of the respective master kernel. See also **Section 19.4**. 00_B Analog converter off 01_B Reserved 10_B Reserved 11_B Normal operation (permanently on)
0	[27:18]	r	**Reserved, write 0, read as 0**
CAL	28	rh	**Start-Up Calibration Active Indication** Indicates the start-up calibration phase of the corresponding analog converter. 0_B Completed or not yet started 1_B Start-up calibration phase is active *Note: Start conversions only after the start-up calibration phase is complete.*
0	29	r	**Reserved, write 0, read as 0**
BUSY	30	rh	**Converter Busy Flag** 0_B Not busy 1_B Converter is busy with a conversion

Field	Bits	Type	Description
SAMPLE	31	rh	**Sample Phase Flag** 0_B Converting or idle 1_B Input signal is currently sampled

1) The default setting of 4 arbitration slots is sufficient for correct arbitration. The duration of an arbitration round can be increased if required to synchronize requests.

The Arbitration Priority Register defines the request source priority and the conversion start mode for each request source.

Note: Only change priority and conversion start mode settings of a request source while this request source is disabled, and a currently running conversion requested by this source is finished.

GxARBPR (x = 0 - 3)
Arbitration Priority Register, Group x

(x * 0400_H + 0484_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**ASEN2**	**ASEN1**	**ASEN0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	rw	rw	rw	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**CSM2**	**0**	**PRIO2**		**CSM1**	**0**	**PRIO1**		**CSM0**	**0**	**PRIO0**	
r	r	r	r	rw	r	rw		rw	r	rw		rw	r	rw	

Field	Bits	Type	Description
PRIO0, **PRIO1,** **PRIO2**	[1:0], [5:4], [9:8]	rw	**Priority of Request Source x** Arbitration priority of request source x (in slot x) 00_B Lowest priority is selected. ... 11_B Highest priority is selected.
CSM0, **CSM1,** **CSM2**	3, 7, 11	rw	**Conversion Start Mode of Request Source x** 0_B Wait-for-start mode 1_B Cancel-inject-repeat mode, i.e. this source can cancel conversion of other sources.

Field	Bits	Type	Description
0	2, 6, 10, [23:12]	r	**Reserved, write 0, read as 0**
ASENy (y = 0 - 2)	24 + y	rw	**Arbitration Slot y Enable** Enables the associated arbitration slot of an arbiter round. The request source bits are not modified by write actions to ASENR. 0_B The corresponding arbitration slot is disabled and considered as empty. Pending conversion requests from the associated request source are disregarded. 1_B The corresponding arbitration slot is enabled. Pending conversion requests from the associated request source are arbitrated.
0	[31:27]	r	**Reserved, write 0, read as 0**

The control register of the queue source selects the external gate and/or trigger signals. Write control bits allow separate control of each function with a simple write access.

GxQCTRL0 (x = 0 - 3)
Queue 0 Source Control Register, Group x
($x * 0400_H + 0500_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TM WC	0	0	TM EN	0	0	0	0	GT WC	0	0	GT LVL	GT SEL			
w	r	r	rw	r	r	r	r	w	r	r	rh	rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XT WC	XT MODE		XT LVL	XT SEL				0	0	0	0	0	0	0	0
w	rw		rh	rw				r	r	r	r	r	r	r	r

Field	Bits	Type	Description
0	[7:0]	r	**Reserved, write 0, read as 0**
XTSEL	[11:8]	rw	**External Trigger Input Selection** The connected trigger input signals are listed in **Table 19-13 “Digital Connections in the XMC4500” on Page 19-131** *Note: XTSEL = 1111_B uses the selected gate input as trigger source (ENGT must be $0X_B$).*
XTLVL	12	rh	**External Trigger Level** Current level of the selected trigger input
XTMODE	[14:13]	rw	**Trigger Operating Mode** 00_B No external trigger 01_B Trigger event upon a falling edge 10_B Trigger event upon a rising edge 11_B Trigger event upon any edge
XTWC	15	w	**Write Control for Trigger Configuration** 0_B No write access to trigger configuration 1_B Bitfields XTMODE and XTSEL can be written
GTSEL	[19:16]	rw	**Gate Input Selection** The connected gate input signals are listed in **Table 19-13 “Digital Connections in the XMC4500” on Page 19-131**

Field	Bits	Type	Description
GTLVL	20	rh	**Gate Input Level** Current level of the selected gate input
0	[22:21]	r	**Reserved, write 0, read as 0**
GTWC	23	w	**Write Control for Gate Configuration** 0_B No write access to gate configuration 1_B Bitfield GTSEL can be written
0	[27:24]	r	**Reserved, write 0, read as 0**
TMEN	28	rw	**Timer Mode Enable** 0_B No timer mode: standard gating mechanism can be used 1_B Timer mode for equidistant sampling enabled: standard gating mechanism must be disabled
0	[30:29]	r	**Reserved, write 0, read as 0**
TMWC	31	w	**Write Control for Timer Mode** 0_B No write access to timer mode 1_B Bitfield TMEN can be written

The Queue Mode Register configures the operating mode of a queued request source.

GxQMR0 (x = 0 - 3)
Queue 0 Mode Register, Group x

($x * 0400_H + 0504_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**RPT DIS**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**CEV**	**FLU SH**	**TR EV**	**CLR V**	**0**	**0**	**0**	**0**	**0**	**EN TR**	**ENGT**	
r	r	r	r	w	w	w	w	r	r	r	r	r	rw	rw	

Field	Bits	Type	Description
ENGT	[1:0]	rw	**Enable Gate** Selects the gating functionality for source 0/2. 00_B No conversion requests are issued 01_B Conversion requests are issued if a valid conversion request is pending in the queue 0 register or in the backup register 10_B Conversion requests are issued if a valid conversion request is pending in the queue 0 register or in the backup register and REQGTx = 1 11_B Conversion requests are issued if a valid conversion request is pending in the queue 0 register or in the backup register and REQGTx = 0 *Note: REQGTx is the selected gating signal.*
ENTR	2	rw	**Enable External Trigger** 0_B External trigger disabled 1_B The selected edge at the selected trigger input signal REQTR generates the trigger event
0	[7:3]	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
CLRV	8	w	**Clear Valid Bit** 0_B No action 1_B The next pending valid queue entry in the sequence and the event flag EV are cleared. If there is a valid entry in the queue backup register (QBUR.V = 1), this entry is cleared, otherwise the entry in queue register 0 is cleared.
TREV	9	w	**Trigger Event** 0_B No action 1_B Generate a trigger event by software
FLUSH	10	w	**Flush Queue** 0_B No action 1_B Clear all queue entries (including backup stage) and the event flag EV. The queue contains no more valid entry.
CEV	11	w	**Clear Event Flag** 0_B No action 1_B Clear bit EV
0	[15:12]	r	**Reserved, write 0, read as 0**
RPTDIS	16	rw	**Repeat Disable** 0_B A cancelled conversion is repeated 1_B A cancelled conversion is discarded
0	[31:17]	r	**Reserved, write 0, read as 0**

The Queue Status Register indicates the current status of the queued source. The filling level and the empty information refer to the queue intermediate stages (if available) and to the queue register 0. An aborted conversion stored in the backup stage is not indicated by these bits (therefore, see QBURx.V).

GxQSR0 (x = 0 - 3)
Queue 0 Status Register, Group x

(x * 0400_H + 0508_H) **Reset Value: 0000 0020_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**EV**	**REQ GT**	**0**	**EMP TY**	**0**	**FILL**			
r	r	r	r	r	r	r	rh	rh	r	rh	r	rh			

Field	Bits	Type	Description
FILL	[3:0]	rh	**Filling Level for Queue 2** Indicates the number of valid queue entries. It is incremented each time a new entry is written to QINRx or by an enabled refill mechanism. It is decremented each time a requested conversion has been started. A new entry is ignored if the filling level has reached its maximum value. 0000_B There is 1 (if EMPTY = 0) or no (if EMPTY = 1) valid entry in the queue 0001_B There are 2 valid entries in the queue 0010_B There are 3 valid entries in the queue … 0111_B There are 8 valid entries in the queue others: Reserved
0	4	r	**Reserved, write 0, read as 0**
EMPTY	5	rh	**Queue Empty** 0_B There are valid entries in the queue (see FILL) 1_B No valid entries (queue is empty)
0	6	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
REQGT	7	rh	**Request Gate Level** Monitors the level at the selected REQGT input. 0_B The gate input is low 1_B The gate input is high
EV	8	rh	**Event Detected** Indicates that an event has been detected while at least one valid entry has been in the queue (queue register 0 or backup stage). Once set, this bit is cleared automatically when the requested conversion is started. 0_B No trigger event 1_B A trigger event has been detected
0	[31:9]	r	**Reserved, write 0, read as 0**

The Queue Input Register is the entry point for conversion requests of a queued request source.

GxQINR0 (x = 0 - 3)
Queue 0 Input Register, Group x

($x * 0400_H + 0510_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**EXTR**	**ENSI**	**RF**	**REQCHNR**				
r	r	r	r	r	r	r	r	w	w	w			w		

Field	Bits	Type	Description
REQCHNR	[4:0]	w	**Request Channel Number** Defines the channel number to be converted
RF	5	w	**Refill** 0_B No refill: this queue entry is converted once and then invalidated 1_B Automatic refill: this queue entry is automatically reloaded into QINRx when the related conversion is started
ENSI	6	w	**Enable Source Interrupt** 0_B No request source interrupt 1_B A request source event interrupt is generated upon a request source event (related conversion is finished)
EXTR	7	w	**External Trigger** Enables the external trigger functionality. 0_B A valid queue entry immediately leads to a conversion request. 1_B A valid queue entry waits for a trigger event to occur before issuing a conversion request.
0	[31:8]	r	**Reserved, write 0, read as 0**

Note: Registers QINRx share addresses with registers QBURx.
Write operations target the control bits in register QINRx. Read operations return the status bits from register QBURx.

The queue registers 0 monitor the status of the pending request (queue stage 0).

GxQ0R0 (x = 0 - 3)
Queue 0 Register 0, Group x (x * 0400_H + $050C_H$) Reset Value: 0000 0000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	V	EXTR	ENSI	RF	REQCHNR				
r	r	r	r	r	r	r	rh	rh	rh	rh	rh				

Field	Bits	Type	Description
REQCHNR	[4:0]	rh	**Request Channel Number** Stores the channel number to be converted.
RF	5	rh	**Refill** Selects the handling of handled requests. 0_B The request is discarded after the conversion start. 1_B The request is automatically refilled into the queue after the conversion start.
ENSI	6	rh	**Enable Source Interrupt** 0_B No request source interrupt 1_B A request source event interrupt is generated upon a request source event (related conversion is finished)
EXTR	7	rh	**External Trigger** Enables external trigger events. 0_B A valid queue entry immediately leads to a conversion request 1_B The request handler waits for a trigger event
V	8	rh	**Request Channel Number Valid** Indicates a valid queue entry in queue register 0. 0_B No valid queue entry 1_B The queue entry is valid and leads to a conversion request

Field	Bits	Type	Description
0	[31:9]	r	**Reserved, write 0, read as 0**

The Queue Backup Registers monitor the status of an aborted queued request.

GxQBUR0 (x = 0 - 3)
Queue 0 Backup Register, Group x

($x * 0400_H + 0510_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**V**	**EXTR**	**ENSI**	**RF**	**REQCHNR**				
r	r	r	r	r	r	r	rh	rh	rh	rh	rh				

Field	Bits	Type	Description
REQCHNR	[4:0]	rh	**Request Channel Number** The channel number of the aborted conversion that has been requested by this request source
RF	5	rh	**Refill** The refill control bit of the aborted conversion
ENSI	6	rh	**Enable Source Interrupt** The enable source interrupt control bit of the aborted conversion
EXTR	7	rh	**External Trigger** The external trigger control bit of the aborted conversion
V	8	rh	**Request Channel Number Valid** Indicates if the entry (REQCHNR, RF, TR, ENSI) in the queue backup register is valid. Bit V is set when a running conversion (that has been requested by this request source) is aborted, it is cleared when the aborted conversion is restarted. 0_B Backup register not valid 1_B Backup register contains a valid entry. This will be requested before a valid entry in queue register 0 (stage 0) will be requested.
r	[31:9]	r	**Reserved, write 0, read as 0**

Note: Registers QBURx share addresses with registers QINRx.
Read operations return the status bits from register QBURx. Write operations target the control bits in register QINRx.

Registers of Group Scan Source

There is a separate register set for each group scan source. These sources can be operated independently.

The control register of the autoscan source selects the external gate and/or trigger signals.
Write control bits allow separate control of each function with a simple write access.

GxASCTRL (x = 0 - 3)
Autoscan Source Control Register, Group x
($x * 0400_H + 0520_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TM WC	0	0	TM EN	0	0	0	0	GT WC	0	0	GT LVL	GT SEL			
w	r	r	rw	r	r	r	r	w	r	r	rh	rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XT WC	XT MODE		XT LVL	XT SEL				0	0	0	0	0	0	0	0
w	rw		rh	rw				r	r	r	r	r	r	r	r

Field	Bits	Type	Description
0	[7:0]	r	**Reserved, write 0, read as 0**
XTSEL	[11:8]	rw	**External Trigger Input Selection** The connected trigger input signals are listed in **Table 19-13 "Digital Connections in the XMC4500" on Page 19-131** *Note: XTSEL = 1111_B uses the selected gate input as trigger source (ENGT must be $0X_B$).*
XTLVL	12	rh	**External Trigger Level** Current level of the selected trigger input
XTMODE	[14:13]	rw	**Trigger Operating Mode** 00_B No external trigger 01_B Trigger event upon a falling edge 10_B Trigger event upon a rising edge 11_B Trigger event upon any edge
XTWC	15	w	**Write Control for Trigger Configuration** 0_B No write access to trigger configuration 1_B Bitfields XTMODE and XTSEL can be written

Field	Bits	Type	Description
GTSEL	[19:16]	rw	**Gate Input Selection** The connected gate input signals are listed in **Table 19-13 “Digital Connections in the XMC4500” on Page 19-131**
GTLVL	20	rh	**Gate Input Level** Current level of the selected gate input
0	[22:21]	r	**Reserved, write 0, read as 0**
GTWC	23	w	**Write Control for Gate Configuration** 0_B No write access to gate configuration 1_B Bitfield GTSEL can be written
0	[27:24]	r	**Reserved, write 0, read as 0**
TMEN	28	rw	**Timer Mode Enable** 0_B No timer mode: standard gating mechanism can be used 1_B Timer mode for equidistant sampling enabled: standard gating mechanism must be disabled
0	[30:29]	r	**Reserved, write 0, read as 0**
TMWC	31	w	**Write Control for Timer Mode** 0_B No write access to timer mode 1_B Bitfield TMEN can be written

The Conversion Request Mode Register configures the operating mode of the channel scan request source.

GxASMR (x = 0 - 3)
Autoscan Source Mode Register, Group x

($x * 0400_H + 0524_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**RPT DIS**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**LD EV**	**CLR PND**	**REQ GT**	**0**	**LDM**	**SCA N**	**EN SI**	**EN TR**	**ENGT**	
r	r	r	r	r	r	w	w	rh	r	rw	rw	rw	rw	rw	

Field	Bits	Type	Description
ENGT	[1:0]	rw	**Enable Gate** Selects the gating functionality for source 1. 00_B No conversion requests are issued 01_B Conversion requests are issued if at least one pending bit is set 10_B Conversion requests are issued if at least one pending bit is set and REQGTx = 1. 11_B Conversion requests are issued if at least one pending bit is set and REQGTx = 0. *Note: REQGTx is the selected gating signal.*
ENTR	2	rw	**Enable External Trigger** 0_B External trigger disabled 1_B The selected edge at the selected trigger input signal REQTR generates the load event
ENSI	3	rw	**Enable Source Interrupt** 0_B No request source interrupt 1_B A request source interrupt is generated upon a request source event (last pending conversion is finished)

Field	Bits	Type	Description
SCAN	4	rw	**Autoscan Enable** 0_B No autoscan 1_B Autoscan functionality enabled: a request source event automatically generates a load event
LDM	5	rw	**Autoscan Source Load Event Mode** 0_B Overwrite mode: Copy all bits from the select registers to the pending registers upon a load event 1_B Combine mode: Set all pending bits that are set in the select registers upon a load event (logic OR)
0	6	r	**Reserved, write 0, read as 0**
REQGT	7	rh	**Request Gate Level** Monitors the level at the selected REQGT input. 0_B The gate input is low 1_B The gate input is high
CLRPND	8	w	**Clear Pending Bits** 0_B No action 1_B The bits in register GxASPNDx are cleared
LDEV	9	w	**Generate Load Event** 0_B No action 1_B A load event is generated
0	[15:10]	r	**Reserved, write 0, read as 0**
RPTDIS	16	rw	**Repeat Disable** 0_B A cancelled conversion is repeated 1_B A cancelled conversion is discarded
0	[31:17]	r	**Reserved, write 0, read as 0**

The Channel Select Register selects the channels to be converted by the group scan request source. Its bits are used to update the pending register, when a load event occurs.

The number of valid channel bits depends on the channels available in the respective product type (please refer to **“Product-Specific Configuration” on Page 19-127**).

GxASSEL (x = 0 - 3)
Autoscan Source Channel Select Register, Group x
($x * 0400_H + 0528_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**CHSEL7**	**CHSEL6**	**CHSEL5**	**CHSEL4**	**CHSEL3**	**CHSEL2**	**CHSEL1**	**CHSEL0**
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
CHSELy (y = 0 - 7)	y	rw	**Channel Selection** Each bit (when set) enables the corresponding input channel of the respective group to take part in the scan sequence. 0_B Ignore this channel 1_B This channel is part of the scan sequence
0	[31:8]	r	**Reserved, write 0, read as 0**

The Channel Pending Register indicates the channels to be converted in the current conversion sequence. They are updated from the select register, when a load event occurs.

GxASPND (x = 0 - 3)
Autoscan Source Pending Register, Group x
(x * 0400_H + $052C_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**CHPND7**	**CHPND6**	**CHPND5**	**CHPND4**	**CHPND3**	**CHPND2**	**CHPND1**	**CHPND0**
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
CHPNDy (y = 0 - 7)	y	rw	**Channels Pending** Each bit (when set) request the conversion of the corresponding input channel of the respective group. 0_B Ignore this channel 1_B Request conversion of this channel
0	[31:8]	r	**Reserved, write 0, read as 0**

Registers of Background Scan Source

There is a single register set for the background scan source. This source is ccommon for the complete VADC.

The control register of the background request source selects the external gate and/or trigger signals.
Write control bits allow separate control of each function with a simple write access.

BRSCTRL
Background Request Source Control Register
(0200_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	0	0	0	0	0	0	0	GT WC	0	0	GT LVL	GT SEL			
r	r	r	r	r	r	r	r	w	r	r	rh	rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XT WC	XT MODE		XT LVL	XT SEL				0	0	0	0	0	0	0	0
w	rw		rh	rw				r	r	r	r	r	r	r	r

Field	Bits	Type	Description
0	[7:0]	r	**Reserved, write 0, read as 0**
XTSEL	[11:8]	rw	**External Trigger Input Selection** The connected trigger input signals are listed in **Table 19-13 “Digital Connections in the XMC4500” on Page 19-131** *Note: XTSEL = 1111_B uses the selected gate input as trigger source (ENGT must be $0X_B$).*
XTLVL	12	rh	**External Trigger Level** Current level of the selected trigger input
XTMODE	[14:13]	rw	**Trigger Operating Mode** 00_B No external trigger 01_B Trigger event upon a falling edge 10_B Trigger event upon a rising edge 11_B Trigger event upon any edge
XTWC	15	w	**Write Control for Trigger Configuration** 0_B No write access to trigger configuration 1_B Bitfields XTMODE and XTSEL can be written

Field	Bits	Type	Description
GTSEL	[19:16]	rw	**Gate Input Selection** The connected gate input signals are listed in **Table 19-13 “Digital Connections in the XMC4500” on Page 19-131**
GTLVL	20	rh	**Gate Input Level** Current level of the selected gate input
0	[22:21]	r	**Reserved, write 0, read as 0**
GTWC	23	w	**Write Control for Gate Configuration** 0_B No write access to gate configuration 1_B Bitfield GTSEL can be written
0	[31:24]	r	**Reserved, write 0, read as 0**

The Conversion Request Mode Register configures the operating mode of the background request source.

BRSMR
Background Request Source Mode Register
(0204_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RPT DIS
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	LD EV	CLR PND	REQ GT	0	LDM	SCA N	EN SI	EN TR	ENGT	
r	r	r	r	r	r	w	w	rh	r	rw	rw	rw	rw	rw	

Field	Bits	Type	Description
ENGT	[1:0]	rw	**Enable Gate** Selects the gating functionality for source 1. 00_B No conversion requests are issued 01_B Conversion requests are issued if at least one pending bit is set 10_B Conversion requests are issued if at least one pending bit is set and REQGTx = 1. 11_B Conversion requests are issued if at least one pending bit is set and REQGTx = 0. *Note: REQGTx is the selected gating signal.*
ENTR	2	rw	**Enable External Trigger** 0_B External trigger disabled 1_B The selected edge at the selected trigger input signal REQTR generates the load event
ENSI	3	rw	**Enable Source Interrupt** 0_B No request source interrupt 1_B A request source interrupt is generated upon a request source event (last pending conversion is finished)

Field	Bits	Type	Description
SCAN	4	rw	**Autoscan Enable** 0_B No autoscan 1_B Autoscan functionality enabled: a request source event automatically generates a load event
LDM	5	rw	**Autoscan Source Load Event Mode** 0_B Overwrite mode: Copy all bits from the select registers to the pending registers upon a load event 1_B Combine mode: Set all pending bits that are set in the select registers upon a load event (logic OR)
0	6	r	**Reserved, write 0, read as 0**
REQGT	7	rh	**Request Gate Level** Monitors the level at the selected REQGT input. 0_B The gate input is low 1_B The gate input is high
CLRPND	8	w	**Clear Pending Bits** 0_B No action 1_B The bits in registers BRSPNDx are cleared
LDEV	9	w	**Generate Load Event** 0_B No action 1_B A load event is generated
0	[15:10]	r	**Reserved, write 0, read as 0**
RPTDIS	16	rw	**Repeat Disable** 0_B A cancelled conversion is repeated 1_B A cancelled conversion is discarded
0	[31:17]	r	**Reserved, write 0, read as 0**

The Channel Select Registers select the channels to be converted by the background request source (channel scan source). Its bits are used to update the pending registers, when a load event occurs.

The number of valid channel bits depends on the channels available in the respective product type (please refer to **"Product-Specific Configuration" on Page 19-127**).

*Note: Priority channels selected in registers **GxCHASS (x = 0 - 3)** will not be converted.*

BRSSELx (x = 0 - 3)
Background Request Source Channel Select Register, Group x
(0180_H + x * 0004_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**CHSELG7**	**CHSELG6**	**CHSELG5**	**CHSELG4**	**CHSELG3**	**CHSELG2**	**CHSELG1**	**CHSELG0**
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
CHSELGy (y = 0 - 7)	y	rw	**Channel Selection Group x** Each bit (when set) enables the corresponding input channel of the respective group to take part in the background scan sequence. 0_B Ignore this channel 1_B This channel is part of the scan sequence
0	[31:8]	r	**Reserved, write 0, read as 0**

The Channel Pending Registers indicate the channels to be converted in the current conversion sequence. They are updated from the select registers, when a load event occurs.

BRSPNDx (x = 0 - 3)
Background Request Source Pending Register, Group x

($01C0_H$ + x * 0004_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**CHPNDG7**	**CHPNDG6**	**CHPNDG5**	**CHPNDG4**	**CHPNDG3**	**CHPNDG2**	**CHPNDG1**	**CHPNDG0**
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
CHPNDGy (y = 0 - 7)	y	rw	**Channels Pending Group x** Each bit (when set) request the conversion of the corresponding input channel of the respective group. 0_B Ignore this channel 1_B Request conversion of this channel
0	[31:8]	r	**Reserved, write 0, read as 0**

Note: Writing to any of registers BRSPNDx generates a load event that copies all bits from registers BRSSELx to BRSPNDx.
Use this shortcut only when writing the last word of the request pattern.

19.13.5 Channel Control Registers

G0CHCTRy (y = 0 - 7)
Group 0, Channel y Ctrl. Reg. (0600_H + y * 0004_H) Reset Value: 0000 0000_H
G1CHCTRy (y = 0 - 7)
Group 1, Channel y Ctrl. Reg. ($0A00_H$ + y * 0004_H) Reset Value: 0000 0000_H
G2CHCTRy (y = 0 - 7)
Group 2, Channel y Ctrl. Reg. ($0E00_H$ + y * 0004_H) Reset Value: 0000 0000_H
G3CHCTRy (y = 0 - 7)
Group 3, Channel y Ctrl. Reg. (1200_H + y * 0004_H) Reset Value: 0000 0000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	BWDEN	BWDCH		0	0	0	0	0	0	RESPOS	RESTBS	RESREG			
r	rw	rw		r	r	r	r	r	r	rw	rw	rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	REFSEL	SYNC	CHEVMODE		BNDSELU		BNDSELL		0	0	ICLSEL	
r	r	r	r	rw	rw	rw		rw		rw		r	r	rw	

Field	Bits	Type	Description
ICLSEL	[1:0]	rw	**Input Class Select** 00_B Use group-specific class 0 01_B Use group-specific class 1 10_B Use global class 0 11_B Use global class 1
0	[3:2]	r	**Reserved, write 0, read as 0**
BNDSELL	[5:4]	rw	**Lower Boundary Select** 00_B Use group-specific boundary 0 01_B Use group-specific boundary 1 10_B Use global boundary 0 11_B Use global boundary 1
BNDSELU	[7:6]	rw	**Upper Boundary Select** 00_B Use group-specific boundary 0 01_B Use group-specific boundary 1 10_B Use global boundary 0 11_B Use global boundary 1

Field	Bits	Type	Description
CHEVMODE	[9:8]	rw	**Channel Event Mode** Generate a channel event in normal compare mode with limit checking[1)] 00_B Never 01_B If result is inside the boundary band 10_B If result is outside the boundary band 11_B Always (ignore band) Generate a channel event in Fast Compare Mode[2)] 00_B Never 01_B If result switches to high (above comp. value) 10_B If result switches to low (below comp. value) 11_B If result switches to either level
SYNC	10	rw	**Synchronization Request** 0_B No synchroniz. request, standalone operation 1_B Request a synchronized conversion of this channel (only taken into account for a master)
REFSEL	11	rw	**Reference Input Selection** Defines the reference voltage input to be used for conversions on this channel. 0_B Standard reference input V_{AREF} 1_B Alternate reference input from CH0[3)]
0	[15:12]	rw	**Reserved, write 0, read as 0**
RESREG	[19:16]	rw	**Result Register** 0000_BStore result in group result register GxRES0 … 1111_BStore result in group result register GxRES15
RESTBS	20	rw	**Result Target for Background Source** 0_B Store results in the selected group result register 1_B Store results in the global result register
RESPOS	21	rw	**Result Position** 0_B Store results left-aligned 1_B Store results right-aligned
0	[27:22]	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
BWDCH	[29:28]	rw	**Broken Wire Detection Channel** 00_B Select V_{AGND} 01_B Select V_{AREF} 10_B Reserved 11_B Reserved
BWDEN	30	rw	**Broken Wire Detection Enable** 0_B Normal operation 1_B Additional preparation phase is enabled
0	31	r	**Reserved, write 0, read as 0**

1) The boundary band is defined as the area where the result is less than or equal to the selected upper boundary and greater than or equal to the selected lower boundary, see **Section 19.7.5**.

2) The result is bit FCR in the selected result register.

3) Some channels cannot select an alternate reference.

GxICLASS0 (x = 0 - 3)
Input Class Register 0, Group x

($x * 0400_H + 04A0_H$) **Reset Value: 0000 0000$_H$**

GxICLASS1 (x = 0 - 3)
Input Class Register 1, Group x

($x * 0400_H + 04A4_H$) **Reset Value: 0000 0000$_H$**

GLOBICLASSy (y = 0 - 1)
Input Class Register y, Global

($00A0_H + y * 0004_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	0	0	0	0	CME			0	0	0	STCE				
r	r	r	r	r	rw			r	r	r	rw				

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	CMS			0	0	0	STCS				
r	r	r	r	r	rw			r	r	r	rw				

Field	Bits	Type	Description
STCS	[4:0]	rw	**Sample Time Control for Standard Conversions** Number of additional clock cycles to be added to the minimum sample phase of 2 analog clock cycles: Coding and resulting sample time see **Table 19-9**. For conversions of external channels, the value from bitfield STCE can be used.
0	[7:5]	r	**Reserved, write 0, read as 0**
CMS	[10:8]	rw	**Conversion Mode for Standard Conversions** 000_B 12-bit conversion 001_B 10-bit conversion 010_B 8-bit conversion 011_B Reserved 100_B Reserved 101_B 10-bit fast compare mode 110_B Reserved 111_B Reserved
0	[15:11]	r	**Reserved, write 0, read as 0**
STCE	[20:16]	rw	**Sample Time Control for EMUX Conversions** Number of additional clock cycles to be added to the minimum sample phase of 2 analog clock cycles: Coding and resulting sample time see **Table 19-9**. For conversions of standard channels, the value from bitfield STCS is used.
0	[23:21]	r	**Reserved, write 0, read as 0**
CME	[26:24]	rw	**Conversion Mode for EMUX Conversions** 000_B 12-bit conversion 001_B 10-bit conversion 010_B 8-bit conversion 011_B Reserved 100_B Reserved 101_B 10-bit fast compare mode 110_B Reserved 111_B Reserved
0	[31:27]	r	**Reserved, write 0, read as 0**

Table 19-9 Sample Time Coding

STCS / STCE	Additional Clock Cycles	Sample Time
$0\ 0000_B$	0	2 / f_{ADCI}
$0\ 0001_B$	1	3 / f_{ADCI}
…	…	…
$0\ 1111_B$	15	17 / f_{ADCI}
$1\ 0000_B$	16	18 / f_{ADCI}
$1\ 0001_B$	32	34 / f_{ADCI}
…	…	…
$1\ 1110_B$	240	242 / f_{ADCI}
$1\ 1111_B$	256	258 / f_{ADCI}

19.13.6 Result Registers

The group result control registers select the behavior of the result registers of a given group.

G0RCRy (y = 0 - 15)
Group 0 Result Control Reg. y (0680$_H$ + y * 0004$_H$) Reset Value: 0000 0000$_H$
G1RCRy (y = 0 - 15)
Group 1 Result Control Reg. y (0A80$_H$ + y * 0004$_H$) Reset Value: 0000 0000$_H$
G2RCRy (y = 0 - 15)
Group 2 Result Control Reg. y (0E80$_H$ + y * 0004$_H$) Reset Value: 0000 0000$_H$
G3RCRy (y = 0 - 15)
Group 3 Result Control Reg. y (1280$_H$ + y * 0004$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SRG EN	**0**	**0**	**0**	**0**	**FEN**		**WFR**	**0**	**0**	**DMM**		**DRCTR**			
rw	r	r	r	r	rw		rw	r	r	rw		rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Field	Bits	Type	Description
0	[15:0]	r	**Reserved, write 0, read as 0**
DRCTR	[19:16]	rw	**Data Reduction Control** Defines how result values are stored/accumulated in this register for the final result. The data reduction counter DRC can be loaded from this bitfield. The function of bitfield DRCTR is determined by bitfield DMM.
DMM	[21:20]	rw	**Data Modification Mode** 00_B Standard data reduction (accumulation) 01_B Result filtering mode[1)] 10_B Difference mode 11_B Reserved See **"Data Modification" on Page 19-39**
0	[23:22]	r	**Reserved, write 0, read as 0**

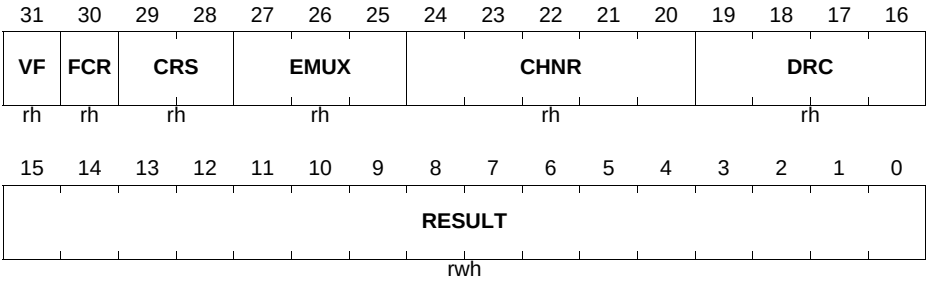
Field	Bits	Type	Description
WFR	24	rw	**Wait-for-Read Mode Enable** 0_B Overwrite mode 1_B Wait-for-read mode enabled for this register
FEN	[26:25]	rw	**FIFO Mode Enable** 00_B Separate result register 01_B Part of a FIFO structure: copy each new valid result $1X_B$ Reserved
0	[30:27]	r	**Reserved, write 0, read as 0**
SRGEN	31	rw	**Service Request Generation Enable** 0_B No service request 1_B Service request after a result event

1) The filter registers are cleared while bitfield DMM ≠ 01_B.

The group result registers provide a selectable storage location for all channels of a given group.

Note: The preset value used in fast compare mode is written to the respective result register. The debug result registers are not writable.

G0RESy (y = 0 - 15)
Group 0 Result Register y **(0700_H + y * 0004_H)** **Reset Value: $0000\ 0000_H$**
G1RESy (y = 0 - 15)
Group 1 Result Register y **($0B00_H$ + y * 0004_H)** **Reset Value: $0000\ 0000_H$**
G2RESy (y = 0 - 15)
Group 2 Result Register y **($0F00_H$ + y * 0004_H)** **Reset Value: $0000\ 0000_H$**
G3RESy (y = 0 - 15)
Group 3 Result Register y **(1300_H + y * 0004_H)** **Reset Value: $0000\ 0000_H$**

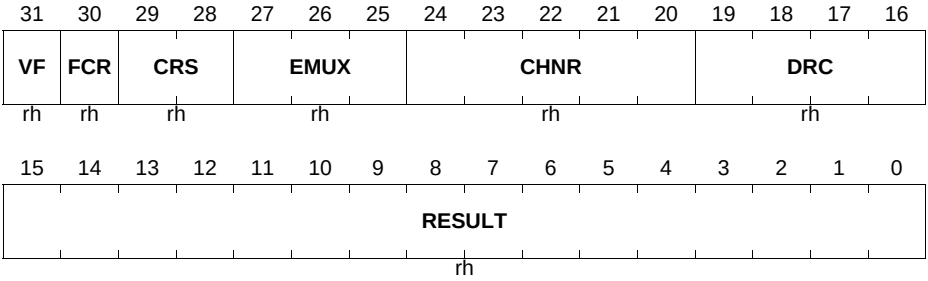


Field	Bits	Type	Description
RESULT	[15:0]	rwh	**Result of Most Recent Conversion** The position of the result bits within this bitfield depends on the configured operating mode. Please, refer to **Section 19.8.2**.
DRC	[19:16]	rh	**Data Reduction Counter** Indicates the number of values still to be accumulated for the final result. The final result is available and valid flag VF is set when bitfield DRC becomes zero (by decrementing or by reload). See **"Data Modification" on Page 19-39**
CHNR	[24:20]	rh	**Channel Number** Indicates the channel number corresponding to the value in bitfield RESULT.

Field	Bits	Type	Description
EMUX	[27:25]	rh	**External Multiplexer Setting** Indicates the setting of the external multiplexer, corresponding to the value in bitfield RESULT. *Note: Available in GxRES0 only. Use GxRES0 if EMUX information is required.*
CRS	[29:28]	rh	**Converted Request Source** Indicates the request source that as requested the conversion to which the result value in bitfield RESULT belongs. 00_B Request source 0 01_B Request source 1 10_B Request source 2 11_B Reserved
FCR	30	rh	**Fast Compare Result** Indicates the result of an operation in Fast Compare Mode. 0_B Signal level was below compare value 1_B Signal level was above compare value
VF	31	rh	**Valid Flag** Indicates a new result in bitfield RESULT or bit FCR. 0_B No new result available 1_B Bitfield RESULT has been updated with new result value and has not yet been read, or bit FCR has been updated

The debug view of the group result registers provides access to all result registers of a given group, however, without clearing the valid flag.

G0RESDy (y = 0 - 15)
Group 0 Result Reg. y, Debug (0780_H + y * 0004_H) Reset Value: $0000\ 0000_H$
G1RESDy (y = 0 - 15)
Group 1 Result Reg. y, Debug ($0B80_H$ + y * 0004_H) Reset Value: $0000\ 0000_H$
G2RESDy (y = 0 - 15)
Group 2 Result Reg. y, Debug ($0F80_H$ + y * 0004_H) Reset Value: $0000\ 0000_H$
G3RESDy (y = 0 - 15)
Group 3 Result Reg. y, Debug (1380_H + y * 0004_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
RESULT	[15:0]	rh	**Result of Most Recent Conversion** The position of the result bits within this bitfield depends on the configured operating mode. Please, refer to **Section 19.8.2**.
DRC	[19:16]	rh	**Data Reduction Counter** Indicates the number of values still to be accumulated for the final result. The final result is available and valid flag VF is set when bitfield DRC becomes zero (by decrementing or by reload). See **"Data Modification" on Page 19-39**
CHNR	[24:20]	rh	**Channel Number** Indicates the channel number corresponding to the value in bitfield RESULT.
EMUX	[27:25]	rh	**External Multiplexer Setting** Indicates the setting of the external multiplexer, corresponding to the value in bitfield RESULT. *Note: Available in GxRESD0 only. Use GxRESD0 if EMUX information is required.*

Field	Bits	Type	Description
CRS	[29:28]	rh	**Converted Request Source** Indicates the request source that as requested the conversion to which the result value in bitfield RESULT belongs. 00_B Request source 0 01_B Request source 1 10_B Request source 2 11_B Reserved
FCR	30	rh	**Fast Compare Result** Indicates the result of an operation in Fast Compare Mode. 0_B Signal level was below compare value 1_B Signal level was above compare value
VF	31	rh	**Valid Flag** Indicates a new result in bitfield RESULT or bit FCR. 0_B No new result available 1_B Bitfield RESULT has been updated with new result value and has not yet been read, or bit FCR has been updated

The global result control register selects the behavior of the global result register.

GLOBRCR
Global Result Control Register (0280_H) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SRGEN	**0**	**0**	**0**	**0**	**0**	**0**	**WFR**	**0**	**0**	**0**	**0**	**DRCTR**			
rw	r	r	r	r	r	r	rw	r	r	r	r	rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

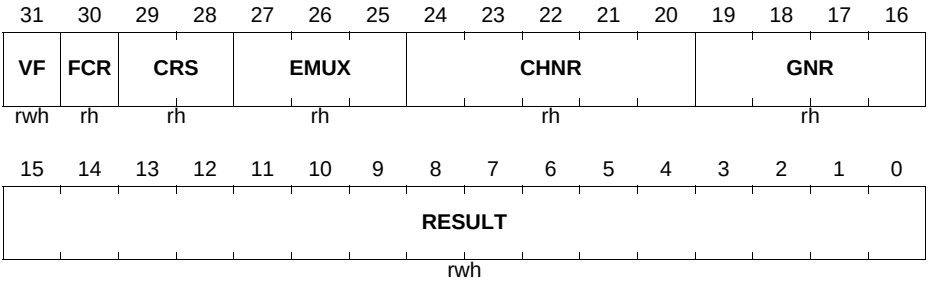
Field	Bits	Type	Description
0	[15:0]	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
DRCTR	[19:16]	rw	**Data Reduction Control** Defines how result values are stored/accumulated in this register for the final result. The data reduction counter DRC can be loaded from this bitfield. 0000_BData reduction disabled others: see **"Function of Bitfield DRCTR" on Page 19-39**[1)]
0	[23:20]	r	**Reserved, write 0, read as 0**
WFR	24	rw	**Wait-for-Read Mode Enable** 0_B Overwrite mode 1_B Wait-for-read mode enabled for this register
0	[30:25]	r	**Reserved, write 0, read as 0**
SRGEN	31	rw	**Service Request Generation Enable** 0_B No service request 1_B Service request after a result event

1) Only standard data reduction is available for the global result register, i.e. DMM is assumed as 00_B.

The global result register provides a common storage location for all channels of all groups.

GLOBRES
Global Result Register **(0300_H)** **Reset Value: 0000 0000_H**
GLOBRESD
Global Result Register, Debug **(0380_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
RESULT	[15:0]	rwh	**Result of most recent conversion** The position of the result bits within this bitfield depends on the configured operating mode.[1)] Please, refer to **Section 19.8.2**.
GNR	[19:16]	rh	**Group Number** Indicates the group to which the channel number in bitfield CHNR refers.
CHNR	[24:20]	rh	**Channel Number** Indicates the channel number corresponding to the value in bitfield RESULT.
EMUX	[27:25]	rh	**External Multiplexer Setting** Indicates the setting of the external multiplexer, corresponding to the value in bitfield RESULT.
CRS	[29:28]	rh	**Converted Request Source** Indicates the request source that as requested the conversion to which the result value in bitfield RESULT belongs.
FCR	30	rh	**Fast Compare Result** Indicates the result of an operation in Fast Compare Mode. 0_B Signal level was below compare value 1_B Signal level was above compare value
VF	31	rwh	**Valid Flag** Indicates a new result in bitfield RESULT or bit FCR. Read access: 0_B No new valid data available 1_B Bitfield RESULT contains valid data and has not yet been read, or bit FCR has been updated Write access:[1)] 0_B No effect 1_B Clear this valid flag and the data reduction counter (overrides a hardware set action)

1) Only writable in register GLOBRES, not in register GLOBRESD.

The valid flag register summarizes the valid flags of all result registers.

GxVFR (x = 0 - 3)
Valid Flag Register, Group x **(x * 0400_H + $05F8_H$)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

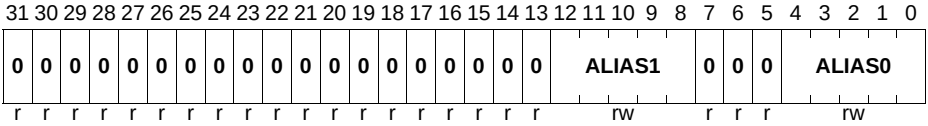
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VF15	**VF14**	**VF13**	**VF12**	**VF11**	**VF10**	**VF9**	**VF8**	**VF7**	**VF6**	**VF5**	**VF4**	**VF3**	**VF2**	**VF1**	**VF0**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
VFy **(y = 0 - 15)**	y	rwh	**Valid Flag of Result Register x** Indicates a new result in bitfield RESULT or in bit FCR. Read access: 0_B No new valid data available 1_B Result register x contains valid data and has not yet been read, or bit FCR has been updated Write access: 0_B No effect 1_B Clear this valid flag and bitfield DRC in register GxRESy/GLOBRES (overrides a hardware set action)
0	[31:16]	r	**Reserved, write 0, read as 0**

19.13.7 Miscellaneous Registers

The alias register can replace the channel numbers of channels CH0 and CH1 with another channel number. The reset value disables this redirection.

GxALIAS (x = 0 - 3)
Alias Register, Group x **($x * 0400_H + 04B0_H$)** **Reset Value: $0000\ 0100_H$**



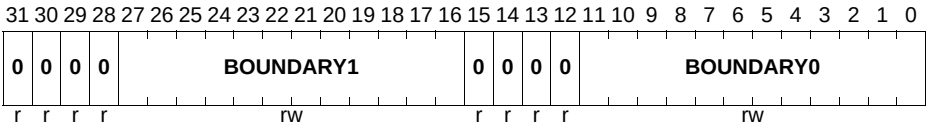
Field	Bits	Type	Description
ALIAS0	[4:0]	rw	**Alias Value for CH0 Conversion Requests** Indicates the channel that is converted instead of channel CH0. The conversion is done with the settings defined for channel CH0.
0	[7:5]	r	**Reserved, write 0, read as 0**
ALIAS1	[12:8]	rw	**Alias Value for CH1 Conversion Requests** Indicates the channel that is converted instead of channel CH1. The conversion is done with the settings defined for channel CH1.
0	[31:13]	r	**Reserved, write 0, read as 0**

The local boundary register GxBOUND defines group-specific boundary values or delta limits for Fast Compare Mode.
The global boundary register GLOBBOUND defines general compare values for all channels.

Depending on the conversion width, the respective left 12/10/8 bits of a bitfield are used. For 10/8-bit results, the lower 2/4 bits must be zero!

GxBOUND (x = 0 - 3)
Boundary Select Register, Group x
($x * 0400_H + 04B8_H$) **Reset Value: 0000 0000$_H$**

GLOBBOUND
Global Boundary Select Register ($00B8_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
BOUNDARY0	[11:0]	rw	**Boundary Value 0 for Limit Checking** Standard Mode: This value is compared against the left-aligned conversion result. Fast Compare Mode: This value is added to the reference value (upper delta).
0	[15:12]	r	**Reserved, write 0, read as 0**
BOUNDARY1	[27:16]	rw	**Boundary Value 1 for Limit Checking** Standard Mode: This value is compared against the left-aligned conversion result. Fast Compare Mode: This value is subtracted from the reference value (lower delta).
0	[31:28]	r	**Reserved, write 0, read as 0**

The Boundary Flag Register holds the boundary flags themselves together with bits to select the activation condition and the output signal polarity for each flag.

GxBFL (x = 0 - 3)
Boundary Flag Register, Group x

(x * 0400_H + $04C8_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**BFE 3**	**BFE 2**	**BFE 1**	**BFE 0**
r	r	r	r	r	r	r	r	r	r	r	r	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**BFL 3**	**BFL 2**	**BFL 1**	**BFL 0**
r	r	r	r	r	r	r	r	r	r	r	r	rh	rh	rh	rh

Field	Bits	Type	Description
BFLy (y = 0 - 3)	y	rh	**Boundary Flag y** 0_B Passive state: result has not yet crossed the activation boundary (see bitfield BFAy), or selected gate signal is inactive, or this boundary flag is disabled 1_B Active state: result has crossed the activation boundary
0	[15:4]	r	**Reserved, write 0, read as 0**
BFEy (y = 0 - 3)	16 + y	rw	**Enable Bit for Boundary Flag y** 0_B Output 0 on this channel 1_B Output BFLy on this channel
0	[31:20]	r	**Reserved, write 0, read as 0**

The Synchronization Control Register controls the synchronization of kernels for parallel conversions.

Note: Program register GxSYNCTR only while bitfield GxARBCFG.ANONS = 00_B in all ADC kernels of the conversion group. Set the master's bitfield ANONC to 11_B afterwards.

GxSYNCTR (x = 0 - 3)
Synchronization Control Register, Group x

(x * 0400_H + $04C0_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**EVALR3**	**EVALR2**	**EVALR1**	**0**	**0**	**STSEL**	
r	r	r	r	r	r	r	r	r	rw	rw	rw	r	r	rw	

Field	Bits	Type	Description
STSEL	[1:0]	rw	**Start Selection** Controls the synchronization mechanism of the ADC kernel. 00_B Kernel is synchronization master: Use own bitfield GxARBCFG.ANONC 01_B Kernel is synchronization slave: Control information from input CI1 10_B Kernel is synchronization slave: Control information from input CI2 11_B Kernel is synchronization slave: Control information from input CI3 *Note: Control inputs CIx see **Figure 19-23**, connected kernels see **Table 19-11**.*
0	[3:2]	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
EVALR1, EVALR2, EVALR3	4, 5, 6	rw	**Evaluate Ready Input Rx** Enables the ready input signal for a kernel of a conversion group. 0_B No ready input control 1_B Ready input Rx is considered for the start of a parallel conversion of this conversion group
0	[31:7]	r	**Reserved, write 0, read as 0**

GLOBTF
Global Test Functions Register (0160_H) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**MDWC**	**0**	**0**	**0**	**0**	**0**	**0**	**PDD**
r	r	r	r	r	r	r	r	w	r	r	r	r	r	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CDWC	**0**	**0**	**0**	**0**	**CDSEL**		**CDEN**	**CDGR**				**0**	**0**	**0**	**0**
w	r	r	r	r	rw		rw	rw				r	r	r	r

Field	Bits	Type	Description
0	[3:0]	r	**Reserved, write 0, read as 0**
CDGR	[7:4]	rw	**Converter Diagnostics Group** Defines the group number to be used for converter diagnostics conversions.
CDEN	8	rw	**Converter Diagnostics Enable** 0_B All diagnostic pull devices are disconnected 1_B Diagnostic pull devices connected as selected by bitfield CDSEL
CDSEL	[10:9]	rw	**Converter Diagnostics Pull-Devices Select** 00_B Connected to VAREF 01_B Connected to VAGND 10_B Connected to 1/3rd VAREF 11_B Connected to 2/3rd VAREF
0	[14:11]	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
CDWC	15	w	**Write Control for Conversion Diagnostics** 0_B No write access to parameters 1_B Bitfields CDSEL, CDEN, CDGR can be written
PDD	16	rw	**Pull-Down Diagnostics Enable** 0_B Disconnected 1_B The pull-down diagnostics device is active *Note: Channels with pull-down diagnostics device are marked in **Table 19-12**.*
0	[22:17]	r	**Reserved, write 0, read as 0**
MDWC	23	w	**Write Control for Multiplexer Diagnostics** 0_B No write access to parameters 1_B Bitfield PDD can be written
0	[31:24]	r	**Reserved, write 0, read as 0**

GxEMUXCTR (x = 0 - 3)
External Multiplexer Control Register, Group x

(x * 0400_H + $05F0_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EMX WC	EMX CSS	EMX ST	EMX COD	EMUX MODE		0	0	0	0	0	EMUX CH				
w	rw	rw	rw	rw		r	r	r	r	r	rw				

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	EMUX ACT			0	0	0	0	0	EMUX SET		
r	r	r	r	r	rh			r	r	r	r	r	rw		

Field	Bits	Type	Description
EMUXSET	[2:0]	rw	**External Multiplexer Start Selection**[1)] Defines the initial setting for the external multiplexer.
0	[7:3]	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
EMUXACT	[10:8]	rh	**External Multiplexer Actual Selection** Defines the current value for the external multiplexer selection. This bitfield is loaded from bitfield EMUXSET and modified according to the operating mode selected by bitfield EMUXMODE.
0	[15:11]	r	**Reserved, write 0, read as 0**
EMUXCH	[20:16]	rw	**External Multiplexer Channel Select** Defines the channel(s) to which the external multiplexer control is applied.
0	[25:21]	r	**Reserved, write 0, read as 0**
EMUXMODE	[27:26]	rw	**External Multiplexer Mode** 00_B Software control (no hardware action) 01_B Steady mode (use EMUXSET value) 10_B Single-step mode[1)2)] 11_B Sequence mode[1)]
EMXCOD	28	rw	**External Multiplexer Coding Scheme** 0_B Output the channel number in binary code 1_B Output the channel number in Gray code
EMXST	29	rw	**External Multiplexer Sample Time Control** 0_B Use STCE whenever the setting changes 1_B Use STCE for each conversion of an external channel
EMXCSS	30	r	**External Multiplexer Channel Selection Style** 0_B Channel number: Bitfield MUXCH selects an arbitrary channel 1_B Channel enable: Each bit of bitfield EMUXCH selects the associated channel for EMUX control
EMXWC	31	w	**Write Control for EMUX Configuration** 0_B No write access to EMUX cfg. 1_B Bitfields EMXMODE, EMXCOD, EMXST, EMXCSS can be written

1) For single-step mode and sequence mode: Select the start value before selecting the respective mode.

2) Single-step mode modifies the EMUX channel number each time an EMUX-enabled channel is converted. Therefore, single-step mode works best with a single channel, because otherwise some external channels may be skipped.

Register EMUXSEL is a global register which assigns an arbitrary group to each of the EMUX interfaces.

EMUXSEL
External Multiplexer Select Register

(03F0$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**		**EMUX GRP1**				**EMUX GRP0**		
r	r	r	r	r	r	r	r		rw				rw		

Field	Bits	Type	Description
EMUXGRP0, EMUXGRP1	[3:0], [7:4]	rw	**External Multiplexer Group for Interface x** Defines the group whose external multiplexer control signals are routed to EMUX interface x.[1)]
0	[31:8]	r	**Reserved, write 0, read as 0**

1) The pins that are associated with each EMUX interface are listed in **Table 19-13 "Digital Connections in the XMC4500" on Page 19-131**.

19.13.8 Service Request Registers

GxSEFLAG (x = 0 - 3)
Source Event Flag Register, Group x

(x * 0400$_H$ + 0588$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**SEV 1**	**SEV 0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	rwh	rwh

Field	Bits	Type	Description
SEV0, SEV1	0, 1	rwh	**Source Event 0/1** 0_B No source event 1_B A source event has occurred
0	[31:2]	r	**Reserved, write 0, read as 0**

Note: Software can set all flags in register GxSEFLAG and trigger the corresponding event by writing 1 to the respective bit. Writing 0 has no effect.
Software can clear all flags in register GxSEFLAG by writing 1 to the respective bit in register GxSEFCLR.

GxCEFLAG (x = 0 - 3)
Channel Event Flag Register, Group x

(x * 0400_H + 0580_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**CEV 7**	**CEV 6**	**CEV 5**	**CEV 4**	**CEV 3**	**CEV 2**	**CEV 1**	**CEV 0**
r	r	r	r	r	r	r	r	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
CEVy (y = 0 - 7)	y	rwh	**Channel Event for Channel y** 0_B No channel event 1_B A channel event has occurred
0	[31:8]	r	**Reserved, write 0, read as 0**

Note: Software can set all flags in register GxCEFLAG and trigger the corresponding event by writing 1 to the respective bit. Writing 0 has no effect.
Software can clear all flags in register GxCEFLAG by writing 1 to the respective bit in register GxCEFCLR.

GxREFLAG (x = 0 - 3)
Result Event Flag Register, Group x

(x * 0400$_H$ + 0584$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REV 15	**REV 14**	**REV 13**	**REV 12**	**REV 11**	**REV 10**	**REV 9**	**REV 8**	**REV 7**	**REV 6**	**REV 5**	**REV 4**	**REV 3**	**REV 2**	**REV 1**	**REV 0**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
REVy (y = 0 - 15)	y	rwh	**Result Event for Result Register y** 0_B No result event 1_B New result was stored in register GxRESy
0	[31:16]	r	**Reserved, write 0, read as 0**

Note: Software can set all flags in register GxREFLAG and trigger the corresponding event by writing 1 to the respective bit. Writing 0 has no effect.
Software can clear all flags in register GxREFLAG by writing 1 to the respective bit in register GxREFCLR.

GxSEFCLR (x = 0 - 3)
Source Event Flag Clear Register, Group x

(x * 0400$_H$ + 0598$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**SEV 1**	**SEV 0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	w	w

Field	Bits	Type	Description
SEV0, SEV1	0, 1	w	**Clear Source Event 0/1** 0_B No action 1_B Clear the source event flag in GxSEFLAG
0	[31:2]	r	**Reserved, write 0, read as 0**

GxCEFCLR (x = 0 - 3)
Channel Event Flag Clear Register, Group x

(x * 0400_H + 0590_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**CEV 7**	**CEV 6**	**CEV 5**	**CEV 4**	**CEV 3**	**CEV 2**	**CEV 1**	**CEV 0**
r	r	r	r	r	r	r	r	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
CEVy (y = 0 - 7)	y	w	**Clear Channel Event for Channel y** 0_B No action 1_B Clear the channel event flag in GxCEFLAG
0	[31:8]	r	**Reserved, write 0, read as 0**

GxREFCLR (x = 0 - 3)
Result Event Flag Clear Register, Group x

(x * 0400_H + 0594_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REV 15	**REV 14**	**REV 13**	**REV 12**	**REV 11**	**REV 10**	**REV 9**	**REV 8**	**REV 7**	**REV 6**	**REV 5**	**REV 4**	**REV 3**	**REV 2**	**REV 1**	**REV 0**
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
REVy (y = 0 - 15)	y	w	**Clear Result Event for Result Register y** 0_B No action 1_B Clear the result event flag in GxREFLAG
0	[31:16]	r	**Reserved, write 0, read as 0**

GLOBEFLAG
Global Event Flag Register

($00E0_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**REV GLB CLR**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**SEV GLB CLR**
r	r	r	r	r	r	r	w	r	r	r	r	r	r	r	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**REV GLB**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**SEV GLB**
r	r	r	r	r	r	r	rwh	r	r	r	r	r	r	r	rwh

Field	Bits	Type	Description
SEVGLB	0	rwh	**Source Event (Background)** 0_B No source event 1_B A source event has occurred

Field	Bits	Type	Description
0	[7:1]	r	**Reserved, write 0, read as 0**
REVGLB	8	rwh	**Global Result Event** 0_B No result event 1_B New result was stored in register GLOBRES
0	[15:9]	r	**Reserved, write 0, read as 0**
SEVGLBCLR	16	w	**Clear Source Event (Background)** 0_B No action 1_B Clear the source event flag SEVGLB
0	[23:17]	r	**Reserved, write 0, read as 0**
REVGLBCLR	24	w	**Clear Global Result Event** 0_B No action 1_B Clear the result event flag REVGLB
0	[31:25]	r	**Reserved, write 0, read as 0**

Note: Software can set flags REVGLB and SEVGLB and trigger the corresponding event by writing 1 to the respective bit. Writing 0 has no effect.
Software can clear these flags by writing 1 to bit REVGLBCLR and SECGLBCLR, respectively.

GxSEVNP (x = 0 - 3)
Source Event Node Pointer Register, Group x
(x * 0400_H + $05C0_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**SEV1NP**				**SEV0NP**			
r	r	r	r	r	r	r	r	rw				rw			

Field	Bits	Type	Description
SEV0NP, SEV1NP	[3:0], [7:4]	rw	**Service Request Node Pointer Source Event i** Routes the corresponding event trigger to one of the service request lines (nodes). 0000_B Select service request line 0 of group x … 0011_B Select service request line 3 of group x 0100_B Select shared service request line 0 … 0111_B Select shared service request line 3 $1xxx_B$ Reserved *Note: For shared service request lines see common groups in **Table 19-10**.*
0	[31:8]	r	**Reserved, write 0, read as 0**

GxCEVNP0 (x = 0 - 3)
Channel Event Node Pointer Register 0, Group x
($x * 0400_H + 05A0_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CEV7NP				CEV6NP				CEV5NP				CEV4NP			
rw				rw				rw				rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CEV3NP				CEV2NP				CEV1NP				CEV0NP			
rw				rw				rw				rw			

Field	Bits	Type	Description
CEV0NP, **CEV1NP,** **CEV2NP,** **CEV3NP,** **CEV4NP,** **CEV5NP,** **CEV6NP,** **CEV7NP**	[3:0], [7:4], [11:8], [15:12], [19:16], [23:20], [27:24], [31:28]	rw	**Service Request Node Pointer Channel Event i** Routes the corresponding event trigger to one of the service request lines (nodes). 0000_B Select service request line 0 of group x … 0011_B Select service request line 3 of group x 0100_B Select shared service request line 0 … 0111_B Select shared service request line 3 $1xxx_B$ Reserved *Note: For shared service request lines see common groups in **Table 19-10**.*

GxREVNP0 (x = 0 - 3)
Result Event Node Pointer Register 0, Group x
(x * 0400_H + $05B0_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
REV7NP				**REV6NP**				**REV5NP**				**REV4NP**			
rw				rw				rw				rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REV3NP				**REV2NP**				**REV1NP**				**REV0NP**			
rw				rw				rw				rw			

Field	Bits	Type	Description
REV0NP, **REV1NP,** **REV2NP,** **REV3NP,** **REV4NP,** **REV5NP,** **REV6NP,** **REV7NP**	[3:0], [7:4], [11:8], [15:12], [19:16], [23:20], [27:24], [31:28]	rw	**Service Request Node Pointer Result Event i** Routes the corresponding event trigger to one of the service request lines (nodes). 0000_B Select service request line 0 of group x … 0011_B Select service request line 3 of group x 0100_B Select shared service request line 0 … 0111_B Select shared service request line 3 $1xxx_B$ Reserved *Note: For shared service request lines see common groups in **Table 19-10**.*

GxREVNP1 (x = 0 - 3)
Result Event Node Pointer Register 1, Group x
(x * 0400_H + $05B4_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
REV15NP				**REV14NP**				**REV13NP**				**REV12NP**			
rw				rw				rw				rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REV11NP				**REV10NP**				**REV9NP**				**REV8NP**			
rw				rw				rw				rw			

Field	Bits	Type	Description
REV8NP, **REV9NP,** **REV10NP,** **REV11NP,** **REV12NP,** **REV13NP,** **REV14NP,** **REV15NP**	[3:0], [7:4], [11:8], [15:12], [19:16], [23:20], [27:24], [31:28]	rw	**Service Request Node Pointer Result Event i** Routes the corresponding event trigger to one of the service request lines (nodes). 0000_B Select service request line 0 of group x … 0011_B Select service request line 3 of group x 0100_B Select shared service request line 0 … 0111_B Select shared service request line 3 $1xxx_B$ Reserved *Note: For shared service request lines see common groups in **Table 19-10**.*

GLOBEVNP
Global Event Node Pointer Register

(0140_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**REV0NP**			
r	r	r	r	r	r	r	r	r	r	r	r	rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**SEV0NP**			
r	r	r	r	r	r	r	r	r	r	r	r	rw			

Field	Bits	Type	Description
SEV0NP	[3:0]	rw	**Service Request Node Pointer Backgr. Source** Routes the corresponding event trigger to one of the service request lines (nodes). 0000_B Select shared service request line 0 of common service request group 0 … 0011_B Select shared service request line 3 of common service request group 0 0100_B Select shared service request line 0 of common service request group 1 … 0111_B Select shared service request line 3 of common service request group 1 $1xxx_B$ Reserved *Note: For shared service request lines see common groups in **Table 19-10**.*
0	[15:4]	r	**Reserved, write 0, read as 0**
REV0NP	[19:16]	rw	**Service Request Node Pointer Backgr. Result** Routes the corresponding event trigger to one of the service request lines (nodes). 0000_B Select shared service request line 0 of common service request group 0 … 0011_B Select shared service request line 3 of common service request group 0 0100_B Select shared service request line 0 of common service request group 1 … 0111_B Select shared service request line 3 of common service request group 1 $1xxx_B$ Reserved *Note: For shared service request lines see common groups in **Table 19-10**.*
0	[31:20]	r	**Reserved, write 0, read as 0**

GxSRACT (x = 0 - 3)
Service Request Software Activation Trigger, Group x
(x * 0400_H + $05C8_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**ASSR3**	**ASSR2**	**ASSR1**	**ASSR0**	**0**	**0**	**0**	**0**	**AGSR3**	**AGSR2**	**AGSR1**	**AGSR0**
r	r	r	r	w	w	w	w	r	r	r	r	w	w	w	w

Field	Bits	Type	Description
AGSRy (y = 0 - 3)	y	w	**Activate Group Service Request Node y** 0_B No action 1_B Activate the associated service request line
0	[7:4]	r	**Reserved, write 0, read as 0**
ASSRy (y = 0 - 3)	8 + y	w	**Activate Shared Service Request Node y** 0_B No action 1_B Activate the associated service request line
0	[31:12]	r	**Reserved, write 0, read as 0**

19.14 Interconnects

This section describes the actual implementation of the ADC module into the XMC4500, i.e. the incorporation into the microcontroller system.

19.14.1 Product-Specific Configuration

The functional description describes the features and operating modes of the A/D Converters in a general way. This section summarizes the configuration that is available in this product (XMC4500).

Each converter group is equipped with a separate analog converter module and a dedicated analog input multiplexer.

Table 19-10 General Converter Configuration in the XMC4500

Converter Group	Input Channels	Channels with Alternate Reference	12-bit Performance	Common Service Request Group
G0	0 … 7	4	Calibrated	C0
G1	0 … 7	8	Calibrated	C0
G2	0 … 7	8	Calibrated	C0
G3	0 … 7	8	Calibrated	C0

Synchronization Groups in the XMC4500

The converter kernels in the XMC4500 can be connected to synchronization groups to achieve parallel conversion of several input channels.

Table 19-11 summarizes which kernels can be synchronized for parallel conversions.

Table 19-11 Synchronization Groups in the XMC4500

ADC Kernel	Synchr. Group	Master selected by control input CIx[1]			
		CI0[2]	CI1	CI2	CI3
ADC00	A	ADC00	ADC01	ADC02	ADC03
ADC01	A	ADC01	ADC00	ADC02	ADC03
ADC02	A	ADC02	ADC00	ADC01	ADC03
ADC03	A	ADC03	ADC00	ADC01	ADC02

1) The control input is selected by bitfield STSEL in register **GxSYNCTR (x = 0 - 3)**. Select the corresponding ready inputs accordingly by bits EVALRx.

2) Control input CI0 always selects the own control signals of the corresponding ADC kernel. This selection is meant for the synchronization master or for stand-alone operation.

19.14.2 Analog Module Connections in the XMC4500

The VADC module accepts a number of analog input signals. The analog input multiplexers select the input channels to be converted from the signals available in this product.

The exact number of analog input channels and the available connection to port pins depend on the employed product type (see also **Table 19-10**). A summary of channels enclosing all versions of the XMC4500 can be found in **Table 19-12**.

Input channels marked “PDD” provide a pull-down device for pull-down diagnostics.

Input channels marked “noAltref” cannot select the alternate reference voltage from channel 0 of the corresponding converter.

Table 19-12 Analog Connections in the XMC4500

Signal	Dir.	Source/Destin.	Description
V_{AREF}	I	VAREF1 / VAREF2	positive analog reference
V_{AGND}	I	VAGND1 / VAGND2	negative analog reference
G0CH0	I	P14.0	analog input channel 0 of group 0
G0CH1	I	P14.1	analog input channel 1 of group 0
G0CH2	I	P14.2	analog input channel 2 of group 0
G0CH3	I	P14.3	analog input channel 3 of group 0
G0CH4	I	P14.4	analog input channel 4 of group 0
G0CH5	I	P14.5	analog input channel 5 of group 0
G0CH6	I	P14.6	analog input channel 6 of group 0
G0CH7 (PDD)	I	P14.7	analog input channel 7 of group 0
G1CH0	I	P14.8	analog input channel 0 of group 1
G1CH1	I	P14.9	analog input channel 1 of group 1
G1CH2	I	P14.2	analog input channel 2 of group 1
G1CH3	I	P14.3	analog input channel 3 of group 1
G1CH4	I	P14.12	analog input channel 4 of group 1
G1CH5	I	P14.13	analog input channel 5 of group 1
G1CH6	I	P14.14	analog input channel 6 of group 1
G1CH7 (PDD)	I	P14.15	analog input channel 7 of group 1
G2CH0	I	P14.4	analog input channel 0 of group 2

Table 19-12 Analog Connections in the XMC4500 (cont'd)

Signal	Dir.	Source/Destin.	Description
G2CH1	I	P14.5	analog input channel 1 of group 2
G2CH2	I	P15.2	analog input channel 2 of group 2
G2CH3	I	P15.3	analog input channel 3 of group 2
G2CH4	I	P15.4	analog input channel 4 of group 2
G2CH5	I	P15.5	analog input channel 5 of group 2
G2CH6	I	P15.6	analog input channel 6 of group 2
G2CH7 (PDD)	I	P15.7	analog input channel 7 of group 2
G3CH0	I	P15.8	analog input channel 0 of group 3
G3CH1	I	P15.9	analog input channel 1 of group 3
G3CH2	I	P14.8	analog input channel 2 of group 3
G3CH3	I	P14.9	analog input channel 3 of group 3
G3CH4	I	P15.12	analog input channel 4 of group 3
G3CH5	I	P15.13	analog input channel 5 of group 3
G3CH6	I	P15.14	analog input channel 6 of group 3
G3CH7 (PDD)	I	P15.15	analog input channel 7 of group 3

19.14.3 Digital Module Connections in the XMC4500

The VADC module accepts a number of digital input signals and generates a number of output signals. This section summarizes the connection of these signals to other on-chip modules or to external resources via port pins.

Table 19-13 Digital Connections in the XMC4500

Signal	Dir.	Source/Destin.	Description
Gate Inputs for Each Group			
VADC.GxREQGTA	I	CCU40.ST3	Gating input A
VADC.GxREQGTB	I	CCU41.ST3	Gating input B
VADC.GxREQGTC	I	CCU40.SR0	Gating input C
VADC.GxREQGTD	I	CCU41.SR1	Gating input D
VADC.GxREQGTE	I	CCU80.ST3A	Gating input E
VADC.GxREQGTF	I	CCU80.ST3B	Gating input F
VADC.GxREQGTG	I	CCU81.ST3A	Gating input G
VADC.GxREQGTH	I	CCU81.ST3B	Gating input H
VADC.G0REQGTI	I (s)	DAC0.SGN	Gating input I
VADC.G1REQGTI	I (s)	DAC1.SGN	Gating input I
VADC.G2REQGTI	I (s)	DAC0.SGN	Gating input I
VADC.G3REQGTI	I (s)	DAC1.SGN	Gating input I
VADC.GxREQGTJ	I (s)	LEDTS.FN	Gating input J
VADC.G0REQGTK	I (s)	VADC.G1BFLOUT0	Gating input K
VADC.G1REQGTK	I (s)	VADC.G0BFLOUT0	Gating input K
VADC.G2REQGTK	I (s)	VADC.G3BFLOUT0	Gating input K
VADC.G3REQGTK	I (s)	VADC.G2BFLOUT0	Gating input K
VADC.G0REQGTL	I (s)	VADC.G3SAMPLE	Gating input L
VADC.G1REQGTL	I (s)	VADC.G0SAMPLE	Gating input L
VADC.G2REQGTL	I (s)	VADC.G1SAMPLE	Gating input L
VADC.G3REQGTL	I (s)	VADC.G2SAMPLE	Gating input L
VADC.GxREQGTM	I	CCU80.SR0	Gating input M
VADC.GxREQGTN	I	CCU80.SR1	Gating input N
VADC.GxREQGTO	I	ERU1.PDOUT0	Gating input O
VADC.GxREQGTP	I	ERU1.PDOUT1	Gating input E

Table 19-13 Digital Connections in the XMC4500 (cont'd)

Signal	Dir.	Source/Destin.	Description
VADC.GxREQGTySEL	O	VADC.GxREQTRyP [1)]	Selected gating signal of the respective source
Gate Inputs for Global Background Source			
VADC.BGREQGTA	I	CCU40.ST3	Gating input A, background source
VADC.BGREQGTB	I	CCU41.ST3	Gating input B, background source
VADC.BGREQGTC	I	CCU40.SR0	Gating input C, background source
VADC.BGREQGTD	I	CCU41.SR1	Gating input D, background source
VADC.BGREQGTE	I	CCU80.ST3A	Gating input E, background source
VADC.BGREQGTF	I	CCU80.ST3B	Gating input F, background source
VADC.BGREQGTG	I	CCU81.ST3A	Gating input G, background source
VADC.BGREQGTH	I	CCU81.ST3B	Gating input H, background source
VADC.BGREQGTI	I (s)	DAC0.SGN	Gating input I, background source
VADC.BGREQGTJ	I (s)	LEDTS.FN	Gating input J, background source
VADC.BGREQGTK	I (s)	VADC.G1BFLOUT0	Gating input K, background source
VADC.BGREQGTL	I (s)	-	Gating input L, background source
VADC.BGREQGTM	I	CCU80.SR0	Gating input M, background source
VADC.BGREQGTN	I	CCU80.SR1	Gating input N, background source
VADC.BGREQGTO	I	ERU1.PDOUT0	Gating input O, background source
VADC.BGREQGTP	I	ERU1.PDOUT1	Gating input E, background source
VADC.BGREQGTSEL	O	VADC.BGREQTRP [1)]	Selected gating signal
Trigger Inputs for Each Group			
VADC.GxREQTRA	I	CCU40.SR2	Trigger input A
VADC.GxREQTRB	I	CCU40.SR3	Trigger input B
VADC.GxREQTRC	I	CCU41.SR2	Trigger input C
VADC.GxREQTRD	I	CCU41.SR3	Trigger input D
VADC.GxREQTRE	I	CCU42.SR3	Trigger input E
VADC.GxREQTRF	I	CCU43.SR3	Trigger input F
VADC.GxREQTRG	I	-	Trigger input G
VADC.GxREQTRH	I	-	Trigger input H
VADC.GxREQTRI	I (s)	CCU80.SR2	Trigger input I

Table 19-13 Digital Connections in the XMC4500 (cont'd)

Signal	Dir.	Source/Destin.	Description
VADC.GxREQTRJ	I (s)	CCU80.SR3	Trigger input J
VADC.GxREQTRK	I (s)	CCU81.SR2	Trigger input K
VADC.GxREQTRL	I (s)	CCU81.SR3	Trigger input L
VADC.GxREQTRM	I	ERU1.IOUT0	Trigger input M
VADC.G0REQTRN	I	ERU1.IOUT1	Trigger input N
VADC.G1REQTRN	I	ERU1.IOUT1	Trigger input N
VADC.G2REQTRN	I	ERU1.IOUT2	Trigger input N
VADC.G3REQTRN	I	ERU1.IOUT2	Trigger input N
VADC.G0REQTRO	I	POSIF0.SR1	Trigger input O
VADC.G1REQTRO	I	POSIF1.SR1	Trigger input O
VADC.G2REQTRO	I	POSIF0.SR1	Trigger input O
VADC.G3REQTRO	I	POSIF1.SR1	Trigger input O
VADC.GxREQTRyP	I	VADC.GxREQGTySEL[1)]	Extend triggers to selected gating input of the respective source
VADC.GxREQTRySEL	O	-	Selected trigger signal of the respective source
Trigger Inputs for Global Background Source			
VADC.BGREQTRA	I	CCU40.SR2	Trigger input A, background source
VADC.BGREQTRB	I	CCU40.SR3	Trigger input B, background source
VADC.BGREQTRC	I	CCU41.SR2	Trigger input C, background source
VADC.BGREQTRD	I	CCU41.SR3	Trigger input D, background source
VADC.BGREQTRE	I	CCU42.SR3	Trigger input E, background source
VADC.BGREQTRF	I	CCU43.SR3	Trigger input F, background source
VADC.BGREQTRG	I	-	Trigger input G, background source
VADC.BGREQTRH	I	-	Trigger input H, background source
VADC.BGREQTRI	I (s)	CCU80.SR2	Trigger input I, background source
VADC.BGREQTRJ	I (s)	CCU80.SR3	Trigger input J, background source
VADC.BGREQTRK	I (s)	CCU81.SR2	Trigger input K, background source
VADC.BGREQTRL	I (s)	CCU81.SR3	Trigger input L, background source
VADC.BGREQTRM	I	ERU1.IOUT0	Trigger input M, background source
VADC.BGREQTRN	I	ERU1.IOUT1	Trigger input N, background source

Table 19-13 Digital Connections in the XMC4500 (cont'd)

Signal	Dir.	Source/Destin.	Description
VADC.BGREQTRO	I	POSIF0.SR1	Trigger input O, background source
VADC.BGREQTRP	I	VADC.BGREQGTSEL[1)]	Extend triggers to selected gating input of the background source
VADC.BGREQTRSEL	O	-	Selected trigger signal of the background source
System-Internal Connections			
VADC.G0SAMPLE	O	VADC.G1REQGTL	Indicates the input signal sample phase
VADC.G1SAMPLE	O	VADC.G2REQGTL	Indicates the input signal sample phase
VADC.G2SAMPLE	O	VADC.G3REQGTL	Indicates the input signal sample phase
VADC.G3SAMPLE	O	VADC.G0REQGTL	Indicates the input signal sample phase
VADC.G0ARBCNT	O	CCU40.IN3G	Outputs a (count) pulse for each arbiter round
VADC.G1ARBCNT	O	CCU41.IN3G	Outputs a (count) pulse for each arbiter round
VADC.G2ARBCNT	O	CCU42.IN3G	Outputs a (count) pulse for each arbiter round
VADC.G3ARBCNT	O	CCU43.IN3G	Outputs a (count) pulse for each arbiter round
VADC.GxSR0	O	NVIC, GPDMA	Service request 0 of group x
VADC.GxSR1	O	NVIC, GPDMA	Service request 1of group x
VADC.GxSR2	O	NVIC, GPDMA	Service request 2 of group x
VADC.G0SR3	O	NVIC, GPDMA CCU80.IN0F CCU81.IN0J	Service request 3 of group 0
VADC.G1SR3	O	NVIC, GPDMA CCU81.IN1J	Service request 3 of group 1
VADC.G2SR3	O	NVIC, GPDMA CCU81.IN2J	Service request 3 of group 2
VADC.G3SR3	O	NVIC, GPDMA CCU81.IN3J	Service request 3 of group 3

Table 19-13 Digital Connections in the XMC4500 (cont'd)

Signal	Dir.	Source/Destin.	Description
VADC.C0SR0	O	NVIC, GPDMA ERU1.OGU01 POSIF0.IN2C	Service request 0 of common block 0
VADC.C0SR1	O	NVIC, GPDMA ERU1.OGU11 POSIF1.IN2C	Service request 1 of common block 0
VADC.C0SR2	O	NVIC, GPDMA ERU1.OGU21	Service request 2 of common block 0
VADC.C0SR3	O	NVIC, GPDMA ERU1.OGU31	Service request 3 of common block 0
VADC.EMUX00	O	GPIO	Control of external analog multiplexer interface 0
VADC.EMUX01	O	GPIO	
VADC.EMUX02	O	GPIO	
VADC.EMUX10	O	GPIO	Control of external analog multiplexer interface 1
VADC.EMUX11	O	GPIO	
VADC.EMUX12	O	GPIO	
VADC.G0BFLOUT0	O	VADC.G1REQGTK VADC.BGREQGTK CCU41.IN0L CCU80.IN0I CCU43.IN0H	Boundary flag 0 output of group 0
VADC.G1BFLOUT0	O	VADC.G0REQGTK CCU43.IN1H POSIF0.IN0C POSIF1.IN0C	Boundary flag 0 output of group 1
VADC.G2BFLOUT0	O	VADC.G3REQGTK CCU43.IN2H	Boundary flag 0 output of group 2
VADC.G3BFLOUT0	O	VADC.G2REQGTK CCU43.IN3H	Boundary flag 0 output of group 3
VADC.GxBFL0	O	-	Boundary flag 0 level of group x
VADC.GxBFSEL0	I	0	Boundary flag 0 (group x) source select
VADC.GxBFDAT0	I	0	Boundary flag 0 (group x) alternate data

Table 19-13 Digital Connections in the XMC4500 (cont'd)

Signal	Dir.	Source/Destin.	Description
VADC.G0BFLOUT1	O	VADC.G0REQGTK CCU41.IN2L CCU80.IN1I	Boundary flag 1 output of group 0
VADC.G1BFLOUT1	O	POSIF0.IN1C POSIF1.IN1C	Boundary flag 1 output of group 1
VADC.G2BFLOUT1	O	-	Boundary flag 1 output of group 2
VADC.G3BFLOUT1	O	-	Boundary flag 1 output of group 3
VADC.GxBFL1	O	-	Boundary flag 1 level of group x
VADC.GxBFSEL1	I	0	Boundary flag 1 (group x) source select
VADC.GxBFDAT1	I	0	Boundary flag 1 (group x) alternate data
VADC.G0BFLOUT2	O	VADC.G3REQGTK CCU41.IN3L CCU80.IN2I	Boundary flag 2 output of group 0
VADC.G1BFLOUT2	O	POSIF0.EWHEA POSIF1.EWHEA	Boundary flag 2 output of group 1
VADC.G2BFLOUT2	O	-	Boundary flag 2 output of group 2
VADC.G3BFLOUT2	O	-	Boundary flag 2 output of group 3
VADC.GxBFL2	O	-	Boundary flag 2 level of group x
VADC.GxBFSEL2	I	0	Boundary flag 2 (group x) source select
VADC.GxBFDAT2	I	0	Boundary flag 2 (group x) alternate data
VADC.G0BFLOUT3	O	VADC.G2REQGTK CCU80.IN3I ERU1.0B2 ERU1.2B2	Boundary flag 3 output of group 0
VADC.G1BFLOUT3	O	ERU1.1B2 ERU1.3B2	Boundary flag 3 output of group 1
VADC.G2BFLOUT3	O	-	Boundary flag 3 output of group 2
VADC.G3BFLOUT3	O	-	Boundary flag 3 output of group 3
VADC.GxBFL3	O	-	Boundary flag 3 level of group x

Table 19-13 Digital Connections in the XMC4500 (cont'd)

Signal	Dir.	Source/Destin.	Description
VADC.GxBFSEL3	I	0	Boundary flag 3 (group x) source select
VADC.GxBFDAT3	I	0	Boundary flag 3 (group x) alternate data

1) Internal signal connection.

20 Delta-Sigma Demodulator (DSD)

The Delta-Sigma Demodulator module (DSD) of the XMC4500 provides a series of digital input channels accepting data streams from external modulators using the Delta/Sigma (DS) conversion principle.

The on-chip demodulator channels convert these inputs to discrete digital values.

The number of inputs and DSD channels depends on the chosen product type (please refer to **"Interconnects" on Page 20-39**).

Table 20-1 Abbreviations used in the DSD Chapter

ADC	Analog to Digital Converter
DMA	Direct Memory Access (controller)
DNL	Differential Non-Linearity (error)
DS	Delta-Sigma (conversion principle)
INL	Integral Non-Linearity (error)
LSB_n	Least Significant Bit: finest granularity of the analog value in digital format, represented by one least significant bit of the conversion result with n bits resolution (measurement range divided in 2^n equally distributed steps)
OSR	Oversampling Ratio
PWM	Pulse Width Modulation

20.1 Overview

Each converter channel can operate independent of the others, controlled by a dedicated set of registers. The results of each channel can be stored in a dedicated channel-specific result register.

The on-chip filter stages generate digital results from the selected modulator signal.

The DSD accepts data from different types of external modulators. Their data streams can be fed to selectable input pins.

Features

The following features describe the functionality of a Delta-Sigma Converter:

- Options to connect external standard Delta-Sigma modulators
  - Selectable data stream inputs
  - Selectable DS clock input or output
- Main demodulator (concatenated hardware filter stages)

  - Configurable comb filter with decimation rates of 4…256
- Automatic offset compensation
- Parallel auxiliary demodulator for limit checking
  - Configurable comb filter with decimation rates of 4…32
  - Two-level boundary comparator
- Carrier signal generator for resolver applications

Table 20-2 DSD Applications

Use Case DSD	Application
Galvanically decoupled phase current measurement	Motor control, Power conversion
Resolver signal evaluation and generation of excitation signal	Motor control
Adjustable resolution for input signals with wide dynamic range	Motorcontrol, Metering, Medical

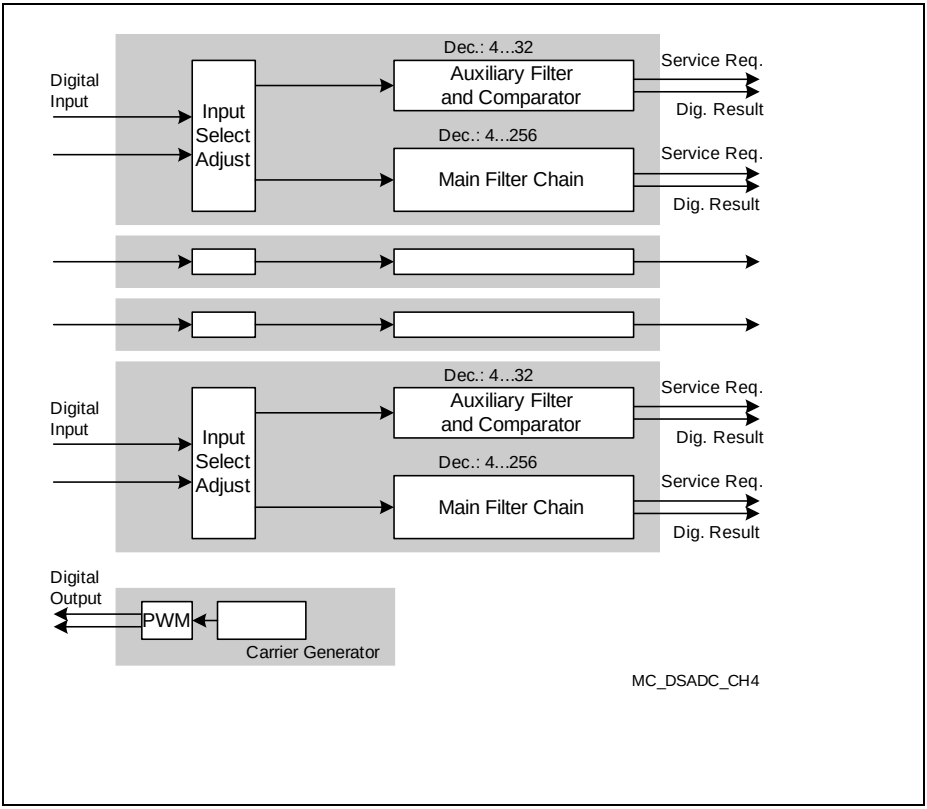


Figure 20-1 DSD Module Overview

20.2 Introduction and Basic Structure

The Delta-Sigma Analog to Digital Converter module of the XMC4500 provides several channels with a main and an auxiliary demodulator including configurable filters for decimation (see **Figure 20-2**).

Several types of external modulators can be connected to the input path. The modulator clock signal can be generated internally or can be fed from an external clock source.

The main chain of digital filters builds the demodulator which produces result values at a configurable output rate. The elements of the filter chain can be selected according to the requirements of the application. The filter chain configuration determines the attenuation and delay properties of the filter. The decimation at a configurable rate reduces the input sampling rate to a lower result data rate.

The configurable comb filter provides the basic filtering and decimation with a selectable decimation rate.

The integrator accumulates a configurable amount of result values. The number of samples is programmable or can be controlled by a hardware signal. This function supports resolver applications to get the baseband signal for the motor position calculation. Furthermore, the integrator can be also used in shunt current measurement applications.

A smaller but quicker parallel auxiliary filter with a comparator supports limit checking, e.g. for overcurrent detection. Two limit values can be defined to restrict the generation of service requests to result values within a configurable area. This saves CPU performance and/or DMA bandwidth.

A carrier signal can be generated to support resolver applications.

The on-chip carrier signal generator produces a selectable output signal (sine, triangle, rectangle) which can be used to drive a resolver. Synchronization of each input signal to the carrier signal ensures correct integration of the resolver input signals.

Service requests can be generated to trigger DMA transfers or to request CPU service.

The basic module clock f_{DSD} is connected to the system clock signal f_{PB}.

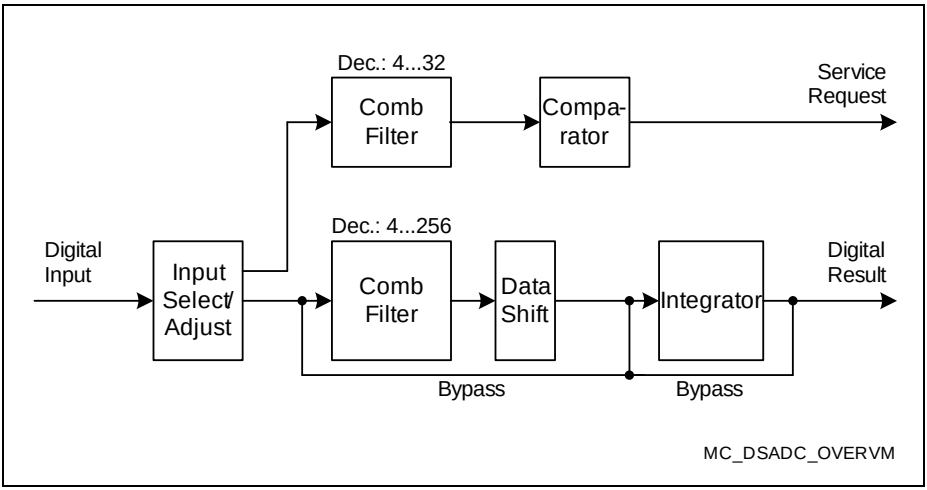


Figure 20-2 DSD Structure Overview

20.3 Configuration of General Functions

Several parameters can be configured to adapt the functionality of the DSD to the requirements of the actual application.

The DSD of the XMC4500 can operate in several configurations:

- Using an external modulator, running on a clock generated on-chip
- Using an external modulator, running on its own clock

The global configuration register **GLOBCFG** selects the source for the internal clock signal which can be used to drive the modulators (alternatively, a clock signal can be input from an external modulator).

The global run control register **GLOBRC** controls the general operation of the available channels.

20.4 Input Channel Configuration

The input data for a channel can be obtained from different sources:

- **External Modulator Without Clock Source**: This type of modulator requires a modulator clock signal. This clock signal is provided by the XMC4500, the data stream produced by the modulator is input as a digital signal.
- **External Modulator With Clock Source**: This type of modulator generates the modulator clock signal along with the data stream. In this case, both the modulator clock and the data stream produced by the modulator are input as digital signals.

Note: An external modulator can be used, in particular, in systems where high voltages are to be sampled. This allows for galvanic decoupling.
Several input pins can be selected.

The modulator clock can be generated in different ways:

- The modulator clock can be derived on-chip from the module clock and is output via a modulator clock pin to be used by an external modulator.
- The external modulator can generate the clock signal which is then input via one of the modulator clock pins.
- The used modulator clock also drives the on-chip carrier generator. This enables synchronous operation of carrier generator and integrator.

A trigger signal can be input from a selectable pin. This trigger signal can be used for different purposes:

- Integration trigger:
  The external signal defines the integration window, i.e. the timespan during which result values are integrated.
- Timestamp trigger:
  The external signal requests the actualization of the timestamp register.
- Input multiplexer trigger:
  The external signal requests the switching of the analog input multiplexer to the next lower input or to the defined start value, respectively.
- Service request gate:
  Service requests for the main filter chain can be restricted to the high or low times of the selected trigger signal.

The figure below summarizes these three signal paths and indicates the source of the corresponding control information.

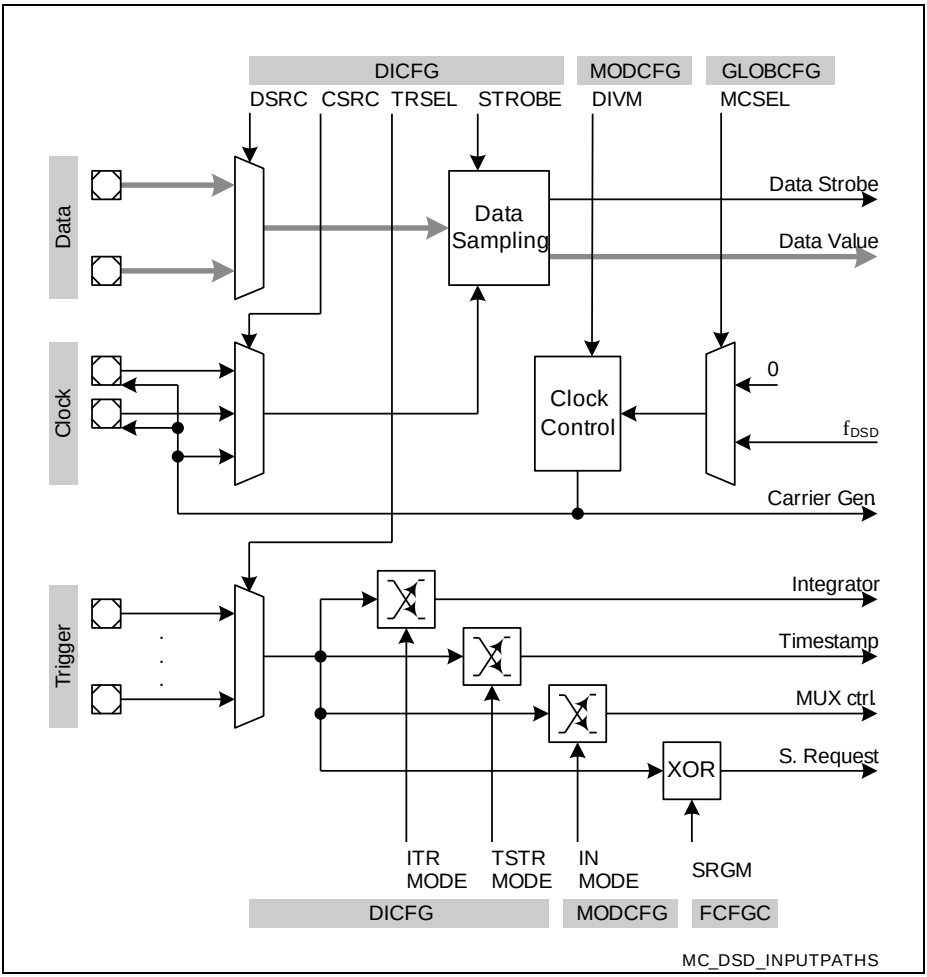


Figure 20-3 Input Path Summary

20.4.1 Modulator Clock Selection and Generation

The modulator clock signal can be generated on-chip by a programmable prescaler (clock output) or can be generated by an external modulator and be input through a pin (clock input). For internal clock generation, the on-chip clock signal is enabled by bitfield MCSEL.

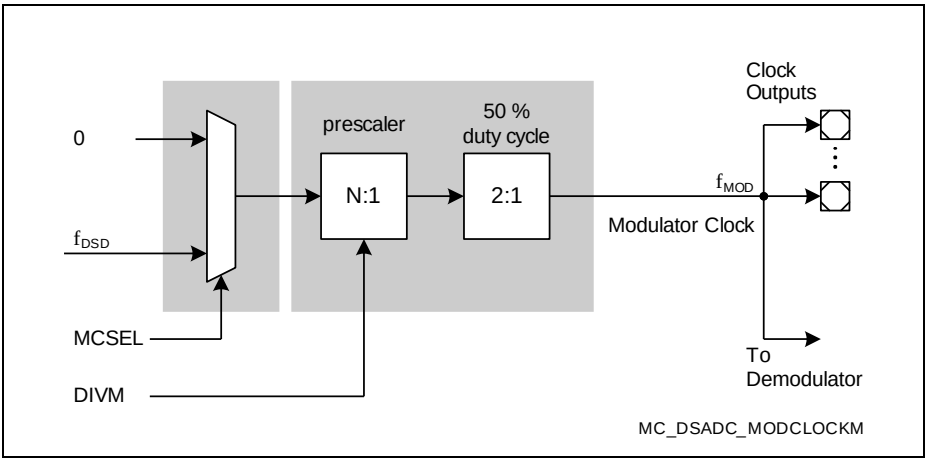


Figure 20-4 Modulator Clock Configuration

The selected clock source is synchronized to the module clock. A configurable edge detector selects the clock signal edges that generate the data strobes which read the next input data and trigger the demodulator (see **Figure 20-5**).

Note: To ensure proper synchronization, an external clock signal must have high/low phases of at least one period of f_{DSD} to be safely synchronized ($f_{IN} < f_{DSD}/2$).

When the clock signal is generated on-chip (see **Figure 20-4**) and is selected as an output signal at a connected pin (see connection list in **Section 20.12**), this clock signal can drive an external modulator.

Bitfield CSRC in register **DICFGx (x = 0 - 3)** selects the clock source for each channel. Bitfield STROBE selects the clocking mode, i.e. the clock edges that put a new input data sample into the filter chain.

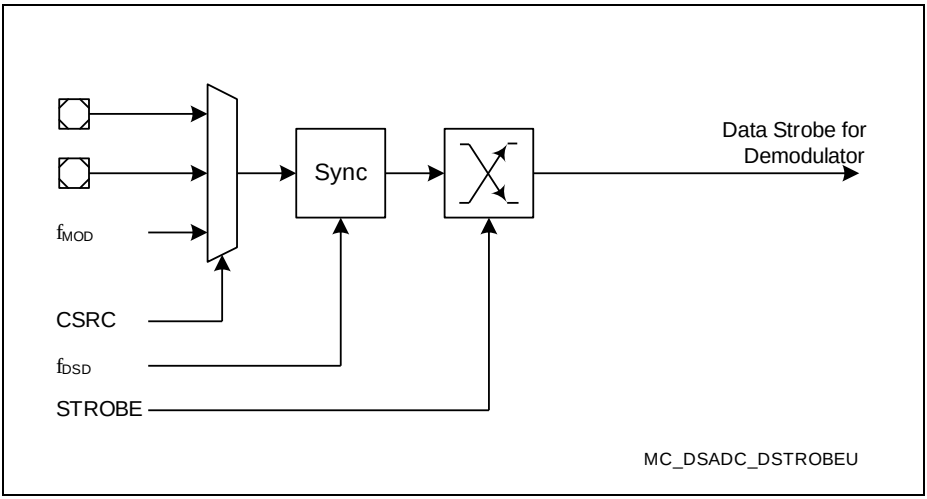


Figure 20-5 Demodulator Data Strobe Selection

20.4.2 Input Data Selection

The data stream of an external modulator can be input from a selectable pin. This signal can optionally be inverted.

The selected input datastream is converted to the comb filter's input format.

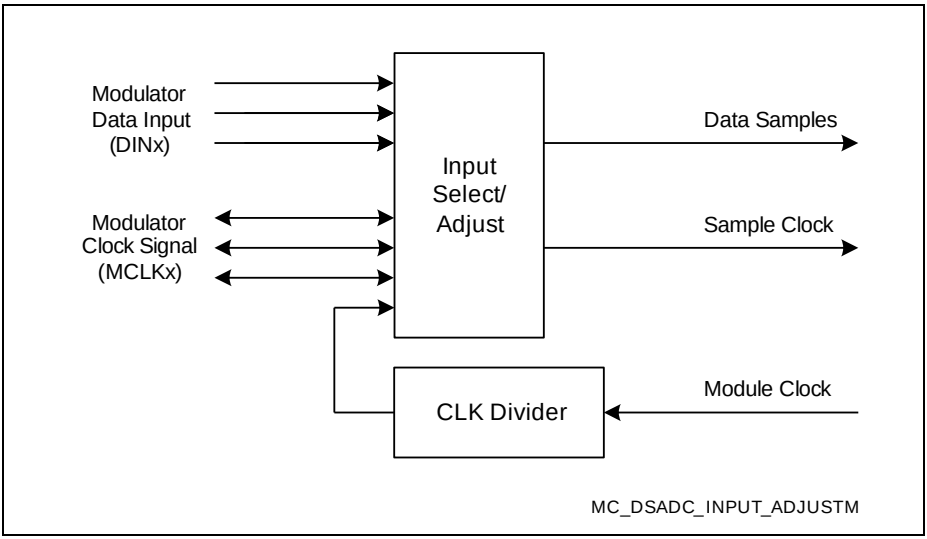


Figure 20-6 Input Control Unit

All options are configured by the demodulator input configuration register **DICFGx (x = 0 - 3)**.

20.4.3 External Modulator

The input data stream can be obtained from an external modulator via a selectable pin. This modulator may, for example, be connected to a high driving voltage and be galvanically decoupled.

In this case, the modulator's data output is connected to the selected pin. The Input Select/Adjust block will then format the selected input data stream to the format required by the subsequent filter.

The source of the 1-bit input data can be selected as well as the source of the sample clock signal. The data strobe that enters a new value into the filter chain can be generated upon configurable clock edges of the modulator clock to support different types of modulators.

20.4.4 Input Path Control

The input for the DSD is fed through several stages before being evaluated and filtered. Registers MODCFGx and DICFGx select the available options for these signal stages.

The following features can be configured:

- Signal input pin selection

- Generation method of input data
- Modulator clock source and/or frequency
- Trigger input pin selection

With this flexibility the DSD can be adjusted to many available types of modulators.

20.5 Main Filter Chain

The result data words are generated by feeding the input data stream through a chain of filter elements and decimating it by a selectable ratio.

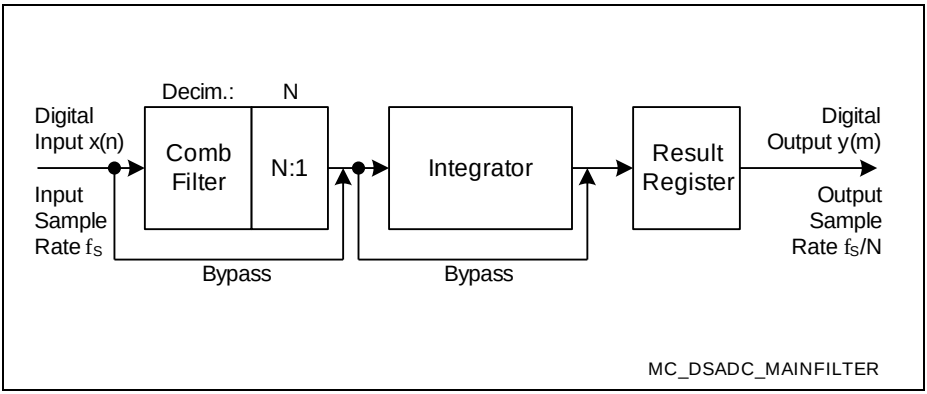


Figure 20-7 Structure of the Main Filter Chain

The elements of the filter can be bypassed, i.e. the filter chain is configurable and its behavior can be adapted to the requirements of the actual application. This comprises the frequency attenuation as well as the total decimation rate.

Note: Only configure or reconfigure filter parameters while the channel is inactive. After the configuration, start the channel by setting the corresponding bit CHxRUN.

20.5.1 Comb Filter

The comb filter (a.k.a. SINC filter) is a simple but very efficient low-pass filter. Up to three comb filter stages can be cascaded to improve the frequency characteristics. The number of active filter stages can be selected (Comb1, Comb2, Comb3) to find the optimum tradeoff between delay and frequency characteristics.

In addition, a combined mode can be selected (CombF) which represents a compromise between a 2-stage and a 3-stage filter.

To synchronize filters of different channels with fine granularity, the decimation counter starts from an arbitrary start value. This start value can be different from the decimation factor and is loaded only once when the counter is started.

The decimation counter is also restarted (i.e. loaded with the start value) when the selected integration trigger event occurs (see **Section 20.5.2**).

The decimation factor can be selected in a wide range from 4 to 256.

Table 20-3 Data Shifter Position[1] Dep. on Filter Mode and Decimation

Decimation	Comb1	Comb2	Comb3	CombF
4 - 32	15:0	15:0	15:0	15:0
33 - 40	15:0	15:0	16:1 *	15:0
41 - 50	15:0	15:0	17:2 *	15:0
51 - 64	15:0	15:0	18:3 *	15:0
65 - 80	15:0	15:0	19:4 *	15:0
81 - 101	15:0	15:0	20:5 *	15:0
102 - 128	15:0	15:0	21:6 *	15:0
129 - 181	15:0	15:0	22:7 *	16:1 *
182 - 203	15:0	16:1 *	23:8 *	17:2
204 - 256	15:0	16:1	24:9 *	17:2

1) * indicates a change. Input data width is 1 bit.

20.5.2 Integrator Stage

The integrator integrates the result values generated during the defined integration window by adding a configurable number of values to build the final result value. The integration window can be started triggered by an internal or external signal.

A configurable number of values can automatically be discarded after the trigger before the integration window is started. This positions the integration window exactly into a timeframe where the filter is stable or where the signal to be measured is free of system-generated noise.

Integration can be used to measure currents through shunt resistors at defined positions in the signal waveform. It also can remove the carrier signal component in resolver applications. In this case, the values to be integrated can be rectified to yield the maximum amplitude of the receiver signal. The delay between the carrier signal (generated by the on-chip carrier generator) and the received position signals can be compensated automatically. Please refer to **Section 20.9**.

Upon the selected integration trigger (INTEN becomes 1) the integration counter starts counting. After NVALDIS values the integrator is started and the counter is reset (if NVALDIS is zero the integration starts immediately). After NVALINT values the integration result is stored in the result register and the integrator and the counter are

cleared. As selected by bit IWS the integration window is either restarted or the integration is stopped (INTEN = 0).

Also, the decimation counter of the comb filter is restarted, i.e. loaded with its start value (see **Section 20.5.1**).

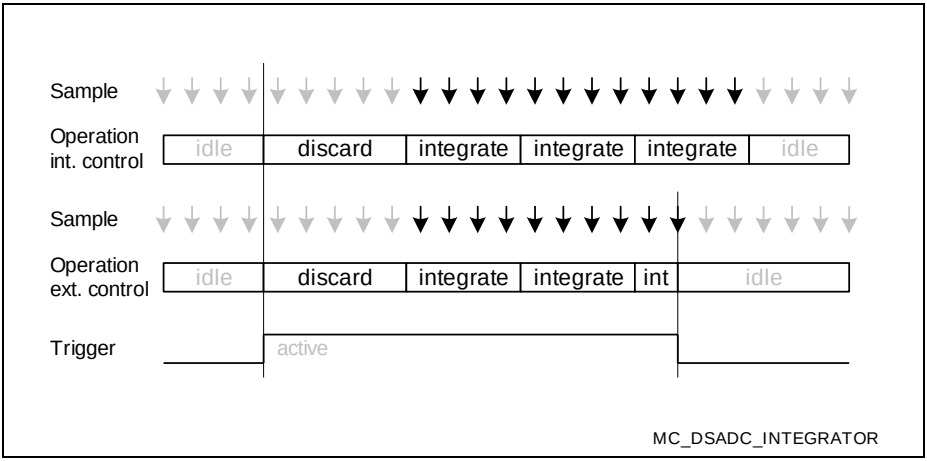


Figure 20-8 Integrator Operation

The external integration trigger signal is selected by bitfield TRSEL in register **DICFGx (x = 0 - 3)**. Bit ITRMODE selects the transition of the selected signal to generate a trigger event. This event starts the integrator by setting bit INTEN. The inverse transition of the selected signal clears bit INTEN.

Note: When the integration window is closed by an external signal (INTEN cleared by inverse trigger), the integrator and the counter are cleared also.

20.6 Auxiliary Filter

The parallel auxiliary filter uses a comb filter for decimation, similar to the one used in the main filter chain. The decimation rate is restricted to 32. This also reduces the filter delay, so the auxiliary filter can be used to supervise the input signal and detect abnormal input values earlier than the main filter chain.

The subsequent comparator provides automatic limit checking by comparing each result to two configurable reference values. The comparator can generate a separate service request.

The two values are defined in the boundary select register and determine the valid result value band if limit checking is enabled.

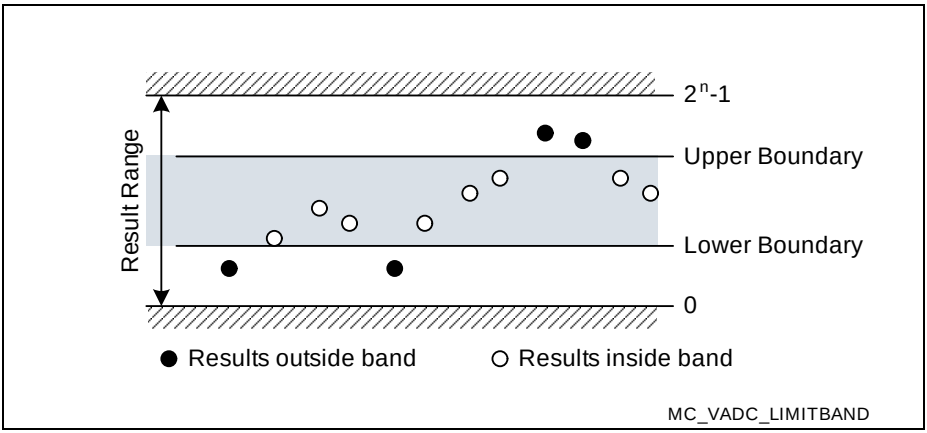


Figure 20-9 Result Monitoring through Limit Checking

A result value is considered inside the defined band when both of the following conditions are true:

- the value is less than or equal to the selected upper boundary
- the value is greater than or equal to the selected lower boundary

The result range can also be divided into two areas:
To select the lower part as valid band, set the lower boundary to the minimum value (000_H) and set the upper boundary to the highest intended value.
To select the upper part as valid band, set the upper boundary to the maximum value (FFF_H) and set the lower boundary to the lowest intended value.

The auxiliary filter can generate two types of output:

- Service requests, optionally restricted by the comparators
- Range signals, indicating when the results are above the upper limit (SAULx) or below the lower limit (SBBLx)

An alarm event can be generated when a new conversion result becomes available. Alarm events can be restricted to result values that are inside or outside a user-defined band (see **Figure 20-9**). This feature supports automatic range monitoring and minimizes the CPU load by issuing service requests only under certain conditions. For example, an input value can be monitored and an alarm indicates a certain threshold.

Note: Limit checking uses the parallel auxiliary filter at a low decimation rate and, therefore, the alarm is generated earlier than the threshold values are seen at the output of the regular filter chain.

Alarm events can also be suppressed completely (see **FCFGAx (x = 0 - 3)**).

The range signals are generated independent of service requests.

Signal SAUL is active while the results are above the upper limit, signal SBLL is active while the results are below the lower limit.

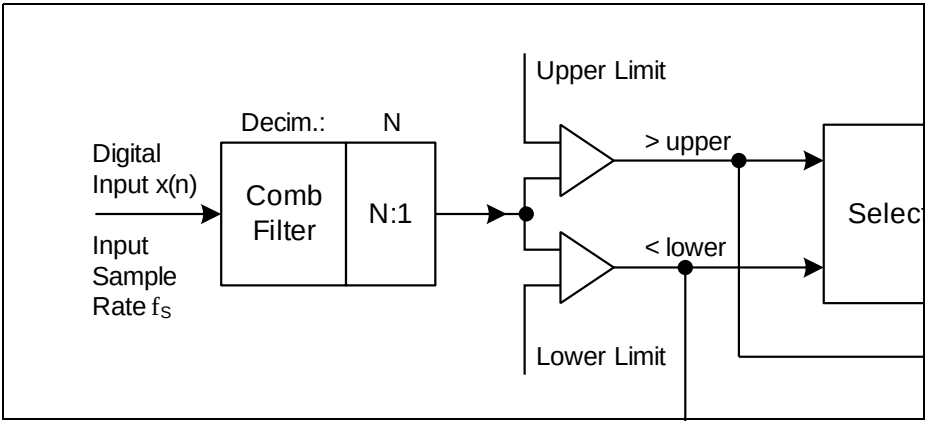


Figure 20-10 Comparator Structure

20.7 Conversion Result Handling

The DSD preprocesses the conversion result data before storing them for retrieval by the CPU or a DMA channel.

Conversion result handling comprises the following functions:

- Filtering and Post-Processing
- Storage of Conversion Results
- Result Event Generation

A programmable offset is subtracted automatically from each result value before being stored in the result register. This offset value is stored in the corresponding register OFFMx. It may be an arbitrary value defined by the application. Usually, the offset value is measured for each channel separately.

The result values of the auxiliary filter are also available for the application, although in many cases the comparator output signal will be sufficient.

The conversion result values are stored in result register RESMx and RESAx, respectively.

20.8 Service Request Generation

The DSD can activate service request output signals to issue an interrupt, to trigger a DMA channel, or to trigger other on-chip modules.

Service request connections can be found in **Table 20-7 “Digital Connections in the XMC4500” on Page 20-40**.

Several events are assigned to each service request output. Service requests can be generated by two types of events:

- **Result events:** indicate a new valid result in a result register. Usually, this triggers a read action by the CPU (or DMA). Result events are generated at the output rate of the configured filter chain.
- **Alarm events:** indicate that a conversion result value is within a programmable value range. This offloads the CPU/DMA from background tasks, i.e. a service request is only activated if the specified conversion result range is met or exceeded.

Each event is indicated by a dedicated flag that can be cleared by software. If a service request is enabled for a certain event, the service request is generated for each event, independent of the status of the corresponding event indication flag. This ensures efficient DMA handling of DSD events (the event can generate a service request without the need to clear the indication flag).

*Note: The **Service Request Registers** provide a set of bits for each available channel. The number of available channels depends on the chosen device type.*

20.9 Resolver Support

Resolver applications determine the rotation angle by evaluating the signals from two orthogonally placed coils. These coils are excited by the magnetic field of a third coil. The DSD can read the two return signals using two input channels and can also generate the excitation sine signal (carrier). It also provides synchronization logic to compensate the delay between the generated carrier signal and the received position signals. The integrator stage converts the carrier-based return signals to position-based values (carrier cancellation).

20.9.1 Carrier Signal Generation

The carrier signal generator (CG) outputs a PWM signal that induces a sine signal in the excitation coil of the resolver. Alternatively, it can generate PWM patterns that resemble triangle or square signals (see **Figure 20-11**). The polarity of the carrier signal can be selected.

A carrier signal period consists of 32 steps. Each step equals a PWM period of 32 cycles.

Bit-reverse generation mode increases the frequency spectrum to yield a smoother induced sine signal. This is done by distributing the 0 and 1 bits over the 32 cycles of a PWM period.

The generated pattern is actually a cosine signal, i.e. it starts at the maximum output value. This is advantageous if the output pin is pulled high before the carrier signal is generated. In case of a pull-down the inverted output signal should be selected.

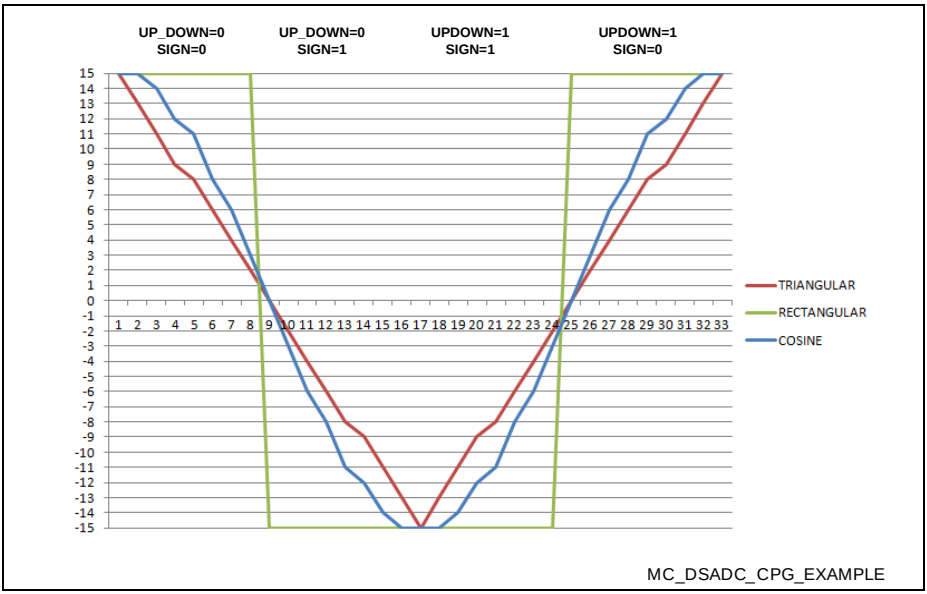


Figure 20-11 Example Pattern/Waveform Outputs

20.9.2 Return Signal Synchronization

In a resolver, the received return signals are induced by the carrier signal and their amplitudes are modulated with the sine and cosine magnitudes corresponding to the current resolver position. These amplitudes are determined by integrating the return signals over a carrier signal period.

To properly integrate their magnitude, the return signals must be rectified. For this purpose the carrier generator provides the sign information of the generated carrier signal (SGNCG in register **CGCFG**).

Alternatively, an external carrier signal generator can be used. If this generator delivers a sign signal, this can be input to a pin and is then used as external carrier sign signal. If no sign signal is available, the carrier signal itself can be converted by the next adjacent input channel and its sign signal is then used as alternate carrier sign signal.

The rectification of the received signals must be delayed to compensate the round trip delay through the system (driver, resolver coils, cables, etc.). For the rectification, the received values are multiplied with the delayed carrier sign signal (SGND in register **RECTCFGx (x = 0 - 3)**). This synchronization is done for each channel separately, to achieve the maximum possible amplitudes for each signal.

The delay is realized with the sign delay counter SDCOUNT. SDCOUNT is cleared and started upon a falling edge of the carrier generator's sign signal (SGNCG), i.e. at the begin of the positive halfwave of the carrier signal. After counting SDPOS results from the filter chain, also the rectification signal (SGND) is cleared, indicating positive values from now on. After counting SDNEG values, the rectification signal is set, indicating negative values (see also **Figure 20-12**).

The compare values SDPOS and SDNEG are stored by the application software. SDPOS is the delay value that accounts for the resolver signal's round trip delay. This delay is constantly measured by capturing the current counter value into bitfield SDCAP when the first positive result (after negative results) is received in the respective channel. Software can read these value and compute a delay value e.g. by averaging a series of measured values to compensate noise. The delay for the negative halfwave (SGND = 0) is determined by adding the duration of a carrier signal halfwave. This value is written to bitfield SDNEG.

A new captured value is indicated by setting the flag SDVAL. This flag is cleared when reading register CGSYNCx.

Capturing a new value can trigger a service request. The service request line of the auxiliary channel is used for this purpose. This alternate request source is selected by bitfield SRGA in register **FCFGAx (x = 0 - 3)**.

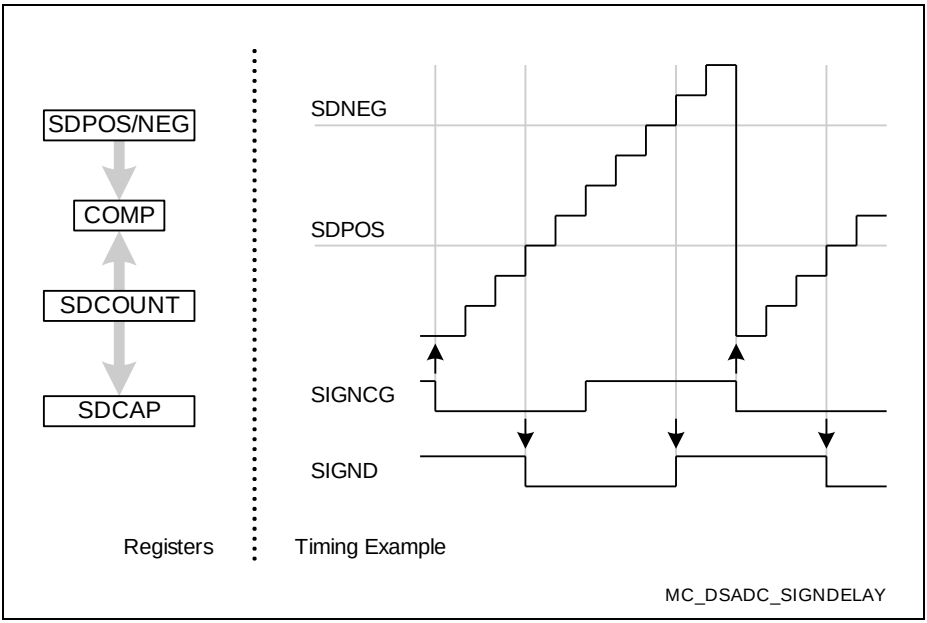


Figure 20-12 Sign Delay Example

20.10 Time-Stamp Support

Some applications need to determine the result value at certain points of time inbetween two regular output values. The interpolation algorithm needs to determine the position of the required point of time in relation to the regular results.

This interpolation is supported by providing a timestamp that marks the delay since the last regular output value. This timestamp is composed of:

- the last regular result value
- the current decimation counter value
- the current integrator counter value

All timestamp information is combined into one register, so the complete timestamp can easily be stored away by a single DMA transfer. The timestamp information is captured into register TSTMPx upon a hardware trigger. For this purpose, the trigger signal is used, which is selected by bitfield TRSEL in register **DICFGx (x = 0 - 3)**. Bitfield TSTRMODE selects the edge(s) that capture timestamp information.

20.11 Registers

The DSD is built from a series of channels that are controlled in an identical way. This makes programming versatile and scalable. The corresponding registers, therefore, have an individual offset assigned (see **Table 20-5**). The exact register location is obtained by adding the respective register offset to the base address (see **Table 20-4**) of the corresponding channel.

Due to the regular structure, several registers appear within each channel. This is indicated in the register overview table by placeholders:

- $0X\#\#_H$ means: $x \times 0100_H + 01\#\#_H$, for x = 0 - 3

Table 20-4 Registers Address Space

Module	Base Address	End Address	Note
DSD	$4000\ 8000_H$	$4000\ BFFF_H$	

Table 20-5 Registers Overview

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Page Num.
			Read	Write	
ID	Module Identification Register	0008_H	U, PV	BE	20-21
CLC	Clock Control Register	0000_H	U, PV	PV	20-21
OCS	OCDS Control and Status Register	0028_H	U, PV	PV	20-22
GLOBCFG	Global Configuration Register	0080_H	U, PV	U, PV	20-23

Table 20-5 Registers Overview (cont'd)

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Page Num.
			Read	Write	
GLOBRC	Global Run Control Register	0088_H	U, PV	U, PV	20-24
MODCFGx (x = 0 - 3)	Modulator Configuration Register x	$0X00_H$	U, PV	U, PV	20-25
DICFGx (x = 0 - 3)	Demodulator Input Configuration Register x	$0X08_H$	U, PV	U, PV	20-26
BOUNDSELx (x = 0 - 3	Global Boundary Select Register	$0X28_H$	U, PV	U, PV	20-32
IWCTRx (x = 0 - 3)	Integration Window Control Register	$0X20_H$	U, PV	U, PV	20-31
FCFGCx (x = 0 - 3)	Filter Configuration Register Main Filter	$0X14_H$	U, PV	U, PV	20-28
FCFGAx (x = 0 - 3)	Filter Configuration Register Auxiliary Filter	$0X18_H$	U, PV	U, PV	20-29
RESMx (x = 0 - 3)	Result Register x Main Filter	$0X30_H$	U, PV	U, PV	20-32
OFFMx (x = 0 - 3)	Offset Register x Main Filter	$0X38_H$	U, PV	U, PV	20-33
RESAx (x = 0 - 3)	Result Register x Auxiliary Filter	$0X40_H$	U, PV	U, PV	20-33
EVFLAG	Event Flag Register	$0XE0_H$	U, PV	U, PV	20-34
EVFLAGCLR	Event Flag Clear Register	$0XE4_H$	U, PV	U, PV	20-34
CGCFG	Carrier Generator Configuration Register	$00A0_H$	U, PV	U, PV	20-35
RECTCFGx (x = 0 - 3)	Rectification Configuration Register	$0XA8_H$	U, PV	U, PV	20-37
CGSYNCx (x = 0 - 3)	Carrier Generator Synchronization Register	$0XA0_H$	U, PV	U, PV	20-38
TSTMPx (x = 0 - 3)	Time Stamp Register	$0X50_H$	U, PV	U, PV	20-39

20.11.1 Module Identification

The module identification register indicates the version of the DSD module that is used in the XMC4500.

ID
Module Identification Register **(0008_H)** **Reset Value: 00A4 C0XX$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MOD_NUMBER																MOD_TYPE								MOD_REV							
r																r								r							

Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision** Indicates the revision number of the implementation. This information depends on the design step.
MOD_TYPE	[15:8]	r	**Module Type** This internal marker is fixed to $C0_H$.
MOD_NUMBER	[31:16]	r	**Module Number** Indicates the module identification number ($00A4_H$ = DSD)

20.11.2 System Registers

A set of standardized registers provides general access to the module and controls basic system functions.

The Clock Control Register **CLC** allows the programmer to adapt the functionality and power consumption of the module to the requirements of the application. Register **CLC** controls the module clock signal and the reactivity to the sleepde signal.

CLC
Clock Control Register **(0000_H)** **Reset Value: 0000 0003$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**EDIS**	**0**	**DISS**	**DISR**
r	r	r	r	r	r	r	r	r	r	r	r	rw	r	r	rw

Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module. 0_B On request: enable the module clock 1_B Off request: stop the module clock
DISS	1	r	**Module Disable Status Bit** 0_B Module clock is enabled 1_B Off: module is not clocked
0	2	r	**Reserved, write 0, read as 0**
EDIS	3	rw	**Sleep Mode Enable Control** Used to control module's reaction to sleep mode. 0_B Sleep mode request is enabled and functional 1_B Module disregards the sleep mode control signal
0	[31:4]	r	**Reserved, write 0, read as 0**

The OCDS control and status register OCS controls the module's behavior in suspend mode (used for debugging).

The OCDS Control and Status (OCS) register is cleared by Debug Reset.
The OCS register can only be written when the OCDS is enabled.
If OCDS is being disabled, the OCS register value will not change.
When OCDS is disabled the OCS suspend control is ineffective.
Write access is 32 bit wide only and requires Supervisor Mode.

OCS
OCDS Control and Status Register (0028_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**SUSSTA**	**SUS_P**	**SUS**				**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	rh	w	rw				r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Field	Bits	Type	Description
0	[23:0]	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
SUS	[27:24]	rw	**OCDS Suspend Control** Controls the sensitivity to the suspend signal coming from the OCDS Trigger Switch (OTGS) 0000_BWill not suspend 0001_BHard suspend: Clock is switched off immediately. 0010_BSoft suspend channel 0 0011_BSoft suspend channel 1 … 0101_BSoft suspend channel 3 Others Reserved *Note: In soft suspend mode, the respective channel is stopped after the next result has been stored.*
SUS_P	28	w	**SUS Write Protection** SUS is only written when SUS_P is 1, otherwise unchanged. Read as 0.
SUSSTA	29	rh	**Suspend State** 0_B Module is not (yet) suspended 1_B Module is suspended
0	[31:30]	r	**Reserved, write 0, read as 0**

20.11.3 General Registers

The global configuration register **GLOBCFG** selects the source for the internal clock signal which can be used to drive the modulators (alternatively, a clock signal can be input from an external modulator).

GLOBCFG
Global Configuration Register **(0080_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**MCSEL**		
r	r	r	r	r	r	r	r	r	r	r	r	r	rw		

Field	Bits	Type	Description
MCSEL	[2:0]	rw	**Modulator Clock Select** Selects the source for the on-chip clock source for the modulator clock. 000_B Internal clock off, no source selected 001_B f_{DSD} All other combinations are reserved.
0	[31:3]	r	**Reserved, write 0, read as 0**

GLOBRC
Global Run Control Register (0088_H) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**CH3 RUN**	**CH2 RUN**	**CH1 RUN**	**CH0 RUN**
r	r	r	r	r	r	r	r	r	r	r	r	rw	rw	rw	rw

Field	Bits	Type	Description
CHxRUN **x = 0 - 3**	x	rw	**Channel x Run Control** Each bit (when set) enables the corresponding demodulator channel. 0_B Stop channel x 1_B Demodulator channel x is enabled and runs When CHxRUN is set, all filter blocks are cleared.
0	[31:4]	r	**Reserved, write 0, read as 0**

20.11.4 Input Path Control

The modulator configuration register selects the Divider factor for modulator clock.

MODCFGx (x = 0 - 3)
Modulator Configuration Register x

(x * 0100_H + 0100_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**DWC**	**0**	**0**	**0**	**DIVM**			
r	r	r	r	r	r	r	r	w	r	r	r	rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Field	Bits	Type	Description
0	[15:0]	r	**Reserved, write 0, read as 0**
DIVM	[19:16]	rw	**Divider Factor for Modulator Clock** Defines the operation frequency for the modulator, derived from the selected internal clock source.[1)] 0_H $f_{MOD} = f_{CLK} / 2$ 1_H $f_{MOD} = f_{CLK} / 4$ 2_H $f_{MOD} = f_{CLK} / 6$ ... F_H $f_{MOD} = f_{CLK} / 32$
0	[22:20]	r	**Reserved, write 0, read as 0**
DWC	23	w	**Write Control for Divider Factor** 0_B No write access to divider factor 1_B Bitfield DIVM can be written
0	[31:24]	r	**Reserved, write 0, read as 0**

1) The limit values for f_{MOD} must not be exceeded when selecting the module input frequency and the prescaler setting.

The demodulator input configuration register selects input signal sources for each channel:

- Source of data stream
- Trigger signal source and mode
- Sample clock source
- Data strobe generation mode

DICFGx (x = 0 - 3)
Demodulator Input Configuration Register x
(x * 0100_H + 0108_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SCWC	0	0	0	0	0	0	0	STROBE				CSRC			
w	r	r	r	r	r	r	r	rw				rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRWC	TRSEL			TSTRMODE		ITRMODE		DSWC	0	0	0	DSRC			
w	rw			rw		rw		w	r	r	r	rw			

Field	Bits	Type	Description
DSRC	[3:0]	rw	**Input Data Source Select** $000X_B$ Disconnected 0010_B External, from input A, direct 0011_B External, from input A, inverted 0100_B External, from input B, direct 0101_B External, from input B, inverted Other combinations are reserved. *Note: Pin association is described in Section 20.12.2*
0	[6:4]	r	**Reserved, write 0, read as 0**
DSWC	7	w	**Write Control for Data Selection** 0_B No write access to data parameters 1_B Bitfield DSRC can be written
ITRMODE	[9:8]	rw	**Integrator Trigger Mode**[1)] 00_B No integration trigger, integrator bypassed 01_B Trigger event upon a falling edge 10_B Trigger event upon a rising edge 11_B No trigger, integrator active all the time *Note: To ensure proper operation, ensure that bitfield ITRMODE is 00_B before selecting any other trigger mode.*

Field	Bits	Type	Description
TSTRMODE	[11:10]	rw	**Timestamp Trigger Mode**[2)] 00_B No timestamp trigger 01_B Trigger event upon a falling edge 10_B Trigger event upon a rising edge 11_B Trigger event upon each edge
TRSEL	[14:12]	rw	**Trigger Select** Selects an input for the trigger signal (for integrator, timestamp, multiplexer control, service request gating). The connected trigger input signals are listed in **Section 20.12.2**
TRWC	15	w	**Write Control for Trigger Parameters** 0_B No write access to trigger parameters 1_B Bitfields TRSEL, TSTRMODE, ITRMODE can be written
CSRC	[19:16]	rw	**Sample Clock Source Select** 0000_BReserved 0001_BExternal, from input A 0010_BExternal, from input B 1111_BInternal clock Other combinations are reserved. *Note: Pin association is described in **Section 20.12.2***
STROBE	[23:20]	rw	**Data Strobe Generatoion Mode** 0000_BNo data strobe 0001_BDirect clock, a sample trigger is generated at each rising clock edge 0010_BDirect clock, a sample trigger is generated at each falling clock edge 0011_BDouble data, a sample trigger is generated at each rising and falling clock edge 0100_BReserved 0101_BDouble clock, a sample trigger is generated at every 2nd rising clock edge 0110_BDouble clock, a sample trigger is generated at every 2nd falling clock edge 0111_BReserved Other combinations are reserved.
0	[30:24]	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
SCWC	31	w	**Write Control for Strobe/Clock Selection** 0_B No write access to strobe/clock parameters 1_B Bitfields STROBE, CSRC can be written

1) The integration trigger mode controls bit INTEN in register **IWCTRx (x = 0 - 3)** and hence the operation of the integrator:
Bit INTEN is set when ITRMODE = 11_B or when the selected trigger signal transition occurs.
Bit INTEN is cleared when ITRMODE = 00_B or when the inverse trigger signal transition occurs.

2) The timestamp trigger mode controls capturing the timestamp information to register **TSTMPx (x = 0 - 3)**.

20.11.5 Filter Configuration

FCFGCx (x = 0 - 3)
Filter Configuration Register x, Main Comb Filter
(x * 0100_H + 0114_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CFMDCNT								CFMSV							
rh								rw							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SRGM		0	0	0	CF EN	CFMC		CFMDF							
rw		r	r	r	rw	rw		rw							

Field	Bits	Type	Description
CFMDF	[7:0]	rw	**Comb Filter (Main Chain) Decimation Factor** The decimation factor of the Main comb filter is CFMDF + 1. Valid values are 03_H to FF_H (4 to 256).
CFMC	[9:8]	rw	**Comb Filter (Main Chain) Configuration** 00_B Comb1 01_B Comb2 10_B Comb3 11_B CombF
CFEN	10	rw	**Comb Filter Enable** 0_B Comb filter disabled and bypassed 1_B Enable comb filter

Field	Bits	Type	Description
0	[13:11]	r	**Reserved, write 0, read as 0**
SRGM	[15:14]	rw	**Service Request Generation Main Chain** 00_B Never, service requests disabled 01_B While gate (selected trigger signal) is high 10_B While gate (selected trigger signal) is low 11_B Always, for each new result value
CFMSV	[23:16]	rw	**Comb Filter (Main Chain) Start Value** The decimation counter begins counting at value CFMSV, when started or restarted. Valid values are 03_H to CFMDF (4 to selected decimation factor).
CFMDCNT	[31:24]	rh	**Comb Filter (Main Chain) Decimation Counter** The decimation counter counts the filter cycles until an output is generated, i.e. the oversampling rate.

FCFGAx (x = 0 - 3)
Filter Configuration Register x, Auxiliary Filter
(x * 0100_H + 0118_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CFADCNT								0	0	0	0	0	0	0	0
rh								r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	EGT	ESEL		SRGA		CFAC		CFADF							
r	rw	rw		rw		rw		rw							

Field	Bits	Type	Description
CFADF	[7:0]	rw	**Comb Filter (Auxiliary) Decimation Factor** The decimation factor of the Auxiliary comb filter is CFADF +1. Valid values are 03_H to $3F_H$ (4 to 64). [1)]

Field	Bits	Type	Description
CFAC	[9:8]	rw	**Comb Filter (Auxiliary) Configuration** 00_B Comb1 01_B Comb2 10_B Comb3 11_B CombF
SRGA	[11:10]	rw	**Service Request Generation Auxiliary Filter** 00_B Never, service requests disabled 01_B Auxiliary filter: As selected by bitfield ESEL 10_B Alternate source: Capturing of a sign delay value to register **CGSYNCx (x = 0 - 3)** 11_B Reserved
ESEL	[13:12]	rw	**Event Select** Defines when an event for the auxiliary filter is generated. 00_B Always, for each new result value 01_B If result is inside the boundary band 10_B If result is outside the boundary band 11_B Reserved
EGT	14	rw	**Event Gating** Defines if events for the auxiliary filter are coupled to the integration window. 0_B Separate: generate events according to ESEL 1_B Coupled: generate events only when the integrator is enabled and after the discard phase defined by bitfield NVALDIS[2)]
0	[23:15]	r	**Reserved, write 0, read as 0**
CFADCNT	[31:24]	rh	**Comb Filter (Auxiliary) Decimation Counter** The decimation counter counts the filter cycles until an output is generated, i.e. the oversampling rate.

1) Most probably the maximum value for the OSR will be limited to 32.

2) While the integrator is bypassed, it does not influence the auxiliary channel. The event gating suppresses service requests, result values are still stored in register RESAx.

IWCTRx (x = 0 - 3)
Integration Window Control Register x

(x * 0100_H + 0120_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**NVALINT**						**IWS**	**0**	**NVALDIS**					
r	r	rw						rw	r	rw					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REPVAL				**REPCNT**				**INT EN**	**0**	**NVALCNT**					
rw				rh				rh	r	rh					

Field	Bits	Type	Description
NVALCNT	[5:0]	rh	**Number of Values Counted** Counts the number of values until integration is started (NVALDIS) or completed (NVALINT)
0	6	r	**Reserved, write 0, read as 0**
INTEN	7	rh	**Integration Enable[1)]** 0_B Integration stopped. INTEN is cleared at the end of the integration window, i.e. upon the inverse trigger event transition of the external trigger signal. 1_B Integration enabled. INTEN is set upon the defined trigger event.
REPCNT	[11:8]	rh	**Integration Cycle Counter** Counts the number of integration cycles if activated (IWS = 0). This number is selected via bitfield REPVAL.
REPVAL	[15:12]	rw	**Number of Integration Cycles** Defines the number of integration cycles to be counted by REPCNT if activated (IWS = 0). The number of cycles is REPVAL+1.
NVALDIS	[21:16]	rw	**Number of Values Discarded** Start the integration cycle after NVALDIS values
0	22	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
IWS	23	rw	**Integration Window SIze** 0_B Internal control: stop integrator after REPVAL+1 integration cycles 1_B External control: stop integrator when bit INTEN becomes 0
NVALINT	[29:24]	rw	**Number of Values Integrated** Stop the integration cycle after NVALINT+1 values
0	[31:30]	r	**Reserved, write 0, read as 0**

1) For the control of bit INTEN, see also bitfield ITRMODE in register **DICFGx (x = 0 - 3)**.

BOUNDSELx (x = 0 - 3)
Boundary Select Register x **(x * 0100_H + 0128_H)** **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BOUNDARYU																BOUNDARYL															
rw																rw															

Field	Bits	Type	Description
BOUNDARYL	[15:0]	rw	**Lower Boundary Value for Limit Checking** This (two's complement) value is compared to the results of the parallel filter.
BOUNDARYU	[31:16]	rw	**Upper Boundary Value for Limit Checking** This (two's complement) value is compared to the results of the parallel filter.

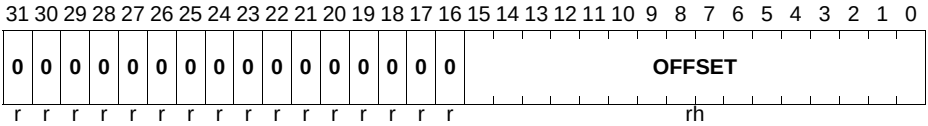
20.11.6 Conversion Result Handling

RESMx (x = 0 - 3)
Result Register x Main Filter **(x * 0100_H + 0130_H)** **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RESULT															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	rh															

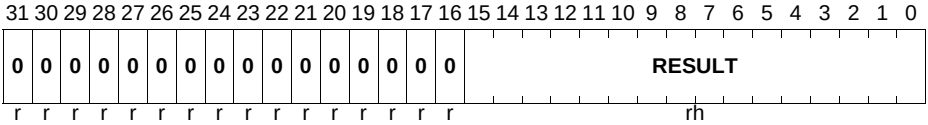
Field	Bits	Type	Description
RESULT	[15:0]	rh	**Result of most recent conversion**
0	[31:16]	r	**Reserved, write 0, read as 0**

OFFMx (x = 0 - 3)
Offset Register x Main Filter **(x * 0100_H + 0138_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
OFFSET	[15:0]	rw	**Offset Value** This signed value is subtracted from each result before being written to the corresponding result register RESMx.
0	[31:16]	r	**Reserved, write 0, read as 0**

RESAx (x = 0 - 3)
Result Register x Auxiliary Filter
(x * 0100_H + 0140_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
RESULT	[15:0]	rh	**Result of most recent conversion**
0	[31:16]	r	**Reserved, write 0, read as 0**

20.11.7 Service Request Registers

EVFLAG
Event Flag Register **(00E0$_H$)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**ALEV3**	**ALEV2**	**ALEV1**	**ALEV0**
r	r	r	r	r	r	r	r	r	r	r	r	rwh	rwh	rwh	rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**RESEV3**	**RESEV2**	**RESEV1**	**RESEV0**
r	r	r	r	r	r	r	r	r	r	r	r	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
RESEVx (x=0-3)	x	rwh	**Result Event** 0_B No result event 1_B A new result has been stored in register RESMx
0	[15:4]	r	**Reserved, write 0, read as 0**
ALEVx (x=0-9)	x+16	rwh	**Alarm Event** 0_B No alarm event 1_B An alarm event has occurred
0	[31:20]	r	**Reserved, write 0, read as 0**

EVFLAGCLR
Event Flag Clear Register **(00E4$_H$)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**ALEC3**	**ALEC2**	**ALEC1**	**ALEC0**
r	r	r	r	r	r	r	r	r	r	r	r	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**RESEC3**	**RESEC2**	**RESEC1**	**RESEC0**
r	r	r	r	r	r	r	r	r	r	r	r	w	w	w	w

Field	Bits	Type	Description
RESECx (x=0-3)	x	w	**Result Event Clear** 0_B No action 1_B Clear bit RESEVx
0	[15:4]	r	**Reserved, write 0, read as 0**
ALECx (x=0-3)	x+16	w	**Alarm Event Clear** 0_B No action 1_B Clear bit ALEVx
0	[31:20]	r	**Reserved, write 0, read as 0**

Note: Software can set flags RESEVx and ALEVx and trigger the corresponding event by writing 1 to the respective bit. Writing 0 has no effect.
Software can clear these flags by writing 1 to bit RESECx and ALECx, respectively.

20.11.8 Miscellaneous Registers

CGCFG
Carrier Generator Configuration Register

(00A0$_H$) **Reset Value: 0710 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**SGNCG**	**STEPD**	**STEPS**	**STEPCOUNT**				**0**	**0**	**0**	**BITCOUNT**				
r	rh	rh	rh	rh				r	r	r	rh				

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RUN	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**DIVCG**				**SIGPOL**	**BREV**	**CGMOD**	
rh	r	r	r	r	r	r	r	rw				rw	rw	rw	

Field	Bits	Type	Description
CGMOD	[1:0]	rw	**Carrier Generator Operating Mode** 00_B Stopped 01_B Square wave 10_B Triangle 11_B Sine wave Stopping the carrier generator (CGMOD = 00_B) terminates the PWM output after completion of the current period (indicated by bit RUN = 0).
BREV	2	rw	**Bit-Reverse PWM Generation** 0_B Normal mode 1_B Bit-reverse mode
SIGPOL	3	rw	**Signal Polarity** 0_B Normal: carrier signal begins with +1 1_B Inverted: carrier signal begins with -1
DIVCG	[7:4]	rw	**Divider Factor for the PWM Pattern Signal Generator** Defines the the carrier signal frequency from the selected internal clock source. 0_H $f_{CG} = f_{CLK} / 2$ 1_H $f_{CG} = f_{CLK} / 4$ 2_H $f_{CG} = f_{CLK} / 6$ ... F_H $f_{CG} = f_{CLK} / 32$
0	[14:8]	r	**Reserved, write 0, read as 0**
RUN	15	rh	**Run Indicator** 0_B Stopped (cleared at the end of a period) 1_B Running
BITCOUNT	[20:16]	rh	**Bit Counter** Counts the 32 cycles generated for each step
0	[23:21]	r	**Reserved, write 0, read as 0**
STEPCOUNT	[27:24]	rh	**Step Counter** Counts the ±16 steps generated for each carrier signal period
STEPS	28	rh	**Step Counter Sign** Indicates the sign of the step counter value 0_B Step counter value is positive 1_B Step counter value is negative

Field	Bits	Type	Description
STEPD	29	rh	**Step Counter Direction** 0_B Step counter is counting up 1_B Step counter is counting down
SGNCG	30	rh	**Sign Signal from Carrier Generator** 0_B Positive values 1_B Negative values
0	31	r	**Reserved, write 0, read as 0**

RECTCFGx (x = 0 - 3)
Rectification Configuration Register x

(x * 0100_H + $01A8_H$) **Reset Value: $8000\ 0000_H$**

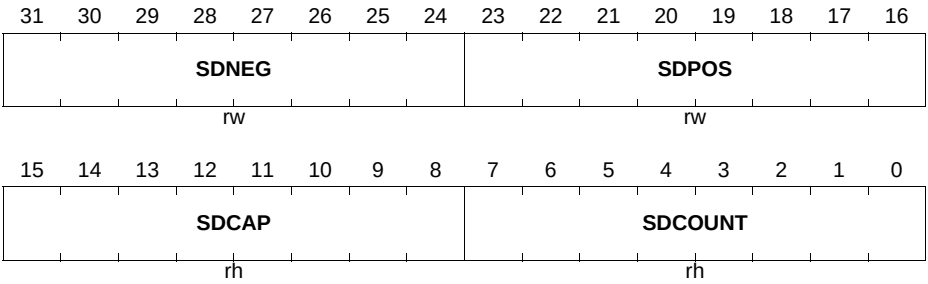
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SGND	**SGNCS**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
rh	rh	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SDCVAL	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**SSRC**		**0**	**0**	**0**	**RFEN**
rh	r	r	r	r	r	r	r	r	r	rw		r	r	r	rw

Field	Bits	Type	Description
RFEN	0	rw	**Rectification Enable** General control of the rectifier circuit. 0_B No rectification, data not altered 1_B Data are rectified according to SGND
0	[3:1]	r	**Reserved, write 0, read as 0**
SSRC	[5:4]	rw	**Sign Source** Selects the sign signal that is to be delayed. 00_B On-chip carrier generator 01_B Sign of result of next channel 10_B External sign signal A 11_B External sign signal B
0	[14:6]	r	**Reserved, write 0, read as 0**

Field	Bits	Type	Description
SDVAL	15	rh	**Valid Flag** Indicates a new value in bitfield SDCAP. 0_B No new result available 1_B Bitfield SDCAP has been updated with a new captured value and has not yet been read
0	[29:16]	r	**Reserved, write 0, read as 0**
SGNCS	30	rh	**Selected Carrier Sign Signal** 0_B Positive values 1_B Negative values
SGND	31	rh	**Sign Signal Delayed** 0_B Positive values 1_B Negative values

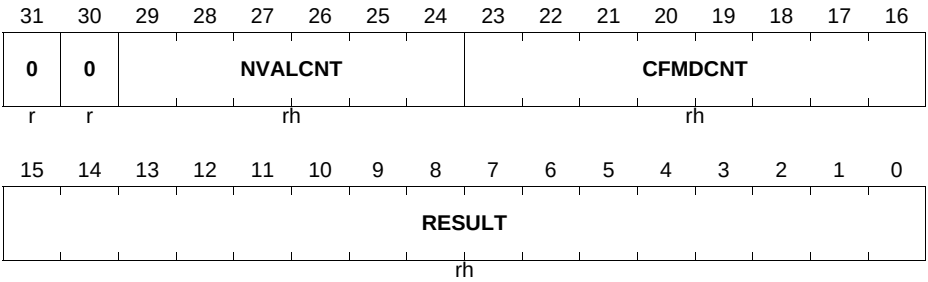
CGSYNCx (x = 0 - 3)
Carrier Generator Synchronization Register x
(x * 0100_H + $01A0_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
SDCOUNT	[7:0]	rh	**Sign Delay Counter** Counts the values to delay the carrier sign signal
SDCAP	[15:8]	rh	**Sign Delay Capture Value** Indicates the values counted between the begin of the positive halfwave of the carrier signal and the first received positive value.
SDPOS	[23:16]	rw	**Sign Delay Value for Positive Halfwave** Defines the content of SDCOUNT to generate a negative delayed sign signal (SGND).

Field	Bits	Type	Description
SDNEG	[31:24]	rw	**Sign Delay Value for Negative Halfwave** Defines the content of SDCOUNT to generate a positive delayed sign signal (SGND).

TSTMPx (x = 0 - 3)
Time-Stamp Register x **($x * 0100_H + 0150_H$)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
RESULT	[15:0]	rh	**Result of most recent conversion** This value is copied from register **RESMx (x = 0 - 3)**.
CFMDCNT	[23:16]	rh	**Comb Filter (Main Chain) Decimation Counter** This value is copied from register **FCFGCx (x = 0 - 3)**.
NVALCNT	[29:24]	rh	**Number of Values Counted** This value is copied from register **IWCTRx (x = 0 - 3)**.
0	[31:30]	r	**Reserved, write 0, read as 0**

20.12 Interconnects

This section describes the actual implementation of the DSD module into the XMC4500, i.e. the incorporation into the microcontroller system.

20.12.1 Product-Specific Configuration

The functional description describes the features and operating modes of the DSD in a general way. This section summarizes the configuration that is available in this product (XMC4500).

Table 20-6 General Converter Configuration in the XMC4500

Channel	Digital Inputs	Notes
0	4	
1	4	
2	4	
3	4	

20.12.2 Digital Module Connections in the XMC4500

The DSD module accepts a number of digital input signals and generates a number of output signals. This section summarizes the connection of these signals to other on-chip modules or to external resources via port pins.

Note: The exact port connections (pins) are listed in the ports chapter.

Table 20-7 Digital Connections in the XMC4500

Signal	Dir.	Source/Destin.	Description
Channel 0			
DIN0A	I	GPIO	Data bitstream channel 0 input A
DIN0B	I	GPIO	Data bitstream channel 0 input B
MCLK0A	I/O	GPIO	Modulator clock channel 0 input/output A
MCLK0B	I/O	GPIO	Modulator clock channel 0 input/output B
ITR0A	I	ERU1.PDOUT0	Trigger signal, channel 0, input A
ITR0B	I	ERU1.PDOUT1	Trigger signal, channel 0, input B
ITR0C	I	ERU1.PDOUT2	Trigger signal, channel 0, input C
ITR0D	I	ERU1.PDOUT3	Trigger signal, channel 0, input D
ITR0E	I	-	Trigger signal, channel 0, input E
ITR0F	I	-	Trigger signal, channel 0, input F
ITR0G	I	-	Trigger signal, channel 0, input G
ITR0H	I	-	Trigger signal, channel 0, input H
SRM0	O	NVIC, GPDMA	Service request output main channel 0
SRA0	O	NVIC	Service request output aux. channel 0
Channel 1			

Table 20-7 Digital Connections in the XMC4500 (cont'd)

Signal	Dir.	Source/Destin.	Description
DIN1A	I	GPIO	Data bitstream channel 1 input A
DIN1B	I	GPIO	Data bitstream channel 1 input B
MCLK1A	I/O	GPIO	Modulator clock channel 1 input/output A
MCLK1B	I/O	GPIO	Modulator clock channel 1 input/output B
ITR1A	I	ERU1.PDOUT0	Trigger signal, channel 1, input A
ITR1B	I	ERU1.PDOUT1	Trigger signal, channel 1, input B
ITR1C	I	ERU1.PDOUT2	Trigger signal, channel 1, input C
ITR1D	I	ERU1.PDOUT3	Trigger signal, channel 1, input D
ITR1E	I	-	Trigger signal, channel 1, input E
ITR1F	I	-	Trigger signal, channel 1, input F
ITR1G	I	-	Trigger signal, channel 1, input G
ITR1H	I	-	Trigger signal, channel 1, input H
SRM1	O	NVIC, GPDMA	Service request output main channel 1
SRA1	O	NVIC	Service request output aux. channel 1
Channel 2			
DIN2A	I	GPIO	Data bitstream channel 2 input A
DIN2B	I	GPIO	Data bitstream channel 2 input B
MCLK2A	I/O	GPIO	Modulator clock channel 2 input/output A
MCLK2B	I/O	GPIO	Modulator clock channel 2 input/output B
ITR2A	I	ERU1.PDOUT0	Trigger signal, channel 2, input A
ITR2B	I	ERU1.PDOUT1	Trigger signal, channel 2, input B
ITR2C	I	ERU1.PDOUT2	Trigger signal, channel 2, input C
ITR2D	I	ERU1.PDOUT3	Trigger signal, channel 2, input D
ITR2E	I	-	Trigger signal, channel 2, input E
ITR2F	I	-	Trigger signal, channel 2, input F
ITR2G	I	-	Trigger signal, channel 2, input G
ITR2H	I	-	Trigger signal, channel 2, input H
SRM2	O	NVIC, GPDMA	Service request output main channel 2
SRA2	O	NVIC	Service request output aux. channel 2

Table 20-7 Digital Connections in the XMC4500 (cont'd)

Signal	Dir.	Source/Destin.	Description
Channel 3			
DIN3A	I	GPIO	Data bitstream channel 3 input A
DIN3B	I	GPIO	Data bitstream channel 3 input B
MCLK3A	I/O	GPIO	Modulator clock channel 3 input/output A
MCLK3B	I/O	GPIO	Modulator clock channel 3 input/output B
ITR3A	I	ERU1.PDOUT0	Trigger signal, channel 3, input A
ITR3B	I	ERU1.PDOUT1	Trigger signal, channel 3, input B
ITR3C	I	ERU1.PDOUT2	Trigger signal, channel 3, input C
ITR3D	I	ERU1.PDOUT3	Trigger signal, channel 3, input D
ITR3E	I	-	Trigger signal, channel 3, input E
ITR3F	I	-	Trigger signal, channel 3, input F
ITR3G	I	-	Trigger signal, channel 3, input G
ITR3H	I	-	Trigger signal, channel 3, input H
SRM3	O	NVIC, GPDMA	Service request output main channel 3
SRA3	O	NVIC	Service request output aux. channel 3
General			
MODCLK	I	SCU: f_{PB}	Module clock
RESET	I	SCU	Reset signal (general)
SGNA	I	ERU1.PDOUT2	Sign input A (carrier signal)
SGNB	I	ERU1.PDOUT3	Sign input B (carrier signal)
CGPWMP	O	GPIO	Positive PWM signal of carrier generator
CGPWMN	O	GPIO	Negative PWM signal of carrier generator

21 Digital to Analog Converter (DAC)

This chapter describes the two Digital to Analog Converter (DAC) channels available in the module.

21.1 Overview

The module consists of two separate 12-bit digital to analog converters (DACs). It converts two digital input signals into two analog voltage signal outputs at a maximum conversion rate of 5 MHz. The available design structure is based on a current steering architecture with internal reference generation and provides buffered voltage outputs. In order to reduce power consumption during inactive periods, a power down mode is available.

A built-in wave generator mode allows the CPU free generation of a selectable choice of wave forms. Alternatively values can be feed via CPU or DMA directly to one or both DAC channels. Additionally an offset can be added and the amplitude can be scaled. Several time trigger sources are possible.

21.1.1 Features

Analog features

- DAC resolution 12 bit;
- Conversion rate up to 5 MHz with reduced accuracy;
- Conversion rate up to 2 MHz at full accuracy;
- Maximum settling time of 2 us for a full scale 12-bit input code transition;
- Buffered voltage output;
- Direct drive of 5 kOhm / 50 pF terminated load;
- Segmented current steering architecture;
- Low glitch energy;
- Power down mode;
- DAC output and ADC input share the same analog input pin. ADC measurement is possible in parallel to DAC usage.
- 3.3 V analog supply;

Digital features

- One Advanced Microcontroller Bus Architecture (AMBA) 32-bit AHB-Lite bus interface for data transfer and control of both DACs;
- Self triggered Direct Memory Access (DMA) handling capability with independent or simultaneous data handling for the two DAC channels (see **Chapter 21.2.4**);
- First In First Out (FIFO) data buffers to allow a longer service request latency and to guarantee a continuous data transfer to the DACs (see **Chapter 21.2.4**);

- Pattern generators available with freely programmable waveforms for both DACs (see **Chapter 21.2.5**);
- Independent noise generators available for both DACs (see **Chapter 21.2.6**);
- Data scaling by shift operation (multiplication and division by 2, 4, 8,..., 128) of the DACs' input data;
- Data offset value addition to the DACs input data;
- 8 selectable external trigger inputs;
- Internal integer clock divider for DAC trigger generation;
- Software trigger option;
- 1.2 V digital supply;

21.1.2 Block Diagram

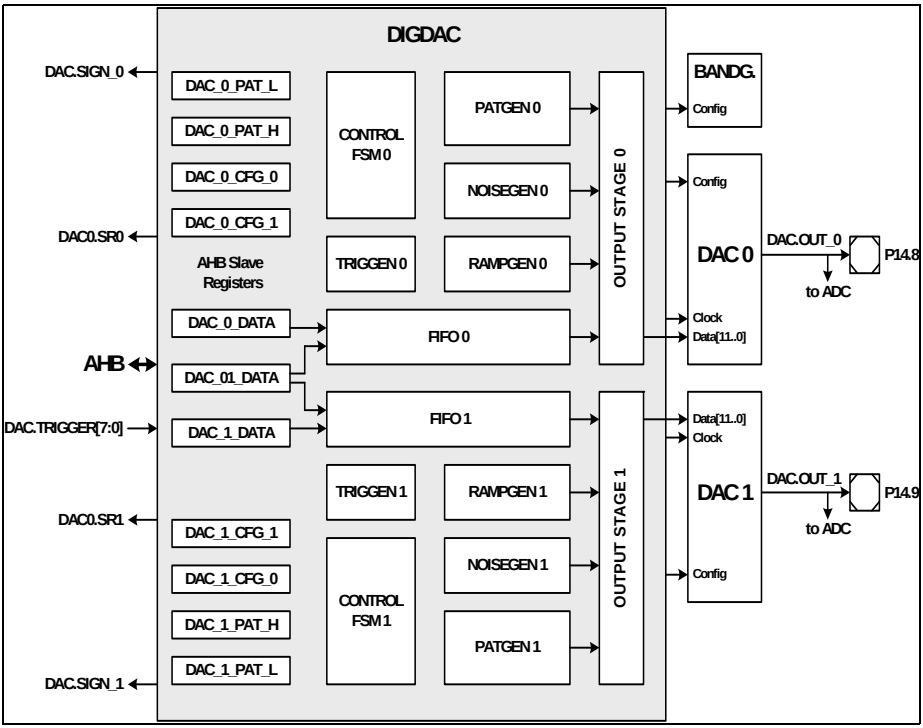


Figure 21-1 Block diagram of DAC module including digdac submodule

21.2 Operating Modes

The following chapters describe all the DAC's functional operating modes and how to use them. All used configuration parameters in this chapter are part of the registers described in **Chapter 21.6**.

21.2.1 Hardware features

To facilitate the understanding of the different operating modes, a brief description of the supporting hardware features is given here:

Control and Data Registers

All control and data registers shown on the left hand side of **Figure 21-1** are described in **Chapter 21.6** in detail. They are connected to the AHB-Lite bus via an AHB-Lite slave interface. The interface also handles the necessary error response generation without introducing any latency.

Control Logic - Finite State Machines (FSM)

The two control finite state machines control all data processing modes of the DAC (see **Figure 21-1**). This means that they are responsible for the start and stop operating sequence, the service request generation for DMA handling for the so called data-mode and the control of the data FIFOs, the pattern generators, the noise generators and the ramp generators. Both FSMs are equivalent in structure and both are able to operate fully independent for the two DAC channels. For the simultaneous data-mode both FSMs are active, but only the service request signal of channel 0 (DAC0.Service Request(SR)0) should be evaluated by the DMA controller. Of course in that simultaneous data-mode the trigger source has to be the same for both channels.

21.2.1.1 Trigger Generators (TG)

The block diagram in **Figure 21-2** shows one of the two DIGDAC's trigger generators.

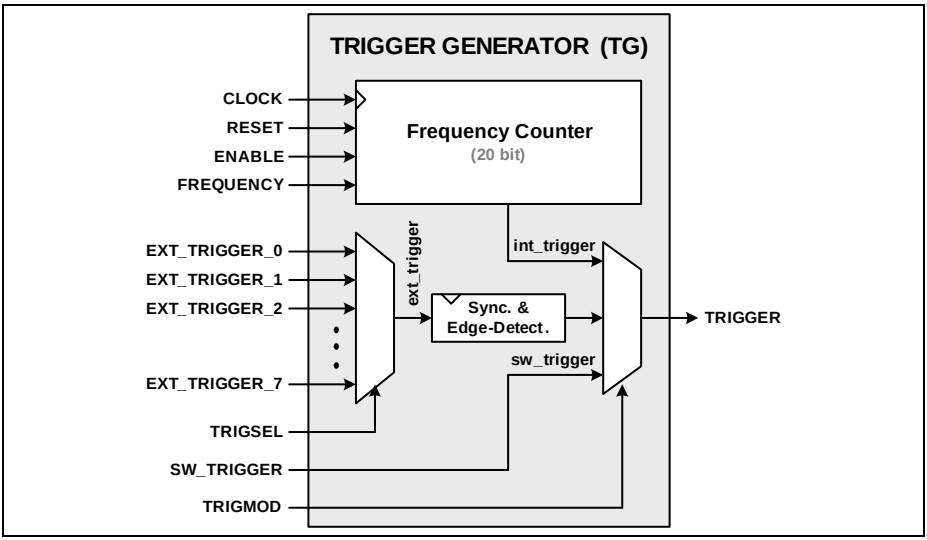


Figure 21-2 Trigger generator block diagram

The TG consists of two multiplexers and one frequency counter. The first multiplexer selects between one of the eight external trigger sources whereas the second one enables switching between this selected external trigger source, an internally generated trigger and an additional software trigger input. The internal trigger is generated by the frequency counter which operates as a simple integer clock divider. So the internal trigger period can only be a multiple of the DACs system clock period. In order to guarantee correct operation of the analog part of the DAC the smallest frequency divider value is limited to 16 by hardware.

The external trigger inputs are synchronized to the system clock and only the rising edge is evaluated by the TG.

The software trigger input is connected to a register bit which is automatically cleared after it has been set to one. For this reason the output trigger of the TG is always a pulse with a pulse width of one system clock cycle for all three trigger possible modes.

21.2.1.2 Data FIFO buffer (FIFO)

The block diagram in **Figure 21-3** shows one of the two data FIFO buffers, one for each DAC.

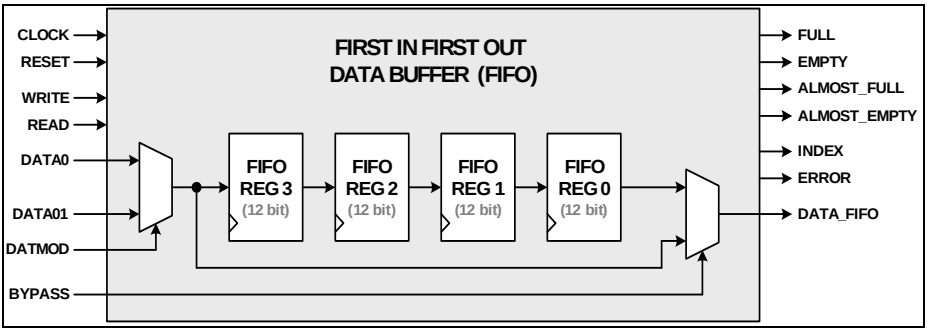


Figure 21-3 Data FIFO block diagram

This data FIFO buffer with four FIFO registers is introduced to allow a longer service request latency and to guarantee a continuous data processing for the DAC channels. It is used for DAC's so called data processing mode described in **Chapter 21.2.4**. All FIFO's read and write operations are controlled by the corresponding Control FSM. A read operation is triggered by the chosen trigger and a write operation is initiated by an Advanced High-performance Bus (AHB) write operation to the currently used data register. The FIFO's status outputs named full, empty and index can be read by the software. A FIFO bypass used for all other DACs operating modes is also available.

21.2.1.3 Data output stage

The block diagram in **Figure 21-4** shows one of the two DACs' data output stages.

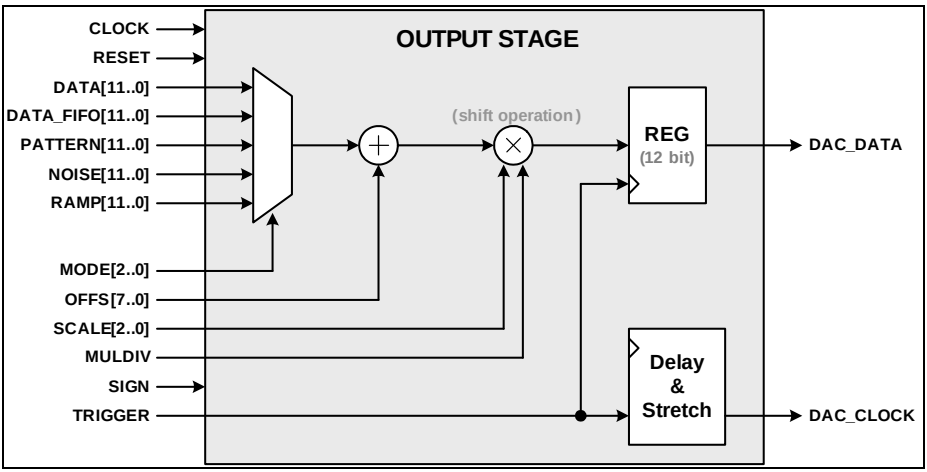


Figure 21-4 Data output stage block diagram

This output stage is the last element in the DAC's data path before the data is converted to the analog. It consists of a multiplexer, an adder, a multiplier, an output register and the generation of the DAC clock output.

The multiplexer selects between the five possible data sources and is programmed with the mode parameter. The adder stage gives the possibility to add an 8-bit offset value which is mainly needed for the PG mode in order to also process unsigned signal patterns to the DACs. In that case a certain offset value can be added to the signed output pattern values. The multiplier enables scaling by simple binary shifting of the data values. Therefore it allows multiplication and division by a programmed 2n scale value. The offset and scaling operations are possible in all functional operating modes. The output register contains the final sample delivered together with the corresponding trigger to the analog converter.

The clock output for operating the analog part of the DAC is generated using the DAC's trigger generator (TG). For that purpose the TG's trigger output is delayed by four system clock periods and stretched to a high-length of eight system clock periods.

21.2.1.4 Pattern Generators (PG) - Waveform Generator

The block diagram in **Figure 21-5** shows one of the two DAC's pattern generators.

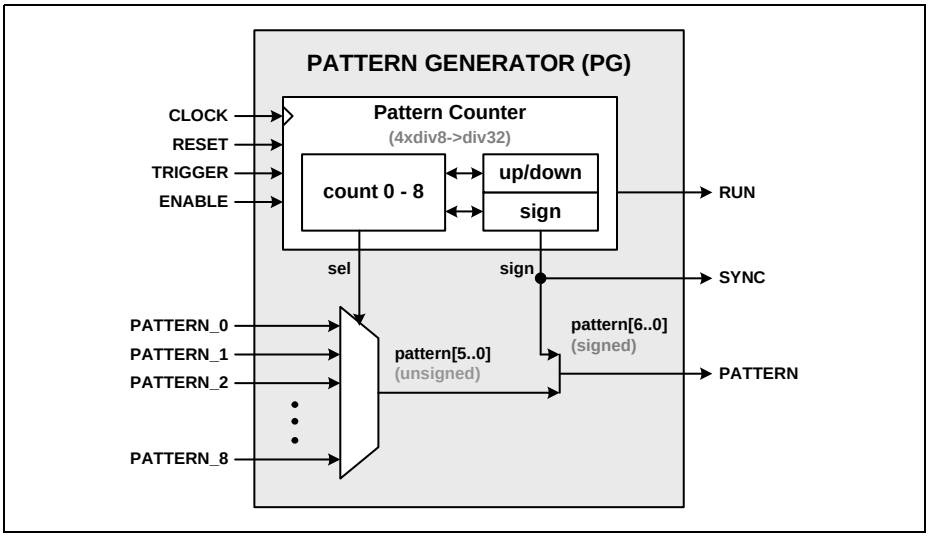


Figure 21-5 Pattern generator block diagram

The nine pattern inputs on the left side of the PG block diagram are directly connected to the pattern registers described in **Chapter 21.6.3.4**. The pattern registers contain only one quarter of the actual programmed periodic pattern. The output of the pattern counter

in the PG is used to select one of the nine input patterns. This pattern counter is an up-down counter with an additional sign output which is inverted every time the counter reaches zero. Since the sign information is concatenated with the currently selected pattern, it is possible to generate a complete pattern sequence for a full period of any 2*π periodic waveform. For a detailed description how to operate the pattern generator please also refer to **Chapter 21.2.5**.

21.2.1.5 Noise Generators (NG) - Pseudo Random Number Generator

The block diagram in **Figure 21-6** shows one of the two DAC's noise generators.

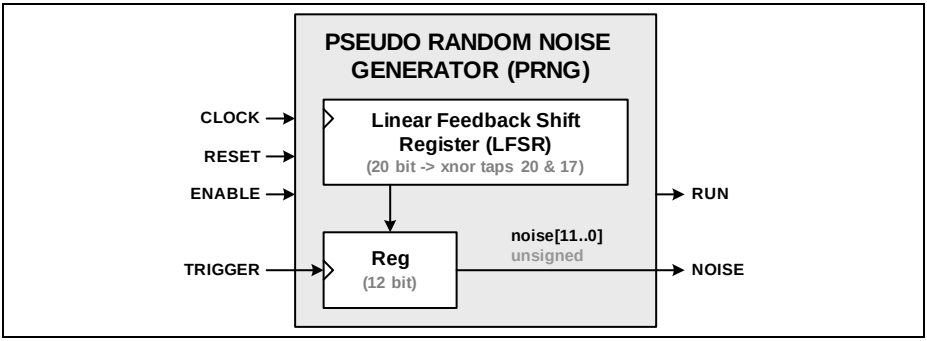


Figure 21-6 Noise generator block diagram

The NG outputs a 12-bit pseudo random number. A 20-bit LFSR (linear feedback shift register) operating at the system clock frequency and a sample output register which is triggered by the NG's trigger inputs are used for this purpose. After enabling the NG, the LFSR is set back to its reset value and therefore it always starts outputting the same pseudo random number sequence.

21.2.1.6 Ramp Generators (RG)

The block diagram in **Figure 21-7** shows one of the two DIGDAC's ramp generators.

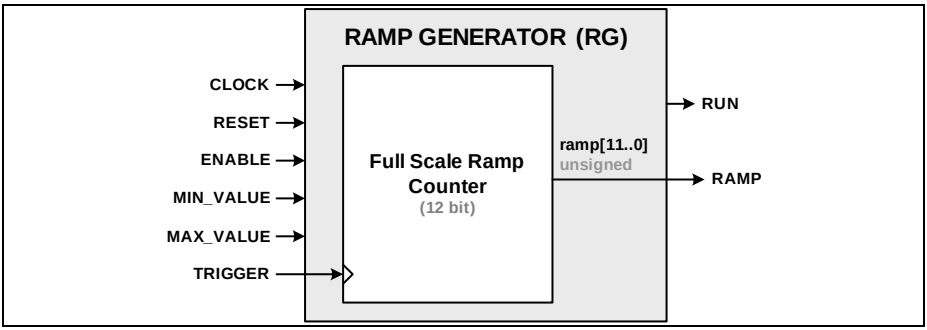


Figure 21-7 Ramp generator block diagram

The ramp generator is basically an 12-bit up counter representing the full DAC range. If the RG is enabled it always starts at the programmed minimum value. The up-counting by ones is triggered by the selected trigger of the DAC channel. If the ramp counter reaches the programmed maximum value it restarts from the minimum value. This allows the generation of ramps within any desired value range and by variation of the trigger frequency also the ramp's slope can be modified.

21.2.2 Entering any Operating Mode

Before entering the desired operating mode with the **MODE** parameter, the corresponding analog DAC channel should be enabled with **ANAEN** and the startup time of the analog DAC channel should be considered.

Setting the **DATMOD** parameter to one enables simultaneous data processing. This means that both DAC channels use the same trigger source and both channels are always started and stopped synchronously. Hence the parameter setting of **MODE**, **TRIGMOD**, **TRIGSEL** and **FREQ** for DAC channel 0 is used for DAC channel 1 also.

21.2.3 Single Value Mode

By setting **MODE** to "single value mode", it is possible to convert only one single data value by the DACs. To start a conversion, a data value can be written to either **DATA0** or **DATA1** in **DAC0DATA** or **DAC1DATA** registers. This write operation itself then initiates a single trigger pulse and the value gets processed by DAC0 or DAC1. Only for this mode, no further external, internal or software trigger pulse is necessary. The DAC holds the processed value until a new value is written to **DATA0** or **DATA1**. This operating mode is intended for outputting static DAC output values. For processing sequential data streams in this "self triggered", single value mode, it is important not to exceed the maximum DAC data rate. If the **DATMOD** parameter is set to zero, data from the independent data registers **DAC0DATA** or **DAC1DATA** is processed. Whereas if

DATMOD is set to one, data from the simultaneous data register **DAC01DATA** is processed for both DACs.

21.2.4 Data Processing Mode

This operating mode is intended for continuous data processing from the system memory to DAC0 and/or DAC1. To enable it, the **MODE** parameter has to be set to data mode. Also, the desired trigger source has to be selected by setting **TRIGMOD**, **TRIGSEL** and **FREQ** in the configuration registers. The DIGDAC can operate either with an internal generated trigger, one of the eight external trigger sources (see **Figure 21-2**) or the software trigger bit **SWTRIG**.

Simultaneous and independent Data Modes

With the **DATMOD** parameter, either the simultaneous or the independent “data mode” can be selected. In the simultaneous mode, both DACs receive their data from the same register **DAC01DATA**. In the independent “data mode” DAC0 gets its data from **DAC0DATA** and DAC1 from **DAC1DATA**. The two data paths are shown in **Figure 21-8** and also in **Figure 21-3**. Both DAC channels can activate a service request output signal to trigger a DMA channel, if they are configured in this mode and if the corresponding **SREN** bit is set to one. The service requests are DAC0.SR0 for the DAC0 channel and DAC0.SR1 for the DAC1 channel. For the simultaneous mode either DAC0.SR0 or DAC0.SR1 can be used by the DMA controller.

Start-Stop Operation

Before the DAC is started in “data mode”, all configuration registers DACx_CFG_x should be set according to the desired processing mode (see **Chapter 21.6.3.2**). Once this has been done, the DAC can be started by setting the **MODE** parameter to “data mode”. The control FSM will then start its operation until it reaches the run status indicated by the read parameter **RUN**. The run state can be left either by an operation error or by setting the **MODE** parameter to “disable DAC” again. An operation error can be a FIFO overflow or a FIFO underflow (see **Figure 21-3** and **Figure 21-8**).

21.2.4.1 FIFO Data Handling

Figure 21-8 shows the data handling for the FIFO of the DAC0 channel. Certainly the same structure also exists for the DAC1 channel (see **Figure 21-1** and **Figure 21-3** also). The data word **DATA0** from either data register **DAC0DATA** or **DAC01DATA** on the left hand side in **Figure 21-8** is loaded into one of the FIFO buffers registers. The load position in the FIFO depends on its actual filling level represented by the read parameters **FIFOIND**, **FIFOEMP** and **FIFOFUL**. If the FIFO is empty, **FIFOIND** = 0. If it is full, **FIFOIND** = 3. If a trigger occurs and the FIFO is not empty, the data is shifted to the next register (from left to right). At the same time, a service request (DAC0.SR0 or DAC0.SR1) is initiated in order to fill up the FIFO again. The service request should end

with a write operation to **DAC0DATA** or **DAC01DATA**. This write operation itself then triggers a write from the data registers to the FIFO buffer registers. If the FIFO stores only one last element (**FIFOIND** = 0 and **FIFOEMP** = 0) and a trigger has occurred, a service request is initiated and additionally **FIFOEMP** is set to 1. On the other hand, if there is only one last free register in the FIFO (**FIFOIND** = 2) and a write operation has been initiated, the **FIFOFUL** bit is set to 1. All the control signals for the FIFO handling are generated by the DAC's control FSM. This includes filling up the FIFO when "data mode" is entered and emptying the FIFO when leaving "data mode".

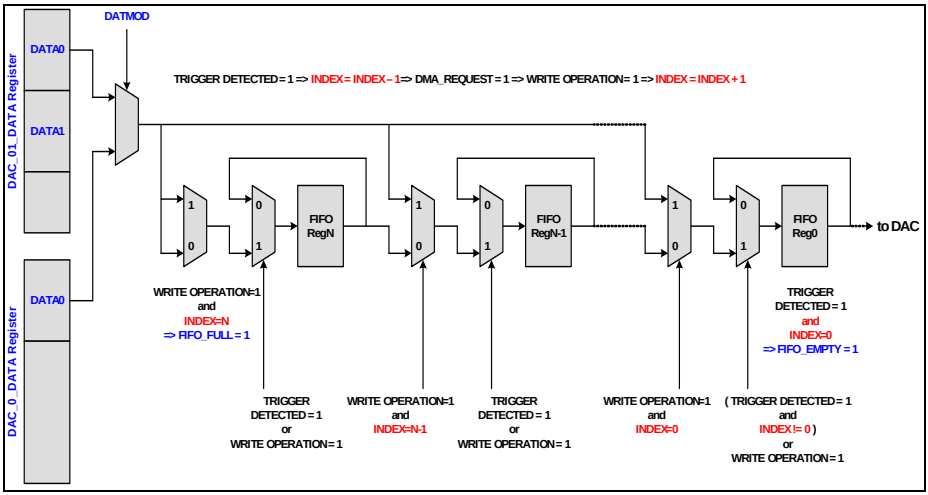


Figure 21-8 Data handling for the FIFO Buffer

21.2.5 Pattern Generation Mode

This chapter describes the operation of the pattern generator. This mode is used to output a pattern or waveform to the DACs and it is activated by setting the **MODE** configuration parameter to "patgen mode".

Like in the "data mode" (see **Chapter 21.2.4**), the DIGDAC in "patgen mode" can operate either with an internally generated trigger, one of the eight external trigger sources or a software trigger bit.

The desired pattern or waveform is freely programmable with the 5-bit parameters **PAT0** to **PAT8**. The pattern registers contain only one quadrant of the full waveform. The other quadrants of the 2*π periodic odd function are generated out of the first one with the help of a counter. The counter generates also the sign bit of the output pattern, therefore the 6-bit output signed values are in the range of -31 to +31.

In order to use the full range of the DAC in signed mode, a scaling by 32 by setting **SCALE** to "101" in the output stage is necessary. If the DAC should output a full range

pattern in unsigned mode, it is also possible to add an offset value programmed with **OFFS** to the output stage before doing the scaling. All the control signals for the pattern generators are generated by the DIGDAC's control FSM.

address	lookup triangle	lookup sine	lookup rectangle
1	0	0	31
2	4	6	31
3	8	12	31
4	12	17	31
5	16	22	31
6	19	26	31
7	23	29	31
8	27	30	31
9	31	31	31

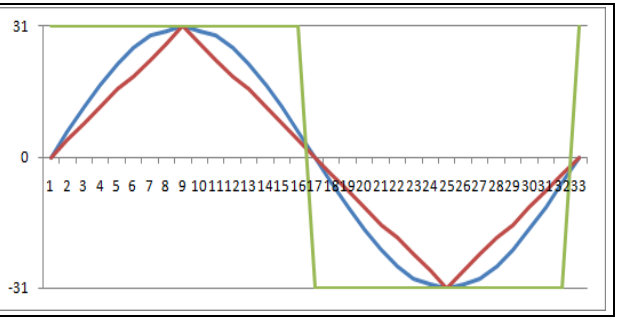


Figure 21-9 Example 5-bit patterns and their corresponding waveform output

Figure 21-9 gives examples of lookup table entries for triangular, sine and rectangular pattern. These values can be programmed to **PAT0** to **PAT8** in order to get the corresponding waveforms at the DAC's output like shown in the chart on the right hand side. If the pattern generation is restarted / enabled again, it always starts with the first value of the first quarter of the actual programmed pattern (positive value and up-counting). The current sign information of the generated pattern is one of the DAC's system on chip outputs (see **Chapter 21.7.2.3**) and can be enabled using the parameter **SIGNEN**.

21.2.6 Noise Generation Mode

A 20-bit linear feedback shift register (LFSR) is used to produce a pseudo random number. In order to enable the noise generator, the **MODE** parameter has to be set to "noise mode". The LFSR itself runs with the system clock. The 12-bit random numbers is sampled with the preselected trigger source using **TRIGMOD**, **TRIGSEL** and optionally also **FREQ** or **SWTRIG** into an output register. The 12-bit values can be interpreted as signed or unsigned values. By setting the **MODE** parameter to "disable DAC" again, the noise generation stops and the DAC holds its last processed value. **Figure 21-10** shows an example pseudo random noise output. After restarting the noise generation mode, the LFSR is set back to its reset value and therefore it always starts outputting the same pseudo random number sequence.

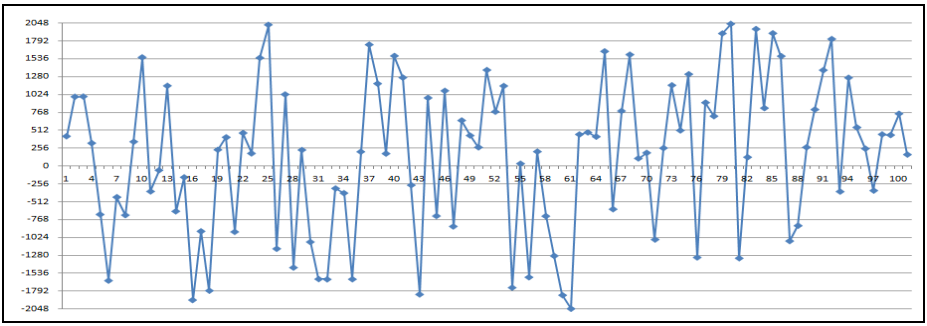


Figure 21-10 Signed 12-bit pseudo random noise example output

21.2.7 Ramp Generation Mode

A 12-bit ramp counter is also part of the DAC. It is activated by setting the **MODE** parameter to “ramp mode”. The trigger source can be selected using **TRIGMOD**, **TRIGSEL** and optionally also **FREQ** or **SWTRIG**. The ramp counter starts at a programmed start value and each trigger pulse increments the counter by one. The start values are programmable via **DATA0** for DAC channel 0 and **DATA1** for DAC channel 1 of the independent data registers. The stop values are programmable via **DATA0** or **DATA1** of the simultaneous data register. If the ramp counter reaches its stop value, it restarts from the start value with the next trigger pulse. This allows the generation of ramps within any desired value range. The ramp’s slope can be modified by varying the trigger frequency. **Figure 21-11** shows two examples of ramp generation output waveforms.

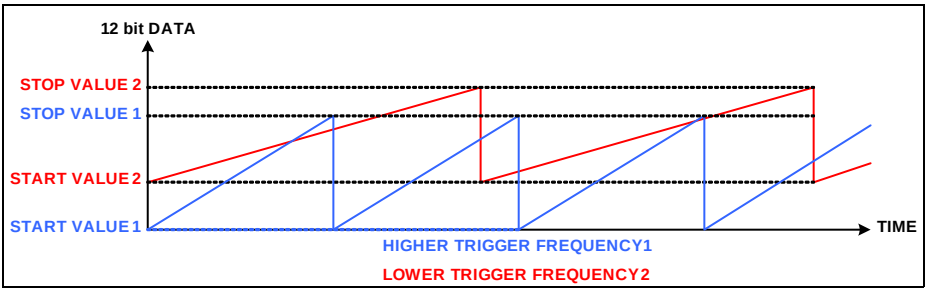


Figure 21-11 Unsigned 12-bit ramp generator example output

21.3 Service Request Generation

Service Requests are available in **Data Processing Mode** only.

21.4 Power, Reset and Clock

As long as **ANAEN** is set to default value “standby”, the corresponding DAC stays in power down mode, and its output is floating.

The DAC module reset is shared with the peripheral bus reset line, as well as the module clock is shared with the peripheral bus clock line.

With **FREQ** the clock divider ratio of the internal trigger generator is set.

21.5 Initialisation

A feasible initialisation sequence of the DAC reads as follows:

1st Step: De-assert the reset of DAC module by setting DACRS bit in PRCLR1 register

2nd Step: Write the DACxCFG0 register values. Here you select the operating mode of the corresponding DAC channel by writing the **MODE** field, e.g. Patgen mode. By setting or clearing the **SIGN** bit, the choice between signed and unsigned input data format is made.

In the same step service request generation can be enabled with the **SREN** bit, as well as sign output with **SIGNEN** bit. Also the frequency divider of the internal trigger generator can be set up by writing the **FREQ** field.

3rd Step: Write the DACxCFG1 register values. Here you select the trigger source by writing the **TRIGMOD** field, e.g. software trigger. The DAC channel output is enabled by setting the **ANAEN** bit. You also need to choose now your values for **SCALE**, **MULDIV**, **OFFS**, and **DATMOD** fields.

4th Step: Configure the chosen data source. E.g. in case you selected Patgen mode, the pattern must be defined by programming DACxPATL and DACxPATH registers.

5th Step: E.g. in case software trigger is selected, the runtime code is responsible for generating the trigger signals by setting **SWTRIG** bit inside DACxCFG1 register.

C code example

The C code of this example is given below:

```
void DAC_init()
{
//RESET MODULE
       SCU_RESET->PRCLR1 |= 0x00000020; //De-asserts reset for
VADC module

//DAC0 CONFIGURATION AS PATTERN GENERATOR
       DAC->DAC0CFG0=0x20300FFF;           //Pattern gen enable,
Signal enabled and Frequency
       DAC->DAC0CFG1=0x010405FD;           //Enable AN, Software
trigger
```

```
                                                    // Offset and
Scale divider set up.

        DAC->DAC0PATL=0x3568B0C0;              //Sinus waveform
configuration
        DAC->DAC0PATH=0x00007FDD;              //Sinus waveform
configuration
   //For triangle waveform use:
     //DAC->DAC0PATL=0x27062080;            //Triangle waveform
configuration
     //DAC->DAC0PATH=0x00007F77;            //Triangle waveform
configuration

//DAC1 CONFIGURATION AS RAMP GENERATOR
      DAC->DAC1CFG0=0x20500000;       //Ramp Mode, Signal enabled
and Frequency
      DAC->DAC1CFG1=0x01000000;    //Enable AN. No offset or
scaling. Internal Trigger

      DAC->DAC1DATA=0x00000003F;       //Start value
      DAC->DAC01DATA=0x0AFF0000;       //Stop value
}
-------------------------------------------------------------------------------------------
For SW trigger in runtime code:

      DAC->DAC0CFG1|=0x00010000;      //Software Trigger of DAC0
```

21.6 Registers

21.6.1 Address Map

The DAC is available at the following base address:

Table 21-1 Registers Address Space

Module	Base Address	End Address	Note
DAC	4801 8000$_H$	4801 BFFF$_H$	16 kB

21.6.2 Register Overview

Table 21-2 shows all registers required for the operation of the DAC module:

Table 21-2 Register Overview of DAC

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
ID	Module Identification Register	000$_H$	U, PV, 32	U, PV, 32	Page 21-16
DAC0CFG0	DAC0 Configuration Register Number 0	004$_H$	U, PV, 32	U, PV, 32	Page 21-17
DAC0CFG1	DAC0 Configuration Register Number 1	008$_H$	U, PV, 32	U, PV, 32	Page 21-18
DAC1CFG0	DAC1 Configuration Register Number 0	00C$_H$	U, PV, 32	U, PV, 32	Page 21-20
DAC1CFG1	DAC1 Configuration Register Number 1	010$_H$	U, PV, 32	U, PV, 32	Page 21-22
DAC0DATA	Data Register for DAC0 for Independent Data Mode	014$_H$	U, PV, 32	U, PV, 32	Page 21-24
DAC1DATA	Data Register for DAC1 for Independent Data Mode	018$_H$	U, PV, 32	U, PV, 32	Page 21-24
DAC01DATA	Data Register for DAC0 and DAC1 for Simultaneous Data Mode	01C$_H$	U, PV, 32	U, PV, 32	Page 21-25
DAC0PATL	Lower Samples of Pattern for DAC0 PATGEN	020$_H$	U, PV, 32	U, PV, 32	Page 21-26

Table 21-2 Register Overview of DAC (cont'd)

Short Name	Description	Offset Addr.[1]	Access Mode		Description See
			Read	Write	
DAC0PATH	Higher Samples of Pattern for DAC0 PATGEN	024_H	U, PV, 32	U, PV, 32	Page 21-26
DAC1PATL	Lower Samples of Pattern for DAC1 PATGEN	028_H	U, PV, 32	U, PV, 32	Page 21-27
DAC1PATH	Higher Samples of Pattern for DAC1 PATGEN	$02C_H$	U, PV, 32	U, PV, 32	Page 21-28

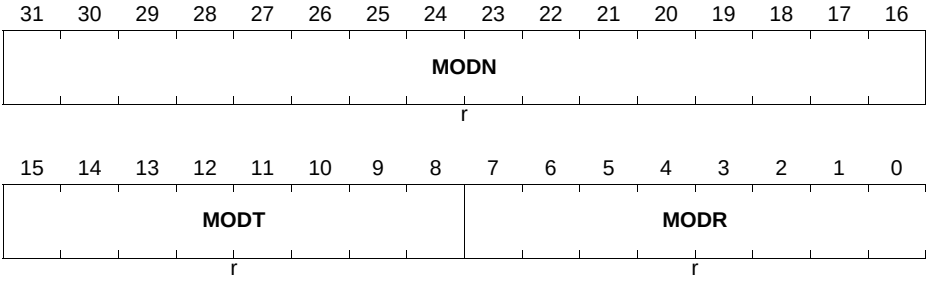
1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

21.6.3 Register Description

21.6.3.1 DAC_ID Register

The DAC module identification register contains the XMC4000 ID code.

DAC_ID
Module Identification Register (000_H) Reset Value: 00A5 C0XX$_H$



Field	Bits	Type	Description
MODR	[7:0]	r	**Module Revision** MOD_REV defines the module revision number. The value of a module revision starts with 01_H (first rev.).
MODT	[15:8]	r	**Module Type** This bit field is $C0_H$. It defines the module as a 32-bit module.

Field	Bits	Type	Description
MODN	[31:16]	r	**Module Number** For the DIGDAC this bit field is $A5_H$

21.6.3.2 DAC Configuration Registers

The DAC configuration registers contain all the necessary bits to set DAC0 and DAC1 in the desired operating mode and to start and stop conversations.

DAC0CFG0
DAC0 Configuration Register 0 (004_H) Reset Value: 0000 0000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RUN	**SREN**	**SIGNEN**	**0**	**FIFOFUL**	**FIFOEMP**	**FIFOIND**		**SIGN**	**MODE**			**FREQ**			
rh	rw	rw	r	rh	rh	rh		rw	rw			rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FREQ															
rw															

Field	Bits	Type	Description
FREQ	[19:0]	rw	**Integer Frequency Divider Value** • 0 to 16: divide by 16 • 16 to 2^20-1 divide by FREQ
MODE	[22:20]	rw	**Enables and Sets the Mode for DAC0** 000_B disable/switch-off DAC 001_B Single Value Mode 010_B Data Mode 011_B Patgen Mode 100_B Noise Mode 101_B Ramp Mode 110_B na 111_B na
SIGN	23	rw	**Selects Between Signed and Unsigned DAC0 Mode** 0_B DAC expects unsigned input data 1_B DAC expects signed input data
FIFOIND	[25:24]	rh	**Current write position inside the data FIFO**

Field	Bits	Type	Description
FIFOEMP	26	rh	**Indicate if the FIFO is empty** 0_B FIFO not empty 1_B FIFO empty
FIFOFUL	27	rh	**Indicate if the FIFO is full** 0_B FIFO not full 1_B FIFO full
0	28	r	**Reserved** Read as 0; Should be written with 0.
SIGNEN	29	rw	**Enable Sign Output of DAC0 Pattern Generator** 0_B Disable 1_B Enable
SREN	30	rw	**Enable DAC0 service request interrupt generation** 0_B disable 1_B enable
RUN	31	rh	**RUN indicates the current DAC0 operation status** 0_B DAC0 channel disabled 1_B DAC0 channel in operation RUN is set/cleared by hardware.

DAC0CFG1
DAC0 Configuration Register 1 (008_H) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
REFCFGL				0			ANA EN		ANACFG				TRIGMOD		SWT RIG
rw				r			rw		rw				rw		rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DAT MOD	TRIGSEL			OFFS								MUL DIV	SCALE ANACFG		
rw	rw			rw								rw	rw		

Field	Bits	Type	Description
SCALE	[2:0]	rw	**Scale value for up- or downscale of the DAC0 input data in steps by the power of 2 (=shift operation)** 000_B no shift = multiplication/division by 1 001_B shift by 1 = multiplication/division by 2 010_B shift by 2 = multiplication/division by 4 011_B shift left by 3 = multiplication/division by 8 100_B shift left by 4 = multiplication/division by 16 101_B shift left by 5 = multiplication/division by 32 110_B shift left by 6 = multiplication/division by 64 111_B shift left by 7 = multiplication/division by 128
MULDIV	3	rw	**Switch between up- and downscale of the DAC0 input data values** 0_B downscale = division (shift SCALE positions to the right) 1_B upscale = multiplication (shift SCALE positions to the left)
OFFS	[11:4]	rw	**8-bit offset value addition** e.g.: PATGEN output is a sine wave -31 to +31 and OFFS = 31 => the DAC0 input data will be a sine wave with an amplitude between 0 and 62. Depending on the SIGN bit this value is interpreted as signed or unsigned.
TRIGSEL	[14:12]	rw	**Selects one of the eight external trigger sources for DAC0**
DATMOD	15	rw	**Switch between independent or simultaneous DAC mode and select the input data register for DAC0 and DAC1** 0_B independent data handling - process data from DATA0 register (bits 11:0) to DAC0 and data from DATA1 register (bits 11:0) to DAC1 1_B simultaneous data handling - process data from DAC01 register to both DACs (bits 11:0 to DAC0 and bits 23:12 to DAC1). Trigger setting and MODE parameter for DAC0 is used for DAC1 also if DATMOD is set to 1 = simultaneous data mode!

Field	Bits	Type	Description
SWTRIG	16	rwh	**Software Trigger** Triggers DAC channel 0 if TRIGMOD is set to 10. Setting the bit to 1 generates one trigger pulse. The bit is cleared (set to 0) automatically. If DATMOD is set to simultaneous data mode this bit is used for both DAC channels (see DATMOD parameter).
TRIGMOD	[18:17]	rw	**Select the trigger source for channel 0** 00_B internal Trigger (integer divided clock - see FREQ parameter) 01_B external Trigger (preselected trigger by TRIGSEL parameter) 10_B software Trigger (see SWTRIG parameter) 11_B reserved
ANACFG	[23:19]	rw	**DAC0 analog configuration/calibration parameters** reserved for future use
ANAEN	24	rw	**Enable analog DAC for channel 0** 0_B DAC0 is set to standby (analog output only) 1_B enable DAC0 (analog output only)
0	[27:25]	r	**Reserved** Read as 0; Should be written with 0.
REFCFGL	[31:28]	rw	**Lower 4 band-gap configuration/calibration parameters** reserved for future use

DAC1CFG0
DAC1 Configuration Register 0 ($00C_H$) Reset Value: 0000 0000_H

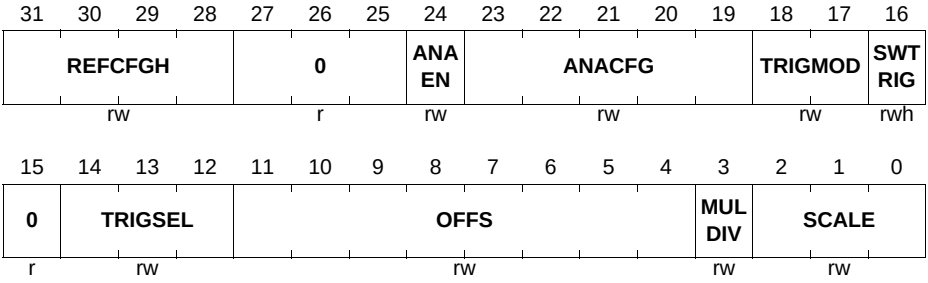
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RUN	**SREN**	**SIGNEN**	**0**	**FIFOFUL**	**FIFOEMP**	**FIFOIND**		**SIGN**	**MODE**			**FREQ**			
rh	rw	rw	r	rh	rh	rh		rw	rw			rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FREQ															
rw															

Field	Bits	Type	Description
FREQ	[19:0]	rw	**Integer Frequency Divider Value** • 0 to 16: divide by 16 • 16 to 2^20-1 divide by FREQ FREQ for DAC1 is not applicable if DATMOD is set to 1 = simultaneous data mode.
MODE	[22:20]	rw	**Enables and sets the Mode for DAC1** 000_B disable/switch-off DAC 001_B Single Value Mode 010_B Data Mode 011_B Patgen Mode 100_B Noise Mode 101_B Ramp Mode 110_B na 111_B na MODE for DAC1 is not applicable if DATMOD is set to 1 = simultaneous data mode.
SIGN	23	rw	**Selects between signed and unsigned DAC1 mode** 0_B DAC expects unsigned input data 1_B DAC expects signed input data
FIFOIND	[25:24]	rh	**Current write position inside the data FIFO**
FIFOEMP	26	rh	**Indicate if the FIFO is empty** 0_B FIFO not empty 1_B FIFO empty
FIFOFUL	27	rh	**Indicate if the FIFO is full** 0_B FIFO not full 1_B FIFO full
0	28	r	**Reserved** Read as 0; Should be written with 0.
SIGNEN	29	rw	**Enable sign output of DAC1 pattern generator** 0_B disable 1_B enable
SREN	30	rw	**Enable DAC1 service request interrupt generation** 0_B disable 1_B enable

Field	Bits	Type	Description
RUN	31	rh	**RUN indicates the current DAC1 operation status** 0_B DAC1 channel disabled 1_B DAC1 channel in operation RUN is set/cleared by hardware.

DAC1CFG1
DAC1 Configuration Register 1 (010_H) Reset Value: 0000 0000_H



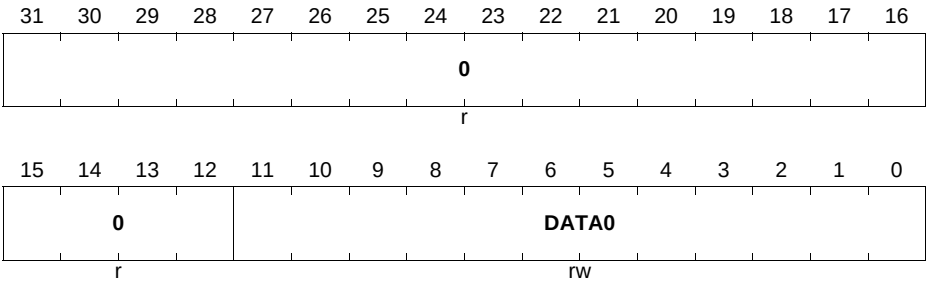
Field	Bits	Type	Description
SCALE	[2:0]	rw	**Scale value for up- or downscale of the DAC1 input data in steps by the power of 2 (=shift operation)** 000_B no shift = multiplication/division by 1 001_B shift by 1 = multiplication/division by 2 010_B shift by 2 = multiplication/division by 4 011_B shift left by 3 = multiplication/division by 8 100_B shift left by 4 = multiplication/division by 16 101_B shift left by 5 = multiplication/division by 32 110_B shift left by 6 = multiplication/division by 64 111_B shift left by 7 = multiplication/division by 128
MULDIV	3	rw	**Switch between up- and downscale of the DAC1 input data values** 0_B downscale = division (shift SCALE positions to the right) 1_B upscale = multiplication (shift SCALE positions to the left)

Field	Bits	Type	Description
OFFS	[11:4]	rw	**8-bit offset value addition** e.g.: PATGEN output is a sine wave -31 to +31 and OFFS = 31 => the DAC1 input data will be a sine wave with an amplitude between 0 and 62. Depending on the SIGN bit this value is interpreted as signed or unsigned.
TRIGSEL	[14:12]	rw	**Selects one of the eight external trigger sources for DAC1** TRIGSEL for DAC1 is not applicable if DATMOD is set to 1 = simultaneous data mode.
0	15	r	**Reserved** Read as 0; Should be written with 0.
SWTRIG	16	rwh	**Software Trigger** Triggers DAC channel 1 if TRIGMOD is set to 10. Setting the bit to 1 generates one trigger pulse. The bit is cleared (set to 0) automatically. If DATMOD is set to simultaneous data mode (see DATMOD parameter) this bit is not applicable and the SWTRIG bit from channel 0 is used for channel 1 also.
TRIGMOD	[18:17]	rw	**Select the trigger source for channel 1** 00_B internal Trigger (integer divided clock - see FREQ parameter) 01_B external Trigger (preselected trigger by TRIGSEL parameter) 10_B software Trigger (see SWTRIG parameter) 11_B reserved
ANACFG	[23:19]	rw	**DAC1 analog configuration/calibration parameters** reserved for future use
ANAEN	24	rw	**Enable analog DAC for channel 1** 0_B DAC1 is set to standby (analog output only) 1_B enable DAC1 (analog output only)
0	[27:25]	r	**Reserved** Read as 0; Should be written with 0.
REFCFGH	[31:28]	rw	**Higher 4 band-gap configuration/calibration parameters** reserved for future use

21.6.3.3 DAC Data Registers

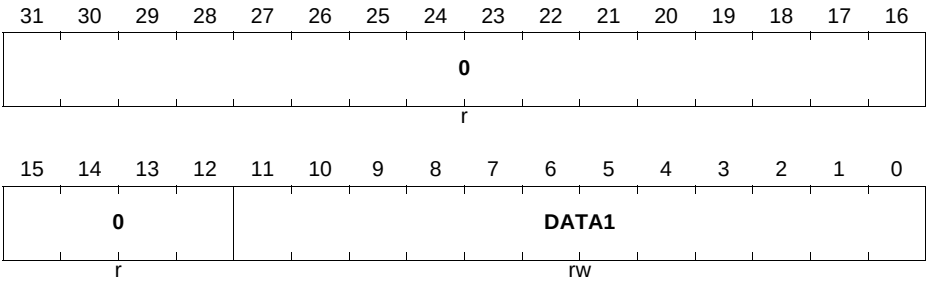
The DAC data registers contain the data provided to DAC0 and DAC1 either in simultaneous data mode (DAC01DATA) or in independent data mode (DAC0DATA and DAC1DATA).

DAC0DATA
DAC0 Data Register **(014_H)** **Reset Value: 0000 0000$_H$**



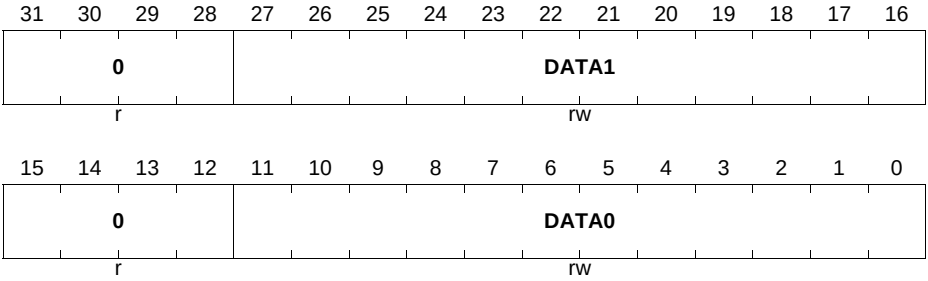
Field	Bits	Type	Description
DATA0	[11:0]	rw	**DAC0 Data Bits** Used as DAC0 data value and as counter start value in ramp generation mode
0	[31:12]	r	**Reserved** Read as 0; Should be written with 0.

DAC1DATA
DAC1 Data Register **(018_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
DATA1	[11:0]	rw	**DAC1 Data Bits** Used as DAC1 data value and as counter start value in ramp generation mode
0	[31:12]	r	**Reserved** Read as 0; Should be written with 0.

DAC01DATA
DAC01 Data Register **($01C_H$)** **Reset Value: 0000 0000$_H$**

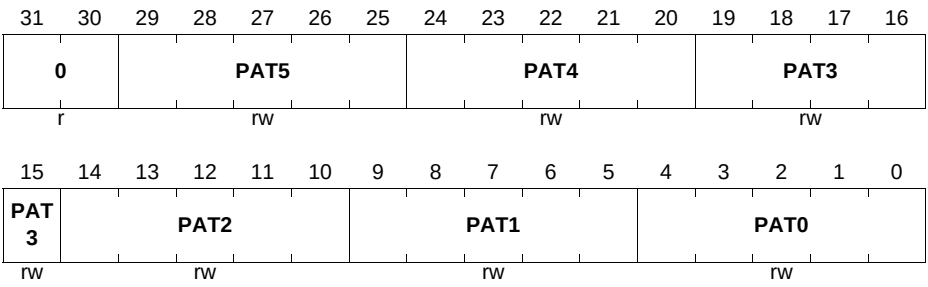


Field	Bits	Type	Description
DATA0	[11:0]	rw	**DAC0 Data Bits** Used as DAC0 data value and as counter stop value in ramp generation mode
0	[15:12]	r	**Reserved** Read as 0; Should be written with 0.
DATA1	[27:16]	rw	**DAC1 Data Bits** Used as DAC1 data value and as counter stop value in ramp generation mode
0	[31:28]	r	**Reserved** Read as 0; Should be written with 0.

21.6.3.4 DAC Pattern Registers

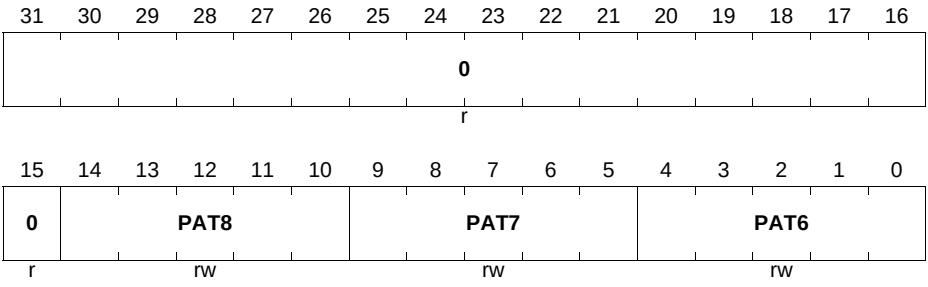
The DAC pattern registers contain the waveform patterns for the pattern generators of DAC0 and DAC1.

DAC0PATL
DAC0 Lower Pattern Register **(020_H)** **Reset Value: 3568 B0C0$_H$**



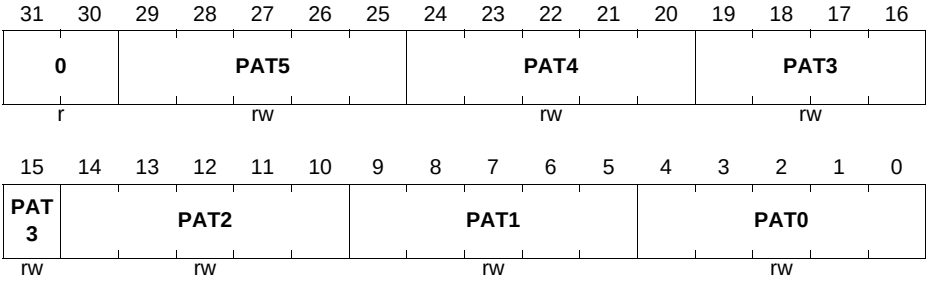
Field	Bits	Type	Description
PAT0	[4:0]	rw	**Pattern Number 0 for PATEGN of DAC0**
PAT1	[9:5]	rw	**Pattern Number 1 for PATEGN of DAC0**
PAT2	[14:10]	rw	**Pattern Number 2 for PATEGN of DAC0**
PAT3	[19:15]	rw	**Pattern Number 3 for PATEGN of DAC0**
PAT4	[24:20]	rw	**Pattern Number 4 for PATEGN of DAC0**
PAT5	[29:25]	rw	**Pattern Number 5 for PATEGN of DAC0**
0	[31:30]	r	**Reserved** Read as 0; Should be written with 0

DAC0PATH
DAC0 Higher Pattern Register **(024_H)** **Reset Value: 0000 7FDD$_H$**



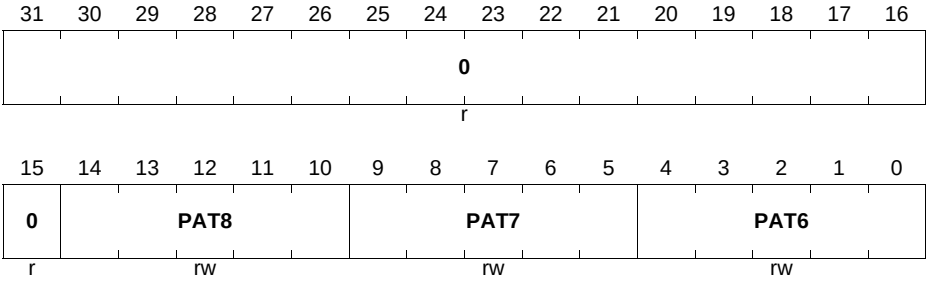
Field	Bits	Type	Description
PAT6	[4:0]	rw	**Pattern Number 6 for PATEGN of DAC0**
PAT7	[9:5]	rw	**Pattern Number 7 for PATEGN of DAC0**
PAT8	[14:10]	rw	**Pattern Number 8 for PATEGN of DAC0**
0	[31:15]	r	**Reserved** Read as 0; Should be written with 0.

DAC1PATL
DAC1 Lower Pattern Register **(028_H)** **Reset Value: 3568 B0C0$_H$**



Field	Bits	Type	Description
PAT0	[4:0]	rw	**Pattern Number 0 for PATEGN of DAC1**
PAT1	[9:5]	rw	**Pattern Number 1 for PATEGN of DAC1**
PAT2	[14:10]	rw	**Pattern Number 2 for PATEGN of DAC1**
PAT3	[19:15]	rw	**Pattern Number 3 for PATEGN of DAC1**
PAT4	[24:20]	rw	**Pattern Number 4 for PATEGN of DAC1**
PAT5	[29:25]	rw	**Pattern Number 5 for PATEGN of DAC1**
0	[31:30]	r	**Reserved** Read as 0; Should be written with 0.

DAC1PATH
DAC1 Higher Pattern Register (02C$_H$) Reset Value: 0000 7FDD$_H$



Field	Bits	Type	Description
PAT6	[4:0]	rw	**Pattern Number 6 for PATEGN of DAC1**
PAT7	[9:5]	rw	**Pattern Number 7 for PATEGN of DAC1**
PAT8	[14:10]	rw	**Pattern Number 8 for PATEGN of DAC1**
0	[31:15]	r	**Reserved** Read as 0; Should be written with 0.

21.7 Interconnects

21.7.1 Analog Connections

The analog interface lines of the DAC are listed below:

Table 21-3 Analog connections

Input/Output	I/O	Connected To	Descriptions
DAC.OUT_0	O	P14.8	Analog output of channel 0
DAC.OUT_1	O	P14.9	Analog output of channel 1

21.7.2 Digital Connections

The DAC has the following system level connections to other modules:

21.7.2.1 Service Request Connections

Two service requests DAC.SR0 and DAC.SR1 are used for simultaneous and independent data mode. DAC.SR1 can be enabled on DMA channel 2 and DAC.SR0 can be enabled on DMA channel 3.

Table 21-4 Service request connections

Input/Output	I/O	Connected To	Descriptions
DAC.SR0	O	NVIC GPDMA	Service request
DAC.SR1	O	NVIC GPDMA	Service request

21.7.2.2 Trigger Connections

The eight trigger inputs are connected to the following sources:

Table 21-5 Trigger connections

Input/Output	I/O	Connected To	Descriptions
DAC.TRIGGER[0]	I	CCU80.SR1	Trigger
DAC.TRIGGER[1]	I	reserved	Trigger
DAC.TRIGGER[2]	I	CCU40.SR1	Trigger
DAC.TRIGGER[3]	I	CCU41.SR1	Trigger
DAC.TRIGGER[4]	I	Port	Trigger
DAC.TRIGGER[5]	I	Port	Trigger
DAC.TRIGGER[6]	I	U0C0.DX1INS	Trigger
DAC.TRIGGER[7]	I	U1C0.DX1INS	Trigger

21.7.2.3 Synchronization Interface of the Pattern Generator

The interface consists of only two output signals called “DAC.SIGN_0” and “DAC:SIGN_1”. They are generated by the pattern generator of the two DAC channels and represent the actual sign information of the processed signal waveform converted by the DACs (see **Figure 21-5**).

Table 21-6 Pattern Generator Synchronization connections

Input/Output	I/O	Connected To	Descriptions
DAC.SIGN_0	O	VADC.G0REQGTI VADC.G2REQGTI VADC.BGREQGTI ERU1.0A3	
DAC.SIGN_1	O	VADC.G1REQGTI VADC.G3REQGTI ERU1.2A3	

Industrial Control Peripherals

22 Capture/Compare Unit 4 (CCU4)

The CCU4 peripheral is a major component for systems that need general purpose timers for signal monitoring/conditioning and Pulse Width Modulation (PWM) signal generation. Power electronic control systems like switched mode power supplies or uninterruptible power supplies, can easily be implemented with the functions inside the CCU4 peripheral.

The internal modularity of CCU4, translates into a software friendly system for fast code development and portability between applications.

Table 22-1 Abreviations table

PWM	Pulse Width Modulation
CCU4x	Capture/Compare Unit 4 module instance x
CC4y	Capture/Compare Unit 4 Timer Slice instance y
ADC	Analog to Digital Converter
POSIF	Position Interface peripheral
SCU	System Control Unit
f_{ccu4}	CCU4 module clock frequency
f_{tclk}	CC4y timer clock frequency

Note: A small “y” or “x” letter in a register indicates an index

22.1 Overview

Each CCU4 module is comprised of four identical 16 bit Capture/Compare Timer slices, CC4y. Each timer slice can work in compare mode or in capture mode. In compare mode one compare channel is available while in capture mode, up to four capture registers can be used in parallel.

Each CCU4 module has four service request lines and each timer slice contains a dedicated output signal, enabling the generation of up to four independent PWM signals.

Straightforward timer slice concatenation is also possible, enabling up to 64 bit timing operations. This offers a flexible frequency measurement, frequency multiplication and pulse width modulation scheme.

A programmable function input selector for each timer slice, that offers up to nine functions, discards the need of complete resource mapping due to input ports availability.

A built-in link between the CCU4 and POSIF modules also enable a flexible digital motor control loop implementation, with direct coupling with a Rotary Encoder.

22.1.1 Features

CCU4 module features

Each CCU4 represents a combination of four timer slices, that can work independently in compare or capture mode. Each timer slice has a dedicated output for PWM signal generation.

All four CCU4 timer slices, CC4y, are identical in terms of available functions and operating modes. Avoiding this way the need of implementing different software routines, depending on which resource of CCU4 is used.

A built-in link between the four timer slices is also available, enabling this way a simplified timer concatenation and sequential operations.

General Features

- 16 bit timer cells
- capture and compare mode for each timer slice
  - four capture registers in capture mode
  - one compare channel in compare mode
- programmable low pass filter for the inputs
- built-in timer concatenation
  - 32, 48 or 64 bit width
- shadow transfer for the period and compare values
- programmable clock prescaler
- normal timer mode
- gated timer mode
- three counting schemes
  - center aligned
  - edge aligned
  - single shot
- PWM generation
- TRAP function
- start/stop can be controlled by external events
- counting external events
- four dedicated service request lines per CCU4

Additional features

- external modulation function
- load controlled by external events
- dithering PWM
- floating point pre scaler
- output state override by an external event
- easy connection with POSIF unit for:
  - rotary encoder mode
  - multi channel/multi phase control

CCU4 features vs. applications

On **Table 22-2** a summary of the major features of the CCU4 unit mapped with the most common applications.

Table 22-2 Applications summary

Feature	Applications
Four independent timer cells	Independent PWM generation: • Multiple buck/boost converter control (with independent frequencies) • Different modes of operation for each timer, increasing the resource optimization • Up to 2 Half-Bridges control • multiple Zero Voltage Switch (ZVS) converter control with easy link to the ADC channels.
Concatenated timer cells	Easy to configure timer extension up to 64 bit: • High dynamic trigger capturing • High dynamic signal measurement
Dithering PWM	Generating a fractional PWM frequency or duty cycle: • To avoid big steps on frequency or duty cycle adjustment in slow control loop applications • Increase the PWM signal resolution over time
Floating prescaler	Automated control signal measurement: • decrease SW activity for monitoring signals with high or unknown dynamics • emulating more than a 16 bit timer for system control
Up to 9 functions via external signals for each timer	Flexible resource optimization: • The complete set of external functions is always available • Several arrangements can be done inside a CCU4, e.g., one timer working in capture mode and one working in compare

Table 22-2 Applications summary (cont'd)

Feature	Applications
4 dedicated service request lines	Specially developed for: • generating interrupts for the microprocessor • flexible connectivity between peripherals, e.g. ADC triggering.
Linking with POSIF	Flexible profiles for: • Rotary Encoder connection • Hall Sensor • Modulating the 4 timer outputs via SW

22.1.2 Block Diagram

Each CCU4 timer slice can operate independently from the other slices for all the available modes. Each timer slice contains a dedicated input selector for functions linked with external events and has a dedicated compare output signal, for PWM signal generation.

The built-in timer concatenation is only possible with adjacent slices, e.g. CC40/CC41. Combinations for slice concatenations like, CC40/CC42 or CC40/CC43 are not possible.

The individual service requests for each timer slice (four per slice) are multiplexed into four module service requests lines, **Figure 22-1**.

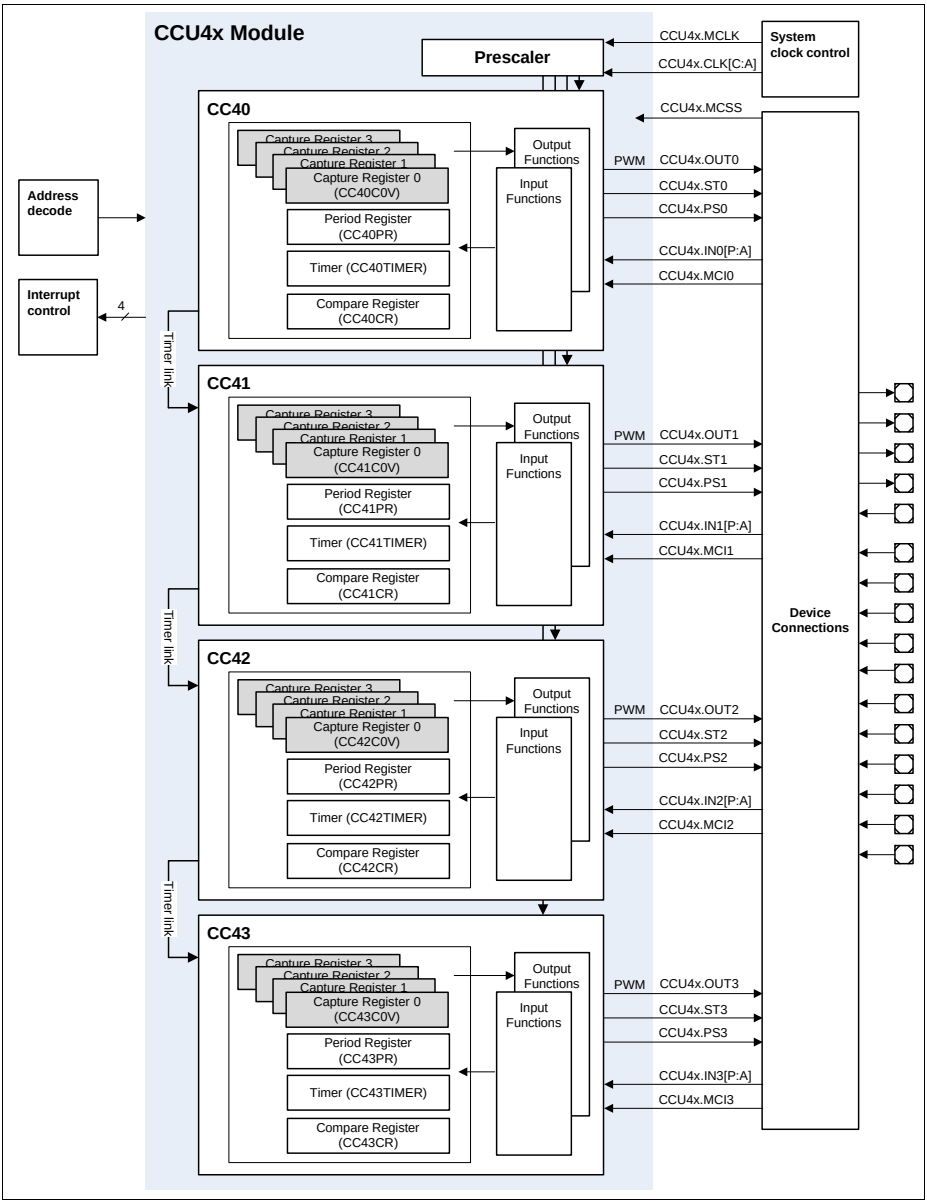


Figure 22-1 CCU4 block diagram

22.2 Functional Description

22.2.1 CC4y Overview

The input path of a CCU4 slice is comprised of a selector (**Section 22.2.2**) and a connection matrix unit (**Section 22.2.3**). The output path contains a service request control unit, a timer concatenation unit and two units that control directly the state of the output signal for each specific slice (for TRAP and modulation handling), see **Figure 22-2**.

The timer core is built of a 16 bit counter one period and one compare register in compare mode, or up to four capture registers in capture mode.

In compare mode the period register sets the maximum counting value while the compare channel is controlling the ACTIVE/PASSIVE state of the dedicated comparison slice output.

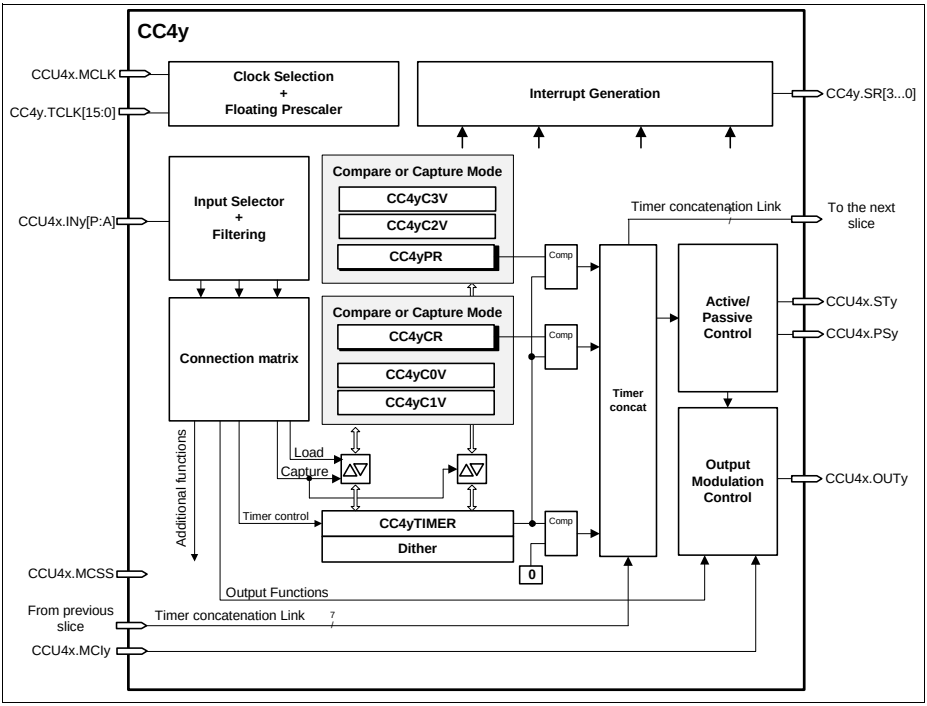


Figure 22-2 CCU4 slice block diagram

Each CCU4 slice, with the exception of the first, contains six dedicated inputs outputs that are used for the built-in timer concatenation functionality.

Inputs and outputs that are not seen at the CCU4 boundaries have a nomenclature of CC4y.<name>, whilst CCU4 module inputs and outputs are described as CCU4x.<signal_name>y (indicating the variable y the object slice).

Table 22-3 CCU4 slice pin description

Pin	I/O	Description
CCU4x.MCLK	I	Module clock
CC4y.TCLK[15:0]	I	Clocks from the pre scaler
CCU4x.INy[P:A]	I	Slice functional inputs (used to control the functionality throughout slice external events)
CCU4x.MCIy	I	Multi Channel mode input
CCU4x.MCSS	I	Multi Channel shadow transfer trigger
CC4y.SR[3...0]	O	Slice service request lines
CC4x.STy	O	Slice comparison status value
CCU4x.PSy	O	Multi channel pattern update trigger
CCU4x.OUTy	O	Slice dedicated output pin

Note:

3. *The status bit outputs of the Kernel, CCU4x.STy, are extended for one more kernel clock cycle.*
4. *The Service Request signals at the output of the kernel are extended for one more kernel clock cycle.*
5. *The maximum output signal frequency of the CCU4x.STy outputs is module clock divided by 4.*

The slice timer, can count up or down depending on the selected operating mode. A direction flag holds the actual counting direction.

The timer is connected to two stand alone comparators, one for the period match and one for a compare match. The registers used for period match and comparison match can be programmed to serve as capture registers enabling sequential capture capabilities on external events.

In normal edge aligned counting scheme, the counter is cleared to 0000_H each time that matches the period value defined in the period register. In center aligned mode, the counter direction changes from ‘up counting’ to ‘down counting’ after reaching the period

value. Both period and compare registers have an aggregated shadow register, which enables the update of the PWM period and duty cycle on the fly.

A single shot mode is also available, where the counter stops after it reaches the value set in the period register.

The start and stop of the counter can be controlled via software access or by a programmable input pin.

Functions like, load, counting direction (up/down), TRAP and output modulation can also be controlled with external events, see **Section 22.2.3**.

22.2.2 Input Selector

The first unit of the slice input path, is used to select which inputs are used to control the available external functions.

Inside this block the user also has the possibility to perform a low pass filtering of the signals and selecting the active edge(s) or level of the external event, see **Figure 22-3**.

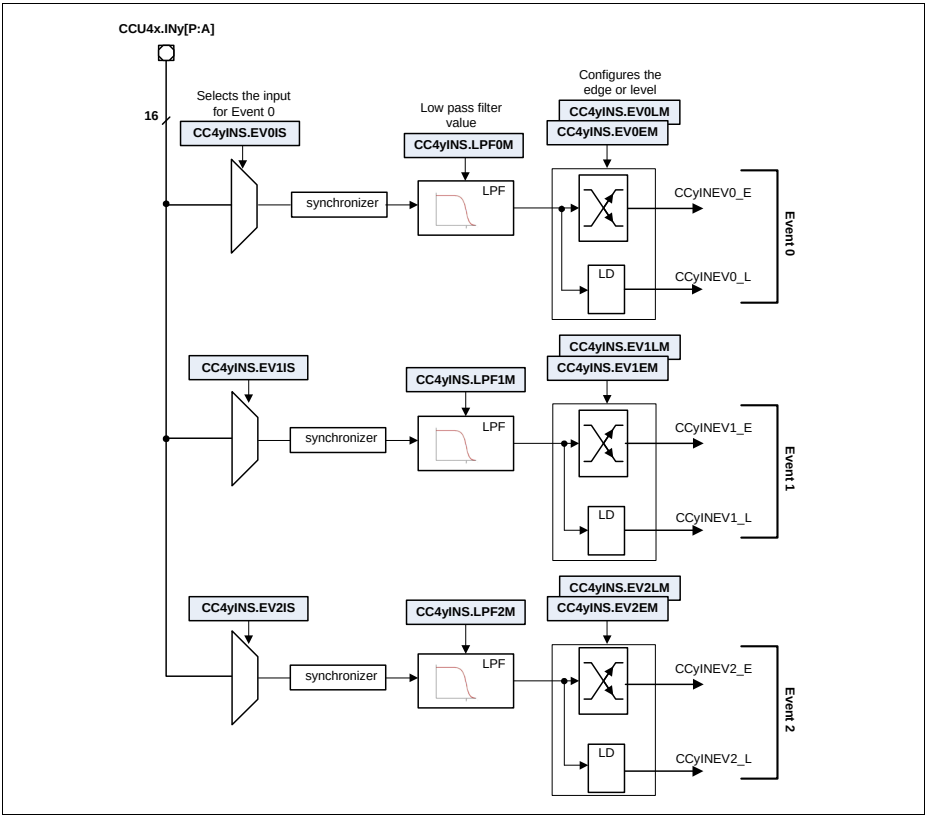


Figure 22-3 Slice input selector diagram

The user has the possibility of selecting any of the CCU4x.INy[P:A] inputs as the source of an event.

At the output of this unit we have a user selection of three events, that were configured to be active at rising, falling or both edges, or level active. These selected events can then be mapped to several functions.

Notice that each decoded event contains two outputs, one edge active and one level active, due to the fact that some functions like counting, capture or load are edge sensitive events while, timer gating or up down counting selection are level active.

22.2.3 Connection Matrix

The connection matrix maps the events coming from the input selector to several user configured functions, **Figure 22-4**.The following functions can be enabled on the connection matrix:

Table 22-4 Connection matrix available functions

Function	Brief description	Map to figure Figure 22-4
Start	Edge signal to start the timer	CCystrt
Stop	Edge signal to stop the timer	CCystp
Count	Edge signal used for counting events	CCycnt
Up/down	Level signal used to select up or down counting direction	CCyupd
Capture 0	Edge signal that triggers a capture into the capture registers 0 and 1	CCycapt0
Capture 1	Edge signal that triggers a capture into the capture register 2 and 3	CCycapt1
Gate	Level signal used to gate the timer clock	CCygate
Load	Edge signal that loads the timer with the value present at the compare register	CCyload
TRAP	Level signal used for fail-safe operation	CCytrap
Modulation	Level signal used to modulate/clear the output	CCymod
Status bit override	Status bit is going to be overridden with an input value	CCyoval for the value CCyoset for the trigger

Inside the connection matrix we also have a unit that performs the built-in timer concatenation. This concatenation enables a completely synchronized operation between the concatenated slices for timing operations and also for capture and load actions. The timer slice concatenation is done via the **CC4yCMC**.TCE bitfield. For a complete description of the concatenation function, please address **Section 22.2.9**.

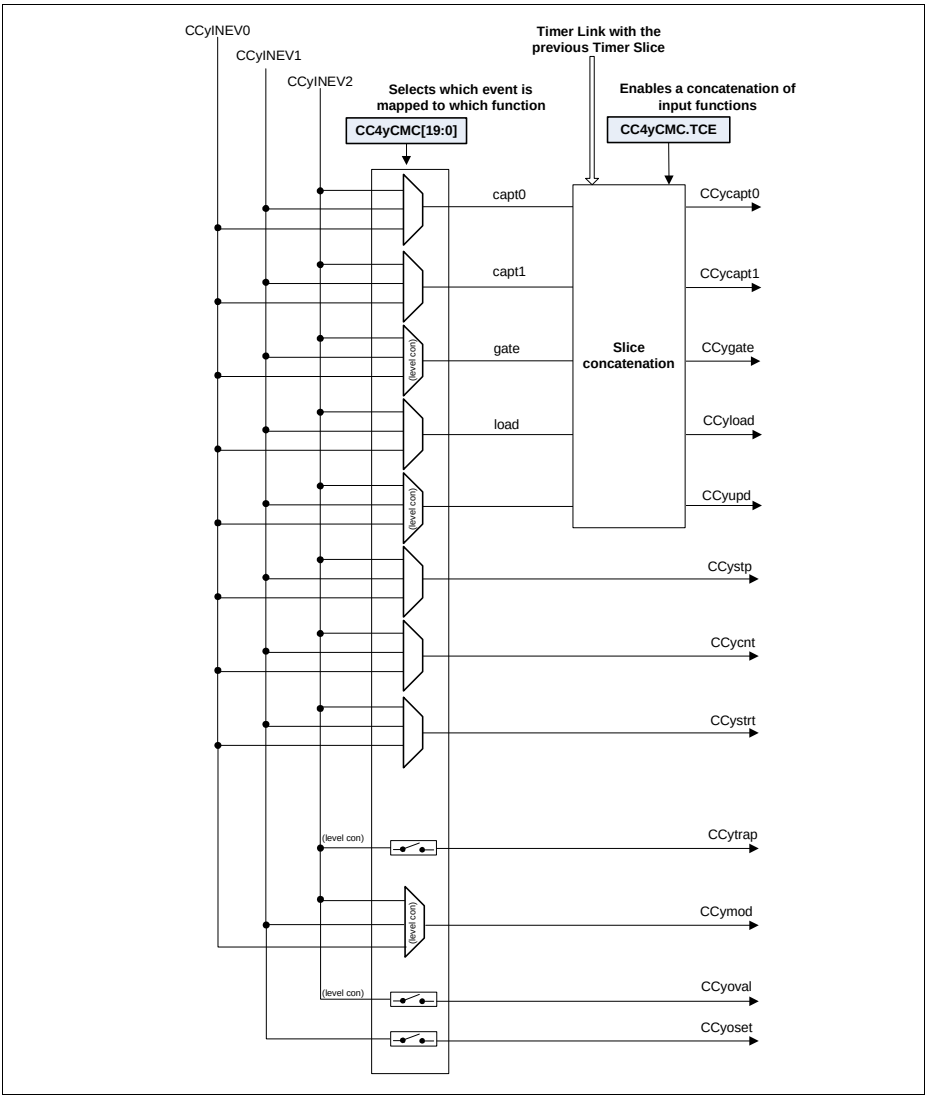


Figure 22-4 Slice connection matrix diagram

22.2.4 Starting/Stopping the Timer

Each timer slice contains a run bit register that indicates the actual status of the timer, **CC4yTCST**.TRB. The start and stop of the timer can be done via software access or can be controlled directly by external events, see **Figure 22-5**.

Selecting an external signal that acts as a start trigger does not force the user to use an external stop trigger and vice versa.

Selecting the single shot mode, imposes that after the counter reaches the period value the run bit, **CC4yTCST**.TRB, is going to be cleared and therefore the timer is stopped.

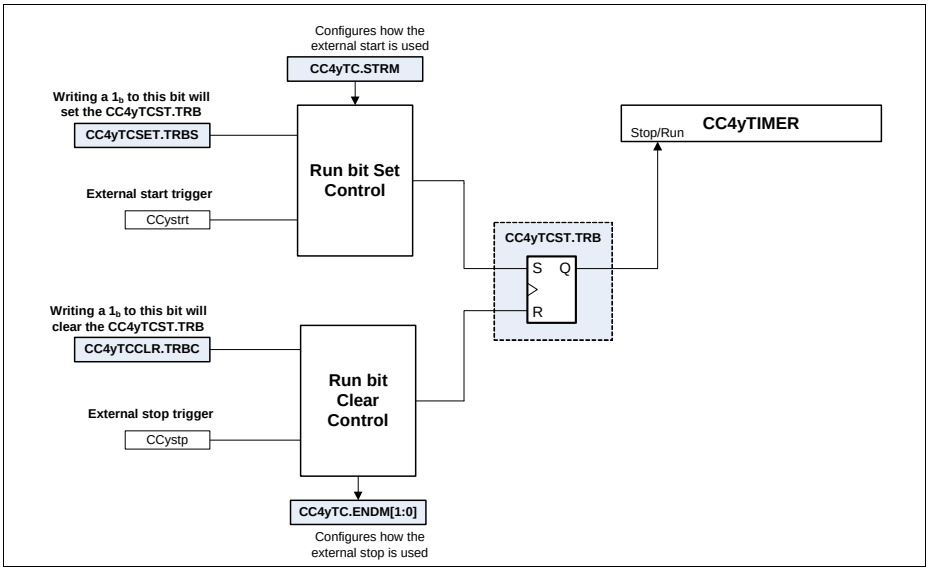


Figure 22-5 Timer start/stop control diagram

One can use the external stop signal to perform the following functions (configuration via **CC4yTC**.ENDM):

- Clear the run bit (stops the timer) - default
- Clear the timer (to 0000_H) but it does not clear the run bit (timer still running)
- Clear the timer and the run bit

One can use the external start to perform the following functions (configuration via **CC4yTC**.STRM):

- Start the timer (resume operation)
- Clear and start the timer

The set (start the timer) of the timer run bit, always has priority over a clear (stop the timer).

22.2.5 Counting Modes

Each CC4y timer slice can be programmed into three different counting schemes:

- Edge aligned (default)
- Center aligned
- Single shot (can be edge or center aligned)

These three counting schemes can be used as stand alone without the need of selecting any inputs as external event sources. Nevertheless it is also possible to control the counting operation via external events like, timer gating, counting trigger, external stop, external start, etc.

For all the counting modes, it is possible to update on the fly the values for the timer period and compare channel. This enables a cycle by cycle update of the PWM frequency and duty cycle.

The compare channel of each CC4y Timer Slice, has an associated Status Bit (**GCST**.CC4yST), that indicates the active or passive state of the channel, **Figure 22-6**. The set and clear of the status bit and the respective PWM signal generation is dictated by the timer period, compare value and the current counting mode. See the different counting mode descriptions, **Section 22.2.5.3** to **Section 22.2.5.5** to understand how this bit is set and cleared.

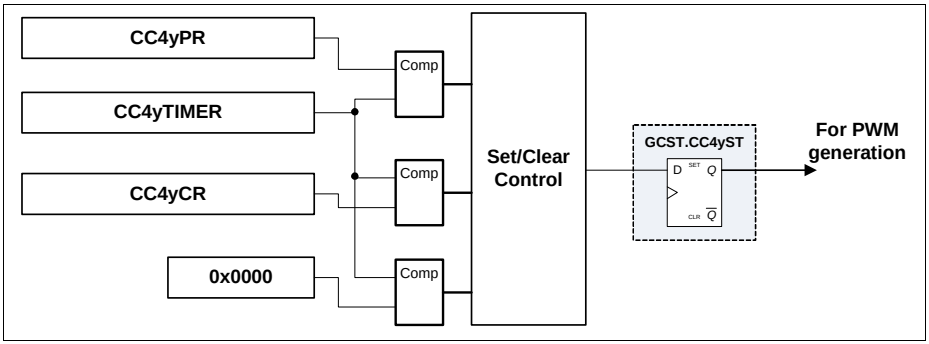


Figure 22-6 CC4y Status Bit

22.2.5.1 Calculating the PWM Period and Duty Cycle

The period of the timer is determined by the value in the period register, **CC4yPR** and by the timer mode.

The base for the PWM signal frequency and duty cycle, is always related to the clock frequency of the timer itself and not to the frequency of the module clock (due to the fact that the timer clock can be a scaled version of the module clock).

In Edge Aligned Mode, the timer period is:

T_{per}= <Period-Value> + 1; in f_{tclk} (22.1)

In Center Aligned Mode, the timer period is:

T_{per}= (<Period-Value> + 1) x 2; in f_{tclk} (22.2)

For each of these counting schemes, the duty cycle of generated PWM signal is dictated by the value programmed into the **CC4yCR** register.

In Edge Aligned and Center Aligned Mode, the PWM duty cycle is:

DC= 1 - <Compare-Value>/(<Period-Value> + 1) (22.3)

Both **CC4yPR** and **CC4yCR** can be updated on the fly via software, enabling a glitch free transition between different period and duty cycle values for the generated PWM signal, **Section 22.2.5.2**

22.2.5.2 Updating the Period and Duty Cycle

Each CCU4 timer slice provides an associated shadow register for the period and compare values. This facilitates a concurrent update by software for these two parameters, with the objective of modifying during run time the PWM signal period and duty cycle.

In addition to the shadow registers for the period and compare values, one also has available shadow registers for the floating prescaler and dither functions, **CC4yFPCS** and **CC4yDITS** respectively (please address **Section 22.2.11** and **Section 22.2.10** for a complete description of these functions).

The structure of the shadow registers can be seen in **Figure 22-7**.

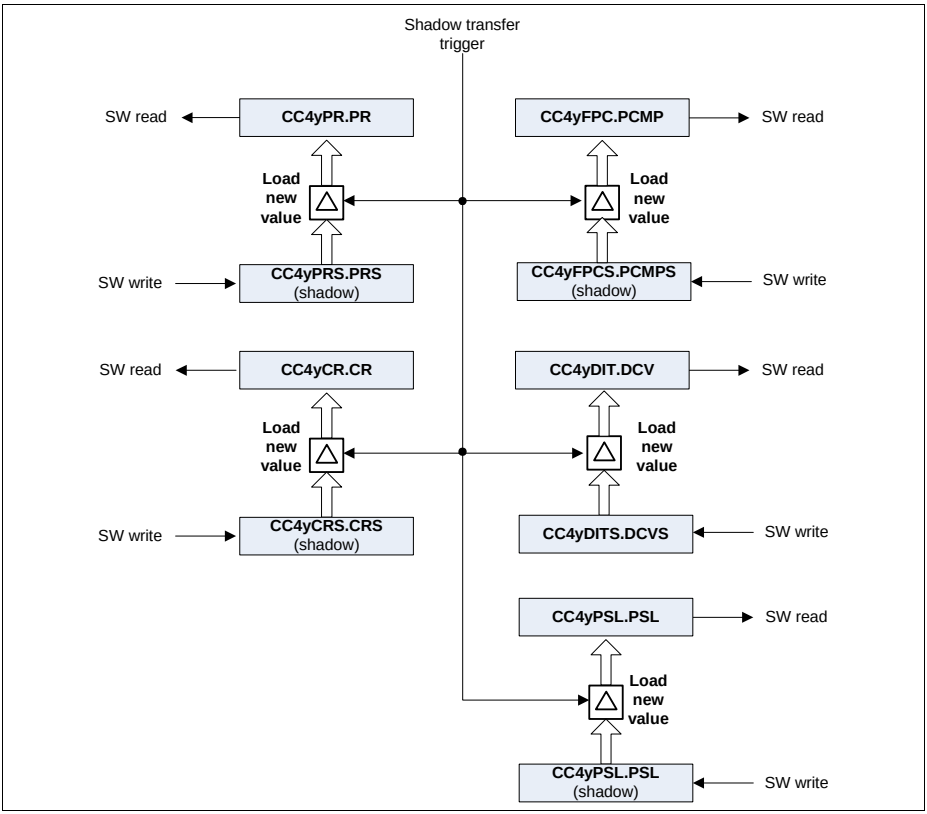


Figure 22-7 Shadow registers overview

The update of these registers can only be done by writing a new value into the associated shadow register and wait for a shadow transfer to occur.

Each group of shadow registers have an individual shadow transfer enable bit, **Figure 22-8**. The software must set this enable bit to 1_B, whenever an update of the values is needed. These bits are automatically cleared by the hardware, whenever an update of the values if finished. Therefore every time that an update of the registers is needed the software must set again the specific bit(s).

Nevertheless it is also possible to clear the enable bit via software. This can be used in the case that an update of the values needs to be cancelled (after the enable bit has already been set).

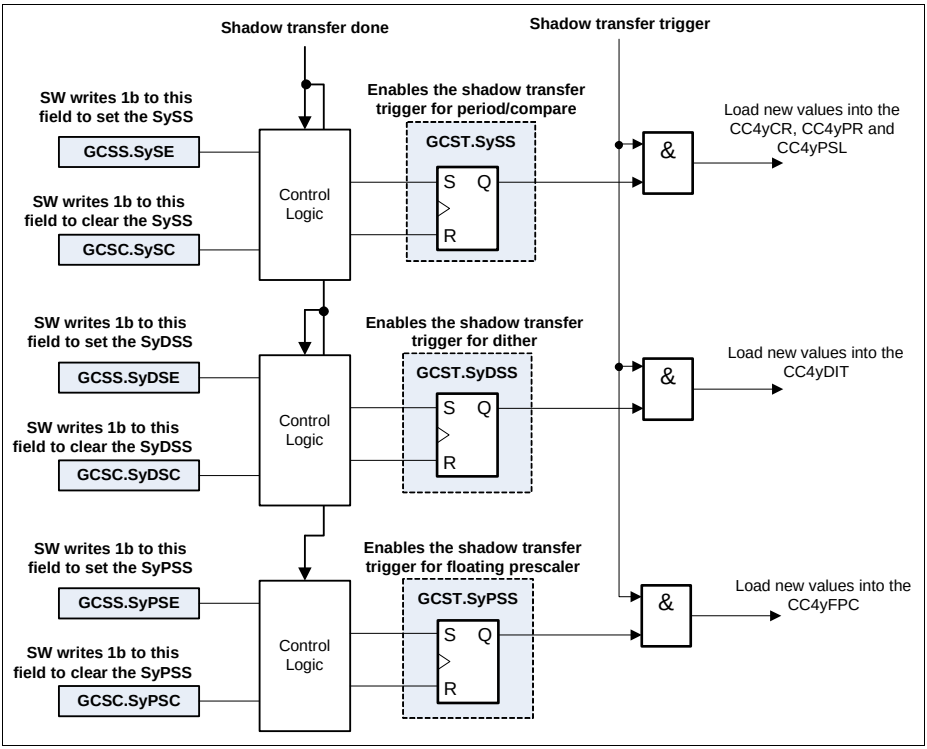


Figure 22-8 Shadow transfer enable logic

The shadow transfer operation is going to be done in the immediately next occurrence of a shadow transfer trigger, after the shadow transfer enable is set (**GCST**.SySS, **GCST**.SyDSS, **GCST**.SyPSS set to 1_B).

The occurrence of the shadow transfer trigger is imposed by the timer counting scheme (edge aligned or center aligned). Therefore the slots when the values are updated can be:

- in the next clock cycle after a Period Match while counting up
- in the next clock cycle after an One Match while counting down
- immediately, if the timer is stopped and the shadow transfer enable bit(s) is set

Figure 22-9 shows an example of the shadow transfer control when the timer slice has been configured into center aligned mode. For a complete description of all the timer slice counting modes, please address **Section 22.2.5.3**, **Section 22.2.5.4** and **Section 22.2.5.5**.

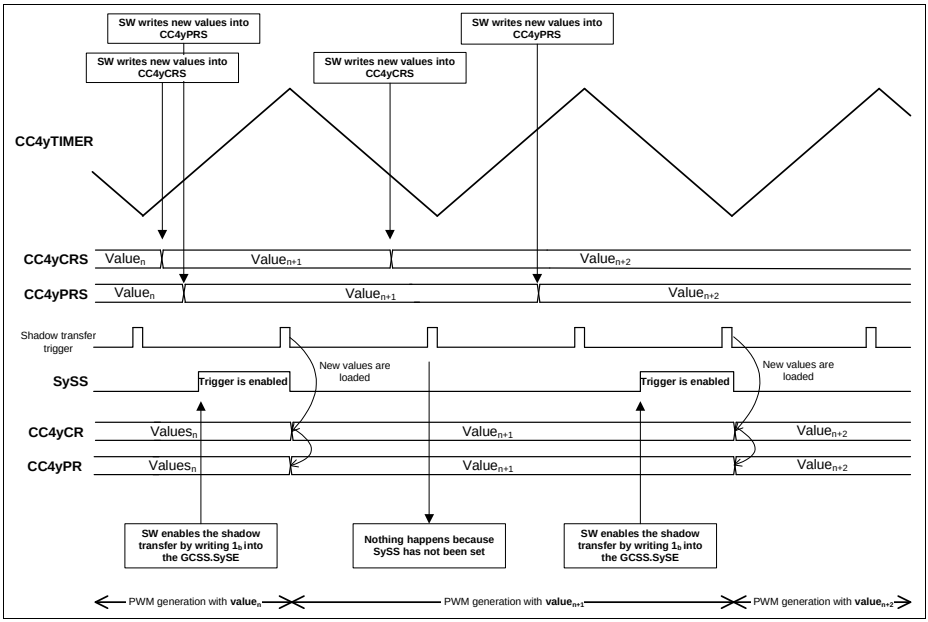


Figure 22-9 Shadow transfer timing example - center aligned mode

When using the CCU4 in conjunction with the POSIF to control the multi channel mode, it can be necessary in some cases, to perform the shadow transfers synchronously with the update of the multi channel pattern. To perform this action, each CCU4 contains a dedicated input that can be used to synchronize the two events, the CCU4x.MCSS.

This input, when enabled, is used to set the shadow transfer enable bitfields (**GCST**.SySS, **GCST**.SyDSS and **GCST**.SyPSS) of the specific slice. It is possible to select which slice is using this input to perform the synchronization via the **GCTRL**.MSEy bit field. It is also possible to enable the usage of this signal for the three different shadow transfer signals: compare and period values, dither compare value and prescaler compare value. This can be configured on the **GCTRL**.MSDE field.

The structure for using the CCU4x.MCSS input signal can be seen in **Figure 22-8**. The usage of this signal is just an add on to the shadow transfer control and therefore all the previous described functions are still available.

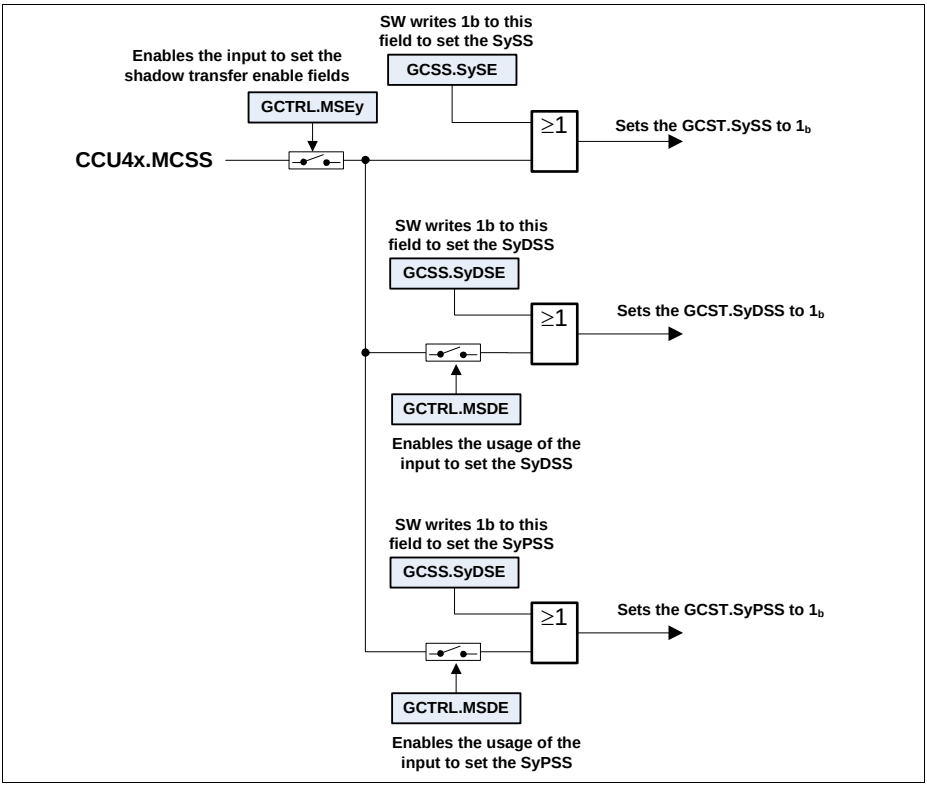


Figure 22-10 Usage of the CCU4x.MCSS input

22.2.5.3 Edge Aligned Mode

Edge aligned mode is the default counting scheme. In this mode, the timer is incremented until it matches the value programmed in the period register, **CC4yPR**. When period match is detected the timer is cleared to 0000_H and continues to be incremented.

In this mode, the value of the period register and compare register are updated with the value written by software into the correspondent shadow register, every time that an overflow occurs (period match), see **Figure 22-11**.

In edge aligned mode, the status bit of the comparison (CC4yST) is set one clock cycle after the timer hits the value programmed into the compare register. The clear of the status bit is done one clock cycle after the timer reaches 0000_H.

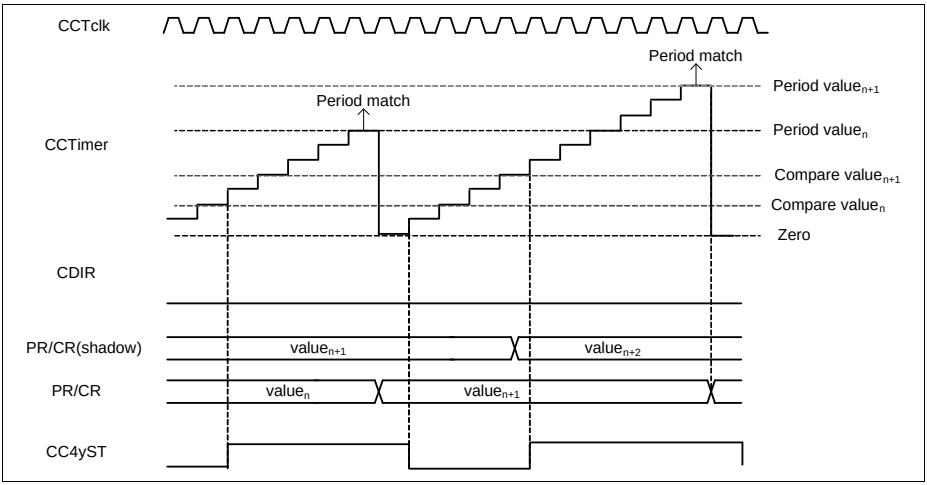


Figure 22-11 Edge aligned mode, CC4yTC.TCM = 0_B

22.2.5.4 Center Aligned Mode

In center aligned mode, the timer is counting up or down with respect to the following rules:

- The counter counts up while **CC4yTCST**.CDIR = 0_B and it counts down while **CC4yTCST**.CDIR = 1_B.
- Within the next clock cycle, the count direction is set to counting up (**CC4yTCST**.CDIR = 0_B) when the counter reaches 0001_H while counting down.
- Within the next clock cycle, the count direction is set to counting down (**CC4yTCST**.CDIR = 1_B), when the period match is detected while counting up.

The status bit (CC4yST) is always 1_B when the counter value is equal or greater than the compare value and 0_B otherwise.

While in edge aligned mode, the shadow transfer for compare and period registers is executed once per period. It is executed twice in center aligned mode as follows

- Within the next clock cycle after the counter reaches the period value, while counting up (**CC4yTCST**.CDIR = 0_B).
- Within the next clock cycle after the counter reaches 0001_H, while counting down (**CC4yTCST**.CDIR = 1_B).

*Note: Bit **CC4yTCST**.CDIR changes within the next timer clock after the one-match or the period-match, which means that the timer continues counting in the previous direction for one more cycle before changing the direction.*

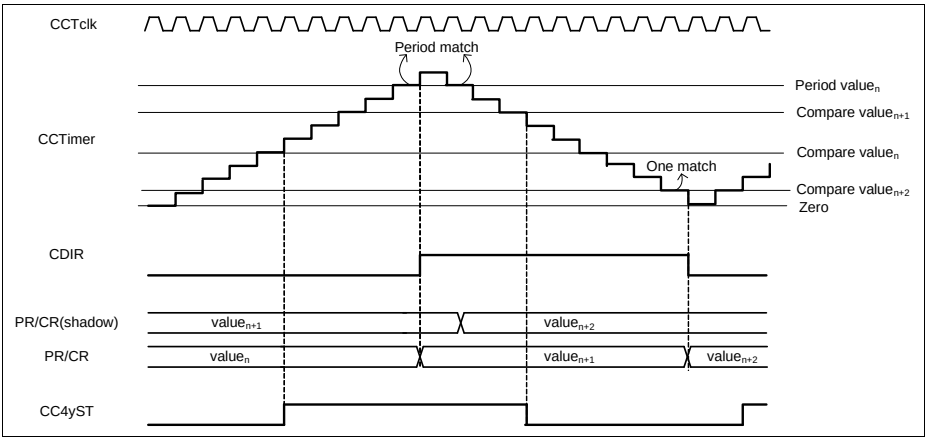


Figure 22-12 Center aligned mode, CC4yTC.TCM = 1_B

22.2.5.5 Single Shot Mode

In single shot mode, the timer is stopped after the current timer period is finished. This mode can be used with center or edge aligned scheme.

In edge aligned mode, **Figure 22-13**, the timer is stopped when it is cleared to 0000_H after having reached the period value. In center aligned mode, **Figure 22-14**, the period is finished when the timer has counted down to 0000_H.

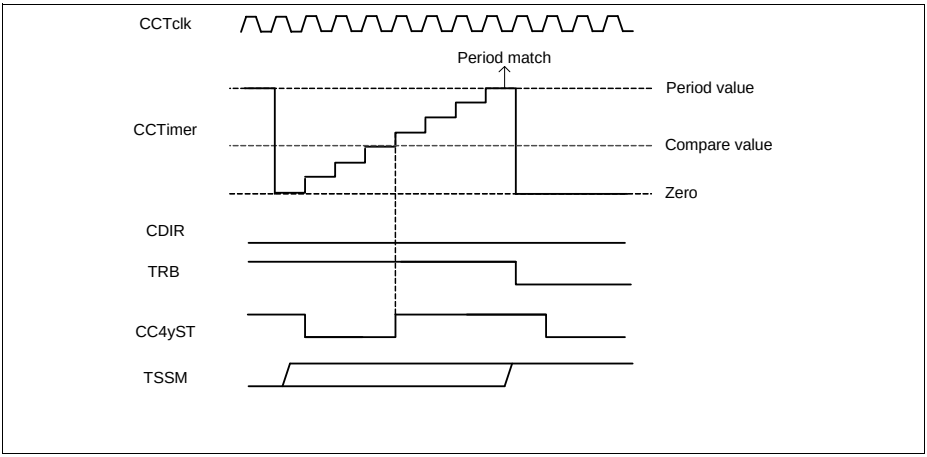


Figure 22-13 Single shot edge aligned - CC4yTC.TSSM = 1_B, CC4yTC.TCM = 0_B

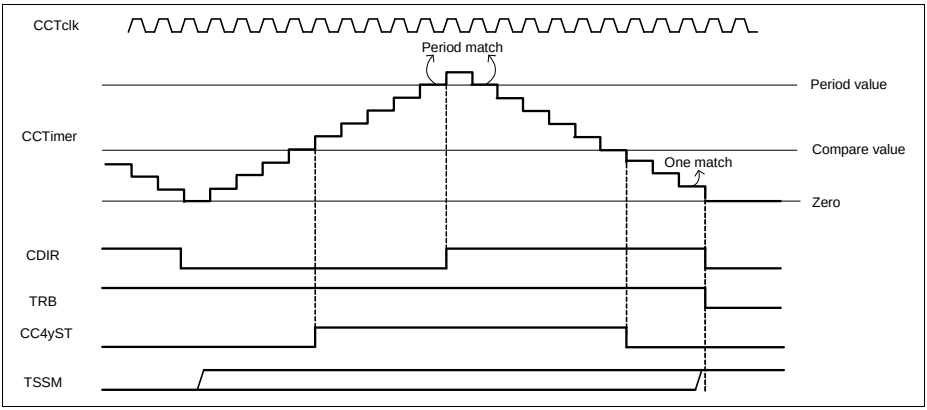


Figure 22-14 Single shot center aligned - CC4yTC.TSSM = 1_B, CC4yTC.TCM = 1_B

22.2.6 Active/Passive Rules

The general rules that set or clear the associated timer slice status bit (CC4yST), can be generalized independently of the timer counting mode.

The following events set the Status bit (CC4yST) to Active:

- in the next f_{tclk} cycle after a compare match while counting up
- in the next f_{tclk} cycle after a zero match while counting down

The following events set the Status bit (CC4yST) to Inactive:

- in the next f_{tclk} cycle after a zero match (and not compare match) while counting up
- in the next f_{tclk} cycle after a compare match while counting down

If external events are being used to control the timer operation, these rules are still applicable.

The status bit state can only be ‘override’ via software or by the external status bit override function, **Section 22.2.7.9**.

The software can at any time write a 1_B into the **GCSS**.SySTS bitfield, which will set the status bit **GCST**.CC4yST of the specific timer slice. Writing a 1_B into the **GCSC**.SySTC bitfield will clear the specific status bit.

22.2.7 External Events Control

Each CCU4 timer slice has the possibility of using up to three different input events, see **Section 22.2.2**. These three events can then be mapped to Timer Slice functions (the full set of available functions is described at **Section 22.2.3**)

These events can be mapped to any of the CCU4x.INy[P...A] inputs and there isn’t any imposition that an event cannot be used to perform several functions, or that an input

cannot be mapped to several events (e.g. input X triggers event 0 with rising edge and triggers event 1 with the falling edge).

22.2.7.1 External Start/Stop

To select an external start function, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC4yINS**.EVxIS field and indicating the active edge of the signal on the **CC4yINS**.EVxEM field.

This event should be then mapped to the start or stop functionality by setting the **CC4yCMC**.STRTS (for the start) or the **CC4yTC**.ENDM (for the stop) with the proper value.

Notice that both start and stop functions are edge and not level active and therefore the active/passive configuration is set only by the **CC4yINS**.EVxEM.

The external stop by default just clears the run bit (**CC4yTCST**.TRB), while the start functions does the opposite. Nevertheless one can select an extended subset of functions for the external start and stop. This subset is controlled by the registers **CC4yTC**.ENDM (for the stop) and **CC4yTC**.STRM (for the start).

For the start subset (**CC4yTC**.STRM):

- sets the run bit/starts the timer (resume operation)
- clears the timer, sets the run bit/starts the timer (flush and start)

For the stop subset (**CC4yTC**.ENDM):

- clears the run/stops the timer (stop)
- clears the timer (flush)
- clears the timer, clears the run bit/stops the timer (flush and stop)

If in conjunction with an external start/stop (configured also/only as flush) and external up/down signal is used, during the flush operation the timer is going to be set to 0000_H if the actual counting direction is up or set with the value of the period register if the counting direction is down.

Figure 22-15 to **Figure 22-18** shows the usage of two signals to perform the start/stop functions in all the previously mentioned subsets. External Signal(1) acts as an active HIGH start signal, while External Signal(2) is used as an active HIGH stop function.

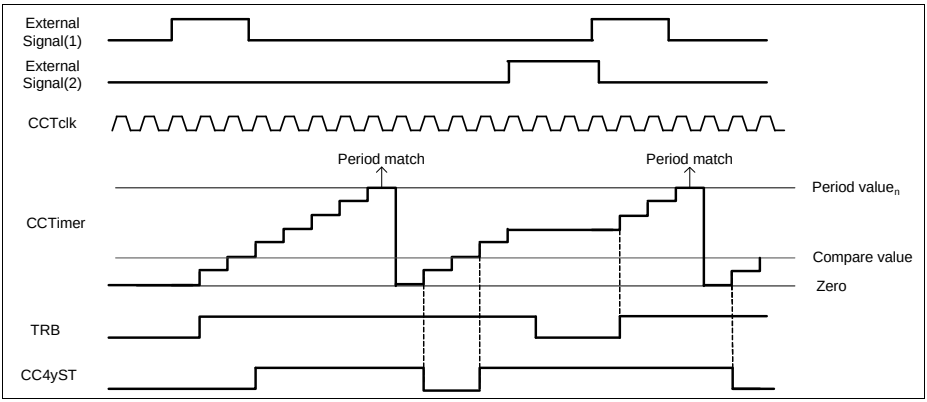


Figure 22-15 Start (as start)/ stop (as stop) - CC4yTC.STRM = 0_B, CC4yTC.ENDM = 00_B

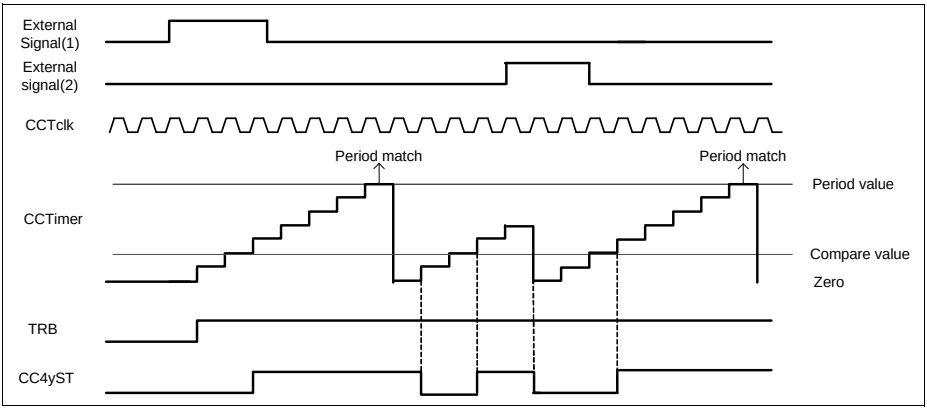


Figure 22-16 Start (as start)/ stop (as flush) - CC4yTC.STRM = 0_B, CC4yTC.ENDM = 01_B

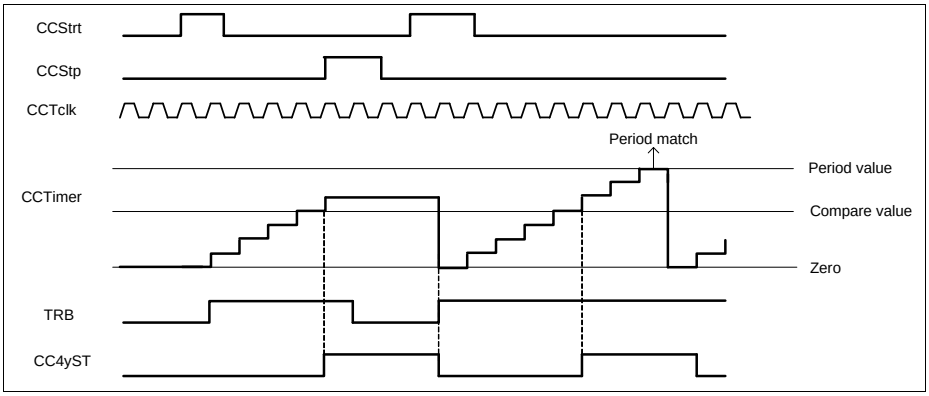


Figure 22-17 Start (as flush and start)/ stop (as stop) - CC4yTC.STRM = 1_B, CC4yTC.ENDM = 00_B

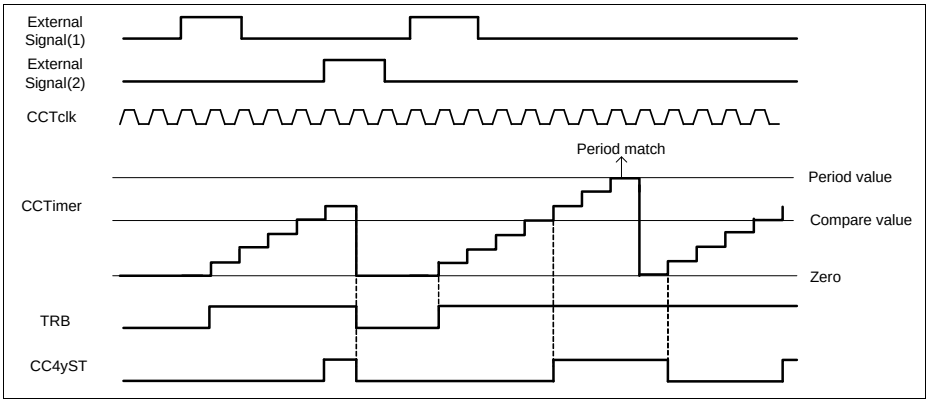


Figure 22-18 Start (as start)/ stop (as flush and stop) - CC4yTC.STRM = 0_B, CC4yTC.ENDM = 10_B

22.2.7.2 External Counting Direction

There is the possibility of selecting an input signal to act as increment/decrement control.

To select an external up/down control, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC4yINS**.EVxIS field and indicating the active edge of the signal on the **CC4yINS**.EVxEM. This event should be then mapped to the up/down functionality by setting **CC4yCMC**.UDS with the proper value.

Notice that the up/down function is level active and therefore the active/passive configuration is set only by the **CC4yINS**.EVxLM.

The status bit of the slice (CC4yST) is always set when the timer value is equal or greater than the value stored in the compare register, see **Section 22.2.6**.

The update of the period and compare register values is done when:

- with the next clock after a period match, while counting up (**CC4yTCST**.CDIR = 0_B)
- with the next clock after a one match, while counting down (**CC4yTCST**.CDIR = 1_B)

The value of the **CC4yTCST**.CDIR register is updated accordingly with the changes on the decoded event. The Up/Down direction is always understood as **CC4yTCST**.CDIR = 1_B when counting down and **CC4yTCST**.CDIR = 0_B when counting up. Using an external signal to perform the up/down counting function and configuring the event as active HIGH means that the timer is counting up when the signal is HIGH and counting down when LOW.

Figure 22-19 shows an external signal being used to control the counting direction of the time. This signal was selected as active HIGH, which means that the timer is counting down while the signal is HIGH and counting up when the signal is LOW.

*Note: For a signal that should impose an increment when LOW and a decrement when HIGH, the user needs to set the **CC4yINS**.EVxLM = 0_B. When the operation is switched, then the user should set **CC4yINS**.EVxLM = 1_B.*

Note: Using an external counting direction control, sets the slice in edge aligned mode.

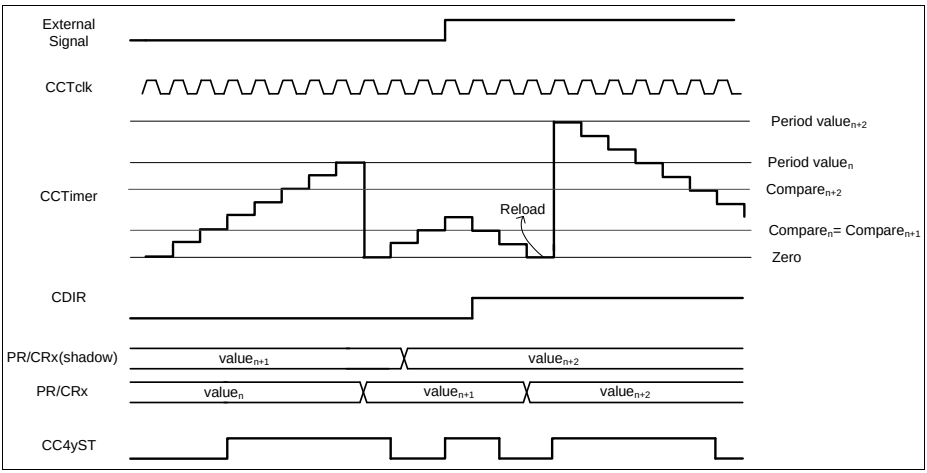


Figure 22-19 External counting direction

22.2.7.3 External Gating Signal

For pulse measurement, the user has the possibility of selecting an input signal that operates as counting gating.

To select an external gating control, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC4yINS**.EVxIS register and indicating the active level of the signal on the **CC4yINS**.EVxLM register. This event should be then mapped to the up/down functionality by setting the **CC4yCMC**.GATES with the proper value.

Notice that the gating function is level active and therefore the active/passive configuration is set only by the **CC4yINS**.EVxLM.

The status bit during an external gating signal continues to be asserted when the compare value is reached and deasserted when the counter reaches 0000_H. One should note that the counter continues to use the period register to identify the wrap around condition. **Figure 22-20** shows the usage of an external signal for gating the slice counter. The signal was set as active LOW, which means the counter gating functionality is active when the external value is zero.

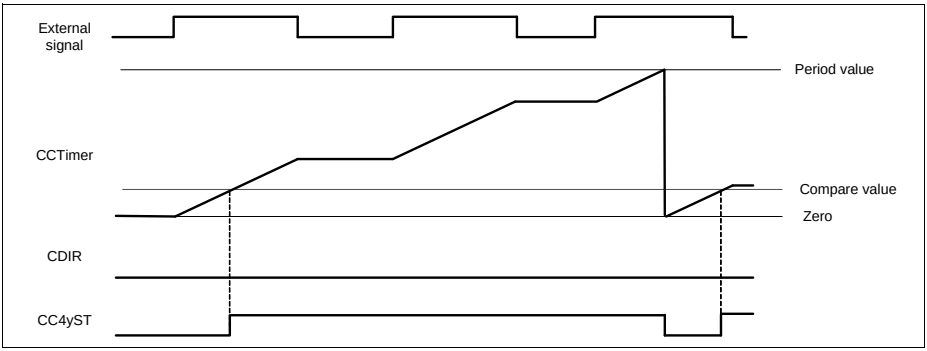


Figure 22-20 External gating

For any type of usage of the external gating function, the specific rung bit of the Timer Slice, **CC4yTCST**.TRB, needs to be set. This can be done via an additional external signal or directly via software.

22.2.7.4 External Count Signal

There is also the possibility of selecting an external signal to act as the counting event.

To select an external counting, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC4yINS**.EVxIS register and indicating the active edge of the signal on the **CC4yINS**.EVxEM register.

This event should be then mapped to the up/down functionality by setting the **CC4yCMC**.CNTS with the proper value.

Notice that the counting function is edge active and therefore the active/passive configuration is set only by the **CC4yINS**.EVxEM.

One can select just a the rising, falling or both edges to perform a count. On **Figure 22-21**, the external signal was selected as a counter event for both falling and rising edges. Wrap around condition is still applied with a comparison with the period register.

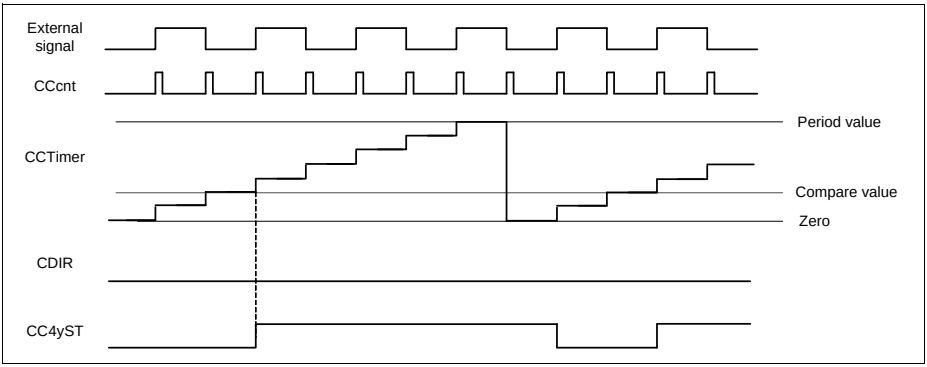


Figure 22-21 External count

For any type of usage of the external gating function, the specific rung bit of the Timer Slice, **CC4yTCST**.TRB, needs to be set. This can be done via an additional external signal or directly via software.

22.2.7.5 External Load

Each slice of the CCU4 also has a functionality that enables the user to select an external signal as trigger for reloading the value of the timer with the current value of the compare register (if **CC4yTCST**.CDIR = 0_B) or with the value of the period register (if **CC4yTCST**.CDIR = 1_B).

To select an external load signal, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC4yINS**.EVxIS register and indicating the active edge of the signal on the **CC4yINS**.EVxEM register. This event should be then mapped to the load functionality by setting the **CC4yCMC**.LDS with the proper value.

Notice that the load function is edge active and therefore the active/passive configuration is set only by the **CC4yINS**.EVxEM.

On figure **Figure 22-22**, the external signal (1) was used to act as a load trigger, active on the rising edge. Every time that a rising edge on external signal (1) is detected, the

timer value is loaded with the value present on the compare register. If an external signal is being used to control the counting direction, up or down, the timer value can be loaded also with the value set in the period register. The External signal (2) represents the counting direction control (active HIGH). If at the moment that a load trigger is detected, the signal controlling the counting direction is imposing a decrement, then the value set in the timer is the period value.

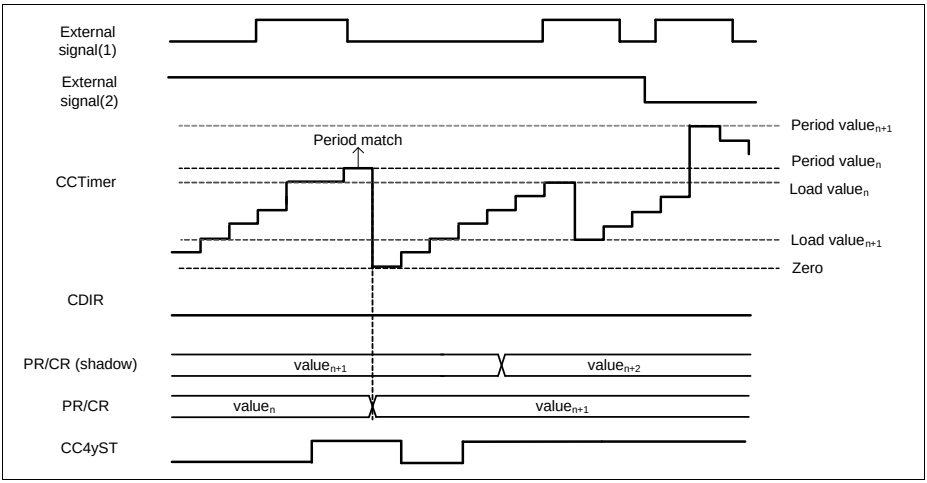


Figure 22-22 External load

22.2.7.6 External Capture

When selecting an external signal to be used as a capture trigger (if **CC4yCMC**.CAP0S or **CC4yCMC**.CAP1S are different from 0_H), the user is automatically setting the specific slice into capture mode.

In capture mode the user can have up to four capture registers, see **Figure 22-25**: capture register 0 (**CC4yC0V**), capture register 1 (**CC4yC1V**), capture register 2 (**CC4yC2V**) and capture register 3 (**CC4yC3V**).

These registers are shared between compare and capture modes which imposes:

- if **CC4yC0V** and **CC4yC1V** are used for capturing, the compare registers **CC4yCR** and **CC4yCRS** are not available (no compare channel)
- if **CC4yC2V** and **CC4yC3V** are used for capturing, the period registers **CC4yPR** and **CC4yPRS** are not available (no period control)

To select an external capture signal, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC4yINS**.EVxIS register and indicating the active edge of the signal on the

CC4yINS.EVxEM register. This event should be then mapped to the capture functionality by setting the **CC4yCMC**.CAP0S/**CC4yCMC**.CAP1S with the proper value.

Notice that the capture function is edge active and therefore the active/passive configuration is set only by the **CC4yINS**.EVxEM.

The user has the possibility of selecting the following capture schemes:

- Different capture events for **CC4yC0V**/**CC4yC1V** and **CC4yC2V**/**CC4yC3V**
- The same capture event for **CC4yC0V**/**CC4yC1V** and **CC4yC2V**/**CC4yC3V** with the same capture edge. For this capture scheme, only the CCcapt1 functionality needs to be programmed. To enable this scheme, the field **CC4yTC**.SCE needs to be set to 1.

Different Capture Events (SCE = 0_B)

Every time that a capture trigger 1 occurs, CCcapt1, the actual value of the timer is captured into the capture register 3 and the previous value stored in this register is transferred into capture register 2.

Every time that a capture trigger 0 occurs, CCcapt0, the actual value of the timer is captured into the capture register 1 and the previous value stored in this register is transferred into capture register 0.

Every time that a capture procedure into one of the registers occurs, the respective full flag is set. This flag is cleared automatically by HW when the SW reads back the value of the capture register (by reading the specific capture register or by reading the extended capture read value, **ECRD**, if **CC4yTC**.ECM = 1_B).

The capture of a new value into a specific capture registers is dictated by the status of the full flag as follows:

$$CC4yC1V_{capt} = \text{NOT}(CC4yC1V_{full\_flag} \text{ AND } CC4yC0V_{full\_flag}) \quad (22.4)$$

$$CC4yC0V_{capt} = CC4yC1V_{full\_flag} \text{ AND NOT}(CC4yC0V_{full\_flag}) \quad (22.5)$$

It is also possible to disable the effect of the full flags reset by setting the **CC4yTC**.CCS = 1_B. This enables a continuous capturing independent if the values captured have been read or not.

*Note: When using the period registers for capturing, **CC4yCMC**.CAP1S different from 00_B, the counter always uses its full 16 bit width as period value.*

On **Figure 22-23**, an external signal was selected as an event for capturing the timer value into the **CC4yC0V**/**CC4yC1V** registers. The status bit, CC4yST, during capture mode is asserted whenever a capture trigger is detected and deasserted when the counter matches 0000_H.

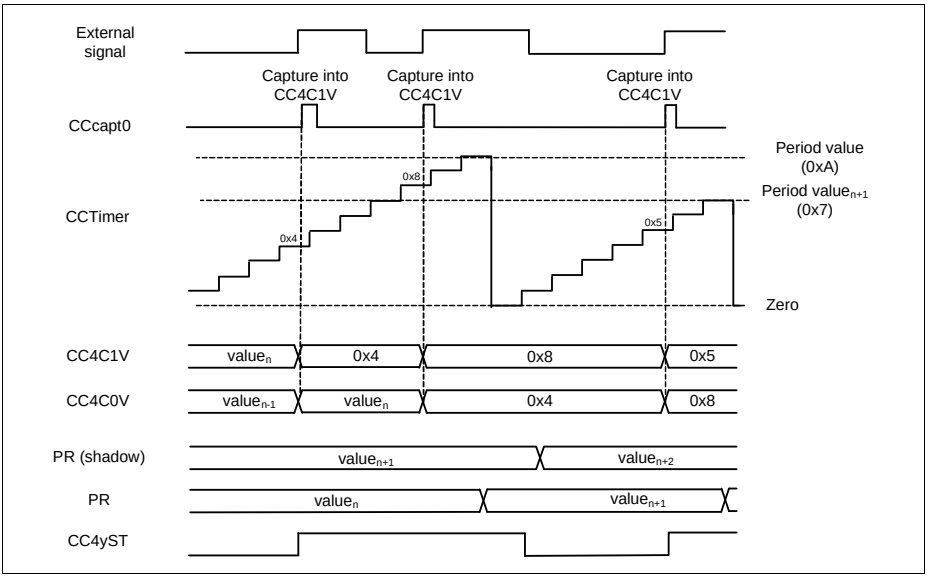


Figure 22-23 External capture - CC4yCMC.CAP0S != 00_B, CC4yCMC.CAP1S = 00_B

On **Figure 22-24**, two different signals were used as source for capturing the timer value into the **CC4yC0V**/**CC4yC1V** and **CC4yC2V**/**CC4yC3V** registers.

External signal(1) was selected as rising edge active capture source for **CC4yC0V**/**CC4yC1V**. External signal(2) was selected has the capture source for **CC4yC2V**/**CC4yC3V**, but as opposite to the external signal(1), the active edge was selected has falling.

See **Section 22.2.12.4**, for the complete capture mode usage description.

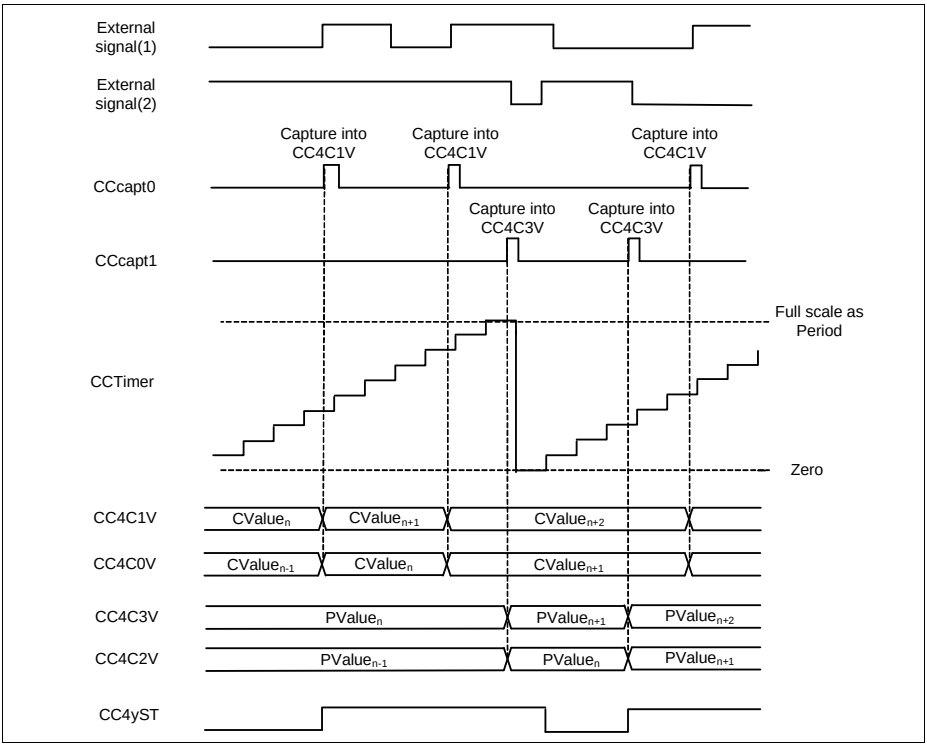


Figure 22-24 External capture - CC4yCMC.CAP0S != 00_B, CC4yCMC.CAP1S != 00_B

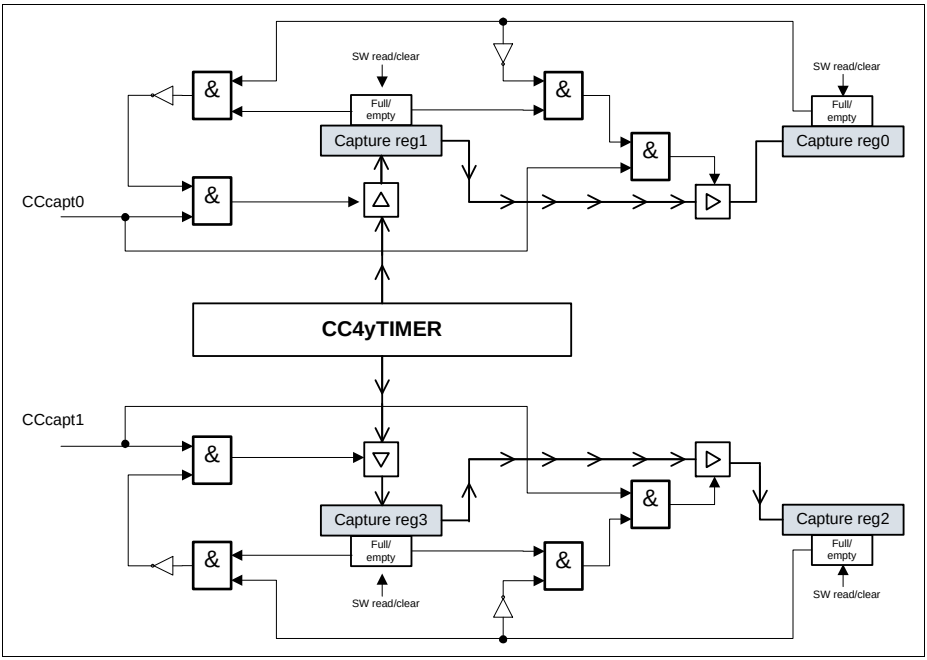


Figure 22-25 Slice capture logic

Same Capture Event (SCE = 1_B)

Setting the field **CC4yTC**.SCE = 1_B, enables the possibility of having 4 capture registers linked with the same capture event, **Figure 22-27**. The function that controls the capture is the CCcapt1.

The capture logic follows the same structure shown in **Figure 22-25** but extended to a four register chain, see **Figure 22-26**. The same full flag lock rules are applied to the four register chain (it also can be disabled by setting the **CC4yTC**.CCS = 1_B):

$$CC4yC3V_{capt} = \text{NOT}(CC4yC3V_{full\_flag} \text{ AND } CC4yC2V_{full\_flag} \text{ AND } CC4yC2V_{full\_flag} \text{ AND } CC4yC1V_{full\_flag}) \quad (22.6)$$

$$CC4yC2V_{capt} = CC4yC3V_{full\_flag} \text{ AND NOT}(CC4yC2V_{full\_flag} \text{ AND } CC4yC1V_{full\_flag} \text{ AND } CC4yC0V_{full\_flag}) \quad (22.7)$$

$$CC4yC1V_{capt} = CC4yC2V_{full\_flag} \text{ AND NOT}(CC4yC1V_{full\_flag} \text{ AND } CC4yC0V_{full\_flag}) \quad (22.8)$$

$$CC4yC0V_{capt} = CC4yC1V_{full\_flag} \text{ AND NOT}(CC4yC0V_{full\_flag}) \quad (22.9)$$

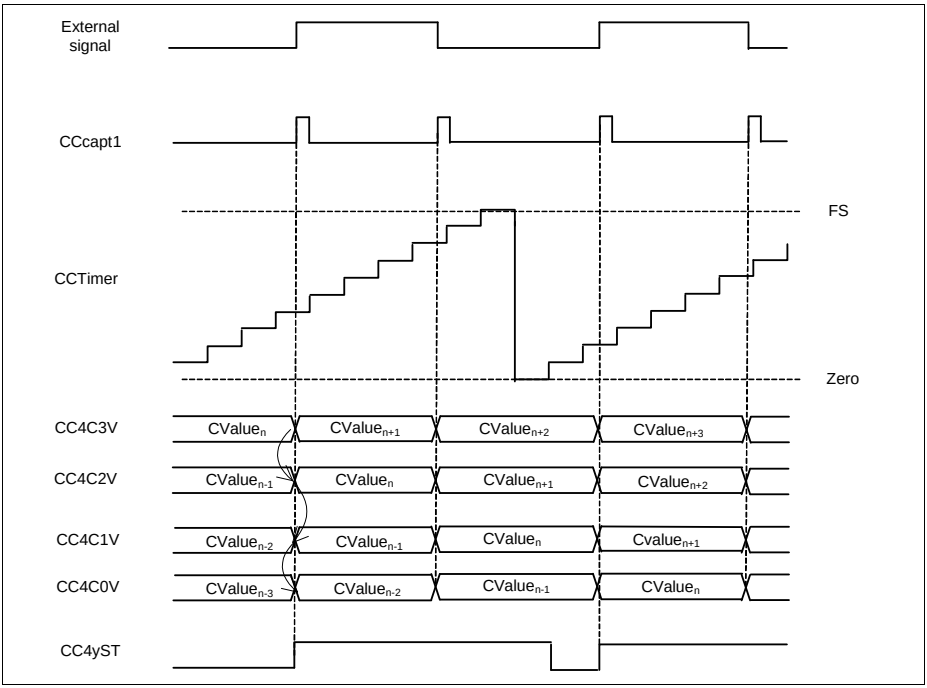


Figure 22-26 External Capture - CC4yTC.SCE = 1_B

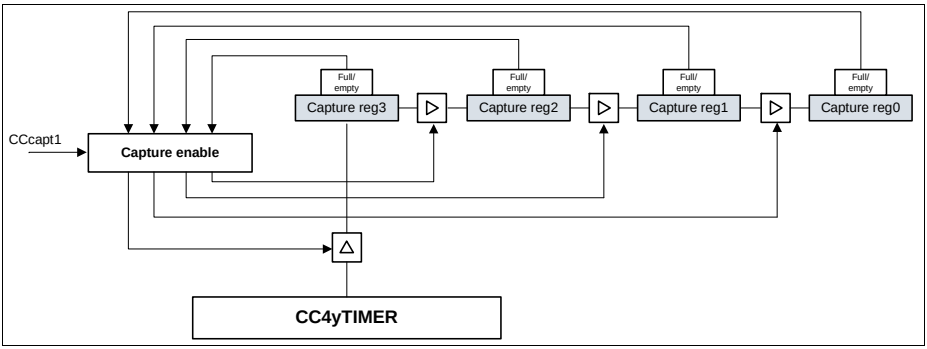


Figure 22-27 Slice Capture Logic - CC4yTC.SCE = 1_B

22.2.7.7 External Modulation

An external signal can be used to perform a modulation at the output of each timer slice.

To select an external modulation signal, one should map one of the input signals to one of the events, by setting the required value in the **CC4yINS**.EVxIS register and indicating the active level of the signal on the **CC4yINS**.EVxLM register. This event should be then mapped to the modulation functionality by setting the **CC4yCMC**.MOS = 01_B if event 0 is being used, **CC4yCMC**.MOS = 10_B if event 1 or **CC4yCMC**.MOS = 11_B if event 2.

Notice that the modulation function is level active and therefore the active/passive configuration is set only by the **CC4yINS**.EVxLM.

The modulation has two modes of operation:

- modulation event is used to clear the CC4yST bit - **CC4yTC**.EMT = 0_B
- modulation event is used to gate the outputs - **CC4yTC**.EMT = 1_B

On **Figure 22-28**, we have a external signal configured to act as modulation source that clears the CC4yST bit, **CC4yTC**.EMT = 0_B. It was programmed to be an active LOW event and therefore, when this signal is LOW the output value follows the normal ACTIVE/PASSIVE rules.

When the signal is HIGH (inactive state), then the CC4yST bit is cleared and the output is forced into the PASSIVE state. Notice that the values of the status bit, CC4yST and the specific output CCU4x.OUTy are not linked together. One can choose for the output to be active LOW or HIGH through the PSL bit.

The exit of the external modulation inactive state is synchronized with the PWM signal due to the fact that the CC4yST bit is cleared and cannot be set while the modulation signal is inactive.

The entering into inactive state also can be synchronized with the PWM signal, by setting **CC4yTC**.EMS = 1_B. With this all possible glitches at the output are avoided, see **Figure 22-29**.

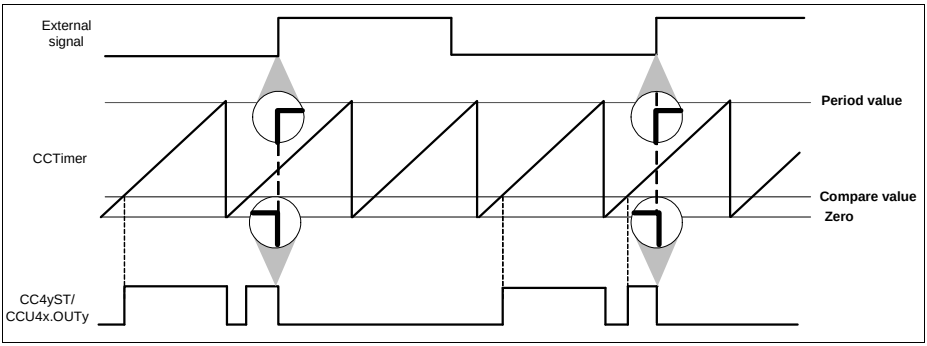


Figure 22-28 External modulation clearing the ST bit - CC4yTC.EMT = 0_B

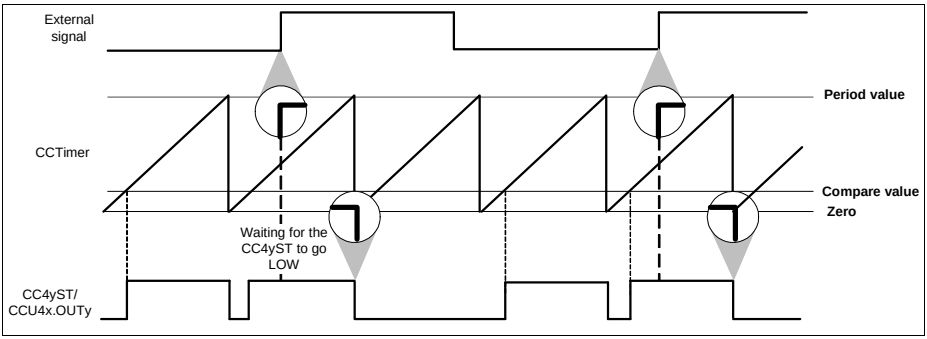


Figure 22-29 External modulation clearing the ST bit - CC4yTC.EMT = 0_B, CC4yTC.EMS = 1_B

On **Figure 22-30**, the external modulation event was used as gating signal of the outputs, **CC4yTC**.EMT = 1_B. The external signal was configured to be active HIGH, **CC4yINS**.EVxLM = 0_B, which means that when the external signal is HIGH the outputs are set to the PASSIVE state.In this mode, the gating event can also be synchronized with the PWM signal by setting the **CC4yTC**.EMS = 1_B.

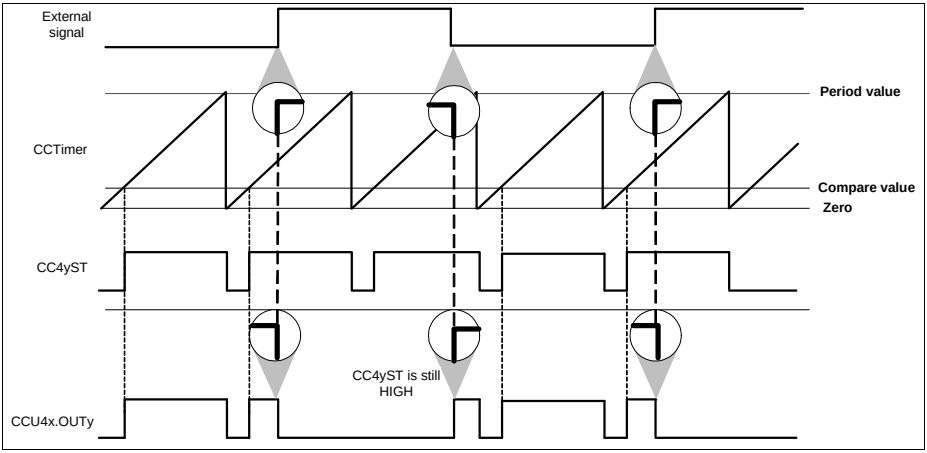


Figure 22-30 External modulation gating the output - CC4yTC.EMT = 1_B

22.2.7.8 TRAP Function

The TRAP functionality allows the PWM outputs to react on the state of an input pin. This functionality can be used to switch off the power devices if the TRAP input becomes active.

To select the TRAP functionality, one should map one of the input signals to event number 2, by setting the required value in the **CC4yINS**.EV2IS register and indicating the active level of the signal on the **CC4yINS**.EV2LM register. This event should be then mapped to the trap functionality by setting the **CC4yCMC**.TS = 1_B.

Notice that the trap function is level active and therefore the active/passive configuration is set only by the **CC4yINTS**.EV2LM.

There are two bitfields that can be monitored via software to crosscheck the TRAP function, **Figure 22-31**:

- The TRAP state bit, **CC4yINTS**.E2AS. This bitfield if the TRAP is currently active or not. This bitfield is therefore setting the specific Timer Slice output, into ACTIVE or PASSIVE state.
- The TRAP Flag, **CC4yINTS**.TRPF. This bitfield is used as a remainder in the case that the TRAP condition is cleared automatically via hardware. This field needs to be cleared by the software.

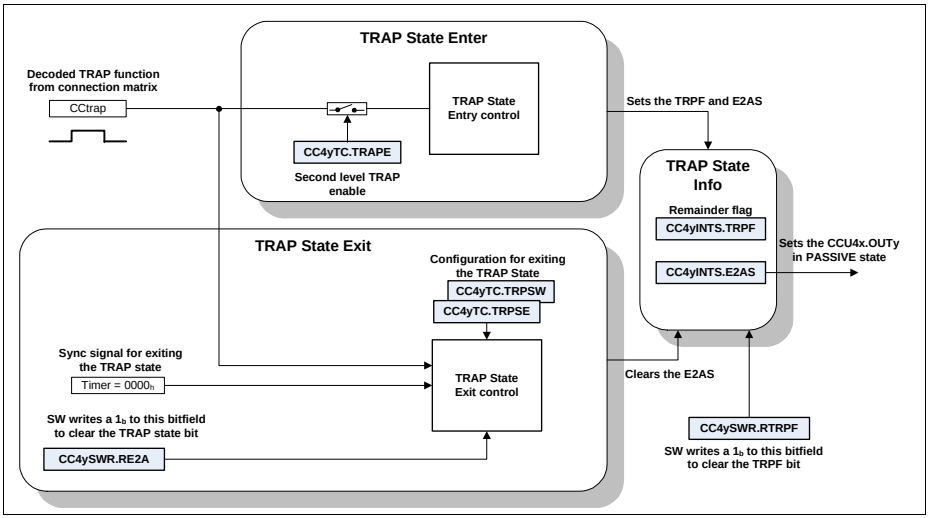


Figure 22-31 Trap control diagram

When a TRAP condition is detected at the selected input pin, both the Trap Flag and the Trap State bit are set to 1_B. The Trap State is entered immediately, by setting the CCU8xOUTy into the programmed PASSIVE state, **Figure 22-32**.

Exiting the Trap State can be done in two ways (**CC4yTC**.TRPSW register):

- automatically via HW, when the TRAP signal becomes inactive - **CC4yTC**.TRPSW = 0_B
- by SW only, by clearing the **CC4yINTS**.E2AS.The clearing is only possible if the input TRAP signal is in inactive state - **CC4yTC**.TRPSW = 1_B

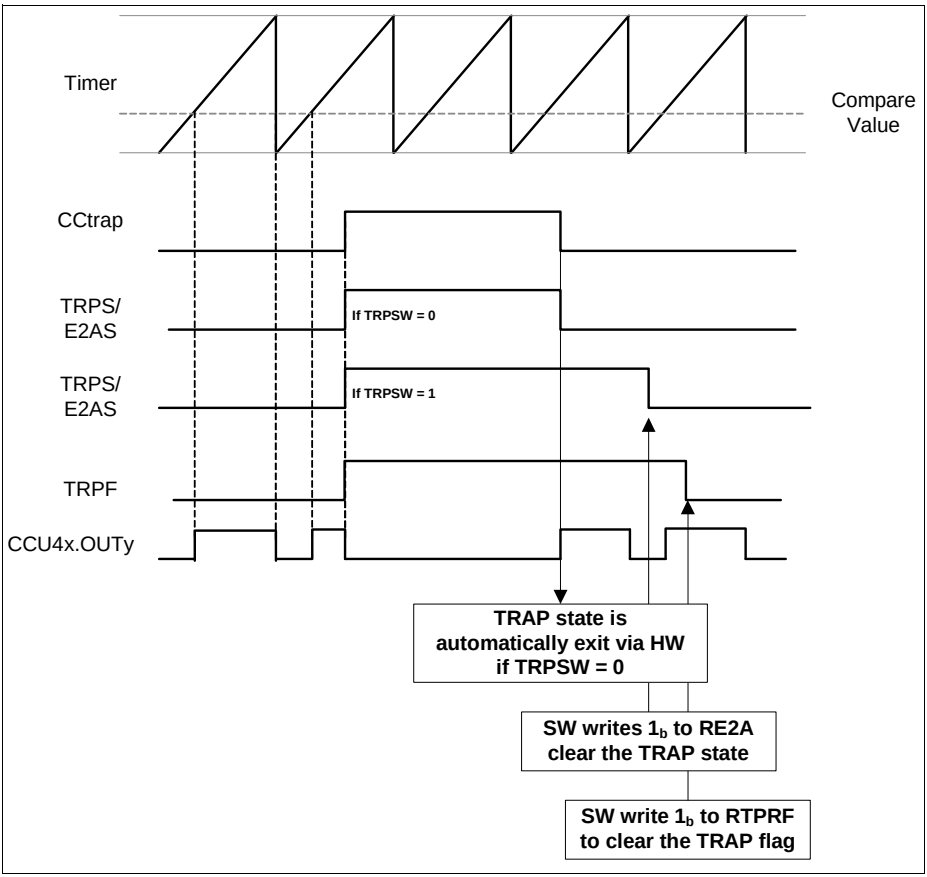


Figure 22-32 Trap timing diagram, CC4yPSL.PSL = 0_B (output passive level is 0_B)

It is also possible to synchronize the exiting of the TRAP state with the PWM signal, **Figure 22-33**. This function is enabled when the bitfield **CC4yTC**.TRPSE = 1_B.

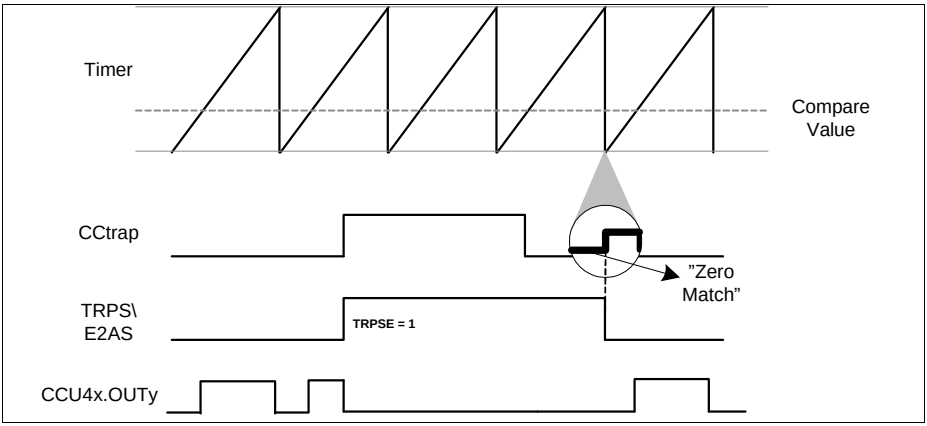


Figure 22-33 Trap synchronization with the PWM signal, CC4yTC.TRPSE = 1_B

22.2.7.9 Status Bit Override

For complex timed output control, each Timer Slice has a functionality that enables the override of the status bit (CC4yST) with a value passed trough an external signal.

The override of the status bit, can then lead to a change on the output pin, CCU8xOUTy (from inactive to active or vice versa).

To enable this functionality, two signals are needed:

- One signal that acts as a trigger to override the status bit (edge active)
- One signal that contains the value to be set in the status bit (level active)

To use the status bit override functionality, one should map the signal that acts as trigger to the event number 1, by setting the required value in the **CC4yINS**.EV1IS register and indicating the active edge of the signal on the **CC4yINS**.EV1EM register.

The signal that carries the value to be set on the status bit, needs to be mapped to the event number 2, by setting the required value in the **CC4yINS**.EV2IS register. The **CC4yINS**.EV2LM register should be set to 0_B if no inversion on the signal is needed and to 1_B otherwise.

The events should be then mapped to the status bit functionality by setting the **CC4yCMC**.OFS = 1_B.

Figure 22-34 shows the functionality of the status bit override, when the external signal(1) was selected as trigger source (rising edge active) and the external signal(2) was selected as override value.

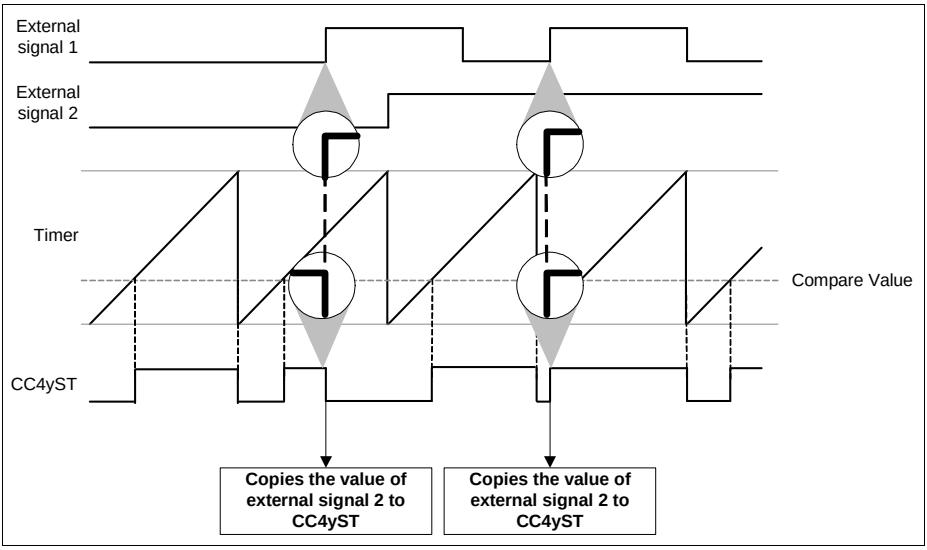


Figure 22-34 Status bit override

22.2.8 Multi-Channel Control

The multi channel control mode is selected individually in each slice by setting the **CC4yTC**.MCME = 1_B.

Within this mode, the output state of the Timer Slices (the ones set in multi channel mode) can be controlled in parallel by a single pattern.

The pattern is controlled via the CCU4 inputs, CCU4x.MCI0, CCU4x.MCI1, CCU4x.MCI2 and CCU4x.MCI3. Each of these inputs is connected directly to the associated slice input, e.g. CCU4x.MCI0 to CC40MCI, CCU4x.MCI1 to CC41MCI.

This pattern can be controlled directly by one of the POSIF modules and be updated in parallel for all the slices.

Using the POSIF module in conjunction with the Multi Channel support of the CCU4, one can achieve a complete synchronicity between the output state update, CCU4x.OUTy, and the update of a new pattern, **Figure 22-35**.

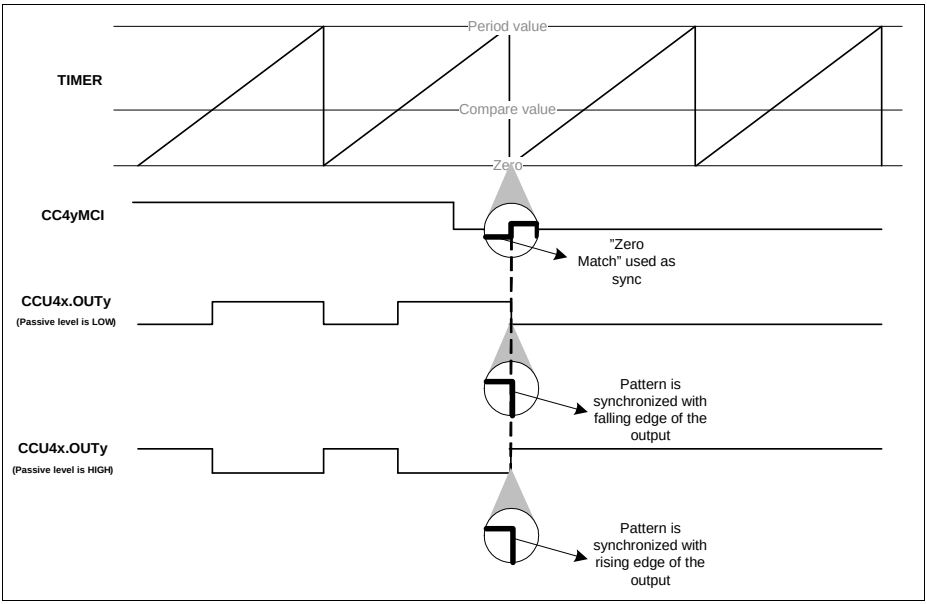


Figure 22-35 Multi channel pattern synchronization

Figure 22-36 shows the usage of the multi channel mode in conjunction with all four Timer Slices inside the CCU4. The multi channel pattern is driven via the POSIF module, which enables a glitch free update of all the outputs of the CCU4.

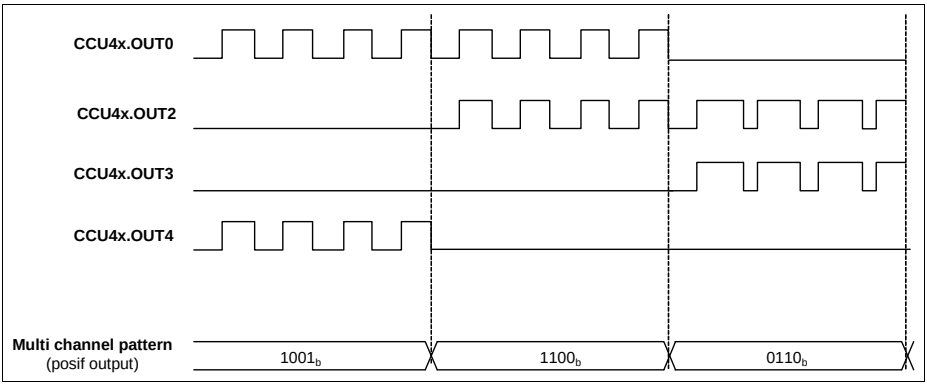


Figure 22-36 Multi Channel mode for multiple Timer Slices

The synchronization between CCU4 and POSIF is achieved, by adding a 3 cycle delay on the output path of each Timer Slice (between the status bit, CC4yST and the direct

control of the output pin). This path is only selected when **CC4yINS**.MCME = 1_B, see **Figure 22-37**.

The multi pattern input synchronization can be seen on **Figure 22-38**. To achieve a synchronization between the update of the status bit, the sampling of a new multi channel pattern input is controlled by the period match or one match signal.

In a normal operation, where no external signal is used to control the counting direction, the signal used to enable the sampling of the pattern is always the period match when in edge aligned and the one match when in center aligned mode. When an external signal is used to control the counting direction, depending if the counter is counting up or counting down, the period match or the one match signal is used, respectively.

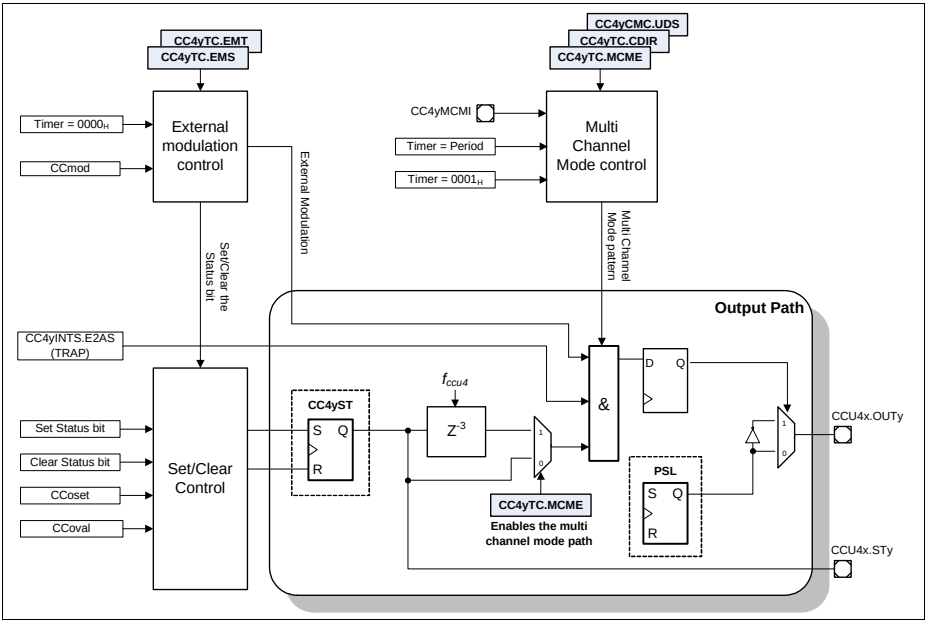


Figure 22-37 CC4y Status bit and Output Path

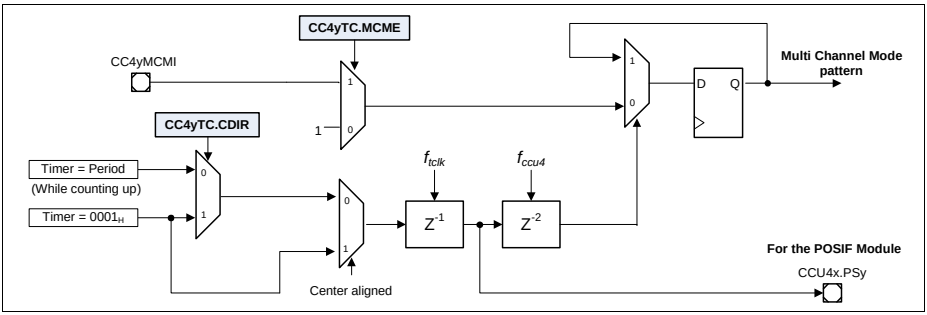


Figure 22-38 Multi Channel Mode Control Logic

22.2.9 Timer Concatenation

The CCU4 offers a very easy mechanism to perform a synchronous timer concatenation. This functionality can be used by setting the **CC4yTC**.TCE = 1_B. By doing this the user is doing a concatenation of the actual CCU4 slice with the previous one, see **Figure 22-39**.

Notice that is not possible to perform concatenation with non adjacent slices and that timer concatenation automatically sets the slice mode into Edge Aligned. It is not possible to perform timer concatenation in Center Aligned mode.

To enable a 64 bit timer, one should set the **CC4yTC**.TCE = 1_B in all the slices (with the exception of the CC40 due to the fact that it doesn't contain this control register).

To enable a 48 bit timer, one should set the **CC4yTC**.TCE = 1_B in two adjacent slices and to enable a 32 bit timer, the **CC4yTC**.TCE is set to 1_B in the slice containing the MSBs. Notice that the timer slice conatining the LSBs should always have the TCE bitfield set to 0_B.

Several combinations for timer concatenation can be made inside a CCU4 module:

- one 64 bit timer
- one 48 bit timer plus a 16 timer
- two 32 bit timers
- one 32 bit timer plus two 16 bit timers

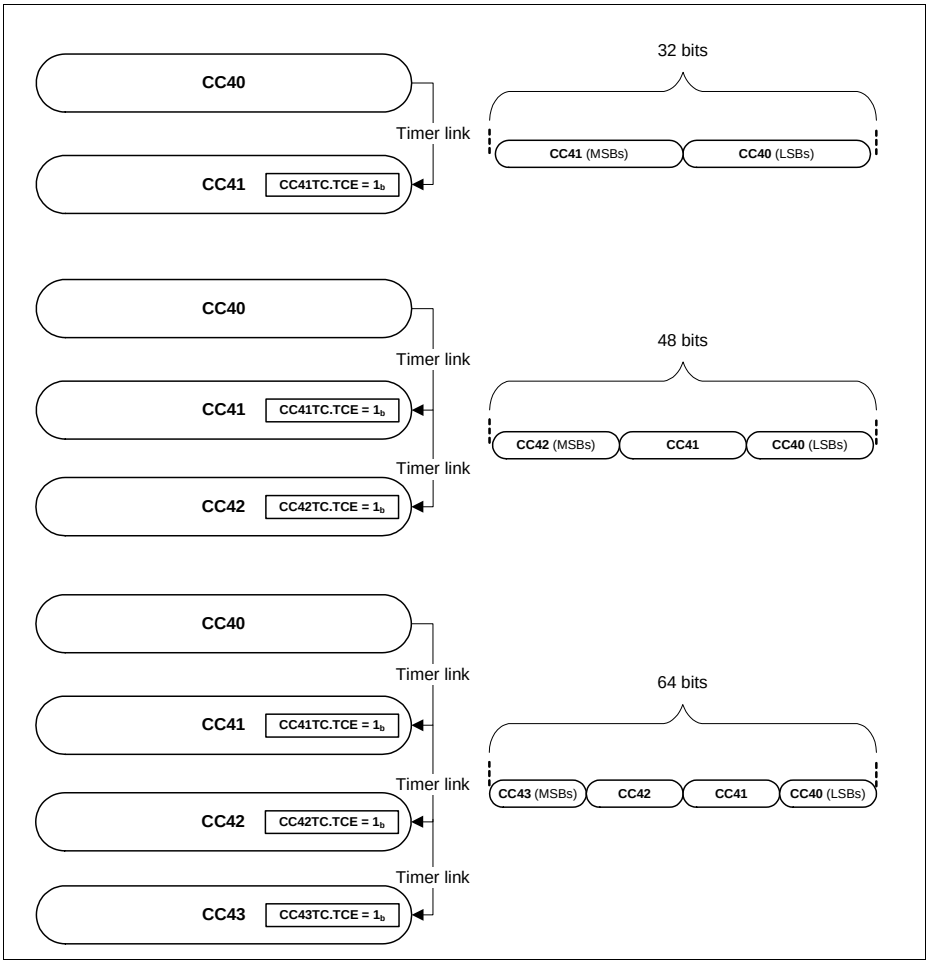


Figure 22-39 Timer Concatenation Example

Each Timer Slice is connected to the adjacent Timer Slices via a dedicated concatenation interface. This interface allows the concatenation of not only the Timer counting operation, but also a synchronous input trigger handling for capturing and loading operations, **Figure 22-40**.

Note: For all cases CC40 and CC43 are not considered adjacent slices

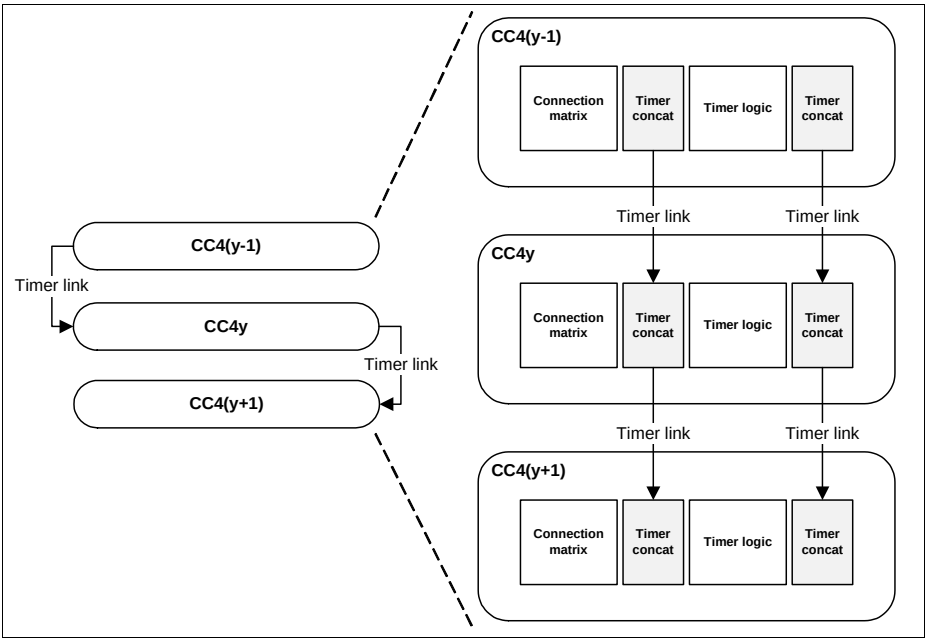


Figure 22-40 Timer Concatenation Link

Seven signals are present in the timer concatenation interface:

- Timer Period match (CC4yPM)
- Timer Zero match (CC4yZM)
- Timer Compare match (CC4yCM)
- Timer counting direction function (CCupd)
- Timer load function (CCload)
- Timer capture function for CC4yC0V and CC4yC1V registers (CCcap0)
- Timer capture function for CC4yC2V and CC4yC3V registers (CCcap1)

The first four signals are used to perform the synchronous timing concatenation at the output of the Timer Logic, like it is seen in **Figure 22-40**. With this link, the timer length can be easily adjusted to 32, 48 or 64 bits (counting up or counting down).

The last three signals are used to perform a synchronous link between the capture and load functions, for the concatenated timer system. This means that the user can have a capture or load function programmed in the first Timer Slice, and propagate this capture or load trigger synchronously from the LSBs until the MSBs, **Figure 22-41**.

The capture or load function only needs to be configured in the first Timer Slice (the one holding the LSBs). From the moment that **CC4yTC**.TCE is set to 1_B, in the following Timer Slices, the link between these functions is done automatically by the hardware.

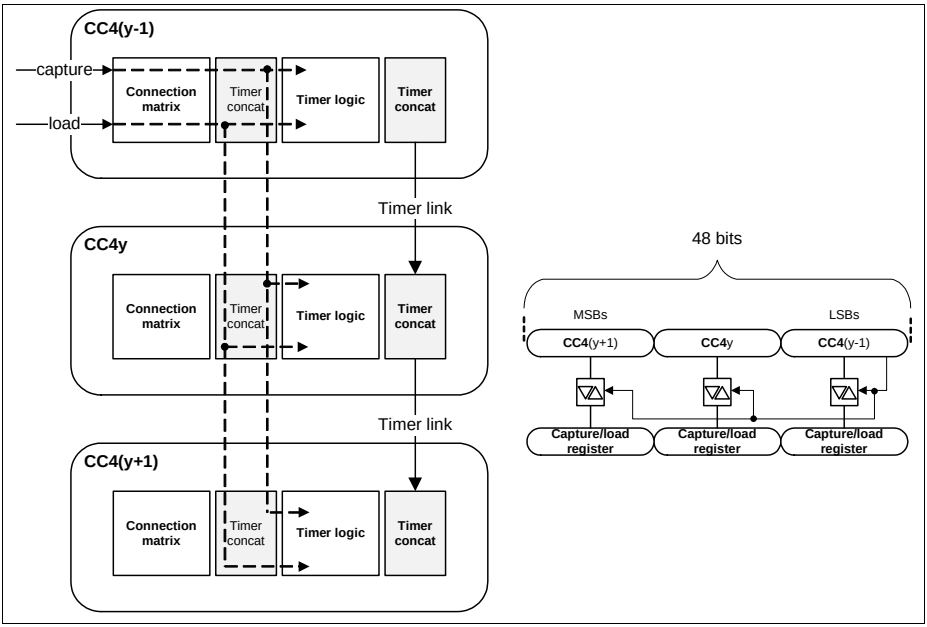


Figure 22-41 Capture/Load Timer Concatenation

The period match (CC4yPM) or zero match (CC4yZM) from the previous Timer Slice (with the immediately next lower index) are used in concatenated mode, as gating signal for the counter. This means that the counting operation of the MSBs only happens when a wrap around condition is detected, avoiding additional DSP operations to extract the counting value.

With the same methodology, the compare match (CC4yCM), zero match and period match are gated with the specific signals from the previous slice. This means that the timing information is propagated throughout all the slices, enabling a completely synchronous match between the LSB and MSB count, see **Figure 22-42**.

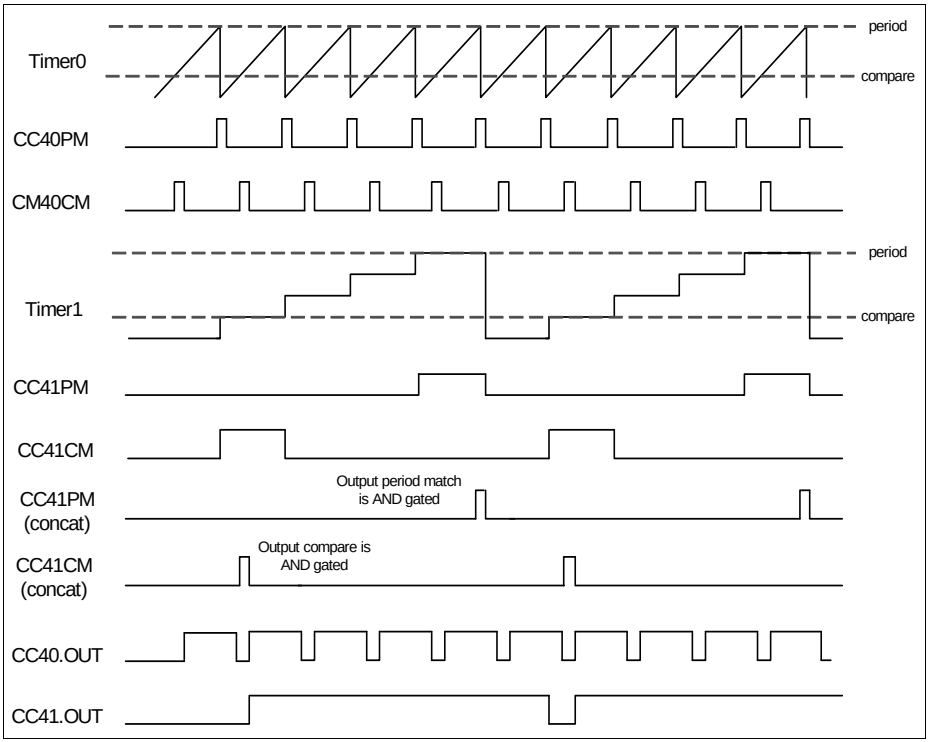


Figure 22-42 32 bit concatenation timing diagram

Note: The counting direction of the concatenated timer needs to be fixed. The timer can count up or count down, but the direction cannot be updated on the fly.

Figure 22-43 gives an overview of the timer concatenation logic. Notice that all the mechanism is controlled solely by the **CC4yTC**.TCE bitfield.

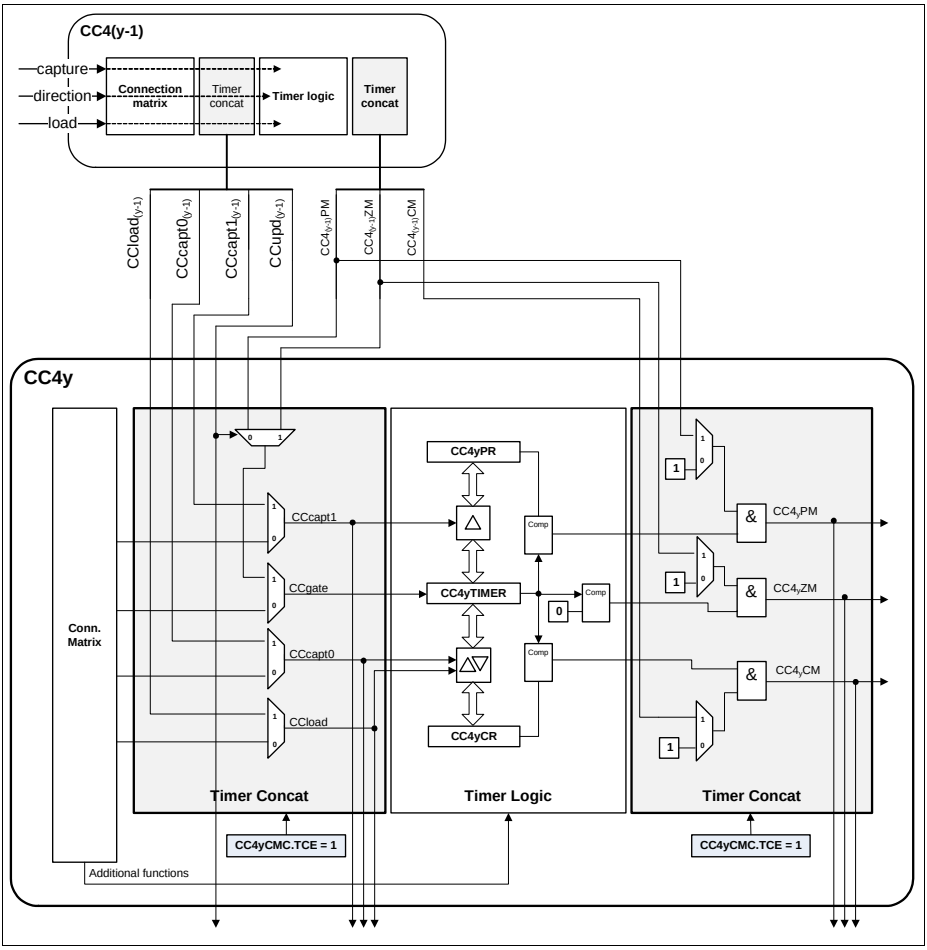


Figure 22-43 Timer concatenation control logic

22.2.10 PWM Dithering

The CCU4 has an automatic PWM dithering insertion function. This functionality can be used with very slow control loops that cannot update the period/compare values in a fast manner, and by that fact the loop can lose precision on long runs. By introducing dither on the PWM signal, the average frequency/duty cycle is then compensated against that error.

Each slice contains a dither control unit, see **Figure 22-44**.

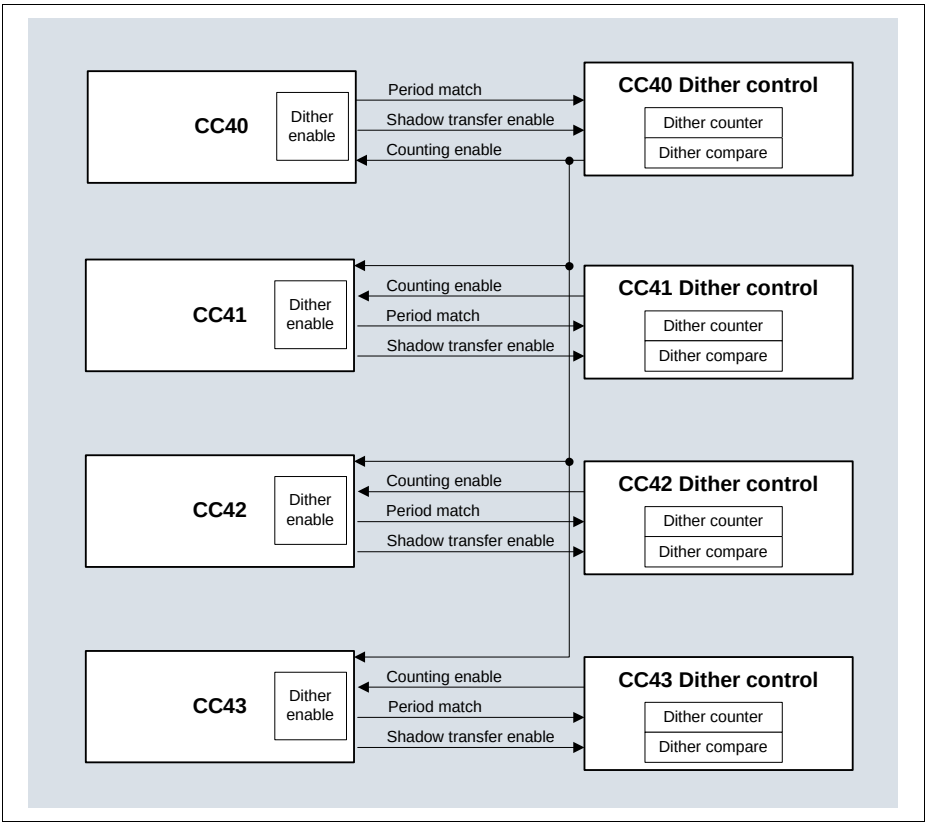


Figure 22-44 Dither structure overview

The dither control unit contains a 4 bit counter and a compare value. The four bit counter is incremented every time that a period match occurs. The counter works in a bit reverse mode so the distribution of increments stays uniform over 16 counter periods, see **Table 22-5**.

Table 22-5 Dither bit reverse counter

counter[3]	counter[2]	counter[1]	counter[0]
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	0
0	0	1	0
1	0	1	0
0	1	1	0
1	1	1	0
0	0	0	1
1	0	0	1
0	1	0	1
1	1	0	1
0	0	1	1
1	0	1	1
0	1	1	1
1	1	1	1

The counter is then compared against a programmed value, **CC4yDIT**.DCS. If the counter value is smaller than the programmed value, a gating signal is generated that can be used to extend the period, to delay the compare or both (controlled by the **CC4yTC**.DITHE field, see **Table 22-6**) for one clock cycle.

Table 22-6 Dither modes

DITHE[1]	DITH[0]	Mode
0	0	Dither is disabled
0	1	Period is increased by 1 cycle
1	0	Compare match is delayed by 1 cycle
1	1	Period is increased by 1 cycle and compare is delayed by 1 cycle

The dither compare value also has an associated shadow register that enables concurrent update with the period/compare register of CC4y. The control logic for the dithering unit is represented on **Figure 22-45**.

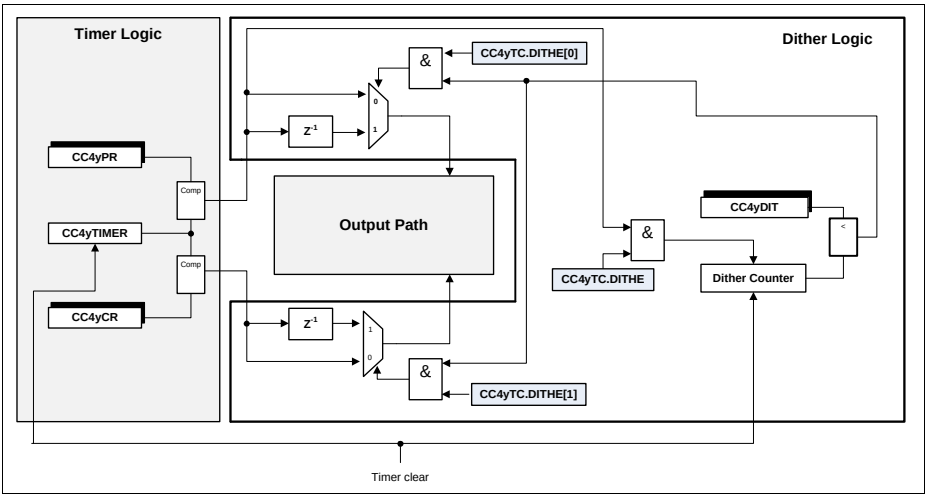


Figure 22-45 Dither control logic

Figure 22-46 to **Figure 22-51** show the effect of the different configurations for the dithering function, **CC4yTC**.DITHE, for both counting schemes, Edge and Center Aligned mode. In each figure, the bit reverse scheme is represented for the dither counter and the compare value was programmed with the value 8_H. In each figure, the variable *T*, represents the period of the counter, while the variable *d* indicates the duty cycle (status bit is set HIGH).

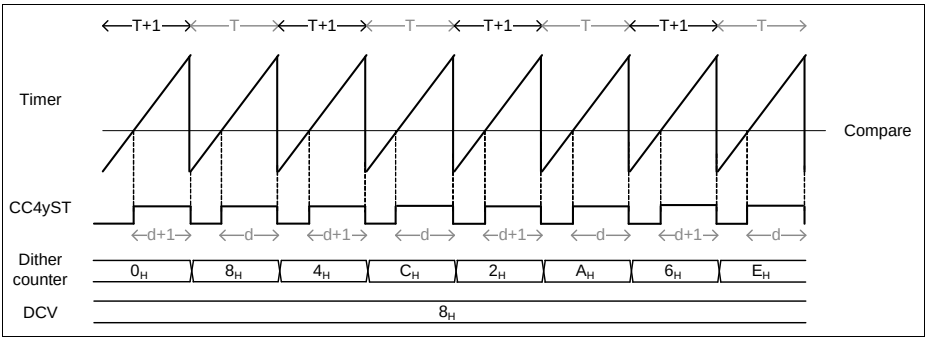


Figure 22-46 Dither timing diagram in edge aligned - CC4yTC.DITHE = 01_B

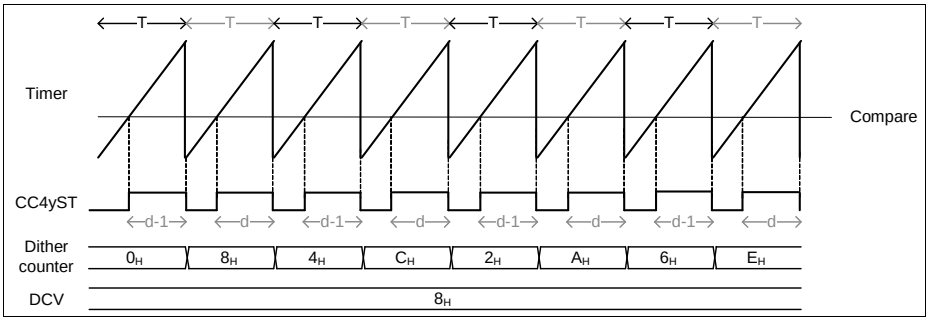


Figure 22-47 Dither timing diagram in edge aligned - CC4yTC.DITHE = 10_B

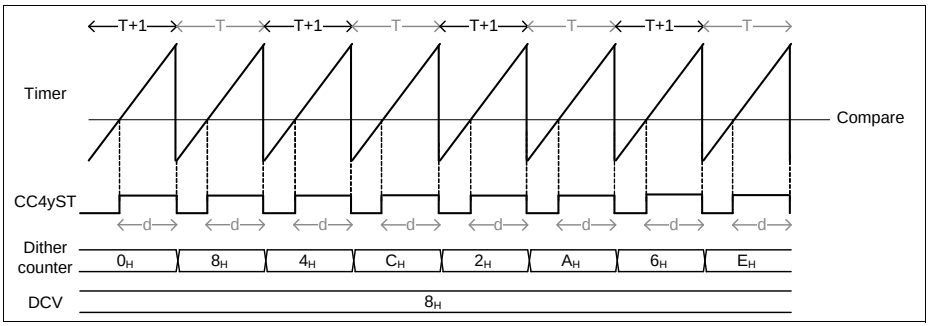


Figure 22-48 Dither timing diagram in edge aligned - CC4yTC.DITHE = 11_B

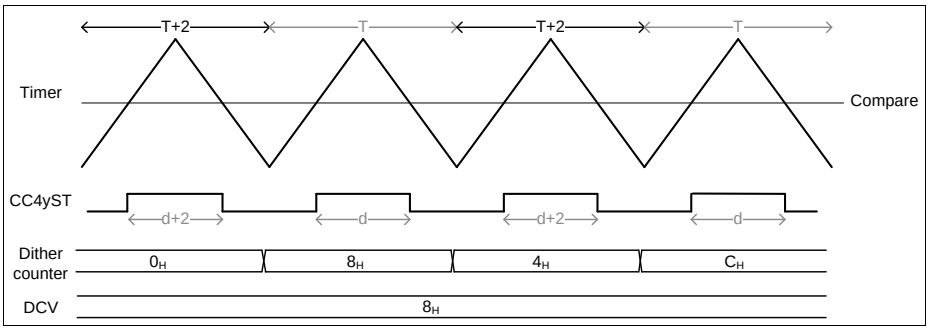


Figure 22-49 Dither timing diagram in center aligned - CC4yTC.DITHE = 01_B

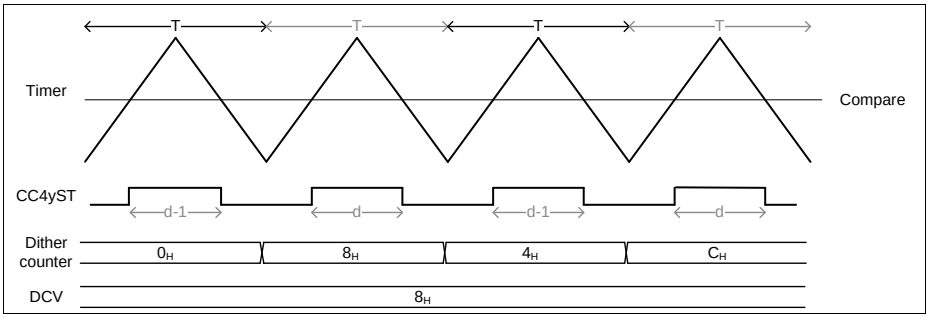


Figure 22-50 Dither timing diagram in center aligned - CC4yTC.DITHE = 10_B

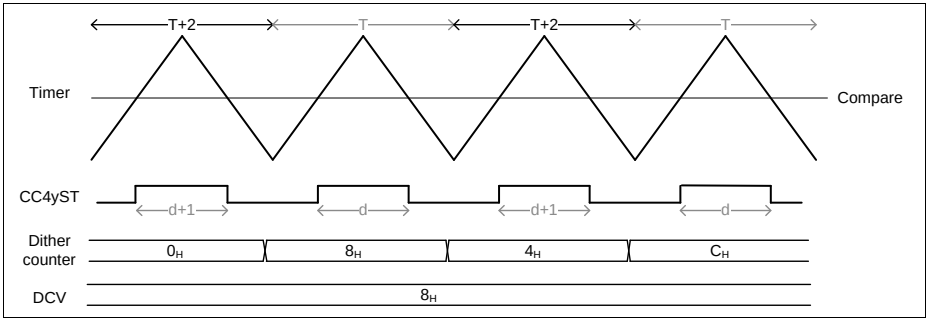


Figure 22-51 Dither timing diagram in center aligned - CC4yTC.DITHE = 11_B

Note: When using the dither, is not possible to select a period value of FS when in edge aligned mode. In center aligned mode, the period value must be at least FS - 2.

22.2.11 Prescaler

The CCU4 contains a 4 bit prescaler that can be used in two operating modes for each individual slice:

- normal prescaler mode
- floating prescaler mode

The run bit of the prescaler can be set/cleared by SW by writing into the registers, **GIDLC**.SPRB and **GIDLS**.CPRB respectively, and it can also be cleared by the run bit of a specific slice. With the last mechanism, the run bit of the prescaler is cleared one clock cycle after the clear of the run bit of the selected slice. To select which slice can perform this action, one should program the **GCTRL**.PRBC register.

22.2.11.1 Normal Prescaler Mode

In Normal prescaler mode the clock fed to the CC4y counter is a normal fixed division by N, accordingly to the value set in the **CC4yPSC**.PSIV register. The values for the possible division values are listed in **Table 22-7**. The **CC4yPSC**.PSIV value is only modified by a SW access. Notice that each slice has a dedicated prescaler value selector (**CC4yPSC**.PSIV), which means that the user can select different counter clocks for each Timer Slice (CC4y).

Table 22-7 Timer clock division options

CC4yPSC.PSIV	**Resulting clock**
0000_B	f_{ccu4}
0001_B	$f_{ccu4}/2$
0010_B	$f_{ccu4}/4$
0011_B	$f_{ccu4}/8$
0100_B	$f_{ccu4}/16$
0101_B	$f_{ccu4}/32$
0110_B	$f_{ccu4}/64$
0111_B	$f_{ccu4}/128$
1000_B	$f_{ccu4}/256$
1001_B	$f_{ccu4}/512$
1010_B	$f_{ccu4}/1024$
1011_B	$f_{ccu4}/2048$
1100_B	$f_{ccu4}/4096$
1101_B	$f_{ccu4}/8192$
1110_B	$f_{ccu4}/16384$
1111_B	$f_{ccu4}/32768$

22.2.11.2 Floating Prescaler Mode

The floating prescaler mode can be used individually in each slice by setting the register **CC4yTC**.FPE = 1_B. With this mode, the user can not only achieve a better precision on the counter clock for compare operations but also reduce the SW read access for the capture mode.

The floating prescaler mode contains additionally to the initial configuration value register, **CC4yPSC**.PSIV, a compare register, **CC4yFPC**.PCMP with an associated shadow register mechanism.

Figure 22-52 shows the structure of the prescaler in floating mode when the specific slice is in compare mode (no external signal is used for capture). In this mode, the value of the clock division is increment by 1_D every time that a timer overflow/underflow (overflow if in Edge Aligned Mode, underflow if in Center Aligned Mode) occurs.

In this mode, the Compare Match from the timer is AND gated with the Compare Match of the prescaler and every time that this event occurs, the value of the clock division is updated with the **CC4yPSC**.PSIV value in the immediately next timer overflow/underflow event.

The shadow transfer of the floating prescaler compare value, **CC4yFPC**.PCMP, is done following the same rules described on **Section 22.2.5.2**.

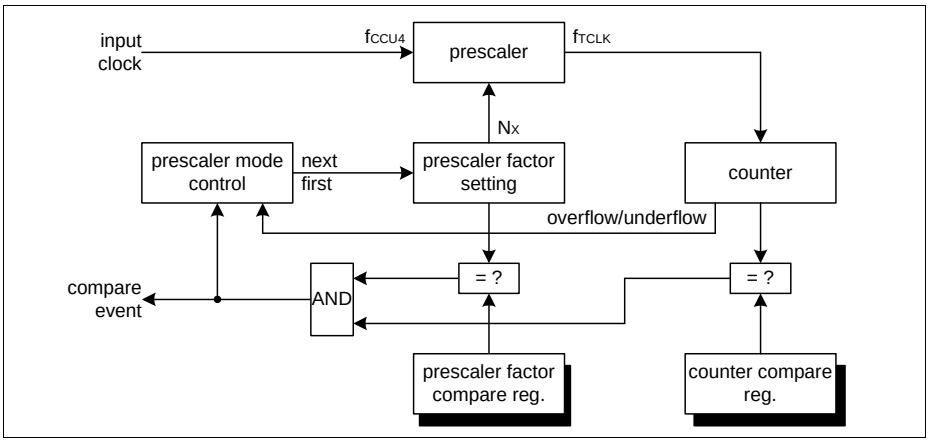


Figure 22-52 Floating prescaler in compare mode overview

When the specific CC4y is operating in capture mode (when at least one external signal is decoded as capture functionality), the actual value of the clock division also needs to be stored every time that a capture event occurs. The floating prescaler can have up to 4 capture registers (the maximum number of capture registers is dictated by the number of capture registers used in the specific slice).

The clock division value continues to be increment by 1_D every time that a timer overflow (in capture mode, the slice is always operating in Edge Aligned Mode) occurs and it is loaded with the PSIV value every time that a capture triggers is detected.

See the **Section 22.2.12.2** for a full description of the usage of the floating prescaler mode in conjunction with compare and capture modes.

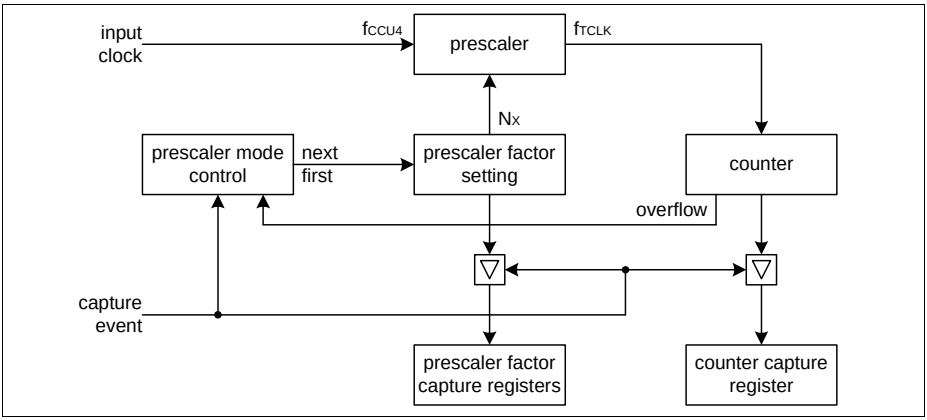


Figure 22-53 Floating Prescaler in capture mode overview

22.2.12 CCU4 Usage

22.2.12.1 PWM Signal Generation

The CCU4 offers a very flexible range in duty cycle configurations. This range is comprised between 0 to 100%.

To generate a PWM signal with a 100% duty cycle in Edge Aligned Mode, one should program the compare value, **CC4yCR**.CR, to 0000_H, see **Figure 22-54**.

In the same manner a 100% duty cycle signal can be generated in Center Aligned Mode, see **Figure 22-55**.

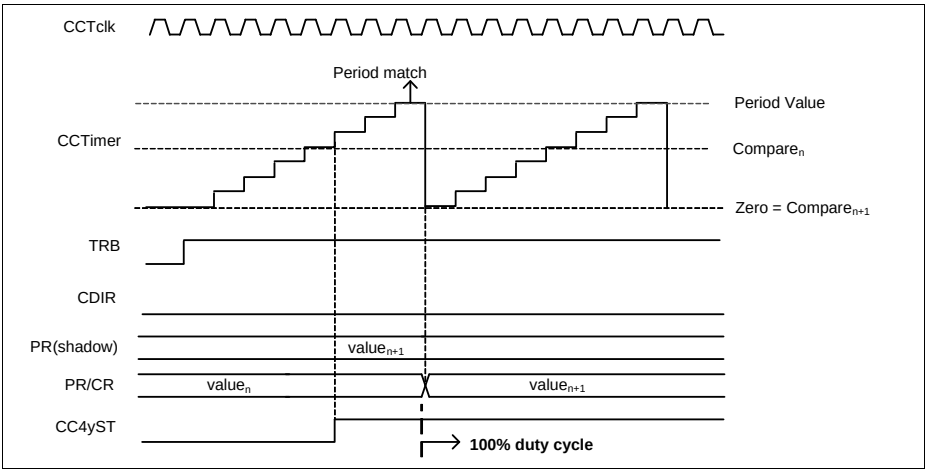


Figure 22-54 PWM with 100% duty cycle - Edge Aligned Mode

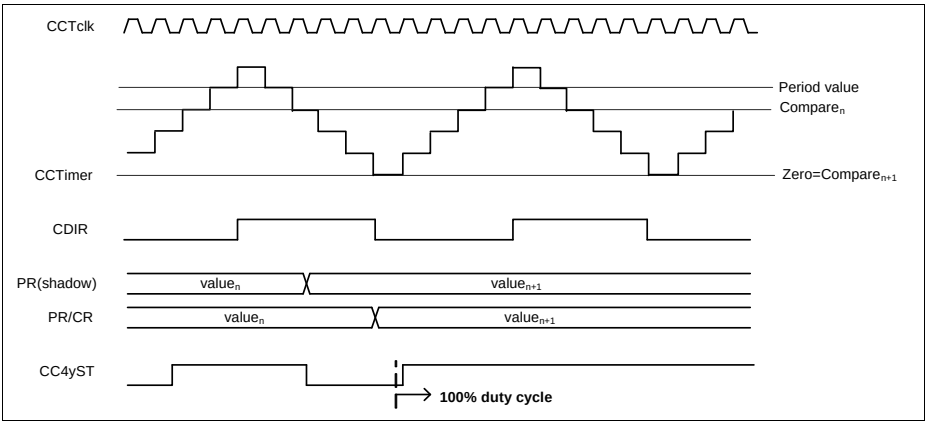


Figure 22-55 PWM with 100% duty cycle - Center Aligned Mode

To generate a PWM signal with 0% duty cycle in Edge Aligned Mode, the compare register should be set with the value programmed into the period value plus 1. In the case that the timer is being used with the full 16 bit capability (counting from 0 to 65535), setting a value bigger than the period value into the compare register is not possible and therefore the smallest duty cycle that can be achieved is 1/FS, see **Figure 22-56**.

In Center Aligned Mode, the counter is never running from 0 to 65535_D, due to the fact that it has to overshoot for one clock cycle the value set in the period register. Therefore the user never has a FS counter, which means that generating a 0% duty cycle signal is

always possible by setting a value in the compare register bigger than the one programmed into the period register, see **Figure 22-57**.

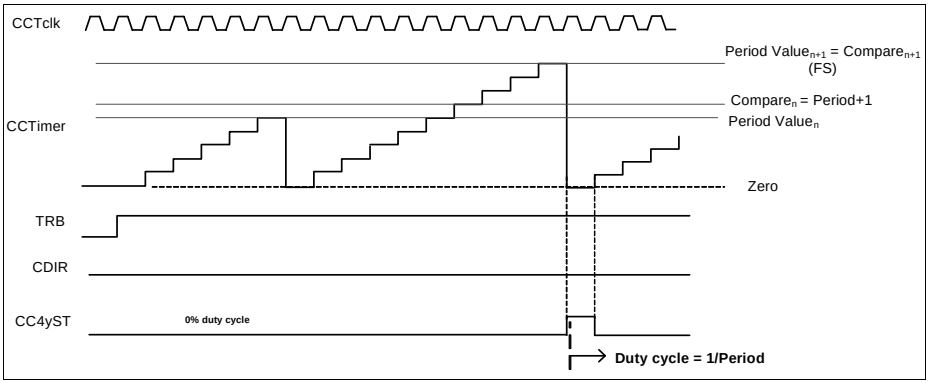


Figure 22-56 PWM with 0% duty cycle - Edge Aligned Mode

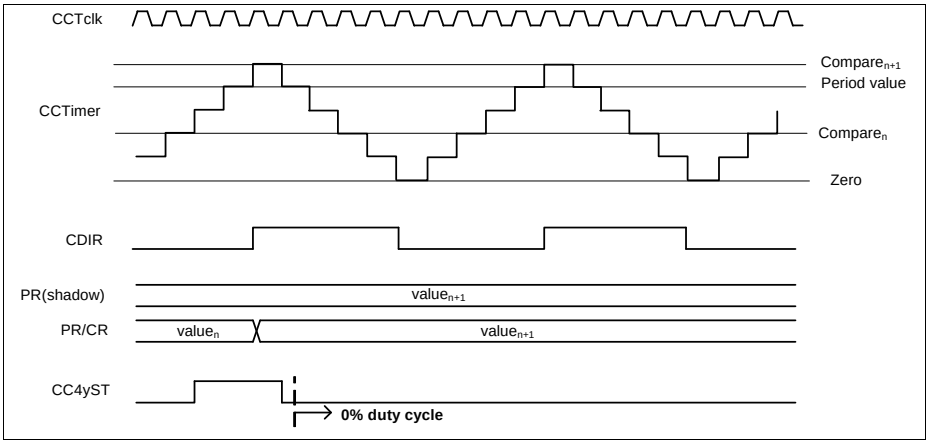


Figure 22-57 PWM with 0% duty cycle - Center Aligned Mode

22.2.12.2 Prescaler Usage

In Normal Prescaler Mode, the frequency of the f_{tclk} fed to the specific CC4y is chosen from the **Table 22-7**, by setting the **CC4yPSC**.PSIV with the required value.

In Floating Prescaler Mode, the frequency of the f_{tclk} can be modified over a selected timeframe, within the values specified in **Table 22-7**. This mechanism is specially useful if, when in capture mode, the dynamic of the capture triggers is very slow or unknown.

In Capture Mode, the Floating Prescaler value is incremented by 1 every time that a timer overflow happens and it is set with the initial programmed value when a capture event happens, see **Figure 22-58**.

When using the Floating Prescaler Mode in Capture Mode, the timer should be cleared each time that a capture event happens, **CC4yTC**.CAPC = 11_B. By operating the Capture mode in conjunction with the Floating Prescaler, even for capture signals that have a periodicity bigger that 16 bits, it is possible to use just a single CCU4 Timer Slice without monitoring the interrupt event of the timer overflow, cycle by cycle. For this the user just needs to know what is the timer captured value and the actual prescaler configuration at the time that the capture event occurred. These values are contained in each CC4yCxV register.

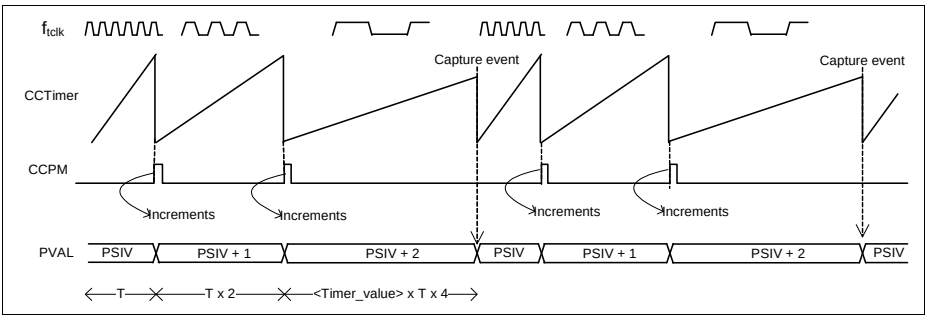


Figure 22-58 Floating Prescaler capture mode usage

When in Compare Mode, the Floating Prescaler function may be used to achieve a fractional PWM frequency or to perform some frequency modulation.

The same incrementing by 1_D mechanism is done every time that a overflow/underflow of the Timer occurs and the actual Prescaler value, doesn't match the one programmed into the **CC4yFPC**.PCMP register.

When a Compare Match from the Timer occurs and the actual Prescaler value is equal to the one programmed on the **CC4yFPC**.PCMP register, then the Prescaler value is set with the initial value, **CC4yPSC**.PSIV, when the next occurrence of a timer overflow/underflow.

In **Figure 22-59**, the Compare value of the Floating Prescaler was set to PSIV + 2. Every time that a timer overflow occurs, the value of the Prescaler is incremented by 1, which means that if we give f_{tclk} as the reference frequency for the **CC4yPSC**.PSIV value, we have $f_{tclk}/2$ for **CC4yPSC**.PSIV + 1 and $f_{tclk}/4$ for **CC4yPSC**.PSIV + 2.

The period over time of the counter becomes:

$$Period = (1/f_{tclk} + 2/f_{tclk} + 4/f_{tclk}) / 3 \quad (22.10)$$

The same mechanism is used in Center Aligned Mode, but to keep the rising arcade and falling arcade always symmetrical, instead of the overflow of the timer, the underflow is used, see **Figure 22-60**.

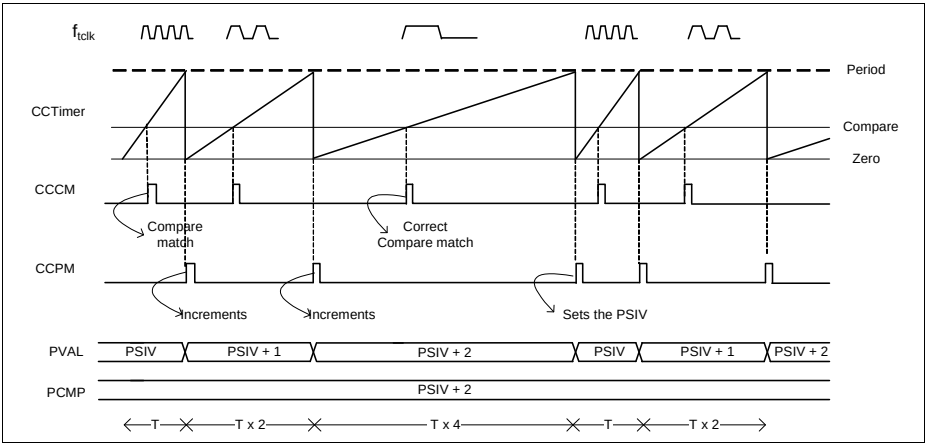


Figure 22-59 Floating Prescaler compare mode usage - Edge Aligned

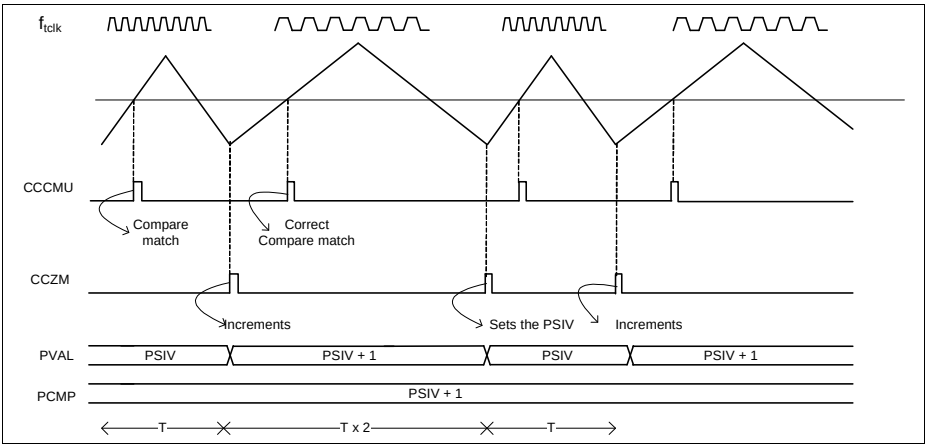


Figure 22-60 Floating Prescaler compare mode usage - Center Aligned

22.2.12.3 PWM Dither

The Dither functionality can be used to achieve a very fine precision on the periodicity of the output state in compare mode. The value set in the dither compare register, **CC4yDIT**.DCV is crosschecked against the actual value of the dither counter and every

time that the dither counter is smaller than the comparison value one of the follows actions is taken:

- The period is extended for 1 clock cycle - **CC4yTC**.DITHE = 01_B; in edge aligned mode
- The period is extended for 2 clock cycles - **CC4yTC**.DITHE = 01_B; in center aligned mode
- The comparison match while counting up (**CC4yTCST**.CDIR = 0_B) is delayed (this means that the status bit is going to stay in the SET state 1 cycle less) for 1 clock cycle - **CC4yTC**.DITHE = 10_B;
- The period is extended for 1 clock cycle and the comparison match while counting up is delayed for 1 clock cycle - **CC4yTC**.DITHE = 11_B; in edge aligned mode
- The period is extended for 2 clock cycles and the comparison match while counting up is delayed for 1 clock cycle; center aligned mode

The bit reverse counter distributes the number programmed in the **CC4yDIT**.DCV throughout a window of 16 timer periods.

Table 22-8 describes the bit reverse distribution versus the programmed value on the **CC4yDIT**.DCV field. The fields marked as '0' indicate that in that counter period, one of the above described actions, is going to be performed.

Table 22-8 Bit reverse distribution

	DCV															
Dither counter	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
8	1	1	1	1	1	1	1	1	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**
4	1	1	1	1	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
C	1	1	1	1	1	1	1	1	1	1	1	1	1	**0**	**0**	**0**
2	1	1	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
A	1	1	1	1	1	1	1	1	1	1	1	**0**	**0**	**0**	**0**	**0**
6	1	1	1	1	1	1	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
E	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**0**
1	1	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
9	1	1	1	1	1	1	1	1	1	1	**0**	**0**	**0**	**0**	**0**	**0**
5	1	1	1	1	1	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
D	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**0**	**0**
3	1	1	1	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
B	1	1	1	1	1	1	1	1	1	1	1	1	**0**	**0**	**0**	**0**

Table 22-8 Bit reverse distribution (cont'd)

	DCV															
Dither counter	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
7	1	1	1	1	1	1	1	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
F	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

The bit reverse distribution versus the programmed **CC4yDIT**.DCV value results in the following values for the Period and duty cycle:

DITHE = 01_B

$Period$ = [(16 - DCV) x T + DCV x (T + 1)]/16; in Edge Aligned Mode (22.11)

$Duty\ cycle$ = [(16 - DCV) x d/T + DCV x ($d+1$)/(T + 1)]/16; in Edge Aligned Mode (22.12)

$Period$ = [(16 - DCV) x T + DCV x (T + 2)]/16; in Center Aligned Mode (22.13)

$Duty\ cycle$ = [(16 - DCV) x d/T + DCV x ($d+2$)/(T + 2)]/16; in Center Aligned Mode (22.14)

DITHE = 10_B

$Period$ = T ; in Edge Aligned Mode (22.15)

$Duty\ cycle$ = [(16 - DCV) x d/T + DCV x (d-1)/T]/16; in Edge Aligned Mode (22.16)

$Period$ = T ; in Center Aligned Mode (22.17)

$Duty\ cycle$ = [(16 - DCV) x d/T + DCV x (d-1)/T]/16; in Center Aligned Mode (22.18)

DITHE = 11_B

$Period$ = [(16 - DCV) x T + DCV x (T + 1)]/16; in Edge Aligned Mode (22.19)

$Duty\ cycle$ = [(16 - DCV) x d/T + DCV x d/(T + 1)]/16; in Edge Aligned Mode (22.20)

$Period$ = [(16 - DCV) x T + DCV x (T + 2)]/16; in Center Aligned Mode (22.21)

$Duty\ cycle$ = [(16 - DCV) x d/T + DCV x ($d+1$)/(T + 2)]/16; in Center Aligned Mode (22.22)

where:

T - Original period of the signal, see **Section 22.2.5.1**

d - Original duty cycle of the signal, see **Section 22.2.5.1**

22.2.12.4 Capture Mode Usage

Each Timer Slice can make use of 2 or 4 capture registers. Using only 2 capture registers means that only 1 Event was linked to a captured trigger. To use the four capture registers, both capture triggers need to be mapped into an Event (it can be the same signal with different edges selected or two different signals) or the **CC4yTC**.SCE field needs to be set to 1, which enables the linking of the 4 capture registers.

The internal slice mechanism for capturing is the same for the capture trigger 1 or capture trigger 0.

Different Capture Events - SCE = 0_B

Capture trigger 1 (CCcapt1) is appointed to the capture register 2, **CC4yC2V** and capture register 3, **CC4yC3V**, while trigger 0 (CCcapt0) is appointed to capture register 1, **CC4yC1V** and 0, **CC4yC0V**.

In each CCcapt0 event, the timer value is stored into **CC4yC1V** and the value of the **CC4yC1V** is transferred into the **CC4yC0V**.

In each CCcapt1 event, the timer value is stored into capture register **CC4yC3V** and the value of the capture register **CC4yC3V** is transferred into **CC4yC2V**.

The capture/transfer mechanism only happens if the specific register is not full. A capture register becomes full when receives a new value and becomes empty after the SW has read back the value.

The full flag is cleared every time that the SW reads back the **CC4yC0V**, **CC4yC1V**, **CC4yC2V** or **CC4yC3V** register. The SW can be informed of a new capture trigger by enabling the interrupt source linked to the specific Event. This means that every time that a capture is made an interrupt pulse is generated.

In the case that the Floating Prescaler Mode is being used, the actual value of the clock division is also stored in the capture register (CC4yCxV).

Figure 22-61 shows an example of how the capture/transfer may be used in a Timer Slice that is using a external signal as count function (to measure the velocity of a rotating device), and an equidistant capture trigger that is used to dictate the timestamp for the velocity calculation (two Timer waveforms are plotted, one that exemplifies the clearing of the timer in each capture event and another without the clearing function active).

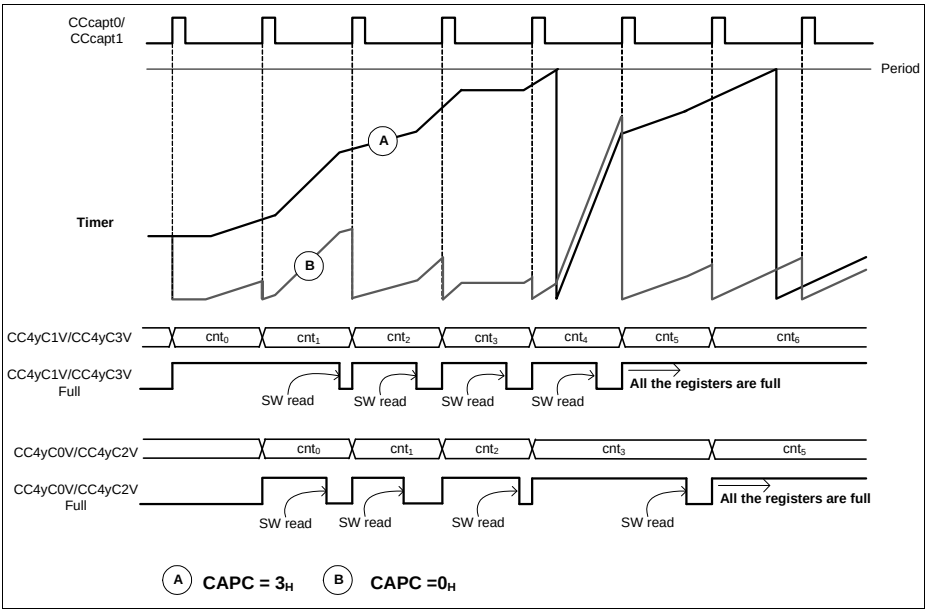


Figure 22-61 Capture mode usage - single channel

Same Capture Event - SCE = 1_B

If the **CC4yTC**.SCE is set to 1_B, all the four capture registers are chained together, emulating a fifo with a depth of 4. In this case, only the capture trigger 1, CCcapt1, is used to perform a capture event.

As an example for this mode, one can consider the case where one Timer Slice is being used in capture mode with SCE = 1_B, with another external signal that controls the counting. This timer slice can be incremented at different speeds, depending on the frequency of the counting signal.

An additional Timer Slice is used to control the capture trigger, dictating the time stamp for the capturing.

A simple scheme for this can be seen in **Figure 22-62**. The CC40ST output of slice 0 was used as capture trigger in the CC41 slice (active on rising and falling edge). The CC40ST output is used as known timebase marker, while the slice timer used for capture is being controlled by external events, e.g. external count.

Due to the fact that we have available 4 capture registers, every time that the SW reads back the complete set of values, 3 speed profiles can be measured.

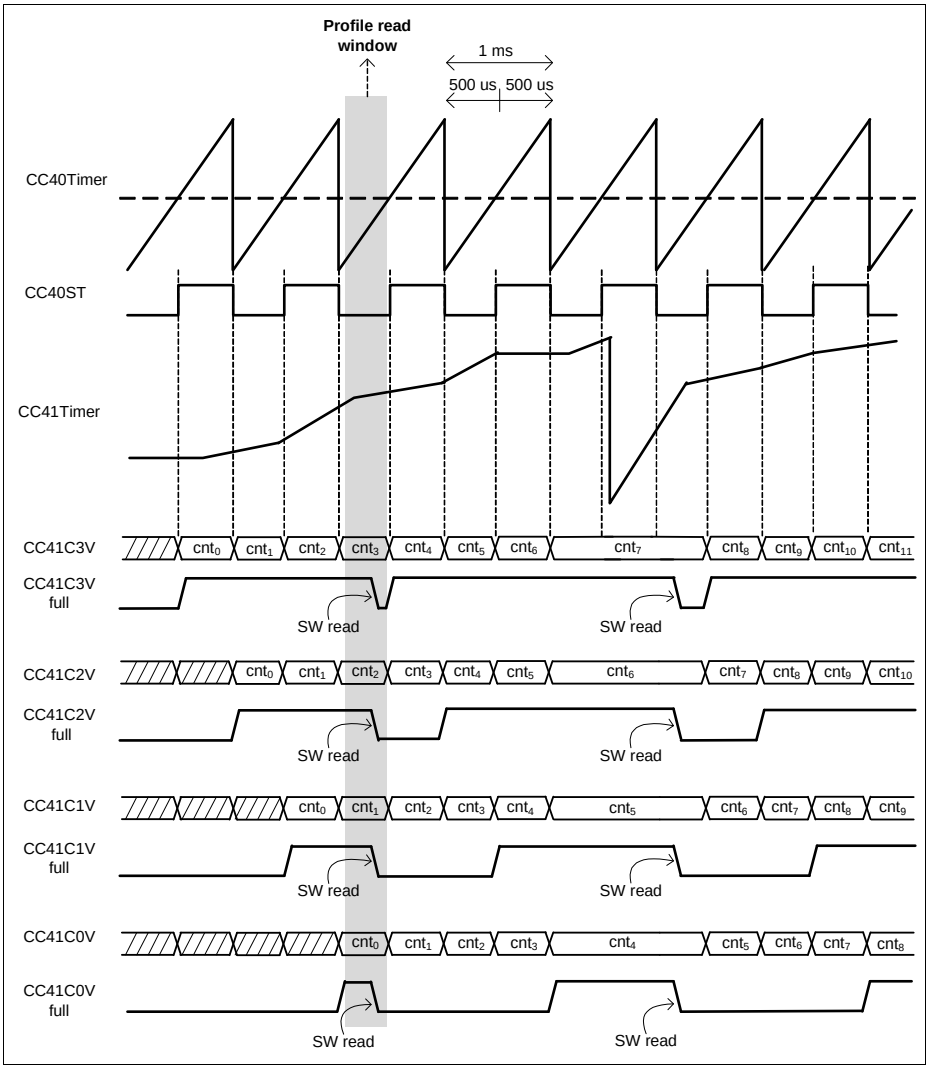


Figure 22-62 Three Capture profiles - CC4yTC.SCE = 1_B

To calculate the three different profiles in **Figure 22-62**, the 4 capture registers need to be read during the pointed read window. After that, the profile calculation is done:

Profile 1 = $CC41C1V_{info}$ - $CC41C0V_{info}$

Profile 2 = $CC41C2V_{info}$ - $CC41C1V_{info}$

Profile 3= $CC41C3V_{info}$ - $CC41C2V_{info}$

Note: This is an example and therefore several Timer Slice configurations and software loops can be implemented.

Extended Capture Read

When multiple Timer Slices need to be programmed into capture mode, it may not be suitable to distribute them over several CCU4 modules. This may be due to resource optimization or availability of Direct Memory Access (DMA) channels.

A simple way to overcome this issue, is to use the Extended Capture Read functionality of CCU4. This mode can be programmed independently for each and every Timer Slice via the **CC4yTC**.ECM bitfield.

The advantage of this mode is that there is only one associated read address for all the capture registers (note that the individual capture registers are still accessible), the **ECRD**. With this one can achieve a DMA channel compression and a better Timer Slice resource optimization through the entire device.

Figure 22-63 exemplifies the usage of the Extended Capture Read function. In this example we have three different Timer Slices that are used to monitor three different applications (in capture mode). An additional Timer Slice (notice that it doesn't need to be in the same CCU4 module) is used to trigger the DMA read of the capture registers.

The read back trigger periodicity can also be updated on the fly to adjust to different system states or operation modes.

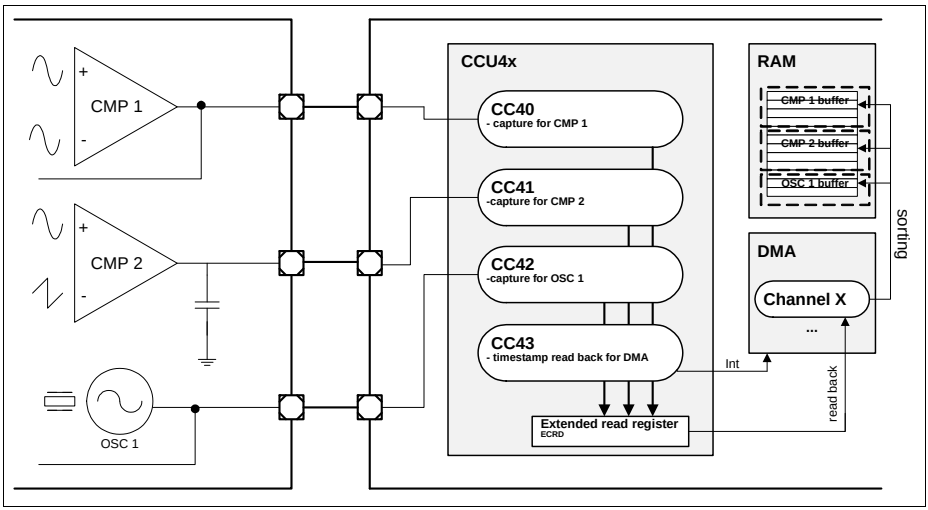


Figure 22-63 Extended read usage scheme example

Every time that the software reads back the **ECRD** register, the CCU4 returns the value of a specific capture register that contains new captured data. The read access of the capture registers follows a circular scheme that is maintained internally by the CCU4, **Figure 22-64**.

For the timer slices that are in capture mode but do not have **CC4yTC**.ECM = 1_B, their captured register values are also read back through the **ECRD**. However, the full flag of the capture registers is not cleared (it is only cleared via a read access to the specific CC4yCxV register).

Only the capture registers of the slices with **CC4yTC**.ECM = 1_B have their full flag cleared with a read access via **ECRD**.

On **Figure 22-65** an example time line is given, in which all the slices were programmed to use extended capture mode, **CC4yTC**.ECM = 1_B. In this example, one can see that the CCU4 doesn't keep memory of which was the first or last captured value between the Timer Slices. Like described on **Figure 22-64**, the read back pointer is incremented until a capture register that has the full flag set, is found.

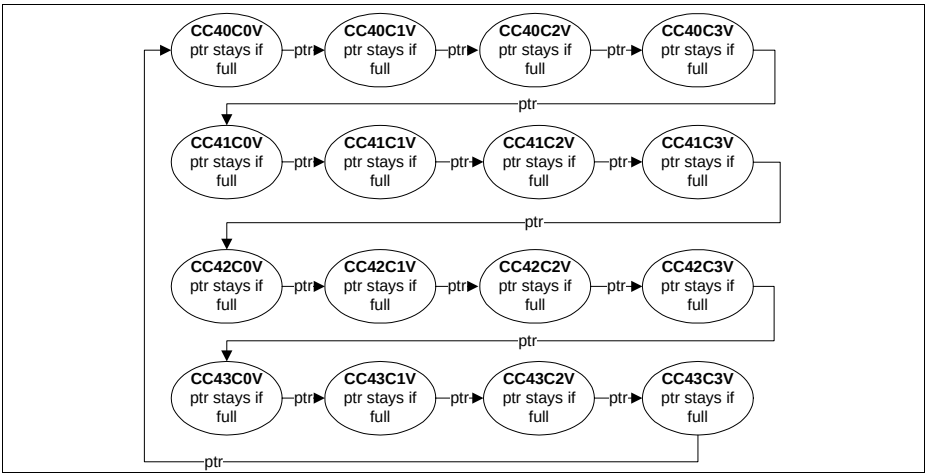


Figure 22-64 Extended Capture Read back

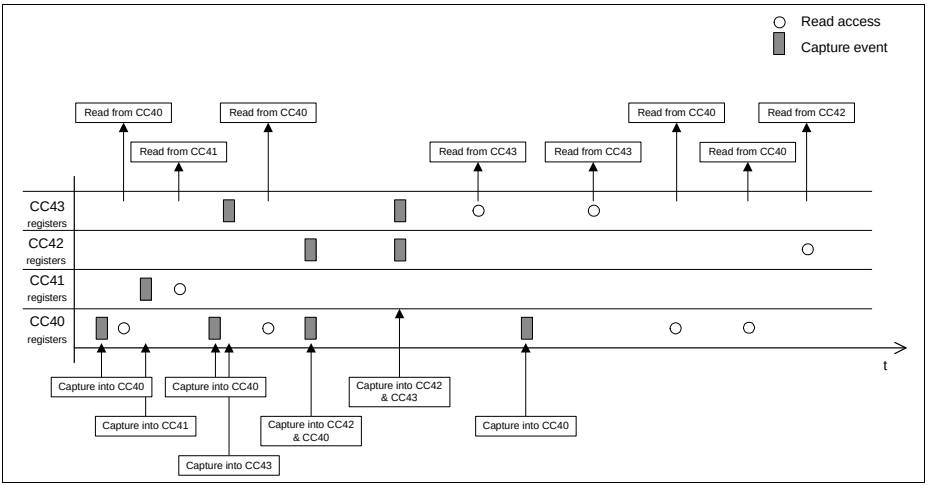


Figure 22-65 Extended Capture Access Example

22.3 Service Request Generation

Each CCU4 slice has an interrupt structure as the one in **Figure 22-66**. The register **CC4yINTS** is the status register for the interrupt sources. Each dedicated interrupt source can be set or cleared by SW, by writing into the specific bit in the **CC4ySWS** and **CC4ySWR** registers respectively.

Each interrupt source can be enabled/disabled via the **CC4yINTE** register. An enabled interrupt source will always generate a pulse on the service request line even if the specific status bit was not cleared. **Table 22-9** describes the interrupt sources of each CCU4 slice.

The interrupt sources, Period Match while counting up and one Match while counting down are ORed together. The same mechanism is applied to the Compare Match while counting up and Compare Match while counting down.

The interrupt sources for the external events are directly linked with the configuration set on the **CC4yINS**.EVxEM. If an event is programmed to be active on both edges, that means that service request pulse is going to be generated when any transition on the external signal is detected. If the event is linked with a level function, the **CC4yINS**.EVxEM still can be programmed to enable a service request pulse. The TRAP event doesn't need any of extra configuration for generating the service request pulse when the slice enters the TRAP state.

Table 22-9 Interrupt sources

Signal	Description
CCINEV0_E	Event 0 edge(s) information from event selector. Used when an external signal should trigger an interrupt.
CCINEV1_E	Event 1 edge(s) information from event selector. Used when an external signal should trigger an interrupt.
CCINEV2_E	Event 2 edge(s) information from event selector. Used when an external signal should trigger an interrupt.
CCPM_U	Period Match while counting up
CCCM_U	Compare Match while counting up
CCCM_D	Compare Match while counting down
CCOM_D	One Match while counting down
Trap state set	Entering Trap State. Will set the E2AS

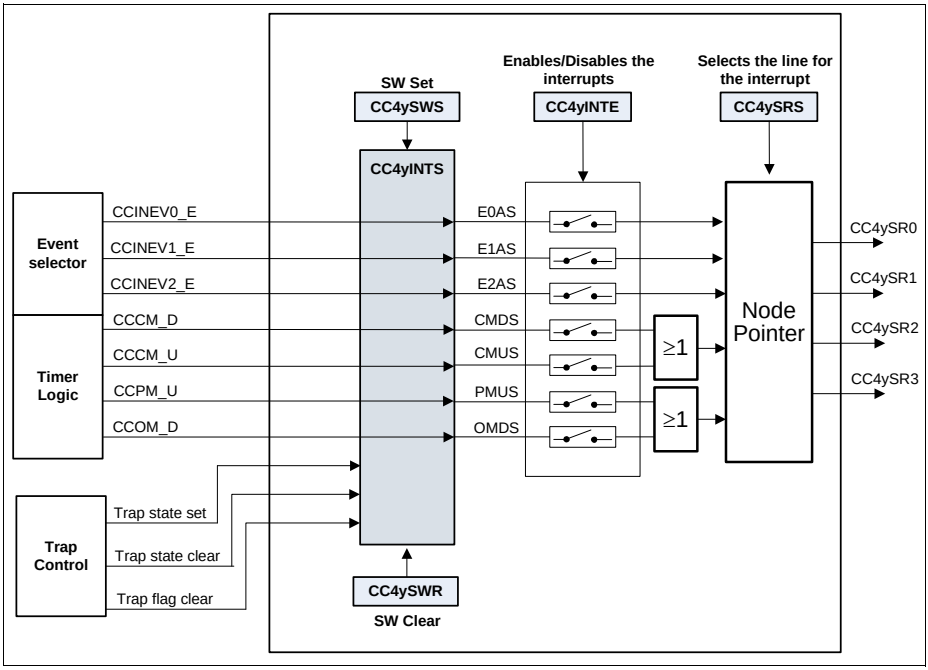


Figure 22-66 Slice interrupt structure overview

Each of the interrupt events can then be forwarded to one of the slice's four service request lines, **Figure 22-67**. The value set on the **CC4ySRS** controls which interrupt event is mapped into which service request line.

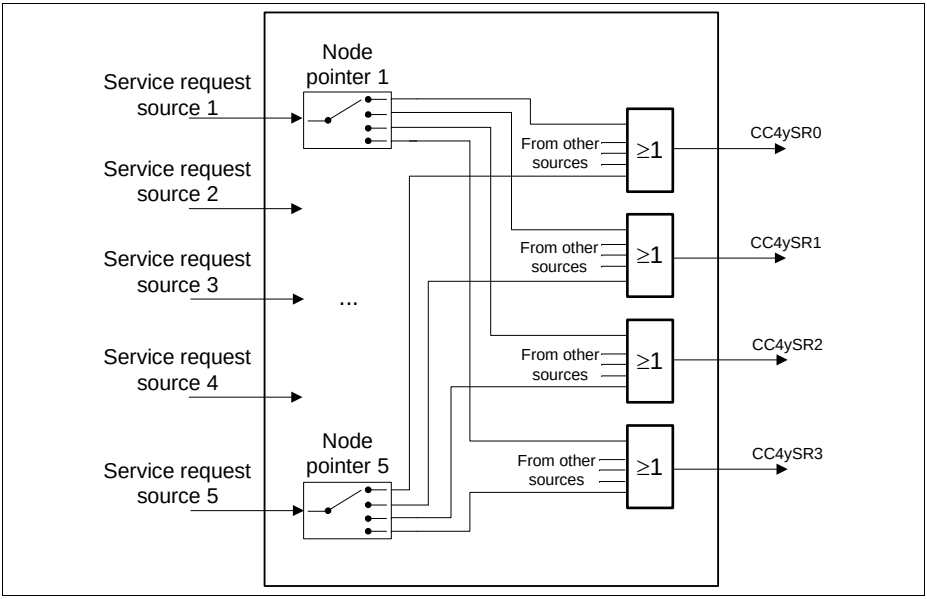


Figure 22-67 Slice Interrupt Node Pointer overview

The four service request lines of each slice are OR together inside the kernel of the CCU4, see **Figure 22-68**. This means that there are only four service request lines per CCU4, that can have in each line interrupt requests coming from different slices.

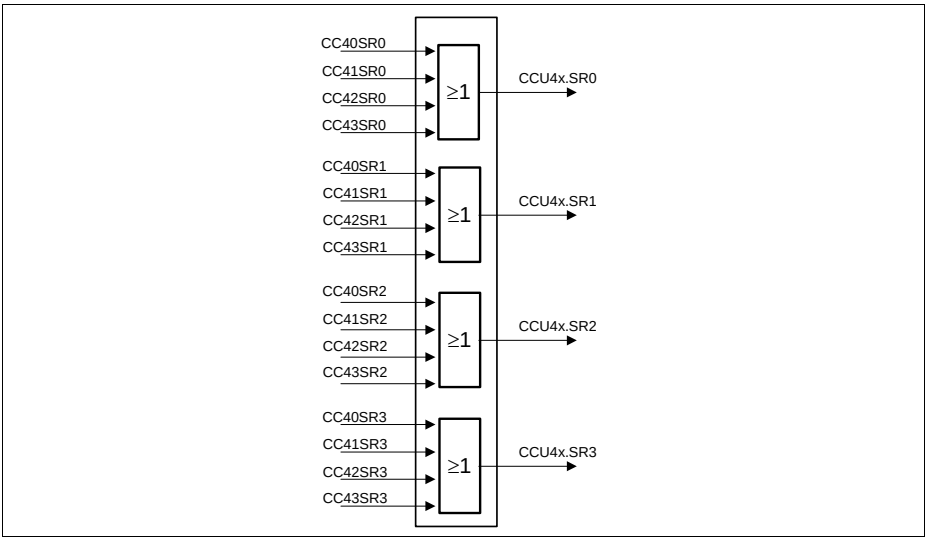


Figure 22-68 CCU4 service request overview

22.4 Debug Behavior

In suspend mode, the functional clocks for all slices as well the prescaler are stopped. The registers can still be accessed by the CPU (read only). This mode is useful for debugging purposes, e.g. where the current device status should be frozen in order to get a snapshot of the internal values. In suspend mode, all the slice timers are stopped. The suspend mode is non-intrusive concerning the register bits. This means register bits are not modified by hardware when entering or leaving the suspend mode.

Entry into suspend mode can be configured at the kernel level by means of the field **GCTRL**.SUSCFG.

The module is only functional after the suspend signal becomes inactive.

22.5 Power, Reset and Clock

The following sections describe the operating conditions, characteristics and timing requirements for the CCU4. All the timing information is related to the module clock, f_{ccu4}.

22.5.1 Clocks

Module Clock

The module clock of the CCU4 module is described in the SCU chapter as f_{CCU}.

The bus interface clock of the CCU4 module is described in the SCU chapter as f_{PERIPH}.

The module clock for the CCU4 is controlled via a specific control bit inside the SCU (System Control Unit), register CLKSET.

It is possible to disable the module clock for the CCU4 via the **GSTAT** register, nevertheless, there may be a dependency of the f_{ccu4} through the different CCU4 instances. One should address the SCU Chapter for a complete description of the product clock scheme.

If module clock dependencies exist through different IP instances, then one can disable the module clock internally inside the specific CCU4, by disabling the prescaler (**GSTAT**.PRB = 0_B).

External Clock

It is possible to use an external clock as source for the prescaler, and consequently for all the timer Slices, CC4y. This external source can be connected to one of the CCU4x.CLK[C...A] inputs.

This external source is nevertheless synchronized against f_{ccu4}.

Table 22-10 External clock operating conditions

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Frequency	f_{eclk}	–	–	$f_{ccu4}/4$	MHz	
ON time	ton_{eclk}	$2T_{ccu4}$ 1)2)	–	–	ns	
OFF time	$toff_{eclk}$	$2T_{ccu4}$ 1)2)	–	–	ns	Only the rising edge is used

1) Only valid if the signal was not previously synchronized/generated with the fccu4 clock (or a synchronous clock)

2) 50% duty cycle is not obligatory

22.5.2 Module Reset

Each CCU4 has one reset source. This reset source is handled at system level and it can be generated independently via a system control register, PRSET0/PRSET1 (address SCU chapter for a full description).

After reset release, the complete IP is set to default configuration. The default configuration for each register field is addressed on **Section 22.7**.

22.5.3 Power

The CCU4 is inside the power core domain, therefore no special considerations about power up or power down sequences need to be taken. For an explanation about the different power domains, please address the SCU (System Control Unit) chapter.

An internal power down mode for the CCU4, can be achieved by disabling the clock inside the CCU4 itself. For this one should set the **GSTAT** register with the default reset value (via the idle mode set register, **GIDLS**).

22.6 Initialization and System Dependencies

22.6.1 Initialization Sequence

The initialization sequence for an application that is using the CCU4, should be the following:

1st Step: Apply reset to the CCU4, via the specific SCU bitfield on the PRSET0/PRSET1 register.

2nd Step: Release reset of the CCU4, via the specific SCU bitfield on the PRCLR0/PRCLR1 register

3rd Step: Enable the CCU4 clock via the specific SCU register, CLKSET.

4th Step: Enable the prescaler block, by writing 1_B to the **GIDLC**.SPRB field.

5th Step: Configure the global CCU4 register **GCTRL**

6th Step: Configure the all the registers related to the required Timer Slice(s) functions, including the interrupt/service request configuration.

7th Step: If needed, configure the startup value for a specific Compare Channel Status, of a Timer Slice, by writing 1_B to the specific **GCSS**.SyTS.

8th Step: Enable the specific timer slice(s), CC4y, by writing 1_B to the specific **GIDLC**.CSyI.

9th Step: For all the Timer Slices that should be started synchronously via SW, the specific system register localized in the SCU, CCUCON, that enables a synchronous timer start should be addressed.

22.6.2 System Dependencies

Each CCU4 may have different dependencies regarding module and bus clock frequencies. This dependencies should be addressed in the SCU and System Architecture Chapters.

Dependencies between several peripherals, regarding different clock operating frequencies may also exist. This should be addressed before configuring the connectivity between the CCU4 and some other peripheral.

The following topics must be taken into consideration for good CCU4 and system operation:

- CCU4 module clock must be at maximum two times faster than the module bus interface clock
- Module input triggers for the CCU4 must not exceed the module clock frequency (if the triggers are generated internally in the device)
- Module input triggers for the CCU4 must not exceed the frequency dictated in **Section 22.5.1**
- Frequency of the CCU4 outputs used as triggers/functions on other modules, must be crosschecked on the end point
- Applying and removing CCU4 from reset, can cause unwanted operations in other modules. This can occur if the modules are using CCU4 outputs as triggers/functions.

22.7 Registers

Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 22-11 Registers Address Space

Module	Base Address	End Address	Note
CCU40	$4000C000_H$	$4000FFFF_H$	
CCU41	40010000_H	$40013FFF_H$	
CCU42	40014000_H	$40017FFF_H$	
CCU43	48004000_H	$48007FFF_H$	

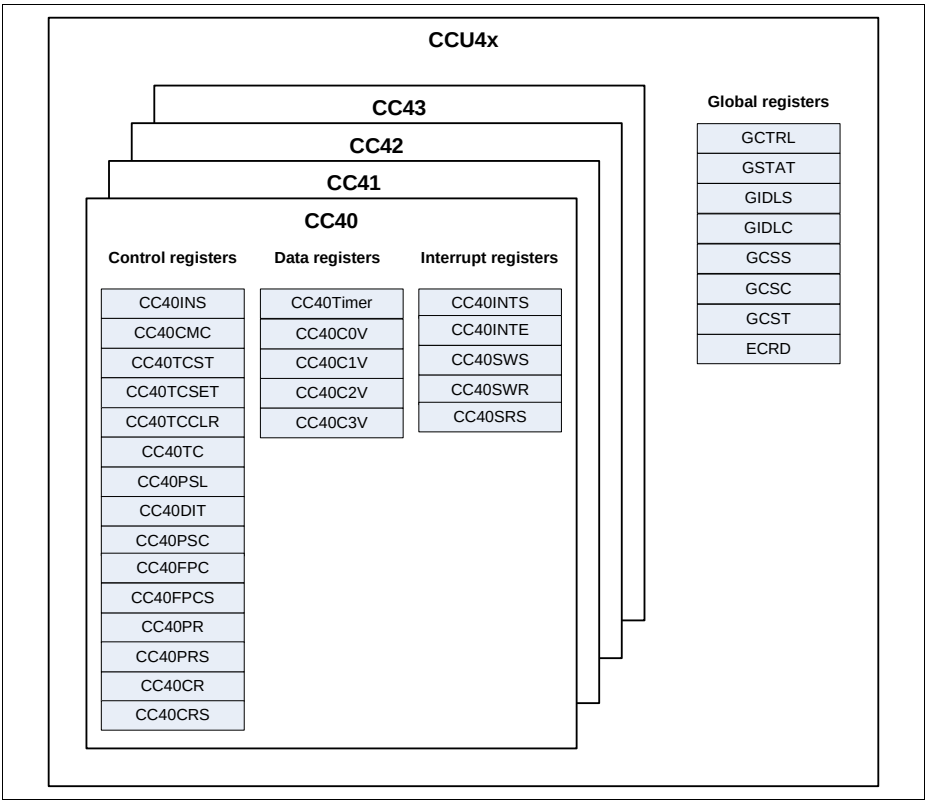


Figure 22-69 CCU4 registers overview

Table 22-12 Register Overview of CCU4

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
CCU4 Global Registers					
GCTRL	Module General Control Register	0000_H	U, PV	U, PV	Page 22-81
GSTAT	General Slice Status Register	0004_H	U, PV	BE	Page 22-84
GIDLS	General Idle Enable Register	0008_H	U, PV	U, PV	Page 22-85
GIDLC	General Idle Disable Register	$000C_H$	U, PV	U, PV	Page 22-87

Table 22-12 Register Overview of CCU4 (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
GCSS	General Channel Set Register	0010_H	U, PV	U, PV	Page 22-88
GCSC	General Channel Clear Register	0014_H	U, PV	U, PV	Page 22-90
GCST	General Channel Status Register	0018_H	U, PV	BE	Page 22-93
ECRD	Extended Read Register	0050_H	U, PV	BE	Page 22-96
MIDR	Module Identification Register	0080_H	U, PV	BE	Page 22-97
CC40 Registers					
CC40INS	Input Selector Unit Configuration	0100_H	U, PV	U, PV	Page 22-98
CC40CMC	Connection Matrix Configuration	0104_H	U, PV	U, PV	Page 22-100
CC40TST	Timer Run Status	0108_H	U, PV	BE	Page 22-103
CC40TCSET	Timer Run Set	$010C_H$	U, PV	U,PV	Page 22-104
CC40TCCLR	Timer Run Clear	0110_H	U, PV	U, PV	Page 22-104
CC40TC	General Timer Configuration	0114_H	U, PV	U, PV	Page 22-105
CC40PSL	Output Passive Level Configuration	0118_H	U, PV	U, PV	Page 22-110
CC40DIT	Dither Configuration	$011C_H$	U, PV	BE	Page 22-111
CC40DITS	Dither Shadow Register	0120_H	U, PV	U, PV	Page 22-112
CC40PSC	Prescaler Configuration	0124_H	U, PV	U, PV	Page 22-112
CC40FPC	Prescaler Compare Value	0128_H	U, PV	U, PV	Page 22-113
CC40FPCS	Prescaler Shadow Compare Value	$012C_H$	U, PV	U, PV	Page 22-114
CC40PR	Timer Period Value	0130_H	U, PV	BE	Page 22-115
CC40PRS	Timer Period Shadow Value	0134_H	U, PV	U, PV	Page 22-115
CC40CR	Timer Compare Value	0138_H	U, PV	BE	Page 22-116
CC40CRS	Timer Compare Shadow Value	$013C_H$	U, PV	U, PV	Page 22-117
CC40TIMER	Timer Current Value	0170_H	U, PV	U, PV	Page 22-118

Table 22-12 Register Overview of CCU4 (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
CC40C0V	Capture Register 0 Value	0174_H	U, PV	BE	Page 22-118
CC40C1V	Capture Register 1Value	0178_H	U, PV	BE	Page 22-119
CC40C2V	Capture Register 2 Value	$017C_H$	U, PV	BE	Page 22-120
CC40C3V	Capture Register 3 Value	0180_H	U, PV	BE	Page 22-121
CC40INTS	Interrupt Status	$01A0_H$	U, PV	BE	Page 22-122
CC40INTE	Interrupt Enable	$01A4_H$	U, PV	U, PV	Page 22-124
CC40SRS	Interrupt Configuration	$01A8_H$	U, PV	U, PV	Page 22-126
CC40SWS	Interrupt Status Set	$01AC_H$	U, PV	U, PV	Page 22-127
CC40SWR	Interrupt Status Clear	$01B0_H$	U, PV	U, PV	Page 22-129
CC41 Registers					
CC41INS	Input Selector Unit Configuration	0200_H	U, PV	U, PV	Page 22-98
CC41CMC	Connection Matrix Configuration	0204_H	U, PV	U, PV	Page 22-100
CC41TST	Timer Run Status	0208_H	U, PV	BE	Page 22-103
CC41TCSET	Timer Run Set	$020C_H$	U, PV	U,PV	Page 22-104
CC41TCCLR	Timer Run Clear	0210_H	U, PV	U, PV	Page 22-104
CC41TC	General Timer Configuration	0214_H	U, PV	U, PV	Page 22-105
CC41PSL	Output Passive Level Configuration	0218_H	U, PV	U, PV	Page 22-110
CC41DIT	Dither Configuration	$021C_H$	U, PV	BE	Page 22-111
CC41DITS	Dither Shadow Register	0220_H	U, PV	U, PV	Page 22-112
CC41PSC	Prescaler Configuration	0224_H	U, PV	U, PV	Page 22-112
CC41FPC	Prescaler Compare Value	0228_H	U, PV	U, PV	Page 22-113
CC41FPCS	Prescaler Shadow Compare Value	$022C_H$	U, PV	U, PV	Page 22-114
CC41PR	Timer Period Value	0230_H	U, PV	BE	Page 22-115
CC41PRS	Timer Period Shadow Value	0234_H	U, PV	U, PV	Page 22-115
CC41CR	Timer Compare Value	0238_H	U, PV	BE	Page 22-116

Table 22-12 Register Overview of CCU4 (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
CC41CRS	Timer Compare Shadow Value	$023C_H$	U, PV	U, PV	Page 22-117
CC41TIMER	Timer Current Value	0270_H	U, PV	U, PV	Page 22-118
CC41C0V	Capture Register 0 Value	0274_H	U, PV	BE	Page 22-118
CC41C1V	Capture Register 1Value	0278_H	U, PV	BE	Page 22-119
CC41C2V	Capture Register 2 Value	$027C_H$	U, PV	BE	Page 22-120
CC41C3V	Capture Register 3 Value	0280_H	U, PV	BE	Page 22-121
CC41INTS	Interrupt Status	$02A0_H$	U, PV	BE	Page 22-122
CC41INTE	Interrupt Enable	$02A4_H$	U, PV	U, PV	Page 22-124
CC41SRS	Interrupt Configuration	$02A8_H$	U, PV	U, PV	Page 22-126
CC41SWS	Interrupt Status Set	$02AC_H$	U, PV	U, PV	Page 22-127
CC41SWR	Interrupt Status Clear	$02B0_H$	U, PV	U, PV	Page 22-129
CC42 Registers					
CC42INS	Input Selector Unit Configuration	0300_H	U, PV	U, PV	Page 22-98
CC42CMC	Connection Matrix Configuration	0304_H	U, PV	U, PV	Page 22-100
CC42TST	Timer Run Status	0308_H	U, PV	BE	Page 22-103
CC42TCSET	Timer Run Set	$030C_H$	U, PV	U,PV	Page 22-104
CC42TCCLR	Timer Run Clear	0310_H	U, PV	U, PV	Page 22-104
CC42TC	General Timer Configuration	0314_H	U, PV	U, PV	Page 22-105
CC42PSL	Output Passive Level Configuration	0318_H	U, PV	U, PV	Page 22-110
CC42DIT	Dither Configuration	$031C_H$	U, PV	BE	Page 22-111
CC42DITS	Dither Shadow Register	0320_H	U, PV	U, PV	Page 22-112
CC42PSC	Prescaler Configuration	0324_H	U, PV	U, PV	Page 22-112
CC42FPC	Prescaler Compare Value	0328_H	U, PV	U, PV	Page 22-113
CC42FPCS	Prescaler Shadow Compare Value	$032C_H$	U, PV	U, PV	Page 22-114
CC42PR	Timer Period Value	0330_H	U, PV	BE	Page 22-115

Table 22-12 Register Overview of CCU4 (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
CC42PRS	Timer Period Shadow Value	0334_H	U, PV	U, PV	Page 22-115
CC42CR	Timer Compare Value	0338_H	U, PV	BE	Page 22-116
CC42CRS	Timer Compare Shadow Value	$033C_H$	U, PV	U, PV	Page 22-117
CC42TIMER	Timer Current Value	0370_H	U, PV	U, PV	Page 22-118
CC42C0V	Capture Register 0 Value	0374_H	U, PV	BE	Page 22-118
CC42C1V	Capture Register 1Value	0378_H	U, PV	BE	Page 22-119
CC42C2V	Capture Register 2 Value	$037C_H$	U, PV	BE	Page 22-120
CC42C3V	Capture Register 3 Value	0380_H	U, PV	BE	Page 22-121
CC42INTS	Interrupt Status	$03A0_H$	U, PV	BE	Page 22-122
CC42INTE	Interrupt Enable	$03A4_H$	U, PV	U, PV	Page 22-124
CC42SRS	Interrupt Configuration	$03A8_H$	U, PV	U, PV	Page 22-126
CC42SWS	Interrupt Status Set	$03AC_H$	U, PV	U, PV	Page 22-127
CC42SWR	Interrupt Status Clear	$03B0_H$	U, PV	U, PV	Page 22-129
CC43 Registers					
CC43INS	Input Selector Unit Configuration	0400_H	U, PV	U, PV	Page 22-98
CC43CMC	Connection Matrix Configuration	0404_H	U, PV	U, PV	Page 22-100
CC43TST	Timer Run Status	0408_H	U, PV	BE	Page 22-103
CC43TCSET	Timer Run Set	$040C_H$	U, PV	U,PV	Page 22-104
CC43TCCLR	Timer Run Clear	0410_H	U, PV	U, PV	Page 22-104
CC43TC	General Timer Configuration	0414_H	U, PV	U, PV	Page 22-105
CC43PSL	Output Passive Level Configuration	0418_H	U, PV	U, PV	Page 22-110
CC43DIT	Dither Configuration	$041C_H$	U, PV	BE	Page 22-111
CC43DITS	Dither Shadow Register	0420_H	U, PV	U, PV	Page 22-112
CC43PSC	Prescaler Configuration	0424_H	U, PV	U, PV	Page 22-112
CC43FPC	Prescaler Compare Value	0428_H	U, PV	U, PV	Page 22-113

Table 22-12 Register Overview of CCU4 (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
CC43FPCS	Prescaler Shadow Compare Value	$042C_H$	U, PV	U, PV	Page 22-114
CC43PR	Timer Period Value	0430_H	U, PV	BE	Page 22-115
CC43PRS	Timer Period Shadow Value	0434_H	U, PV	U, PV	Page 22-115
CC43CR	Timer Compare Value	0438_H	U, PV	BE	Page 22-116
CC43CRS	Timer Compare Shadow Value	$043C_H$	U, PV	U, PV	Page 22-117
CC43TIMER	Timer Current Value	0470_H	U, PV	U, PV	Page 22-118
CC43C0V	Capture Register 0 Value	0474_H	U, PV	BE	Page 22-118
CC43C1V	Capture Register 1Value	0478_H	U, PV	BE	Page 22-119
CC43C2V	Capture Register 2 Value	$047C_H$	U, PV	BE	Page 22-120
CC43C3V	Capture Register 3 Value	0480_H	U, PV	BE	Page 22-121
CC43INTS	Interrupt Status	$04A0_H$	U, PV	BE	Page 22-122
CC43INTE	Interrupt Enable	$04A4_H$	U, PV	U, PV	Page 22-124
CC43SRS	Interrupt Configuration	$04A8_H$	U, PV	U, PV	Page 22-126
CC43SWS	Interrupt Status Set	$04AC_H$	U, PV	U, PV	Page 22-127
CC43SWR	Interrupt Status Clear	$04B0_H$	U, PV	U, PV	Page 22-129

1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

22.7.1 Global Registers

GCTRL

The register contains the global configuration fields that affect all the timer slices inside CCU4.

GCTRL
Global Control Register **(0000_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSDE		MSE3	MSE2	MSE1	MSE0	SUSCFG		0		PCIS		0	PRBC		
rw		rw	rw	rw	rw	rw		r		rw		r	rw		

Field	Bits	Type	Description
PRBC	[2:0]	rw	**Prescaler Clear Configuration** This register controls the how the prescaler Run Bit and internal registers are cleared. 000_B SW only 001_B **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC40 is cleared. 010_B **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC41 is cleared. 011_B **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC42 is cleared. 100_B **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC43 is cleared.
PCIS	[5:4]	rw	**Prescaler Input Clock Selection** 00_B Module clock 01_B CCU4x.ECLKA 10_B CCU4x.ECLKB 11_B CCU4x.ECLKC

Field	Bits	Type	Description
SUSCFG	[9:8]	rw	**Suspend Mode Configuration** This field controls the entering in suspend mode for all the CAPCOM4 slices. 00_B Suspend request ignored. The module never enters in suspend 01_B Stops all the running slices immediately. Safe stop is not applied. 10_B Stops the block immediately and clamps all the outputs to PASSIVE state. Safe stop is applied. 11_B Waits for the roll over of each slice to stop and clamp the slices outputs. Safe stop is applied.
MSE0	10	rw	**Slice 0 Multi Channel shadow transfer enable** When this field is set, a shadow transfer of slice 0 can be requested not only by SW but also via the CCU4x.MCSS input. 0_B Shadow transfer can only be requested by SW 1_B Shadow transfer can be requested via SW and via the CCU4x.MCSS input.
MSE1	11	rw	**Slice 1 Multi Channel shadow transfer enable** When this field is set, a shadow transfer of slice 1 can be requested not only by SW but also via the CCU4x.MCSS input. 0_B Shadow transfer can only be requested by SW 1_B Shadow transfer can be requested via SW and via the CCU4x.MCSS input.
MSE2	12	rw	**Slice 2 Multi Channel shadow transfer enable** When this field is set, a shadow transfer of slice 2 can be requested not only by SW but also via the CCU4x.MCSS input. 0_B Shadow transfer can only be requested by SW 1_B Shadow transfer can be requested via SW and via the CCU4x.MCSS input.

Field	Bits	Type	Description
MSE3	13	rw	**Slice 3 Multi Channel shadow transfer enable** When this field is set, a shadow transfer of slice 3 can be requested not only by SW but also via the CCU4x.MCSS input. 0_B Shadow transfer can only be requested by SW 1_B Shadow transfer can be requested via SW and via the CCU4x.MCSS input.
MSDE	[15:14]	rw	**Multi Channel shadow transfer request configuration** This field configures the type of shadow transfer requested via the CCU4x.MCSS input. The field **CC4yTC**.MSEy needs to be set in order for this configuration to have any effect. 00_B Only the shadow transfer for period and compare values is requested 01_B Shadow transfer for the compare, period and prescaler compare values is requested 10_B Reserved 11_B Shadow transfer for the compare, period, prescaler and dither compare values is requested
0	3, [7:6], [31:16]	r	**Reserved** A read always returns 0.

GSTAT

The register contains the status of the prescaler and each timer slice (idle mode or running).

GSTAT
Global Status Register **(0004_H)** **Reset Value: $0000000F_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							PRB	0				S3I	S2I	S1I	S0I
r							rh	r				r	r	r	r

Field	Bits	Type	Description
S0I	0	r	**CC40 IDLE status** This bit indicates if the CC40 slice is in IDLE mode or not. In IDLE mode the clocks for the CC40 slice are stopped. 0_B Running 1_B Idle
S1I	1	r	**CC41 IDLE status** This bit indicates if the CC41 slice is in IDLE mode or not. In IDLE mode the clocks for the CC41 slice are stopped. 0_B Running 1_B Idle
S2I	2	r	**CC42 IDLE status** This bit indicates if the CC42 slice is in IDLE mode or not. In IDLE mode the clocks for the CC42 slice are stopped. 0_B Running 1_B Idle
S3I	3	r	**CC43 IDLE status** This bit indicates if the CC43 slice is in IDLE mode or not. In IDLE mode the clocks for the CC43 slice are stopped. 0_B Running 1_B Idle

Field	Bits	Type	Description
PRB	8	rh	**Prescaler Run Bit** 0_B Prescaler is stopped 1_B Prescaler is running
0	[7:4], [31:9]	r	**Reserved** Read always returns 0.

GIDLS

Through this register one can set the prescaler and the specific timer slices into idle mode.

GIDLS
Global Idle Set **(0008_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						PSIC	CPRB	0				SS3I	SS2I	SS1I	SS0I
r						w	w	r				w	w	w	w

Field	Bits	Type	Description
SS0I	0	w	**CC40 IDLE mode set** Writing a 1_B to this bit sets the CC40 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared. A read access always returns 0.
SS1I	1	w	**CC41 IDLE mode set** Writing a 1_B to this bit sets the CC41 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared. A read access always returns 0.

Field	Bits	Type	Description
SS2I	2	w	**CC42 IDLE mode set** Writing a 1_B to this bit sets the CC42 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared. A read access always returns 0.
SS3I	3	w	**CC43 IDLE mode set** Writing a 1_B to this bit sets the CC43 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared. A read access always returns 0.
CPRB	8	w	**Prescaler Run Bit Clear** Writing a 1_B into this register clears the Run Bit of the prescaler. Prescaler internal registers are not cleared. A read always returns 0.
PSIC	9	w	**Prescaler clear** Writing a 1_B to this register clears the prescaler counter. It also loads the PSIV into the PVAL field for all Timer Slices. This performs a re alignment of the timer clock for all Slices. The Run Bit of the prescaler is not cleared. A read always returns 0.
0	[7:4], [31:10]	r	**Reserved** Read always returns 0.

GIDLC

Through this register one can remove the prescaler and the specific timer slices from idle mode.

GIDLC
Global Idle Clear **($000C_H$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							SPRB	0				CS3I	CS2I	CS1I	CS0I
r							w	r				w	w	w	w

Field	Bits	Type	Description
CS0I	0	w	**CC40 IDLE mode clear** Writing a 1_B to this bit removes the CC40 from IDLE mode. A read access always returns 0.
CS1I	1	w	**CC41 IDLE mode clear** Writing a 1_B to this bit removes the CC41 from IDLE mode. A read access always returns 0.
CS2I	2	w	**CC42 IDLE mode clear** Writing a 1_B to this bit removes the CC42 from IDLE mode. A read access always returns 0.
CS3I	3	w	**CC43 IDLE mode clear** Writing a 1_B to this bit removes the CC43 from IDLE mode. A read access always returns 0.
SPRB	8	w	**Prescaler Run Bit Set** Writing a 1_B into this register sets the Run Bit of the prescaler. A read always returns 0.
0	[7:4], [31:9]	r	**Reserved** Read always returns 0.

GCSS

Through this register one can request a shadow transfer for the specific timer slice(s) and set the status bit for each of the compare channels.

GCSS
Global Channel Set **(0010_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0												S3STS	S2STS	S1STS	S0STS
r												w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	S3PSE	S3DSE	S3SE	0	S2PSE	S2DSE	S2SE	0	S1PSE	S1DSE	S1SE	0	S0PSE	S0DSE	S0SE
r	w	w	w	r	w	w	w	r	w	w	w	r	w	w	w

Field	Bits	Type	Description
S0SE	0	w	**Slice 0 shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S0SS field, enabling then a shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S0DSE	1	w	**Slice 0 Dither shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S0DSS field, enabling then a shadow transfer for the Dither compare value. A read always returns 0.
S0PSE	2	w	**Slice 0 Prescaler shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S0PSS field, enabling then a shadow transfer for the prescaler compare value. A read always returns 0.
S1SE	4	w	**Slice 1 shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S1SS field, enabling then a shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S1DSE	5	w	**Slice 1 Dither shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S1DSS field, enabling then a shadow transfer for the Dither compare value. A read always returns 0.

Field	Bits	Type	Description
S1PSE	6	w	**Slice 1 Prescaler shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S1PSS field, enabling then a shadow transfer for the prescaler compare value. A read always returns 0.
S2SE	8	w	**Slice 2 shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S2SS field, enabling then a shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S2DSE	9	w	**Slice 2 Dither shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S2DSS field, enabling then a shadow transfer for the Dither compare value. A read always returns 0.
S2PSE	10	w	**Slice 2 Prescaler shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S2PSS field, enabling then a shadow transfer for the prescaler compare value. A read always returns 0.
S3SE	12	w	**Slice 3 shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S3SS field, enabling then a shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S3DSE	13	w	**Slice 3 Dither shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S3DSS field, enabling then a shadow transfer for the Dither compare value. A read always returns 0.
S3PSE	14	w	**Slice 3 Prescaler shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S3PSS field, enabling then a shadow transfer for the prescaler compare value. A read always returns 0.
S0STS	16	w	**Slice 0 status bit set** Writing a 1_B into this field sets the status bit of slice 0 (**GCST**.CC40ST) to 1_B. A read always returns 0.

Field	Bits	Type	Description
S1STS	17	w	**Slice 1 status bit set** Writing a 1_B into this field sets the status bit of slice 1 (**GCST**.CC41ST) to 1_B. A read always returns 0.
S2STS	18	w	**Slice 2 status bit set** Writing a 1_B into this field sets the status bit of slice 2 (**GCST**.CC42ST) to 1_B. A read always returns 0.
S3STS	19	w	**Slice 3 status bit set** Writing a 1_B into this field sets the status bit of slice 3 (**GCST**.CC43ST) to 1_B. A read always returns 0.
0	3, 7, 11, 15, [31:20]	r	**Reserved** Read always returns 0.

GCSC

Through this register one can reset a shadow transfer request for the specific timer slice and clear the status bit for each the compare channels.

GCSC
Global Channel Clear **(0014_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0												**S3STC**	**S2STC**	**S1STC**	**S0STC**
r												w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**S3PSC**	**S3DSC**	**S3SC**	**0**	**S2PSC**	**S2DSC**	**S2SC**	**0**	**S1PSC**	**S1DSC**	**S1SC**	**0**	**S0PSC**	**S0DSC**	**S0SC**
r	w	w	w	r	w	w	w	r	w	w	w	r	w	w	w

Field	Bits	Type	Description
S0SC	0	w	**Slice 0 shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S0SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S0DSC	1	w	**Slice 0 Dither shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S0DSS field, canceling any pending shadow transfer for the Dither compare value. A read always returns 0.
S0PSC	2	w	**Slice 0 Prescaler shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S0PSS field, canceling any pending shadow transfer for the prescaler compare value. A read always returns 0.
S1SC	4	w	**Slice 1 shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S1SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S1DSC	5	w	**Slice 1 Dither shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S1DSS field, canceling any pending shadow transfer for the Dither compare value. A read always returns 0.
S1PSC	6	w	**Slice 1 Prescaler shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S1PSS field, canceling any pending shadow transfer for the prescaler compare value. A read always returns 0.
S2SC	8	w	**Slice 2 shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S2SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values. A read always returns 0.

Field	Bits	Type	Description
S2DSC	9	w	**Slice 2 Dither shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S2DSS field, canceling any pending shadow transfer for the Dither compare value. A read always returns 0.
S2PSC	10	w	**Slice 2 Prescaler shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S2PSS field, canceling any pending shadow transfer for the prescaler compare value. A read always returns 0.
S3SC	12	w	**Slice 3 shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S3SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S3DSC	13	w	**Slice 3 Dither shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S3DSS field, canceling any pending shadow transfer for the Dither compare value. A read always returns 0.
S3PSC	14	w	**Slice 3 Prescaler shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S3PSS field, canceling any pending shadow transfer for the prescaler compare value. A read always returns 0.
S0STC	16	w	**Slice 0 status bit clear** Writing a 1_B into this field clears the status bit of slice 0 (**GCST**.CC40ST) to 0_B. A read always returns 0.
S1STC	17	w	**Slice 1 status bit clear** Writing a 1_B into this field clears the status bit of slice 1 (**GCST**.CC41ST) to 0_B. A read always returns 0.
S2STC	18	w	**Slice 2 status bit clear** Writing a 1_B into this field clears the status bit of slice 2 (**GCST**.CC42ST) to 0_B. A read always returns 0.

Field	Bits	Type	Description
S3STC	19	w	**Slice 3 status bit clear** Writing a 1_B into this field clears the status bit of slice 3 (**GCST**.CC43ST) to 0_B. A read always returns 0.
0	3, 7, 11, 15, [31:20]	r	**Reserved** Read always returns 0.

GCST

This register holds the information of the shadow transfer requests and of each timer slice status bit.

GCST
Global Channel Status **(0018_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0												**CC43ST**	**CC42ST**	**CC41ST**	**CC40ST**
r												rh	rh	rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**S3PSS**	**S3DSS**	**S3SS**	**0**	**S2PSS**	**S2DSS**	**S2SS**	**0**	**S1PSS**	**S1DSS**	**S1SS**	**0**	**S0PSS**	**S0DSS**	**S0SS**
r	rh	rh	rh	r	rh	rh	rh	r	rh	rh	rh	r	rh	rh	rh

Field	Bits	Type	Description
S0SS	0	rh	**Slice 0 shadow transfer status** 0_B Shadow transfer has not been requested 1_B Shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S0DSS	1	rh	**Slice 0 Dither shadow transfer status** 0_B Dither shadow transfer has not been requested 1_B Dither shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.

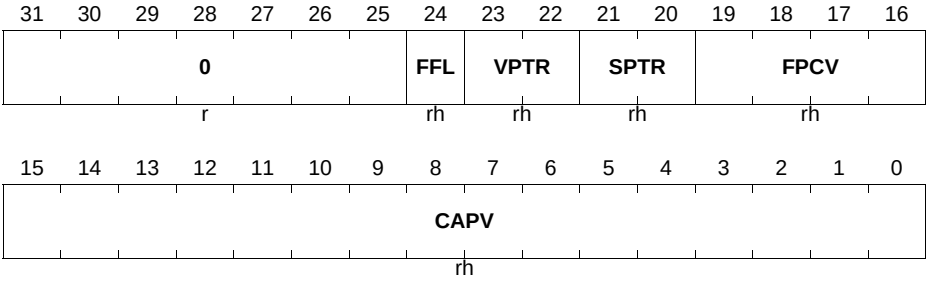
Field	Bits	Type	Description
S0PSS	2	rh	**Slice 0 Prescaler shadow transfer status** 0_B Prescaler shadow transfer has not been requested 1_B Prescaler shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S1SS	4	rh	**Slice 1 shadow transfer status** 0_B Shadow transfer has not been requested 1_B Shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S1DSS	5	rh	**Slice 1 Dither shadow transfer status** 0_B Dither shadow transfer has not been requested 1_B Dither shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S1PSS	6	rh	**Slice 1 Prescaler shadow transfer status** 0_B Prescaler shadow transfer has not been requested 1_B Prescaler shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S2SS	8	rh	**Slice 2 shadow transfer status** 0_B Shadow transfer has not been requested 1_B Shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S2DSS	9	rh	**Slice 2 Dither shadow transfer status** 0_B Dither shadow transfer has not been requested 1_B Dither shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.

Field	Bits	Type	Description
S2PSS	10	rh	**Slice 2 Prescaler shadow transfer status** 0_B Prescaler shadow transfer has not been requested 1_B Prescaler shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S3SS	12	rh	**Slice 3 shadow transfer status** 0_B Shadow transfer has not been requested 1_B Shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S3DSS	13	rh	**Slice 3 Dither shadow transfer status** 0_B Dither shadow transfer has not been requested 1_B Dither shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S3PSS	14	rh	**Slice 3 Prescaler shadow transfer status** 0_B Prescaler shadow transfer has not been requested 1_B Prescaler shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
CC40ST	16	rh	**Slice 0 status bit**
CC41ST	17	rh	**Slice 1 status bit**
CC42ST	18	rh	**Slice 2 status bit**
CC43ST	19	rh	**Slice 3 status bit**
0	3, 7, 11, 15, [31:20]	r	**Reserved** Read always returns 0.

ECRD

This register holds the information related to the extended capture mode.

ECRD
Extended Capture Mode Read **(0050_H)** **Reset Value: 00000000_H**



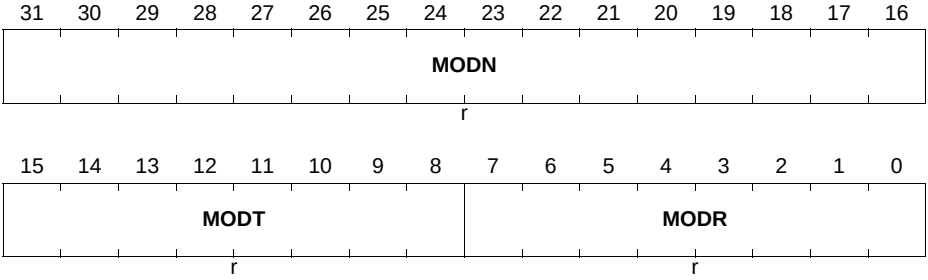
Field	Bits	Type	Description
CAPV	[15:0]	rh	**Timer Capture Value** This field contains the timer captured value
FPCV	[19:16]	rh	**Prescaler Capture value** This field contains the value of the prescaler clock division associated with the specific CAPV field
SPTR	[21:20]	rh	**Slice pointer** This field indicates the slice index in which the value was captured. 00_B CC40 01_B CC41 10_B CC42 11_B CC43
VPTR	[23:22]	rh	**Capture register pointer** This field indicates the capture register index in which the value was captured. 00_B Capture register 0 01_B Capture register 1 10_B Capture register 2 11_B Capture register 3
FFL	24	rh	**Full Flag** This bit indicates if the associated capture register contains a value. 0_B No new value was captured into this register 1_B A new value has been captured into this register

Field	Bits	Type	Description
0	[31:25]	r	**Reserved** Read always returns 0.

MIDR

This register contains the module identification number.

MIDR
Module Identification **(0080_H)** **Reset Value: $00A6C0XX_H$**



Field	Bits	Type	Description
MODR	[7:0]	r	**Module Revision** This bit field indicates the revision number of the module implementation (depending on the design step). The given value of 00_H is a placeholder for the actual number.
MODT	[15:8]	r	**Module Type**
MODN	[31:16]	r	**Module Number**

22.7.2 Slice (CC4y) Registers

CC4yINS

The register contains the configuration for the input selector.

CC4yINS (y = 0 - 3)
Input Selector Configuration **(0100_H + 0100_H * y)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	LPF2M		LPF1M		LPF0M		EV2LM	EV1LM	EV0LM	EV2EM		EV1EM		EV0EM	
r	rw		rw		rw		rw	rw	rw	rw		rw		rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				EV2IS				EV1IS				EV0IS			
r				rw				rw				rw			

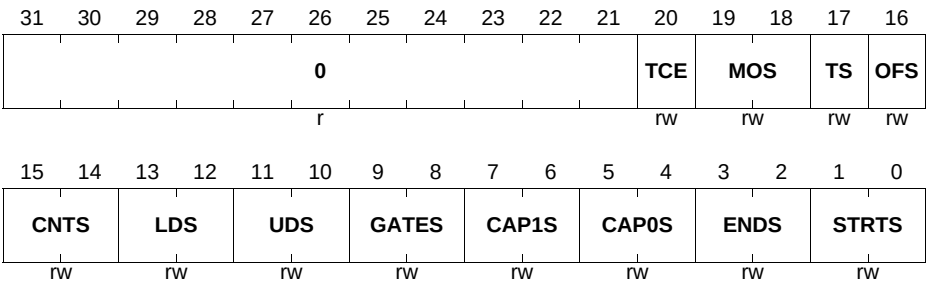
Field	Bits	Type	Description
EV0IS	[3:0]	rw	**Event 0 signal selection** This field selects which pins is used for the event 0. 0000_B CCU4x.INyA 0001_B CCU4x.INyB 0010_B CCU4x.INyC 0011_B CCU4x.INyD 0100_B CCU4x.INyE 0101_B CCU4x.INyF 0110_B CCU4x.INyG 0111_B CCU4x.INyH 1000_B CCU4x.INyI 1001_B CCU4x.INyJ 1010_B CCU4x.INyK 1011_B CCU4x.INyL 1100_B CCU4x.INyM 1101_B CCU4x.INyN 1110_B CCU4x.INyO 1111_B CCU4x.INyP
EV1IS	[7:4]	rw	**Event 1 signal selection** Same as EV0IS description
EV2IS	[11:8]	rw	**Event 2 signal selection** Same as EV0IS description

Field	Bits	Type	Description
EV0EM	[17:16]	rw	**Event 0 Edge Selection** 00_B No action 01_B Signal active on rising edge 10_B Signal active on falling edge 11_B Signal active on both edges
EV1EM	[19:18]	rw	**Event 1 Edge Selection** Same as EV0EM description
EV2EM	[21:20]	rw	**Event 2 Edge Selection** Same as EV0EM description
EV0LM	22	rw	**Event 0 Level Selection** 0_B Active on HIGH level 1_B Active on LOW level
EV1LM	23	rw	**Event 1 Level Selection** Same as EV0LM description
EV2LM	24	rw	**Event 2 Level Selection** Same as EV0LM description
LPF0M	[26:25]	rw	**Event 0 Low Pass Filter Configuration** This field sets the number of consecutive counts for the Low Pass Filter of Event 0. The input signal value needs to remain stable for this number of counts (f_{CCU4}), so that a level/transition is accepted. 00_B LPF is disabled 01_B 3 clock cycles of f_{CCU4} 10_B 5 clock cycles of f_{CCU4} 11_B 7 clock cycles of f_{CCU4}
LPF1M	[28:27]	rw	**Event 1 Low Pass Filter Configuration** Same description as LPF0M
LPF2M	[30:29]	rw	**Event 2 Low Pass Filter Configuration** Same description as LPF0M
0	[15:12], 31	r	**Reserved** Read always returns 0.

CC4yCMC

The register contains the configuration for the connection matrix.

CC4yCMC (y = 0 - 3)
Connection Matrix Control **($0104_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
STRTS	[1:0]	rw	**External Start Functionality Selector** Selects the Event that is going to be linked with the external start functionality. 00_B External Start Function deactivated 01_B External Start Function triggered by Event 0 10_B External Start Function triggered by Event 1 11_B External Start Function triggered by Event 2
ENDS	[3:2]	rw	**External Stop Functionality Selector** Selects the Event that is going to be linked with the external stop functionality. 00_B External Stop Function deactivated 01_B External Stop Function triggered by Event 0 10_B External Stop Function triggered by Event 1 11_B External Stop Function triggered by Event 2
CAP0S	[5:4]	rw	**External Capture 0 Functionality Selector** Selects the Event that is going to be linked with the external capture for capture registers number 1 and 0. 00_B External Capture 0 Function deactivated 01_B External Capture 0 Function triggered by Event 0 10_B External Capture 0 Function triggered by Event 1 11_B External Capture 0 Function triggered by Event 2

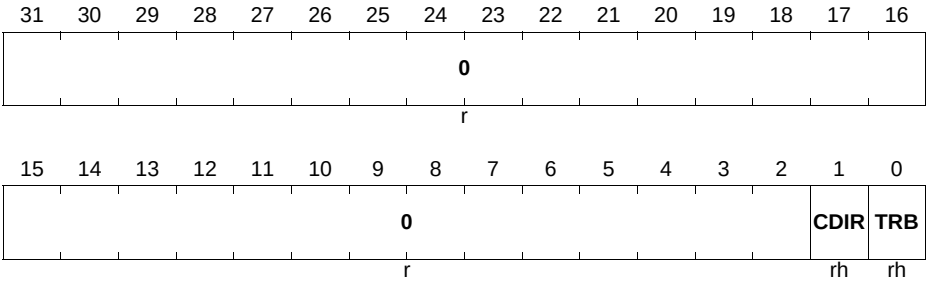
Field	Bits	Type	Description
CAP1S	[7:6]	rw	**External Capture 1 Functionality Selector** Selects the Event that is going to be linked with the external capture for capture registers number 3 and 2. 00_B External Capture 1 Function deactivated 01_B External Capture 1 Function triggered by Event 0 10_B External Capture 1 Function triggered by Event 1 11_B External Capture 1 Function triggered by Event 2
GATES	[9:8]	rw	**External Gate Functionality Selector** Selects the Event that is going to be linked with the counter gating function. This function is used to gate the timer increment/decrement procedure. 00_B External Gating Function deactivated 01_B External Gating Function triggered by Event 0 10_B External Gating Function triggered by Event 1 11_B External Gating Function triggered by Event 2
UDS	[11:10]	rw	**External Up/Down Functionality Selector** Selects the Event that is going to be linked with the Up/Down counting direction control. 00_B External Up/Down Function deactivated 01_B External Up/Down Function triggered by Event 0 10_B External Up/Down Function triggered by Event 1 11_B External Up/Down Function triggered by Event 2
LDS	[13:12]	rw	**External Timer Load Functionality Selector** Selects the Event that is going to be linked with the timer load function. 00_B - External Load Function deactivated 01_B - External Load Function triggered by Event 0 10_B - External Load Function triggered by Event 1 11_B - External Load Function triggered by Event 2

Field	Bits	Type	Description
CNTS	[15:14]	rw	**External Count Selector** Selects the Event that is going to be linked with the count function. The counter is going to be increment/decremented each time that a specific transition on the event is detected. 00_B External Count Function deactivated 01_B External Count Function triggered by Event 0 10_B External Count Function triggered by Event 1 11_B External Count Function triggered by Event 2
OFS	16	rw	**Override Function Selector** This field enables the ST bit override functionality. 0_B Override functionality disabled 1_B Status bit trigger override connected to Event 1; Status bit value override connected to Event 2
TS	17	rw	**Trap Function Selector** This field enables the trap functionality. 0_B Trap function disabled 1_B TRAP function connected to Event 2
MOS	[19:18]	rw	**External Modulation Functionality Selector** Selects the Event that is going to be linked with the external modulation function. 00_B - Modulation Function deactivated 01_B - Modulation Function triggered by Event 0 10_B - Modulation Function triggered by Event 1 11_B - Modulation Function triggered by Event 2
TCE	20	rw	**Timer Concatenation Enable** This bit enables the timer concatenation with the previous slice. 0_B Timer concatenation is disabled 1_B Timer concatenation is enabled *Note: In CC40 this field doesn't exist. This is a read only reserved field. Read access always returns 0.*
0	[31:21]	r	**Reserved** A read always returns 0

CC4yTCST

The register holds the status of the timer (running/stopped) and the information about the counting direction (up/down).

CC4yTCST (y = 0 - 3)
Slice Timer Status **($0108_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
TRB	0	rh	**Timer Run Bit** This field indicates if the timer is running. 0_B Timer is stopped 1_B Timer is running
CDIR	1	rh	**Timer Counting Direction** This filed indicates if the timer is being increment or decremented 0_B Timer is counting up 1_B Timer is counting down
0	[31:2]	r	**Reserved** Read always returns 0

CC4yTCSET

Through this register it is possible to start the timer.

CC4yTCSET (y = 0 - 3)
Slice Timer Run Set **($010C_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0															**TRBS**
r															w

Field	Bits	Type	Description
TRBS	0	w	**Timer Run Bit set** Writing a 1_B into this field sets the run bit of the timer. Read always returns 0.
0	[31:1]	r	**Reserved** Read always returns 0

CC4yTCCLR

Through this register it is possible to stop and clear the timer, and clearing also the dither counter

CC4yTCCLR (y = 0 - 3)
Slice Timer Clear **($0110_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0													**DITC**	**TCC**	**TRBC**
r													w	w	w

Field	Bits	Type	Description
TRBC	0	w	**Timer Run Bit Clear** Writing a 1_B into this field clears the run bit of the timer. The timer is not cleared. Read always returns 0.
TCC	1	w	**Timer Clear** Writing a 1_B into this field clears the timer to 0000_H. Read always returns 0.
DITC	2	w	**Dither Counter Clear** Writing a 1_B into this field clears the dither counter to 0_H. Read always returns 0.
0	[31:3]	r	**Reserved** Read always returns 0

CC4yTC

This register holds the several possible configurations for the timer operation.

CC4yTC (y = 0 - 3)
Slice Timer Control **($0114_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0						MCME	EMT	EMS	TRPSW	TRPSE	0			TRAPE	FPE
r						rw	rw	rw	rw	rw	r			rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIM	DITHE		CCS	SCE	STRM	ENDM		0	CAPC		ECM	CMOD	CLST	TSSM	TCM
rw	rw		rw	rw	rw	rw		r	rw		rw	rh	rw	rw	rw

Field	Bits	Type	Description
TCM	0	rw	**Timer Counting Mode** This field controls the actual counting scheme of the timer. 0_B Edge aligned mode 1_B Center aligned mode *Note: When using an external signal to control the counting direction, the counting scheme is always edge aligned.*
TSSM	1	rw	**Timer Single Shot Mode** This field controls the single shot mode. This is applicable in edge and center aligned modes. 0_B Single shot mode is disabled 1_B Single shot mode is enabled
CLST	2	rw	**Shadow Transfer on Clear** Setting this bit to 1_B enables a shadow transfer when a timer clearing action is performed. Notice that the shadow transfer enable bitfields on the **GCST** register still need to be set to 1_B via software.
CMOD	3	rh	**Capture Compare Mode** This field indicates in which mode the slice is operating. The default value is compare mode. The capture mode is automatically set by the HW when an external signal is mapped to a capture trigger. 0_B Compare Mode 1_B Capture Mode

Field	Bits	Type	Description
ECM	4	rw	**Extended Capture Mode** This field control the Capture mode of the specific slice. It only has effect if the CMOD bit is 1_B. 0_B Normal Capture Mode. Clear of the Full Flag of each capture register is done by accessing the registers individually only. 1_B Extended Capture Mode. Clear of the Full Flag of each capture register is done not only by accessing the individual registers but also by accessing the **ECRD** register. When reading the **ECRD** register, only the capture register register full flag pointed by the **ECRD**.VPTR is cleared
CAPC	[6:5]	rw	**Clear on Capture Control** 00_B Timer is never cleared on a capture event 01_B Timer is cleared on a capture event into capture registers 2 and 3. (When SCE = 1_B, Timer is always cleared in a capture event) 10_B Timer is cleared on a capture event into capture registers 0 and 1. (When SCE = 1_B, Timer is always cleared in a capture event) 11_B Timer is always cleared in a capture event.
ENDM	[9:8]	rw	**Extended Stop Function Control** This field controls the extended functions of the external Stop signal. 00_B Clears the timer run bit only (default stop) 01_B Clears the timer only (flush) 10_B Clears the timer and run bit (flush/stop) 11_B Reserved *Note: When using an external up/down signal the flush operation sets the timer with zero if the counter is counting up and with the Period value if the counter is being decremented.*

Field	Bits	Type	Description
STRM	10	rw	**Extended Start Function Control** This field controls the extended functions of the external Start signal. 0_B Sets run bit only (default start) 1_B Clears the timer and sets run bit (flush/start) *Note: When using an external up/down signal the flush operation sets the timer with zero if the counter is being incremented and with the Period value if the counter is being decremented.*
SCE	11	rw	**Equal Capture Event enable** 0_B Capture into **CC4yC0V**/**CC4yC1V** registers control by CCycapt0 and capture into **CC4yC3V**/**CC4yC2V** control by CCycapt1 1_B Capture into **CC4yC0V**/**CC4yC1V** and **CC4yC3V**/**CC4yC2V** control by CCycapt1
CCS	12	rw	**Continuous Capture Enable** 0_B The capture into a specific capture register is done with the rules linked with the full flags, described at **Section 22.2.7.6**. 1_B The capture into the capture registers is always done regardless of the full flag status (even if the register has not been read back).
DITHE	[14:13]	rw	**Dither Enable** This field controls the dither mode for the slice. See **Section 22.2.10**. 00_B Dither is disabled 01_B Dither is applied to the Period 10_B Dither is applied to the Compare 11_B Dither is applied to the Period and Compare
DIM	15	rw	**Dither input selector** This fields selects if the dither control signal is connected to the dither logic of the specific slice of is connected to the dither logic of slice 0. Notice that even if this field is set to 1_B, the field DITHE still needs to be programmed. 0_B Slice is using its own dither unit 1_B Slice is connected to the dither unit of slice 0.

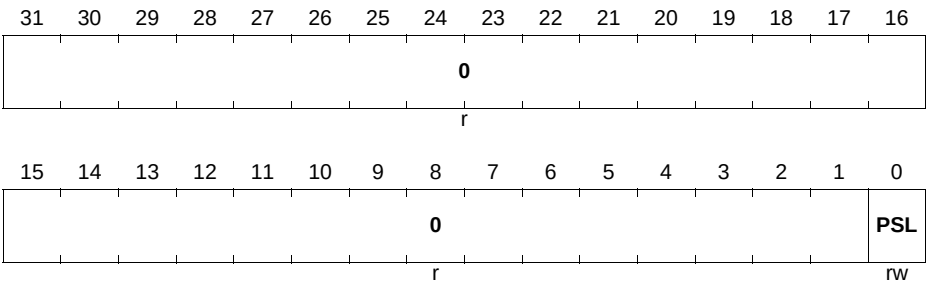
Field	Bits	Type	Description
FPE	16	rw	**Floating Prescaler enable** Setting this bit to 1_B enables the floating prescaler mode. 0_B Floating prescaler mode is disabled 1_B Floating prescaler mode is enabled
TRAPE	17	rw	**TRAP enable** Setting this bit to 1_B enables the TRAP action at the output pin. After mapping an external signal to the TRAP functionality, the user must set this field to 1_B to activate the effect of the TRAP on the output pin. Writing a 0_B into this field disables the effect of the TRAP function regardless of the state of the input signal. 0_B TRAP functionality has no effect on the output 1_B TRAP functionality affects the output
TRPSE	21	rw	**TRAP Synchronization Enable** Writing a 1_B into this bit enables a synchronous exiting with the PWM signal of the trap state. 0_B Exiting from TRAP state isn't synchronized with the PWM signal 1_B Exiting from TRAP state is synchronized with the PWM signal
TRPSW	22	rw	**TRAP State Clear Control** 0_B The slice exits the TRAP state automatically when the TRAP condition is not present 1_B The TRAP state can only be exited by a SW request.
EMS	23	rw	**External Modulation Synchronization** Setting this bit to 1_B enables the synchronization of the external modulation functionality with the PWM period. 0_B External Modulation functionality is not synchronized with the PWM signal 1_B External Modulation functionality is synchronized with the PWM signal

Field	Bits	Type	Description
EMT	24	rw	**External Modulation Type** This field selects if the external modulation event is clearing the CC4yST bit or if is gating the outputs. 0_B External Modulation functionality is clearing the CC4yST bit. 1_B External Modulation functionality is gating the outputs.
MCME	25	rw	**Multi Channel Mode Enable** 0_B Multi Channel Mode is disabled 1_B Multi Channel Mode is enabled
0	7, [20:18], [31:26]	r	**Reserved** Read always returns 0

CC4yPSL

This register holds the configuration for the output passive level control.

CC4yPSL (y = 0 - 3)
Passive Level Config **($0118_H + 0100_H * y$)** **Reset Value: 00000000_H**



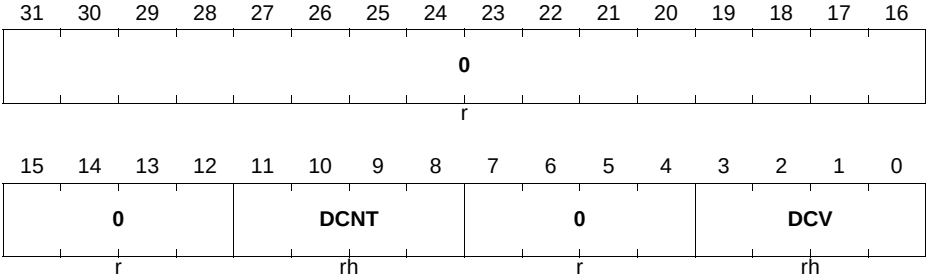
Field	Bits	Type	Description
PSL	0	rw	**Output Passive Level** This field controls the passive level of the output pin. 0_B Passive Level is LOW 1_B Passive Level is HIGH A write always addresses the shadow register, while a read always returns the current used value.

Field	Bits	Type	Description
0	[31:1]	r	**Reserved** A read access always returns 0

CC4yDIT

This register holds the current dither compare and dither counter values.

CC4yDIT (y = 0 - 3)
Dither Config **($011C_H + 0100_H * y$)** **Reset Value: 00000000_H**

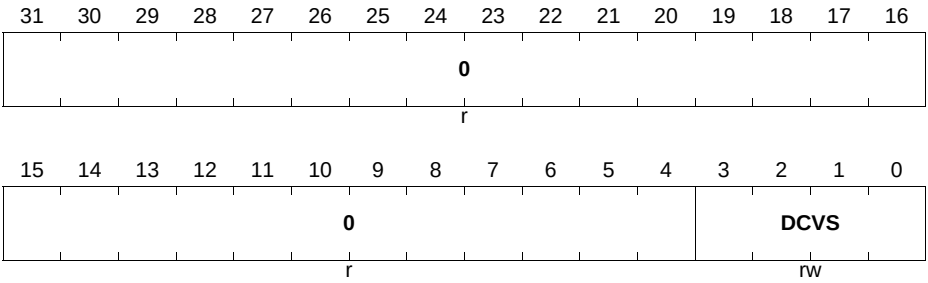


Field	Bits	Type	Description
DCV	[3:0]	rh	**Dither compare Value** This field contains the value used for the dither comparison. This value is updated when a shadow transfer occurs with the **CC4yDITS**.DCVS.
DCNT	[11:8]	rh	**Dither counter actual value**
0	[7:4], [31:12]	r	**Reserved** Read always returns 0.

CC4yDITS

This register contains the value that is going to be loaded into the **CC4yDIT**.DCV when the next shadow transfer occurs.

CC4yDITS (y = 0 - 3)
Dither Shadow Register **($0120_H + 0100_H * y$)** **Reset Value: 00000000_H**

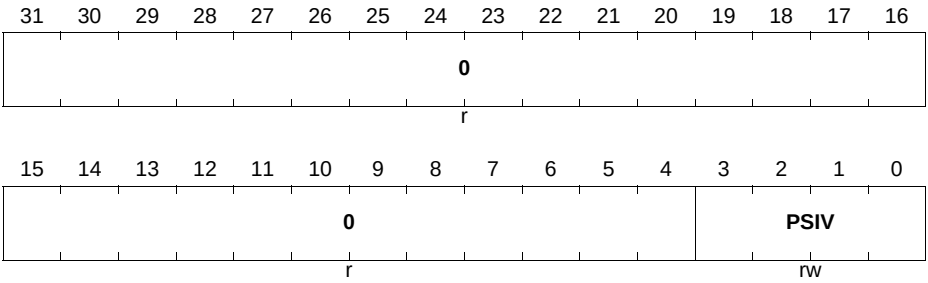


Field	Bits	Type	Description
DCVS	[3:0]	rw	**Dither Shadow Compare Value** This field contains the value that is going to be set on the dither compare value, **CC4yDIT**.DCV, within the next shadow transfer.
0	[31:4]	r	**Reserved** Read always returns 0.

CC4yPSC

This register contains the value that is loaded into the prescaler during restart.

CC4yPSC (y = 0 - 3)
Prescaler Control **($0124_H + 0100_H * y$)** **Reset Value: 00000000_H**

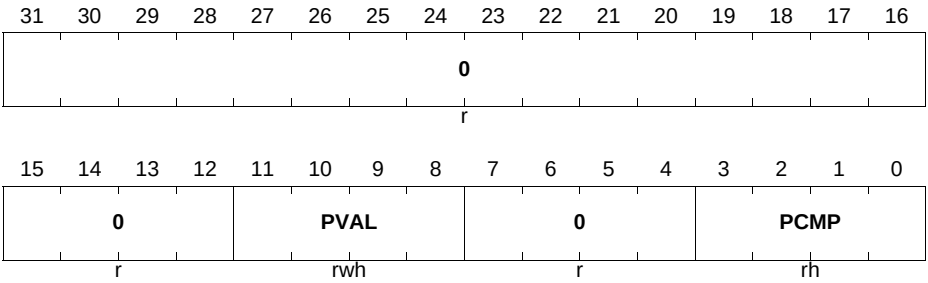


Field	Bits	Type	Description
PSIV	[3:0]	rw	**Prescaler Initial Value** This field contains the value that is applied to the Prescaler at startup. When floating prescaler mode is used, this value is applied when a timer compare match AND prescaler compare match occurs or when a capture event is triggered.
0	[31:4]	r	**Reserved** Read always returns 0.

CC4yFPC

This register contains the value used for the floating prescaler compare and the actual prescaler division value.

CC4yFPC (y = 0 - 3)
Floating Prescaler Control **(0128_H + 0100_H * y)** **Reset Value: 00000000_H**



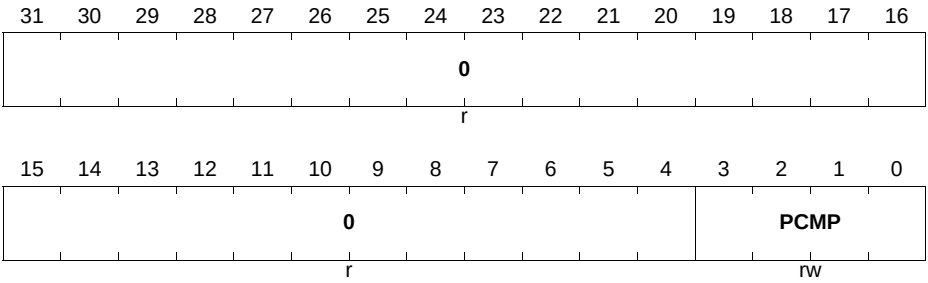
Field	Bits	Type	Description
PCMP	[3:0]	rh	**Floating Prescaler Compare Value** This field contains comparison value used in floating prescaler mode. The comparison is triggered by the Timer Compare match event. See **Section 22.2.11.2**.

Field	Bits	Type	Description
PVAL	[11:8]	rwh	**Actual Prescaler Value** See **Table 22-7**. Writing into this register is only possible when the prescaler is stopped. When the floating prescaler mode is not used, this value is equal to the **CC4yPSC**.PSIV.
0	[7:4], [15:12], [31:16]	r	**Reserved** Read always returns 0.

CC4yFPCS

This register contains the value that is going to be transferred to the **CC4yFPC**.PCMP field within the next shadow transfer update.

CC4yFPCS (y = 0 - 3)
Floating Prescaler Shadow **($012C_H + 0100_H * y$)** **Reset Value: 00000000_H**

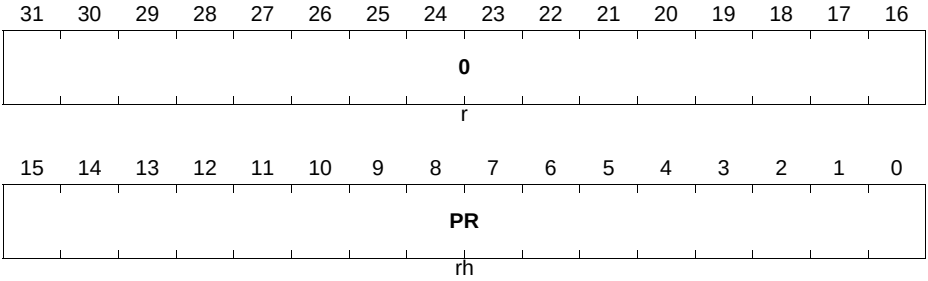


Field	Bits	Type	Description
PCMP	[3:0]	rw	**Floating Prescaler Shadow Compare Value** This field contains the value that is going to be set on the **CC4yFPC**.PCMP within the next shadow transfer. See **Table 22-7**.
0	[31:4]	r	**Reserved** Read always returns 0.

CC4yPR

This register contains the actual value for the timer period.

CC4yPR (y = 0 - 3)
Timer Period Value **($0130_H + 0100_H * y$)** **Reset Value: 00000000_H**

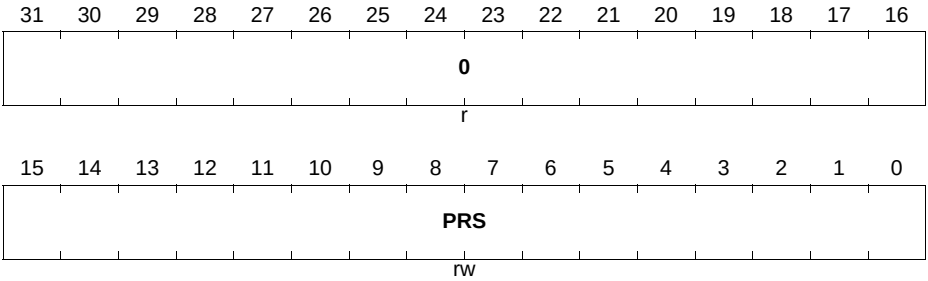


Field	Bits	Type	Description
PR	[15:0]	rh	**Period Register** Contains the value of the timer period. *Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 2 and 3, a read always returns 0.*
0	[31:16]	r	**Reserved** A read always returns 0.

CC4yPRS

This register contains the value for the timer period that is going to be transferred into the **CC4yPR**.PR field when the next shadow transfer occurs.

CC4yPRS (y = 0 - 3)
Timer Shadow Period Value **($0134_H + 0100_H * y$)** **Reset Value: 00000000_H**

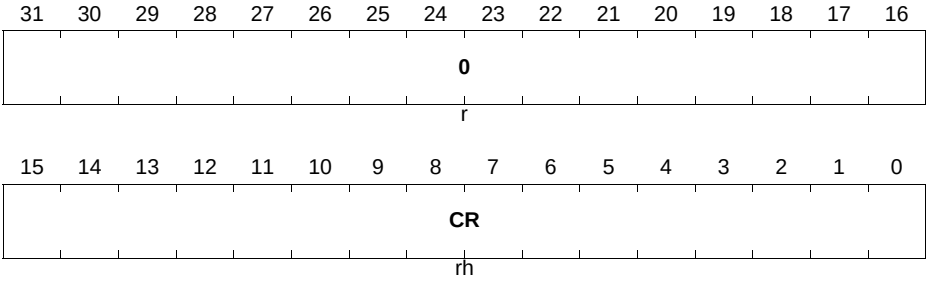


Field	Bits	Type	Description
PRS	[15:0]	rw	**Period Register** Contains the value of the timer period, that is going to be passed into the **CC4yPR**.PR field when the next shadow transfer occurs. *Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 2 and 3, the PRS is not accessible for writing. A read always returns 0.*
0	[31:16]	r	**Reserved** A read always returns 0.

CC4yCR

This register contains the value for the timer comparison.

CC4yCR (y = 0 - 3)
Timer Compare Value **(0138_H + 0100_H * y)** **Reset Value: 00000000_H**

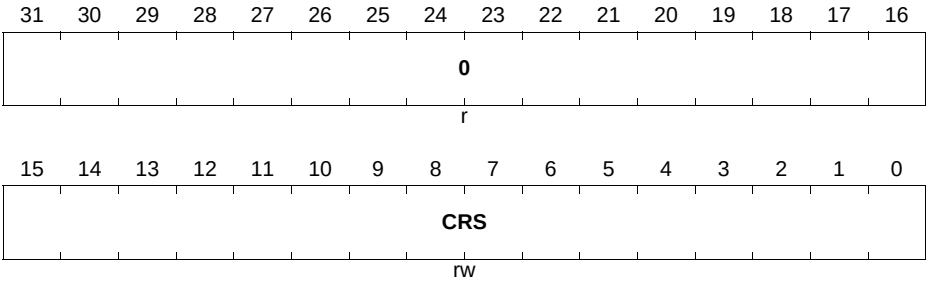


Field	Bits	Type	Description
CR	[15:0]	rh	**Compare Register** Contains the value for the timer comparison. *Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 0 and 1, a read always returns 0.*
0	[31:16]	r	**Reserved** A read always returns 0.

CC4yCRS

This register contains the value that is going to be loaded into the **CC4yCR**.CR field when the next shadow transfer occurs.

CC4yCRS (y = 0 - 3)
Timer Shadow Compare Value ($013C_H$ + 0100_H * y) **Reset Value: 00000000_H**

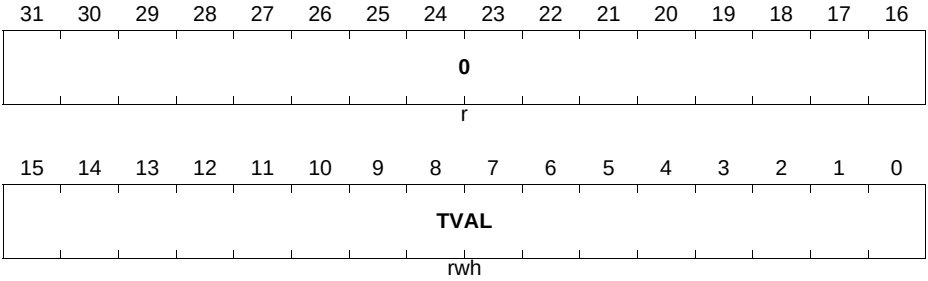


Field	Bits	Type	Description
CRS	[15:0]	rw	**Compare Register** Contains the value for the timer comparison, that is going to be passed into the **CC4yCR**.CR field when the next shadow transfer occurs. *Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 0 and 1, a read always returns 0.*
0	[31:16]	r	**Reserved** A read always returns 0.

CC4yTIMER

This register contains the current value of the timer.

CC4yTIMER (y = 0 - 3)
Timer Value **($0170_H + 0100_H * y$)** **Reset Value: 00000000_H**

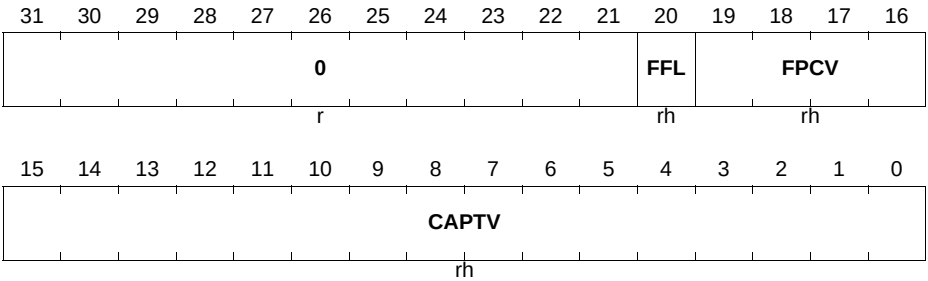


Field	Bits	Type	Description
TVAL	[15:0]	rwh	**Timer Value** This field contains the actual value of the timer. A write access is only possible when the timer is stopped.
0	[31:16]	r	**Reserved** A read access always returns 0

CC4yC0V

This register contains the values associated with the Capture 0 field.

CC4yC0V (y = 0 - 3)
Capture Register 0 **($0174_H + 0100_H * y$)** **Reset Value: 00000000_H**

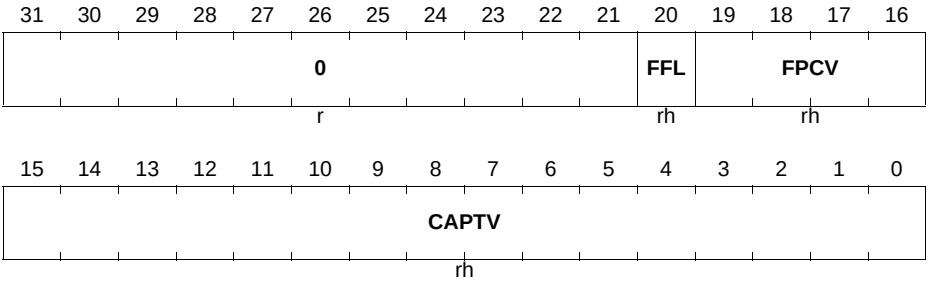


Field	Bits	Type	Description
CAPTV	[15:0]	rh	**Capture Value** This field contains the capture register 0 value. See **Figure 22-25**. In compare mode a read access always returns 0.
FPCV	[19:16]	rh	**Prescaler Value** This field contains the prescaler value at the time of the capture event into the capture register 0. In compare mode a read access always returns 0.
FFL	20	rh	**Full Flag** This bit indicates if a new value was capture into the capture register 0 after the last read access. See **Figure 22-25**. In compare mode a read access always returns 0. 0_B No new value was captured into the specific capture register 1_B A new value was captured into the specific register
0	[31:21]	r	**Reserved** A read always returns 0

CC4yC1V

This register contains the values associated with the Capture 1 field.

CC4yC1V (y = 0 - 3)
Capture Register 1 **($0178_H + 0100_H * y$)** **Reset Value: 00000000_H**

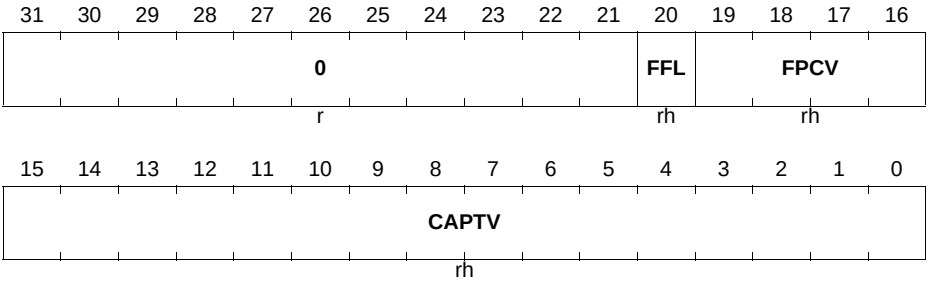


Field	Bits	Type	Description
CAPTV	[15:0]	rh	**Capture Value** This field contains the capture register 1 value. See **Figure 22-25**. In compare mode a read access always returns 0.
FPCV	[19:16]	rh	**Prescaler Value** This field contains the prescaler value at the time of the capture event into the capture register 1. In compare mode a read access always returns 0.
FFL	20	rh	**Full Flag** This bit indicates if a new value was capture into the capture register 1 after the last read access. See **Figure 22-25**. In compare mode a read access always returns 0. 0_B No new value was captured into the specific capture register 1_B A new value was captured into the specific register
0	[31:21]	r	**Reserved** A read always returns 0

CC4yC2V

This register contains the values associated with the Capture 2 field.

CC4yC2V (y = 0 - 3)
Capture Register 2 **($017C_H + 0100_H * y$)** **Reset Value: 00000000_H**

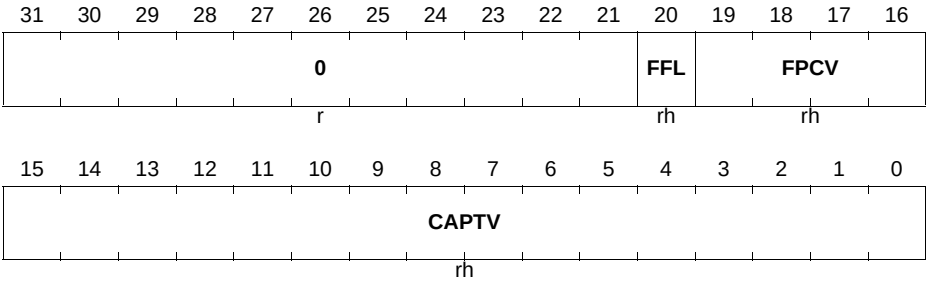


Field	Bits	Type	Description
CAPTV	[15:0]	rh	**Capture Value** This field contains the capture register 2 value. See **Figure 22-25**. In compare mode a read access always returns 0.
FPCV	[19:16]	rh	**Prescaler Value** This field contains the prescaler value at the time of the capture event into the capture register 2. In compare mode a read access always returns 0.
FFL	20	rh	**Full Flag** This bit indicates if a new value was capture into the capture register 2 after the last read access. See **Figure 22-25**. In compare mode a read access always returns 0. 0_B No new value was captured into the specific capture register 1_B A new value was captured into the specific register
0	[31:21]	r	**Reserved** A read always returns 0

CC4yC3V

This register contains the values associated with the Capture 3 field.

CC4yC3V (y = 0 - 3)
Capture Register 3 **($0180_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
CAPTV	[15:0]	rh	**Capture Value** This field contains the capture register 3 value. See **Figure 22-25**. In compare mode a read access always returns 0.
FPCV	[19:16]	rh	**Prescaler Value** This field contains the prescaler value at the time of the capture event into the capture register 3. In compare mode a read access always returns 0.
FFL	20	rh	**Full Flag** This bit indicates if a new value was capture into the capture register 3 after the last read access. See **Figure 22-25**. In compare mode a read access always returns 0. 0_B No new value was captured into the specific capture register 1_B A new value was captured into the specific register
0	[31:21]	r	**Reserved** A read always returns 0

CC4yINTS

This register contains the status of all interrupt sources.

CC4yINTS (y = 0 - 3)
Interrupt Status **($01A0_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				TRPF	E2AS	E1AS	E0AS	0				CMDS	CMUS	OMDS	PMUS
r				rh	rh	rh	rh	r				rh	rh	rh	rh

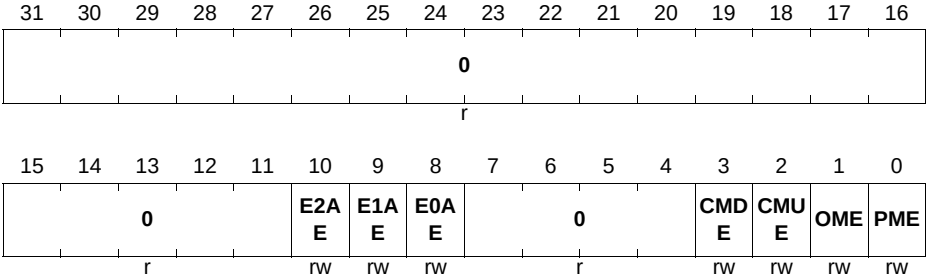
Field	Bits	Type	Description
PMUS	0	rh	**Period Match while Counting Up** 0_B Period match while counting up not detected 1_B Period match while counting up detected
OMDS	1	rh	**One Match while Counting Down** 0_B One match while counting down not detected 1_B One match while counting down detected
CMUS	2	rh	**Compare Match while Counting Up** 0_B Compare match while counting up not detected 1_B Compare match while counting up detected
CMDS	3	rh	**Compare Match while Counting Down** 0_B Compare match while counting down not detected 1_B Compare match while counting down detected
E0AS	8	rh	**Event 0 Detection Status** Depending on the user selection on the **CC4yINS**.EV0EM, this bit can be set when a rising, falling or both transitions are detected. 0_B Event 0 not detected 1_B Event 0 detected
E1AS	9	rh	**Event 1 Detection Status** Depending on the user selection on the **CC4yINS**.EV1EM, this bit can be set when a rising, falling or both transitions are detected. 0_B Event 1 not detected 1_B Event 1 detected
E2AS	10	rh	**Event 2 Detection Status** Depending on the user selection on the **CC4yINS**.EV1EM, this bit can be set when a rising, falling or both transitions are detected. 0_B Event 2 not detected 1_B Event 2 detected *Note: If this event is linked with the TRAP function, this field is automatically cleared when the slice exits the Trap State.*
TRPF	11	rh	**Trap Flag Status** This field contains the status of the Trap Flag.

Field	Bits	Type	Description
0	[7:4], [31:12]	r	**Reserved** A read always returns 0.

CC4yINTE

Through this register it is possible to enable or disable the specific interrupt source(s).

CC4yINTE (y = 0 - 3)
Interrupt Enable Control **($01A4_H + 0100_H * y$)** **Reset Value: 00000000_H**



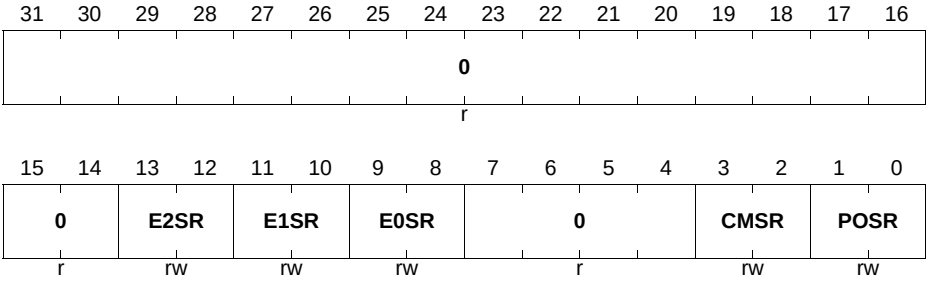
Field	Bits	Type	Description
PME	0	rw	**Period match while counting up enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time a period match while counting up occurs. 0_B Period Match interrupt is disabled 1_B Period Match interrupt is enabled
OME	1	rw	**One match while counting down enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time an one match while counting down occurs. 0_B One Match interrupt is disabled 1_B One Match interrupt is enabled

Field	Bits	Type	Description
CMUE	2	rw	**Compare match while counting up enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time a compare match while counting up occurs. 0_B Compare Match while counting up interrupt is disabled 1_B Compare Match while counting up interrupt is enabled
CMDE	3	rw	**Compare match while counting down enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time a compare match while counting down occurs. 0_B Compare Match while counting down interrupt is disabled 1_B Compare Match while counting down interrupt is enabled
E0AE	8	rw	**Event 0 interrupt enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time that Event 0 is detected. 0_B Event 0 detection interrupt is disabled 1_B Event 0 detection interrupt is enabled
E1AE	9	rw	**Event 1 interrupt enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time that Event 1 is detected. 0_B Event 1 detection interrupt is disabled 1_B Event 1 detection interrupt is enabled
E2AE	10	rw	**Event 2 interrupt enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time that Event 2 is detected. 0_B Event 2 detection interrupt is disabled 1_B Event 2 detection interrupt is enabled
0	[7:4], [31:11]	r	**Reserved** A read always returns 0

CC4ySRS

Through this register it is possible to select to which service request line each interrupt source is forwarded.

CC4ySRS (y = 0 - 3)
Service Request Selector **($01A8_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
POSR	[1:0]	rw	**Period/One match Service request selector** This field selects to which slice Service request line, the interrupt(s) generated by the Period match while counting up and One match while counting down are going to be forward. 00_B Forward to CC4ySR0 01_B Forward to CC4ySR1 10_B Forward to CC4ySR2 11_B Forward to CC4ySR3
CMSR	[3:2]	rw	**Compare match Service request selector** This field selects to which slice Service request line, the interrupt(s) generated by the Compare match while counting up and Compare match while counting down are going to be forward. 00_B Forward to CC4ySR0 01_B Forward to CC4ySR1 10_B Forward to CC4ySR2 11_B Forward to CC4ySR3
E0SR	[9:8]	rw	**Event 0 Service request selector** This field selects to which slice Service request line, the interrupt generated by the Event 0 detection is going to be forward. 00_B Forward to CC4ySR0 01_B Forward to CC4ySR1 10_B Forward to CC4ySR2 11_B Forward to CC4ySR3

Field	Bits	Type	Description
E1SR	[11:10]	rw	**Event 1 Service request selector** This field selects to which slice Service request line, the interrupt generated by the Event 1detection is going to be forward. 00_B Forward to CC4ySR0 01_B Forward to CC4ySR1 10_B Forward to CC4ySR2 11_B Forward to CC4ySR3
E2SR	[13:12]	rw	**Event 2 Service request selector** This field selects to which slice Service request line, the interrupt generated by the Event 2 detection is going to be forward. 00_B Forward to CC4ySR0 01_B Forward to CC4ySR1 10_B Forward to CC4ySR2 11_B Forward to CC4ySR3
0	[7:4], [31:14]	r	**Reserved** Read always returns 0.

CC4ySWS

Through this register it is possible for the SW to set a specific interrupt status flag.

CC4ySWS (y = 0 - 3)
Interrupt Status Set **($01AC_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				STRPF	SE2A	SE1A	SE0A	0				SCMD	SCMU	SOM	SPM
r				w	w	w	w	r				w	w	w	w

Field	Bits	Type	Description
SPM	0	w	**Period match while counting up set** Writing a 1_B into this field sets the **CC4yINTS**.PMUS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SOM	1	w	**One match while counting down set** Writing a 1_B into this bit sets the **CC4yINTS**.OMDS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SCMU	2	w	**Compare match while counting up set** Writing a 1_B into this field sets the **CC4yINTS**.CMUS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SCMD	3	w	**Compare match while counting down set** Writing a 1_B into this bit sets the **CC4yINTS**.CMDS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SE0A	8	w	**Event 0 detection set** Writing a 1_B into this bit sets the **CC4yINTS**.E0AS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SE1A	9	w	**Event 1 detection set** Writing a 1_B into this bit sets the **CC4yINTS**.E1AS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SE2A	10	w	**Event 2 detection set** Writing a 1_B into this bit sets the **CC4yINTS**.E2AS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
STRPF	11	w	**Trap Flag status set** Writing a 1_B into this bit sets the **CC4yINTS**.TRPF bit. A read always returns 0.
0	[7:4], [31:12]	r	**Reserved** Read always returns 0

CC4ySWR

Through this register it is possible for the SW to clear a specific interrupt status flag.

CC4ySWR (y = 0 - 3)
Interrupt Status Clear **($01B0_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				RTRPF	RE2A	RE1A	RE0A	0				RCMD	RCMU	ROM	RPM
r				w	w	w	w	r				w	w	w	w

Field	Bits	Type	Description
RPM	0	w	**Period match while counting up clear** Writing a 1_B into this field clears the **CC4yINTS**.PMUS bit. A read always returns 0.
ROM	1	w	**One match while counting down clear** Writing a 1_B into this bit clears the **CC4yINTS**.OMDS bit. A read always returns 0.
RCMU	2	w	**Compare match while counting up clear** Writing a 1_B into this field clears the **CC4yINTS**.CMUS bit. A read always returns 0.
RCMD	3	w	**Compare match while counting down clear** Writing a 1_B into this bit clears the **CC4yINTS**.CMDS bit. A read always returns 0.
RE0A	8	w	**Event 0 detection clear** Writing a 1_B into this bit clears the **CC4yINTS**.E0AS bit. A read always returns 0.
RE1A	9	w	**Event 1 detection clear** Writing a 1_B into this bit clears the **CC4yINTS**.E1AS bit. A read always returns 0.
RE2A	10	w	**Event 2 detection clear** Writing a 1_B into this bit clears the **CC4yINTS**.E2AS bit. A read always returns 0.

Field	Bits	Type	Description
RTRPF	11	w	**Trap Flag status clear** Writing a 1_B into this bit clears the **CC4yINTS**.TRPF bit. Not valid if **CC4yTC**.TRPEN = 1_B and the Trap State is still active. A read always returns 0.
0	[7:4], [31:12]	r	**Reserved** Read always returns 0

22.8 Interconnects

The tables that refer to the “global pins” are the ones that contain the inputs/outputs of each module that are common to all slices.

The GPIO connections are available at the Ports chapter.

22.8.1 CCU40 Pins

Table 22-13 CCU40 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
CCU40.MCLK	I	SCU.CCUCLK	Kernel clock
CCU40.CLKA	I	ERU1.IOUT0	another count source for the prescaler
CCU40.CLKB	I	ERU1.IOUT1	another count source for the prescaler
CCU40.CLKC	I	0	another count source for the prescaler
CCU40.MCSS	I	0	Multi pattern sync with shadow transfer trigger
CCU40.SR0	O	NVIC; DMA; POSIF0.MSETA; VADC.G0REQGTC; VADC.G1REQGTC; VADC.G2REQGTC; VADC.G3REQGTC; VADC.BGREQGTC;	Service request line

Table 22-13 CCU40 Pin Connections (cont'd)

Global Inputs/Outputs	I/O	Connected To	Description
CCU40.SR1	O	NVIC; DMA; DAC.TRIGGER[2]; U0C0.DX2E;	Service request line
CCU40.SR2	O	NVIC; VADC.G0REQTRA; VADC.G1REQTRA; VADC.G2REQTRA; VADC.G3REQTRA; VADC.BGREQTRA;	Service request line
CCU40.SR3	O	NVIC; CCU80.IGBTB; VADC.G0REQTRB; VADC.G1REQTRB; VADC.G2REQTRB; VADC.G3REQTRB; VADC.BGREQTRB; CCU80.IN0K; CCU81.IN0K;	Service request line

Table 22-14 CCU40 - CC40 Pin Connections

Input/Output	I/O	Connected To	Description
CCU40.IN0A	I	GPIO	General purpose function
CCU40.IN0B	I	GPIO	General purpose function
CCU40.IN0C	I	GPIO	General purpose function
CCU40.IN0D	I	ERU1.PDOUT1	General purpose function
CCU40.IN0E	I	POSIF0.OUT0	General purpose function
CCU40.IN0F	I	POSIF0.OUT1	General purpose function
CCU40.IN0G	I	POSIF0.OUT3	General purpose function
CCU40.IN0H	I	CAN.SR7	General purpose function
CCU40.IN0I	I	SCU.GLCCST40	General purpose function
CCU40.IN0J	I	ERU1.PDOUT0	General purpose function
CCU40.IN0K	I	ERU1.IOUT0	General purpose function

Table 22-14 CCU40 - CC40 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU40.IN0L	I	U0C0.DX2INS	General purpose function
CCU40.IN0M	I	CCU40.ST0	General purpose function
CCU40.IN0N	I	CCU40.ST1	General purpose function
CCU40.IN0O	I	CCU40.ST2	General purpose function
CCU40.IN0P	I	CCU40.ST3	General purpose function
CCU40.MCI0	I	0	Multi Channel pattern input
CCU40.OUT0	O	GPIO	Slice compare output
CCU40.GP00	O	NOT CONNECTED	Selected signal for event 0
CCU40.GP01	O	NOT CONNECTED	Selected signal for event 1
CCU40.GP02	O	NOT CONNECTED	Selected signal for event 2
CCU40.ST0	O	ERU1.0A2; ERU1.OGU02; POSIF0.HSDA	Slice status bit
CCU40.PS0	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 22-15 CCU40 - CC41 Pin Connections

Input/Output	I/O	Connected To	Description
CCU40.IN1A	I	GPIO	General purpose function
CCU40.IN1B	I	GPIO	General purpose function
CCU40.IN1C	I	GPIO	General purpose function
CCU40.IN1D	I	ERU1.PDOUT0	General purpose function
CCU40.IN1E	I	POSIF0.OUT0	General purpose function
CCU40.IN1F	I	POSIF0.OUT1	General purpose function
CCU40.IN1G	I	POSIF0.OUT3	General purpose function
CCU40.IN1H	I	POSIF0.OUT4	General purpose function
CCU40.IN1I	I	SCU.GLCCST40	General purpose function
CCU40.IN1J	I	ERU1.PDOUT1	General purpose function
CCU40.IN1K	I	ERU1.IOUT1	General purpose function

Table 22-15 CCU40 - CC41 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU40.IN1L	I	POSIF0.OUT2	General purpose function
CCU40.IN1M	I	CCU40.ST0	General purpose function
CCU40.IN1N	I	CCU40.ST1	General purpose function
CCU40.IN1O	I	CCU40.ST2	General purpose function
CCU40.IN1P	I	CCU40.ST3	General purpose function
CCU40.MCI1	I	0	Multi Channel pattern input
CCU40.OUT1	O	GPIO	Slice compare output
CCU40.GP10	O	NOT CONNECTED	Selected signal for event 0
CCU40.GP11	O	NOT CONNECTED	Selected signal for event 1
CCU40.GP12	O	NOT CONNECTED	Selected signal for event 2
CCU40.ST1	O	POSIF0.MSETB; ERU1.1A2	Slice status bit
CCU40.PS1	O	POSIF0.SYNCC	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 22-16 CCU40 - CC42 Pin Connections

Input/Output	I/O	Connected To	Description
CCU40.IN2A	I	GPIO	General purpose function
CCU40.IN2B	I	GPIO	General purpose function
CCU40.IN2C	I	GPIO	General purpose function
CCU40.IN2D	I	ERU1.PDOUT0	General purpose function
CCU40.IN2E	I	POSIF0.OUT0	General purpose function
CCU40.IN2F	I	POSIF0.OUT2	General purpose function
CCU40.IN2G	I	POSIF0.OUT3	General purpose function
CCU40.IN2H	I	POSIF0.OUT4	General purpose function
CCU40.IN2I	I	SCU.GLCCST40	General purpose function
CCU40.IN2J	I	ERU1.PDOUT2	General purpose function
CCU40.IN2K	I	ERU1.IOUT2	General purpose function

Table 22-16 CCU40 - CC42 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU40.IN2L	I	U0C1.DX2INS	General purpose function
CCU40.IN2M	I	CCU40.ST0	General purpose function
CCU40.IN2N	I	CCU40.ST1	General purpose function
CCU40.IN2O	I	CCU40.ST2	General purpose function
CCU40.IN2P	I	CCU40.ST3	General purpose function
CCU40.MCI2	I	0	Multi Channel pattern input
CCU40.OUT2	O	GPIO	Slice compare output
CCU40.GP20	O	NOT CONNECTED	Selected signal for event 0
CCU40.GP21	O	NOT CONNECTED	Selected signal for event 1
CCU40.GP22	O	NOT CONNECTED	Selected signal for event 2
CCU40.ST2	O	ERU1.2A2	Slice status bit
CCU40.PS2	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 22-17 CCU40 - CC43 Pin Connections

Input/Output	I/O	Connected To	Description
CCU40.IN3A	I	GPIO	General purpose function
CCU40.IN3B	I	GPIO	General purpose function
CCU40.IN3C	I	GPIO	General purpose function
CCU40.IN3D	I	ERU1.PDOUT0	General purpose function
CCU40.IN3E	I	POSIF0.OUT3	General purpose function
CCU40.IN3F	I	POSIF0.OUT5	General purpose function
CCU40.IN3G	I	VADC.G0ARBCNT	General purpose function
CCU40.IN3H	I	CCU80.IGBTO	General purpose function
CCU40.IN3I	I	SCU.GLCCST40	General purpose function
CCU40.IN3J	I	ERU1.PDOUT3	General purpose function
CCU40.IN3K	I	ERU1.IOUT3	General purpose function
CCU40.IN3L	I	U1C0.DX2INS	General purpose function

Table 22-17 CCU40 - CC43 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU40.IN3M	I	CCU40.ST0	General purpose function
CCU40.IN3N	I	CCU40.ST1	General purpose function
CCU40.IN3O	I	CCU40.ST2	General purpose function
CCU40.IN3P	I	CCU40.ST3	General purpose function
CCU40.MCI3	I	0	Multi Channel pattern input
CCU40.OUT3	O	GPIO	Slice compare output
CCU40.GP30	O	NOT CONNECTED	Selected signal for event 0
CCU40.GP31	O	NOT CONNECTED	Selected signal for event 1
CCU40.GP32	O	NOT CONNECTED	Selected signal for event 2
CCU40.ST3	O	VADC.G0REQGTA; VADC.G1REQGTA; VADC.G2REQGTA; VADC.G3REQGTA; VADC.BGREQGTA; ERU1.3A2; CCU80.IGBTA;	Slice status bit
CCU40.PS3	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

22.8.2 CCU41 Pins

Table 22-18 CCU41 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
CCU41.MCLK	I	SCU.CCUCLK	Kernel clock
CCU41.CLKA	I	ERU1.IOUT0	another count source for the prescaler
CCU41.CLKB	I	ERU1.IOUT1	another count source for the prescaler

Table 22-18 CCU41 Pin Connections (cont'd)

Global Inputs/Outputs	I/O	Connected To	Description
CCU41.CLKC	I	0	another count source for the prescaler
CCU41.MCSS	I	0	Multi pattern sync with shadow transfer trigger
CCU41.SR0	O	NVIC; DMA; POSIF1.MSETA;	Service request line
CCU41.SR1	O	NVIC; DMA; DAC.TRIGGER[3]; VADC.G0REQGTD; VADC.G1REQGTD; VADC.G2REQGTD; VADC.G3REQGTD; VADC.BGREQGTD; U1C0.DX2E; U2C0.DX2E;	Service request line
CCU41.SR2	O	NVIC; VADC.G0REQTRC; VADC.G1REQTRC; VADC.G2REQTRC; VADC.G3REQTRC; VADC.BGREQTRC;	Service request line
CCU41.SR3	O	NVIC; CCU81.IGBTB; VADC.G0REQTRD; VADC.G1REQTRD; VADC.G2REQTRD; VADC.G3REQTRD; VADC.BGREQTRD; CCU81.IN1K; CCU80.IN1K;	Service request line

Table 22-19 CCU41 - CC40 Pin Connections

Input/Output	I/O	Connected To	Description
CCU41.IN0A	I	GPIO	General purpose function
CCU41.IN0B	I	GPIO	General purpose function
CCU41.IN0C	I	GPIO	General purpose function
CCU41.IN0D	I	ERU1.PDOUT1	General purpose function
CCU41.IN0E	I	POSIF1.OUT0	General purpose function
CCU41.IN0F	I	POSIF1.OUT1	General purpose function
CCU41.IN0G	I	POSIF1.OUT3	General purpose function
CCU41.IN0H	I	CAN.SR7	General purpose function
CCU41.IN0I	I	SCU.GLCCST41	General purpose function
CCU41.IN0J	I	ERU1.PDOUT0	General purpose function
CCU41.IN0K	I	ERU1.IOUT0	General purpose function
CCU41.IN0L	I	VADC.G0BFL0	General purpose function
CCU41.IN0M	I	CCU41.ST0	General purpose function
CCU41.IN0N	I	CCU41.ST1	General purpose function
CCU41.IN0O	I	CCU41.ST2	General purpose function
CCU41.IN0P	I	CCU41.ST3	General purpose function
CCU41.MCI0	I	0	Multi Channel pattern input
CCU41.OUT0	O	GPIO	Slice compare output
CCU41.GP00	O	NOT CONNECTED	Selected signal for event 0
CCU41.GP01	O	NOT CONNECTED	Selected signal for event 1
CCU41.GP02	O	NOT CONNECTED	Selected signal for event 2
CCU41.ST0	O	POSIF1.HSDA; ERU1.OGU12	Slice status bit
CCU41.PS0	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 22-20 CCU41 - CC41 Pin Connections

Input/Output	I/O	Connected To	Description
CCU41.IN1A	I	GPIO	General purpose function
CCU41.IN1B	I	GPIO	General purpose function
CCU41.IN1C	I	GPIO	General purpose function
CCU41.IN1D	I	ERU1.PDOUT0	General purpose function
CCU41.IN1E	I	POSIF1.OUT0	General purpose function
CCU41.IN1F	I	POSIF1.OUT1	General purpose function
CCU41.IN1G	I	POSIF1.OUT3	General purpose function
CCU41.IN1H	I	POSIF1.OUT4	General purpose function
CCU41.IN1I	I	SCU.GLCCST41	General purpose function
CCU41.IN1J	I	ERU1.PDOUT1	General purpose function
CCU41.IN1K	I	ERU1.IOUT1	General purpose function
CCU41.IN1L	I	POSIF1.OUT2	General purpose function
CCU41.IN1M	I	CCU41.ST0	General purpose function
CCU41.IN1N	I	CCU41.ST1	General purpose function
CCU41.IN1O	I	CCU41.ST2	General purpose function
CCU41.IN1P	I	CCU41.ST3	General purpose function
CCU41.MCI1	I	0	Multi Channel pattern input
CCU41.OUT1	O	GPIO	Slice compare output
CCU41.GP10	O	NOT CONNECTED	Selected signal for event 0
CCU41.GP11	O	NOT CONNECTED	Selected signal for event 1
CCU41.GP12	O	NOT CONNECTED	Selected signal for event 2
CCU41.ST1	O	POSIF1.MSETB;	Slice status bit
CCU41.PS1	O	POSIF1.MSYNCC	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 22-21 CCU41 - CC42 Pin Connections

Input/Output	I/O	Connected To	Description
CCU41.IN2A	I	GPIO	General purpose function
CCU41.IN2B	I	GPIO	General purpose function
CCU41.IN2C	I	GPIO	General purpose function
CCU41.IN2D	I	ERU1.PDOUT0	General purpose function
CCU41.IN2E	I	POSIF1.OUT0	General purpose function
CCU41.IN2F	I	POSIF1.OUT2	General purpose function
CCU41.IN2G	I	POSIF1.OUT3	General purpose function
CCU41.IN2H	I	POSIF1.OUT4	General purpose function
CCU41.IN2I	I	SCU.GLCCST41	General purpose function
CCU41.IN2J	I	ERU1.PDOUT2	General purpose function
CCU41.IN2K	I	ERU1.IOUT2	General purpose function
CCU41.IN2L	I	VADC.G0BFL1	General purpose function
CCU41.IN2M	I	CCU41.ST0	General purpose function
CCU41.IN2N	I	CCU41.ST1	General purpose function
CCU41.IN2O	I	CCU41.ST2	General purpose function
CCU41.IN2P	I	CCU41.ST3	General purpose function
CCU41.MCI2	I	0	Multi Channel pattern input
CCU41.OUT2	O	GPIO	Slice compare output
CCU41.GP20	O	NOT CONNECTED	Selected signal for event 0
CCU41.GP21	O	NOT CONNECTED	Selected signal for event 1
CCU41.GP22	O	NOT CONNECTED	Selected signal for event 2
CCU41.ST2	O	NOT CONNECTED	Slice status bit
CCU41.PS2	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 22-22 CCU41 - CC43 Pin Connections

Input/Output	I/O	Connected To	Description
CCU41.IN3A	I	GPIO	General purpose function
CCU41.IN3B	I	GPIO	General purpose function
CCU41.IN3C	I	GPIO	General purpose function
CCU41.IN3D	I	ERU1.PDOUT0	General purpose function
CCU41.IN3E	I	POSIF1.OUT3	General purpose function
CCU41.IN3F	I	POSIF1.OUT5	General purpose function
CCU41.IN3G	I	VADC.G1ARBCNT	General purpose function
CCU41.IN3H	I	CCU81.IGBTO	General purpose function
CCU41.IN3I	I	SCU.GLCCST41	General purpose function
CCU41.IN3J	I	ERU1.PDOUT3	General purpose function
CCU41.IN3K	I	ERU1.IOUT3	General purpose function
CCU41.IN3L	I	VADC.G0BFL2	General purpose function
CCU41.IN3M	I	CCU41.ST0	General purpose function
CCU41.IN3N	I	CCU41.ST1	General purpose function
CCU41.IN3O	I	CCU41.ST2	General purpose function
CCU41.IN3P	I	CCU41.ST3	General purpose function
CCU41.MCI3	I	0	Multi Channel pattern input
CCU41.OUT3	O	GPIO	Slice compare output
CCU41.GP30	O	NOT CONNECTED	Selected signal for event 0
CCU41.GP31	O	NOT CONNECTED	Selected signal for event 1
CCU41.GP32	O	NOT CONNECTED	Selected signal for event 2
CCU41.ST3	O	CCU81.IGBTA; VADC.G0REQGTB; VADC.G1REQGTB; VADC.G2REQGTB; VADC.G3REQGTB; VADC.BGREQGTB;	Slice status bit
CCU41.PS3	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

22.8.3 CCU42 pins

Table 22-23 CCU42 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
CCU42.MCLK	I	SCU.CCUCLK	Kernel clock
CCU42.CLKA	I	ERU1.IOUT0	another count source for the prescaler
CCU42.CLKB	I	ERU1.IOUT1	another count source for the prescaler
CCU42.CLKC	I	0	another count source for the prescaler
CCU42.MCSS	I	POSIF0.OUT6	Multi pattern sync with shadow transfer trigger
CCU42.SR0	O	NVIC; DMA; POSIF0.MSETC; CCU80.IGBTD;	Service request line
CCU42.SR1	O	NVIC; DMA; U0C1.DX2E;	Service request line
CCU42.SR2	O	NVIC;	Service request line
CCU42.SR3	O	NVIC; VADC.G0REQTRE; VADC.G1REQTRE; VADC.G2REQTRE; VADC.G3REQTRE; VADC.BGREQTRE; CCU80.IN2K; CCU81.IN2K;	Service request line

Table 22-24 CCU42 - CC40 Pin Connections

Input/Output	I/O	Connected To	Description
CCU42.IN0A	I	GPIO	General purpose function
CCU42.IN0B	I	GPIO	General purpose function
CCU42.IN0C	I	GPIO	General purpose function

Table 22-24 CCU42 - CC40 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU42.IN0D	I	ERU1.PDOUT1	General purpose function
CCU42.IN0E	I	POSIF0.OUT2	General purpose function
CCU42.IN0F	I	POSIF0.OUT5	General purpose function
CCU42.IN0G	I	CCU80.SR3	General purpose function
CCU42.IN0H	I	CCU80.IGBTO	General purpose function
CCU42.IN0I	I	SCU.GLCCST42	General purpose function
CCU42.IN0J	I	ERU1.PDOUT0	General purpose function
CCU42.IN0K	I	ERU1.IOUT0	General purpose function
CCU42.IN0L	I	U0C0.DX2INS	General purpose function
CCU42.IN0M	I	CCU42.ST0	General purpose function
CCU42.IN0N	I	CCU42.ST1	General purpose function
CCU42.IN0O	I	CCU42.ST2	General purpose function
CCU42.IN0P	I	CCU42.ST3	General purpose function
CCU42.MCI0	I	POSIF0.MOUT[0]	Multi Channel pattern input
CCU42.OUT0	O	GPIO	Slice compare output
CCU42.GP00	O	NOT CONNECTED	Selected signal for event 0
CCU42.GP01	O	NOT CONNECTED	Selected signal for event 1
CCU42.GP02	O	NOT CONNECTED	Selected signal for event 2
CCU42.ST0	O	POSIF0.MSETD; CCU80.IGBTC;	Slice status bit
CCU42.PS0	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 22-25 CCU42 - CC41 Pin Connections

Input/Output	I/O	Connected To	Description
CCU42.IN1A	I	GPIO	General purpose function
CCU42.IN1B	I	GPIO	General purpose function
CCU42.IN1C	I	GPIO	General purpose function

Table 22-25 CCU42 - CC41 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU42.IN1D	I	ERU1.PDOUT0	General purpose function
CCU42.IN1E	I	POSIF0.OUT2	General purpose function
CCU42.IN1F	I	POSIF0.OUT5	General purpose function
CCU42.IN1G	I	CCU81.SR3	General purpose function
CCU42.IN1H	I	0	General purpose function
CCU42.IN1I	I	SCU.GLCCST42	General purpose function
CCU42.IN1J	I	ERU1.PDOUT1	General purpose function
CCU42.IN1K	I	ERU1.IOUT1	General purpose function
CCU42.IN1L	I	U0C1.DX2INS	General purpose function
CCU42.IN1M	I	CCU42.ST0	General purpose function
CCU42.IN1N	I	CCU42.ST1	General purpose function
CCU42.IN1O	I	CCU42.ST2	General purpose function
CCU42.IN1P	I	CCU42.ST3	General purpose function
CCU42.MCI1	I	POSIF0.MOUT[1]	Multi Channel pattern input
CCU42.OUT1	O	GPIO	Slice compare output
CCU42.GP10	O	NOT CONNECTED	Selected signal for event 0
CCU42.GP11	O	NOT CONNECTED	Selected signal for event 1
CCU42.GP12	O	NOT CONNECTED	Selected signal for event 2
CCU42.ST1	O	NOT CONNECTED	Slice status bit
CCU42.PS1	O	POSIF0.SYNCD	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 22-26 CCU42 - CC42 Pin Connections

Input/Output	I/O	Connected To	Description
CCU42.IN2A	I	GPIO	General purpose function
CCU42.IN2B	I	GPIO	General purpose function
CCU42.IN2C	I	GPIO	General purpose function
CCU42.IN2D	I	ERU1.PDOUT0	General purpose function

Table 22-26 CCU42 - CC42 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU42.IN2E	I	POSIF0.OUT2	General purpose function
CCU42.IN2F	I	POSIF0.OUT5	General purpose function
CCU42.IN2G	I	CAN.SR7	General purpose function
CCU42.IN2H	I	0	General purpose function
CCU42.IN2I	I	SCU.GLCCST42	General purpose function
CCU42.IN2J	I	ERU1.PDOUT2	General purpose function
CCU42.IN2K	I	ERU1.IOUT2	General purpose function
CCU42.IN2L	I	U1C0.DX2INS	General purpose function
CCU42.IN2M	I	CCU42.ST0	General purpose function
CCU42.IN2N	I	CCU42.ST1	General purpose function
CCU42.IN2O	I	CCU42.ST2	General purpose function
CCU42.IN2P	I	CCU42.ST3	General purpose function
CCU42.MCI2	I	POSIF0.MOUT[2]	Multi Channel pattern input
CCU42.OUT2	O	GPIO	Slice compare output
CCU42.GP20	O	NOT CONNECTED	Selected signal for event 0
CCU42.GP21	O	NOT CONNECTED	Selected signal for event 1
CCU42.GP22	O	NOT CONNECTED	Selected signal for event 2
CCU42.ST2	O	NOT CONNECTED	Slice status bit
CCU42.PS2	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 22-27 CCU42 - CC43 Pin Connections

Input/Output	I/O	Connected To	Description
CCU42.IN3A	I	GPIO	General purpose function
CCU42.IN3B	I	GPIO	General purpose function
CCU42.IN3C	I	GPIO	General purpose function
CCU42.IN3D	I	ERU1.PDOUT0	General purpose function
CCU42.IN3E	I	POSIF0.OUT2	General purpose function

Table 22-27 CCU42 - CC43 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU42.IN3F	I	POSIF0.OUT5	General purpose function
CCU42.IN3G	I	VADC.G2ARBCNT	General purpose function
CCU42.IN3H	I	0	General purpose function
CCU42.IN3I	I	SCU.GLCCST42	General purpose function
CCU42.IN3J	I	ERU1.PDOUT3	General purpose function
CCU42.IN3K	I	ERU1.IOUT3	General purpose function
CCU42.IN3L	I	U1C1.DX2INS	General purpose function
CCU42.IN3M	I	CCU42.ST0	General purpose function
CCU42.IN3N	I	CCU42.ST1	General purpose function
CCU42.IN3O	I	CCU42.ST2	General purpose function
CCU42.IN3P	I	CCU42.ST3	General purpose function
CCU42.MCI3	I	POSIF0.MOUT[3]	Multi Channel pattern input
CCU42.OUT3	O	GPIO	Slice compare output
CCU42.GP30	O	NOT CONNECTED	Selected signal for event 0
CCU42.GP31	O	NOT CONNECTED	Selected signal for event 1
CCU42.GP32	O	NOT CONNECTED	Selected signal for event 2
CCU42.ST3	O	NOT CONNECTED	Slice status bit
CCU42.PS3	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

22.8.4 CCU43 pins

Table 22-28 CCU43 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
CCU43.MCLK	I	SCU.CCUCLK	Kernel clock
CCU43.CLKA	I	ERU1.IOUT0	another count source for the prescaler

Table 22-28 CCU43 Pin Connections (cont'd)

Global Inputs/Outputs	I/O	Connected To	Description
CCU43.CLKB	I	ERU1.IOUT1	another count source for the prescaler
CCU43.CLKC	I	0	another count source for the prescaler
CCU43.MCSS	I	POSIF1.OUT6	Multi pattern sync with shadow transfer trigger
CCU43.SR0	O	NVIC; DMA; POSIF1.MSETC; CCU81.IGBTD;	Service request line
CCU43.SR1	O	NVIC; DMA; U1C1.DX2E; U2C0.DX2E;	Service request line
CCU43.SR2	O	NVIC;	Service request line
CCU43.SR3	O	NVIC; VADC.G0REQTRF; VADC.G1REQTRF; VADC.G2REQTRF; VADC.G3REQTRF; VADC.BGREQTRF; CCU80.IN3K; CCU81.IN3K	Service request line

Table 22-29 CCU43 - CC40 Pin Connections

Input/Output	I/O	Connected To	Description
CCU43.IN0A	I	GPIO	General purpose function
CCU43.IN0B	I	GPIO	General purpose function
CCU43.IN0C	I	GPIO	General purpose function
CCU43.IN0D	I	ERU1.PDOUT1	General purpose function
CCU43.IN0E	I	POSIF1.OUT2	General purpose function
CCU43.IN0F	I	POSIF1.OUT5	General purpose function
CCU43.IN0G	I	CCU81.IGBTO	General purpose function

Table 22-29 CCU43 - CC40 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU43.IN0H	I	VADC.G0BFL0	General purpose function
CCU43.IN0I	I	SCU.GLCCST43	General purpose function
CCU43.IN0J	I	ERU1.PDOUT0	General purpose function
CCU43.IN0K	I	ERU1.IOUT0	General purpose function
CCU43.IN0L	I	U0C0.DX2INS	General purpose function
CCU43.IN0M	I	CCU43.ST0	General purpose function
CCU43.IN0N	I	CCU43.ST1	General purpose function
CCU43.IN0O	I	CCU43.ST2	General purpose function
CCU43.IN0P	I	CCU43.ST3	General purpose function
CCU43.MCI0	I	POSIF1.MOUT[0]	Multi Channel pattern input
CCU43.OUT0	O	GPIO	Slice compare output
CCU43.GP00	O	NOT CONNECTED	Selected signal for event 0
CCU43.GP01	O	NOT CONNECTED	Selected signal for event 1
CCU43.GP02	O	NOT CONNECTED	Selected signal for event 2
CCU43.ST0	O	POSIF1.MSETD; CCU81.IGBTC;	Slice status bit
CCU43.PS0	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 22-30 CCU43 - CC41 Pin Connections

Input/Output	I/O	Connected To	Description
CCU43.IN1A	I	GPIO	General purpose function
CCU43.IN1B	I	GPIO	General purpose function
CCU43.IN1C	I	GPIO	General purpose function
CCU43.IN1D	I	ERU1.PDOUT0	General purpose function
CCU43.IN1E	I	POSIF1.OUT2	General purpose function
CCU43.IN1F	I	POSIF1.OUT5	General purpose function
CCU43.IN1G	I	CAN.SR7	General purpose function

Table 22-30 CCU43 - CC41 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU43.IN1H	I	VADC.G1BFL0	General purpose function
CCU43.IN1I	I	SCU.GLCCST43	General purpose function
CCU43.IN1J	I	ERU1.PDOUT1	General purpose function
CCU43.IN1K	I	ERU1.IOUT1	General purpose function
CCU43.IN1L	I	U0C1.DX2INS	General purpose function
CCU43.IN1M	I	CCU43.ST0	General purpose function
CCU43.IN1N	I	CCU43.ST1	General purpose function
CCU43.IN1O	I	CCU43.ST2	General purpose function
CCU43.IN1P	I	CCU43.ST3	General purpose function
CCU43.MCI1	I	POSIF1.MOUT[1]	Multi Channel pattern input
CCU43.OUT1	O	GPIO	Slice compare output
CCU43.GP10	O	NOT CONNECTED	Selected signal for event 0
CCU43.GP11	O	NOT CONNECTED	Selected signal for event 1
CCU43.GP12	O	NOT CONNECTED	Selected signal for event 2
CCU43.ST1	O	NOT CONNECTED	Slice status bit
CCU43.PS1	O	POSIF1.MSYNCD	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 22-31 CCU43 - CC42 Pin Connections

Input/Output	I/O	Connected To	Description
CCU43.IN2A	I	GPIO	General purpose function
CCU43.IN2B	I	GPIO	General purpose function
CCU43.IN2C	I	GPIO	General purpose function
CCU43.IN2D	I	ERU1.PDOUT0	General purpose function
CCU43.IN2E	I	POSIF1.OUT2	General purpose function
CCU43.IN2F	I	POSIF1.OUT5	General purpose function
CCU43.IN2G	I	0	General purpose function
CCU43.IN2H	I	VADC.G2BFL0	General purpose function

Table 22-31 CCU43 - CC42 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU43.IN2I	I	SCU.GLCCST43	General purpose function
CCU43.IN2J	I	ERU1.PDOUT2	General purpose function
CCU43.IN2K	I	ERU1.IOUT2	General purpose function
CCU43.IN2L	I	U1C0.DX2INS	General purpose function
CCU43.IN2M	I	CCU43.ST0	General purpose function
CCU43.IN2N	I	CCU43.ST1	General purpose function
CCU43.IN2O	I	CCU43.ST2	General purpose function
CCU43.IN2P	I	CCU43.ST3	General purpose function
CCU43.MCI2	I	POSIF1.MOUT[2]	Multi Channel pattern input
CCU43.OUT2	O	GPIO	Slice compare output
CCU43.GP20	O	NOT CONNECTED	Selected signal for event 0
CCU43.GP21	O	NOT CONNECTED	Selected signal for event 1
CCU43.GP22	O	NOT CONNECTED	Selected signal for event 2
CCU43.ST2	O	NOT CONNECTED	Slice status bit
CCU43.PS2	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 22-32 CCU43 - CC43 Pin Connections

Input/Output	I/O	Connected To	Description
CCU43.IN3A	I	GPIO	General purpose function
CCU43.IN3B	I	GPIO	General purpose function
CCU43.IN3C	I	GPIO	General purpose function
CCU43.IN3D	I	ERU1.PDOUT0	General purpose function
CCU43.IN3E	I	POSIF1.OUT2	General purpose function
CCU43.IN3F	I	POSIF1.OUT5	General purpose function
CCU43.IN3G	I	VADC.G3ARBCNT	General purpose function
CCU43.IN3H	I	VADC.G3BFL0	General purpose function
CCU43.IN3I	I	SCU.GLCCST43	General purpose function

Table 22-32 CCU43 - CC43 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU43.IN3J	I	ERU1.PDOUT3	General purpose function
CCU43.IN3K	I	ERU1.IOUT3	General purpose function
CCU43.IN3L	I	U1C1.DX2INS	General purpose function
CCU43.IN3M	I	CCU43.ST0	General purpose function
CCU43.IN3N	I	CCU43.ST1	General purpose function
CCU43.IN3O	I	CCU43.ST2	General purpose function
CCU43.IN3P	I	CCU43.ST3	General purpose function
CCU43.MCI3	I	POSIF1.MOUT[3]	Multi Channel pattern input
CCU43.OUT3	O	GPIO	Slice compare output
CCU43.GP30	O	NOT CONNECTED	Selected signal for event 0
CCU43.GP31	O	NOT CONNECTED	Selected signal for event 1
CCU43.GP32	O	NOT CONNECTED	Selected signal for event 2
CCU43.ST3	O	NOT CONNECTED	Slice status bit
CCU43.PS3	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

23 Capture/Compare Unit 8 (CCU8)

The CCU8 peripheral functions play a major role in applications that need complex Pulse Width Modulation (PWM) signal generation, with complementary high side and low side switches, multi phase control or output parity checking. These functions in conjunction with a very flexible and programmable signal conditioning scheme, make the CCU8 the must have peripheral for state of the art motor control, multi phase and multi level power electronics systems.

The internal modularity of CCU8, translates into a software friendly system for fast code development and portability between applications.

Table 23-1 Abreviations table

PWM	Pulse Width Modulation
CCU8x	Capture/Compare Unit 8 module instance x
CC8y	Capture/Compare Unit 8 Timer Slice instance y
ADC	Analog to Digital Converter
POSIF	Position Interface peripheral
SCU	System Control Unit
f_{ccu8}	CCU8 module clock frequency
f_{tclk}	CC4y timer clock frequency

Note: A small “y” or “x” letter in a register indicates an index

23.1 Overview

The CCU8 unit is comprised of four identical 16 bit Capture/Compare Timer slices, CC8y. Each Timer Slice can work in Compare or in Capture Mode. In Compare Mode, one has two dedicated compare channels that enable the generation of up to 4 PWM signals per Timer Slice (up to 16 PWM outputs per CCU8 unit), with dead time insertion to prevent short circuits in the switches. In Capture Mode a set of up to four capture registers is available.

Each CCU8 module has four service request lines that can be easily programmed to act as synchronized triggers between the PWM signal generation and an ADC conversion.

Straightforward timer slice concatenation is also possible, enabling up to 64 bit timing operations. This offers a flexible frequency measurement, frequency multiplication and pulse width modulation scheme.

A programmable function input selector for each timer slice, that offers up to nine functions, discards the need of complete resource mapping due to input ports availability.

A built-in link between the CCU8 and POSIF modules also enable a flexible digital motor control loop implementation, with direct coupling with Hall Sensors for Brushless DC Motor Control.

23.1.1 Features

CCU8 Module Features

Each CCU8 represents a combination of four Timer Slices, that can work independently in compare or capture mode. Each timer slice has 4 dedicated outputs for PWM signal generation.

All four CCU8 timer slices, CC4y, are identical in terms of available functions and operating modes. Avoiding this way the need of implementing different software routines, depending on which resource of CCU8 is used.

A built-in link between the four timer slices is also available, enabling this way a simplified timer concatenation and sequential operations.

General Features

- 16 bit timer cells
- programmable low pass filter for the inputs
- built-in timer concatenation
  - 32, 48 or 64 bit width
- shadow transfer for the period and compare channels
- four capture registers in capture mode
- programmable clock prescaler
- normal timer mode
- gated timer mode
- three counting schemes
  - center aligned
  - edge aligned
  - single shot
- PWM generation
- asymmetric PWM generation
- TRAP function
- dead time generation
- start/stop can be controlled by external events
- counting external events
- four dedicated service request lines per CCU8

Additional features

- external modulation function
- load controlled by external events

- dithering PWM
- floating point pre scaler
- output state override by an external event
- programmable output parity checker
- easy connection with POSIF unit for
  - hall sensor mode
  - rotary encoder mode
  - multi channel/multi phase control

CCU8 features vs. applications

On **Table 23-2** a summary of the major features of the CCU8 unit mapped with the most common applications.

Table 23-2 Applications summary

Feature	Applications
Four independent timer cells	Independent PWM generation: • Multiple buck/boost converter control (with independent frequencies) • Different mode of operation for each timer, increasing the resources optimization • Up to 2 H-Bridge control • multiple Zero Voltage Switch (ZVS) converter control with easy link to the ADC channels. • Multi Level Inverters
Two compare channels per Timer Slice	Linking between the two compare channels or linking between two Timer Slices: • Asymmetric PWM signal generation possibility decreases the number of current sensors • Linking between timer slices enable Phase Shift Full Bridge topologies control • Linking between slices enable N-Phase DC/DC converter control

Table 23-2 Applications summary (cont'd)

Feature	Applications
Two Dead Time Generators	Independent dead time values for rising and falling transitions and independent channel dead time counter: • Each channel can work stand alone with different dead time values. This enables the control of up to 2 Half-Bridges with different dead time values and the same frequency • Different dead time values for rising and falling transitions can be used to optimize the switching activity of the MOSFETs
Concatenated timer cells	Easy to configure timer extension up to 64 bit: • High dynamic trigger capturing • High dynamic signal measurement
Dithering PWM	Generating a fractional PWM frequency or duty cycle: • To avoid big steps on frequency or duty cycle adjustment in slow control loop applications • Increase the PWM signal resolution over time
Floating prescaler	Automated control signal measurement: • decrease SW activity for monitoring signals with high or unknown dynamics • generating a more than 16 bit timer for system control
Up to 9 functions via external signals for each timer	Flexible resource optimization: • The complete set of external functions is always available • Several arrangements can be done inside a CCU8, e.g., one timer working in capture mode and one working in compare
Output Parity Checker	Automated Mosfet signal monitoring: • parity checker can be used to monitor the output of the IGBTs and comparing them against the complete set of PWM outputs of CCU8. • Avoiding short circuits in a multi Mosfet system.

Table 23-2 Applications summary (cont'd)

Feature	Applications
4 dedicated service request lines	Specially developed for: • generating interrupts for the microprocessor • flexible connectivity between peripherals, e.g., ADC triggering.
Linking with POSIF	Flexible profiles for: • Rotary Encoder connection • Hall Sensor • Modulating the 4 timer outputs via SW

23.1.2 Block Diagram

Each CCU8 timer slice can operate independently from the other slices for all the available modes. Each timer slice contains a dedicated input selector for functions linked with external events and has 4 dedicated compare output signals, for PWM signal generation.

The built-in timer concatenation is only possible with adjacent slices, e.g. CC80/CC81. Combinations for slice concatenations like, CC80/CC82 or CC40/CC83 are not possible.

The individual service requests for each timer slice (four per slice) are multiplexed into four module service requests lines, **Figure 23-1**.

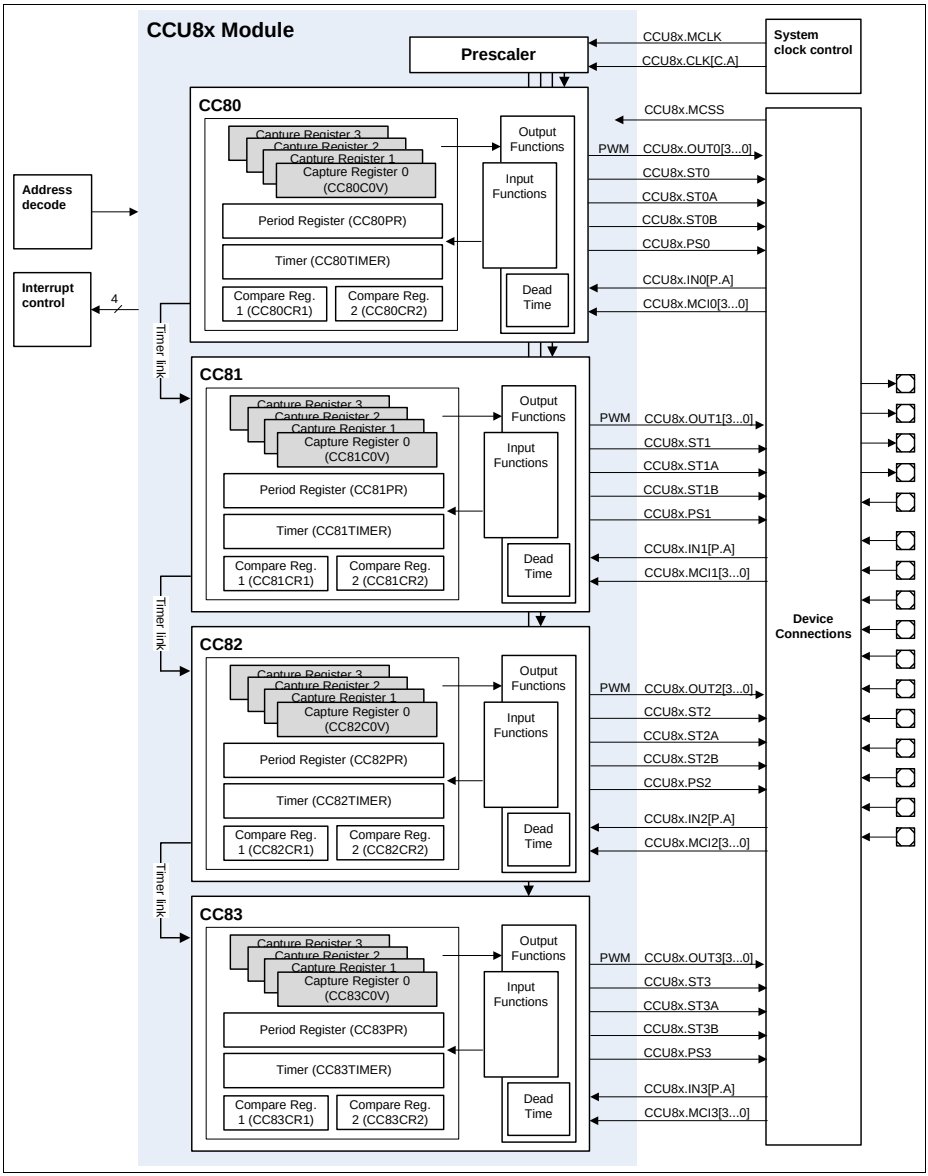


Figure 23-1 CCU8 block diagram

23.2 Functional Description

23.2.1 Overview

The input path of a CCU8 slice is comprised of a selector (**Section 23.2.2**) and a connection matrix unit (**Section 23.2.3**). The output path contains a service request control unit, a timer concatenation unit and two units that control directly the state of the output signal for each specific slice (for TRAP, dead time generation and modulation handling), see **Figure 23-2**.

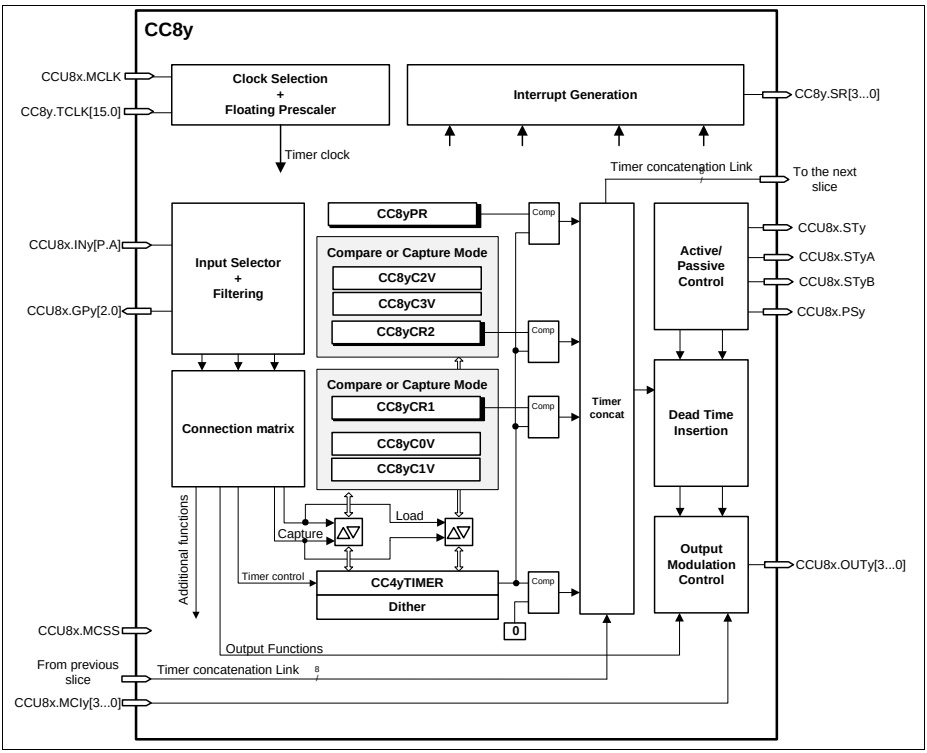


Figure 23-2 CCU8 slice block diagram

The timer core is built of 16 bit counter and one period register and two compare channels in compare mode, or four capture channels plus the period register in capture mode.

Individual timer clocks can be selected for each and every Timer Slice, enabling a very flexible resource organization inside each CCU8 module.

In compare mode the period sets the maximum counting value while the two compare registers are used to control the ACTIVE/PASSIVE state of the four dedicated comparison slice outputs.

Each CCU8 slice contains a dedicated timer link interface that is used to perform timer concatenation, up to 64 bits. This timer concatenation is controlled via a single bit field configuration.

Table 23-3 describes the inputs and outputs for each CCU8 Timer Slice.

Inputs and outputs that are not seen at the CCU8 module boundaries have a nomenclature of CC8y.<name>, whilst CCU8 module inputs and outputs are described as CCU8x.<signal_name>y (indicating the variable y the object slice).

Table 23-3 CCU8 slice pin description
Table 23-4

Pin	I/O	Description
CCU8x.MCLK	I	Module clock
CC8y.TCLK		Clock from the pre scaler
CCU8x.INy[P:A]	I	Slice functional inputs (used to control the functionality throughout slice external events)
CCU8x.MCIy[3...0]	I	Multi Channel mode inputs
CCU8x.MCSS	I	Multi Channel shadow transfer trigger
CC8y.SR[3...0]	O	Slice service request lines
CCU8x.GPy[2...0]	O	Signals decoded from the input selector (used for the parity checker function)
CCU8x.STy	O	This signal can be the slice comparison status value of channel 1, channel 2 or a AND between both
CCU8x.STyA	O	Slice comparison status value of channel 1
CCU8x.STyB	O	Slice comparison status value of channel 2
CCU8x.PSy	O	Period match
CCU8x.OUTy[3...0]	O	Slice dedicated output pins

Note:

6. *The status bit outputs of the Kernel, CCU8x.STy, CCU8x.STyA and CCU8x.STyB are extended for one more kernel clock cycle.*

7. *The Service Request signals at the output of the kernel are extended for one more kernel clock cycle.*
8. *The maximum output signal frequency of the CCU8x.STy, CCU8x.STyA and CCU8x.STyA is module clock divided by 4.*

The slice timer, can count up or down depending on the selected operating mode. A direction flag contains the actual counting direction.

The timer is connected to three stand alone comparators, one for the period match and two for the compare match of each compare channel. The registers used for comparison match of both compare channels, can be programmed to serve as capture registers, enabling sequential capture capabilities on external events.

In normal edge aligned counting scheme, the counter is cleared to 0000_H each time it matches the period value defined in the period register. In center aligned mode, the counter direction changes from ‘up counting’ to ‘down counting’ after reaching the period value. Both period and compare registers have an aggregated shadow register, which enables the update of the PWM period and duty cycle on the fly.

A single shot mode is also available, where the counter stops after it reaches the value set in the period register.

The start and stop of the counter can be set/clear by software access or by a programmable input pin.

The dead time generator can be programmed with different values for the rising and falling edge of the output.

Functions like, load, counting direction (up/down), TRAP, output modulation can also be controlled with external events, see **Section 23.2.3**.

23.2.2 Input Selector

The first unit of the slice input path, is used to select from which are used to control the available external functions.

Inside this block the user also has the possibility to perform a low pass filtering of the signals and selecting the active edge(s) or level of the external event, see **Figure 23-3**.

The user has the possibility of selecting any of the CCU8x.INy[P:A] inputs has the source of an event.

At the output of this unit we have a user selection of three events, that were configured to be active at rising, falling or both edges, or level active. These selected events can then be mapped to several functions.

Notice that each decoded event contains two outputs, one edge active and one level active, due to the fact that some functions like counting, capture or load are edge sensitive events while, timer gating or up down counting selection are level active.

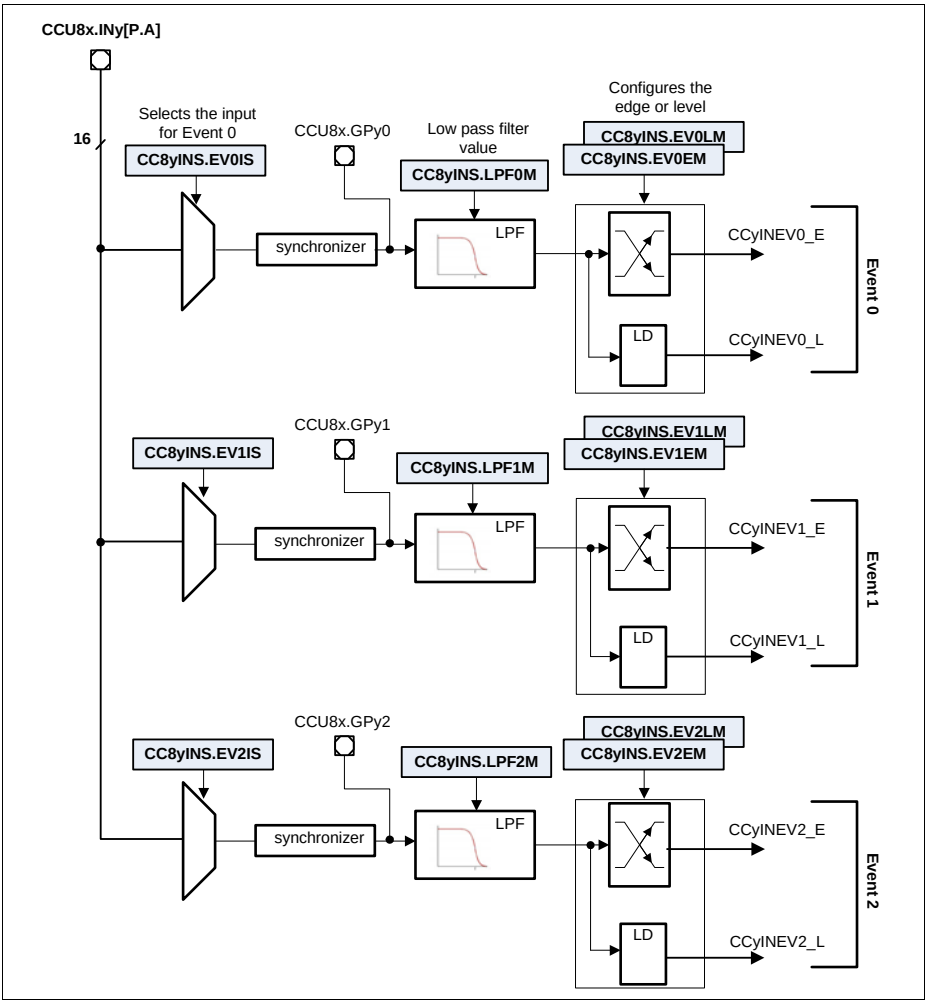


Figure 23-3 Slice input selector diagram

23.2.3 Connection Matrix

The connection matrix maps the events coming from the input selector to several user configured functions, **Figure 23-4**. The following functions can be enabled on the connection matrix:

Table 23-5 Connection matrix available functions

Function	Brief description	Map to figure Figure 23-4
Start	Edge signal to start the timer	CCystrt
Stop	Edge signal to stop the timer	CCystp
Count	Edge signal used for counting events	CCycnt
Up/down	Level signal used to select up or down counting direction	CCyupd
Capture 0	Edge signal that triggers a capture into the capture registers 0 and 1	CCycapt0
Capture 1	Edge signal that triggers a capture into the capture registers 2 and 3	CCycapt1
Gate	Level signal used to gate the timer clock	CCygate
Load	Edge signal that loads the timer with the value present at the compare register	CCyload
TRAP	Level signal used for fail-safe operation	CCytrap
Modulation	Level signal used to modulate/clear the output	CCymod
Status bit override	Status bit is going to be overridden with an input value	CCyoval for the value CCyoset for the trigger

Inside the connection matrix we also have a unit that performs the built-in timer concatenation. This concatenation enables a completely synchronized operation between the concatenated slices for timing operations and also for capture and load actions. The timer slice concatenation is done via the **CC8yCMC**.TCE bitfield. For a complete description of the concatenation function, please address **Section 23.2.10**.

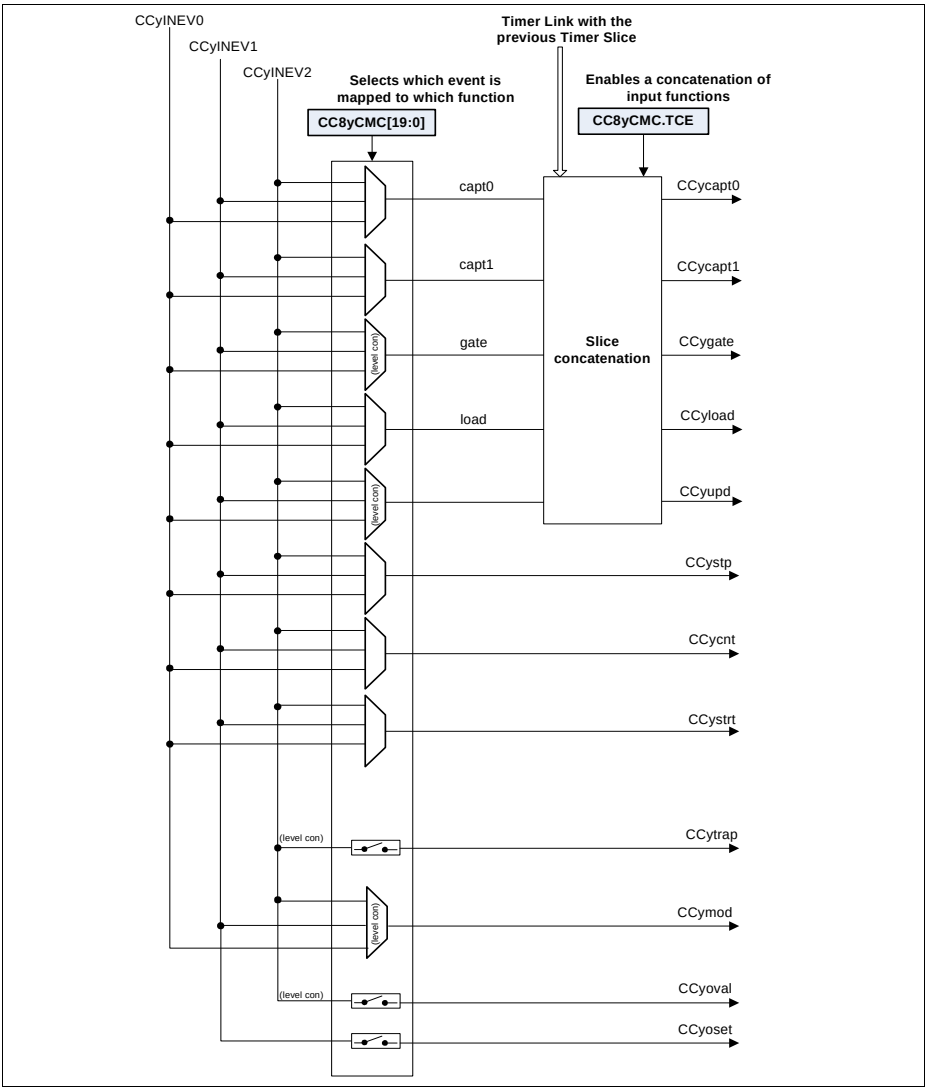


Figure 23-4 Slice connection matrix diagram

23.2.4 Start/Stop Control

Each slice contains a run bit register, that indicates the actual status of the timer, **CC8yTCST**.TRB. The start and stop of the timer can be done by software access or can be controlled directly by external events, see **Figure 23-5**.

Selecting an external signal that acts as a start trigger does not force the user to use an external stop trigger and vice versa.

Selecting the single shot mode, imposes that after the counter reaches the period value the run bit, **CC8yTCST**.TRB, is going to be cleared and therefore the timer is stopped.

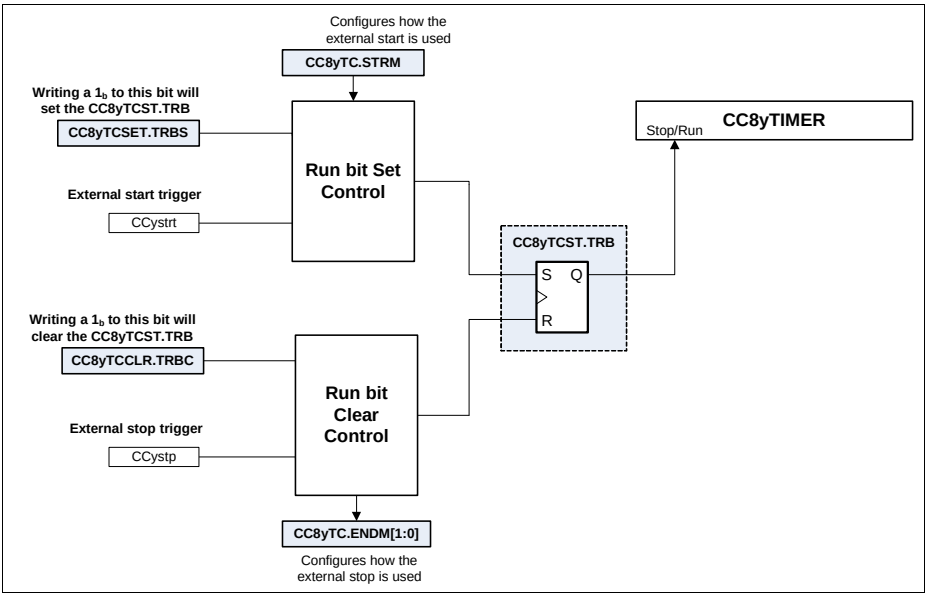


Figure 23-5 Timer start/stop control diagram

One can use the external stop signal to perform the following functions (configuration via **CC8yTC**.ENDM):

- Clear the run bit (stops the timer) - default
- Clear the timer (to 0000_H) but it does not clear the run bit (timer still running)
- Clear the timer and the run bit

One can use the external start to perform the following functions (configuration via **CC8yTC**.STRM):

- Start the timer (resume operation)
- Clear and starts the timer

The set (start the timer) of the timer run bit, always has priority over a clear (stop the timer).

23.2.5 Counting Modes

Each CC8y timer can be programmed into three different counting schemes:

- Edge aligned (default)
- Center aligned
- Single shot (edge or center aligned)

These three counting schemes can be used as stand alone without the need of selecting any inputs as external event sources. Nevertheless it is also possible to control the counting operation via external events like, timer gating, counting trigger, external stop, external start, etc.

For all the counting modes, it is possible to update on the fly the values for the timer period and compare channel. This enables a cycle by cycle update of the PWM frequency and duty cycle.

Each compare channel of the CC4y Timer Slice has an associated Status Bit (**GCST**.CC8yST1 for compare channel 1 and **GCST**.CC8yST2 for compare channel 2), that indicates the active or passive state of the channel, **Figure 23-6**. The set and clear of the status bits and the respective PWM signal generation is dictated by the timer period, compare value and the current counting mode. See the different counting mode descriptions, **Section 23.2.5.3** to **Section 23.2.5.5** to understand how these bits are set and cleared.

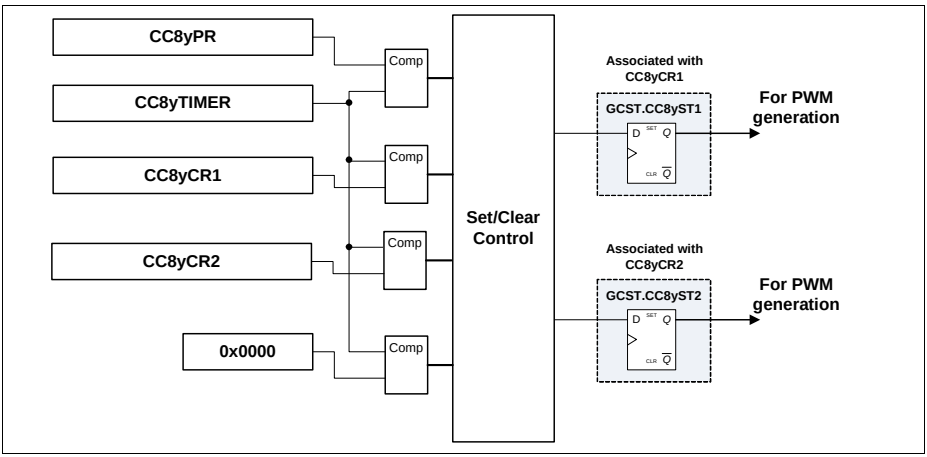


Figure 23-6 CC8y Status Bits

23.2.5.1 Calculating the PWM Period and Duty Cycle

The period of the timer is determined by the value in the period register, **CC8yPR** and by the timer mode.

The base for the PWM signal frequency and duty cycle, is always related to the clock frequency of the timer itself and not to the frequency of the module clock (due to the fact that the timer clock can be a scaled version of the module clock).

In Edge Aligned Mode, the timer period is:

T_{per}= <Period-Value> + 1; in f_{tclk} (23.1)

In Center Aligned Mode, the timer period is:

T_{per}= (<Period-Value> + 1) x 2; in f_{tclk} (23.2)

For each of these counting schemes, the duty cycle of generated PWM signal is dictated by the value programmed into the compare channel registers, **CC8yCR1** and **CC8yCR2**. Notice that one can have different duty cycle values for each of the compare channels.

In Edge Aligned and Center Aligned Mode, the PWM duty cycle is:

DC= 1 - <Compare-Value>/(<Period-Value> + 1) (23.3)

Both the period and compare registers, **CC8yPR**, **CC8yCR1** and **CC8yCR2** respectively, can be updated on the fly via software enabling a glitch free transition between different period and duty cycle values for the generated PWM signal, **Section 23.2.5.2**

23.2.5.2 Updating the Period and Duty Cycle

Each CCU8 timer slice provides an associated shadow register for the period and the two compare values. This facilitates a concurrent update by software for these three parameters, with the objective of modifying during run time the PWM signal period and duty cycle.

In addition to the shadow registers for the period and compare values, one also has available shadow registers for the floating prescaler, dither and passive level, **CC8yFPCS**, **CC8yDITS** and **CC8yPSL** respectively (please address **Section 23.2.13** and **Section 23.2.12** for a complete description of these functions).

The structure of the shadow registers can be seen in **Figure 23-7**.

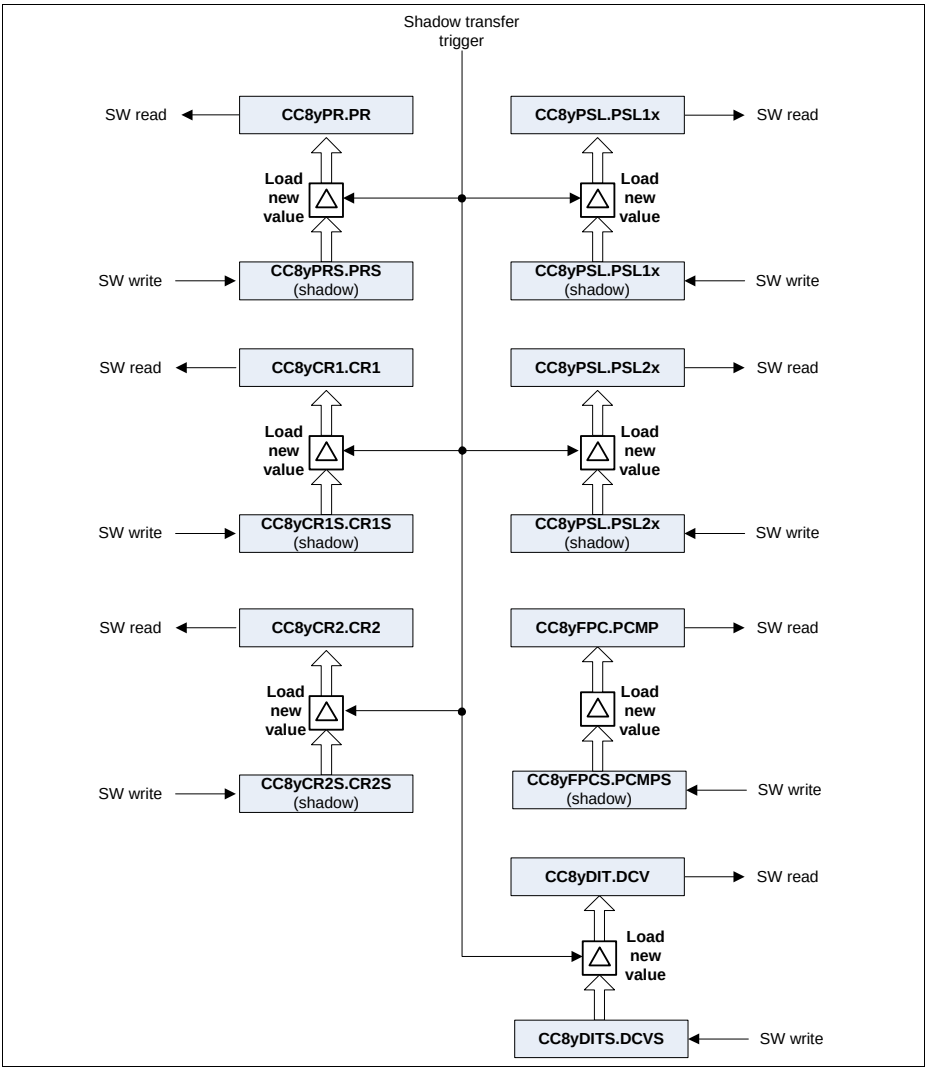


Figure 23-7 Shadow registers overview

The update of these registers can only be done by writing a new value into the associated shadow register and wait for a shadow transfer to occur.

Each group of shadow registers have an individual shadow transfer enable bit, **Figure 23-8**. The software must set this enable bit to 1_B, whenever an update of the

values is needed. These bits are automatically cleared by the hardware, whenever an update of the values if finished. Therefore every time that an update of the registers is needed the software must set again the specific bit(s).

Nevertheless it is also possible to clear the enable bit via software. This can be used in the case that an update of the values needs to be cancelled (after the enable bit has already been set).

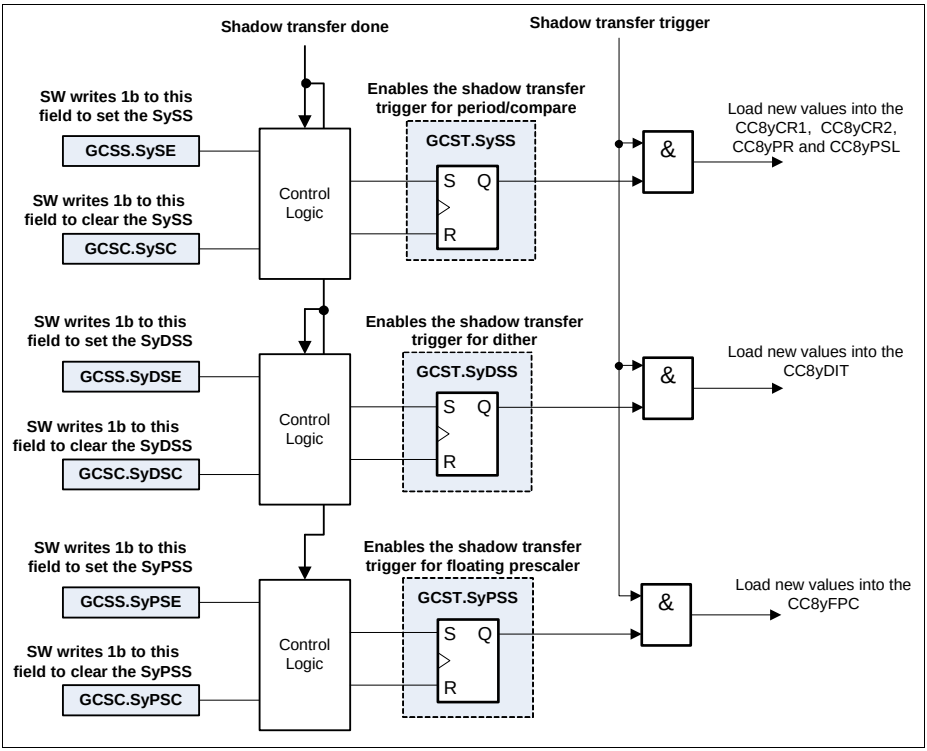


Figure 23-8 Shadow transfer enable logic

The shadow transfer operation is going to be done in the immediately next occurrence of a shadow transfer trigger, after the shadow transfer enable is set (**GCST**.SySS, **GCST**.SyDSS, **GCST**.SyPSS set to 1_B).

The occurrence of the shadow transfer trigger is imposed by the timer counting scheme (edge aligned or center aligned). Therefore the slots when the values are updated can be:

- in the next clock cycle after a Period Match while counting up
- in the next clock cycle after an One Match while counting down

- immediately, if the timer is stopped and the shadow transfer enable bit(s) is set

Figure 23-9 shows an example of the shadow transfer control when the timer slice has been configured into center aligned mode. For a complete description of all the timer slice counting modes, please address **Section 23.2.5.3**, **Section 23.2.5.4** and **Section 23.2.5.5**.

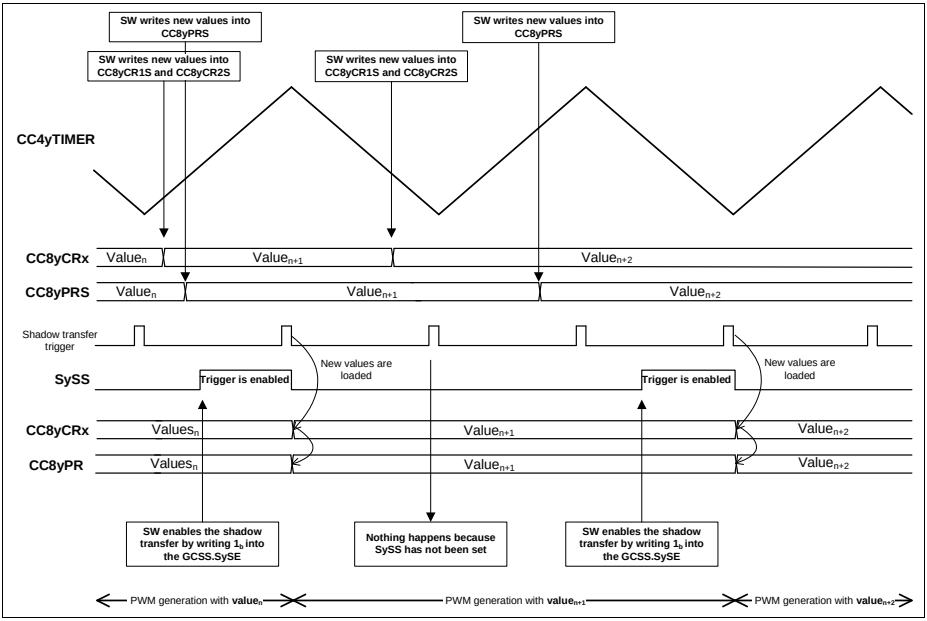


Figure 23-9 Shadow transfer timing example - center aligned mode

When using the CCU8 in conjunction with the POSIF to control the multi channel mode, it can be necessary in some cases, to perform the shadow transfers synchronously with the update of the multi channel pattern. To perform this action, each CCU8 contains a dedicated input that can be used to synchronize the two events, the CCU8x.MCSS.

This input, when enabled, is used to set the shadow transfer enable bitfields (**GCST**.SySS, **GCST**.SyDSS and **GCST**.SyPSS) of the specific slice. It is possible to select which slice is using this input to perform the synchronization via the **GCTRL**.MSEy bit field. It is also possible to enable the usage of this signal for the three different shadow transfer signals: compare and period values, dither compare value and prescaler compare value. This can be configured on the **GCTRL**.MSDE field.

The structure for using the CCU8x.MCSS input signal can be seen in **Figure 23-10**. The usage of this signal is just an add on to the shadow transfer control and therefore all the previous described functions are still available.

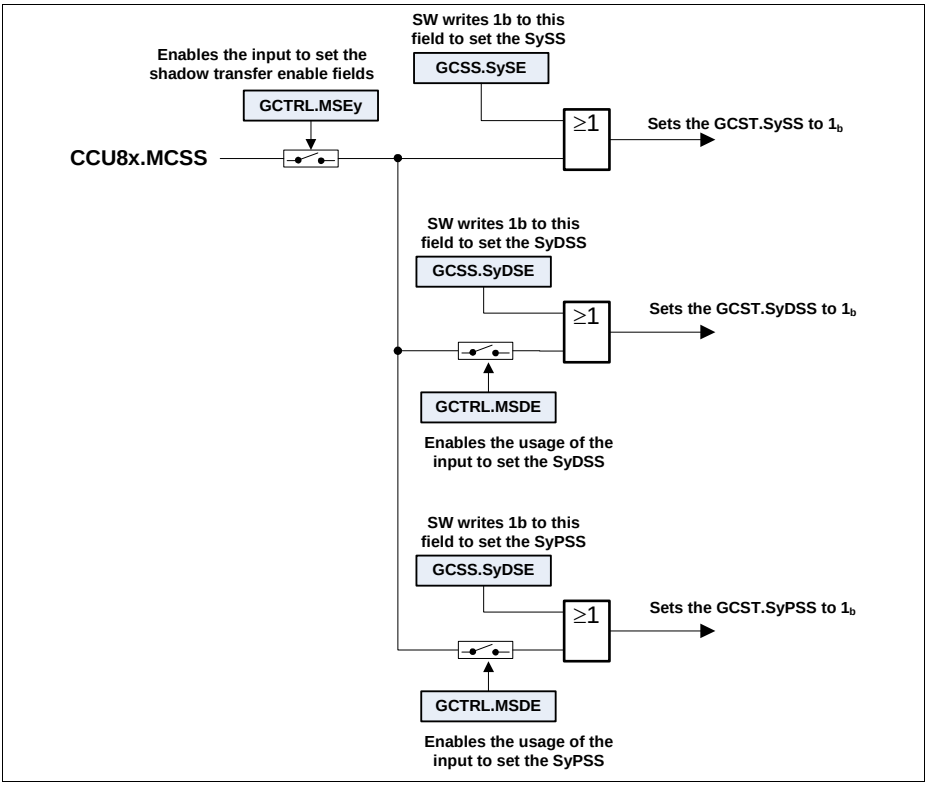


Figure 23-10 Usage of the CCU8x.MCSS input

23.2.5.3 Edge Aligned Mode

Edge aligned mode is the default counting scheme. In this mode, the timer is incremented until it matches the value programmed in the period register, **CC8yPR**. When period match is detected the timer is cleared to 0000_H and continues to be incremented.

In this mode, the value of the period register and compare registers are updated with the values written by software into the correspondent shadow register, every time that an overflow occurs (period match), see **Figure 23-11**.

In edge aligned mode, the status bit of the comparison (CC8ySTx) is set one clock cycle after the timer hits the value programmed into the compare register. The clear of the status bit is done one clock cycle after the timer reaches 0000_H.

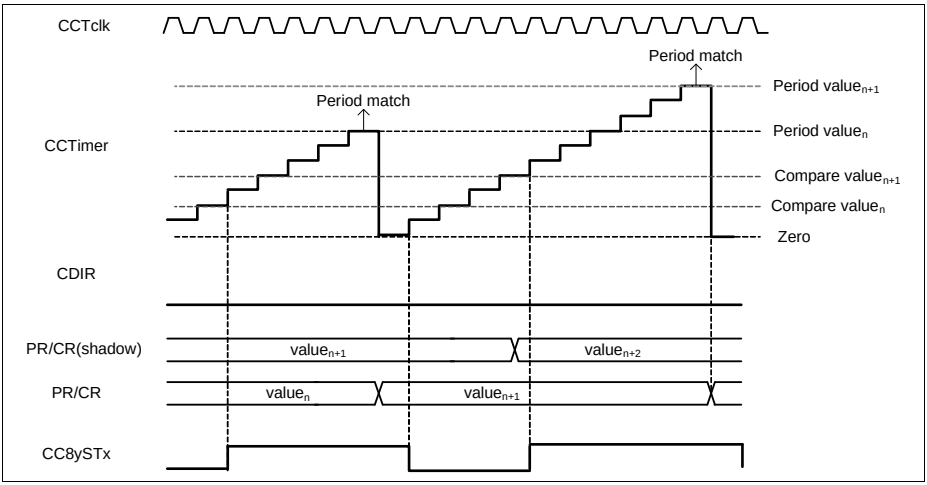


Figure 23-11 Edge aligned mode, CC8yTC.TCM = 0_B

23.2.5.4 Center Aligned Mode

In center aligned mode, the timer is counting up or down with respect to the following rules:

- The counter counts up while **CC8yTCST**.CDIR = 0_B and it counts down while **CC8yTCST**.CDIR = 1_B.
- Within the next clock cycle, the count direction is set to counting up (**CC8yTCST**.CDIR = 0_B) when the counter reaches 0001_H while counting down.
- Within the next clock cycle, the count direction is set to counting down (**CC8yTCST**.CDIR = 1_B), when the period match is detected while counting up.

The status bit (CC8ySTx) is always 1_B when the counter value is equal or greater than the compare value and 0_B otherwise.

While in edge aligned mode, the shadow transfer for compare and period registers is executed once per period. It is executed twice in center aligned mode as follows

- Within the next clock cycle after the counter reaches the period value, while counting up (**CC8yTCST**.CDIR = 0_B).
- Within the next clock cycle after the counter reaches 0001_H, while counting down (**CC8yTCST**.CDIR = 1_B).

*Note: Bit **CC8yTCST**.CDIR changes within the next timer clock after the one-match or the period-match, which means that the timer continues counting in the previous direction for one more cycle before changing the direction.*

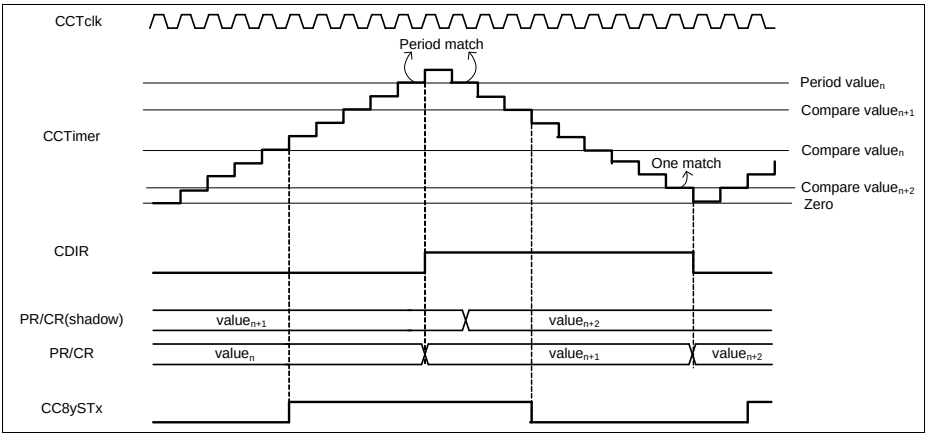


Figure 23-12 Center aligned mode, CC8yTC.TCM = 1_B

23.2.5.5 Single Shot Mode

In single shot mode, the timer is stopped after the current timer period is finished. This mode can be used with a center or edge aligned scheme.

In edge aligned mode, **Figure 23-13**, the timer is stopped when it is cleared to 0000_H after having reached the period value. In center aligned mode, **Figure 23-14**, the period is finished when the timer has counted down to 0000_H.

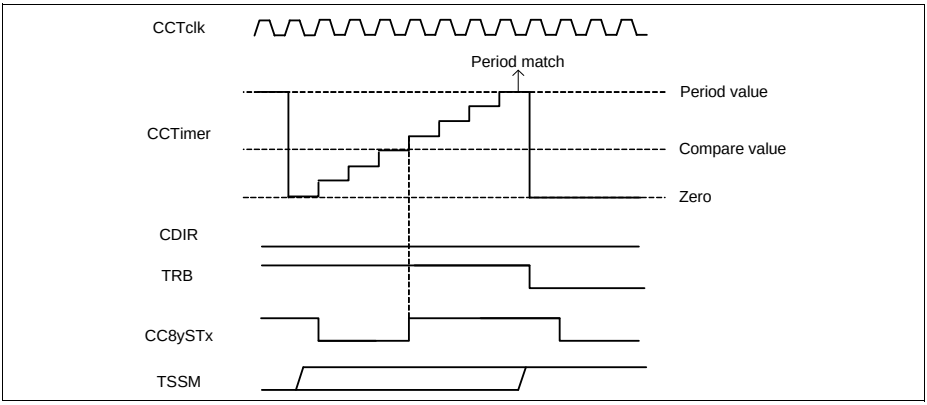


Figure 23-13 Single shot edge aligned - CC8yTC.TSSM = 1_B, CC8yTC.TCM = 0_B

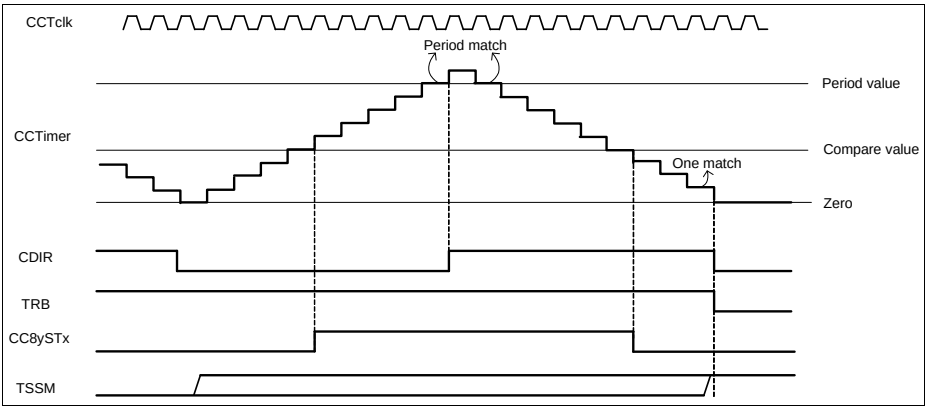


Figure 23-14 Single shot center aligned - CC8yTC.TSSM = 1_B, CC8yTC.TCM = 1_B

23.2.6 Active/Passive Rules

The general rules that set or clear the associated timer slice status bit, can be generalized independently of the timer counting mode.

The following events set the Status bit to Active:

- in the next f_{tclk} cycle after a compare match while counting up
- in the next f_{tclk} cycle after a zero match while counting down

The following events set the Status bit to Inactive:

- in the next f_{tclk} cycle after a zero match (and not compare match) while counting up
- in the next f_{tclk} cycle after a compare match while counting down

If external events are being used to control the timer operation, these rules are still applicable.

The status bit state can only be ‘override’ via software or by the external status bit override function, **Section 23.2.8.9**.

The software at any time can write a 1_B into the **GCSS**.SySTS bitfield, which will set the status bit **GCST**.CC4yST of the specific timer slice. By writing a 1_B into the **GCSC**.SySTC bitfield, the software imposes a clear of the specific status bit.

23.2.7 Compare Modes

Compare Channel Scheme

Each CCU8 slice has two compare channels and two dead time generators, one for each channel, see **Figure 23-15**. Each compare uses the information of the status bit, CC8ySTx, to generate two complementary outputs. All the outputs, CCU8x.OUTy0,

CCU8x.OUTy1, CCU8x.OUTy2 and CCU8x.OUTy3, have a dedicated passive level control bit.

Each compare channel can work in an individual manner for both edge and center aligned modes. This means that two different complementary PWM signals can be generated by using the available compare channels. The PWM frequency is the same for both channels, but the duty cycle can be programmed independently for each channel.

It is also possible to select an asymmetric output scheme, by setting the field **CC8yCHC**.ASE = 1_B. In the asymmetric mode, the compare channels are grouped together to generate a single complementary PWM signal at the CCU8x.OUTy0 and CCU8x.OUTy1 pins.

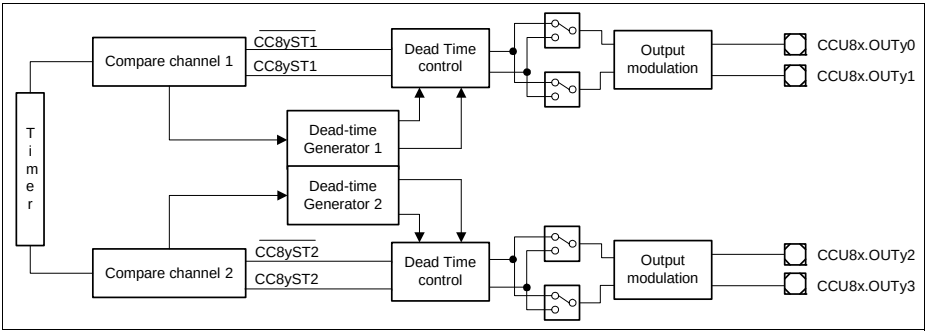


Figure 23-15 Compare channels diagram

Dead Time Generator

In most cases the switching behavior regarding the switch-on and switch-off times is not symmetrical, which can lead to a short circuit if the switch-on time is smaller than the switch-off time. To overcome this problem, each Timer Slice channel contains a dead time generator, which is able to delay the switching edges of the output signals, **Figure 23-16**.

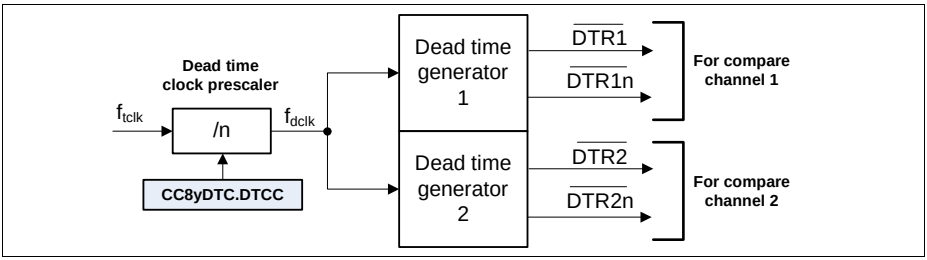


Figure 23-16 Dead Time scheme

Each dead time generator contains an eight bit counter with a different programmable reload value for rise and fall times. The dead time generators contain a programmable prescaler for the dead time counter clock, to enable large dead time insertion values, **Table 23-6**.

Table 23-6 Dead time prescaler values

CC8yDTC.DTCC[1:0]	**Frequency**
00_B	f_{tclk}
01_B	$f_{tclk}/2$
10_B	$f_{tclk}/4$
11_B	$f_{tclk}/8$

Any transition on the associated status bits, CC8ySTx, will trigger the start of the specific dead time generator, **Figure 23-17**.

When a SET (CC8ySTx passes from 0_B to 1_B) action for the CC8ySTx bit is detected, the dead time counter is reloaded with the value present on the **CC8yDC1R**.DT1R or **CC8yDC2R**.DT2R (depending on which channel we are addressing).

When a CLEAR action for the CC8ySTx bit is detected (CC8ySTx passes from 0_B to 1_B), the dead time counter is reloaded with the value present on the **CC8yDC1R**.DT1F or **CC8yDC2R**.DT2F (depending on which channel we are addressing).

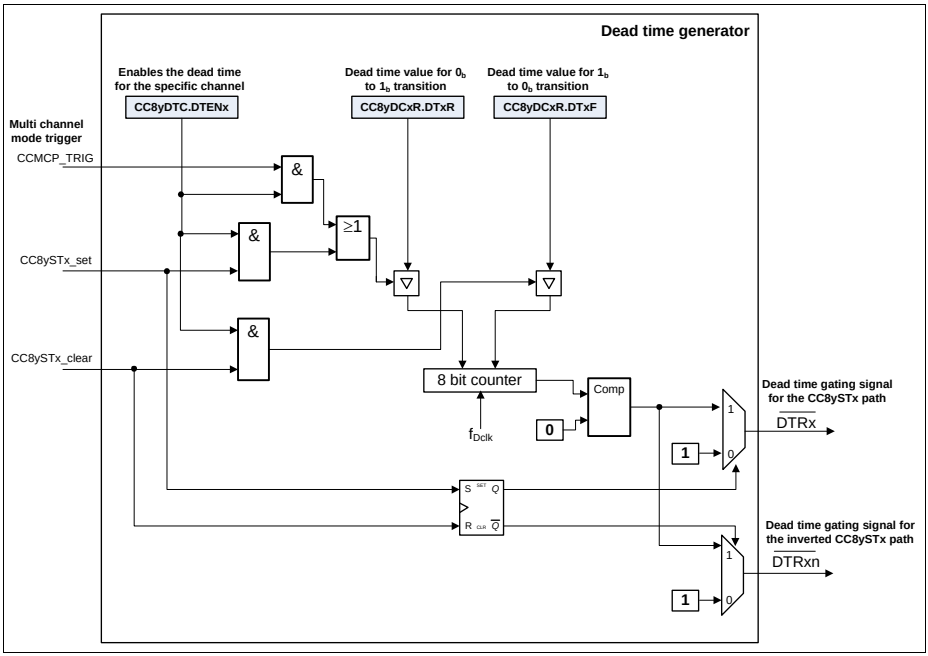


Figure 23-17 Dead Time generator scheme

Each dead time generator outputs two signals that are used to control the two complementary outputs (the CC8ySTx and the inverted CC8ySTx). The separation of the control signals enable a flexible enable/disable scheme inside of each compare channel, **Figure 23-18**. This means that the dead time generator can be enabled for one compare channel, but the dead time insertion can be discarded for one of the outputs.

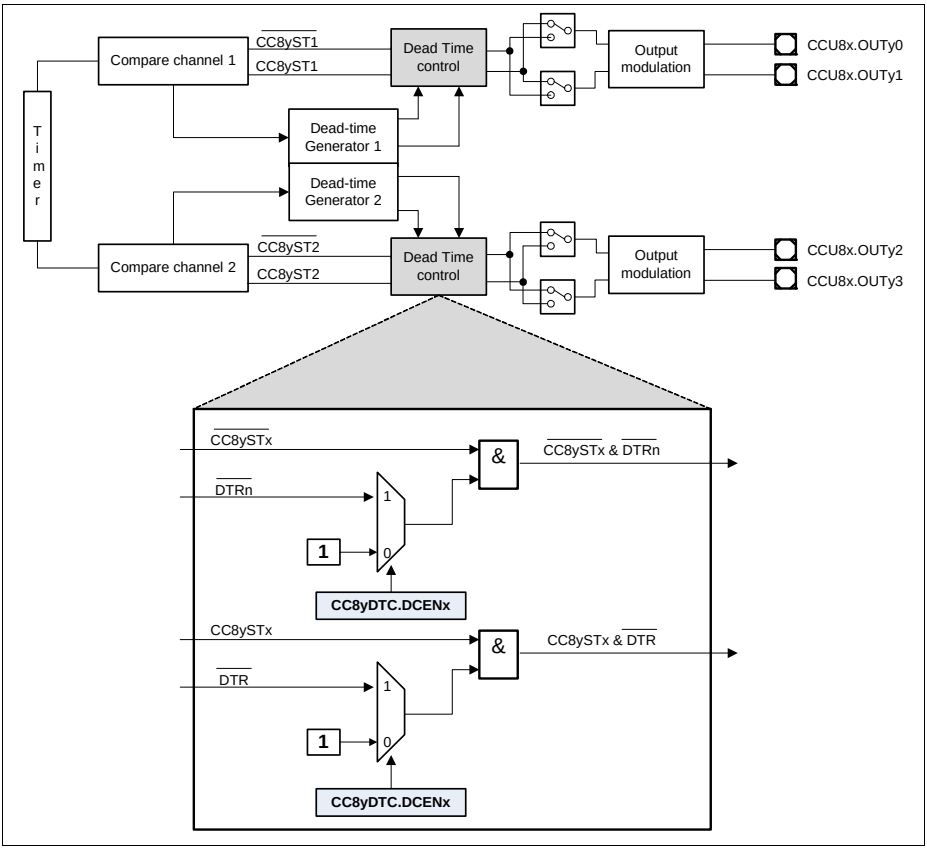


Figure 23-18 Dead Time control cell

When using the Multi Channel mode, **CC8yTC**.MCMEy = 1_B, there can be the scenario where the generated PWM signal has 100% duty cycle. This means that the respective status bit is always set and it is the Multi Channel pattern that is controlling the output modulation. In this case, we can have a transition from Inactive to Active state at the output, without having a transition on the specific status bit, creating a short on the switches due to the non existence of dead time insertion.

To overcome this possible short on the switches, a trigger from the multi channel control, CCMP_TRIG on **Figure 23-17**, is fed to the dead time generators. **Figure 23-19** shows the scheme for the generation of the CCMP_TRIG, where the signals, CCMCMx0 and CCMCMx1 represent the sampled multi channel pattern for a specific channel.

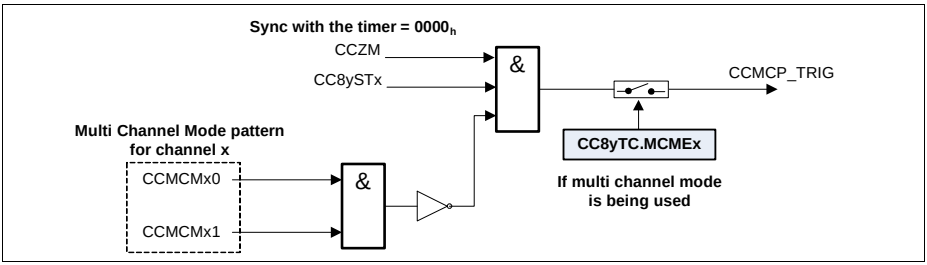


Figure 23-19 Dead Time trigger with the Multi Channel pattern

23.2.7.1 Edge Aligned Compare Modes

Standard Edge Aligned Mode

When the Timer Slice is programmed in edge aligned mode, the two channels can work independently, which means that the compare values can be programmed with different values (originating different duty cycles). In this scenario, each channel can output a pair of PWM signals used to control a high and low side switches, see **Figure 23-20**.

In this mode, for each channel the dead time for rise and fall transitions are controlled by the values programmed in the **CC8yDC1R**.DT1R and **CC8yDC2R**.DT2R, and **CC8yDC1R**.DT1F and **CC8yDC2R**.DT2F fields, respectively.

Figure 23-21 shows the timing diagrams for a specific slice when the compare values of each channel are different.

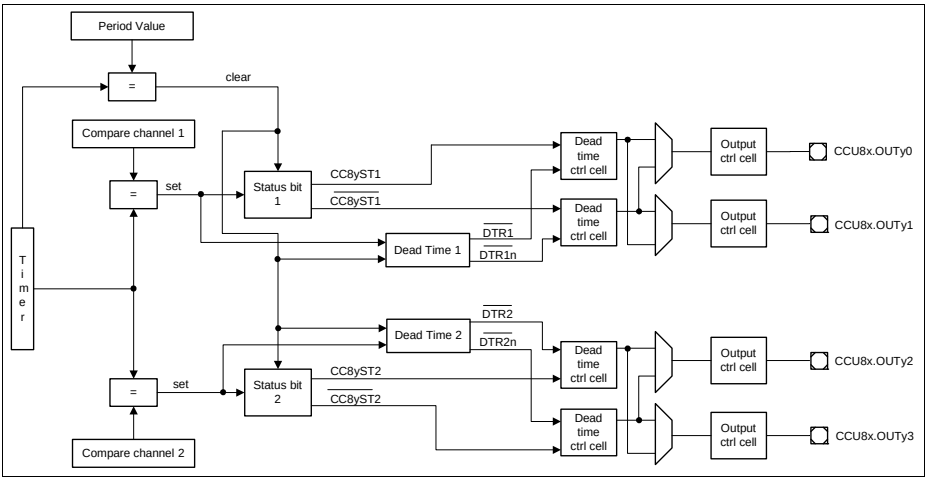


Figure 23-20 Edge Aligned with two independent channels scheme

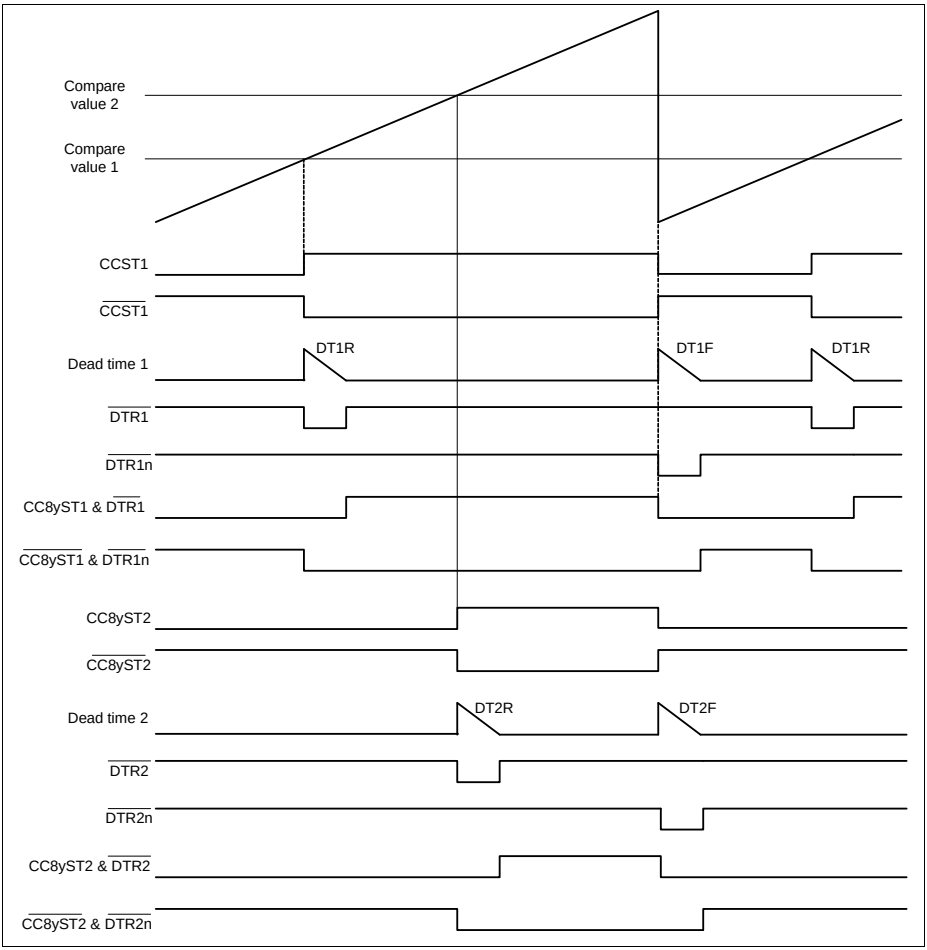


Figure 23-21 Edge Aligned - four outputs with dead time

Asymmetrical Edge Aligned Mode

There is also the possibility of using the two channels combined to generate an asymmetric PWM output. This mode is selected by setting the field **CC8yCHC**.ASE = 1_B.

In this mode, the compare channel 2 is disabled and therefore the outputs linked with this path are always in the passive state.

The status bit of the compare channel 1 is set when a compare match with the compare value 1 (field **CC8yCR1**.CR1) occurs and is cleared when a compare match with the compare value 2 (field **CC8yCR2**.CR2) occurs, see **Figure 23-22**.

When the **CC8yCR2**.CR2 is programmed with a value smaller than the one present in **CC8yCR1**.CR1, the CCST1 bit is always 0_B.

The dead time values for the rising and falling transitions are controlled by the fields **CC8yDC1R**.DT1R and **CC8yDC1R**.DT1F, respectively.

Figure 23-23 and **Figure 23-24** show the timing diagram for the Edge Aligned mode when the asymmetric scheme is active.

Note: When an external signal is used to control the counting direction, the asymmetric mode cannot be enabled.

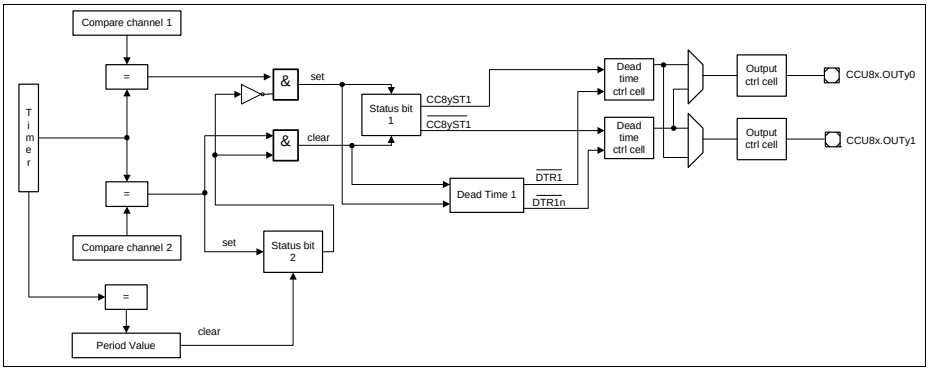


Figure 23-22 Edge Aligned with combined channels scheme

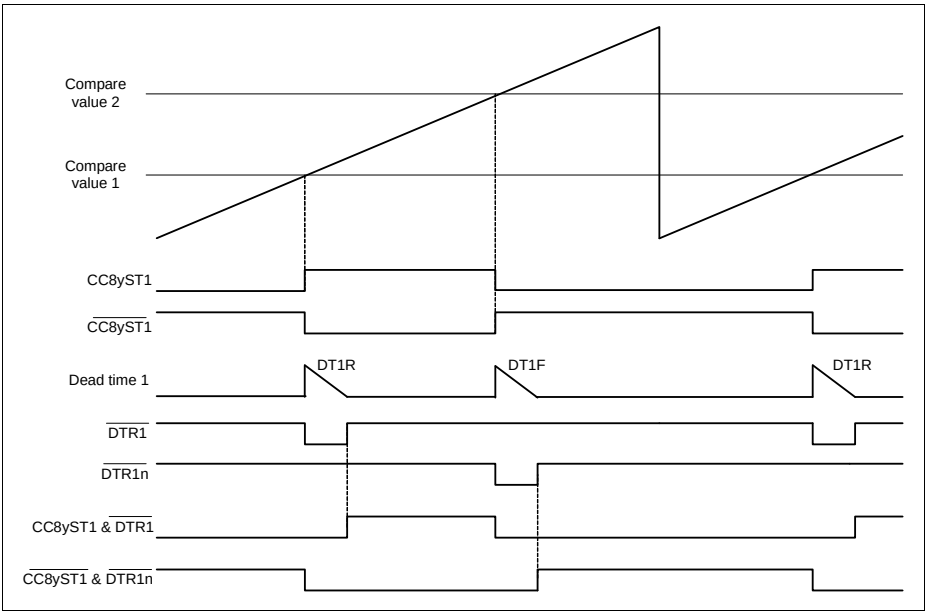


Figure 23-23 Edge Aligned - Asymmetric PWM timing, CC8yCR1.CR1 < CC8yCR2.CR2

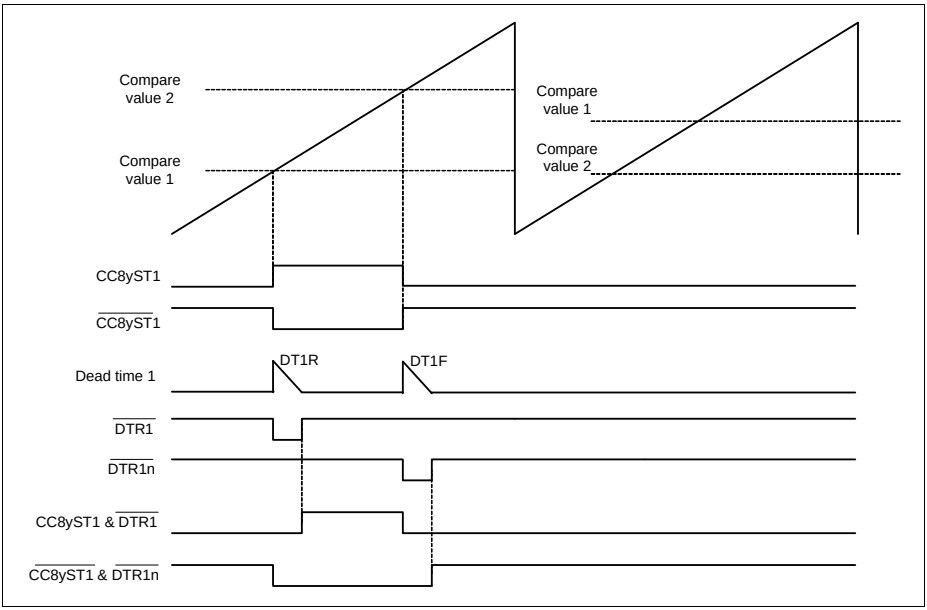


Figure 23-24 Edge Aligned - Asymmetric PWM timing, CC8yCR1.CR1 > CC8yCR2.CR2

23.2.7.2 Center Aligned Compare Modes

Standard Center Aligned Mode

In center aligned mode, like in edge aligned, it is possible to use the two compare channels independently. In this mode, each channel can generate a pair of PWM complementary signals with different duty cycle values, controlled via the **CC8yCR1** for channel 1 and **CC8yCR2** for channel 2.

For the dead time insertion, each channel as a pair of programmable values for the rise and fall transitions: **CC8yDC1R**.DT1R and **CC8yDC1R**.DT1F for channel 1; **CC8yDC2R**.DT2R and **CC8yDC2R**.DT2F for channel 2.

The major difference between the center and the edge aligned mode is directly linked to the set/clear logic of the status bit, see **Section 23.2.5**.

Figure 23-25 shows the scheme for both channels for this operating mode and **Figure 23-26** shows the timing diagrams for a specific channel.

Note: When an external signal is used to control the counting direction, the counting scheme is always edge aligned.

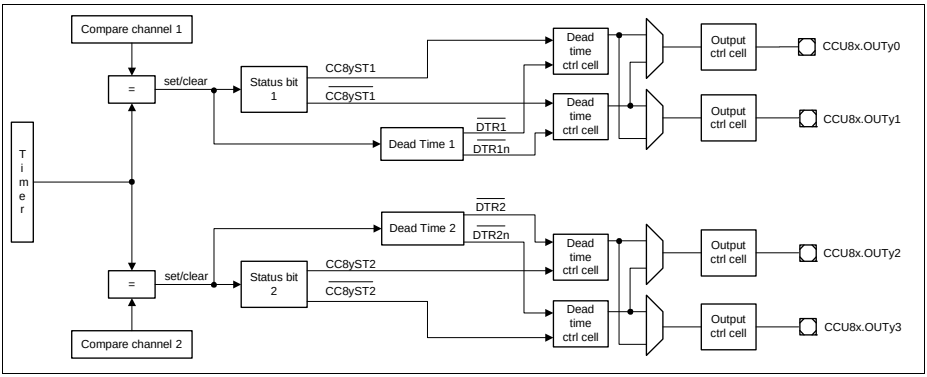


Figure 23-25 Center Aligned with two independent channels scheme

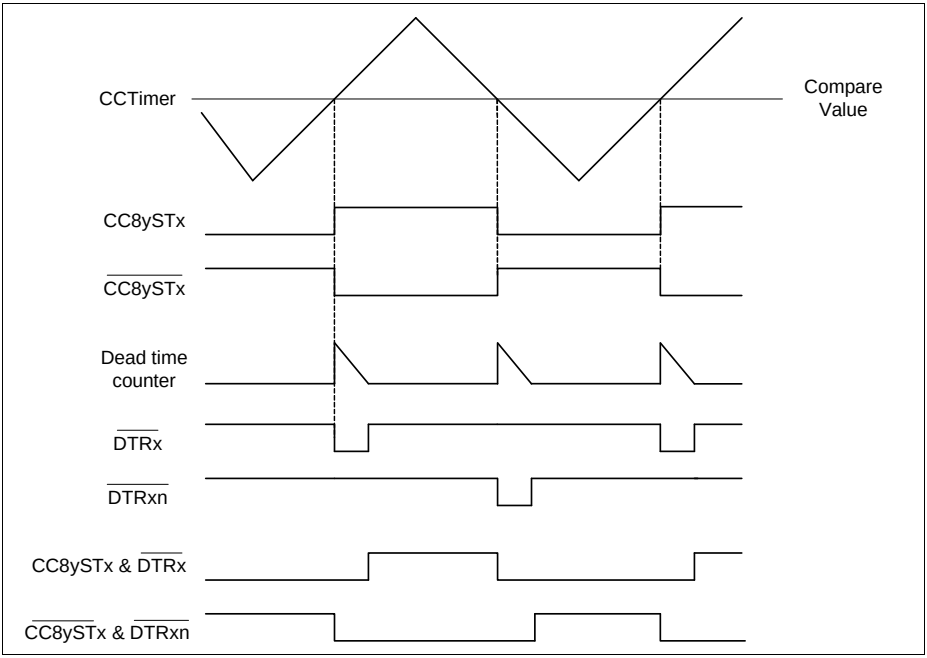


Figure 23-26 Center aligned - Independent channel with dead time

Asymmetrical Center Aligned Mode

The asymmetric mode is enabled in center aligned by setting the field **CC8yCHC**.ASE to 1_B.

In this mode, like in Edge Aligned, the outputs linked with the compare channel 2 are set to their passive levels.

The status bit, CC8yST1, is set when a compare match of channel 1 occurs while counting up, and is cleared when a compare match of channel 2 occurs while counting down, see **Figure 23-27**.

The dead time rise and fall times are controlled by the values programmed into the fields, **CC8yDC1R**.DT1R and **CC8yDC1R**.DT1F, respectively.

Figure 23-28 shows the timing diagram for the asymmetric mode. Notice that even in asymmetric mode the dead time can be disabled in each of the outputs independently.

Note: When an external signal is used to control the counting direction, the asymmetric mode cannot be enabled.

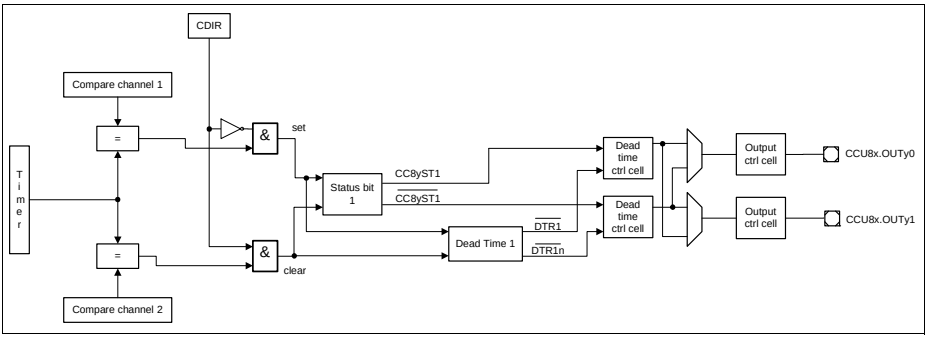


Figure 23-27 Center Aligned Asymmetric mode scheme

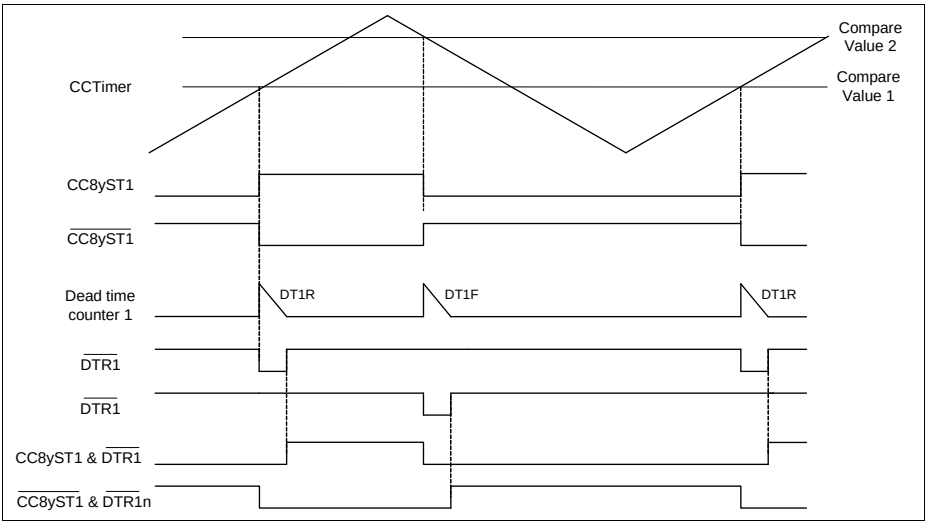


Figure 23-28 Asymmetric Center aligned mode with dead time

23.2.8 External Events Control

Each CCU8 slice has the possibility of using up to three different input events, see **Section 23.2.2**. These three events can then be mapped to Timer Slice functions (the full set of available functions is described at **Section 23.2.3**).

These events can be mapped to any of the CCU8x.INy[P...A] inputs and there isn't any imposition that an event cannot be used to perform several functions or, that an input cannot be mapped to several events (e.g. input X triggers event 0 with rising edge and triggers event 1 with the falling edge).

23.2.8.1 External Start/Stop

To select an external start function, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC8yINS**.EVxIS register and indicating the active edge of the signal on the **CC8yINS**.EVxEM register.

This event should be then mapped to the start or stop functionality by setting the **CC8yCMC**.STRTS (for the start) or the **CC8yTC**.ENDM (for the stop) with the proper value.

The same procedure is applicable to the stop functionality.

Notice that both start and stop functions are edge and not level active and therefore the active/passive configuration is set only by the **CC8yINS**.EVxEM.

The external stop by default just clears the run bit (**CC8yTCST**.TRB), while the start functions does the opposite. Nevertheless one can select an extended subset of functions for the external start and stop. This subset is controlled by the registers **CC8yTC**.ENDM (for the stop) and **CC8yTC**.STRM (for the start).

For the start subset (**CC8yTC**.STRM):

- sets the run bit/starts the timer (resume operation)
- clears the timer, sets the run bit/starts the timer (flush and start)

For the stop subset (**CC8yTC**.ENDM):

- clears the run/stops the timer (stop)
- clears the timer (flush)
- clears the timer, clears the run bit/stops the timer (flush and stop)

If in conjunction with an external start/stop (configured also/only as flush) and external up/down signal is used, during the flush operation the timer is going to be set to 0000_H if the actual counting direction is up or set with the value of the period register if the counting direction is down.

Figure 23-29 to **Figure 23-32** shows the usage of two signals to perform the start/stop functions in all the previously mentioned subsets. External Signal(1) acts as an active HIGH start signal, while External Signal(2) is used as an active HIGH stop function.

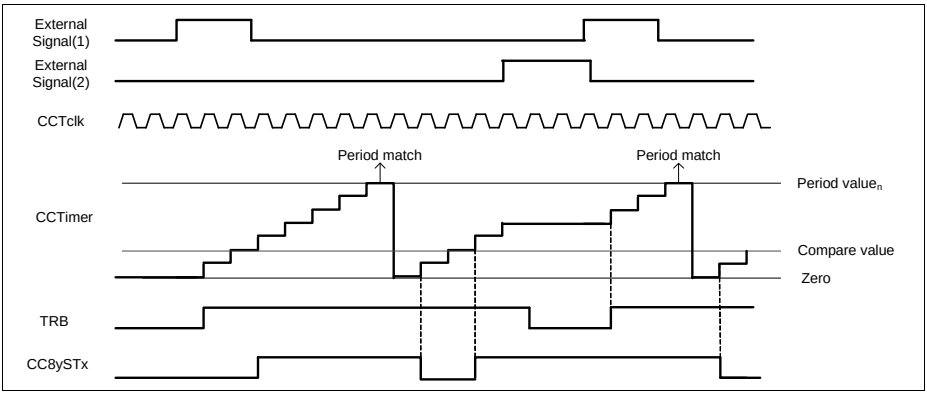


Figure 23-29 Start (as start)/ stop (as stop) - CC8yTC.STRM = 0_B, CC8yTC.ENDM = 00_B

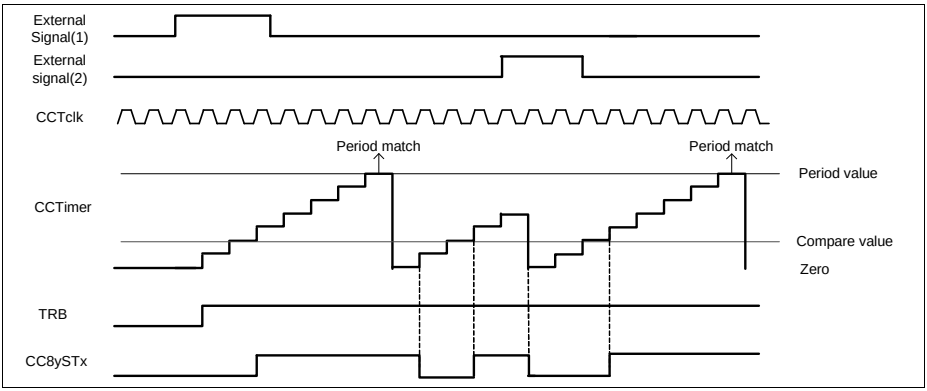


Figure 23-30 Start (as start)/ stop (as flush) - CC8yTC.STRM = 0_B, CC8yTC.ENDM = 01_B

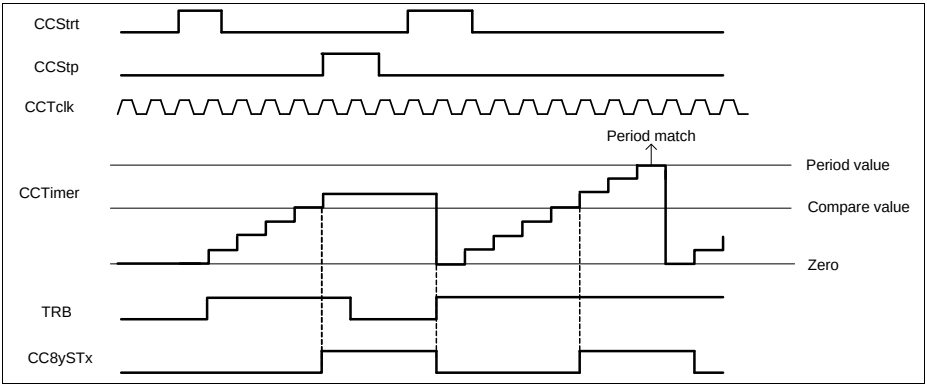


Figure 23-31 Start (as flush and start)/ stop (as stop) - CC8yTC.STRM = 1_B, CC8yTC.ENDM = 00_B

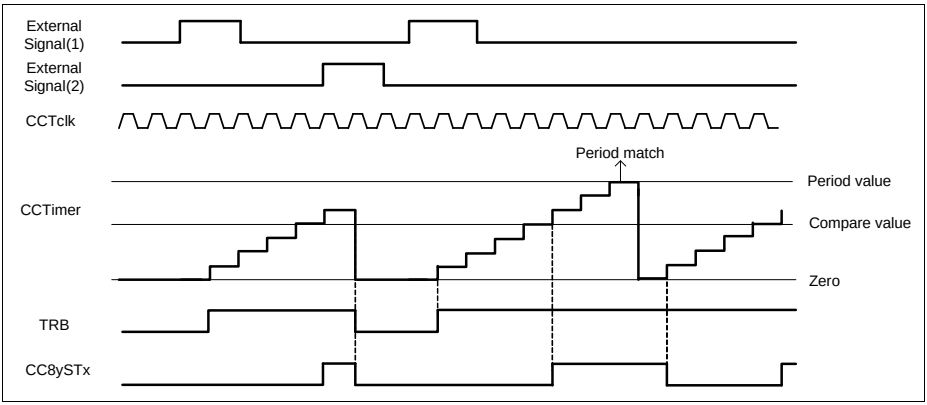


Figure 23-32 Start (as start)/ stop (as flush and stop) - CC8yTC.STRM = 0_B, CC8yTC.ENDM = 10_B

23.2.8.2 External Counting Direction

There is the possibility of selecting an input signal to act as counting up/counting down control.

To select an external up/down control, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC8yINS**.EVxIS register and indicating the active edge of the signal on the **CC8yINS**.EVxEM register. This event should be then mapped to the up/down functionality by setting the **CC8yCMC**.UDS with the proper value.

Notice that the up/down function is level active and therefore the active/passive configuration is set only by the **CC8yINS**.EVxLM.

The status bit of the slice (CCSTx) is always set when the timer value is equal or greater than the value stored in the compare register, see **Section 23.2.6**.

The update of the period and compare register values is done when:

- with the next clock after a period match, while counting up (**CC8yTCST**.CDIR = 0_B)
- with the next clock after a one match, while counting down (**CC8yTCST**.CDIR = 1_B)

The value of the **CC8yTCST**.CDIR register is updated accordingly with the changes on the decoded event. The Up/Down direction is always understood as **CC8yTCST**.CDIR = 1 when counting down and **CC8yTCST**.CDIR = 0_B when counting up. Using an external signal to perform the up/down counting function and configuring the event as active HIGH means that the timer is counting up when the signal is HIGH and counting down when LOW.

Figure 23-33 shows an external signal being used to control the counting direction of the time. This signal was selected as active HIGH, which means that the timer is counting down while the signal is HIGH and counting up when the signal is LOW.

*Note: For a signal that should impose an increment when LOW and a decrement when HIGH, the user needs to set the **CC8yINS**.EVxLM = 0_B. When the operation is switched, then the user should set **CC8yINS**.EVxLM = 1_B.*

Note: Using an external counting direction control, sets the slice in edge aligned mode.

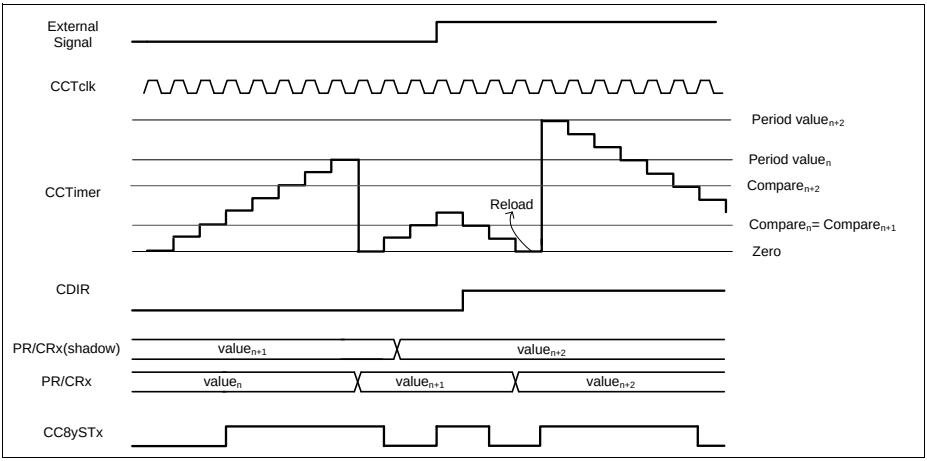


Figure 23-33 External counting direction

23.2.8.3 External Gating Signal

For pulse measurement, the user has the possibility of selecting an input signal that operates as counting gating.

To select an external gating control, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC8yINS**.EVxIS register and indicating the active level of the signal on the **CC8yINS**.EVxLM register. This event should be then mapped to the up/down functionality by setting the **CC8yCMC**.GATES with the proper value.

Notice that the gating function is level active and therefore the active/passive configuration is set only by the **CC8yINS**.EVxLM.

The status bit during an external gating signal continues to be asserted when the compare value is reached and deasserted when the counter reaches 0000_H. One should note that the counter continues to use the period register to identify a wrap around condition. **Figure 23-34** shows the usage of an external signal for gating the slice

counter. The signal was set as active LOW, which means the counter gating functionality is active when the external value is zero.

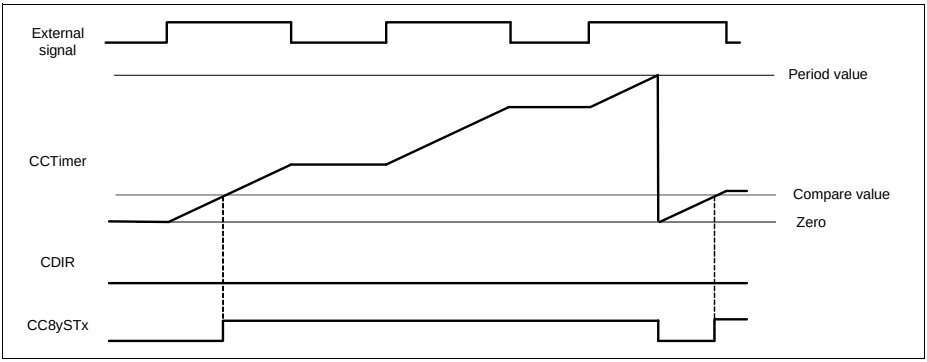


Figure 23-34 External gating

23.2.8.4 External Count Signal

There is also the possibility of selecting an external signal to act as the counting event.

To select an external counting, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC8yINS**.EVxIS register and indicating the active edge of the signal on the **CC8yINS**.EVxEM register. This event should be then mapped to the up/down functionality by setting the **CC8yCMC**.CNTS with the proper value.

Notice that the counting function is edge active and therefore the active/passive configuration is set only by the **CC8yINS**.EVxEM.

One can select just a the rising, falling or both edges to perform a count. On **Figure 23-35**, the external signal was selected as a counter event for both falling and rising edges. Wrap around condition is still applied with a comparison with the period register.

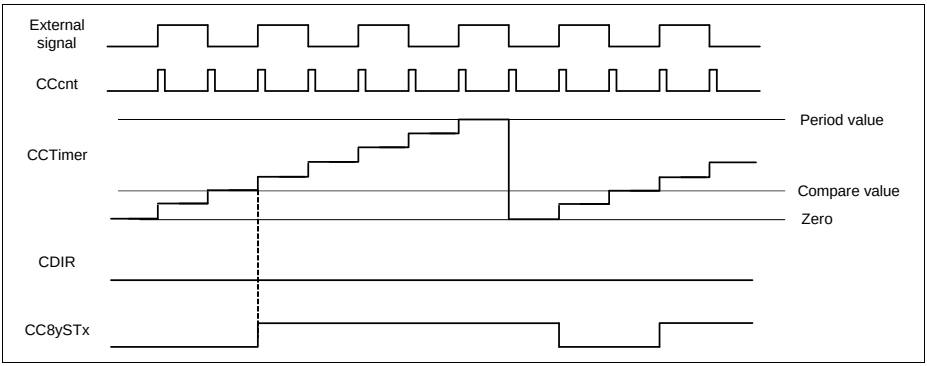


Figure 23-35 External count

23.2.8.5 External Load

Each slice of the CCU8 also has a functionality that enables the user to select an external signal as trigger for reloading the value of the timer with the current value of one compare register (if **CC8yTCST**.CDIR = 0_B) or with the value of the period register (if **CC8yTCST**.CDIR = 1_B).

The timer can be reloaded with the value from the compare channel 1 or compare channel 2 depending on the value set in the **CC8yTC**.TLS field, see **Figure 23-36**.

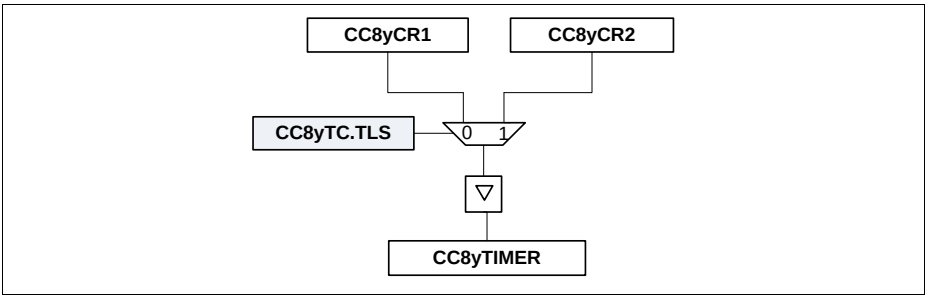


Figure 23-36 Timer load selection

To select an external load signal, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC8yINS**.EVxIS register and indicating the active edge of the signal on the **CC8yINS**.EVxEM register. This event should be then mapped to the up/down functionality by setting the **CC8yCMC**.LDS with the proper value.

Notice that the load function is edge active and therefore the active/passive configuration is set only by the **CC8yINS**.EVxEM.

On figure **Figure 23-37**, the external signal (1) was used to act as a load trigger, active on the rising edge. Every time that a rising edge on external signal (1) is detected, the timer value is loaded with the value present on the compare register. If an external signal is being used to control the counting direction, up or down, the timer value can be loaded also with the value set in the period register. The External signal (2) represents the counting direction control (active HIGH). If at the moment that a load trigger is detected, the signal controlling the counting direction is imposing a decrement, then the value set in the timer is the period value.

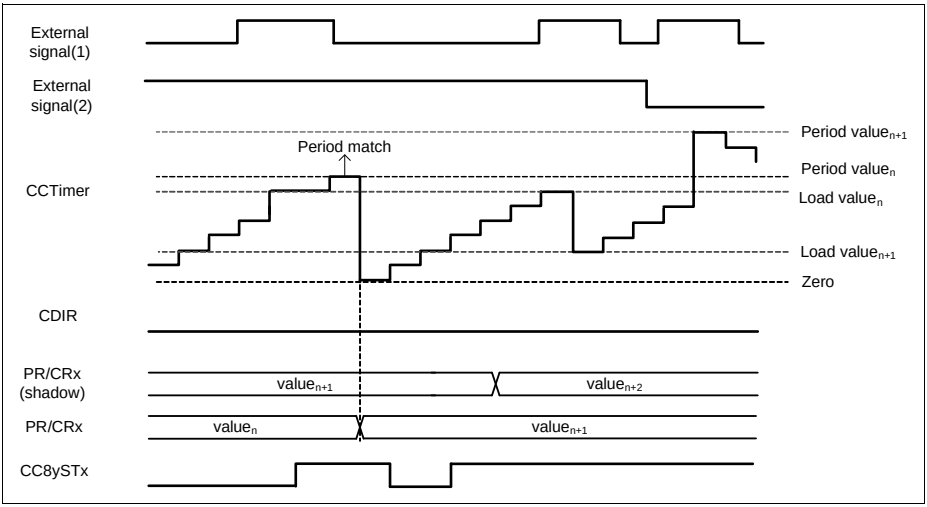


Figure 23-37 External load

23.2.8.6 External Capture

When selecting an external signal to be used as a capture trigger (if **CC8yCMC**.CAP0S or **CC8yCMC**.CAP1S are different from 0_H), the user is automatically setting the specific slice into capture mode.

In capture mode the user can have up to four capture registers, see **Figure 23-40**: capture register 0 (**CC8yC0V**), capture register 1 (**CC8yC1V**), capture register 2 (**CC8yC2V**) and capture register 3 (**CC8yC3V**).

These registers are shared between compare and capture modes, which imposes:

- if **CC8yC0V** and **CC8yC1V** are used for capturing, the compare registers **CC8yCR1** and **CC8yCR1S** are not available (compare channel 1 is not available)
- if **CC8yC2V** and **CC8yC3V** are used for capturing, the compare registers **CC8yCR2** and **CC8yCR2S** are not available (compare channel 2 is not available)

To select an external capture signal, one should map one of the events (output of the input selector) to a specific input signal, by setting the required value in the **CC8yINS**.EVxIS register and indicating the active edge of the signal on the **CC8yINS**.EVxEM register.

This event should be then mapped to the capture functionality by setting the **CC8yCMC**.CAP0S/**CC8yCMC**.CAP1S with the proper value.

Notice that the capture function is edge active and therefore the active/passive configuration is set only by the **CC8yINS**.EVxEM.

The user has the possibility of selecting the following capture schemes:

- Different capture events for **CC8yC0V**/**CC8yC1V** and **CC8yC2V**/**CC8yC3V**
- The same capture event for **CC8yC0V**/**CC8yC1V** and **CC8yC2V**/**CC8yC3V** with the same capture edge. For this capture scheme, only the CCcapt1 functionality needs to be programmed. To enable this scheme, the field **CC8yTC**.SCE needs to be set to 1_B.

Different Capture Events (SCE = 0_B)

Every time that a capture trigger 1 occurs, CCcapt1, the actual value of the timer is captured into the capture register 3 and the previous value stored in this register is transferred into capture register 2.

Every time that a capture trigger 0 occurs, CCcapt0, the actual value of the timer is captured into the capture register 1 and the previous value stored in this register is transferred into capture register 0.

Every time that a capture procedure into one of the registers occurs, the respective full flag is set. This flag is cleared automatically by HW when the SW reads back the value of the capture register (by reading the specific CC8yCxV register or by reading the extended capture read value, **ECRD** if **CC8yTC**.ECM = 1_B).

The capture of a new value into a specific capture registers is dictated by the status of the full flag as follows:

$$CC8yC1V_{capt} = \text{NOT}(CC8yC1V_{full\_flag} \text{ AND } CC8yC0V_{full\_flag}) \quad (23.4)$$

$$CC8yC0V_{capt} = CC8yC1V_{full\_flag} \text{ AND NOT}(CC8yC0V_{full\_flag}) \quad (23.5)$$

It is also possible to disable the effect of the full flags by setting the **CC8yTC**.CCS = 1_B. This enables a continuous capturing independent if the values captured have been read or not.

On **Figure 23-38**, an external signal was selected as an event for capturing the timer value into the **CC8yC0V**/**CC8yC1V** registers. The status bit, CC8ySTx, during capture mode is asserted whenever a capture trigger is detected and de asserted when the counter matches 0000_H.

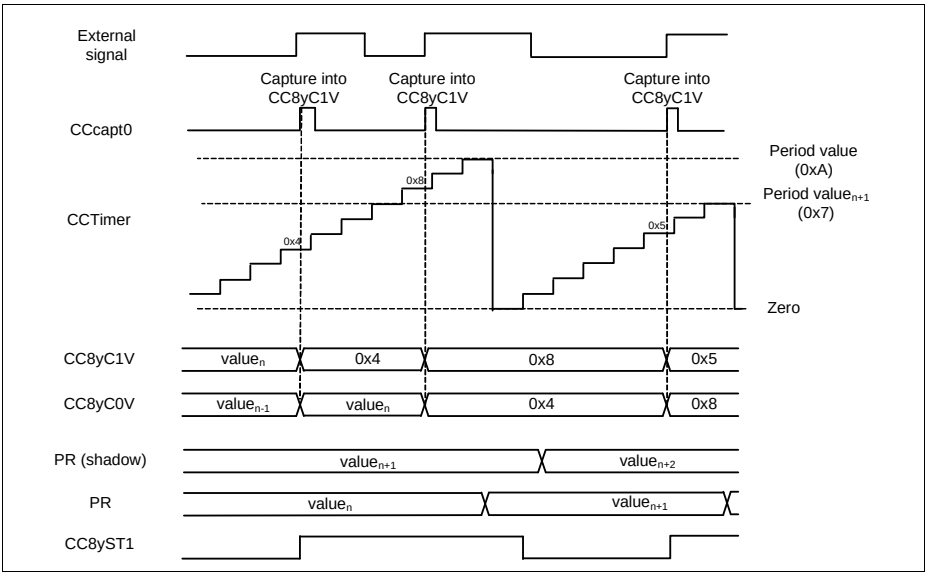


Figure 23-38 External capture - CC8yCMC.CAP0S != 00_B, CC8yCMC.CAP1S = 00_B

On **Figure 23-39**, two different signals were used as trigger sources for capturing the timer value into the **CC8yC0V**/**CC8yC1V** and **CC8yC2V**/**CC8yC3V** registers. External signal(1) was selected as an rising edge active source for the channel 1 capture trigger. External signal(2) was selected has the source for the channel 2 capture trigger, but as opposite to the external signal(1), the active edge was set as falling.

See **Section 23.2.14.4** for the complete capture mode usage description.

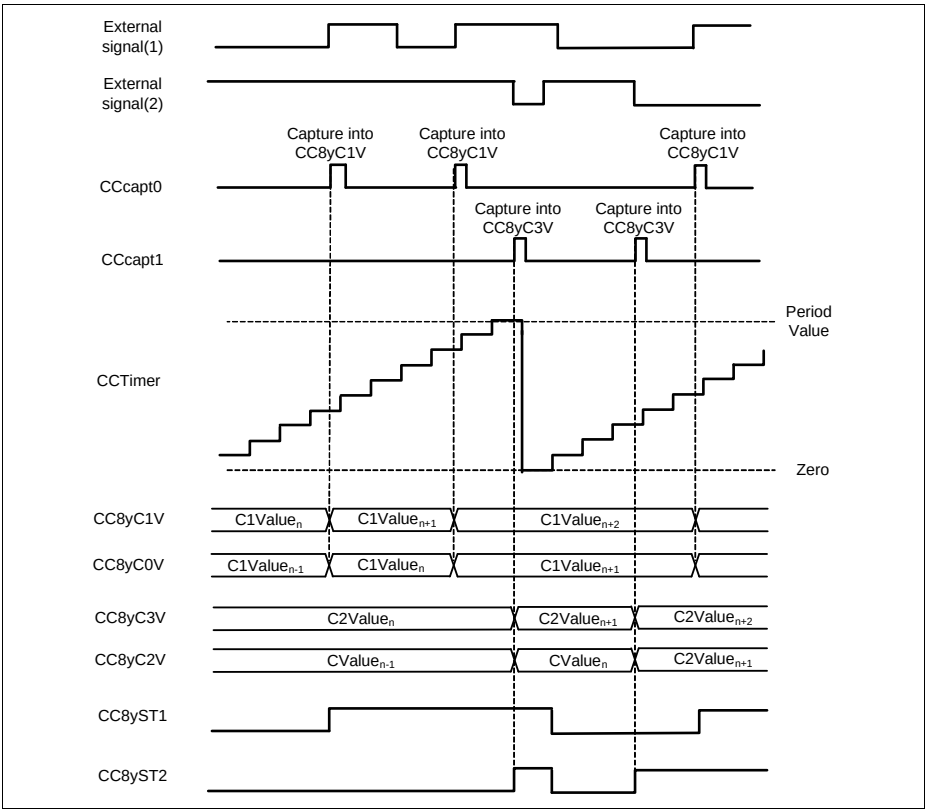


Figure 23-39 External capture - CC8yCMC.CAP0S != 00_B, CC8yCMC.CAP1S != 00_B

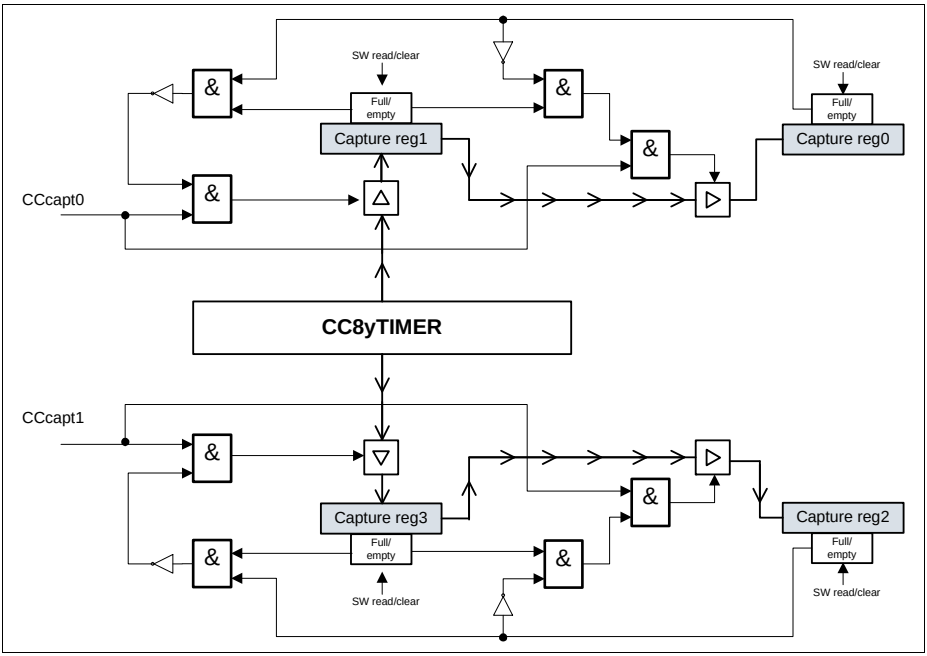


Figure 23-40 Slice capture logic

Same Capture Event (SCE = 1_B)

Setting the field **CC8yTC**.SCE = 1_B, enables the possibility of having 4 capture registers linked with the same capture event, **Figure 23-42**.The functionality that controls the capture is the CCcapt1.

The capture logic follows the same structure shown in **Figure 23-40** but extended to a four register chain, see **Figure 23-41**.The same full flag lock rules are applied to the four register chain (it also can be disabled by setting the **CC8yTC**.CCS = 1_B):

$$CC8yC3V_{capt} = \text{NOT}(CC8yC3V_{full\_flag} \text{ AND } CC8yC2V_{full\_flag} \text{ AND } CC8yC2V_{full\_flag} \text{ AND } CC8yC1V_{full\_flag}) \quad (23.6)$$

$$CC8yC2V_{capt} = CC8yC3V_{full\_flag} \text{ AND NOT}(CC8yC2V_{full\_flag} \text{ AND } CC8yC1V_{full\_flag} \text{ AND } CC8yC0V_{full\_flag}) \quad (23.7)$$

$$CC8yC1V_{capt} = CC8yC2V_{full\_flag} \text{ AND NOT}(CC8yC1V_{full\_flag} \text{ AND } CC8yC0V_{full\_flag}) \quad (23.8)$$

$$CC8yC0V_{capt} = CC8yC1V_{full\_flag} \text{ AND NOT}(CC8yC0V_{full\_flag}) \quad (23.9)$$

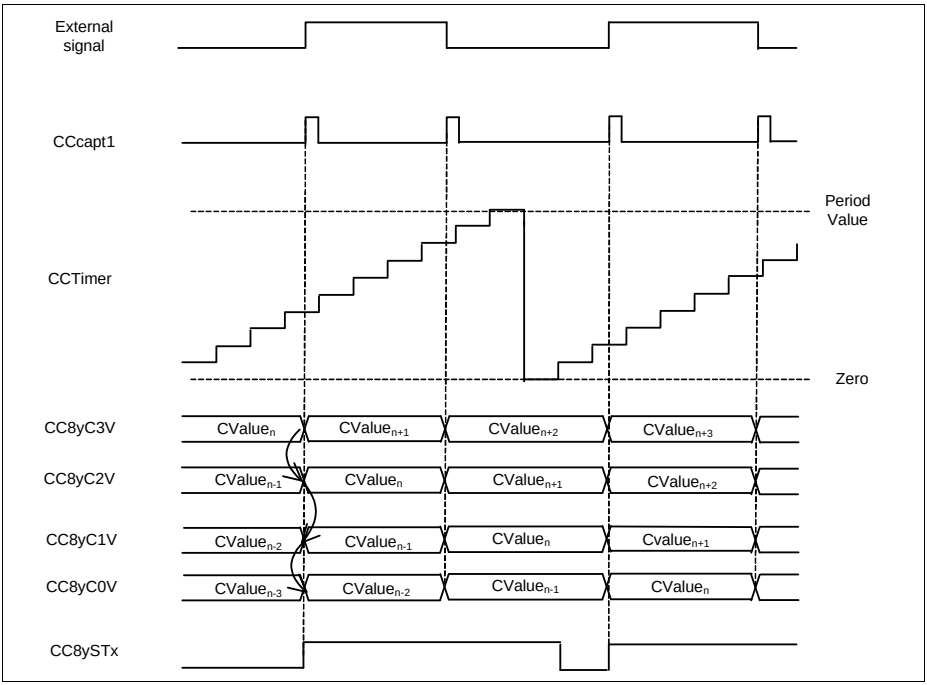


Figure 23-41 External Capture - CC8yTC.SCE = 1_B

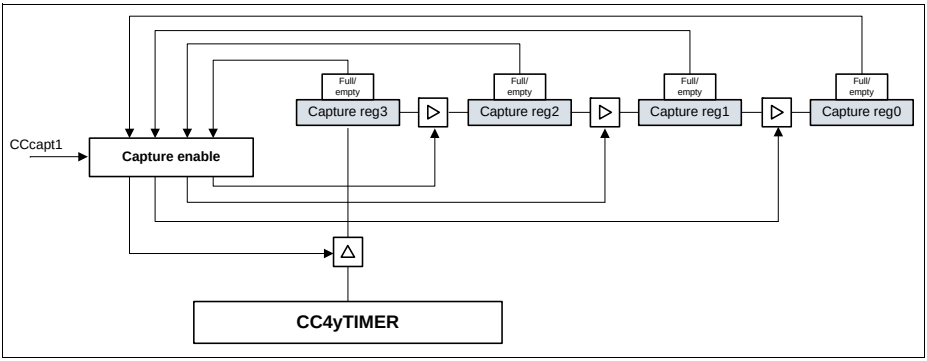


Figure 23-42 Slice Capture Logic - CC8yTC.SCE = 1_B

23.2.8.7 External Modulation

An external signal can be used also to perform a modulation at the output of each slice.

To select an external modulation signal, one should map one of the input signals to one of the events, by setting the required value in the **CC8yINS**.EVxIS register and indicating the active level of the signal on the **CC8yINS**.EVxLM register. This event should be then mapped to the modulation functionality by setting the **CC8yCMC**.MOS = 01_B if event 0 is being used, **CC8yCMC**.MOS = 10_B if event 1 or **CC8yCMC**.MOS = 11_B if event 2.

Notice that the modulation function is level active and therefore the active/passive configuration is set only by the **CC8yINS**.EVxLM.

The external modulation signal can be applied to each compare channel independently, or it can be applied to both channels, by setting **CC8yTC**.EME = 11_B.

The modulation has two modes of operation:

- modulation event is used to reset the CC8ySTx bit - **CC8yTC**.EMT = 0_B
- modulation event is used to gate the outputs - **CC8yTC**.EMT = 1_B

On **Figure 23-43**, we have a external signal configured to act as modulation source that clears the ST bit, **CC8yTC**.EMT = 0_B. It was programmed to be an active LOW event and therefore, when this signal is LOW the output value is following the normal ACTIVE/PASSIVE rules.

When the signal is HIGH (inactive state), then the CC8ySTx bit is cleared and the output is force into the PASSIVE state. Notice that the values of the status bit, CC8ySTx and the specific output CCU8x.OUTy are not linked together. One can choose for the output to be active LOW through the **CC8yPSL**.PSLx bit.

The exit of the external modulation inactive state is synchronized with the PWM period due to the fact that the CC8ySTx bit is cleared and cannot be set while the modulation signal is inactive.

The entering into inactive state also can be synchronized with the PWM period, by setting **CC8yTC**.EMS = 1_B. With this all possible glitches at the output are avoided, see **Figure 23-44**.

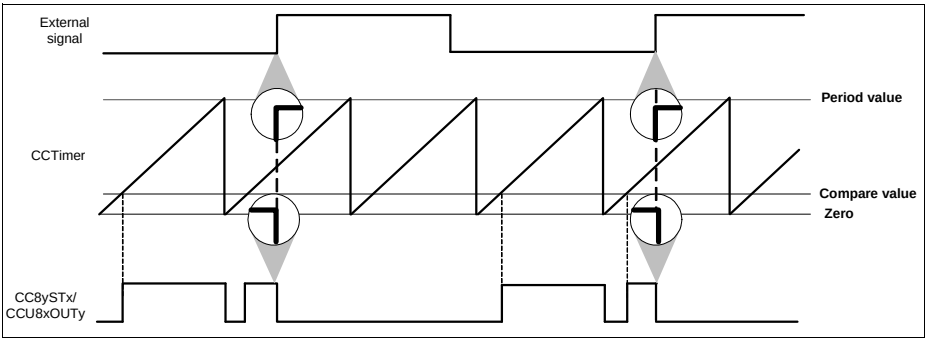


Figure 23-43 External modulation resets the ST bit - CC8yTC.EMS = 0_B

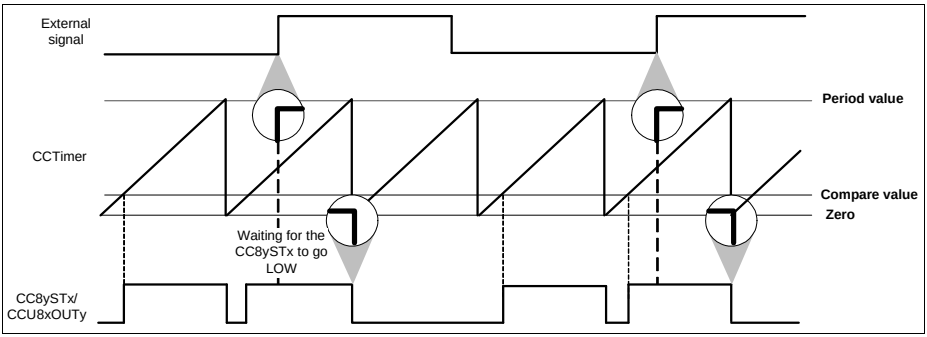


Figure 23-44 External modulation clearing the ST bit - CC8yTC.EMS = 1_B

On **Figure 23-45**, the external modulation event was used as gating signal of the outputs, **CC8yTC**.EMT = 1_B. The external signal was configured to be active HIGH, **CC8yINS**.EVxLM = 0_B, which means that when the external signal is HIGH the outputs are set to the PASSIVE state.In this mode, the gating event can also be synchronized with the PWM signal by setting the **CC8yTC**.EMS = 1_B.

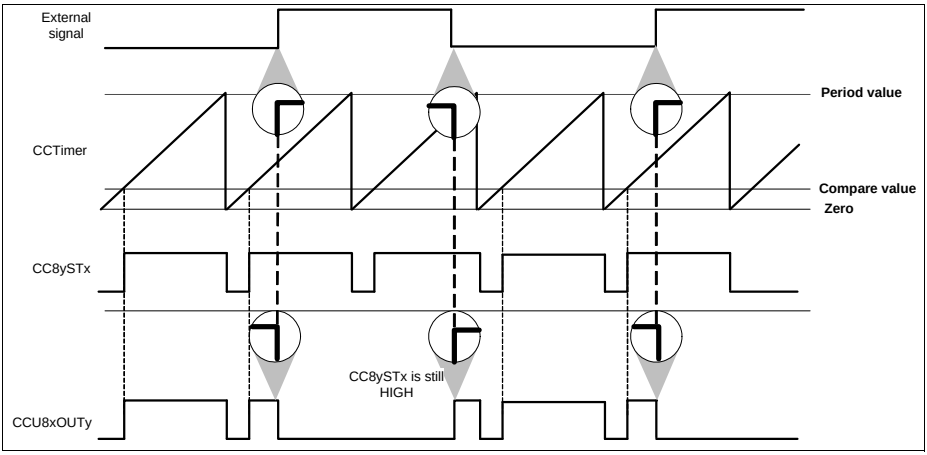


Figure 23-45 External modulation gating the output - CC8yTC.EMT = 1_B

23.2.8.8 Trap Function

The TRAP functionality allows the PWM outputs to react on the state of an input pin. This functionality can be used to switch off the power devices if the TRAP input becomes active.

To select the trap functionality, one should map one of the input signals to event number 2, by setting the required value in the **CC8yINS**.EV2IS register and indicating the active level of the signal on the **CC8yINS**.EV2LM register. This event should be then mapped to the trap functionality by setting the **CC8yCMC**.TS = 1_B.

Notice that the trap function is level active and therefore the active/passive configuration is set only by the **CC8yINS**.EV2LM.

There are two bitfields that can be monitored via software to crosscheck the TRAP function, **Figure 23-46**:

- The TRAP state bit, **CC8yINTS**.E2AS. This bitfield if the TRAP is currently active or not. This bitfield is therefore setting the specific Timer Slice output, into ACTIVE or PASSIVE state.
- The TRAP Flag, **CC8yINTS**.TRPF. This bitfield is used as a remainder in the case that the TRAP condition is cleared automatically via hardware. This field needs to be cleared by the software.

The E2AS can be configured to affect all of the CCU8 slice outputs, or a specific sub set of outputs via the **CC8yTC**.TRAPEy bit fields.

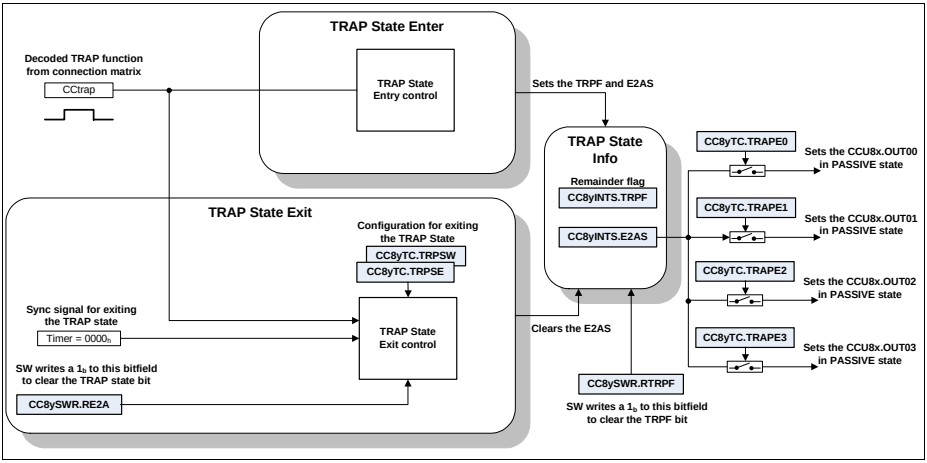


Figure 23-46 Trap control diagram

When a TRAP condition is detected at the selected input pin, both the Trap Flag and the Trap State bit are set to 1_B. The Trap State is entered immediately, by setting the CCU8xOUTy into the programmed PASSIVE state, **Figure 23-47**.

Exiting the Trap State can be done in two ways (**CC8yTC**.TRPSW register):

- automatically via HW, when the TRAP signal becomes inactive - **CC8yTC**.TRPSW = 0_B

- by SW only, by clearing the **CC8yINTS**.E2AS.The clearing is only possible if the input TRAP signal is in inactive state - **CC8yTC**.TRPSW = 1_B

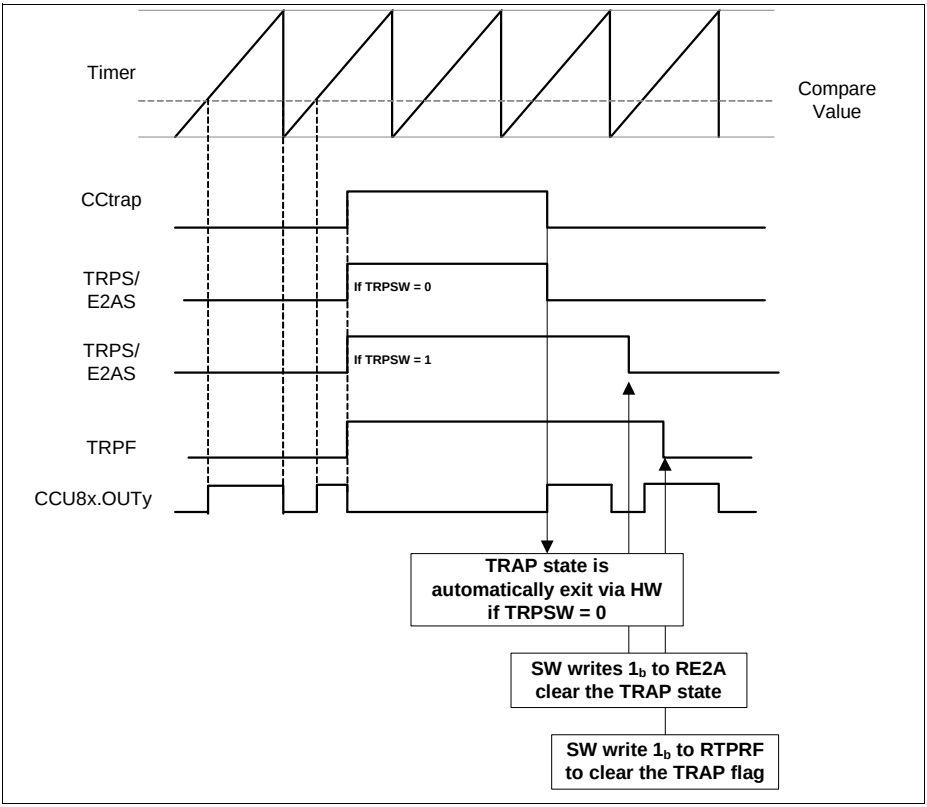


Figure 23-47 Trap timing diagram, CC8yTCST.CDIR = 0 CC8yPSL.PSL = 0

It is also possible to synchronize the exiting of the TRAP state with the PWM signal, **Figure 23-48**. This function is enabled when the bitfield **CC8yTC**.TRPSE = 1_B.

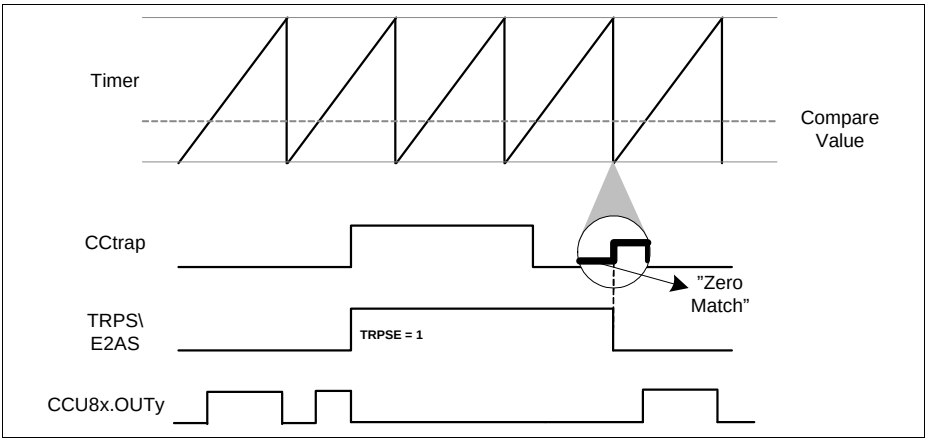


Figure 23-48 Trap synchronization with the PWM signal

23.2.8.9 Status Bit Override

For complex timed output control, each slice has a functionality that enables the override of the status bit of compare channel 1 (CC8yST1) with a value passed trough an external signal.

The override of the status bit, can then lead to a change on the output pins CCU8x.OUTy0 and CCU8x.OUTy1 (from inactive to active or vice versa).

To enable this functionality, two signals are needed:

- One signal that acts as a trigger to override the status bit (edge active)
- One signal that contains the value to be set in the status bit (level active)

To select the status bit override functionality, one should map the signal that acts as trigger to the event number 1, by setting the required value in the **CC8yINS**.EV1IS register and indicating the active edge of the signal on the **CC8yINS**.EV1EM register.

The signal that carries the value to be set on the status bit, needs to be mapped to the event number 2, by setting the required value in the **CC8yINS**.EV2IS register. The **CC8yINS**.EV2LM register should be set to 0_B if no inversion on the signal is needed and to 1_B otherwise.

The events should be then mapped to the status bit functionality by setting the **CC8yCMC**.OFS = 1_B.

Figure 23-49 shows the functionality of the status bit override, when the external signal(1) was selected as trigger source (rising edge active) and the external signal(2) was selected as override value.

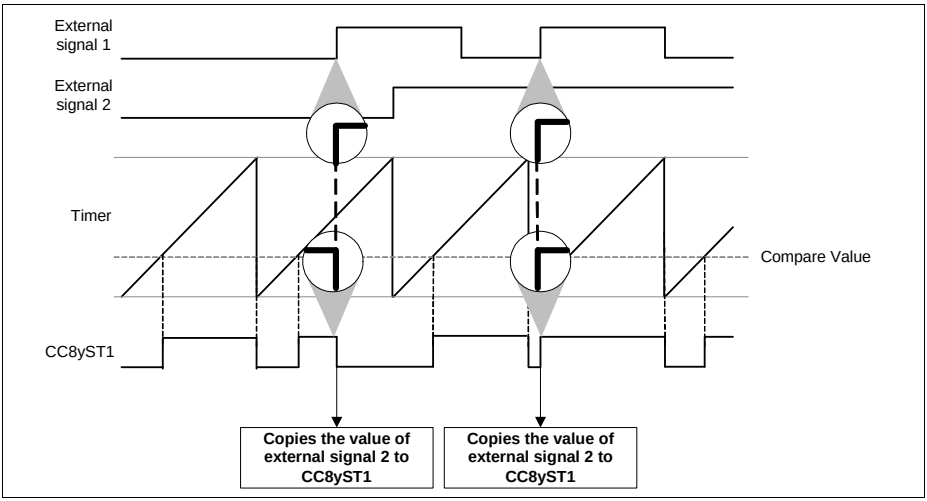


Figure 23-49 Status bit override

23.2.9 Multi-Channel Support

The multi channel control mode is selected individually in each slice by setting the **CC8yTC**.MCMEx = 1_B.

With this mode, the output state of the Timer Slices PWM signal(s) (the ones set in multichannel mode) can be controlled in parallel by a single pattern.

The pattern is controlled via the CCU8 inputs, CCU8x.MCIy[3:0]. Each group of these inputs is connected accordingly to the specific Timer Slice: for slice 0, CCU8xMCI1[3:0] for slice 1, CCU8xMCI2[3:0] for slice 2 and CCU8xMCI3[3:0] for slice 3.

This pattern can be controlled directly by one of the POSIF modules and be updated in parallel for all the Timer Slices.

Using the POSIF module in conjunction with the Multi Channel support of the CCU8, one can achieve a complete synchronicity between the output state update, CCU8x.OUTy and the update of a new pattern, **Figure 23-50**.

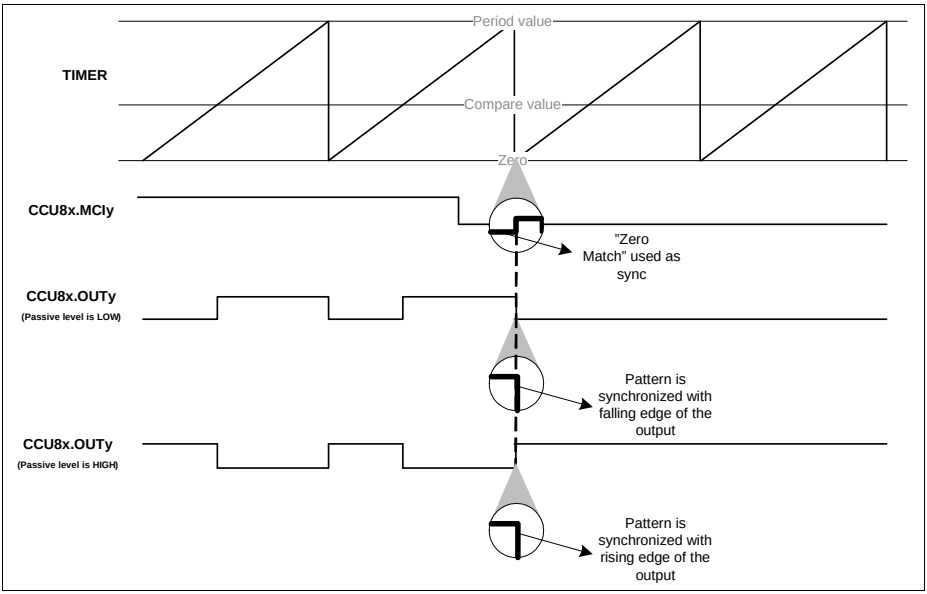


Figure 23-50 Multi channel pattern synchronization

These pattern inputs are going to be used in the output modulation control unit to put the specific PWM output into active or passive state: CCU8x.MCIy[0] has effect on the CC8yST1 path and therefore controls the CC8xOUT00 pin, CCU8x.MCy[1] is used in the same manner for the inverted CC8yST1 path, CCU8x.MCIy[2] and CCU8x.MCIy[3] are linked to the CC8yST2 and inverted CC8yST2 path respectively. **Figure 23-51** shows the simplified scheme for the multi channel control.

Figure 23-52, shows the usage of the multi channel mode in conjunction with two Timer Slices of the CCU8. The multi channel pattern is driven via the POSIF module, which enables a glitch free update of all the outputs of the CCU8.

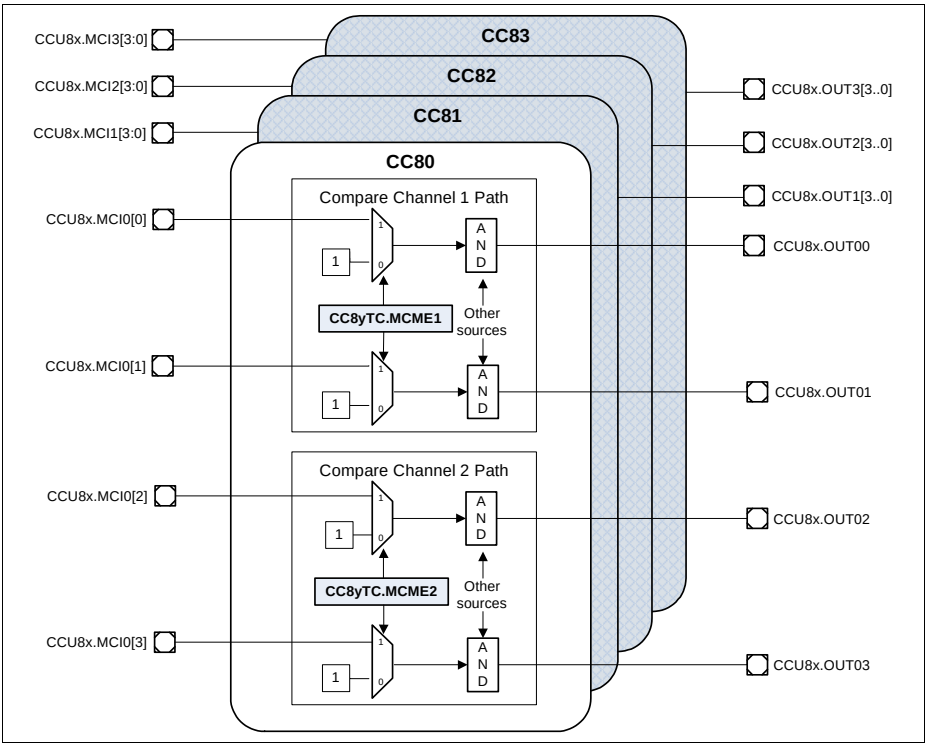


Figure 23-51 CCU8 Multi Channel overview

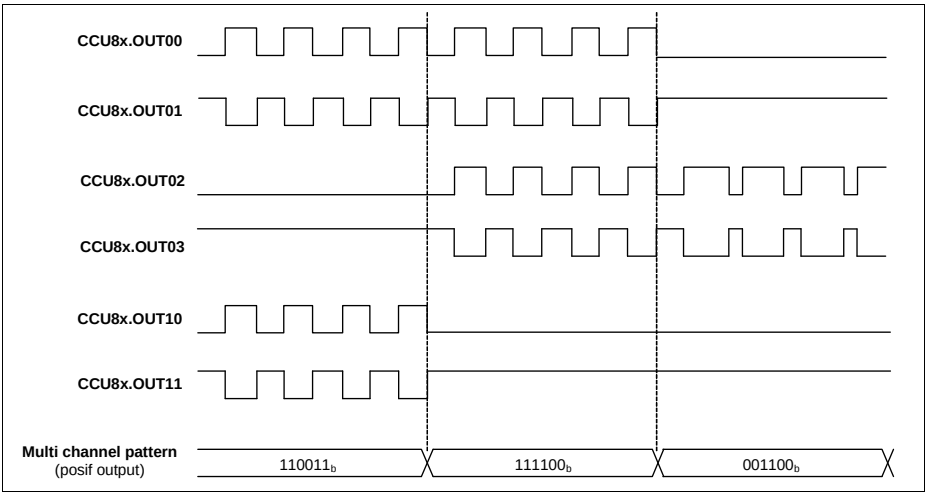


Figure 23-52 Multi Channel mode for multiple Timer Slices

The synchronization between the CCU8 and the POSIF is achieved, by adding a 3 cycle delay on the output path of each Timer Slice (between the status bit, CC8ySTx and the direct control of the output pin). This path is only selected when **CC8yTC**.MCMEx = 1_B.

On **Figure 23-53** the control of the CC8yST1 path is represented. The control of remaining paths follows the same mechanism (the multi channel is only enabled for the CC8yST2 path if **CC8yTC**.MCME2 = 1_B).

The multi pattern input synchronization can be seen on **Figure 23-54**. To achieve a synchronization between the update of the status bit, the sampling of a new multi channel pattern input is controlled by the period match or one match signal.

In a normal operation, where no external signal is used to control the counting direction, the signal used to enable the sampling of the pattern is always the period match when in edge aligned and the one match when in center aligned mode. When an external signal is used to control the counting direction, depending if the counter is counting up or counting down, the period match or the one match signal is used, respectively.

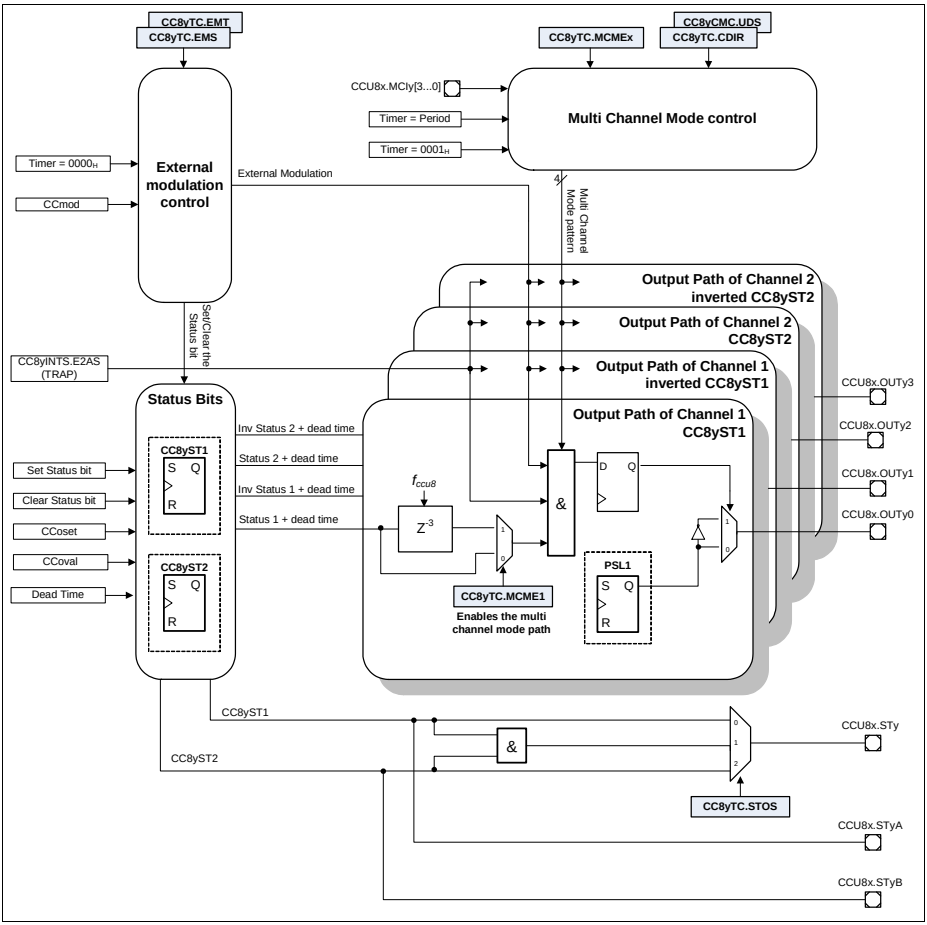


Figure 23-53 Output Control Diagram

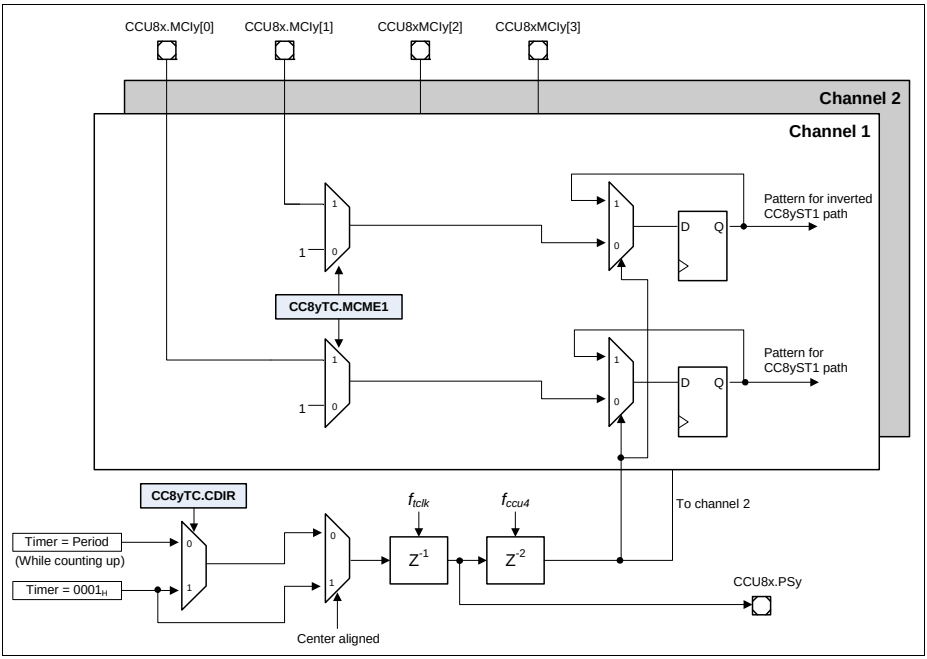


Figure 23-54 Multi Channel Pattern Synchronization Control

23.2.10 Timer Concatenation

The CCU8 offers a very easy mechanism to perform a synchronous timer concatenation. This functionality can be used by setting the **CC8yTC**.TCE = 1_B. By doing this the user is doing a concatenation of the actual CCU8 slice with the previous one, see **Figure 23-55**.

Notice that is not possible to perform concatenation with non adjacent slices and that timer concatenation automatically sets the slice mode into Edge Aligned. It is not possible to perform timer concatenation in Center Aligned mode.

To enable a 64 bit timer, one should set the **CC8yTC**.TCE = 1_B in all the slices (with the exception of the CC80 due to the fact that it doesn't contain this control field).

To enable a 48 bit timer, one should set the **CC8yTC**.TCE = 1_B in two adjacent slices and to enable a 32 bit timer, the **CC8yTC**.TCE is set to 1_B in the slice containing the MSBs. Notice that the timer slice conatining the LSBs should always have the TCE bitfield set to 0_B.

Several combinations for timer concatenation can be made inside a CCU8 module:

- one 64 bit timer

- one 48 bit timer plus a 16 bit timer
- two 32 bit timers
- one 32 bit timer plus two 16 bit timers

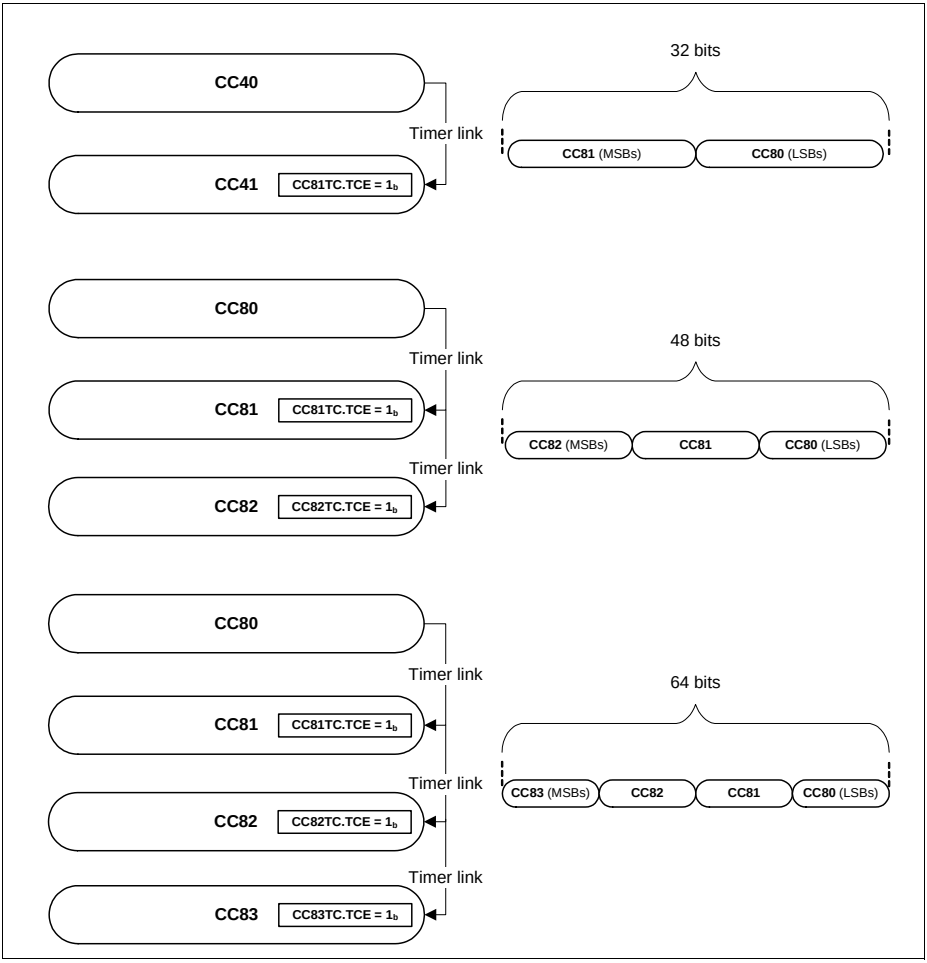


Figure 23-55 Timer concatenation example

Each Timer Slice is connected to the adjacent Timer Slices via a dedicated concatenation interface. This interface allows the concatenation of not only the Timer counting operation, but also a synchronous input trigger handling for capturing and loading operations, **Figure 23-56**.

Note: For all the cases, CC80 and CC83 are not considered adjacent slices

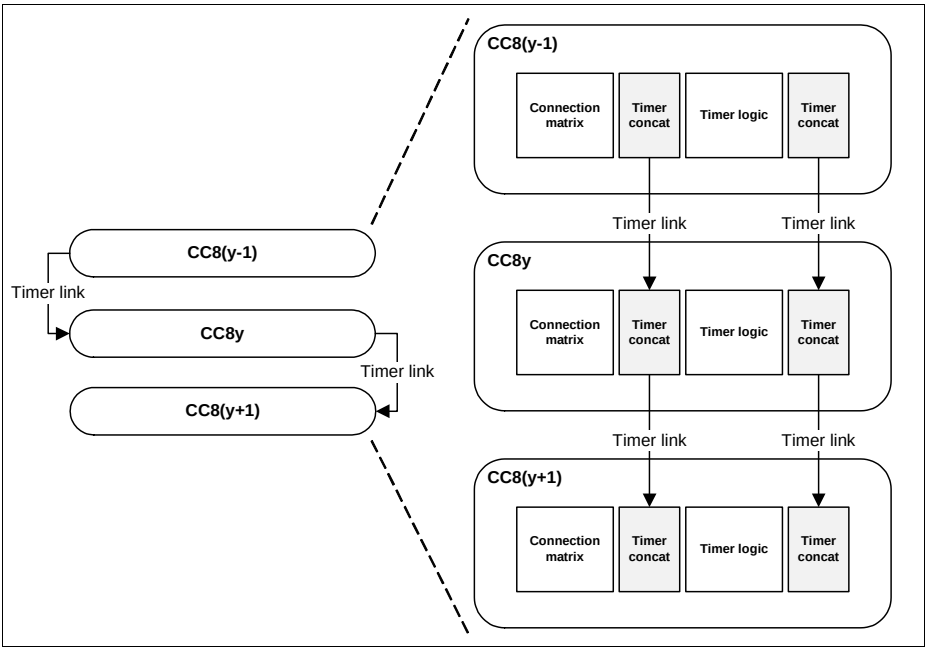


Figure 23-56 Timer concatenation link

Eight signals are present in the timer concatenation interface:

- Timer Period Match (CC8yPM)
- Timer Zero Match (CC8yZM)
- Timer Compare Match from channel 1 (CC8yCM1)
- Timer Compare Match from channel 2 (CC8yCM2)
- Timer counting direction function (CCupd)
- Timer load function (CCload)
- Timer capture function for CC8yC0V and CC8yC1V registers (CCcap0)
- Timer capture function for CC8yC2V and CC8yC3V registers (CCcap1)

The first five signals are used to perform the synchronous timing concatenation at the output of the Timer Logic, like it is seen in **Figure 23-56**. With this link, the timer length can be easily adjusted to 32, 48 or 64 bits (counting up or counting down)

The last three signals are used to perform a synchronous link between the capture and load functions, for the concatenated timer system. This means that the user can have a capture or load function programmed in the first Timer Slice, and propagate this capture or load trigger synchronously from the LSBs until the MSBs, **Figure 23-57**.

The capture or load function only needs to be configured in the first Timer Slice (the one holding the LSBs). From the moment that **CC8yTC**.TCE is set to 1_B, in the following Timer Slices, the link between these functions is done automatically by the hardware.

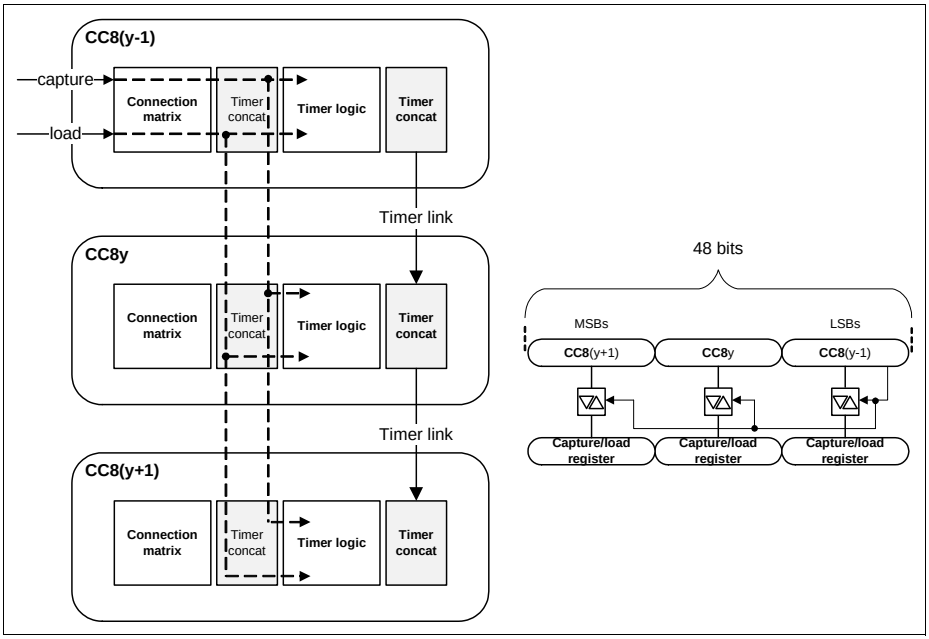


Figure 23-57 Capture/Load Timer Concatenation

the period match (CC8yPM) or zero match (CC8yZM) from the previous Timer Slice (with the immediately next lower index) are used in concatenated mode, as gating signal for the counter. This means that the counting operation of the MSBs only happens when a wrap around condition is detected (in the previous Timer Slice), avoiding additional DSP operations to extract the counting value.

With the same methodology, the compare match (CC8yCM1 and CC8yCM2), zero match and period match are gated with the specific signals from the previous Timer Slice. This means that the timing information is propagated throughout all the slices, enabling a completely synchronous match between LSB and MSB count, see **Figure 23-58**.

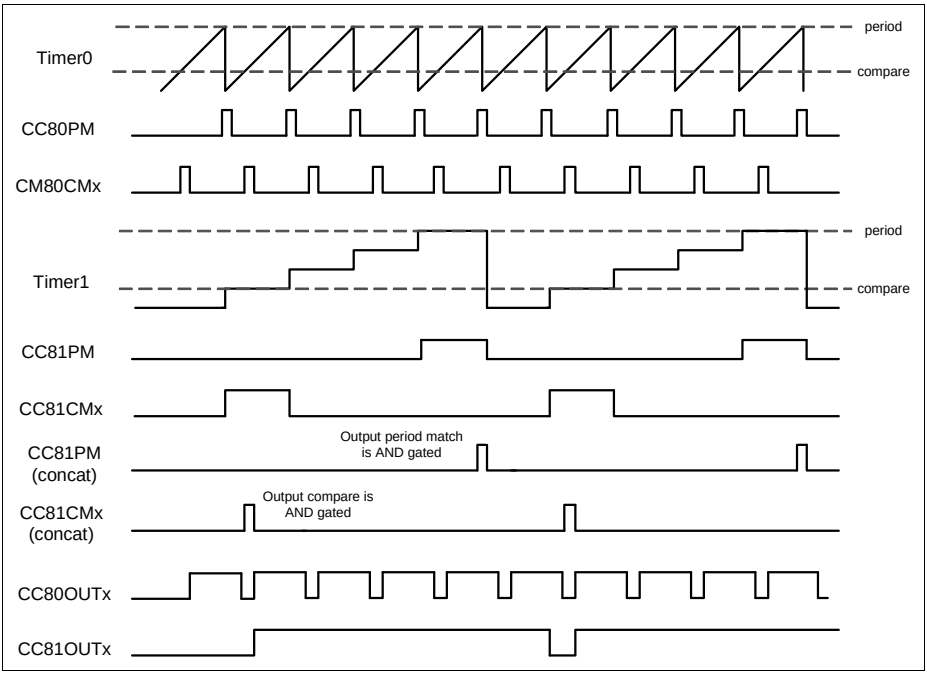


Figure 23-58 32 bit concatenation timing diagram

Note: the counting direction of the concatenated timer needs to be fixed. The timer can count up or count down, but the direction cannot be updated on the fly.

Figure 23-59 gives an overview of the timer concatenation logic. Notice that all the mechanism is controlled solely by the **CC8yTC**.TCE bitfield.

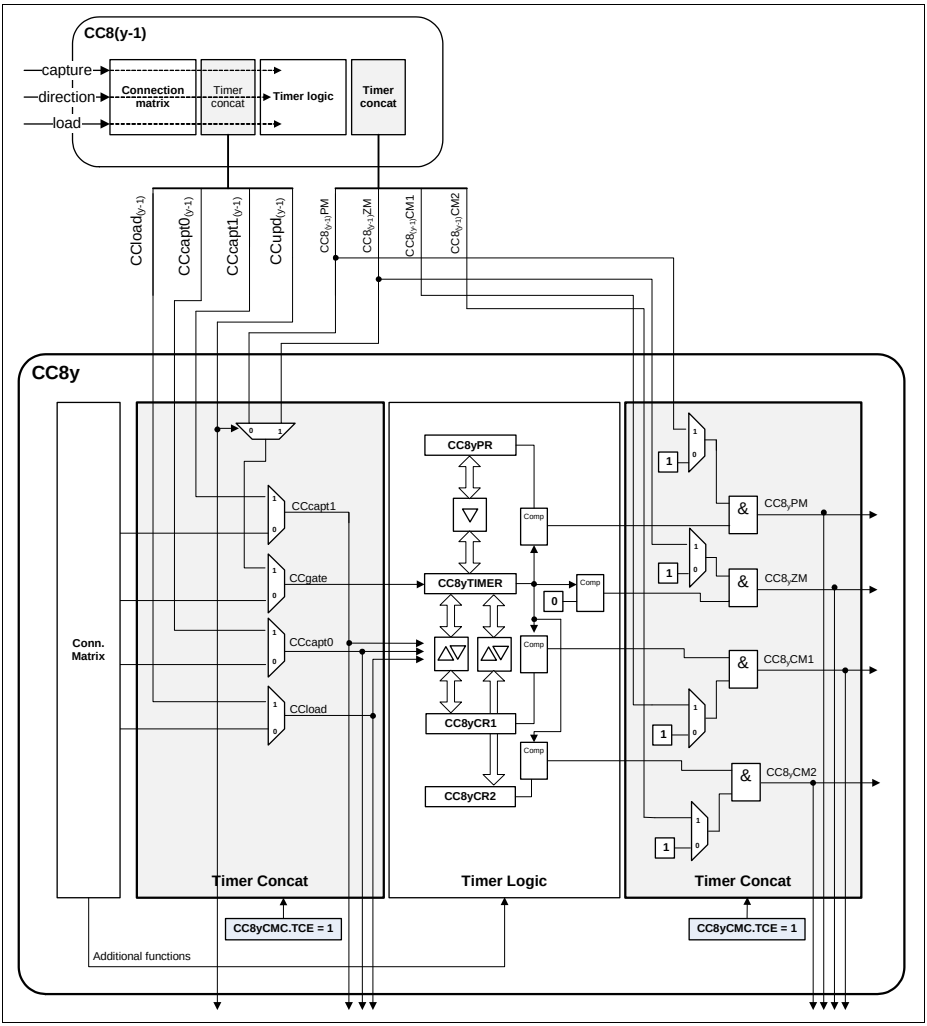


Figure 23-59 Timer concatenation control logic

23.2.11 Output Parity Checker

The parity checker function can be enabled by setting the **GIDLC**.PCH bit field to 1_B (parity checker is disabled while **GSTAT**.PCRB is 0_B).

This parity checker function, crosschecks the value at the output of the CCU8 module versus an input signal that should be connected to a driver XOR structure.

It is also possible to add a delay between the switching of the outputs and the evaluation of the input signal coming from the driver structure, and select which type of parity, even or odd (via the **GPCHK**.PCTS bit field). **Figure 23-60** shows the structure of the parity checker unit.

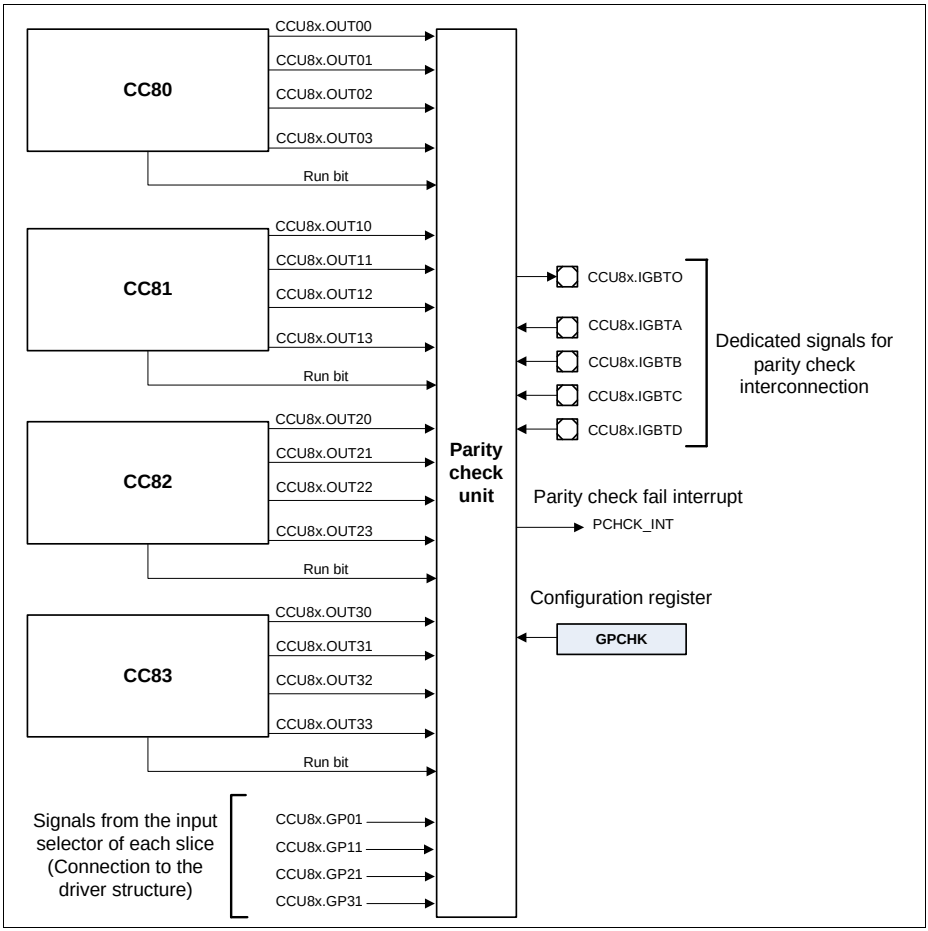


Figure 23-60 Parity checker structure

To use the parity check function, the user must select which signal is connected to the driver parity structure:

- The signal can be connected to any of the slices inputs
- The signal must be selected throughout the input selector mux of each slice. The signal must be mapped to the Event 1 of a slice.

Each of the CCU8 outputs can be individually selected to be part of the parity string and an interrupt is generated every time the input signal, coming from the driver structure, does not match the internally generated XOR result.

The interrupt is connected to the E1AS status bit of the slice where the driver parity output is connected:

- If **GPCHK**.PISEL = 00_B then the error status is in CC80INTS.E1AS
- If **GPCHK**.PISEL = 01_B then the error status is in CC81INTS.E1AS
- If **GPCHK**.PISEL = 10_B then the error status is in CC82INTS.E1AS
- If **GPCHK**-PISEL = 11_B then the error status is in CC83INTS.E1AS

The logic structure of the parity check is described on **Figure 23-61**. For a more detailed description of resource usage for the parity checker function, please address **Section 23.2.14.5**.

Configuration Example:

Driver parity output is connected to the input CCU8x.IN1B (where x = CCU8 unit). The input used to control the switching delay is the CCU8x.IGBTCCCU8. The driver is using 12 outputs coming from the first three slices with an even parity (PCTS field is in default).

The following registers should then be programmed:

CC8yINS.EV1IS = 0001_B; selects the input CCU8x.IN1B

GPCHK.PISEL = 01_B; selects the Event 1 coming from slice 1

GPCHK.PCDS = 10_B; selects the CCU8x.IGTBC input for delay control

GPCHK.PCSEL = $0FFF_H$; selects only the output signals of the first three slices for parity check

GIDLC.SPCH = 1_B; starts the parity function

When a mismatch between the driver output and the parity checker is detected, an interrupt is generated on Timer Slice 1. The interrupt status bit that stores the information is **CC8yINTS**.E1AS.

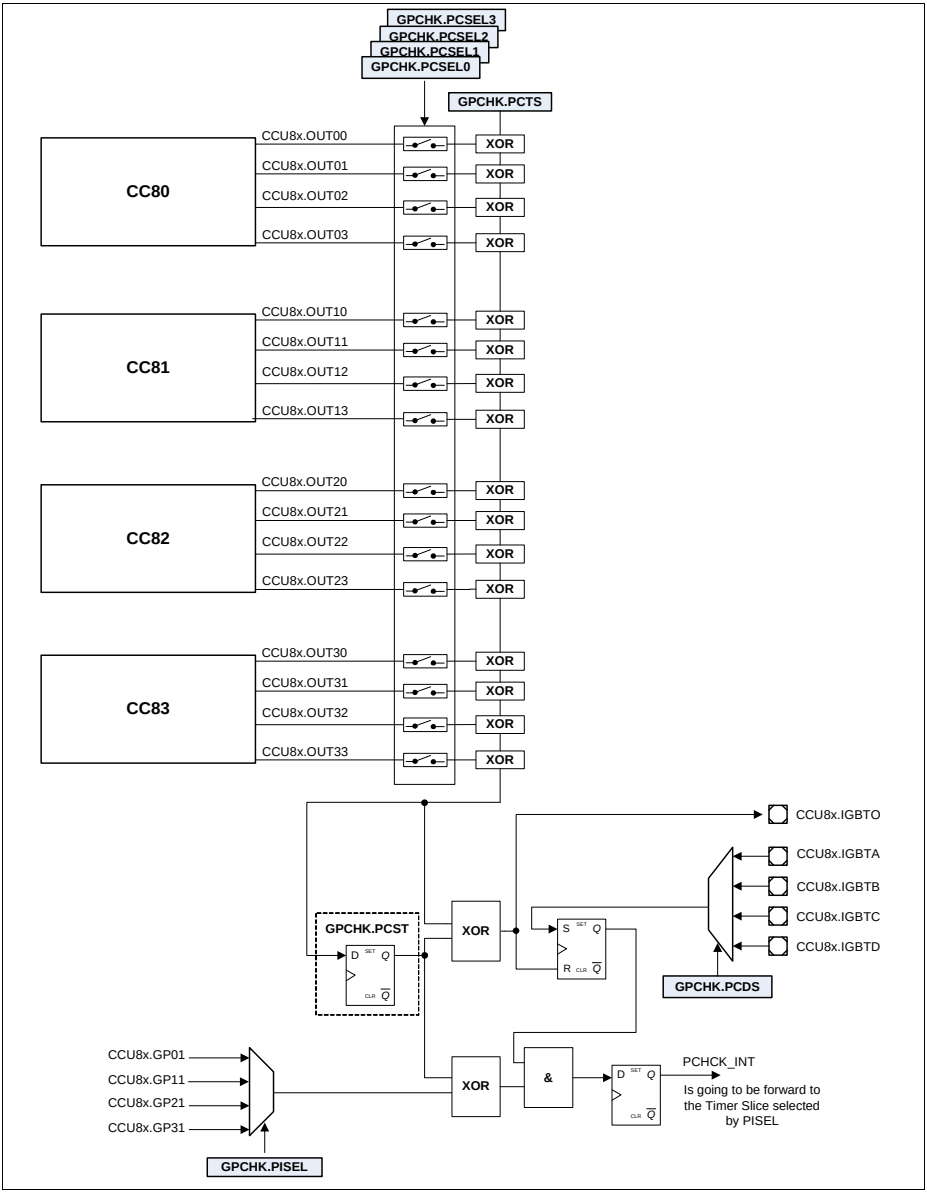


Figure 23-61 Parity checker logic

23.2.12 PWM Dithering

The CCU8 has an automatic PWM dithering insertion function. This functionality can be used with very slow control loops that cannot update the period/compare values in a fast manner, and by that fact the loop can lose precision on long runs. By introducing dither on the PWM signal, the average frequency/duty cycle is then compensated against that error.

Each slice contains a dither control unit, see **Figure 23-62**.

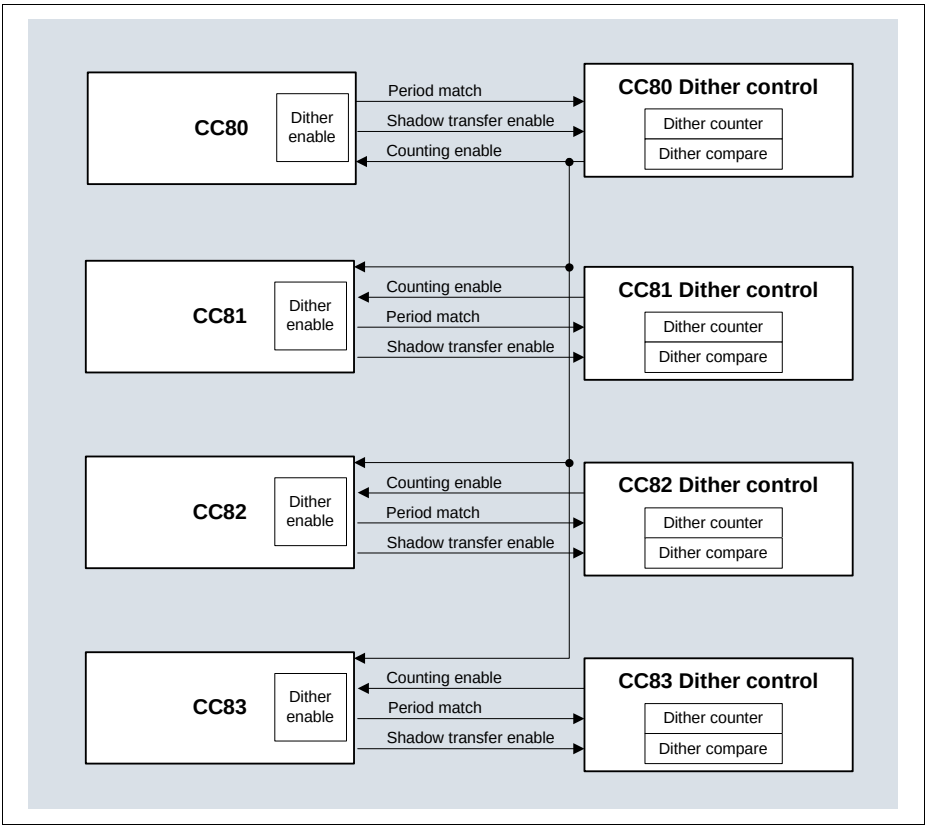


Figure 23-62 Dither structure overview

The dither control unit contains a 4 bit counter and a compare value. The four bit counter is incremented every time that a period match occurs. The counter works in a bit reverse mode so the distribution of increments stays uniform over 16 counter periods, see **Table 23-7**.

Table 23-7 Dither bit reverse counter

counter[3]	counter[2]	counter[1]	counter[0]
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	0
0	0	1	0
1	0	1	0
0	1	1	0
1	1	1	0
0	0	0	1
1	0	0	1
0	1	0	1
1	1	0	1
0	0	1	1
1	0	1	1
0	1	1	1
1	1	1	1

The counter is then compared against a programmed value, **CC8yDIT**.DCS. If the counter value is smaller than the programmed value, a gating signal is generated that can be used to extend the period, to delay the compare or both (controlled by the **CC8yTC**.DITHE field, see **Table 23-8**) for one clock cycle.

Table 23-8 Dither modes

DITHE[1]	DITH[0]	Mode
0	0	Dither is disabled
0	1	Period is increased by 1 cycle
1	0	Compare match is delayed by 1 cycle
1	1	Period is increased by 1 cycle and compare is delayed by 1 cycle

The dither compare value also has an associated shadow register that enables concurrent update with the period/compare registers of each CC8y. The control logic for the dithering unit is represented on **Figure 23-63**.

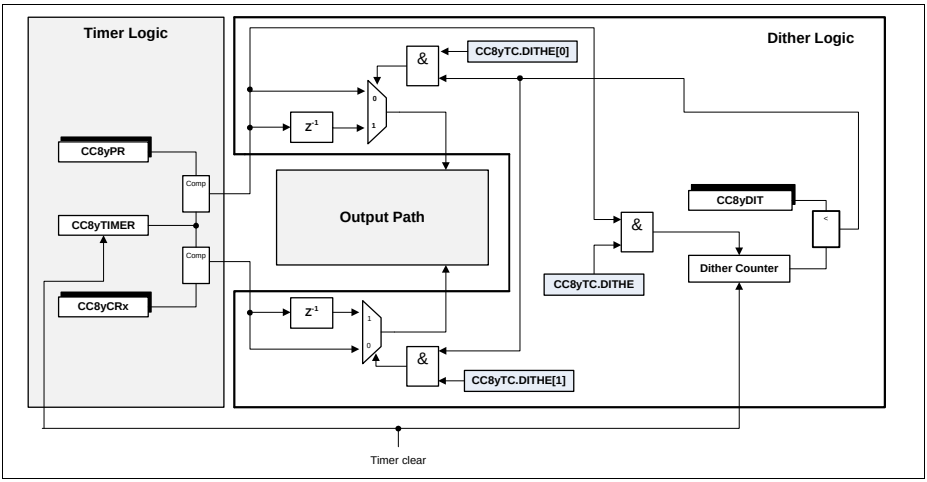


Figure 23-63 Dither control logic

Figure 23-64 to **Figure 23-69** show the effect of the different configurations of the dither, **CC8yTC**.DITHE, for both counting schemes, Edge and Center Aligned mode. In each figure, the bit reverse scheme is represented for the dither counter and the compare value was programmed with the value 8_H. In each figure, the variable *T*, represents the period of the counter, while the variable *d* indicates the duty cycle (status bit is set HIGH).

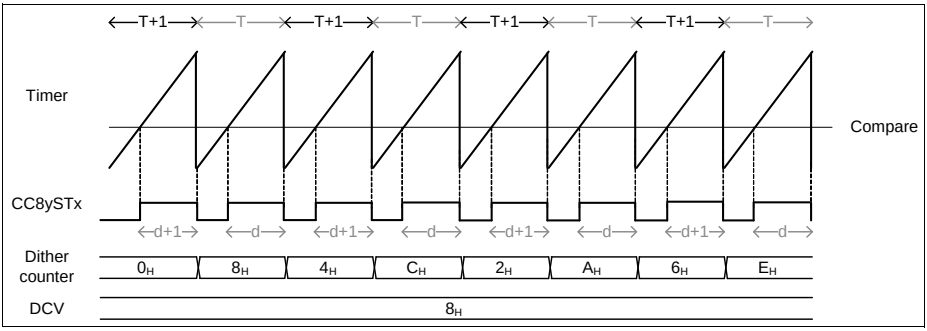


Figure 23-64 Dither timing diagram in edge aligned - CC8yTC.DITHE = 01_B

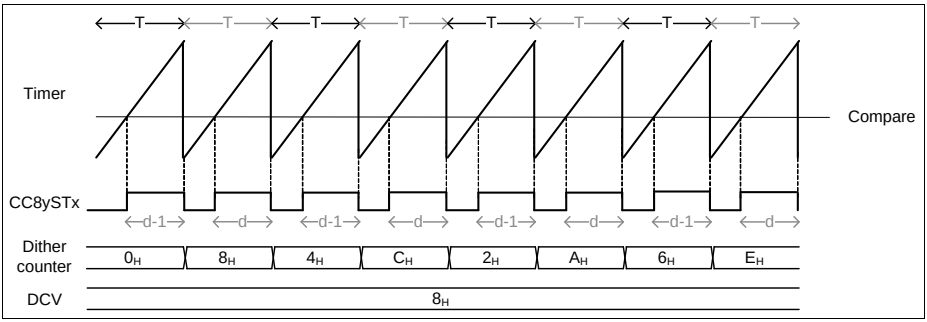


Figure 23-65 Dither timing diagram in edge aligned - CC8yTC.DITHE = 10_B

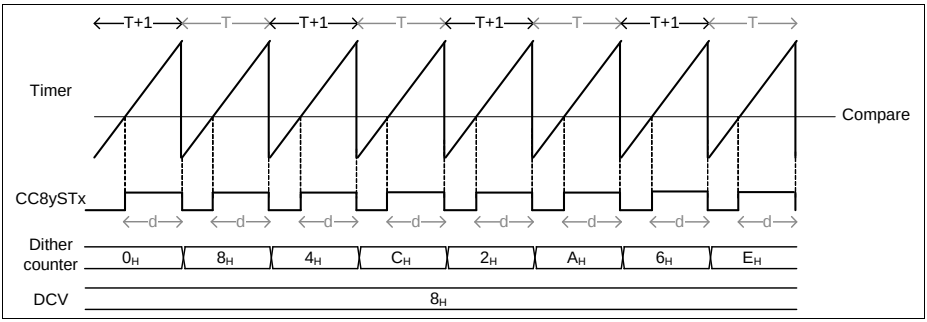


Figure 23-66 Dither timing diagram in edge aligned - CC8yTC.DITHE = 11_B

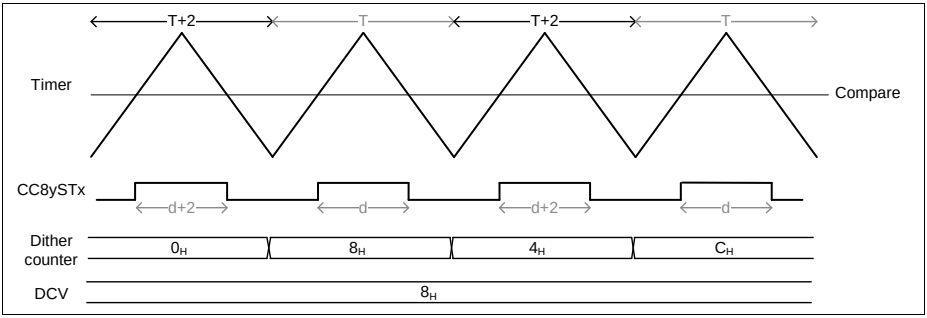


Figure 23-67 Dither timing diagram in center aligned - CC8yTC.DITHE = 01_B

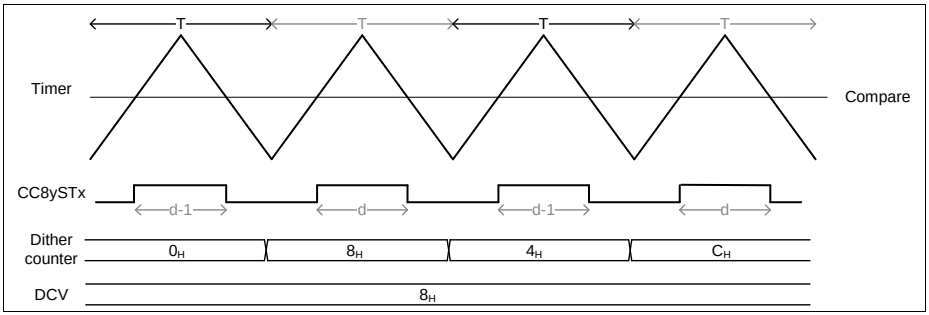


Figure 23-68 Dither timing diagram in center aligned - CC8yTC.DITHE = 10_B

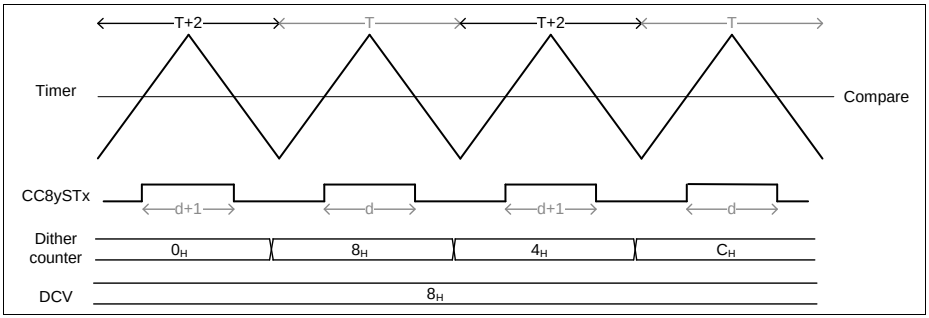


Figure 23-69 Dither timing diagram in edge aligned - CC8yTC.DITHE = 11_B

Note: When using the dither, is not possible to select a period value of FS when in edge aligned mode. In center aligned mode, the period value must be at least FS - 2.

23.2.13 Prescaler

The CCU8 contains a 4 bit prescaler that can be used in two operating modes for each individual slice:

- normal prescaler mode
- floating prescaler mode

The run bit of the prescaler can be set/cleared by SW by writing into the registers, **GIDLC**.SPRB and **GIDLS**.CPRB respectively or can also be cleared by the run bit of a specific slice. With the last mechanism, the run bit of the prescaler is cleared one clock cycle after the clear of the run bit of the selected sliced. To select which slice can perform this action, one should program the **GCTRL**.PRBC register.

23.2.13.1 Normal Prescaler Mode

In Normal prescaler mode the clock fed to the CC8y counter is a normal fixed division by N, accordingly to the value set in the **CC8yPSC**.PSIV register. The values for the possible division values are listed in **Table 23-9**. The **CC8yPSC**.PSIV value is only modified by a SW access. Notice that each slice has a dedicated prescaler value selector (**CC8yPSC**.PSIV), which means that the user can select different counter clocks for every and each Timer Slice (CC8y).

Table 23-9 Timer clock division options

CC8yPSC.PSIV	**Resulting clock**
0000_B	f_{ccu8}
0001_B	$f_{ccu8}/2$
0010_B	$f_{ccu8}/4$
0011_B	$f_{ccu8}/8$
0100_B	$f_{ccu8}/16$
0101_B	$f_{ccu8}/32$
0110_B	$f_{ccu8}/64$
0111_B	$f_{ccu8}/128$
1000_B	$f_{ccu8}/256$
1001_B	$f_{ccu8}/512$
1010_B	$f_{ccu8}/1024$
1011_B	$f_{ccu8}/2048$
1100_B	$f_{ccu8}/4096$
1101_B	$f_{ccu8}/8192$
1110_B	$f_{ccu8}/16384$
1111_B	$f_{ccu8}/32768$

23.2.13.2 Floating Prescaler Mode

The floating prescaler mode can be used individually in each slice by setting the register **CC8yTC**.FPE = 1_B. With this mode, the user can not only achieve a better precision on the counter clock for compare operations but also reduce the SW read access for the capture mode.

The floating prescaler mode contains additionally to the initial value register, **CC8yPSC**.PSIV, a compare register, **CC8yFPC**.PCMP with an associated shadow register.

Figure 23-70 shows the structure of the prescaler in floating mode when the specific slice is in compare mode (no external signal is used for capture). In this mode, the value of the clock division is incremented by 1_D every time that a timer overflow/underflow (overflow if in Edge Aligned Mode, underflow if in Center Aligned Mode) occurs.

In this mode, the Compare Match (both channels) from the timer is AND gated with the Compare Match of the prescaler and every time that this event occurs, the value of the clock division is updated with the **CC8yPSC**.PSIV value in the immediately next timer overflow/underflow event.

To use just one compare channel to control the floating prescaler, the other compare channel must be disabled. To due this, the compare value, **CC8yCR1** or **CC8yCR2** (depending on which channel is used) needs to be set with a value bigger than the period, **CC8yPR**. This means that in edge aligned more, the maximum value for the timer period is 65534_D, because the compare value of one channel needs to be set to 65535_D.

The shadow transfer of the floating prescaler compare value, **CC8yFPC**.PCMP, is done following the same rules described on **Section 23.2.5.2**.

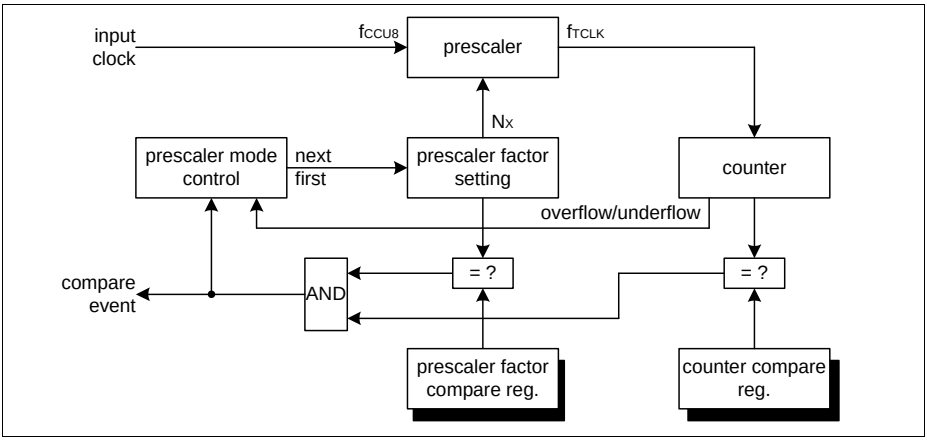


Figure 23-70 Floating prescaler in compare mode overview

When the specific CCU8 is operating in capture mode (when at least one external signal is decoded as capture functionality), the actual value of the clock division also needs to be stored every time that a capture event occurs. The floating prescaler can have up to 4 capture registers (the maximum number of capture registers is dictated by the number of capture registers used in the specific slice).

The clock division value continues to be increment by 1 every time that a timer overflow (in capture mode, the slice is always operating in Edge Aligned Mode) occurs and it is loaded with the PSIV value every time that a capture triggers is detected.

See the **Section 23.2.14** for a full description of the usage of the floating prescaler mode in conjunction with compare and capture modes.

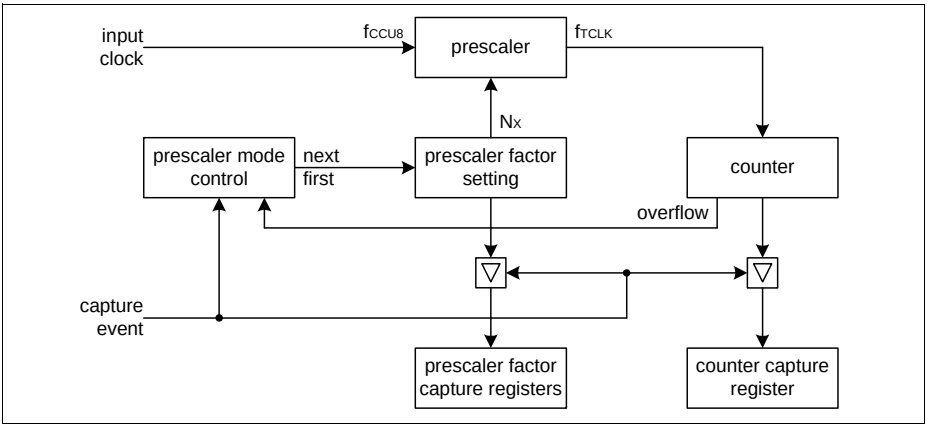


Figure 23-71 Floating Prescaler in capture mode overview

23.2.14 CCU8 Usage

23.2.14.1 PWM Signal Generation

The CCU8 offers a very flexible range in duty cycle configurations. This range is comprised between 0 to 100%.

To generate a PWM signal with a 100% duty cycle in Edge Aligned Mode, one should program the compare value, **CC8yCR1**.CR1/**CC8yCR2**.CR2, to 0000_H, **Figure 23-72**.

In the same manner a 100% duty cycle signal can be generated in Center Aligned Mode, **Figure 23-73**.

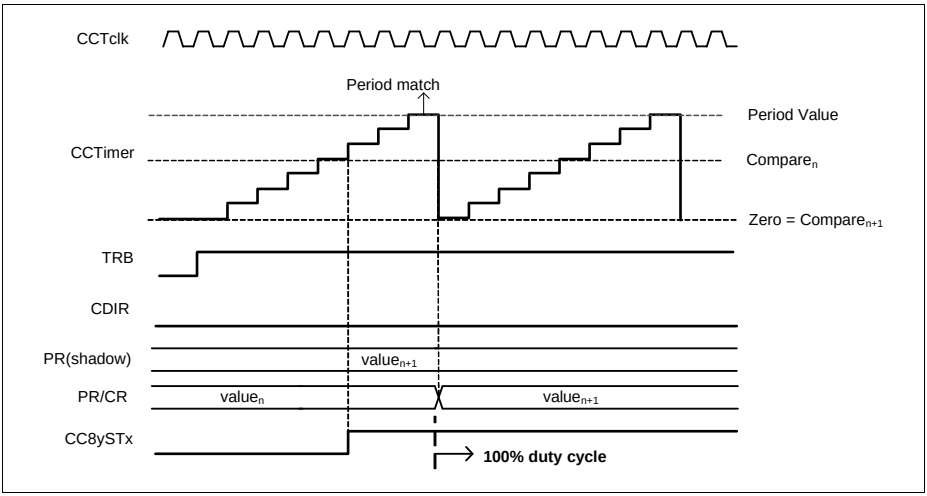


Figure 23-72 PWM with 100% duty cycle - Edge Aligned Mode

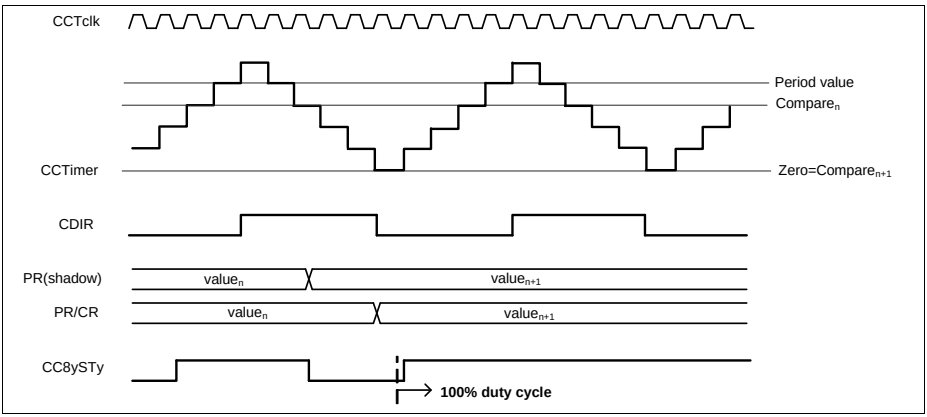


Figure 23-73 PWM with 100% duty cycle - Center Aligned Mode

To generate a PWM signal with 0% duty cycle in Edge Aligned Mode, the compare register should be set with the value programmed into the period value plus 1. In the case that the timer is being used with the full 16 bit capability (counting from 0 to 65535), setting a value bigger than the period value into the compare register is not possible and therefore the smallest duty cycle that can be achieved is 1/FS, see **Figure 23-74**.

In Center Aligned Mode, the counter is never running from 0_D to 65535_D, due to the fact that it has to overshoot for one clock cycle the value set in the period register. Therefore

the user never has a FS counter, which means that generating a 0% duty cycle signal is always possible by setting a value in the compare register bigger than the one programmed into the period register, see **Figure 23-75**.

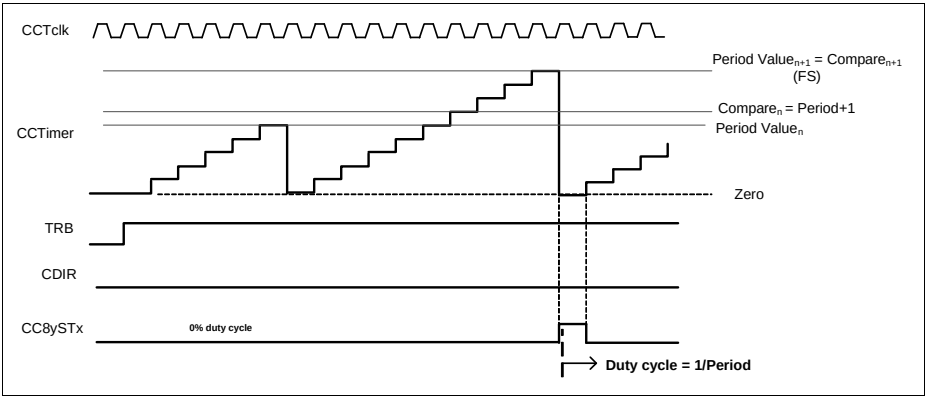


Figure 23-74 PWM with 0% duty cycle - Edge Aligned Mode

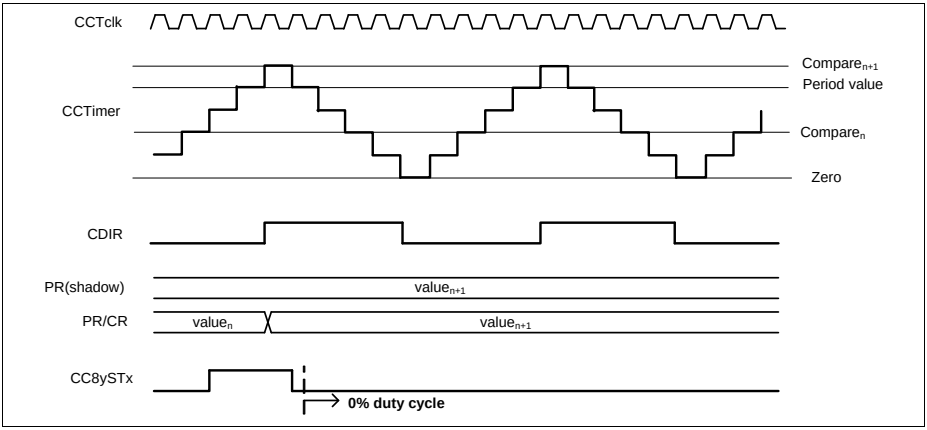


Figure 23-75 PWM with 0% duty cycle - Center Aligned Mode

23.2.14.2 Prescaler Usage

In Normal Prescaler Mode, the frequency of the f_{tclk} fed to the specific CC8y is chosen from the **Table 23-9**, by setting the **CC8yPSC**.PSIV with the required value.

In Floating Prescaler Mode, the frequency of the f_{tclk} can be modified over a selected timeframe, within the values specified in **Table 23-9**. This mechanism is specially useful if, in capture mode, the dynamic of the capture triggers is very slow or unknown.

In Capture Mode, the Floating Prescaler value is incremented by 1_D every time that a timer overflow happens and it is set with the initial programmed value when a capture event happens, see **Figure 23-76**.

When using the Floating Prescaler Mode in Capture Mode, the timer should be cleared each time that a capture event happens, **CC8yTC**.CAPC = 11_B. By operating the Capture mode in conjunction with the Floating Prescaler, even for capture signals that have a periodicity bigger that 16 bits, it is possible to use just a single CCU8 slice without monitoring the interrupt events triggered by the timer overflow. For this the user just needs to know what is the timer capture value and the actual prescaler configuration at the time that the capture event occurred. These values are contained in each CC8yCxV register.

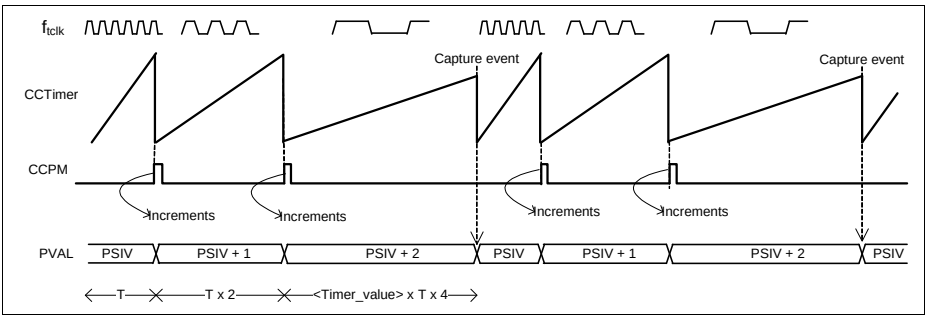


Figure 23-76 Floating Prescaler capture mode usage

When used in Compare Mode, the Floating Prescaler function may be used to achieve a fractional PWM frequency or to perform some frequency modulation.

The same incrementing by 1_D mechanism is done every time that a overflow/underflow of the Timer occurs (from any of the compare channels) and the actual Prescaler value doesn't match the one programmed into the **CC8yFPC**.PCMP register.

When a Compare Match from the Timer (from any of the compare channels) occurs and the actual Prescaler value is equal to the one programmed on the **CC8yFPC**.PCMP register, then the Prescaler value is set with the initial value, **CC8yPSC**.PSIV, in the immediately next occurrence of a timer overflow/underflow.

To use just one compare channel to control the floating prescaler, the other compare channel must be disabled. To due this, the compare value, **CC8yCR1** or **CC8yCR2** (depending on which channel is used) needs to be set with a value bigger than the period, **CC8yPR**. This means that in edge aligned more, the maximum value for the timer period is 65534_D, because the compare value of one channel needs to be set to 65535_D (so the compare match of the associated channel is disabled).

In **Figure 23-77**, the Compare value of the Floating Prescaler was set to PSIV + 2. Every time that a timer overflow occurs, the value of the Prescaler is incremented by 1, which

means that if we give f_{tclk} as the reference frequency for the **CC8yPSC**.PSIV value, we have $f_{tclk}/2$ for **CC8yPSC**.PSIV + 1 and $f_{tclk}/4$ for **CC8yPSC**.PSIV + 2. With the period overtime of the counter becomes:

Period = $(1/f_{tclk}+2/f_{tclk}+4/f_{tclk})/3$

The same mechanism is used in Center Aligned Mode, but to keep the rising arcade and falling arcade always symmetrical, instead of the overflow of the timer, the underflow is used, see **Figure 23-78**.

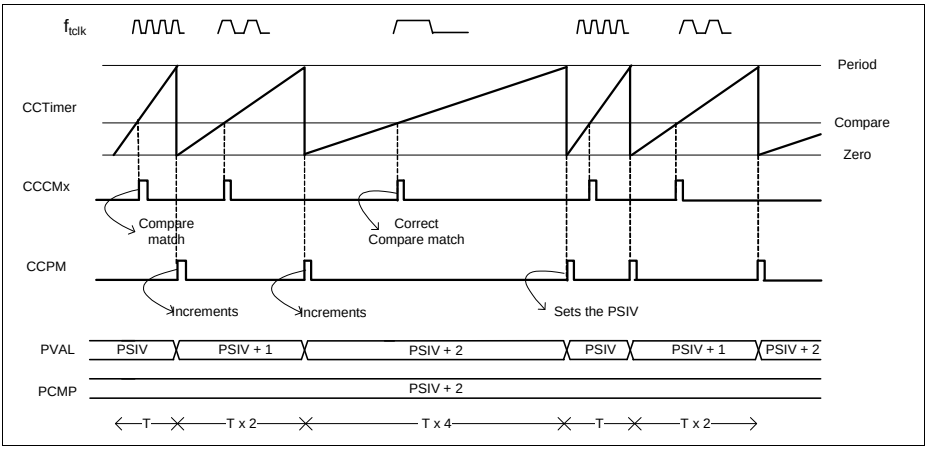


Figure 23-77 Floating Prescaler compare mode usage - Edge Aligned

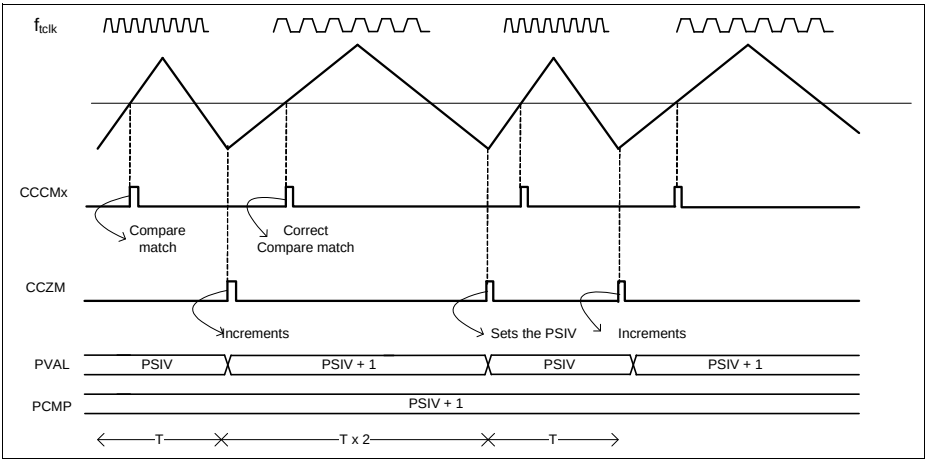


Figure 23-78 Floating Prescaler compare mode usage - Center Aligned

23.2.14.3 PWM Dither

The Dither function can be used to achieve a very fine precision on the periodicity of the output state in compare mode. The value set in the dither compare register, **CC8yDIT**.DCV is crosschecked against the actual value of the dither counter and every time that the dither counter is smaller than the comparison value one of the follows actions is taken:

- The period is extended for 1 clock cycle - **CC8yTC**.DITHE = 01_B; in edge aligned mode
- The period is extended for 2 clock cycles - **CC8yTC**.DITHE = 01_B; in center aligned mode
- The comparison match while counting up (**CC8yTCST**.CDIR = 0_B) is delayed (this means that the status bit is going to stay in the SET state 1 cycle less) for 1 clock cycle - **CC8yTC**.DITHE = 10_B;
- The period is extended for 1 clock cycle and the comparison match while counting up is delayed for 1 clock cycle - **CC8yTC**.DITHE = 11_B; in edge aligned mode
- The period is extended for 2 clock cycles and the comparison match while counting up is delayed for 1 clock cycle; center aligned mode

The bit reverse counter distributes the number programmed in the **CC8yDIT**.DCV throughout 16 timer periods.

Table 23-10, describes the bit reverse distribution versus the programmed value on the **CC8yDIT**.DCV field. The fields marked as '0' indicate that in that counter period, one of the above described actions, is going to be performed.

Table 23-10 Bit reverse distribution

	DCV															
Dither counter	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
8	1	1	1	1	1	1	1	1	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**
4	1	1	1	1	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
C	1	1	1	1	1	1	1	1	1	1	1	1	1	**0**	**0**	**0**
2	1	1	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
A	1	1	1	1	1	1	1	1	1	1	1	**0**	**0**	**0**	**0**	**0**
6	1	1	1	1	1	1	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
E	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**0**
1	1	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
9	1	1	1	1	1	1	1	1	1	1	**0**	**0**	**0**	**0**	**0**	**0**

Table 23-10 Bit reverse distribution (cont'd)

	DCV															
Dither counter	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
5	1	1	1	1	1	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
D	1	1	1	1	1	1	1	1	1	1	1	1	1	1	**0**	**0**
3	1	1	1	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
B	1	1	1	1	1	1	1	1	1	1	1	1	**0**	**0**	**0**	**0**
7	1	1	1	1	1	1	1	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
F	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

The bit reverse distribution versus the programmed **CC8yDIT**.DCV value results in the following values for the Period and duty cycle:

DITHE = 01_B

$Period$ = [(16 - DCV) x T + DCV x (T + 1)]/16; in Edge Aligned Mode (23.10)

$Duty\ cycle$ = [(16 - DCV) x d/T + DCV x ($d+1$)/(T + 1)]/16; in Edge Aligned Mode (23.11)

$Period$ = [(16 - DCV) x T + DCV x (T + 2)]/16; in Center Aligned Mode (23.12)

$Duty\ cycle$ = [(16 - DCV) x d/T + DCV x ($d+2$)/(T + 2)]/16; in Center Aligned Mode (23.13)

DITHE = 10_B

$Period$ = T ; in Edge Aligned Mode (23.14)

$Duty\ cycle$ = [(16 - DCV) x d/T + DCV x (d-1)/T]/16; in Edge Aligned Mode (23.15)

$Period$ = T ; in Center Aligned Mode (23.16)

$Duty\ cycle$ = [(16 - DCV) x d/T + DCV x (d-1)/T]/16; in Center Aligned Mode (23.17)

DITHE = 11_B

$Period$ = [(16 - DCV) x T + DCV x (T + 1)]/16; in Edge Aligned Mode (23.18)

$Duty\ cycle$ = [(16 - DCV) x d/T + DCV x d/(T + 1)]/16; in Edge Aligned Mode (23.19)

$Period$ = [(16 - DCV) x T + DCV x (T + 2)]/16; in Center Aligned Mode (23.20)

$Duty\ cycle$ = [(16 - DCV) x d/T + DCV x ($d+1$)/(T + 2)]/16; in Center Aligned Mode (23.21)

where:

T - Original period of the signal, see **Section 23.2.5.1**

d - Original duty cycle of the signal, see **Section 23.2.5.1**

23.2.14.4 Capture Mode Usage

Each slice has the possibility of using 2 or 4 capture registers. Using only 2 capture registers means that only 1 Event was linked to a captured trigger. To use the four capture registers, both capture triggers need to be mapped into an Event (it can be the same signal with different edges selected or two different signals) or the **CC8yTC**.SCE field needs to be set to 1_B, which enables the linking of the 4 capture registers.

The internal slice mechanism for capturing is the same for the capture trigger 1 or capture trigger 0.

Different Capture Events - SCE = 0_B

Capture trigger 1 (CCcapt1) is appointed to the capture register 2, **CC8yC2V** and capture register 3, **CC8yC3V**, while trigger 0 is appointed to capture register 1, **CC8yC1V** and 0, **CC8yC0V**.

In each CCcapt0 event, the timer value is stored into **CC8yC1V** and the value of the **CC8yC1V** is transferred into the **CC8yC0V**.

In each CCcapt1 event, the timer value is stored into capture register **CC8yC3V** and the value of the capture register **CC8yC3V** is transferred into **CC8yC2V**.

The previous capture/transfer mechanism only happens if the specific register is not full. A capture register becomes full when receives a new value and becomes empty after the SW has read back the value.

The full flag is cleared every time that the SW reads back the **CC8yC0V**, **CC8yC1V**, **CC8yC2V** or **CC8yC3V** register. The SW can be informed of a new capture trigger by enabling the interrupt source linked to the specific Event. This means that every time that a capture is made an interrupt pulse is generated.

In the case that the Floating Prescaler Mode is being used, the actual value of the clock division is also stored in the capture register (CC8yCxV).

Figure 23-79 shows an example of how the capture/transfer may be used in a Timer Slice that is using a external signal as count function (to measure the velocity of a rotating device), and an equidistant capture trigger that is used to dictate the timestamp

for the velocity calculation (two Timer waveforms are plotted, one that exemplifies the clearing of the timer in each capture event and another without the clearing function active).

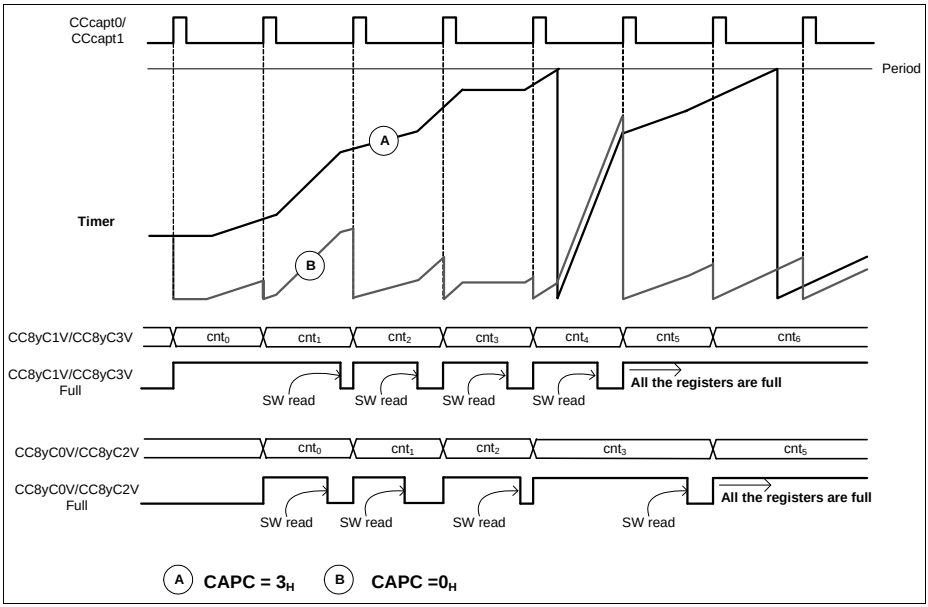


Figure 23-79 Capture mode usage - single channel

Same Capture Event - SCE = 1_B

If **CC8yTC**.SCE is set to 1_B, all the four capture registers are chained together, emulating a fifo with a depth of 4. In this case, only the capture trigger 1, CCcapt1, is used to perform a capture event.

As an example for this mode, one can consider the case where one Timer Slice is being used in capture mode with SCE = 1_B, with another external signal that controls the counting. This timer slice can be incremented at different speeds, depending on the frequency of the counting signal.

An additional Timer Slice is used to control the capture trigger, dictating the time stamp for the capturing.

A simple scheme for this can be seen in **Figure 23-80**. The CC80ST output of slice 0 was used as capture trigger in the CC81 slice (active on rising and falling edge). The CC80ST output is used as known timebase marker, while the slice timer used for capture is being controlled by external events, e.g. external count.

Due to the fact that we have 4 capture registers available, every time that the SW reads back the complete set of values, 3 speed profiles can be measured.

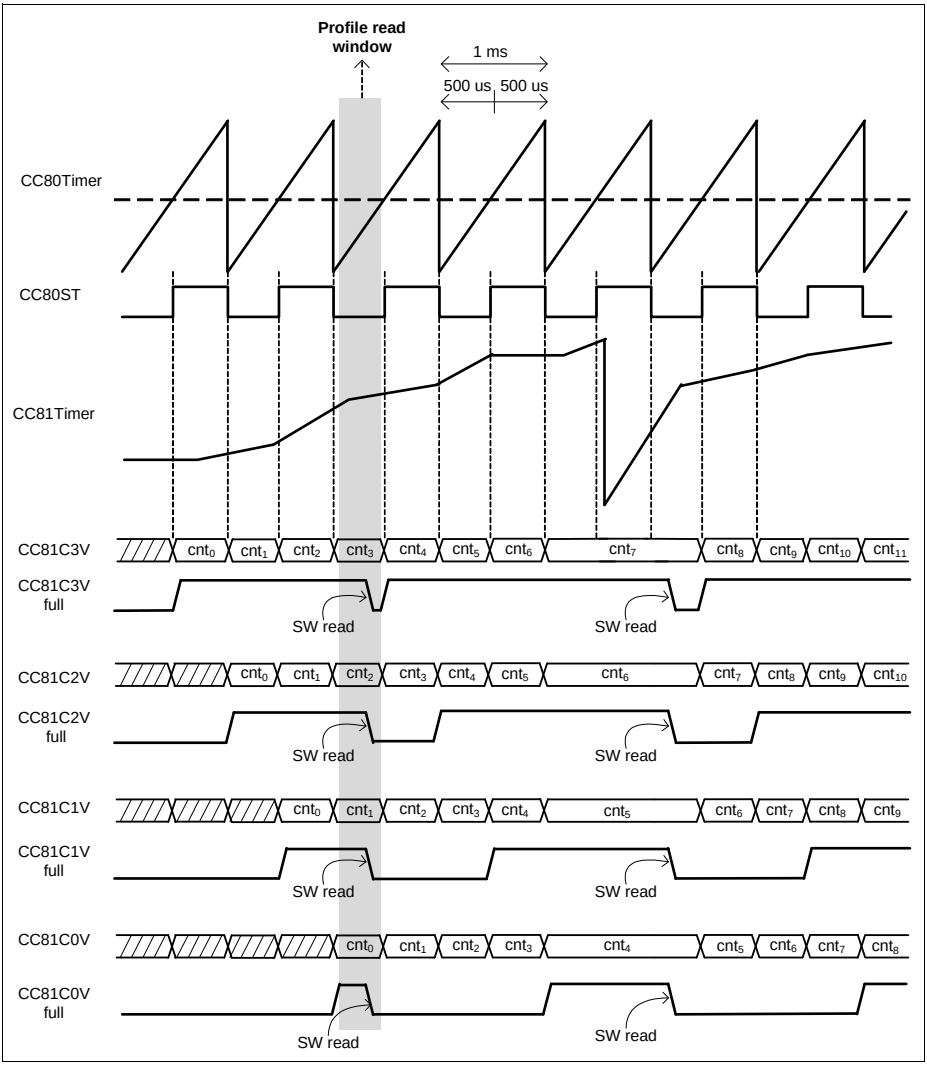


Figure 23-80 Three Capture profiles - CC8yTC.SCE = 1_B

To calculate the three different profiles in **Figure 23-80**, the 4 capture registers need to be read during the pointed read window. After that, the profile calculation is done:

Profile 1 = $CC81C1V_{info}$ - $CC81C0V_{info}$

Profile 2 = $CC81C2V_{info}$ - $CC81C1V_{info}$

Profile 3= $CC81C3V_{info}$ - $CC81C2V_{info}$

Note: This is an example and therefore several Timer Slice configurations and software loops can be implemented.

Extended Capture Read

When multiple Timer Slices need to be programmed into capture mode, it may not be suitable to distribute them over several CCU8 modules. This may be due to resource optimization or availability of Direct Memory Access (DMA) channels.

A simple way to overcome this issue, is to use the Extended Capture Read functionality of CCU8. This mode can be programmed independently for each and every Timer Slice via the **CC8yTC**.ECM bit field.

The advantage of this mode is that there is only one associated read address for all the capture registers (notice that the individual capture registers are still accessible), the **ECRD**. With this one can achieve DMA channel compression and a better Timer Slice resource optimization throughout the entire device.

Figure 23-81 exemplifies the usage of the Extended Capture Read function. In this example we have three different Timer Slices that are used to monitor three different applications (in capture mode). An additional Timer Slice (notice that it doesn't need to be in the same CCU8 module) is used to trigger the DMA read of the capture registers.

The read back trigger periodicity can also be updated on the fly to adjust to different system states or operation modes.

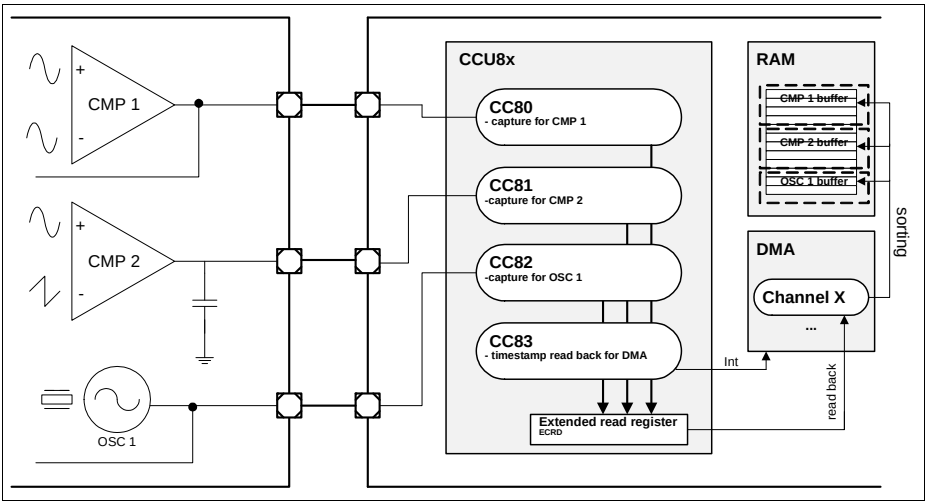


Figure 23-81 Extended read usage scheme example

Every time that the software reads back the **ECRD** register, the CCU8 returns the value of a specific capture register that contains new captured data. The read access of the capture registers follows a circular scheme that is maintained internally by the CCU8, **Figure 23-82**.

For the timer slices that are in capture mode but do not have **CC8yTC**.ECM = 1_B, their captured register values are also read back through the **ECRD**. However, the full flag of the capture registers is not cleared (it is only cleared via a read access to the specific CC4yCxV register).

Only the capture registers of the slices with **CC8yTC**.ECM = 1_B have their full flag cleared with a read access via **ECRD**.

On **Figure 23-83** an example time line is given, in which all the slices were programmed to use extended capture mode, **CC8yTC**.ECM = 1_B. In this example, one can see that the CCU8 doesn't keep memory of which was the first or last captured value between the Timer Slices. Like described on **Figure 23-82**, the read back pointer is incremented until a capture register that has the full flag set, is found.

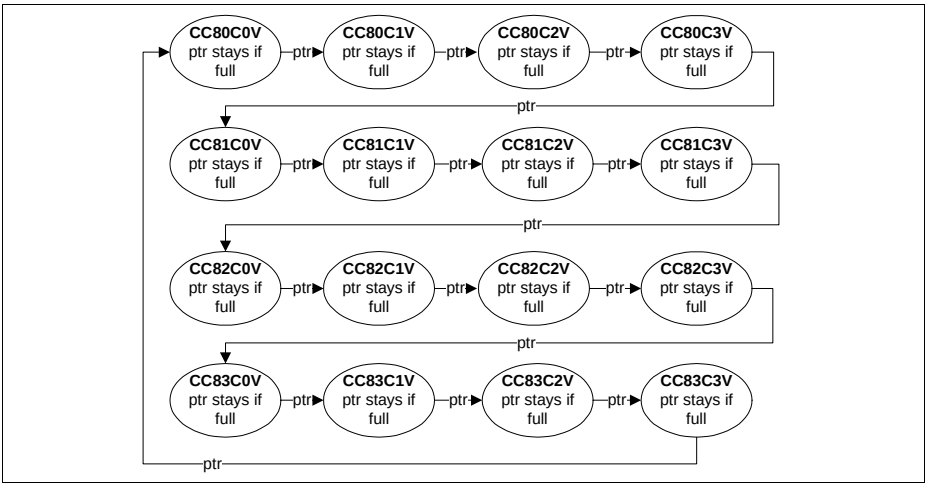


Figure 23-82 Extended Capture read back

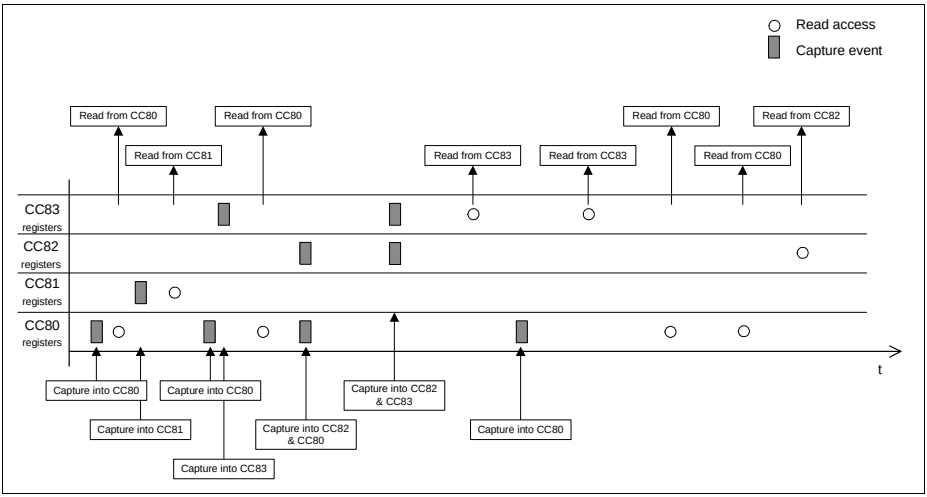


Figure 23-83 Extended Capture Access Example

23.2.14.5 Parity Checker Usage

The parity checker function available on the CCU8 uses of one CCU4 timer slice, to control the delay between the update of the outputs and the consequent update on the external switch/driver.

This CCU4 timer slice is configured to work in edge aligned mode, with single shot mode active and with a configured external flush & start function. This function is connected to the parity checker update output signal, CCU8x.IGBTO. The timer slice compare and period values need to be programmed accordingly to the delay that is foreseen between the outputs of the CCU8 and the consequent update on the driver/switch parity output. The connections between the CCU8, CCU4 and the external HW can be seen on **Figure 23-84**.

Figure 23-85 shows the timing waveforms for an usage example of the parity checker (case of even parity, **GPCHK**.PCTS field takes the default value). In this example only two outputs of CCU8 were considered to avoid extreme complexity on the diagram.

Every time an update of the selected outputs leads to a modification of the value **GPCHK**.PCST (parity checker status), or in other words, every time the result of the XOR chain changes, a trigger is generated on the CCU8x.IGBTO. This signal, as described previously, is used as a flush & start for a CCU4 slice.

When the CCU4 slice timer reaches the compare value, the specific status output, CCU4x.STy is asserted. After this elapsed delay, the value on the CCU8x.GPy1 (the parity value coming from the external HW) is crosschecked against the result of the internal XOR chain (**GPCHK**.PCST). If the values are different, a Service Request pulse can be generated, if it was previously enabled.

Notice that when an update of the CCU8 outputs leads to an equal result on the XOR chain, the CCU4 slice is not retriggered (the output parity from the external hardware remains with the same value).

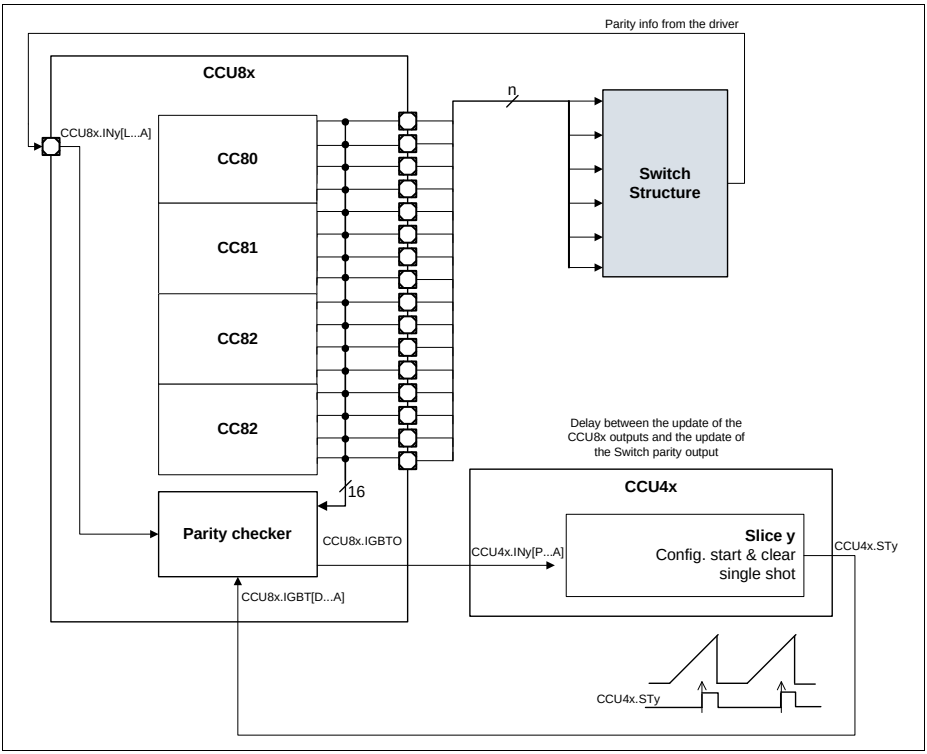


Figure 23-84 Parity Checker connections

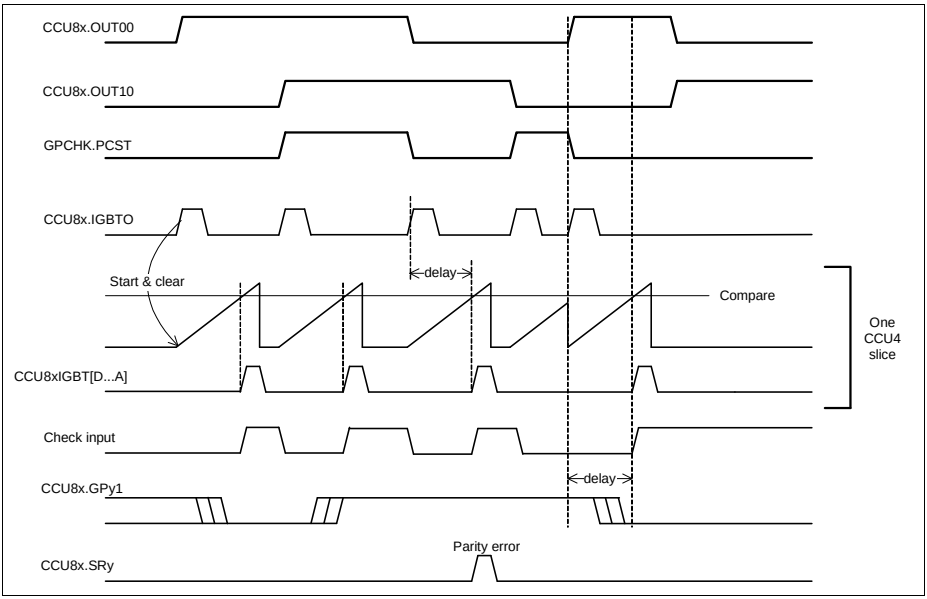


Figure 23-85 Parity Checker timing example

23.3 Service Request Generation

Each CCU8 slice has a interrupt structure as the one see in **Figure 23-86**. The register **CC8yINTS** is the status register for the interrupt sources. Each dedicated interrupt source can be set or cleared by SW, by writing into the specific bit in the **CC8ySWS** and **CC8ySWR** registers respectively.

Each interrupt source can be enabled/disabled via the **CC8yINTE** register. An enabled interrupt source will always generate a pulse on the service request line even if the specific status bit was not cleared. **Table 23-11** describes the interrupt sources of each CCU8 slice.

The interrupt sources, Period Match while counting up and one Match while counting down are ORed together. The same mechanism is applied to the Compare Match while counting up and Compare Match while counting down of both compare channels.

The interrupt sources for the external events are directly linked with the configuration set on the **CC8yINS**.EVxEM. If an event is programmed to be active on both edges, that means that service request pulse is going to be generated when any transition on the external signal is detected. If the event is linked with a level function, the **CC8yINS**.EVxEM still can be programmed to enable a service request pulse. The TRAP

event doesn't need any extra configuration for generating the service request pulse when the slice enters the TRAP state.

Table 23-11 Interrupt sources

Signal	Description
CCINEV0_E	Event 0 edge(s) information from event selector. Used when an external signal should trigger an interrupt.
CCINEV1_E	Event 1 edge(s) information from event selector. Used when an external signal should trigger an interrupt (It also can be the parity checker pattern fail information, if the parity checker was enabled).
CCINEV2_E	Event 2 edge(s) information from event selector. Used when an external signal should trigger an interrupt.
CCPM_U	Period Match while counting up.
CCCM1_U	Compare Match while counting up from compare channel 1.
CCCM1_D	Compare Match while counting down from compare channel 1.
CCCM2_U	Compare Match while counting up from compare channel 2.
CCCM2_D	Compare Match while counting down from compare channel 2.
CCOM_D	One Match while counting down.
Trap state set	Entering Trap State. Will set the E2AS.

Each of the interrupt events can then be forwarded to one, of the slice's four service request lines, **Figure 23-87**. The value set on the **CC8ySRS** controls which interrupt event is mapped into which service request line.

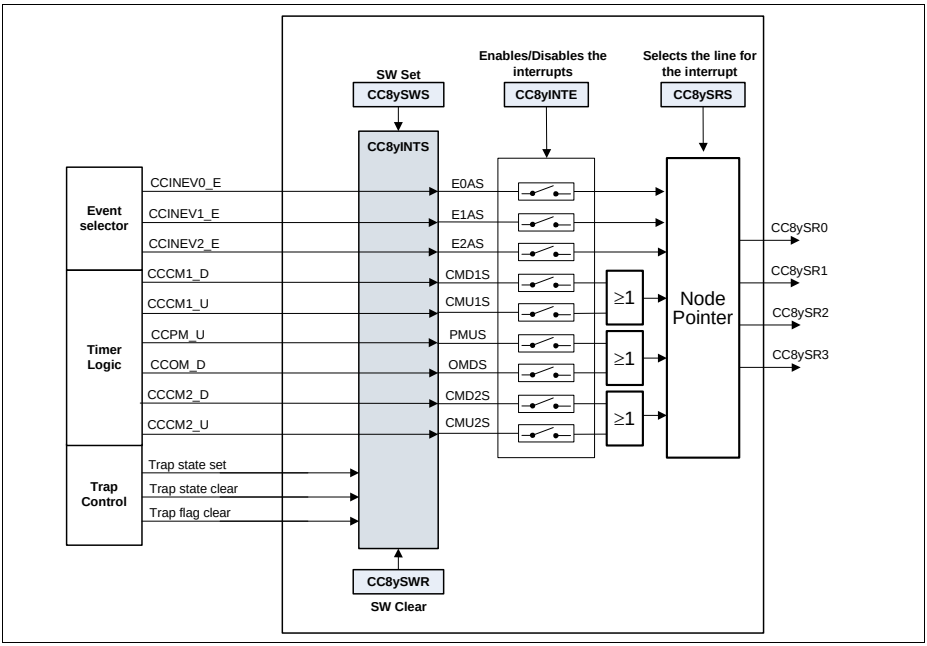


Figure 23-86 Slice interrupt node pointer overview

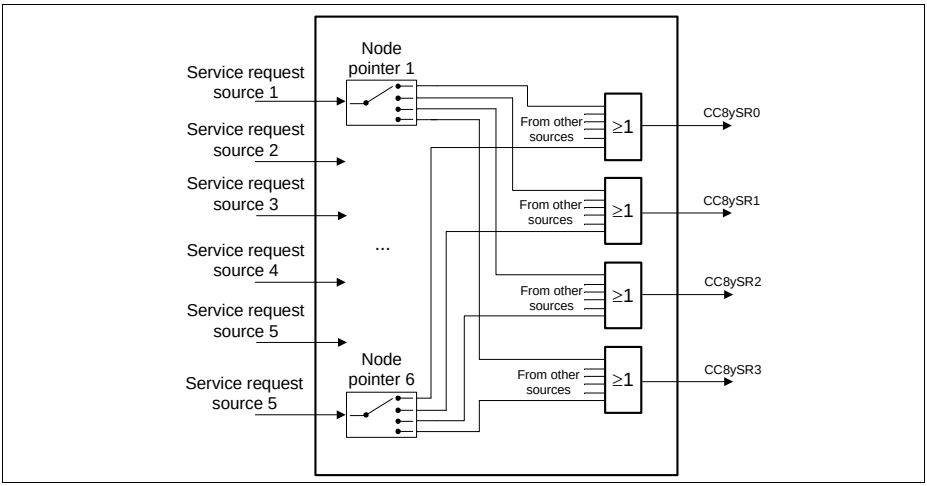


Figure 23-87 Slice interrupt selector overview

The four service request lines of each slice are OR together inside the kernel of the CCU8, see **Figure 23-88**. This means that there are only four service request lines per CCU8, that can have in each line interrupt requests coming from different slices.

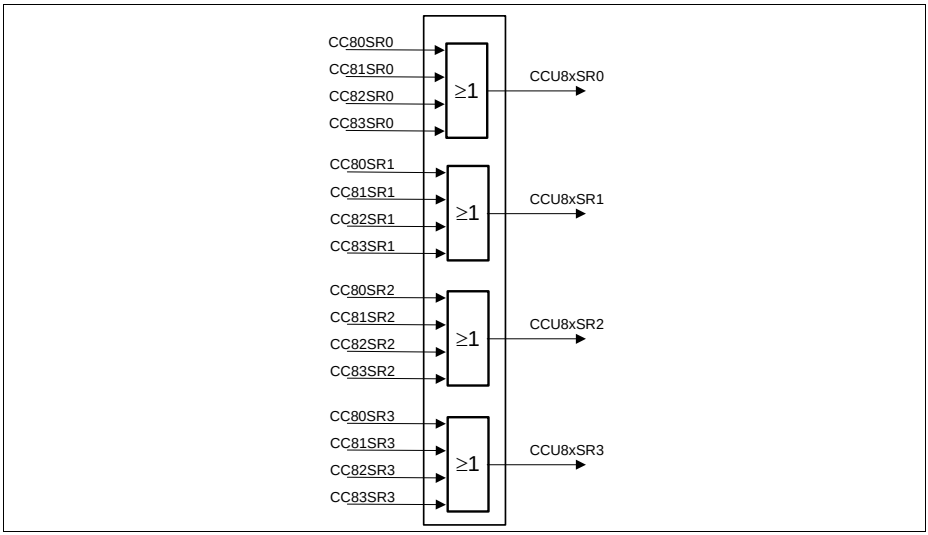


Figure 23-88 CCU8 service request overview

23.4 Debug Behavior

In suspend mode, the functional clocks for all slices as well the prescaler are stopped. The registers can still be accessed by the CPU (read only). This mode is useful for debugging purposes, e.g., where the current device status should be frozen in order to get a snapshot of the internal values. In suspend mode, all the slice counters are stopped. The suspend mode is non-intrusive concerning the register bits. This means register bits are not modified by hardware when entering or leaving the suspend mode.

Entry into suspend mode can be configured at the kernel level by means of the field **GCTRL**.SUSCFG.

The module is only functional after the suspend signal becomes inactive.

23.5 Power, Reset and Clock

The following sections describe the operating conditions, characteristics and timing requirements for the CCU8. All the timing information is related to the module clock, f_{ccu8}.

23.5.1 Clocks

Module Clock

The module clock of the CCU8 module is described in the SCU chapter as f_{CCU}.

The bus interface clock of the CCU8 module is described in the SCU chapter as f_{PERIPH}.

The module clock for the CCU8 is controlled via a specific control bit inside the SCU (System Control Unit), register CLKSET.

It is possible to disable the module clock for the CCU8 via the **GSTAT**, nevertheless, there may be a dependency of the f_{ccu8} through the different CCU8 instances. One should address the SCU Chapter for a complete description of the product clock scheme.

If module clock dependencies exist through different IP instances, then one can disable the module clock internally inside the specific CCU8, by disabling the prescaler (**GSTAT**.PRB = 0_B).

External Clock

It is possible to use an external clock as source for the prescaler, and consequently for all the timer Slices, CC8y. This external source can be connected to one of the CCU8x.CLK[C...A] inputs.

This external source is nevertheless synchronized against f_{ccu4}.

Table 23-12 External clock operating conditions

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Frequency	f_{eclk}	–	–	$f_{ccu8}/4$	MHz	
ON time	ton_{eclk}	$2T_{ccu8}$ 1)2)	–	–	ns	
OFF time	$toff_{eclk}$	$2T_{ccu8}$ 1)2)	–	–	ns	Only the rising edge is used

1) Only valid if the signal was not previously synchronized/generated with the fccu4 clock (or a synchronous clock)

2) 50% duty cycle is not obligatory

23.5.2 Module Reset

Each CCU8 has one reset source. This reset source is handled at system level and it can be generated independently via a system control register, PRSET0/PRSET1 (address SCU chapter for a full description).

After reset release, the complete IP is set to default configuration. The default configuration for each register field is addressed on **Section 23.7**.

23.5.3 Power

The CCU8 is inside the power core domain, therefore no special considerations about power up or power down sequences need to be taken. For a explanation about the different power domains, please address the SCU (System Control Unit) chapter.

An internal power down mode for the CCU8, can be achieved by disabling the clock inside the CCU8 itself. For this one should set the **GSTAT** register with the default reset value (via the idle mode set register, **GIDLS**).

23.6 Initialization and System Dependencies

23.6.1 Initialization Sequence

The initialization sequence for an application that is using the CCU8, should be the following:

1st Step: Apply reset to the CCU8, via the specific SCU bitfield on the PRSET0/PRSET1 register.

2nd Step: Release reset of the CCU8, via the specific SCU bitfield on the PRCLR0/PRCLR1 register.

3rd Step: Enable the CCU8 clock via the specific SCU register, CLKSET.

4th Step: Enable the prescaler block, by writing 1_B to the **GIDLC**.SPRB field.

5th Step: Configure the global CCU8 register **GCTRL**.

6th Step: Configure the all the registers related to the required Timer Slice(s) functions, including the interrupt/service request configuration.

7th Step: If needed, configure the startup value for a specific Compare Channel Status, of a Timer Slice, by writing 1_B to the specific **GCSS**.SyTS.

8th Step: Configure the parity checker function if used, by programming the **GPCHK** register

9th Step: Enable the specific timer slice(s), CC8y, by writing 1_B to the specific **GIDLC**.CSyI.

10th Step: For all the Timer Slices that should be started synchronously via SW, the specific system register localized in the SCU, CCUCON, that enables a synchronous timer start should be addressed.

23.6.2 System Dependencies

Each CCU8 may have different dependencies regarding module and bus clock frequencies. This dependencies should be addressed in the SCU and System Architecture Chapters.

Dependencies between several peripherals, regarding different clock operating frequencies may also exist. This should be addressed before configuring the connectivity between the CCU8 and some other peripheral.

The following topics must be taken into consideration for good CCU8 and system operation:

- CCU8 module clock must be at maximum two times faster than the module bus interface clock
- Module input triggers for the CCU8 must not exceed the module clock frequency (if the triggers are generated internally in the device)
- Module input triggers for the CCU8 must not exceed the frequency dictated in **Section 23.5.1**
- Frequency of the CCU8 outputs used as triggers/functions on other modules, must be crosschecked on the end point
- Applying and removing CCU8 from reset, can cause unwanted operations in other modules. This can occur if the modules are using CCU8 outputs as triggers/functions.

23.7 Registers

Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 23-13 Registers Address Space

Module	Base Address	End Address	Note
CCU80	40020000_H	$40023FFF_H$	
CCU81	40024000_H	$40027FFF_H$	

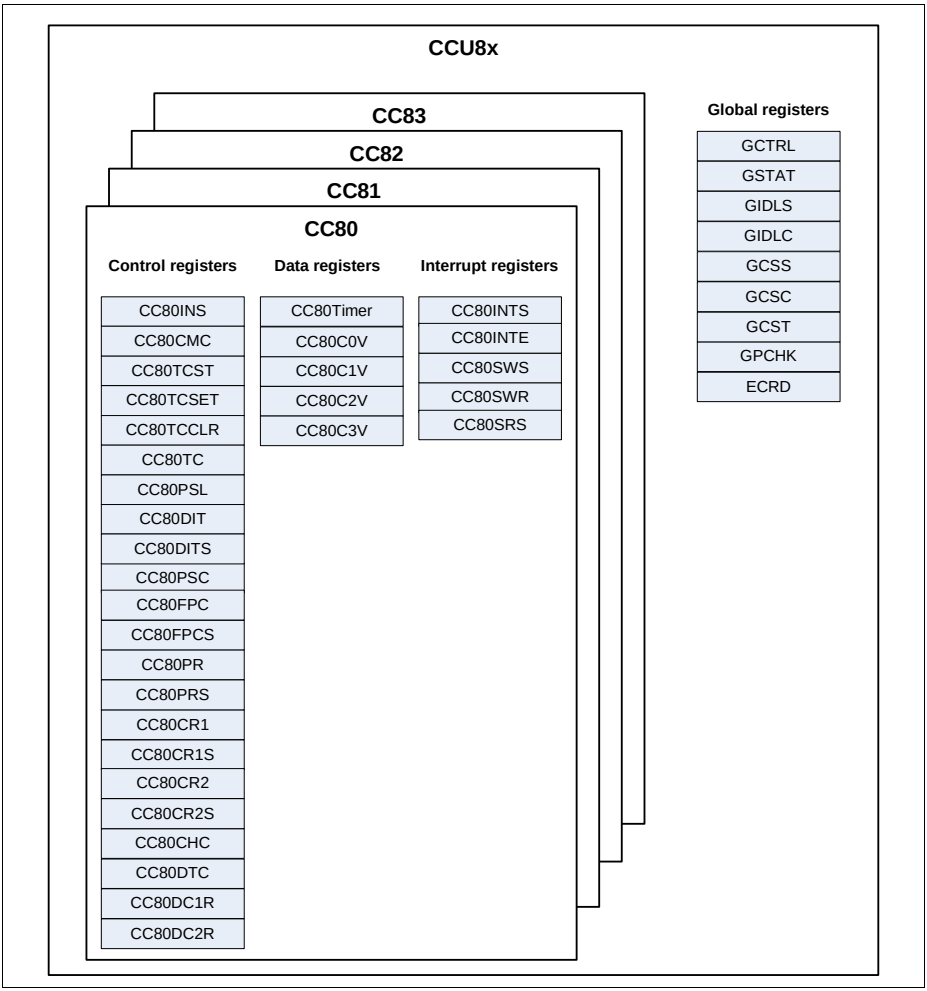


Figure 23-89 CCU8 registers overview

Table 23-14 Register Overview of CCU8

Short Name	Description	Offset Addr.[1]	Access Mode		Description See
			Read	Write	
CCU8 Global Registers					
GCTRL	Module General Control Register	0000_H	U, PV	U, PV	Page 23-103
GSTAT	General Slice Status Register	0004_H	U, PV	BE	Page 23-106
GIDLS	General Idle Enable Register	0008_H	U, PV	U, PV	Page 23-107
GIDLC	General Idle Disable Register	$000C_H$	U, PV	U, PV	Page 23-109
GCSS	General Channel Set Register	0010_H	U, PV	U, PV	Page 23-110
GCSC	General Channel Clear Register	0014_H	U, PV	U, PV	Page 23-113
GCST	General Channel Status Register	0018_H	U, PV	BE	Page 23-116
GPCHK	Parity Checker Configuration Register	$001C_H$	U, PV	U, PV	Page 23-119
ECRD	Extended Read Register	0050_H	U, PV	BE	Page 23-121
MIDR	Module Identification Register	0080_H	U, PV	BE	Page 23-123
CC80 Registers					
CC80INS	Input Selector Unit Configuration	0100_H	U, PV	U, PV	Page 23-123
CC80CMC	Connection Matrix Configuration	0104_H	U, PV	U, PV	Page 23-125
CC80TCST	Timer Run Status	0108_H	U, PV	BE	Page 23-129
CC80TCSET	Timer Run Set	$010C_H$	U, PV	U,PV	Page 23-130
CC80TCCLR	Timer Run Clear	0110_H	U, PV	U, PV	Page 23-131
CC80TC	General Timer Configuration	0114_H	U, PV	U, PV	Page 23-132
CC80PSL	Output Passive Level Configuration	0118_H	U, PV	U, PV	Page 23-138
CC80DIT	Dither Configuration	$011C_H$	U, PV	BE	Page 23-139
CC80DITS	Dither Shadow Register	0120_H	U, PV	U, PV	Page 23-140

Table 23-14 Register Overview of CCU8 (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
CC80PSC	Prescaler Configuration	0124_H	U, PV	U, PV	Page 23-140
CC80FPC	Prescaler Compare Value	0128_H	U, PV	U, PV	Page 23-141
CC80FPCS	Prescaler Shadow Compare Value	$012C_H$	U, PV	U, PV	Page 23-142
CC80PR	Timer Period Value	0130_H	U, PV	BE	Page 23-143
CC80PRS	Timer Period Shadow Value	0134_H	U, PV	U, PV	Page 23-144
CC80CR1	Timer Compare Value for Channel 1	0138_H	U, PV	BE	Page 23-144
CC80CR1S	Timer Compare Shadow Value for Channel 1	$013C_H$	U, PV	U, PV	Page 23-145
CC80CR2	Timer Compare Value for Channel 2	0140_H	U, PV	BE	Page 23-146
CC80CR2S	Timer Compare Shadow Value for Channel 2	0144_H	U, PV	U, PV	Page 23-147
CC80CHC	Channel Control	0148_H	U, PV	U, PV	Page 23-148
CC80DTC	Dead Time Control	$014C_H$	U, PV	U, PV	Page 23-150
CC80DC1R	Channel 1 Dead Time Counter Values	0150_H	U, PV	U, PV	Page 23-151
CC80DC2R	Channel 2 Dead Time Counter Values	0154_H	U, PV	U, PV	Page 23-152
CC80TIMER	Timer Current Value	0170_H	U, PV	U, PV	Page 23-152
CC80C0V	Capture Register 0 Value	0174_H	U, PV	BE	Page 23-153
CC80C1V	Capture Register 1 Value	0178_H	U, PV	BE	Page 23-154
CC80C2V	Capture Register 2 Value	$017C_H$	U, PV	BE	Page 23-155
CC80C3V	Capture Register 3 Value	0180_H	U, PV	BE	Page 23-156
CC80INTS	Interrupt Status	$01A0_H$	U, PV	BE	Page 23-157
CC80INTE	Interrupt Enable	$01A4_H$	U, PV	U, PV	Page 23-159
CC80SRS	Interrupt Configuration	$01A8_H$	U, PV	U, PV	Page 23-161
CC80SWS	Interrupt Status Set	$01AC_H$	U, PV	U, PV	Page 23-163
CC80SWR	Interrupt Status Clear	$01B0_H$	U, PV	U, PV	Page 23-165

Table 23-14 Register Overview of CCU8 (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
CC81 Registers					
CC81INS	Input Selector Unit Configuration	0200_H	U, PV	U, PV	Page 23-123
CC81CMC	Connection Matrix Configuration	0204_H	U, PV	U, PV	Page 23-125
CC81TCST	Timer Run Status	0208_H	U, PV	BE	Page 23-129
CC81TCSET	Timer Run Set	$020C_H$	U, PV	U,PV	Page 23-130
CC81TCCLR	Timer Run Clear	0210_H	U, PV	U, PV	Page 23-131
CC81TC	General Timer Configuration	0214_H	U, PV	U, PV	Page 23-132
CC81PSL	Output Passive Level Configuration	0218_H	U, PV	U, PV	Page 23-138
CC81DIT	Dither Configuration	$021C_H$	U, PV	BE	Page 23-139
CC81DITS	Dither Shadow Register	0220_H	U, PV	U, PV	Page 23-140
CC81PSC	Prescaler Configuration	0224_H	U, PV	U, PV	Page 23-140
CC81FPC	Prescaler Compare Value	0228_H	U, PV	U, PV	Page 23-141
CC81FPCS	Prescaler Shadow Compare Value	$022C_H$	U, PV	U, PV	Page 23-142
CC81PR	Timer Period Value	0230_H	U, PV	BE	Page 23-143
CC81PRS	Timer Period Shadow Value	0234_H	U, PV	U, PV	Page 23-144
CC81CR1	Timer Compare Value for Channel 1	0238_H	U, PV	BE	Page 23-144
CC81CR1S	Timer Compare Shadow Value for Channel 1	$023C_H$	U, PV	U, PV	Page 23-145
CC81CR2	Timer Compare Value for Channel 2	0240_H	U, PV	BE	Page 23-146
CC81CR2S	Timer Compare Shadow Value for Channel 2	0244_H	U, PV	U, PV	Page 23-147
CC81CHC	Channel Control	0248_H	U, PV	U, PV	Page 23-148
CC81DTC	Dead Time Control	$024C_H$	U, PV	U, PV	Page 23-150
CC81DC1R	Channel 1 Dead Time Counter Values	0250_H	U, PV	U, PV	Page 23-151

Table 23-14 Register Overview of CCU8 (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
CC81DC2R	Channel 2 Dead Time Counter Values	0254_H	U, PV	U, PV	Page 23-152
CC81TIMER	Timer Current Value	0270_H	U, PV	U, PV	Page 23-152
CC81C0V	Capture Register 0 Value	0274_H	U, PV	BE	Page 23-153
CC81C1V	Capture Register 1 Value	0278_H	U, PV	BE	Page 23-154
CC81C2V	Capture Register 2 Value	$027C_H$	U, PV	BE	Page 23-155
CC81C3V	Capture Register 3 Value	0280_H	U, PV	BE	Page 23-156
CC81INTS	Interrupt Status	$02A0_H$	U, PV	BE	Page 23-157
CC81INTE	Interrupt Enable	$02A4_H$	U, PV	U, PV	Page 23-159
CC81SRS	Interrupt Configuration	$02A8_H$	U, PV	U, PV	Page 23-161
CC81SWS	Interrupt Status Set	$02AC_H$	U, PV	U, PV	Page 23-163
CC81SWR	Interrupt Status Clear	$02B0_H$	U, PV	U, PV	Page 23-165
CC82 Registers					
CC82INS	Input Selector Unit Configuration	0300_H	U, PV	U, PV	Page 23-123
CC82CMC	Connection Matrix Configuration	0304_H	U, PV	U, PV	Page 23-125
CC82TCST	Timer Run Status	0308_H	U, PV	BE	Page 23-129
CC82TCSET	Timer Run Set	$030C_H$	U, PV	U,PV	Page 23-130
CC82TCCLR	Timer Run Clear	0310_H	U, PV	U, PV	Page 23-131
CC82TC	General Timer Configuration	0314_H	U, PV	U, PV	Page 23-132
CC82PSL	Output Passive Level Configuration	0318_H	U, PV	U, PV	Page 23-138
CC82DIT	Dither Configuration	$031C_H$	U, PV	BE	Page 23-139
CC82DITS	Dither Shadow Register	0320_H	U, PV	U, PV	Page 23-140
CC82PSC	Prescaler Configuration	0324_H	U, PV	U, PV	Page 23-140
CC82FPC	Prescaler Compare Value	0328_H	U, PV	U, PV	Page 23-141
CC82FPCS	Prescaler Shadow Compare Value	$032C_H$	U, PV	U, PV	Page 23-142
CC82PR	Timer Period Value	0330_H	U, PV	BE	Page 23-143

Table 23-14 Register Overview of CCU8 (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
CC82PRS	Timer Period Shadow Value	0334_H	U, PV	U, PV	Page 23-144
CC82CR1	Timer Compare Value for Channel 1	0338_H	U, PV	BE	Page 23-144
CC82CR1S	Timer Compare Shadow Value for Channel 1	$033C_H$	U, PV	U, PV	Page 23-145
CC82CR2	Timer Compare Value for Channel 2	0340_H	U, PV	BE	Page 23-146
CC82CR2S	Timer Compare Shadow Value for Channel 2	0344_H	U, PV	U, PV	Page 23-147
CC82CHC	Channel Control	0348_H	U, PV	U, PV	Page 23-148
CC82DTC	Dead Time Control	$034C_H$	U, PV	U, PV	Page 23-150
CC82DC1R	Channel 1 Dead Time Counter Values	0350_H	U, PV	U, PV	Page 23-151
CC82DC2R	Channel 2 Dead Time Counter Values	0354_H	U, PV	U, PV	Page 23-152
CC82TIMER	Timer Current Value	0370_H	U, PV	U, PV	Page 23-152
CC82C0V	Capture Register 0 Value	0374_H	U, PV	BE	Page 23-153
CC82C1V	Capture Register 1 Value	0378_H	U, PV	BE	Page 23-154
CC82C2V	Capture Register 2 Value	$037C_H$	U, PV	BE	Page 23-155
CC82C3V	Capture Register 3 Value	0380_H	U, PV	BE	Page 23-156
CC82INTS	Interrupt Status	$03A0_H$	U, PV	BE	Page 23-157
CC82INTE	Interrupt Enable	$03A4_H$	U, PV	U, PV	Page 23-159
CC82SRS	Interrupt Configuration	$03A8_H$	U, PV	U, PV	Page 23-161
CC82SWS	Interrupt Status Set	$03AC_H$	U, PV	U, PV	Page 23-163
CC82SWR	Interrupt Status Clear	$03B0_H$	U, PV	U, PV	Page 23-165
CC83 Registers					
CC83INS	Input Selector Unit Configuration	0400_H	U, PV	U, PV	Page 23-123
CC83CMC	Connection Matrix Configuration	0404_H	U, PV	U, PV	Page 23-125
CC83TCST	Timer Run Status	0408_H	U, PV	BE	Page 23-129

Table 23-14 Register Overview of CCU8 (cont'd)

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
CC83TCSET	Timer Run Set	$040C_H$	U, PV	U,PV	Page 23-130
CC83TCCLR	Timer Run Clear	0410_H	U, PV	U, PV	Page 23-131
CC83TC	General Timer Configuration	0414_H	U, PV	U, PV	Page 23-132
CC83PSL	Output Passive Level Configuration	0418_H	U, PV	U, PV	Page 23-138
CC83DIT	Dither Configuration	$041C_H$	U, PV	BE	Page 23-139
CC83DITS	Dither Shadow Register	0420_H	U, PV	U, PV	Page 23-140
CC83PSC	Prescaler Configuration	0424_H	U, PV	U, PV	Page 23-140
CC83FPC	Prescaler Compare Value	0428_H	U, PV	U, PV	Page 23-141
CC83FPCS	Prescaler Shadow Compare Value	$042C_H$	U, PV	U, PV	Page 23-142
CC83PR	Timer Period Value	0430_H	U, PV	BE	Page 23-143
CC83PRS	Timer Period Shadow Value	0434_H	U, PV	U, PV	Page 23-144
CC83CR1	Timer Compare Value for Channel 1	0438_H	U, PV	BE	Page 23-144
CC83CR1S	Timer Compare Shadow Value for Channel 1	$043C_H$	U, PV	U, PV	Page 23-145
CC83CR2	Timer Compare Value for Channel 2	0440_H	U, PV	BE	Page 23-146
CC83CR2S	Timer Compare Shadow Value for Channel 2	0444_H	U, PV	U, PV	Page 23-147
CC83CHC	Channel Control	0448_H	U, PV	U, PV	Page 23-148
CC83DTC	Dead Time Control	$044C_H$	U, PV	U, PV	Page 23-150
CC83DC1R	Channel 1 Dead Time Counter Values	0450_H	U, PV	U, PV	Page 23-151
CC83DC2R	Channel 2 Dead Time Counter Values	0454_H	U, PV	U, PV	Page 23-152
CC83TIMER	Timer Current Value	0470_H	U, PV	U, PV	Page 23-152
CC83C0V	Capture Register 0 Value	0474_H	U, PV	BE	Page 23-153
CC83C1V	Capture Register 1 Value	0478_H	U, PV	BE	Page 23-154
CC83C2V	Capture Register 2 Value	$047C_H$	U, PV	BE	Page 23-155

Table 23-14 Register Overview of CCU8 (cont'd)

Short Name	Description	Offset Addr.[1]	Access Mode		Description See
			Read	Write	
CC83C3V	Capture Register 3 Value	0480_H	U, PV	BE	Page 23-156
CC83INTS	Interrupt Status	$04A0_H$	U, PV	BE	Page 23-157
CC83INTE	Interrupt Enable	$04A4_H$	U, PV	U, PV	Page 23-159
CC83SRS	Interrupt Configuration	$04A8_H$	U, PV	U, PV	Page 23-161
CC83SWS	Interrupt Status Set	$04AC_H$	U, PV	U, PV	Page 23-163
CC83SWR	Interrupt Status Clear	$04B0_H$	U, PV	U, PV	Page 23-165

1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

23.7.1 Global Registers

GCTRL

The register contains the global configuration fields that affect all the timer slices inside CCU8.

GCTRL
Global Control Register (0000_H) Reset Value: 00000000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSDE		MSE3	MSE2	MSE1	MSE0	SUSCFG		0		PCIS		0	PRBC		
rw		rw	rw	rw	rw	rw		r		rw		r	rw		

Field	Bits	Type	Description
PRBC	[2:0]	rw	**Prescaler Clear Configuration** This register controls the how the prescaler Run Bit and internal registers are cleared. 000_B SW only 001_B **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC80 is cleared. 010_B **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC81 is cleared. 011_B **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC82 is cleared. 100_B **GSTAT**.PRB and prescaler registers are cleared when the Run Bit of CC83 is cleared.
PCIS	[5:4]	rw	**Prescaler Input Clock Selection** 00_B Module clock 01_B CCU8x.ECLKA 10_B CCU8x.ECLKB 11_B CCU8x.ECLKC
SUSCFG	[9:8]	rw	**Suspend Mode Configuration** This field controls the entering in suspend mode for all the CCU8 slices. 00_B Suspend request ignored. The module never enters in suspend 01_B Stops all the running slices immediately. Safe stop is not applied. 10_B Stops the block immediately and clamps all the outputs to PASSIVE state. Safe stop is applied. 11_B Waits for the roll over of each slice to stop and clamp the slices outputs. Safe stop is applied.
MSE0	10	rw	**Slice 0 Multi Channel shadow transfer enable** When this field is set, a shadow transfer of slice 0 can be requested not only by SW but also via the CCU8x.MCSS input. 0_B Shadow transfer can only be requested by SW 1_B Shadow transfer can be requested via SW and via the CCU8x.MCSS input.

Field	Bits	Type	Description
MSE1	11	rw	**Slice 1 Multi Channel shadow transfer enable** When this field is set, a shadow transfer of slice 1 can be requested not only by SW but also via the CCU8x.MCSS input. 0_B Shadow transfer can only be requested by SW 1_B Shadow transfer can be requested via SW and via the CCU8x.MCSS input.
MSE2	12	rw	**Slice 2 Multi Channel shadow transfer enable** When this field is set, a shadow transfer of slice 2 can be requested not only by SW but also via the CCU8x.MCSS input. 0_B Shadow transfer can only be requested by SW 1_B Shadow transfer can be requested via SW and via the CCU8xMCSS input.
MSE3	13	rw	**Slice 3 Multi Channel shadow transfer enable** When this field is set, a shadow transfer of slice 3 can be requested not only by SW but also via the CCU8x.MCSS input. 0_B Shadow transfer can only be requested by SW 1_B Shadow transfer can be requested via SW and via the CCU8x.MCSS input.
MSDE	[15:14]	rw	**Multi Channel shadow transfer request configuration** This field configures the type of shadow transfer requested via the CCU8x.MCSS input. The field **CC8yTC**.MSEx needs to be set in order for this configuration to have any effect. 00_B Only the shadow transfer for period and compare values is requested 01_B Shadow transfer for the compare, period and prescaler compare values is requested 10_B Reserved 11_B Shadow transfer for the compare, period, prescaler and dither compare values is requested

Field	Bits	Type	Description
0	3, [7:6], [31:16]	r	**Reserved** A read always returns 0.

GSTAT

The register contains the status of the prescaler and each timer slice (idle mode or running).

GSTAT
Global Status Register **(0004_H)** **Reset Value: $0000000F_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					PCRB	0	PRB	0				S3I	S2I	S1I	S0I
r					rh	r	rh	r				rh	rh	rh	rh

Field	Bits	Type	Description
S0I	0	rh	**CC80 IDLE status** This bit indicates if the CC80 slice is in IDLE mode or not. In IDLE mode the clocks for the CC80 slice are stopped. 0_B Running 1_B Idle
S1I	1	rh	**CC81 IDLE status** This bit indicates if the CC81 slice is in IDLE mode or not. In IDLE mode the clocks for the CC81 slice are stopped. 0_B Running 1_B Idle

Field	Bits	Type	Description
S2I	2	rh	**CC82 IDLE status** This bit indicates if the CC82 slice is in IDLE mode or not. In IDLE mode the clocks for the CC82 slice are stopped. 0_B Running 1_B Idle
S3I	3	rh	**CC83 IDLE status** This bit indicates if the CC83 slice is in IDLE mode or not. In IDLE mode the clocks for the CC83 slice are stopped. 0_B Running 1_B Idle
PRB	8	rh	**Prescaler Run Bit** 0_B Prescaler is stopped 1_B Prescaler is running
PCRB	10	rh	**Parity Checker Run Bit** 0_B Parity Checker is stopped 1_B Parity Checker is running
0	[7:4], 9, [31:11]	r	**Reserved** Read always returns 0.

GIDLS

Through this register one can set the prescaler and the specific timer slices into idle mode.

GIDLS
Global Idle Set **(0008_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					CPCH	PSIC	CPRB	0				SS3I	SS2I	SS1I	SS0I
r					w	w	w	r				w	w	w	w

Field	Bits	Type	Description
SS0I	0	w	**CC80 IDLE mode set** Writing a 1_B to this bit sets the CC80 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared. A read access always returns 0.
SS1I	1	w	**CC81 IDLE mode set** Writing a 1_B to this bit sets the CC81 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared. A read access always returns 0.
SS2I	2	w	**CC82 IDLE mode set** Writing a 1_B to this bit sets the CC82 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared. A read access always returns 0.
SS3I	3	w	**CC83 IDLE mode set** Writing a 1_B to this bit sets the CC83 slice in IDLE mode. The clocks for the slice are stopped when in IDLE mode. When entering IDLE, the internal slice registers are not cleared. A read access always returns 0.
CPRB	8	w	**Prescaler$_B$ Run Bit Clear** Writing a 1 into this register clears the Run Bit of the prescaler. Prescaler internal registers are not cleared. A read always returns 0.
PSIC	9	w	**Prescaler clear** Writing a 1_B to this register clears the prescaler counter. It also loads the PSIV into the PVAL field for all Timer Slices. This performs a re alignment of the timer clock for all Slices. The Run Bit of the prescaler is not cleared. A read always returns 0.

Field	Bits	Type	Description
CPCH	10	w	**Parity Checker Run bit clear** Writing a 1_B to this register clears the run bit of the parity checker. All the internal registers are cleared. The status bit value is kept, **GPCHK**.PCST. A read always returns 0.
0	[7:4], [31:11]	r	**Reserved** Read always returns 0.

GIDLC

Through this register one can remove the prescaler and the specific timer slices from idle mode.

GIDLC
Global Idle Clear **(000C$_H$)** **Reset Value: 00000000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					SPCH	0	SPRB	0				CS3I	CS2I	CS1I	CS0I
r					w	r	w	r				w	w	w	w

Field	Bits	Type	Description
CS0I	0	w	**CC80 IDLE mode clear** Writing a 1_B to this bit removes the CC80 from IDLE mode. No clear to the internal slice register is done. A read access always returns 0.
CS1I	1	w	**CC81 IDLE mode clear** Writing a 1_B to this bit removes the CC81 from IDLE mode. No clear to the internal slice register is done. A read access always returns 0.
CS2I	2	w	**CC82 IDLE mode clear** Writing a 1_B to this bit removes the CC82 from IDLE mode. No clear to the internal slice register is done. A read access always returns 0.

Field	Bits	Type	Description
CS3I	3	w	**CC83 IDLE mode clear** Writing a 1_B to this bit removes the CC83 from IDLE mode. No clear to the internal slice register is done. A read access always returns 0.
SPRB	8	w	**Prescaler Run Bit Set** Writing a 1_B into this register sets the Run Bit of the prescaler. Prescaler internal registers are not cleared. A read always returns 0.
SPCH	10	w	**Parity Checker run bit set** Writing a 1_B into this register sets the Run Bit of the parity checker. A read always returns 0.
0	[7:4], 9, [31:11]	r	**Reserved** Read always returns 0.

GCSS

Through this register one can request a shadow transfer for the specific timer slice(s) and set the status bit for each of the compare channels.

GCSS
Global Channel Set **(0010_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								**S3S T2S**	**S2S T2S**	**S1S T2S**	**S0S T2S**	**S3S T1S**	**S2S T1S**	**S1S T1S**	**S0S T1S**
r								w	w	w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**S3P SE**	**S3D SE**	**S3S E**	**0**	**S2P SE**	**S2D SE**	**S2S E**	**0**	**S1P SE**	**S1D SE**	**S1S E**	**0**	**S0P SE**	**S0D SE**	**S0S E**
r	w	w	w	r	w	w	w	r	w	w	w	r	w	w	w

Field	Bits	Type	Description
S0SE	0	w	**Slice 0 shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S0SS field, enabling then a shadow transfer for the Period, Compare and Passive level values. A read always returns 0.

Field	Bits	Type	Description
S0DSE	1	w	**Slice 0 Dither shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S0DSS field, enabling then a shadow transfer for the Dither compare value. A read always returns 0.
S0PSE	2	w	**Slice 0 Prescaler shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S0PSS field, enabling then a shadow transfer for the prescaler compare value. A read always returns 0.
S1SE	4	w	**Slice 1 shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S1SS field, enabling then a shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S1DSE	5	w	**Slice 1 Dither shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S1DSS field, enabling then a shadow transfer for the Dither compare value. A read always returns 0.
S1PSE	6	w	**Slice 1 Prescaler shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S1PSS field, enabling then a shadow transfer for the prescaler compare value. A read always returns 0.
S2SE	8	w	**Slice 2 shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S2SS field, enabling then a shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S2DSE	9	w	**Slice 2 Dither shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S2DSS field, enabling then a shadow transfer for the Dither compare value. A read always returns 0.

Field	Bits	Type	Description
S2PSE	10	w	**Slice 2 Prescaler shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S2PSS field, enabling then a shadow transfer for the prescaler compare value. A read always returns 0.
S3SE	12	w	**Slice 3 shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S3SS field, enabling then a shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S3DSE	13	w	**Slice 3 Dither shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S3DSS field, enabling then a shadow transfer for the Dither compare value. A read always returns 0.
S3PSE	14	w	**Slice 3 Prescaler shadow transfer set enable** Writing a 1_B to this bit will set the **GCST**.S3PSS field, enabling then a shadow transfer for the prescaler compare value. A read always returns 0.
S0ST1S	16	w	**Slice 0 status bit 1 set** Writing a 1_B into this register sets the compare channel 1 status bit of slice 0 (**GCST**.CC80ST1). A read always returns 0.
S1ST1S	17	w	**Slice 1 status bit 1 set** Writing a 1_B into this register sets the compare channel 1 status bit of slice 1 (**GCST**.CC81ST1). A read always returns 0.
S2ST1S	18	w	**Slice 2 status bit 1 set** Writing a 1_B into this register sets the compare channel 1 status bit of slice 2 (**GCST**.CC82ST1). A read always returns 0.
S3ST1S	19	w	**Slice 3 status bit 1 set** Writing a 1_B into this register sets the compare channel 2 status bit of slice 3 (**GCST**.CC83ST1). A read always returns 0.

Field	Bits	Type	Description
S0ST2S	20	w	**Slice 0 status bit 2 set** Writing a 1_B into this register sets the compare channel 2 status bit of slice 0 (**GCST**.CC80ST2). A read always returns 0.
S1ST2S	21	w	**Slice 1 status bit 2 set** Writing a 1_B into this register sets the compare channel 2 status bit of slice 1 (**GCST**.CC81ST2). A read always returns 0.
S2ST2S	22	w	**Slice 2 status bit 2 set** Writing a 1_B into this register sets the compare channel 2 status bit of slice 2 (**GCST**.CC82ST2). A read always returns 0.
S3ST2S	23	w	**Slice 3 status bit 2 set** Writing a 1_B into this register sets the compare channel 2 status bit of slice 3 (**GCST**.CC83ST2). A read always returns 0.
0	3, 7, 11, 15, [31:24]	r	**Reserved** Read always returns 0.

GCSC

Through this register one can reset a shadow transfer request for the specific timer slice and clear the status bit for each the compare channels.

GCSC
Global Channel Clear **(0014_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								**S3S T2C**	**S2S T2C**	**S1S T2C**	**S0S T2C**	**S3S T1C**	**S2S T1C**	**S1S T1C**	**S0S T1C**
r								w	w	w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**S3P SC**	**S3D SC**	**S3S C**	**0**	**S2P SC**	**S2D SC**	**S2S C**	**0**	**S1P SC**	**S1D SC**	**S1S C**	**0**	**S0P SC**	**S0D SC**	**S0S C**
r	w	w	w	r	w	w	w	r	w	w	w	r	w	w	w

Field	Bits	Type	Description
S0SC	0	w	**Slice 0 shadow transfer request clear** Writing a 1_B to this bit will clear the **GCST**.S0SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S0DSC	1	w	**Slice 0 Dither shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S0DSS field, canceling any pending shadow transfer for the Dither compare value. A read always returns 0.
S0PSC	2	w	**Slice 0 Prescaler shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S0PSS field, canceling any pending shadow transfer for the prescaler compare value. A read always returns 0.
S1SC	4	w	**Slice 1 shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S1SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S1DSC	5	w	**Slice 1 Dither shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S1DSS field, canceling any pending shadow transfer for the Dither compare value. A read always returns 0.
S1PSC	6	w	**Slice 1 Prescaler shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S1PSS field, canceling any pending shadow transfer for the prescaler compare value. A read always returns 0.
S2SC	8	w	**Slice 2 shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S2SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values. A read always returns 0.

Field	Bits	Type	Description
S2DSC	9	w	**Slice 2 Dither shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S2DSS field, canceling any pending shadow transfer for the Dither compare value. A read always returns 0.
S2PSC	10	w	**Slice 2 Prescaler shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S2PSS field, canceling any pending shadow transfer for the prescaler compare value. A read always returns 0.
S3SC	12	w	**Slice 3 shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S3SS field, canceling any pending shadow transfer for the Period, Compare and Passive level values. A read always returns 0.
S3DSC	13	w	**Slice 3 Dither shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S3DSS field, canceling any pending shadow transfer for the Dither compare value. A read always returns 0.
S3PSC	14	w	**Slice 3 Prescaler shadow transfer clear** Writing a 1_B to this bit will clear the **GCST**.S3PSS field, canceling any pending shadow transfer for the prescaler compare value. A read always returns 0.
S0ST1C	16	w	**Slice 0 status bit 1 clear** Writing a 1_B into this register clears the compare channel 1 status bit of slice 0 (**GCST**.CC80ST1). A read always returns 0.
S1ST1C	17	w	**Slice 1 status bit 1 clear** Writing a 1_B into this register clears the compare channel 1 status bit of slice 1 (**GCST**.CC81ST1). A read always returns 0.
S2ST1C	18	w	**Slice 2 status bit 1 clear** Writing a 1_B into this register clears the compare channel 1 status bit of slice 2 (**GCST**.CC82ST1). A read always returns 0.

Field	Bits	Type	Description
S3ST1C	19	w	**Slice 3 status bit 1 clear** Writing a 1_B into this register clears the compare channel 1 status bit of slice 3 (**GCST**.CC83ST1). A read always returns 0.
S0ST2C	20	w	**Slice 0 status bit 2 clear** Writing a 1_B into this register clears the compare channel 2 status bit of slice 0 (**GCST**.CC80ST2). A read always returns 0.
S1ST2C	21	w	**Slice 1 status bit 2 clear** Writing a 1_B into this register clears the compare channel 2 status bit of slice 1 (**GCST**.CC81ST2). A read always returns 0.
S2ST2C	22	w	**Slice 2 status bit 2 clear** Writing a 1_B into this register clears the compare channel 2 status bit of slice 2 (**GCST**.CC82ST2). A read always returns 0.
S3ST2C	23	w	**Slice 3 status bit 2 clear** Writing a 1_B into this register clears the compare channel 2 status bit of slice 3 (**GCST**.CC83ST2). A read always returns 0.
0	3, 7, 11, 15, [31:24]	r	**Reserved** Read always returns 0.

GCST

This register holds the information of the shadow transfer requests and of each timer slice status bit.

GCST
Global Channel status **(0018_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								CC83ST2	CC82ST2	CC81ST2	CC80ST2	CC83ST1	CC82ST1	CC81ST1	CC80ST1
r								rh	rh	rh	rh	rh	rh	rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	S3PSS	S3DSS	S3SS	0	S2PSS	S2DSS	S2SS	0	S1PSS	S1DSS	S1SS	0	S0PSS	S0DSS	S0SS
r	rh	rh	rh	r	rh	rh	rh	r	rh	rh	rh	r	rh	rh	rh

Field	Bits	Type	Description
S0SS	0	rh	**Slice 0 shadow transfer status** 0_B Shadow transfer has not been requested 1_B Shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S0DSS	1	rh	**Slice 0 Dither shadow transfer status** 0_B Dither shadow transfer has not been requested 1_B Dither shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S0PSS	2	rh	**Slice 0 Prescaler shadow transfer status** 0_B Prescaler shadow transfer has not been requested 1_B Prescaler shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S1SS	4	rh	**Slice 1 shadow transfer status** 0_B Shadow transfer has not been requested 1_B Shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.

Field	Bits	Type	Description
S1DSS	5	rh	**Slice 1 Dither shadow transfer status** 0_B Dither shadow transfer has not been requested 1_B Dither shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S1PSS	6	rh	**Slice 1 Prescaler shadow transfer status** 0_B Prescaler shadow transfer has not been requested 1_B Prescaler shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S2SS	8	rh	**Slice 2 shadow transfer status** 0_B Shadow transfer has not been requested 1_B Shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S2DSS	9	rh	**Slice 2 Dither shadow transfer status** 0_B Dither shadow transfer has not been requested 1_B Dither shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S2PSS	10	rh	**Slice 2 Prescaler shadow transfer status** 0_B Prescaler shadow transfer has not been requested 1_B Prescaler shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S3SS	12	rh	**Slice 3 shadow transfer status** 0_B Shadow transfer has not been requested 1_B Shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.

Field	Bits	Type	Description
S3DSS	13	rh	**Slice 3 Dither shadow transfer status** 0_B Dither shadow transfer has not been requested 1_B Dither shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
S3PSS	14	rh	**Slice 3 Prescaler shadow transfer status** 0_B Prescaler shadow transfer has not been requested 1_B Prescaler shadow transfer has been requested This field is cleared by HW after the requested shadow transfer has been executed.
CC80ST1	16	rh	**Slice 0 compare channel 1 status bit**
CC81ST1	17	rh	**Slice 1 compare channel 1 status bit**
CC82ST1	18	rh	**Slice 2 compare channel 1 status bit**
CC83ST1	19	rh	**Slice 3 compare channel 1 status bit**
CC80ST2	20	rh	**Slice 0 compare channel 2 status bit**
CC81ST2	21	rh	**Slice 1 compare channel 2 status bit**
CC82ST2	22	rh	**Slice 2 compare channel 2 status bit**
CC83ST2	23	rh	**Slice 3 compare channel 2 status bit**
0	3, 7, 11, 15, [31:24]	r	**Reserved** Read always returns 0.

GPCHK

This register contains the configuration for the Parity Check function of CCU8.

GPCHK
Parity Checker Configuration **(001C$_H$)** **Reset Value: 00000000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PCSEL3				PCSEL2				PCSEL1				PCSEL0			
rw				rw				rw				rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PCST	0							PCTS	PCDS		PISEL		PACS		PASE
rh	r							rw	rw		rw		rw		rw

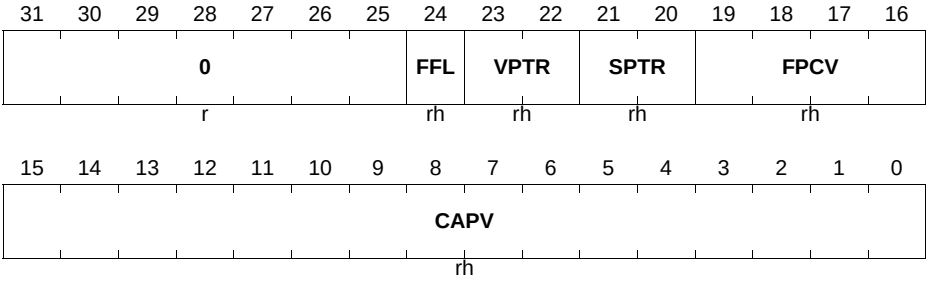
Field	Bits	Type	Description
PASE	0	rw	**Parity Checker Automatic start/stop** If this field is set, the parity checker run bit is automatically set, when the run bit of the selected slice is set and it is cleared when the run bit of the slice is cleared. The field PACS needs to be programmed accordingly.
PACS	[2:1]	rw	**Parity Checker Automatic start/stop selector** This fields selects to which slice the automatic start/stop of the parity checker is associated: 00_B CC80 01_B CC81 10_B CC82 11_B CC83
PISEL	[4:3]	rw	**Driver Input signal selector** This fields selects which signal contains the driver parity information: 00_B CC8x.GP01 - driver output is connected to event 1 of slice 0 01_B CC8x.GP11 - drive output is connected to event 1 of slice 1 10_B CC8x.GP21 - driver output is connected to event 1 of slice 2 11_B CC8x.GP31 - driver output is connected to event 1 of slice 3

Field	Bits	Type	Description
PCDS	[6:5]	rw	**Parity Checker Delay Input Selector** This fields selects which signal is controlling the delay between the change at the CCU8 outputs and effective change at the driver parity output: 00_B CCU8x.IGBTA 01_B CCU8x.IGBTB 10_B CCU8x.IGBTC 11_B CCU8x.IGBTD
PCTS	7	rw	**Parity Checker type selector** This fields selects if we have an odd or even parity: 0_B Even parity enabled 1_B Odd parity enabled
PCST	15	rh	**Parity Checker XOR status** This field contains the current value of the XOR chain.
PCSEL0	[19:16]	rw	**Parity Checker Slice 0 output selection** This fields selects which slice 0 outputs are going to be used to perform the parity check. The respective bit field needs to be set to 1_B to enable the output in the parity function. PCSEL0[0] - CCU8x.OUT00 PCSEL0[1] - CCU8x.OUT01 PCSEL0[2] - CCU8x.OUT02 PCSEL0[3] - CCU8x.OUT03
PCSEL1	[23:20]	rw	**Parity Checker Slice 1 output selection** Same description as PCSEL0.
PCSEL2	[27:24]	rw	**Parity Checker Slice 2 output selection** Same description as PCSEL0.
PCSEL3	[31:28]	rw	**Parity Checker Slice 3 output selection** Same description as PCSEL0.
0	[14:8]	r	**Reserved** Read always returns 0.

ECRD

This register holds the information related to the extended capture mode.

ECRD
Extended Capture Mode Read **(0050_H)** **Reset Value: 00000000_H**



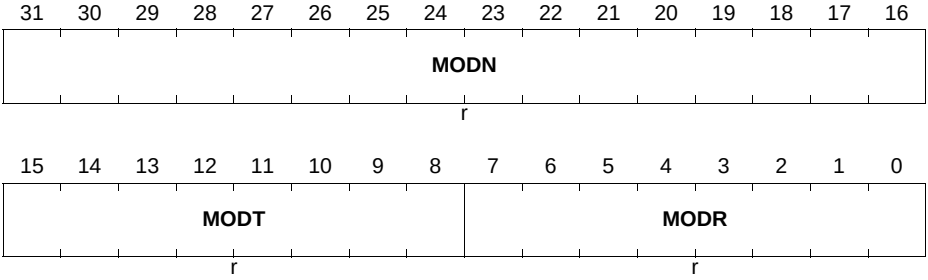
Field	Bits	Type	Description
CAPV	[15:0]	rh	**Timer Capture Value** This field contains the timer captured value
FPCV	[19:16]	rh	**Prescaler Capture value** This field contains the value of the prescaler clock division associated with the specific CAPV field
SPTR	[21:20]	rh	**Slice pointer** This field indicates the slice index in which the value was captured. 00_B CC80 01_B CC81 10_B CC82 11_B CC83
VPTR	[23:22]	rh	**Capture register pointer** This field indicates the capture register index in which the value was captured. 00_B Capture register 0 01_B Capture register 1 10_B Capture register 2 11_B Capture register 3
FFL	24	rh	**Full Flag** This bit indicates if the associated capture register contains a value. 0_B No new value was captured into this register 1_B A new value has been captured into this register

Field	Bits	Type	Description
0	[31:25]	r	**Reserved** Read always returns 0.

MIDR

This register contains the module identification number.

MIDR
Module Identification **(0080_H)** **Reset Value: $00A7C0XX_H$**



Field	Bits	Type	Description
MODR	[7:0]	r	**Module Revision** This bit field indicates the revision number of the module implementation (depending on the design step). The given value of 00_H is a placeholder for the actual number.
MODT	[15:8]	r	**Module Type**
MODN	[31:16]	r	**Module Number**

23.7.2 Slice (CC8y) Registers

CC8yINS

The register contains the configuration for the input selector.

CC8yINS (y = 0 - 3)
Input Selector Configuration **(0100_H + 0100_H * y)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	LPF2M		LPF1M		LPF0M		EV2LM	EV1LM	EV0LM	EV2EM		EV1EM		EV0EM	
r	rw		rw		rw		rw	rw	rw	rw		rw		rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				EV2IS				EV1IS				EV0IS			
r				rw				rw				rw			

Field	Bits	Type	Description
EV0IS	[3:0]	rw	**Event 0 signal selection** This field selects which pins is used for the event 0. 0000_B CCU8x.INyA 0001_B CCU8x.INyB 0010_B CCU8x.INyC 0011_B CCU8x.INyD 0100_B CCU8x.INyE 0101_B CCU8x.INyF 0110_B CCU8x.INyG 0111_B CCU8x.INyH 1000_B CCU8x.INyI 1001_B CCU8x.INyJ 1010_B CCU8x.INyK 1011_B CCU8x.INyL 1100_B CCU8x.INyM 1101_B CCU8x.INyN 1110_B CCU8x.INyO 1111_B CCU8x.INyP
EV1IS	[7:4]	rw	**Event 1 signal selection** Same as EV0IS description
EV2IS	[11:8]	rw	**Event 2 signal selection** Same as EV0IS description

Field	Bits	Type	Description
EV0EM	[17:16]	rw	**Event 0 Edge Selection** 00_B No action 01_B Signal active on rising edge 10_B Signal active on falling edge 11_B Signal active on both edges
EV1EM	[19:18]	rw	**Event 1 Edge Selection** Same as EV0EM description
EV2EM	[21:20]	rw	**Event 2 Edge Selection** Same as EV0EM description
EV0LM	22	rw	**Event 0 Level Selection** 0_B Active on HIGH level 1_B Active on LOW level
EV1LM	23	rw	**Event 1 Level Selection** Same as EV0LM description
EV2LM	24	rw	**Event 2 Level Selection** Same as EV0LM description
LPF0M	[26:25]	rw	**Event 0 Low Pass Filter Configuration** This field sets the number of consecutive counts for the Low Pass Filter of Event 0. The input signal value needs to remain stable for this number of counts (f_{CCU8}), so that a level/transition is accepted. 00_B LPF is disabled 01_B 3 clock cycles of f_{CCU8} 10_B 5 clock cycles of f_{CCU8} 11_B 7 clock cycles of f_{CCU8}
LPF1M	[28:27]	rw	**Event 1 Low Pass Filter Configuration** Same description as LPF0M
LPF2M	[30:29]	rw	**Event 2 Low Pass Filter Configuration** Same description as LPF0M
0	[15:12], 31	r	**Reserved** Read always returns 0.

CC8yCMC

The register contains the configuration for the connection matrix.

CC8yCMC (y = 0 - 3)
Connection Matrix Control **($0104_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0											TCE	MOS		TS	OFS
r											rw	rw		rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNTS		LDS		UDS		GATES		CAP1S		CAP0S		ENDS		STRTS	
rw		rw		rw		rw		rw		rw		rw		rw	

Field	Bits	Type	Description
STRTS	[1:0]	rw	**External Start Functionality Selector** Selects the Event that is going to be linked with the external start functionality. 00_B External Start Function deactivated 01_B External Start Function triggered by Event 0 10_B External Start Function triggered by Event 1 11_B External Start Function triggered by Event 2
ENDS	[3:2]	rw	**External Stop Functionality Selector** Selects the Event that is going to be linked with the external stop functionality. 00_B External Stop Function deactivated 01_B External Stop Function triggered by Event 0 10_B External Stop Function triggered by Event 1 11_B External Stop Function triggered by Event 2

Field	Bits	Type	Description
CAP0S	[5:4]	rw	**External Capture 0 Functionality Selector** Selects the Event that is going to be linked with the external capture for capture registers number 1 and 0. This function is used to capture the value of the timer into the capture registers 1 and 0. 00_B External Capture 0 Function deactivated 01_B External Capture 0 Function triggered by Event 0 10_B External Capture 0 Function triggered by Event 1 11_B External Capture 0 Function triggered by Event 2 *Note: If the field SCE is set, this functionality is deactivated.*
CAP1S	[7:6]	rw	**External Capture 1 Functionality Selector** Selects the Event that is going to be linked with the external capture for capture registers number 3 and 2. This function is used to capture the value of the timer into the capture registers 3 and 2. 00_B External Capture 1 Function deactivated 01_B External Capture 1 Function triggered by Event 0 10_B External Capture 1 Function triggered by Event 1 11_B External Capture 1 Function triggered by Event 2
GATES	[9:8]	rw	**External Gate Functionality Selector** Selects the Event that is going to be linked with the counter gating function. This function is used to gate the timer increment/decrement procedure. 00_B External Gating Function deactivated 01_B External Gating Function triggered by Event 0 10_B External Gating Function triggered by Event 1 11_B External Gating Function triggered by Event 2

Field	Bits	Type	Description
UDS	[11:10]	rw	**External Up/Down Functionality Selector** Selects the Event that is going to be linked with the Up/Down counting direction control. This function is used to control externally the timer increment/decrement operation. 00_B External Up/Down Function deactivated 01_B External Up/Down Function triggered by Event 0 10_B External Up/Down Function triggered by Event 1 11_B External Up/Down Function triggered by Event 2
LDS	[13:12]	rw	**External Timer Load Functionality Selector** Selects the Event that is going to be linked with the timer load function. The value present in the **CC8yCR1**/**CC8yCR2** (depending on the value of **CC8yTC**.TLS) is loaded into the specific slice timer. 00_B - External Load Function deactivated 01_B - External Load Function triggered by Event 0 10_B - External Load Function triggered by Event 1 11_B - External Load Function triggered by Event 2
CNTS	[15:14]	rw	**External Count Selector** Selects the Event that is going to be linked with the count function. The counter is going to be increment/decremented each time that a specific transition on the event is detected. 00_B External Count Function deactivated 01_B External Count Function triggered by Event 0 10_B External Count Function triggered by Event 1 11_B External Count Function triggered by Event 2 *Note: In CC40 this field doesn't exist. This is a read only reserved field. Read access always returns 0.*
OFS	16	rw	**Override Function Selector** This field enables the ST bit override functionality. 0_B Override functionality disabled 1_B Status bit trigger override connected to Event 1; Status bit value override connected to Event 2

Field	Bits	Type	Description
TS	17	rw	**Trap Function Selector** This field enables the trap functionality. 0_B Trap function disabled 1_B TRAP function connected to Event 2
MOS	[19:18]	rw	**External Modulation Functionality Selector** Selects the Event that is going to be linked with the external modulation function. 00_B - Modulation Function deactivated 01_B - Modulation Function triggered by Event 0 10_B - Modulation Function triggered by Event 1 11_B - Modulation Function triggered by Event 2
TCE	20	rw	**Timer Concatenation Enable** This bit enables the timer concatenation with the previous slice. 0_B Timer concatenation is disabled 1_B Timer concatenation is enabled *Note: In CC80 this field doesn't exist. This is a read only reserved field. Read access always returns 0.*
0	[31:21]	r	**Reserved** A read always returns 0

CC8yTCST

The register holds the status of the timer (running/stopped) and the information about the counting direction (up/down).

CC8yTCST (y = 0 - 3)
Slice Timer Status **($0108_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

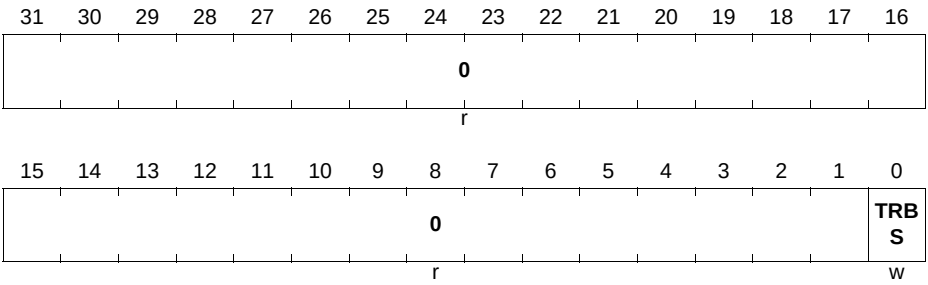
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0											DTR2	DTR1	0	CDIR	TRB
r											rh	rh	r	rh	rh

Field	Bits	Type	Description
TRB	0	rh	**Timer Run Bit** This field indicates if the timer is running. 0_B Timer is stopped 1_B Timer is running
CDIR	1	rh	**Timer Counting Direction** This filed indicates if the timer is being increment or decremented 0_B Timer is counting up 1_B Timer is counting down
DTR1	3	rh	**Dead Time Counter 1 Run bit** This field indicates if the dead time counter for linked with channel 1 is running. 0_B Dead Time counter is idle 1_B Dead Time counter is running
DTR2	4	rh	**Dead Time Counter 2 Run bit** This field indicates if the dead time counter for linked with channel 2 is running. 0_B Dead Time counter is idle 1_B Dead Time counter is running
0	2, [31:5]	r	**Reserved** Read always returns 0

CC8yTCSET

Through this register it is possible to start the timer.

CC8yTCSET (y = 0 - 3)
Slice Timer Run Set **($010C_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
TRBS	0	w	**Timer Run Bit set** Writing a 1_B into this field sets the run bit of the timer. The timer is not cleared. Read always returns 0.
0	[31:1]	r	**Reserved** Read always returns 0

CC8yTCCLR

Through this register it is possible to stop and clear the timer, and clearing also the dither counter

CC8yTCCLR (y = 0 - 3)
Slice Timer Clear **($0110_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0											DTC 2C	DTC 1C	DITC	TCC	TRB C
r											w	w	w	w	w

Field	Bits	Type	Description
TRBC	0	w	**Timer Run Bit Clear** Writing a 1_B into this field clears the run bit of the timer. The timer is not cleared. Read always returns 0.
TCC	1	w	**Timer Clear** Writing a 1_B into this field clears the timer value. Read always returns 0.
DITC	2	w	**Dither Counter Clear** Writing a 1_B into this field clears the dither counter. Read always returns 0.

Field	Bits	Type	Description
DTC1C	3	w	**Dead Time Counter 1 Clear** Writing a 1_B into this field clears the channel 1 dead time counter. The counter is stopped until a new start trigger is detected. Read always returns 0.
DTC2C	4	w	**Dead Time Counter 2 Clear** Writing a 1_B into this field clears the channel 2 dead time counter. The counter is stopped until a new start trigger is detected. Read always returns 0.
0	[31:5]	r	**Reserved** Read always returns 0

CC8yTC

This register holds the several possible configurations for the timer operation.

CC8yTC (y = 0 - 3)
Slice Timer Control **($0114_H + 0100_H * y$)** **Reset Value: 18000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**STOS**		**EME**		**MCME2**	**MCME1**	**EMT**	**EMS**	**TRPSW**	**TRPSE**	**TRAPE3**	**TRAPE2**	**TRAPE1**	**TRAPE0**	**FPE**
r	rw		rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIM	**DITHE**		**CCS**	**SCE**	**STRM**	**ENDM**		**TLS**	**CAPC**		**ECM**	**CMOD**	**CLST**	**TSSM**	**TCM**
rw	rw		rw	rw	rw	rw		rw	rw		rw	rh	rw	rw	rw

Field	Bits	Type	Description
TCM	0	rw	**Timer Counting Mode** This field controls the actual counting scheme of the timer. 0_B Edge aligned mode 1_B Center aligned mode *Note: When using an external signal to control the counting direction, the counting scheme is always edge aligned.*

Field	Bits	Type	Description
TSSM	1	rw	**Timer Single Shot Mode** This field controls the single shot mode. This is applicable in edge and center aligned modes. 0_B Single shot mode is disabled 1_B Single shot mode is enabled
CLST	2	rw	**Shadow Transfer on Clear** Setting this bit to 1_B enables a shadow transfer when a timer clearing action is done (by SW or by an external event). Notice that the shadow transfer enable bitfields on the **GCST** register still need to be set to 1_B via software.
CMOD	3	rh	**Capture Compare Mode** This field indicates in which mode the slice is operating. The default value is compare mode. The capture mode is automatically set by the HW when an external signal is mapped to a capture trigger. 0_B Compare Mode 1_B Capture Mode
ECM	4	rw	**Extended Capture Mode** This field control the Capture mode of the specific slice. It only has effect if the CMOD bit is 1_B. 0_B Normal Capture Mode. Clear of the Full Flag of each capture register is done by accessing the registers individually only. 1_B Extended Capture Mode. Clear of the Full Flag of each capture register is done not only by accessing the individual registers but also by accessing the **ECRD** register. When reading the **ECRD** register, only the capture register register full flag pointed by the **ECRD**.VPTR is cleared

Field	Bits	Type	Description
CAPC	[6:5]	rw	**Clear on Capture Control** 00_B Timer is never cleared on a capture event 01_B Timer is cleared on a capture event into capture registers 2 and 3. (When SCE = 1_B, Timer is always cleared in a capture event) 10_B Timer is cleared on a capture event into capture registers 0 and 1. (When SCE = 1_B, Timer is always cleared in a capture event) 11_B Timer is always cleared in a capture event.
TLS	7	rw	**Timer Load selector** 0_B Timer is loaded with the value of CR1 1_B Timer is loaded with the value of CR2
ENDM	[9:8]	rw	**Extended Stop Function Control** This field controls the extended functions of the external Stop signal. 00_B Clears the timer run bit only (default stop) 01_B Clears the timer only (flush) 10_B Clears the timer and run bit (flush/stop) 11_B Reserved *Note: When using an external up/down signal the flush operation sets the timer with zero if the counter is counting up and with the Period value if the counter is being decremented.*
STRM	10	rw	**Extended Start Function Control** This field controls the extended functions od the external Start signal. 0_B Sets run bit only (default start) 1_B Clears the timer and sets run bit, if not set (flush/start) *Note: When using an external up/down signal the flush operation sets the timer with zero if the counter is being incremented and with the Period value if the counter is being decremented.*

Field	Bits	Type	Description
SCE	11	rw	**Equal Capture Event enable** 0_B Capture into **CC8yC0V**/**CC8yC1V** registers control by CCycapt0 and capture into **CC8yC3V**/**CC8yC2V** control by CCycapt1 1_B Capture into **CC8yC0V**/**CC8yC1V** and **CC8yC3V**/**CC8yC2V** control by CCycapt1
CCS	12	rw	**Continuous Capture Enable** 0_B The capture into a specific capture register is done with the rules linked with the full flags, described at **Section 23.2.8.6**. 1_B The capture into the capture registers is always done regardless of the full flag status (even if the register has not been read back).
DITHE	[14:13]	rw	**Dither Enable** This field controls the dither mode for the slice. See **Section 23.2.12**. 00_B Dither is disabled 01_B Dither is applied to the Period 10_B Dither is applied to the Compare 11_B Dither is applied to the Period and Compare
DIM	15	rw	**Dither input selector** This fields selects if the dither control signal is connected to the dither logic of the specific slice of is connected to the dither logic of slice 0. Notice that even if this field is set to 1_B, the field DITHE still needs to be programmed. 0_B Slice is using it own dither unit 1_B Slice is connected to the dither unit of slice 0.
FPE	16	rw	**Floating Prescaler enable** Setting this bit to 1_B enables the floating prescaler mode. 0_B Floating prescaler mode is disabled 1_B Floating prescaler mode is enabled

Field	Bits	Type	Description
TRAPE0	17	rw	**TRAP enable for CCU8x.OUTy0** Setting this bit to 1 enables the TRAP action at the CCU8x.OUTy0 output pin. After mapping an external signal to the TRAP functionality, the user must set this field to 1 to activate the effect of the TRAP on the specific output pin. Writing a 0 into this field disables the effect of the TRAP function regardless of the state of the input signal. 0_B TRAP functionality has no effect on the CCU8x.OUTy0 output 1_B TRAP functionality affects the CCU8x.OUTy0 output
TRAPE1	18	rw	**TRAP enable for CCU8x.OUTy1** TRAP enable for the CCU8x.OUTy1. Same description as for the TRAPE0 field.
TRAPE2	19	rw	**TRAP enable for CCU8x.OUTy2** TRAP enable for the CCU8x.OUTy2. Same description as for the TRAPE0 field.
TRAPE3	20	rw	**TRAP enable for CCU8x.OUTy3** TRAP enable for the CCU8x.OUTy3. Same description as for the TRAPE0 field.
TRPSE	21	rw	**TRAP Synchronization Enable** Writing a 1 into this bit enables a synchronous exiting with the PWM period of the trap state. 0_B Exiting from TRAP state isn't synchronized with the PWM signal 1_B Exiting from TRAP state is synchronized with the PWM signal
TRPSW	22	rw	**TRAP State Clear Control** 0_B The slice exits the TRAP state automatically when the TRAP condition is not present (Trap state cleared by HW and SW) 1_B The TRAP state can only be exited by a SW request.

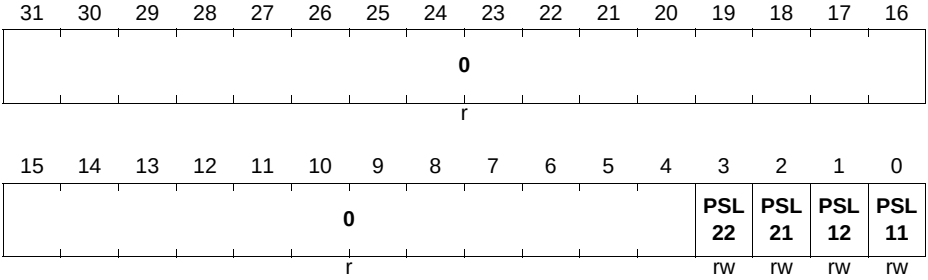
Field	Bits	Type	Description
EMS	23	rw	**External Modulation Synchronization** Setting this bit to 1 enables the synchronization of the external modulation functionality with the PWM period. 0_B External Modulation functionality is not synchronized with the PWM signal 1_B External Modulation functionality is synchronized with the PWM signal
EMT	24	rw	**External Modulation Type** This field selects if the external modulation event is clearing the CC8ySTx bits or if is gating the outputs. 0_B External Modulation functionality is clearing the CC8ySTx bits. 1_B External Modulation functionality is gating the outputs.
MCME1	25	rw	**Multi Channel Mode Enable for Channel 1** 0_B Multi Channel Mode in Channel 1 is disabled 1_B Multi Channel Mode in Channel 1 is enabled
MCME2	26	rw	**Multi Channel Mode Enable for Channel 2** 0_B Multi Channel Mode in Channel 2 is disabled 1_B Multi Channel Mode in Channel 2 is enabled
EME	[28:27]	rw	**External Modulation Channel enable** This field controls in which channel, the modulation functionality has effect. The modulations functionality needs to be previously enabled by setting the **CC8yCMC**.OFS = 11_B. 00_B External Modulation functionality doesn't affect any channel 01_B External Modulation only applied on channel 1 10_B External Modulation only applied on channel 2 11_B External Modulation applied on both channels
STOS	[30:29]	rw	**Status bit output selector** This field selects to which channel the output CC8ySTy is mapped. 00_B CC8yST1 forward to CCU8x.STy 01_B CC8yST2 forward to CCU8x.STy 10_B CC8yST1 AND CC8yST2 forward to CCU8x.STy 11_B Reserved

Field	Bits	Type	Description
0	31	r	**Reserved** Read always returns 0.

CC8yPSL

This register holds the configuration for the output passive level control.

CC8yPSL (y = 0 - 3)
Passive Level Config **($0118_H + 0100_H * y$)** **Reset Value: 00000000_H**



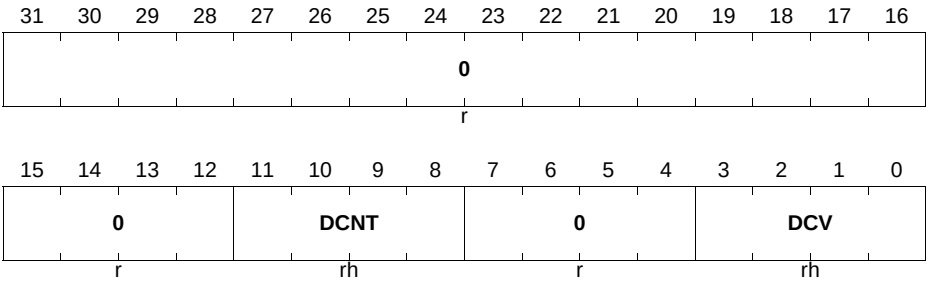
Field	Bits	Type	Description
PSL11	0	rw	**Output Passive Level for CCU8x.OUTy0** This field controls the passive level of the CCU8x.OUTy0. 0_B Passive Level is LOW 1_B Passive Level is HIGH A write always addresses the shadow register, while a read always returns the current used value.
PSL12	1	rw	**Output Passive Level for CCU8x.OUTy1** This field controls the passive level of the CCU8x.OUTy1. 0_B Passive Level is LOW 1_B Passive Level is HIGH A write always addresses the shadow register, while a read always returns the current used value.

Field	Bits	Type	Description
PSL21	2	rw	**Output Passive Level for CCU8x.OUTy2** This field controls the passive level of the CCU8x.OUTy2. 0_B Passive Level is LOW 1_B Passive Level is HIGH A write always addresses the shadow register, while a read always returns the current used value.
PSL22	3	rw	**Output Passive Level for CCU8x.OUTy1** This field controls the passive level of the CCU8x.OUTy3. 0_B Passive Level is LOW 1_B Passive Level is HIGH A write always addresses the shadow register, while a read always returns the current used value.
0	[31:4]	r	**Reserved** A read access always returns 0

CC8yDIT

This register holds the current dither compare and dither counter values.

CC8yDIT (y = 0 - 3)
Dither Config **($011C_H + 0100_H * y$)** **Reset Value: 00000000_H**

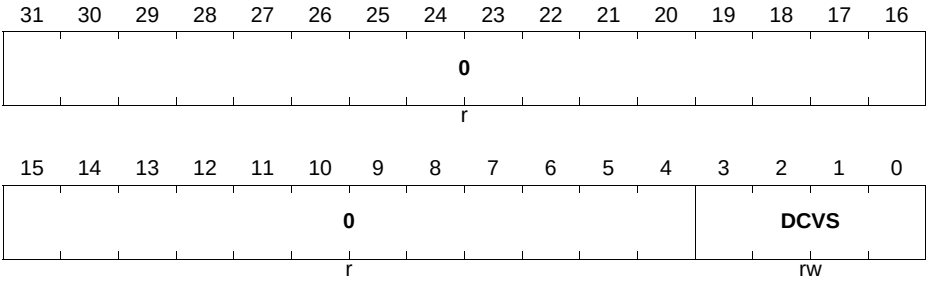


Field	Bits	Type	Description
DCV	[3:0]	rh	**Dither compare Value** This field contains the value used for the dither comparison. This value is updated when a shadow transfer occurs with the **CC8yDITS**.DCVS.
DCNT	[11:8]	rh	**Dither counter actual value**
0	[7:4], [31:12]	r	**Reserved** Read always returns 0.

CC8yDITS

This register contains the value that is going to be loaded into the **CC8yDIT**.DCV when the next shadow transfer occurs.

CC8yDITS (y = 0 - 3)
Dither Shadow Register **($0120_H + 0100_H * y$)** **Reset Value: 00000000_H**

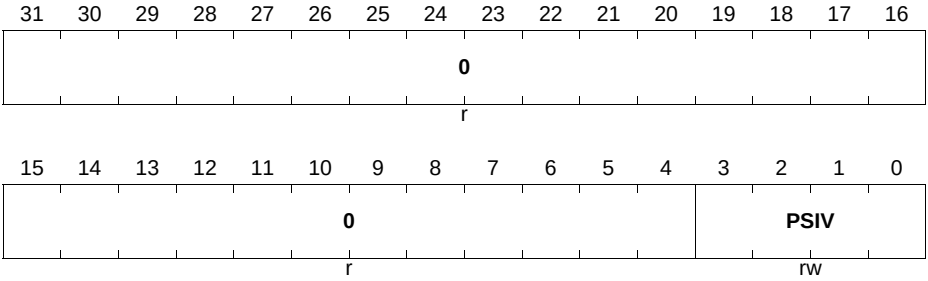


Field	Bits	Type	Description
DCVS	[3:0]	rw	**Dither Shadow Compare Value** This field contains the value that is going to be set on the dither compare value, **CC8yDIT**.DCV, within the next shadow transfer.
0	[31:4]	r	**Reserved** Read always returns 0.

CC8yPSC

This register contains the value that is loaded into the prescaler during restart.

CC8yPSC (y = 0 - 3)
Prescaler Control **($0124_H + 0100_H * y$)** **Reset Value: 00000000_H**

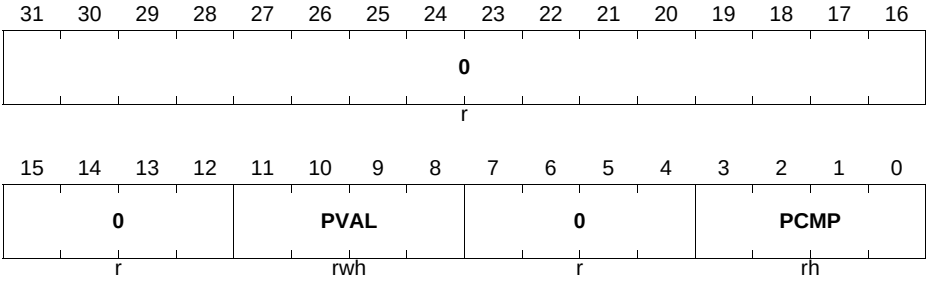


Field	Bits	Type	Description
PSIV	[3:0]	rw	**Prescaler Initial Value** This field contains the value that is applied to the Prescaler at startup. When floating prescaler mode is used, this value is applied when a timer compare match AND prescaler compare match occurs or when a capture event is triggered.
0	[31:4]	r	**Reserved** Read always returns 0.

CC8yFPC

This register contains the value used for the floating prescaler compare and the actual prescaler division value.

CC8yFPC (y = 0 - 3)
Floating Prescaler Control **(0128_H + 0100_H * y)** **Reset Value: 00000000_H**

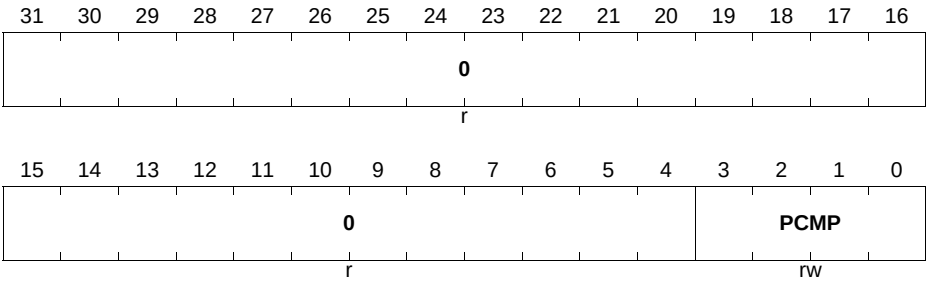


Field	Bits	Type	Description
PCMP	[3:0]	rh	**Floating Prescaler Compare Value** This field contains the value used to compare the actual prescaler value. The comparison is triggered by the Timer Compare match event. See **Section 23.2.13.2**.
PVAL	[11:8]	rwh	**Actual Prescaler Value** See **Table 23-9**. Writing into this register is only possible when the prescaler is stopped. When the floating prescaler mode is not used, this value is equal to the **CC8yPSC**.PSIV.
0	[7:4], [15:12], [31:16]	r	**Reserved** Read always returns 0.

CC8yFPCS

This register contains the value that is going to be transferred to the **CC8yFPC**.PCMP field within the next shadow transfer update.

CC8yFPCS (y = 0 - 3)
Floating Prescaler Shadow **($012C_H + 0100_H * y$)** **Reset Value: 00000000_H**

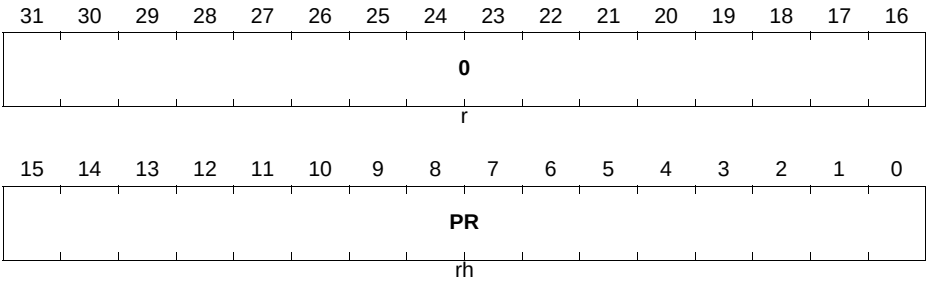


Field	Bits	Type	Description
PCMP	[3:0]	rw	**Floating Prescaler Shadow Compare Value** This field contains the value that is going to be set on the **CC8yFPC**.PCMP within the next shadow transfer. See **Table 23-9**.
0	[31:4]	r	**Reserved** Read always returns 0.

CC8yPR

This register contains the actual value for the timer period.

CC8yPR (y = 0 - 3)
Timer Period Value **($0130_H + 0100_H * y$)** **Reset Value: 00000000_H**

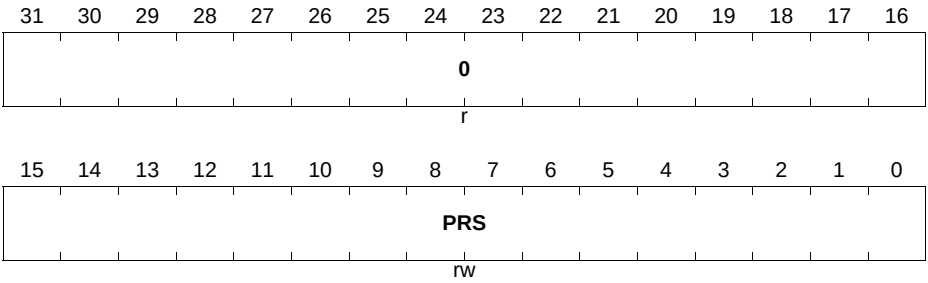


Field	Bits	Type	Description
PR	[15:0]	rh	**Period Register** Contains the value of the timer period.
0	[31:16]	r	**Reserved** A read always returns 0.

CC8yPRS

This register contains the value for the timer period that is going to be transferred into the **CC8yPR**.PR field when the next shadow transfer occurs.

CC8yPRS (y = 0 - 3)
Timer Shadow Period Value **($0134_H + 0100_H * y$)** **Reset Value: 00000000_H**

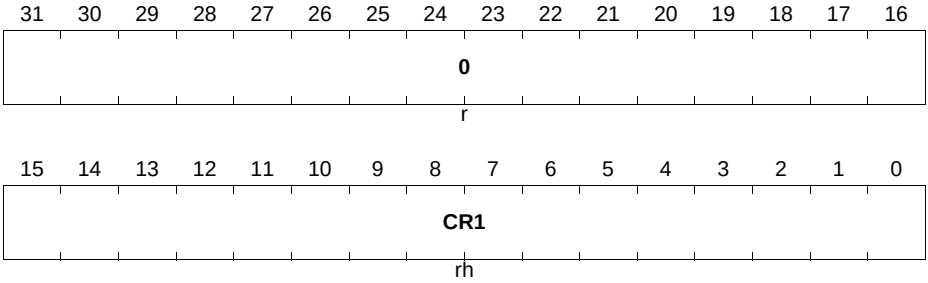


Field	Bits	Type	Description
PRS	[15:0]	rw	**Period Register** Contains the value of the timer period, that is going to be passed into the **CC8yPR**.PR field when the next shadow transfer occurs.
0	[31:16]	r	**Reserved** A read always returns 0.

CC8yCR1

This register contains the value for the timer comparison of channel 1.

CC8yCR1 (y = 0 - 3)
Channel 1 Compare Value **($0138_H + 0100_H * y$)** **Reset Value: 00000000_H**

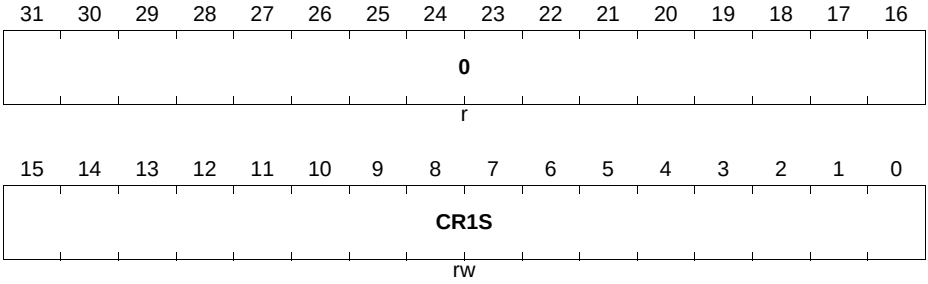


Field	Bits	Type	Description
CR1	[15:0]	rh	**Compare Register for Channel 1** Contains the value for the timer comparison. A write always addresses the shadow register, while a read returns the actual value. *Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 0 and 1, the CR is not accessible for writing. A read always returns 0.*
0	[31:16]	r	**Reserved** A read always returns 0.

CC8yCR1S

This register contains the value that is going to be loaded into the **CC8yCR1**.CR field when the next shadow transfer occurs.

CC8yCR1S (y = 0 - 3)
Channel 1 Compare Shadow Value($013C_H$ + 0100_H * y) **Reset Value: 00000000_H**

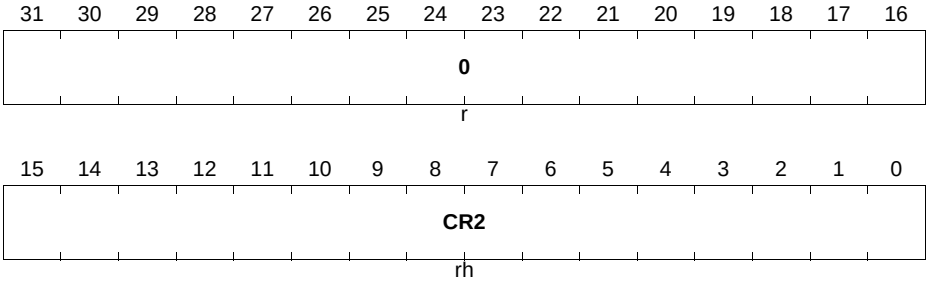


Field	Bits	Type	Description
CR1S	[15:0]	rw	**Shadow Compare Register for Channel 1** Contains the value for the timer comparison, that is going to be passed into the **CC8yCR1**.CR1 field when the next shadow transfer occurs. *Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 0 and 1, the CR is not accessible for writing. A read always returns 0.*
0	[31:16]	r	**Reserved** A read always returns 0.

CC8yCR2

This register contains the value for the timer comparison of channel 2.

CC8yCR2 (y = 0 - 3)
Channel 2 Compare Value **(0140_H + 0100_H * y)** **Reset Value: 00000000_H**

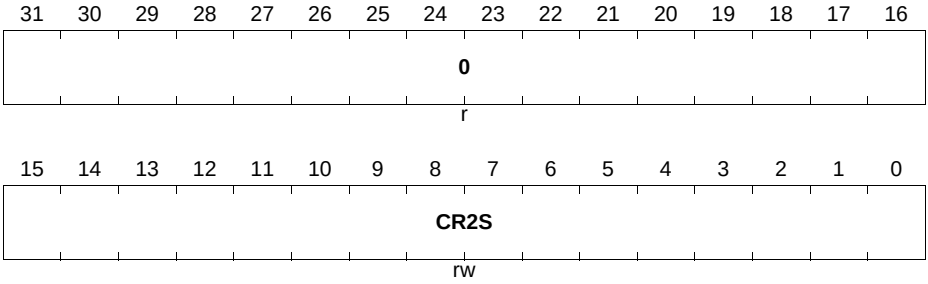


Field	Bits	Type	Description
CR2	[15:0]	rh	**Compare Register for Channel 2** Contains the value for the timer comparison. A write always addresses the shadow register, while a read returns the actual value. *Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 2 and 3, the CR is not accessible for writing. A read always returns 0.*
0	[31:16]	r	**Reserved** A read always returns 0.

CC8yCR2S

This register contains the value that is going to be loaded into the **CC8yCR2**.CR field when the next shadow transfer occurs.

CC8yCR2S (y = 0 - 3)
Channel 2 Compare Shadow Value(0144_H + 0100_H * y) **Reset Value: 00000000_H**

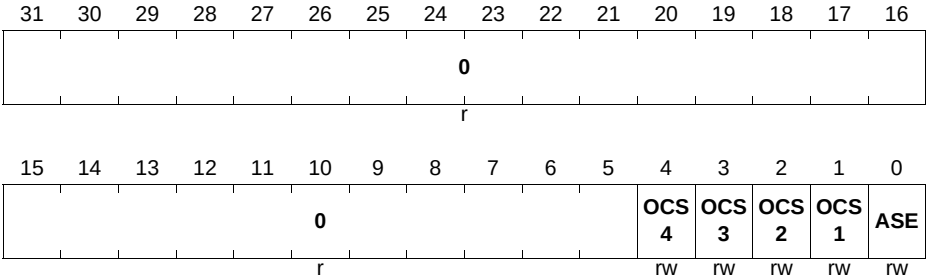


Field	Bits	Type	Description
CR2S	[15:0]	rw	**Shadow Compare Register for Channel 2** Contains the value for the timer comparison, that is going to be passed into the **CC8yCR2**.CR2 field when the next shadow transfer occurs. *Note: In Capture Mode when a external signal is selected for capturing the timer value into the capture registers 2 and 3, the CR is not accessible for writing. A read always returns 0.*
0	[31:16]	r	**Reserved** A read always returns 0.

CC8yCHC

This register contains the configuration for the output connections from the two compare channels and the enable for the asymmetric mode.

CC8yCHC (y = 0 - 3)
Channel Control **($0148_H + 0100_H * y$)** **Reset Value: 00000000_H**

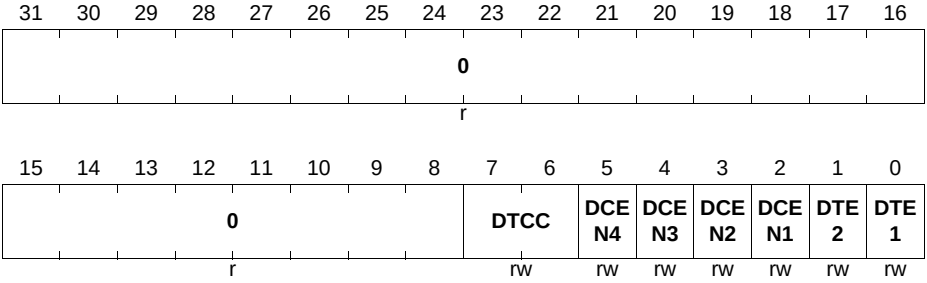


Field	Bits	Type	Description
ASE	0	rw	**Asymmetric PWM mode Enable** 0_B Asymmetric PWM is disabled 1_B Asymmetric PWM is enabled
OCS1	1	rw	**Output selector for CCU8x.OUTy0** 0_B CC8yST1 signal path is connected to the CCU8x.OUTy0 1_B Inverted CC8yST1 signal path is connected to the CCU8x.OUTy0
OCS2	2	rw	**Output selector for CCU8x.OUTy1** 0_B Inverted CC8yST1 signal path is connected to the CCU8x.OUTy1 1_B CC8yST1 signal path is connected to the CCU8x.OUTy1
OCS3	3	rw	**Output selector for CCU8x.OUTy2** 0_B CC8yST2 signal path is connected to the CCU8x.OUTy2 1_B Inverted CCST2 signal path is connected to the CCU8x.OUTy2
OCS4	4	rw	**Output selector for CCU8x.OUTy3** 0_B Inverted CC8yST2 signal path is connected to the CCU8x.OUTy3 1_B CC8yST2 signal path is connected to the CCU8x.OUTy3
0	[31:5]	r	**Reserved** A read access always returns 0

CC8yDTC

This register contains the configuration for the dead time generator.

CC8yDTC (y = 0 - 3)
Dead Time Control **($014C_H + 0100_H * y$)** **Reset Value: 00000000_H**



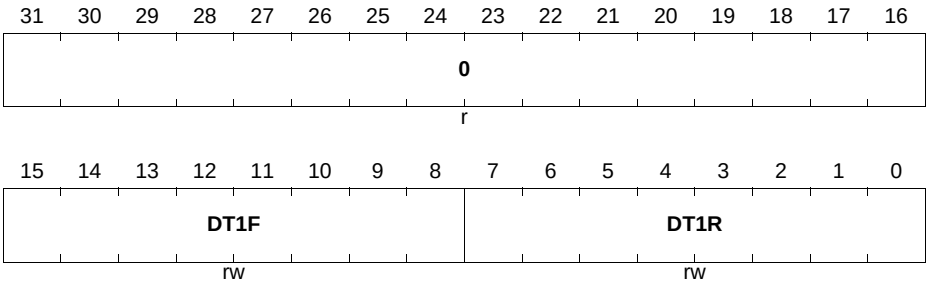
Field	Bits	Type	Description
DTE1	0	rw	**Dead Time Enable for Channel 1** This field enables the dead time counter for the compare channel 1. 0_B Dead Time for channel 1 is disabled 1_B Dead Time for channel 1 is enabled
DTE2	1	rw	**Dead Time Enable for Channel 2** This field enables the dead time counter for the compare channel 2. 0_B Dead Time for channel 2 is disabled 1_B Dead Time for channel 2 is enabled
DCEN1	2	rw	**Dead Time Enable for CC8yST1** 0_B Dead Time for CC8yST1 path is disabled 1_B Dead Time for CC8yST1 path is enabled
DCEN2	3	rw	**Dead Time Enable for inverted CC8yST1** 0_B Dead Time for inverted CC8yST1 path is disabled 1_B Dead Time for inverted CC8yST1 path is enabled
DCEN3	4	rw	**Dead Time Enable for CC8yST2** 0_B Dead Time for CC8yST2 path is disabled 1_B Dead Time for CC8yST2 path is enabled

Field	Bits	Type	Description
DCEN4	5	rw	**Dead Time Enable for inverted CC8yST2** 0_B Dead Time for inverted CC8yST2 path is disabled 1_B Dead Time for inverted CC8yST2 path is enabled
DTCC	[7:6]	rw	**Dead Time clock control** This field controls the prescaler clock configuration for the dead time counters. 00_B f_{tclk} 01_B $f_{tclk}/2$ 10_B $f_{tclk}/4$ 11_B $f_{tclk}/8$
0	[15:8], [31:16]	r	**Reserved** A read always returns 0.

CC8yDC1R

This register contains the dead time value for the rising transition.

CC8yDC1R (y = 0 - 3)
Channel 1 Dead Time Values **(0150_H + 0100_H * y)** **Reset Value: 00000000_H**



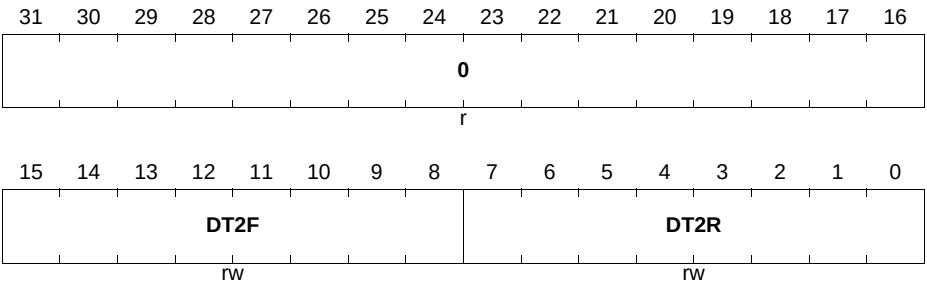
Field	Bits	Type	Description
DT1R	[7:0]	rw	**Rise Value for Dead Time of Channel 1** This field contains the delay value that is applied every time that a 0 to 1 transition occurs in the CC8yST1.

Field	Bits	Type	Description
DT1F	[15:8]	rw	**Fall Value for Dead Time of Channel 1** This field contains the delay value that is applied every time that a 1 to 0 transition occurs in the CC8yST1.
0	[31:16]	r	**Reserved** A read access always returns 0

CC8yDC2R

This register contains the dead time value for the falling transition.

CC8yDC2R (y = 0 - 3)
Channel 2 Dead Time Values **(0154_H + 0100_H * y)** **Reset Value: 00000000_H**

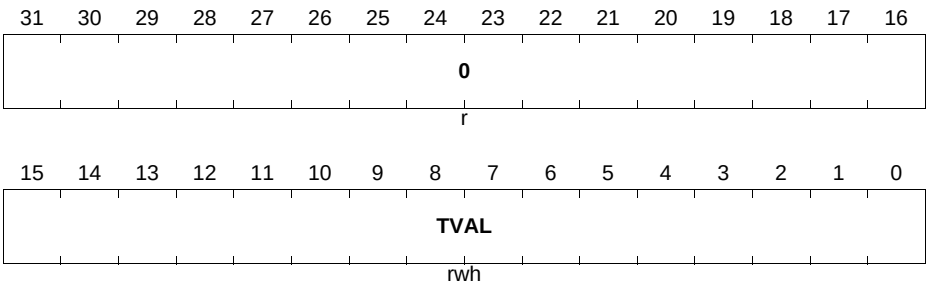


Field	Bits	Type	Description
DT2R	[7:0]	rw	**Rise Value for Dead Time of Channel 2** This field contains the delay value that is applied every time that a 0 to 1 transition occurs in the CC8yST2.
DT2F	[15:8]	rw	**Fall Value for Dead Time of Channel 2** This field contains the delay value that is applied every time that a 1 to 0 transition occurs in the CC8yST2.
0	[31:16]	r	**Reserved** A read access always returns 0

CC8yTIMER

This register contains the current value of the timer.

CC8yTIMER (y = 0 - 3)
Timer Value **($0170_H + 0100_H * y$)** **Reset Value: 00000000_H**

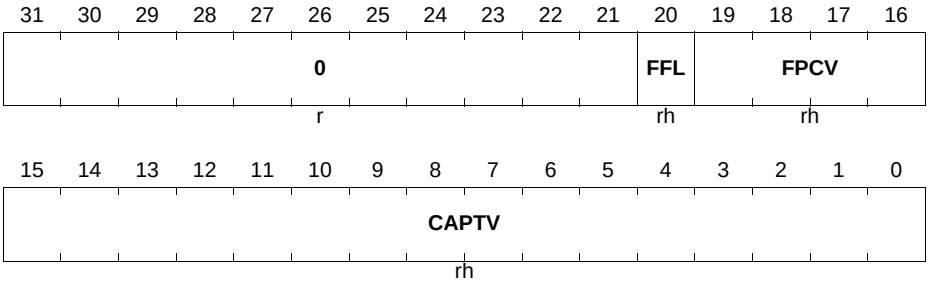


Field	Bits	Type	Description
TVAL	[15:0]	rwh	**Timer Value** This field contains the actual value of the timer. A write access is only possible when the timer is stopped.
0	[31:16]	r	**Reserved** A read access always returns 0

CC8yC0V

This register contains the values associated with the Capture 0 field.

CC8yC0V (y = 0 - 3)
Capture Register 0 **($0174_H + 0100_H * y$)** **Reset Value: 00000000_H**

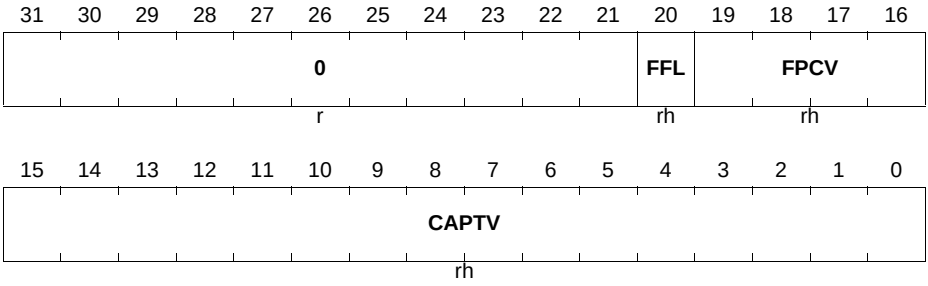


Field	Bits	Type	Description
CAPTV	[15:0]	rh	**Capture Value** This field contains the capture register 0 value. See **Figure 23-40**. In compare mode a read access always returns 0.
FPCV	[19:16]	rh	**Prescaler Value** This field contains the prescaler value when the time of the capture event into the capture register 0. In compare mode a read access always returns 0.
FFL	20	rh	**Full Flag** This bit indicates if a new value was capture into the capture register 0 after the last read access. See **Figure 23-40**. In compare mode a read access always returns 0. 0_B No new value was captured into the specific capture register 1_B A new value was captured into the specific register
0	[31:21]	r	**Reserved** A read always returns 0

CC8yC1V

This register contains the values associated with the Capture 1 field.

CC8yC1V (y = 0 - 3)
Capture Register 1 **($0178_H + 0100_H * y$)** **Reset Value: 00000000_H**

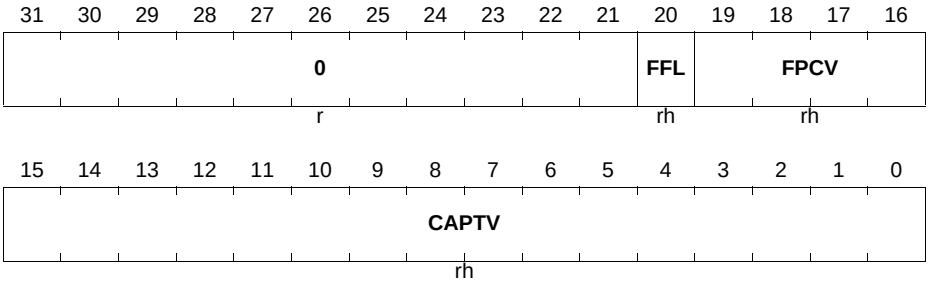


Field	Bits	Type	Description
CAPTV	[15:0]	rh	**Capture Value** This field contains the capture register 1 value. See **Figure 23-40**. In compare mode a read access always returns 0.
FPCV	[19:16]	rh	**Prescaler Value** This field contains the prescaler value when the time of the capture event into the capture register 1. In compare mode a read access always returns 0.
FFL	20	rh	**Full Flag** This bit indicates if a new value was capture into the capture register 1 after the last read access. See **Figure 23-40**. In compare mode a read access always returns 0. 0_B No new value was captured into the specific capture register 1_B A new value was captured into the specific register
0	[31:21]	r	**Reserved** A read always returns 0

CC8yC2V

This register contains the values associated with the Capture 2 field.

CC8yC2V (y = 0 - 3)
Capture Register 2 **($017C_H + 0100_H * y$)** **Reset Value: 00000000_H**

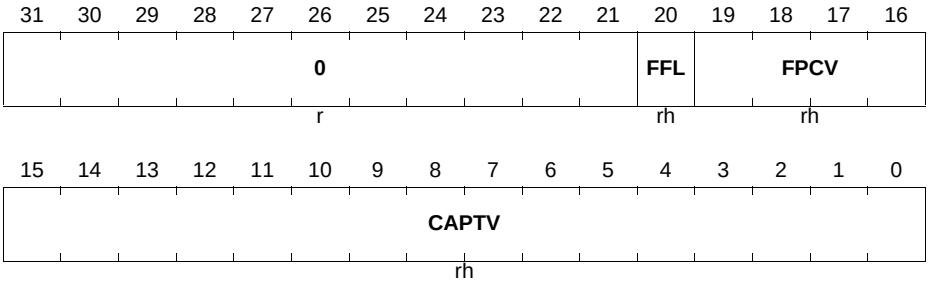


Field	Bits	Type	Description
CAPTV	[15:0]	rh	**Capture Value** This field contains the capture register 2 value. See **Figure 23-40**. In compare mode a read access always returns 0.
FPCV	[19:16]	rh	**Prescaler Value** This field contains the prescaler value when the time of the capture event into the capture register 2. In compare mode a read access always returns 0.
FFL	20	rh	**Full Flag** This bit indicates if a new value was capture into the capture register 2 after the last read access. See **Figure 23-40**. In compare mode a read access always returns 0. 0_B No new value was captured into the specific capture register 1_B A new value was captured into the specific register
0	[31:21]	r	**Reserved** A read always returns 0

CC8yC3V

This register contains the values associated with the Capture 3 field.

CC8yC3V (y = 0 - 3)
Capture Register 3 **($0180_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
CAPTV	[15:0]	rh	**Capture Value** This field contains the capture register 3 value. See **Figure 23-40**. In compare mode a read access always returns 0.
FPCV	[19:16]	rh	**Prescaler Value** This field contains the prescaler value when the time of the capture event into the capture register 3. In compare mode a read access always returns 0.
FFL	20	rh	**Full Flag** This bit indicates if a new value was capture into the capture register 3 after the last read access. See **Figure 23-40**. In compare mode a read access always returns 0. 0_B No new value was captured into the specific capture register 1_B A new value was captured into the specific register
0	[31:21]	r	**Reserved** A read always returns 0

CC8yINTS

This register contains the status of all interrupt sources.

CC8yINTS (y = 0 - 3)
Interrupt Status **($01A0_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				TRPF	E2AS	E1AS	E0AS	0		CMD2S	CMU2S	CMD1S	CMU1S	OMDS	PMUS
r				rh	rh	rh	rh	r		rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
PMUS	0	rh	**Period Match while Counting Up** 0_B Period match while counting up not detected 1_B Period match while counting up detected
OMDS	1	rh	**One Match while Counting Down** 0_B One match while counting down not detected 1_B One match while counting down detected
CMU1S	2	rh	**Channel 1 Compare Match while Counting Up** 0_B Compare match while counting up not detected 1_B Compare match while counting up detected
CMD1S	3	rh	**Channel 1 Compare Match while Counting Down** 0_B Compare match while counting down not detected 1_B Compare match while counting down detected
CMU2S	4	rh	**Channel 2 Compare Match while Counting Up** 0_B Compare match while counting up not detected 1_B Compare match while counting up detected
CMD2S	5	rh	**Channel 2 Compare Match while Counting Down** 0_B Compare match while counting down not detected 1_B Compare match while counting down detected
E0AS	8	rh	**Event 0 Detection Status** Depending on the user selection on the **CC8yINS**.EV0EM, this bit can be set when a rising, falling or both transitions are detected. 0_B Event 0 not detected 1_B Event 0 detected
E1AS	9	rh	**Event 1 Detection Status** Depending on the user selection on the **CC8yINS**.EV1EM, this bit can be set when a rising, falling or both transitions are detected. 0_B Event 1 not detected 1_B Event 1 detected

Field	Bits	Type	Description
E2AS	10	rh	**Event 2 Detection Status** Depending on the user selection on the **CC8yINS**.EV1EM, this bit can be set when a rising, falling or both transitions are detected. 0_B Event 2 not detected 1_B Event 2 detected *Note: If this event is linked with the TRAP function, this field is automatically cleared when the slice exits the Trap State.*
TRPF	11	rh	**Trap Flag Status** This field contains the status of the Trap Flag.
0	[7:6], [31:12]	r	**Reserved** A read always returns 0.

CC8yINTE

Through this register it is possible to enable or disable the specific interrupt source(s).

CC8yINTE (y = 0 - 3)
Interrupt Enable Control **($01A4_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					E2AE	E1AE	E0AE	0		CMD2E	CMU2E	CMD1E	CMU1E	OME	PME
r					rw	rw	rw	r		rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
PME	0	rw	**Period match while counting up enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time a period match while counting up occurs. 0_B Period Match interrupt is disabled 1_B Period Match interrupt is enabled

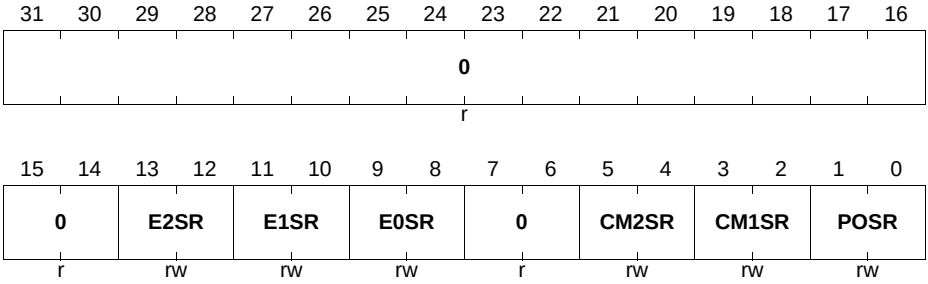
Field	Bits	Type	Description
OME	1	rw	**One match while counting down enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time an one match while counting down occurs. 0_B One Match interrupt is disabled 1_B One Match interrupt is enabled
CMU1E	2	rw	**Channel 1 Compare match while counting up enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time a compare match while counting up occurs. 0_B Compare Match while counting up interrupt is disabled 1_B Compare Match while counting up interrupt is enabled
CMD1E	3	rw	**Channel 1 Compare match while counting down enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time a compare match while counting down occurs. 0_B Compare Match while counting down interrupt is disabled 1_B Compare Match while counting down interrupt is enabled
CMU2E	4	rw	**Channel 2 Compare match while counting up enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time a compare match while counting up occurs. 0_B Compare Match while counting up interrupt is disabled 1_B Compare Match while counting up interrupt is enabled

Field	Bits	Type	Description
CMD2E	5	rw	**Channel 2 Compare match while counting down enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time a compare match while counting down occurs. 0_B Compare Match while counting down interrupt is disabled 1_B Compare Match while counting down interrupt is enabled
E0AE	8	rw	**Event 0 interrupt enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time that Event 0 is detected. 0_B Event 0 detection interrupt is disabled 1_B Event 0 detection interrupt is enabled
E1AE	9	rw	**Event 1 interrupt enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time that Event 1 is detected. 0_B Event 1 detection interrupt is disabled 1_B Event 1 detection interrupt is enabled
E2AE	10	rw	**Event 2 interrupt enable** Setting this bit to 1_B enables the generation of an interrupt pulse every time that Event 2 is detected. 0_B Event 2 detection interrupt is disabled 1_B Event 2 detection interrupt is enabled
0	[7:6], [31:11]	r	**Reserved** A read always returns 0

CC8ySRS

Through this register it is possible to select to which service request line each interrupt source is forwarded.

CC8ySRS (y = 0 - 3)
Service Request Selector **($01A8_H + 0100_H * y$)** **Reset Value: 00000000_H**



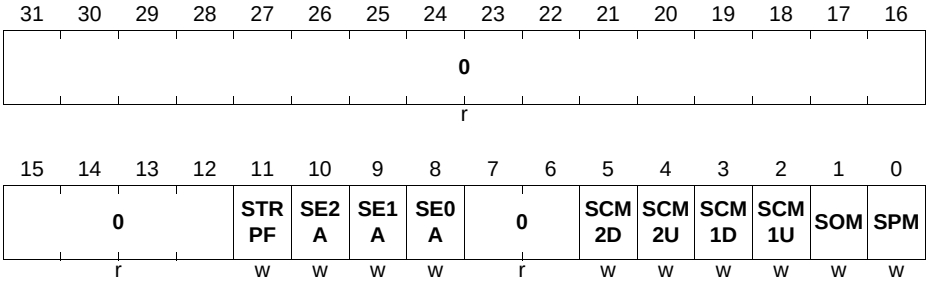
Field	Bits	Type	Description
POSR	[1:0]	rw	**Period/One match Service request selector** This field selects to which slice Service request line, the interrupt(s) generated by the Period match while counting up and One match while counting down are going to be forward. 00_B Forward to CC8ySR0 01_B Forward to CC8ySR1 10_B Forward to CC8ySR2 11_B Forward to CC8ySR3
CM1SR	[3:2]	rw	**Channel 1 Compare match Service request selector** This field selects to which slice Service request line, the interrupt(s) generated by the Compare match, of channel 1, while counting up and Compare match while counting down are going to be forward. 00_B Forward to CC8ySR0 01_B Forward to CC8ySR1 10_B Forward to CC8ySR2 11_B Forward to CC8ySR3

Field	Bits	Type	Description
CM2SR	[5:4]	rw	**Channel 2 Compare match Service request selector** This field selects to which slice Service request line, the interrupt(s) generated by the Compare match, of channel 2, while counting up and Compare match while counting down are going to be forward. 00_B Forward to CC8ySR0 01_B Forward to CC8ySR1 10_B Forward to CC8ySR2 11_B Forward to CC8ySR3
E0SR	[9:8]	rw	**Event 0 Service request selector** This field selects to which slice Service request line, the interrupt generated by the Event 0 detection are going to be forward. 00_B Forward to CCvySR0 01_B Forward to CC8ySR1 10_B Forward to CC8ySR2 11_B Forward to CC8ySR3
E1SR	[11:10]	rw	**Event 1 Service request selector** This field selects to which slice Service request line, the interrupt generated by the Event 1detection are going to be forward. 00_B Forward to CC8ySR0 01_B Forward to CC8ySR1 10_B Forward to CC8ySR2 11_B Forward to CC8ySR3
E2SR	[13:12]	rw	**Event 2 Service request selector** This field selects to which slice Service request line, the interrupt generated by the Event 2 detection are going to be forward. 00_B Forward to CC8ySR0 01_B Forward to CCvySR1 10_B Forward to CC8ySR2 11_B Forward to CC8ySR3
0	[7:6], [31:14]	r	**Reserved** Read always returns 0.

CC8ySWS

Through this register it is possible for the SW to set a specific interrupt status flag.

CC8ySWS (y = 0 - 3)
Interrupt Status Set **($01AC_H + 0100_H * y$)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
SPM	0	w	**Period match while counting up set** Writing a 1_B into this field sets the **CC8yINTS**.PMUS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SOM	1	w	**One match while counting down set** Writing a 1_B into this bit sets the **CC8yINTS**.OMDS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SCM1U	2	w	**Channel 1 Compare match while counting up set** Writing a 1_B into this field sets the **CC8yINTS**.CMU1S bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SCM1D	3	w	**Channel 1 Compare match while counting down set** Writing a 1_B into this bit sets the **CC8yINTS**.CMD1S bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SCM2U	4	w	**Compare match while counting up set** Writing a 1_B into this field sets the **CC8yINTS**.CMU2S bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.

Field	Bits	Type	Description
SCM2D	5	w	**Compare match while counting down set** Writing a 1_B into this bit sets the **CC8yINTS**.CMD2S bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SE0A	8	w	**Event 0 detection set** Writing a 1_B into this bit sets the **CC8yINTS**.E0AS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SE1A	9	w	**Event 1 detection set** Writing a 1_B into this bit sets the **CC8yINTS**.E1AS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
SE2A	10	w	**Event 2 detection set** Writing a 1_B into this bit sets the **CC8yINTS**.E2AS bit. An interrupt pulse is generated if the source is enabled. A read always returns 0.
STRPF	11	w	**Trap Flag status set** Writing a 1_B into this bit sets the **CC8yINTS**.TRPF bit. A read always returns 0.
0	[7:6], [31:12]	r	**Reserved** Read always returns 0

CC8ySWR

Through this register it is possible for the SW to clear a specific interrupt status flag.

CC8ySWR (y = 0 - 3)
Interrupt Status Clear **($01B0_H + 0100_H * y$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				RTRPF	RE2A	RE1A	RE0A	0		RCM2D	RCM2U	RCM1D	RCM1U	ROM	RPM
r				w	w	w	w	r		w	w	w	w	w	w

Field	Bits	Type	Description
RPM	0	w	**Period match while counting up clear** Writing a 1_B into this field clears the **CC8yINTS**.PMUS bit. A read always returns 0.
ROM	1	w	**One match while counting down clear** Writing a 1 into this bit clears the **CC8yINTS**.OMDS bit. A read always returns 0.
RCM1U	2	w	**Channel 1 Compare match while counting up clear** Writing a 1_B into this field clears the **CC8yINTS**.CMU1S bit. A read always returns 0.
RCM1D	3	w	**Channel 1 Compare match while counting down clear** Writing a 1_B into this bit clears the **CC8yINTS**.CMD1S bit. A read always returns 0.
RCM2U	4	w	**Channel 2 Compare match while counting up clear** Writing a 1_B into this field clears the **CC8yINTS**.CMU2S bit. A read always returns 0.
RCM2D	5	w	**Channel 2 Compare match while counting down clear** Writing a 1_B into this bit clears the **CC8yINTS**.CMD2S bit. A read always returns 0.
RE0A	8	w	**Event 0 detection clear** Writing a 1_B into this bit clears the **CC8yINTS**.E0AS bit. A read always returns 0.
RE1A	9	w	**Event 1 detection clear** Writing a 1_B into this bit clears the **CC8yINTS**.E1AS bit. A read always returns 0.
RE2A	10	w	**Event 2 detection clear** Writing a 1_B into this bit clears the **CC8yINTS**.E2AS bit. A read always returns 0.
RTRPF	11	w	**Trap Flag status clear** Writing a 1_B into this bit clears the **CC8yINTS**.TRPF bit. Not valid if **CC8yTC**.TRPEN = 1_B and the Trap State is still active. A read always returns 0.

Field	Bits	Type	Description
0	[7:6], [31:12]	r	**Reserved** Read always returns 0

CC8ySTC

Through this register it is possible to configure the extended options for the shadow transfer mechanism.

23.8 Interconnects

The tables that refer to the “global pins” are the ones that contain the inputs/outputs of each module that are common to all slices.

The GPIO mapping is available at the Ports unit.

23.8.1 CCU80 Pins

Table 23-15 CCU80 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
CCU80.MCLK	I	SCU.CCUCLK	Kernel clock
CCU80.CLKA	I	ERU1.IOUT0	another count source for the prescaler
CCU80.CLKB	I	ERU1.IOUT1	another count source for the prescaler
CCU80.CLKC	I	0	another count source for the prescaler
CCU80.MCSS	I	POSIF0.OUT6	Multi pattern sync with shadow transfer trigger
CCU80.IGBTA	I	CCU40.ST3;	Parity Checker delay finish trigger
CCU80.IGBTB	I	CCU40.SR3;	Parity Checker delay finish trigger
CCU80.IGBTC	I	CCU42.ST0;	Parity Checker delay finish trigger
CCU80.IGBTD	I	CCU42.SR0;	Parity Checker delay finish trigger

Table 23-15 CCU80 Pin Connections (cont'd)

Global Inputs/Outputs	I/O	Connected To	Description
CCU80.IGBTO	O	CCU40.IN3H; CCU42.IN0H;	Parity Checker delay start trigger
CCU80.SR0	O	NVIC; DMA; VADC.G0REQGTM; VADC.G1REQGTM; VADC.G2REQGTM; VADC.G3REQGTM; VADC.BGREQGTM;	Service request line
CCU80.SR1	O	NVIC; DMA; DAC.TRIGGER[0]; POSIF0.MSETE; VADC.G0REQGTN; VADC.G1REQGTN; VADC.G2REQGTN; VADC.G3REQGTN; U0C0.DX2F; U0C1.DX2F; VADC.BGREQGTN	Service request line
CCU80.SR2	O	NVIC; VADC.G0REQTRI; VADC.G1REQTRI; VADC.G2REQTRI; VADC.G3REQTRI; VADC.BGREQTRI;	Service request line
CCU80.SR3	O	NVIC; VADC.G0REQTRJ; VADC.G1REQTRJ; VADC.G2REQTRJ; VADC.G3REQTRJ; VADC.BGREQTRJ; CCU42.IN0G	Service request line

Table 23-16 CCU80 - CC80 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.IN0A	I	GPIO	General purpose function
CCU80.IN0B	I	GPIO	General purpose function
CCU80.IN0C	I	GPIO	General purpose function
CCU80.IN0D	I	POSIF0.OUT2	General purpose function
CCU80.IN0E	I	POSIF0.OUT5	General purpose function
CCU80.IN0F	I	VADC.G0SR3	General purpose function
CCU80.IN0G	I	ERU1.IOUT0	General purpose function
CCU80.IN0H	I	SCU.GLCCST80	General purpose function
CCU80.IN0I	I	VADC.G0BFL0	General purpose function
CCU80.IN0J	I	ERU1.PDOUT0	General purpose function
CCU80.IN0K	I	CCU40.SR3	General purpose function
CCU80.IN0L	I	CCU81.SR3	General purpose function
CCU80.IN0M	I	CCU80.ST0	General purpose function
CCU80.IN0N	I	CCU80.ST1	General purpose function
CCU80.IN0O	I	CCU80.ST2	General purpose function
CCU80.IN0P	I	CCU80.ST3	General purpose function
CCU80.MCI00	I	POSIF0.MOUT[0]	Multi Channel pattern input for CCST1
CCU80.MCI01	I	POSIF0.MOUT[1]	Multi Channel pattern input for NOT(CCST1)
CCU80.MCI02	I	POSIF0.MOUT[2]	Multi Channel pattern input for CCST2
CCU80.MCI03	I	POSIF0.MOUT[3]	Multi Channel pattern input for NOT(CCST2)
CCU80.OUT00	O	GPIO	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU80.OUT01	O	GPIO	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path

Table 23-16 CCU80 - CC80 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU80.OUT02	O	GPIO	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU80.OUT03	O	GPIO	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU80.GP00	O	NOT CONNECTED	Selected signal for event 0
CCU80.GP01	O	NOT CONNECTED	Selected signal for event 1
CCU80.GP02	O	NOT CONNECTED	Selected signal for event 2
CCU80.ST0	O	ERU1.0B1	Output of the status bit multiplexer. It can be CCST1 or CCST2
CCU80.ST0A	O	NOT CONNECTED	Channel 1 status bit: CCST1
CCU80.ST0B	O	NOT CONNECTED	Channel 2 status bit: CCST2
CCU80.PS0	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 23-17 CCU80 - CC81 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.IN1A	I	GPIO	General purpose function
CCU80.IN1B	I	GPIO	General purpose function
CCU80.IN1C	I	GPIO	General purpose function
CCU80.IN1D	I	POSIF0.OUT2	General purpose function
CCU80.IN1E	I	POSIF0.OUT5	General purpose function
CCU80.IN1F	I	ERU1.PDOUT1	General purpose function
CCU80.IN1G	I	ERU1.IOUT1	General purpose function
CCU80.IN1H	I	SCU.GLCCST80	General purpose function
CCU80.IN1I	I	VADC.G0BFL1	General purpose function
CCU80.IN1J	I	ERU1.PDOUT0	General purpose function
CCU80.IN1K	I	CCU41.SR3	General purpose function

Table 23-17 CCU80 - CC81 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU80.IN1L	I	CCU81.SR3	General purpose function
CCU80.IN1M	I	CCU80.ST0	General purpose function
CCU80.IN1N	I	CCU80.ST1	General purpose function
CCU80.IN1O	I	CCU80.ST2	General purpose function
CCU80.IN1P	I	CCU80.ST3	General purpose function
CCU80.MCI10	I	POSIF0.MOUT[4]	Multi Channel pattern input for CCST1
CCU80.MCI11	I	POSIF0.MOUT[5]	Multi Channel pattern input for NOT(CCST1)
CCU80.MCI12	I	POSIF0.MOUT[6]	Multi Channel pattern input for CCST2
CCU80.MCI13	I	POSIF0.MOUT[7]	Multi Channel pattern input for NOT(CCST2)
CCU80.OUT10	O	GPIO	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU80.OUT11	O	GPIO	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU80.OUT12	O	GPIO	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU80.OUT13	O	GPIO	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU80.GP10	O	NOT CONNECTED	Selected signal for event 0
CCU80.GP11	O	NOT CONNECTED	Selected signal for event 1
CCU80.GP12	O	NOT CONNECTED	Selected signal for event 2
CCU80.ST1	O	ERU1.1B1	Output of the status bit multiplexer. It can be CCST1 or CCST2
CCU80.ST1A	O	NOT CONNECTED	Channel 1 status bit: CCST1

Table 23-17 CCU80 - CC81 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU80.ST1B	O	NOT CONNECTED	Channel 2 status bit: CCST2
CCU80.PS1	O	POSIF0.MSYNCA	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 23-18 CCU80 - CC82 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.IN2A	I	GPIO	General purpose function
CCU80.IN2B	I	GPIO	General purpose function
CCU80.IN2C	I	GPIO	General purpose function
CCU80.IN2D	I	POSIF0.OUT2	General purpose function
CCU80.IN2E	I	POSIF0.OUT5	General purpose function
CCU80.IN2F	I	ERU1.PDOUT2	General purpose function
CCU80.IN2G	I	ERU1.IOUT2	General purpose function
CCU80.IN2H	I	SCU.GLCCST80	General purpose function
CCU80.IN2I	I	VADC.G0BFL2	General purpose function
CCU80.IN2J	I	ERU1.PDOUT0	General purpose function
CCU80.IN2K	I	CCU42.SR3	General purpose function
CCU80.IN2L	I	CCU81.SR3	General purpose function
CCU80.IN2M	I	CCU80.ST0	General purpose function
CCU80.IN2N	I	CCU80.ST1	General purpose function
CCU80.IN2O	I	CCU80.ST2	General purpose function
CCU80.IN2P	I	CCU80.ST3	General purpose function
CCU80.MCI20	I	POSIF0.MOUT[8]	Multi Channel pattern input for CCST1
CCU80.MCI21	I	POSIF0.MOUT[9]	Multi Channel pattern input for NOT(CCST1)
CCU80.MCI22	I	POSIF0.MOUT[10]	Multi Channel pattern input for CCST2

Table 23-18 CCU80 - CC82 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU80.MCI23	I	POSIF0.MOUT[11]	Multi Channel pattern input for NOT(CCST2)
CCU80.OUT20	O	GPIO	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU80.OUT21	O	GPIO	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU80.OUT22	O	GPIO	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU80.OUT23	O	GPIO	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU80.GP20	O	NOT CONNECTED	Selected signal for event 0
CCU80.GP21	O	NOT CONNECTED	Selected signal for event 1
CCU80.GP22	O	NOT CONNECTED	Selected signal for event 2
CCU80.ST2	O	ERU1.2B1	Output of the status bit multiplexer. It can be CCST1 or CCST2
CCU80.ST2A	O	NOT CONNECTED	Channel 1 status bit: CCST1
CCU80.ST2B	O	NOT CONNECTED	Channel 2 status bit: CCST2
CCU80.PS2	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 23-19 CCU80 - CC83 Pin Connections

Input/Output	I/O	Connected To	Description
CCU80.IN3A	I	GPIO	General purpose function
CCU80.IN3B	I	GPIO	General purpose function
CCU80.IN3C	I	GPIO	General purpose function
CCU80.IN3D	I	POSIF0.OUT2	General purpose function

Table 23-19 CCU80 - CC83 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU80.IN3E	I	POSIF0.OUT5	General purpose function
CCU80.IN3F	I	ERU1.PDOUT3	General purpose function
CCU80.IN3G	I	ERU1.IOUT3	General purpose function
CCU80.IN3H	I	SCU.GLCCST80	General purpose function
CCU80.IN3I	I	VADC.G0BFL3	General purpose function
CCU80.IN3J	I	ERU1.PDOUT0	General purpose function
CCU80.IN3K	I	CCU43.SR3	General purpose function
CCU80.IN3L	I	CCU81.SR3	General purpose function
CCU80.IN3M	I	CCU80.ST0	General purpose function
CCU80.IN3N	I	CCU80.ST1	General purpose function
CCU80.IN3O	I	CCU80.ST2	General purpose function
CCU80.IN3P	I	CCU80.ST3	General purpose function
CCU80.MCI30	I	POSIF0.MOUT[12]	Multi Channel pattern input for CCST1
CCU80.MCI31	I	POSIF0.MOUT[13]	Multi Channel pattern input for NOT(CCST1)
CCU80.MCI32	I	POSIF0.MOUT[14]	Multi Channel pattern input for CCST2
CCU80.MCI33	I	POSIF0.MOUT[15]	Multi Channel pattern input for NOT(CCST2)
CCU80.OUT30	O	GPIO	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU80.OUT31	O	GPIO	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU80.OUT32	O	GPIO	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU80.OUT33	O	GPIO	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU80.GP30	O	NOT CONNECTED	Selected signal for event 0

Table 23-19 CCU80 - CC83 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU80.GP31	O	NOT CONNECTED	Selected signal for event 1
CCU80.GP32	O	NOT CONNECTED	Selected signal for event 2
CCU80.ST3	O	ERU1.3B1	Output of the status bit multiplexer. It can be CCST1 or CCST2
CCU80.ST3A	O	VADC.G0REQGTE; VADC.G1REQGTE; VADC.G2REQGTE; VADC.G3REQGTE; VADC.BGREQGTE;	Channel 1 status bit: CCST1
CCU80.ST3B	O	VADC.G0REQGTF; VADC.G1REQGTF; VADC.G2REQGTF; VADC.G3REQGTF; VADC.BGREQGTF;	Channel 2 status bit: CCST2
CCU80.PS3	O	POSIF0.MSYNCB	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

23.8.2 CCU81 Pins

Table 23-20 CCU81 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
CCU81.MCLK	I	SCU.CCUCLK	Kernel clock
CCU81.CLKA	I	ERU1.IOUT0	another count source for the prescaler
CCU81.CLKB	I	ERU1.IOUT1	another count source for the prescaler
CCU81.CLKC	I	0	another count source for the prescaler
CCU81.MCSS	I	POSIF1.OUT6	Multi pattern sync with shadow transfer trigger

Table 23-20 CCU81 Pin Connections (cont'd)

Global Inputs/Outputs	I/O	Connected To	Description
CCU81.IGBTA	I	CCU41.ST3;	Parity Checker delay finish trigger
CCU81.IGBTB	I	CCU41.SR3;	Parity Checker delay finish trigger
CCU81.IGBTC	I	CCU43.ST0;	Parity Checker delay finish trigger
CCU81.IGBTD	I	CCU43.SR0;	Parity Checker delay finish trigger
CCU81.IGBTO	O	CCU41.IN3H; CCU43.IN0H;	Parity Checker delay start trigger
CCU81.SR0	O	NVIC; DMA;	Service request line
CCU81.SR1	O	NVIC; DMA; POSIF1.MSETE; U1C0.DX2F; U1C1.DX2F; U2C0.DX2F; U2C1.DX2F;	Service request line
CCU81.SR2	O	NVIC; VADC.G0REQTRK; VADC.G1REQTRK; VADC.G2REQTRK; VADC.G3REQTRK; VADC.BGREQTRK;	Service request line
CCU81.SR3	O	NVIC; VADC.G0REQTRL; VADC.G1REQTRL; VADC.G2REQTRL; VADC.G3REQTRL; VADC.BGREQTRL; CCU42.IN1G;	Service request line

Table 23-21 CCU81 - CC80 Pin Connections

Input/Output	I/O	Connected To	Description
CCU81.IN0A	I	GPIO	General purpose function
CCU81.IN0B	I	GPIO	General purpose function
CCU81.IN0C	I	GPIO	General purpose function
CCU81.IN0D	I	POSIF1.OUT2	General purpose function
CCU81.IN0E	I	POSIF1.OUT5	General purpose function
CCU81.IN0F	I	ERU1.PDOUT0	General purpose function
CCU81.IN0G	I	ERU1.IOUT0	General purpose function
CCU81.IN0H	I	SCU.GLCCST81	General purpose function
CCU81.IN0I	I	ERU1.PDOUT1	General purpose function
CCU81.IN0J	I	VADC.G0SR3	General purpose function
CCU81.IN0K	I	CCU40.SR3	General purpose function
CCU81.IN0L	I	CCU80.SR3	General purpose function
CCU81.IN0M	I	CCU81.ST0	General purpose function
CCU81.IN0N	I	CCU81.ST1	General purpose function
CCU81.IN0O	I	CCU81.ST2	General purpose function
CCU81.IN0P	I	CCU81.ST3	General purpose function
CCU81.MCI00	I	POSIF1.MOUT[0]	Multi Channel pattern input for CCST1
CCU81.MCI01	I	POSIF1.MOUT[1]	Multi Channel pattern input for NOT(CCST1)
CCU81.MCI02	I	POSIF1.MOUT[2]	Multi Channel pattern input for CCST2
CCU81.MCI03	I	POSIF1.MOUT[3]	Multi Channel pattern input for NOT(CCST2)
CCU81.OUT00	O	GPIO	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU81.OUT01	O	GPIO	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path

Table 23-21 CCU81 - CC80 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU81.OUT02	O	GPIO	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU81.OUT03	O	GPIO	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU81.GP00	O	NOT CONNECTED	Selected signal for event 0
CCU81.GP01	O	NOT CONNECTED	Selected signal for event 1
CCU81.GP02	O	NOT CONNECTED	Selected signal for event 2
CCU81.ST0	O	NOT CONNECTED	Output of the status bit multiplexer. It can be CCST1 or CCST2
CCU81.ST0A	O	NOT CONNECTED	Channel 1 status bit: CCST1
CCU81.ST0B	O	NOT CONNECTED	Channel 2 status bit: CCST2
CCU81.PS0	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 23-22 CCU81 - CC81 Pin Connections

Input/Output	I/O	Connected To	Description
CCU81.IN1A	I	GPIO	General purpose function
CCU81.IN1B	I	GPIO	General purpose function
CCU81.IN1C	I	GPIO	General purpose function
CCU81.IN1D	I	POSIF1.OUT2	General purpose function
CCU81.IN1E	I	POSIF1.OUT5	General purpose function
CCU81.IN1F	I	0	General purpose function
CCU81.IN1G	I	ERU1.IOUT1	General purpose function
CCU81.IN1H	I	SCU.GLCCST81	General purpose function
CCU81.IN1I	I	ERU1.PDOUT1	General purpose function
CCU81.IN1J	I	VADC.G1SR3	General purpose function
CCU81.IN1K	I	CCU41.SR3	General purpose function

Table 23-22 CCU81 - CC81 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU81.IN1L	I	CCU80.SR3	General purpose function
CCU81.IN1M	I	CCU81.ST0	General purpose function
CCU81.IN1N	I	CCU81.ST1	General purpose function
CCU81.IN1O	I	CCU81.ST2	General purpose function
CCU81.IN1P	I	CCU81.ST3	General purpose function
CCU81.MCI10	I	POSIF1.MOUT[4]	Multi Channel pattern input for CCST1
CCU81.MCI11	I	POSIF1.MOUT[5]	Multi Channel pattern input for NOT(CCST1)
CCU81.MCI12	I	POSIF1.MOUT[6]	Multi Channel pattern input for CCST2
CCU81.MCI13	I	POSIF1.MOUT[7]	Multi Channel pattern input for NOT(CCST2)
CCU81.OUT10	O	GPIO	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU81.OUT11	O	GPIO	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU81.OUT12	O	GPIO	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU81.OUT13	O	GPIO	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU81.GP10	O	NOT CONNECTED	Selected signal for event 0
CCU81.GP11	O	NOT CONNECTED	Selected signal for event 1
CCU81.GP12	O	NOT CONNECTED	Selected signal for event 2
CCU81.ST1	O	NOT CONNECTED	Output of the status bit multiplexer. It can be CCST1 or CCST2
CCU81.ST1A	O	NOT CONNECTED	Channel 1 status bit: CCST1

Table 23-22 CCU81 - CC81 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU81.ST1B	O	NOT CONNECTED	Channel 2 status bit: CCST2
CCU81.PS1	O	POSIF1.MSYNCA	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 23-23 CCU81 - CC82 Pin Connections

Input/Output	I/O	Connected To	Description
CCU81.IN2A	I	GPIO	General purpose function
CCU81.IN2B	I	GPIO	General purpose function
CCU81.IN2C	I	GPIO	General purpose function
CCU81.IN2D	I	POSIF1.OUT2	General purpose function
CCU81.IN2E	I	POSIF1.OUT5	General purpose function
CCU81.IN2F	I	ERU1.PDOUT2	General purpose function
CCU81.IN2G	I	ERU1.IOUT2	General purpose function
CCU81.IN2H	I	SCU.GLCCST81	General purpose function
CCU81.IN2I	I	ERU1.PDOUT1	General purpose function
CCU81.IN2J	I	VADC.G2SR3	General purpose function
CCU81.IN2K	I	CCU42.SR3	General purpose function
CCU81.IN2L	I	CCU80.SR3	General purpose function
CCU81.IN2M	I	CCU81.ST0	General purpose function
CCU81.IN2N	I	CCU81.ST1	General purpose function
CCU81.IN2O	I	CCU81.ST2	General purpose function
CCU81.IN2P	I	CCU81.ST3	General purpose function
CCU81.MCI20	I	POSIF1.MOUT[8]	Multi Channel pattern input for CCST1
CCU81.MCI21	I	POSIF1.MOUT[9]	Multi Channel pattern input for NOT(CCST1)
CCU81.MCI22	I	POSIF1.MOUT[10]	Multi Channel pattern input for CCST2

Table 23-23 CCU81 - CC82 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU81.MCI23	I	POSIF1.MOUT[11]	Multi Channel pattern input for NOT(CCST2)
CCU81.OUT20	O	GPIO	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU81.OUT21	O	GPIO	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU81.OUT22	O	GPIO	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU81.OUT23	O	GPIO	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU81.GP20	O	NOT CONNECTED	Selected signal for event 0
CCU81.GP21	O	NOT CONNECTED	Selected signal for event 1
CCU81.GP22	O	NOT CONNECTED	Selected signal for event 2
CCU81.ST2	O	NOT CONNECTED	Output of the status bit multiplexer. It can be CCST1 or CCST2
CCU81.ST2A	O	NOT CONNECTED	Channel 1 status bit: CCST1
CCU81.ST2B	O	NOT CONNECTED	Channel 2 status bit: CCST2
CCU81.PS2	O	NOT CONNECTED	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

Table 23-24 CCU81 - CC83 Pin Connections

Input/Output	I/O	Connected To	Description
CCU81.IN3A	I	GPIO	General purpose function
CCU81.IN3B	I	GPIO	General purpose function
CCU81.IN3C	I	GPIO	General purpose function
CCU81.IN3D	I	POSIF1.OUT2	General purpose function

Table 23-24 CCU81 - CC83 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU81.IN3E	I	POSIF1.OUT5	General purpose function
CCU81.IN3F	I	ERU1.PDOUT3	General purpose function
CCU81.IN3G	I	ERU1.IOUT3	General purpose function
CCU81.IN3H	I	SCU.GLCCST81	General purpose function
CCU81.IN3I	I	ERU1.PDOUT1	General purpose function
CCU81.IN3J	I	VADC.G3SR3	General purpose function
CCU81.IN3K	I	CCU43.SR3	General purpose function
CCU81.IN3L	I	CCU80.SR3	General purpose function
CCU81.IN3M	I	CCU81.ST0	General purpose function
CCU81.IN3N	I	CCU81.ST1	General purpose function
CCU81.IN3O	I	CCU81.ST2	General purpose function
CCU81.IN3P	I	CCU81.ST3	General purpose function
CCU81.MCI30	I	POSIF1.MOUT[12]	Multi Channel pattern input for CCST1
CCU81.MCI31	I	POSIF1.MOUT[13]	Multi Channel pattern input for NOT(CCST1)
CCU81.MCI32	I	POSIF1.MOUT[14]	Multi Channel pattern input for CCST2
CCU81.MCI33	I	POSIF1.MOUT[15]	Multi Channel pattern input for NOT(CCST2)
CCU81.OUT30	O	GPIO	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU81.OUT31	O	GPIO	Slice compare output from channel 1. Can be the CCST1 or NOT(CCST1) path
CCU81.OUT32	O	GPIO	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU81.OUT33	O	GPIO	Slice compare output from channel 2. Can be the CCST2 or NOT(CCST2) path
CCU81.GP30	O	NOT CONNECTED	Selected signal for event 0

Table 23-24 CCU81 - CC83 Pin Connections (cont'd)

Input/Output	I/O	Connected To	Description
CCU81.GP31	O	NOT CONNECTED	Selected signal for event 1
CCU81.GP32	O	NOT CONNECTED	Selected signal for event 2
CCU81.ST3	O	NOT CONNECTED	Output of the status bit multiplexer. It can be CCST1 or CCST2
CCU81.ST3A	O	VADC.G0REQGTG; VADC.G1REQGTG; VADC.G2REQGTG; VADC.G3REQGTG; VADC.BGREQGTG; ERU1.OGU22;	Channel 1 status bit: CCST1
CCU81.ST3B	O	VADC.G0REQGTH; VADC.G1REQGTH; VADC.G2REQGTH; VADC.G3REQGTH; VADC.BGREQGTH; ERU1.OGU32;	Channel 2 status bit: CCST2
CCU81.PS3	O	POSIF1.MSYNCB	Multi channel pattern sync trigger: PM when counting UP (edge aligned) or OM when counting DOWN (center aligned)

24 Position Interface Unit (POSIF)

The POSIF unit is a flexible and powerful component for motor control systems that use Rotary Encoders or Hall Sensors as feedback loop. The several configuration schemes of the module, target a very large universe of motor control application requirements. This enables the build of simple and complex control feedback loops, for industrial and automotive motor applications, targeting high performance motion and position monitoring.

Table 24-1 Abreviations table

PWM	Pulse Width Modulation
POSIFx	Position Interface module instance x
CCU8x	Capture/Compare Unit 8 module instance x
CCU4x	Capture/Compare Unit 4 module instance x
CC8y	Capture/Compare Unit 8 Timer Slice instance y
CC4y	Capture/Compare Unit 4 Timer Slice instance y
SCU	System Control Unit
f_{posif}	POSIF module clock frequency

Note: A small “y” or “x” letter in a register indicates an index

24.1 Overview

The POSIF module is comprised of three major sub units, the Quadrature Decoder unit, the Hall Sensor Control unit and the Multi-Channel Mode unit.

The Quadrature Decoder Unit is used for position control linked with a rotary incremental encoder.

The Hall Sensor Control Unit is used for direct control of brushless DC motors.

The Multi-Channel Mode unit is used in conjunction with the Hall Sensor mode to output the wanted motor control pattern but also can be used in a stand-alone manner to perform a simple multi-channel control of several control units.

The POSIF module is used in conjunction with a CCU4 or CCU8 which enables a very flexible resource arrangement and optimization for any type of application.

24.1.1 Features

POSIF module features

The POSIF is built of three dedicated control units that can operate in a stand-alone manner.

The Quadrature Decoder Unit contains an output interface that enables position and velocity measurements when linked with a CCU4 module. The Hall Sensor Unit enables the control of a multi-channel pattern, that can be linked with up 8 output control sources. The Multi-Channel unit offers a complete built-in interaction with the Hall Sensor Mode and an easy stand-alone control loop.

General Features

- Quadrature Decoder
  - interface for position measurement
  - interface for motor revolution measurement
  - interface for velocity measurement
  - interrupt sources for phase error, motor revolution, direction change and error on phase decoding
- Hall Sensor Mode
  - Simple build-in mode for brushless DC motor control
  - Shadow register for the multi-channel pattern
  - Complete synchronization with the PWM signals and the multi-channel pattern update
  - interrupt sources for Correct Hall Event detection, Wrong Hall Event Detection
- Multi-Channel Mode
  - Simple usage with Hall Sensor Mode
  - stand-alone Multi-Channel mode
  - Shadow register for the multi-channel pattern

Additional features

- Quadrature Decoder mode can be used in parallel with the stand-alone Multi-Channel mode
- Several profiles (via CCU4) to perform position and velocity measurement for the Quadrature Decoder mode
- Programmable delay times (via CCU4) for input pattern evaluation and new pattern value for the Hall Sensor Mode

POSIF features vs. applications

On **Table 24-2** a summary of the major features of the POSIF unit mapped with the most common applications.

Table 24-2 Applications summary

Feature	Applications
Quadrature Decoder Mode	Easy plug-in for rotary encoders: • with or without index/top marker signal • gear-slip or shaft winding compensation • separate outputs for position, velocity and revolution control - matching different system requirements • extended profile for position tracking - with revolution measurement and multiple position triggers for each revolution • support for high dynamic speed changes due to tick-to-tick and tick-to-sync capturing method
Hall Sensor Mode	Easy plug-in for Motor control using Hall Sensors: • 2 or 3 Hall sensor topologies • extended input filtering to avoid unwanted pattern switch due to noisy input signals • synchronization with the PWM signals of the Capture/Compare Unit • Active freewheeling/synchronous rectification with dead time support (link with Capture/Compare Unit 8) • easy velocity measurement function by using a Capture/Compare unit Timer Slice
Multi-Channel Mode	Modulating multiple PWM signals: • parallel modulation controlled via SW for N PWM signals - for systems with multiple power converters • generating proprietary PWM modulations • parallel and synchronous shut down of N PWM signals due to system feedback

24.1.2 Block Diagram

Each POSIF module can operate in three different modes, Quadrature Decoder, Hall Sensor and stand-alone Multi-channel Mode. To complete the control/measurement loop of all these three modes, the POSIF needs to be linked with a CCU4/CCU8 module.

In the case of the Quadrature Decoder mode, one CCU4 unit is needed, for the Hall Sensor Mode, one needs one slice of one CCU4 and a CCU8 unit. The connectivity between the stand-alone Multi-Channel mode and CCU4/CCU8 module(s) does not follow any usage constraints.

Each POSIF module contains 30 inputs and 25 outputs (including service requests), **Figure 24-1**, that are going to be mapped to available functions of the module, depending in which configuration it was selected by the user: Quadrature Decoder, Hall Sensor or stand-alone Multi-Channel.

The module also has two dedicated Service request Lines, see **Section 24.3**.

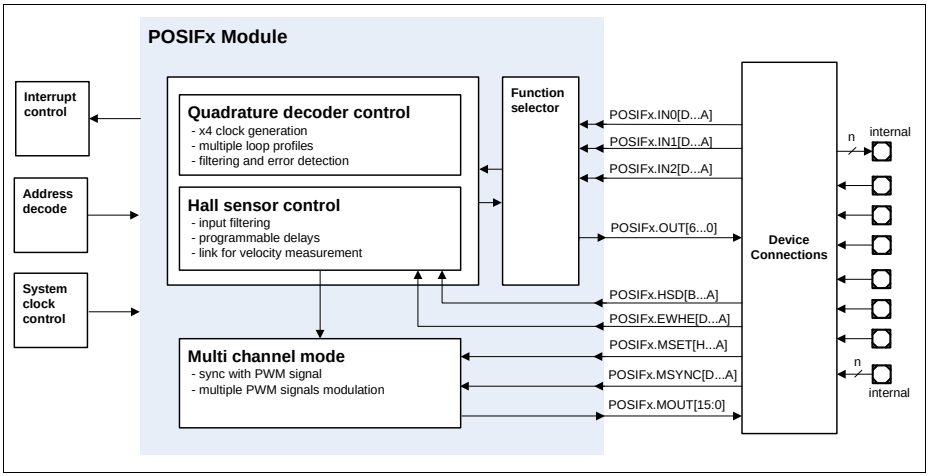


Figure 24-1 POSIF block diagram

24.2 Functional Description

24.2.1 Overview

The POSIF module contains a function selector unit, that is used in parallel by both Quadrature Decoder and Hall Sensor Control units. This block selects which input signals should be decoded for each control unit. The function selector is also decoding the outputs coming from these two modes (Quadrature and Hall Sensor Mode).

The POSIF module also contains a subset of inputs that are only used in Hall Sensor/Multi-Channel Mode. These inputs are connected to a timer structure (CCU4/CCU8) and are used to control the delays between pattern sampling, multi-channel pattern update and synchronization, etc.

The Multi-Channel unit contains 16 dedicated outputs, that contain the current multi-channel pattern and that can be connected to a CCU8/CCU4.

The POSIF control unit contain a dedicated Run bit, that can be set/clear by SW. It can also generate a Sync start signal that can be connected to the timer units (CCU4/CCU8).

For the Quadrature Decoder Mode, there is the possibility to define which of the signals is the leading phase and also the active level for each one.

The Quadrature Decoder Mode can also receive the clock and direction signals directly from an external source.

The Multi-Channel offers a shadow register, for the Multi-Channel pattern, enabling this way an update on the fly of these parameter. A write access always addresses the shadow register while a read, always returns the actual value of the Multi-Channel pattern.

Table 24-3 POSIF slice pin description

Pin	I/O	Hall Sensor Mode	Quadrature Decoder Mode	Multi-Channel Mode (stand-alone)
POSIFx.IN0[D...A]	I	Hall Input 1	Encoder Phase A or Clock	Not used
POSIFx.IN1[D...A]	I	Hall Input 2	Encoder Phase B or Direction	Not used
POSIFx.IN2[D...A]	I	Hall Input 3	Index/Zero marker	Not used
POSIFx.HSD[B...A]	I	Hall pattern sample delay	Not used	Not used
POSIFx.EWHE[D...A]	I	Wrong hall event emulation	Not used	Wrong hall event emulation
POSIFx.MSET[H...A]	I	Multi-Channel next pattern update set	Not used	Multi-Channel next pattern update set
POSIFx.MSYNC[D...A]	I	Multi-Channel pattern update synchronization	Not used	Multi-Channel pattern update synchronization
POSIFx.OUT0	O	Hall inputs edge detection trigger	Quadrature clock	Not used

Table 24-3 POSIF slice pin description (cont'd)

Pin	I/O	Hall Sensor Mode	Quadrature Decoder Mode	Multi-Channel Mode (stand-alone)
POSIFx.OUT1	O	Hall Correct event	Direction	Not used
POSIFx.OUT2	O	Idle/ wrong hall event	Period clock	Not used
POSIFx.OUT3	O	Stop	Clear/capt	Not used
POSIFx.OUT4	O	Multi-Channel pattern update	Index	Not used
POSIFx.OUT5	O	Sync start	Sync start	Not used
POSIFx.OUT6	O	Multi Pattern sync trigger	Not used	Multi Pattern sync trigger
POSIFx.MOUT[15:0]	O	Multi-Channel pattern	Not used	Multi-Channel pattern
POSIFx.SR0	O	Service request line 0	Service request line 0	Service request line 0
POSIFx.SR1	O	Service request line 1	Service request line 1	Service request line 1

Note: The Service Request signals at the output of the kernel are extended for one more kernel clock cycle.

24.2.2 Function Selector

The Function selector maps the input function signals to the selected operating mode, wether Quadrature or Hall Sensor Mode, **Figure 24-2**. The outputs are also decoded throughout this unit.

For each function input, POSI0, POSI1 and POSI2, the user can select one of the 4 input pins via the fields **PCONF**.INSEL0, **PCONF**.INSEL1 and **PCONF**.INSEL2. It is also possible to perform a low pass filtering on the three inputs, the field **PCONF**.LPC controls the low pass filters cut frequency.

The Hall Sensor Mode is the default function, **PCONF**.FSEL = 00_B. In this mode, the Function selector maps the POSI0 to the Hall Input 1, the POSI1 to the Hall input 2 and the POSI2 to the Hall input 3.

When the Quadrature Decoder mode is selected, **PCONF**.FSEL = 01_B, POSI0 is mapped to Phase A or Clock, POSI1 to the Phase B or Direction and POSI2 to the Index signals coming from the rotary motor encoder. Notice that it is also possible to select the

Quadrature Decoder and stand-alone Multi-Channel mode, by setting **PCONF**.FSEL = 11_B.

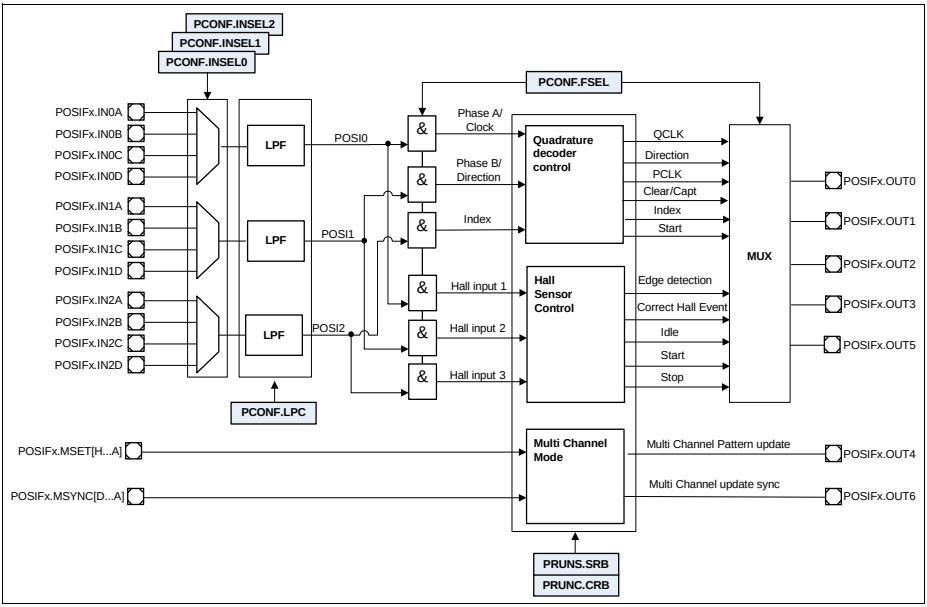


Figure 24-2 Function selector diagram

24.2.3 Hall Sensor Control

The Hall Sensor mode is divided in three major loops: detection of any update in the Hall inputs, delay between the detection and the sampling of the Hall inputs for comparison against the expected Hall Pattern and the update of the Multi-Channel pattern.

The Hall inputs are directly connected to an edge detection circuitry and with any modification in any of these three inputs, a signal is generated on the POSIFx.OUT0 pin, see **Figure 24-3**. This pin can be connected to one CCU4 slice that is controlling the delay between the edge detection and the next step on the Hall sensor mode, the sampling of the Hall Inputs.

The signal used to trigger the sample of the Hall inputs is selected via the **PCONF**.DSEL field, and this trigger can be active at the rising or falling edge (**PCONF**.SPES).

When the sampling trigger is sensed, the Hall inputs are sampled and compared against the Current Pattern, **HALP**.HCP and the Expected Hall Pattern, **HALP**.HEP (to evaluate if the input pattern match the HEP or HCP values).

The edge detection circuit generates an enable signal for sampling the hall inputs, PIFHSP and the sample logic generates a pulse every time that new values are captured, PIFHRDY, that is used inside the pattern compare logic.

When the sampled value matches the Expected Hall Pattern, a pulse is generated in the POSIFx.OUT1 pin to indicate a correct hall event. At the same time the next values programmed into the shadow registers are loaded. The **HALP**.HCP[LSB] is linked to the Hall input 1, and the **HALP**.HCP[MSB] is linked to the Hall input 3 (the same is applicable for the **HALP**.HEP register).

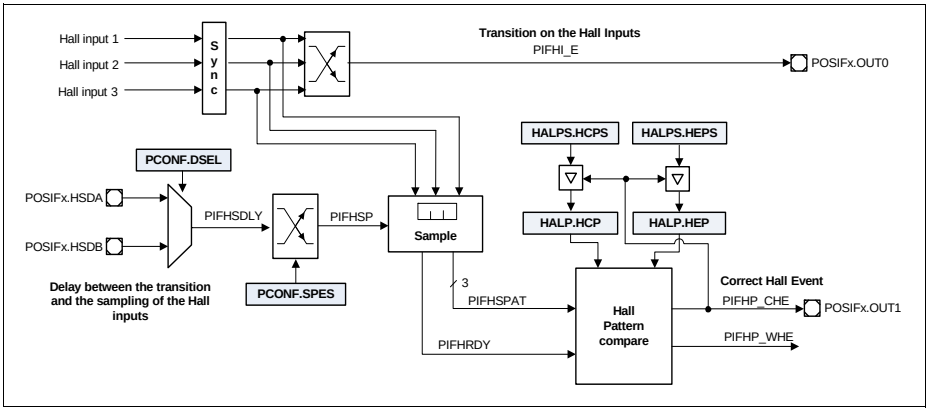


Figure 24-3 Hall Sensor Control Diagram

When the sampled value matches the Current Hall Pattern, a line glitch deemed to have occurred and no action is taken. When the sampled value does not match any of the values (the current and the expected pattern), a major error is deemed to have occurred and the Wrong Hall Event signal is generated. Every time that a sampled pattern leads to a wrong hall event or when it matches the current pattern a stop signal is generated throughout the POSIFx.OUT3 pin, **Figure 24-4**.

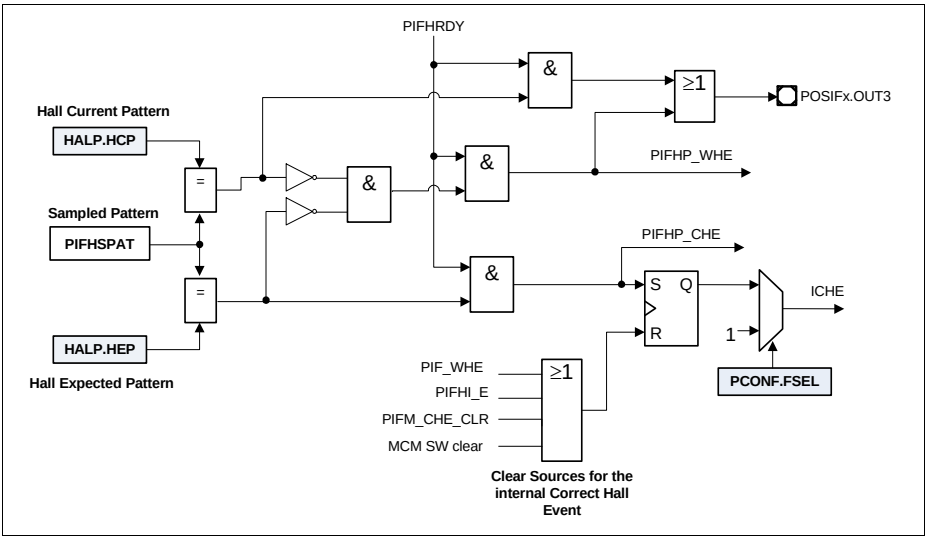


Figure 24-4 Hall Sensor Compare logic

A wrong hall event can generate an IDLE signal that is connected to the POSIFx.OUT2 and can be used to clear the run bit of the Hall Sensor Control unit.

The IDLE signal can also be connected to the PWM unit to perform a forced stop operation, **Figure 24-5**. The wrong hall event/idle function can also be controlled via a pin.

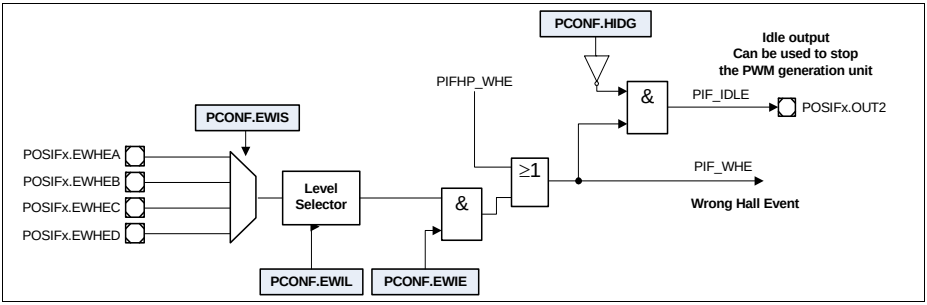


Figure 24-5 Wrong Hall Event/Idle logic

After the Correct Hall Event is detected, a delay can be generated between this detection and the update of the Multi-Channel pattern. On **Figure 24-6** it is demonstrated the control logic of the Multi-Channel mode.

The delay for the update of the Multi-Channel pattern can be controlled directly by a CCU4 slice. The trigger that indicates that the delay is finished, can be mapped to one of the input signals POSIFx.MSET[H...A] (**PCONF**.MSETS selects the input signal used for this purpose). One can also select the active edge for the trigger via **PCONF**.MSES.

The **PCONF**.MCUE field selects the source that enables an update of the Multi-Channel pattern. When set to 1_B, the Multi-Channel pattern can only be updated after the SW has written a 1_B into the **MCMS**.MNPS field.

After the update delay, the Multi-Channel pattern still needs to be synchronized with the PWM signal. For this, the user selects a signal from the POSIFx.MSYNC[D...A] inputs via **PCONF**.MSYNS field. When a falling edge is detected in this signal, then the new multi pattern is applied at the POSIFx.MOUT[15:0] outputs, with the **MCM**.MCMP[15] linked to the POSIFx.MOUT[15] and **MCM**.MCMP[0] to POSIFx.MOUT[0].

The POSIFxOUT6 pin is connected to the **MCMF**.MSS register field. This register field is enabling the Multi-Channel pattern update (that is done upon receiving the sync signal, PIFMSYNC) and can be used in conjunction with a CAPCOM module to perform a synchronous update of the Multi-Channel pattern and the compare values used inside of the CAPCOM.

When a wrong hall event is configured to set the Hall Sensor Control into IDLE, the Multi-Channel pattern is also cleared.

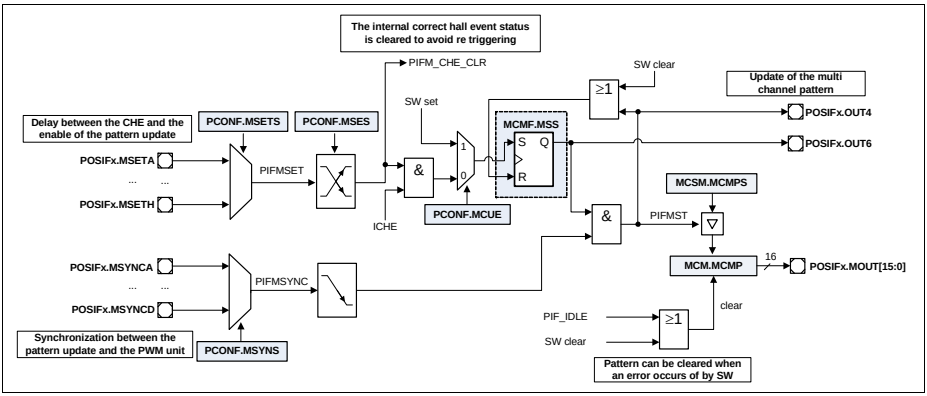


Figure 24-6 Multi-Channel Mode Diagram

Figure 24-7 shows all the previous described steps in the Hall Sensor Mode. Every time that a transition on a Hall input is detected, the pin POSIFx.OUT0 is asserted. This signal is used to start the timing delay between the transition detection and the sampling of the hall inputs.

In this scenario the status output (ST in **Figure 24-7**) of a timer cell was used to control the timing 1 (delay between the transition and the sampling of the hall inputs). With the

rising edge of the ST signal, the hall inputs are sampled and if they match the expected pattern, signal POSIFx.OUT1 is asserted.

A service request line (SR signal) mapped to the same timer cell is used to control the delay between a correct Hall Event and the update of the Multi-Channel pattern. This service request is asserted when a period match is detected.

Another timer cell is used to measure the time between each correct hall event. This timer cell is represented by the Timing 2 on **Figure 24-7** (the CR symbolizes the capture register of the timer cell).

After the delay controlled by the Timing 1 (SR signal) is over, the update of the Multi-Channel pattern needs to be synchronized with the PWM signal. This is done by using one of the pattern synchronization outputs of the Capture/Compare Unit that is used to generate the PWM signals (as an example a CCU8 was used).

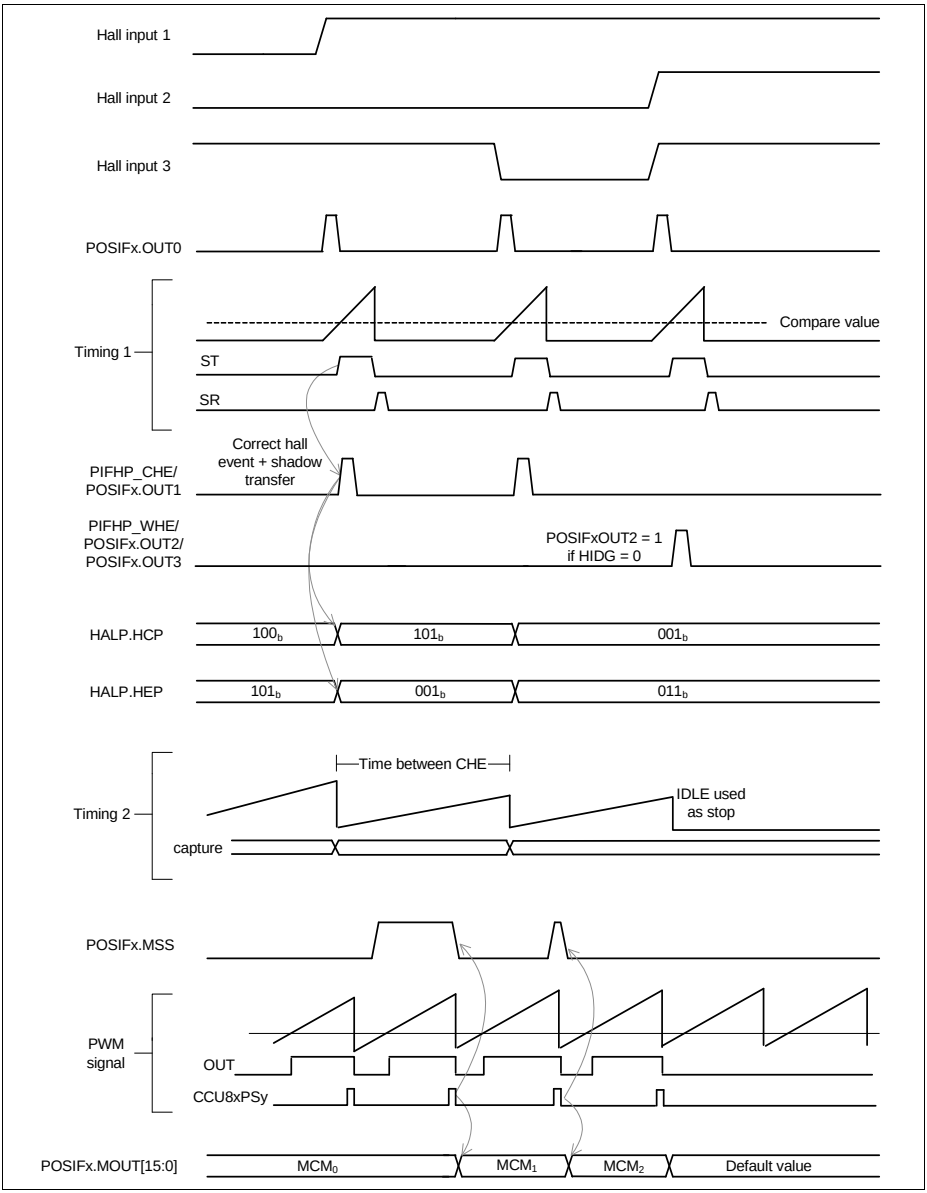


Figure 24-7 Hall Sensor timing diagram

24.2.4 Quadrature Decoder Control

The Quadrature Decoder Mode is selected by setting **PCONF**.FSEL = 01_B or **PCONF**.FSEL = 11_B (in this case the Multi-Channel mode is also enabled).

Inside the Quadrature Decoder Mode, two different subsets are available:

- standard Quadrature Mode
- Direction Count Mode

The standard mode is used when the external rotary encoder provides two phase signals and additionally and index/marker signal that is generated once per shaft revolution. The Direction Count Mode is used when the external encoder only provides a clock and a direction signal, **Figure 24-8**.

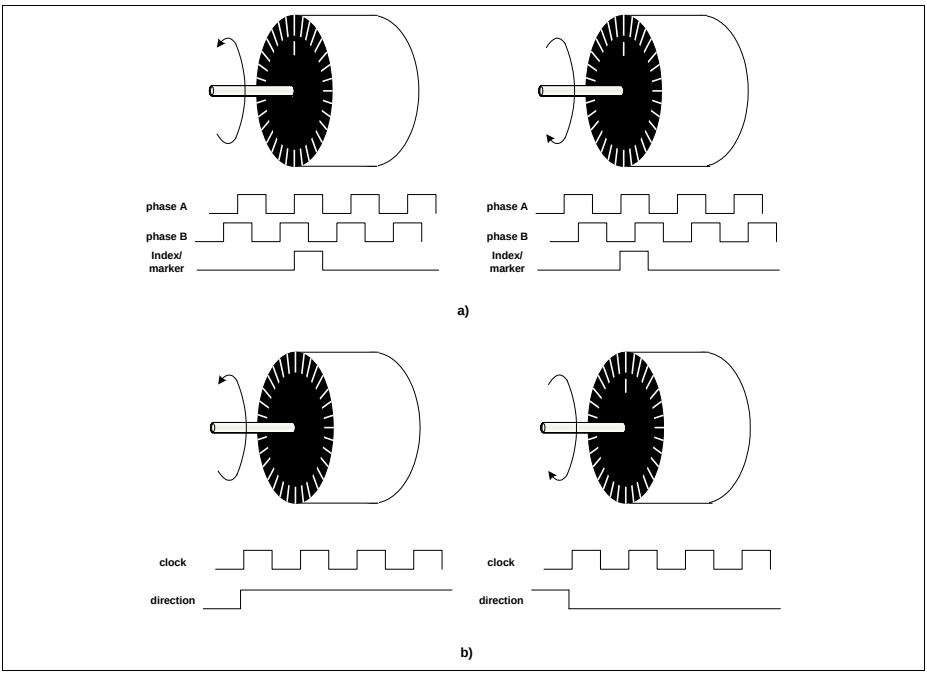


Figure 24-8 Rotary encoder types - a) standard two phase plus index signal; b) clock plus direction

Standard Quadrature Mode

The Quadrature Decoder unit offers a very flexible PhaseA/PhaseB configuration stage. Normally for a clockwise motor shaft rotation, Phase A should precede Phase B but nevertheless, the user can configure the leading phase as well the specific active state for each signal, **Figure 24-9**.

There are two major blocks that build the Quadrature Decoder Control unit: the block that decodes the quadrature clock and motor shaft direction and the block that handles the index (motor revolution) control.

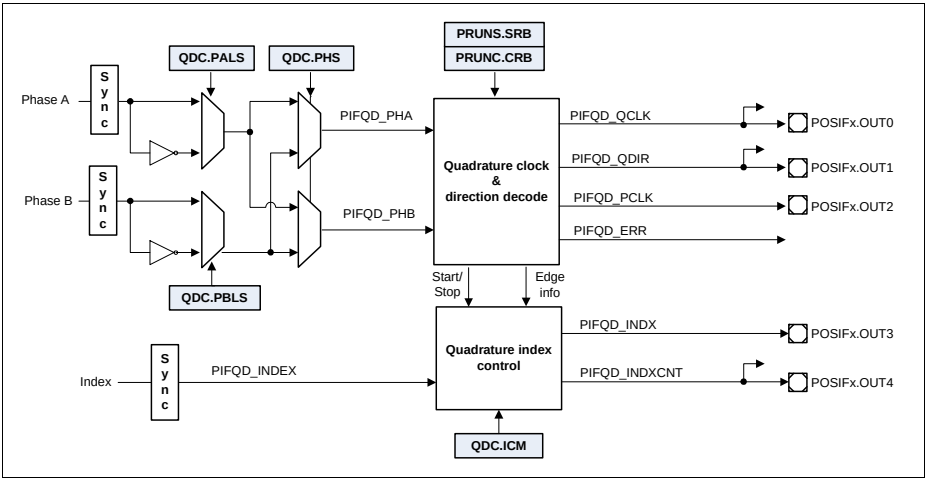


Figure 24-9 Quadrature Decoder Control Overview

The quadrature clock is connected to pin POSIFx.OUT0 and is used for position measurement. This clock is decoded from every edge of the phase signals and therefore there are 4 clock pulses per phase period.

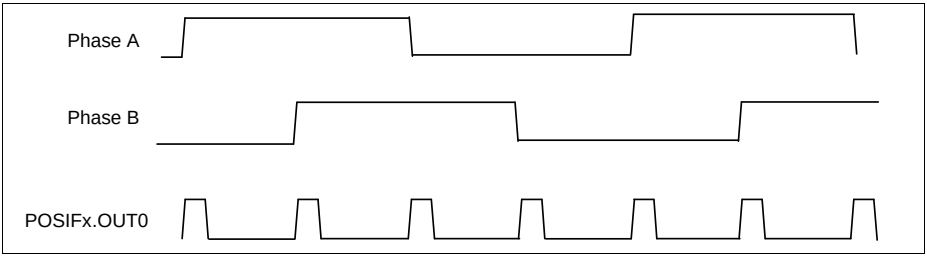


Figure 24-10 Quadrature clock generation

A period clock is also generated for velocity measurement operations.

The direction of the motor rotation is connected to the POSIFx.OUT1 pin and is asserted HIGH when the motor is rotating clockwise and LOW when it is turning in the counterclockwise direction.

The index control logic, memorizes which was the first edge after the assertion of the index signal, so the same quadrature transition is used for index event operations. It also

memorizes the direction of the first index, so that it can control when the revolution increment signal should be asserted.

An error signal, connected to a flag (and if enable an interrupt can be generated) also can be generated when a wrong phase pair is detected.

The Quadrature Decoder Control, uses the information of the current and previous phase pair to decode the direction and clocks. Both phase signals pass through a edge detection logic, from which the outputs are going to be used against the valid/invalid transitions stages, see **Figure 24-11**. There is the possibility to reset the decoder state machine (but not the flags and static configuration) by writing a 1_B into the **PRUNC**.CSM field.

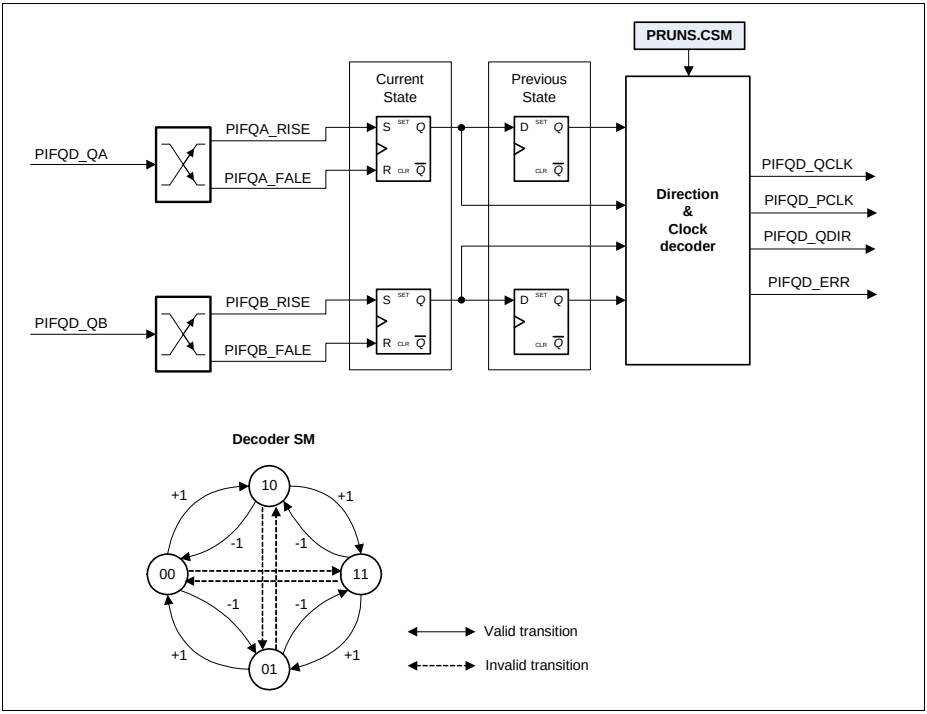


Figure 24-11 Quadrature Decoder States

Direction Count Mode

Some position encoders do not have the phase signals as outputs, instead, they provide two signals that contain the clock and direction information. This is know as the Direct Count Mode.

When using a position encoder of this type, the user should set the **PCONF**.QDCM bit field to 1_B (enabling the direction count mode). In this case, the signal selected from the POSIFx.IN0[D...A] is used as clock and the selected signal from the POSIFx.IN1[D...A] contains the direction information.

The input signals are synchronized with the POSIF module clock and sent to the respective outputs. In this case the inputs and outputs linked with the index/zero marker are not used. The POSIFx.OUT2 output is also inactive.

24.2.4.1 Quadrature Clock and Direction decoding

The Quadrature Decoder unit outputs two clocks. One clock is used for position control and therefore is generated in each edge transition of the phase signals. The second clock is used for velocity measurements and is immune to possible glitches on the phase signals that can be present at very slow rotation speeds. These glitches are not normal line noise, but are due to the slow movement of the engine.

The decoding of the direction signal is done following the rules on **Figure 24-11**. **Figure 24-12** shows all the valid transitions of Phase A and Phase B signals.

Figure 24-13 shows the difference between the two clock signals in the case that some glitches are present in the phase signals.

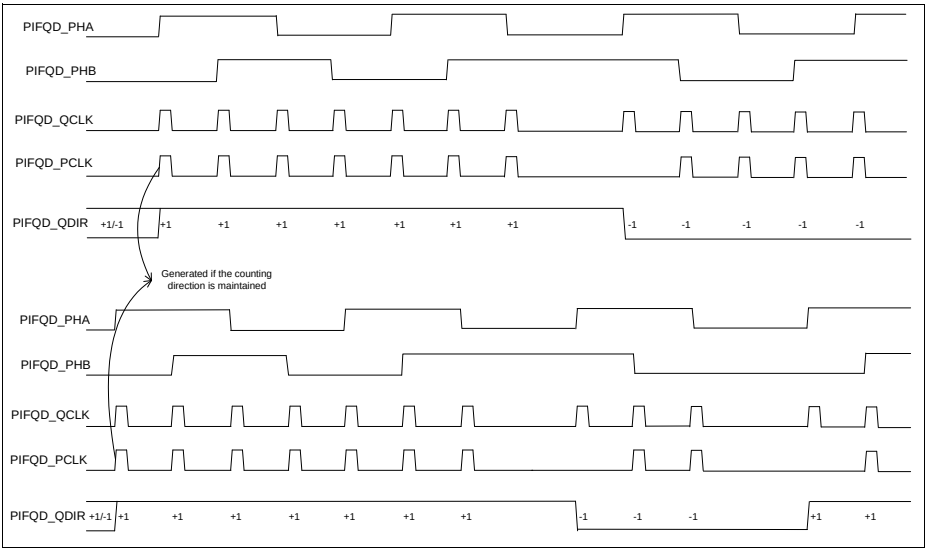


Figure 24-12 Quadrature clock and direction timings

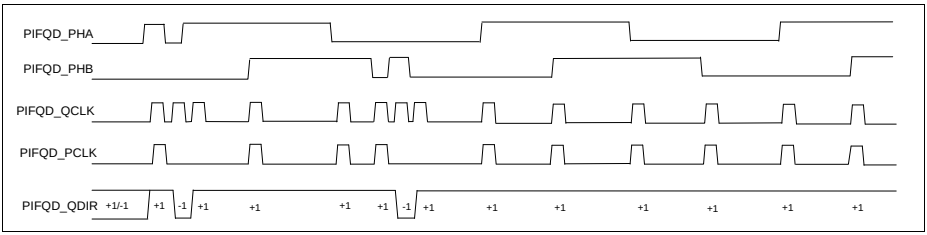


Figure 24-13 Quadrature clock with jitter

24.2.4.2 Index Control

The Index Control logic has two different outputs. One is asserted every time that an index signal is detected (and the motor shaft rotation is the same), POSIFx.OUT4, and therefore it can be used in a revolution counter. With this, the SW can monitor not only the position of the shaft but also the total number of revolutions that have occurred.

The activity of the other output, POSIFx.OUT3, can be programmed via the **QDC**.ICM field. Depending on the value set in the **QDC**.ICM, this signal can be generated:

- in every index signal occurrence
- only on the first index signal occurrence
- disabled - this outputs is never asserted

This is useful for applications that need to reset the counters on every index event or only at the first one.

The Index Control logic memorizes the immediately next phase edge that follows a index so the generated signals have always the same reference. If the first phase edge after the index is the rising edge of Phase B signal, then the index signals are going to be generated in the next index event with the rising edge of the Phase B signal if the direction is kept or with the falling edge of Phase B signal if the direction has changed.

Figure 24-14 shows the timing diagram for the generation of the index signals. In this case the **QDC**.ICM = 10_B, which means that the POSIFx.OUT3 index signal is going to be generated in every occurrence of the input index.

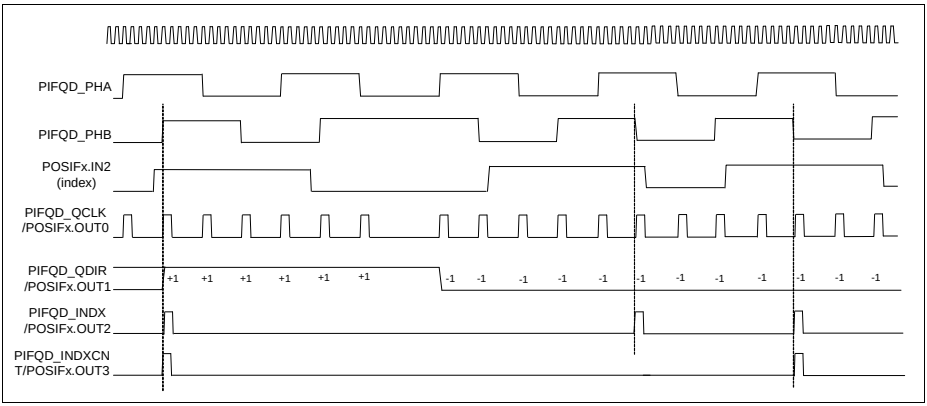


Figure 24-14 Index signals timing

24.2.5 Stand-Alone Multi-Channel Mode

The Multi-Channel mode (Multi-Channel Mode logic can be see on **Figure 24-6**) can be used without the Hall Sensor Control, by setting the **PCONF**.FSEL = 10_B or if the Quadrature Decoder Mode is also needed, by setting **PCONF**.FSEL = 11_B.

In stand-alone Multi-Channel Mode, the mechanism to update the multi-channel pattern is the same as the one described in **Section 24.2.3**.

The trigger for a pattern update can come from the SW, by writing 1_B to **MCMS**.MNPS, or it can come from an external signal like in Hall sensor Mode. This external signal should then be mapped to the PIFMSET function by selecting the appropriate value in the **PCONF**.MSETS field.

The synchronization between the update of the new Multi-Channel pattern and control signal still needs to be done. The user needs to map one input signal of the POSIFx.MSYNC[D...A] range to this function.

The SW has the possibility of clearing the actual Multi-Channel pattern, by writing 1 into the **MCMC**.MPC field.

24.2.6 Synchronous Start

The POSIF module has a synchronous start output, POSIFx.OUT5, that can be used together with the CAPCOM for a complete synchronous start of both modules. The synchronous start is linked with the run bit of the POSIF module, which means that every time that the run bit is set, a pulse is generated throughout the POSIFx.OUT5 pin.

By using the synchronous start output, the SW does not perform two independent accesses to start the POSIF and the CCU4/CCU8 and therefore is guaranteed that both modules start their operation at the exact same time.

24.2.7 Using the POSIF

The POSIF module needs to be linked with a CCU4/CCU8 module to perform the full set of functions in each of the possible modes (due to the fact that doesn't contain built-in counters/timers).

To operate the POSIF in the Quadrature Decoder Mode, one CCU4 module is needed. The Hall Sensor Mode, needs a CCU8 (at least 3 slices are need to control a brushless DC motor) and also (at least) two CCU4 or CCU8 slices. The stand-alone Multi-Channel Mode linking configuration depends heavily on the use cases and therefore the number of slices of CCU4 or CCU8 used is freely chosen by the user.

24.2.7.1 Hall Sensor Mode Usage

When using the Hall Sensor Mode of the POSIF, the Multi-Channel Mode is also working. Due to that fact, the CCU8 module used to perform the Multi-Channel Modulation, needs to be configured in Multi-Channel Mode.

Standard Hall Sensor Mode Usage

On **Figure 24-15**, the Hall Sensor Mode is used in conjunction with two CCU4 slices and one CCU8 module. The first slice of CCU4, slice 0 is being used to control the delays between the edge detection of the Hall Inputs and the actual sampling, and also to control the delay between a Correct Hall Event and the Multi-Channel Pattern update enable.

The rising edge of the CCU4x.ST0 is used as finish trigger for the first delay, while the Service request line is used for triggering the update of a new pattern after a Correct Hall Event. The service request is configured to be active on each period match hit of the specific slice.

Slice 0 is configured in single shot mode, so that the time delay can be re triggered every time that a request from the POSIF occurs.

The second slice of the CCU4 unit, Slice 1, is being used in Capture Mode, to capture the time between Correct Hall Events (storing this way the motor speed between two correct hall events). The POSIFx.OUT1 of the POSIF is used as capture trigger for the slice while the POSIFx.OUT3 is used as stop. The capture and stop triggers are configured in the specific timer slices as active on the rising edge.

The CCU8 is the module that is generating the PWM signals to control the motor and therefore, the Multi-Channel Pattern outputs POSIFx.MOUT[7:0] are linked to this unit.

To close the Multi-Channel loop, an output of the CCU8 needs to be connected to the POSIF module (one of the CCU8x.PSy signals), so that the Multi-Channel Pattern is updated synchronously with the PWM period.

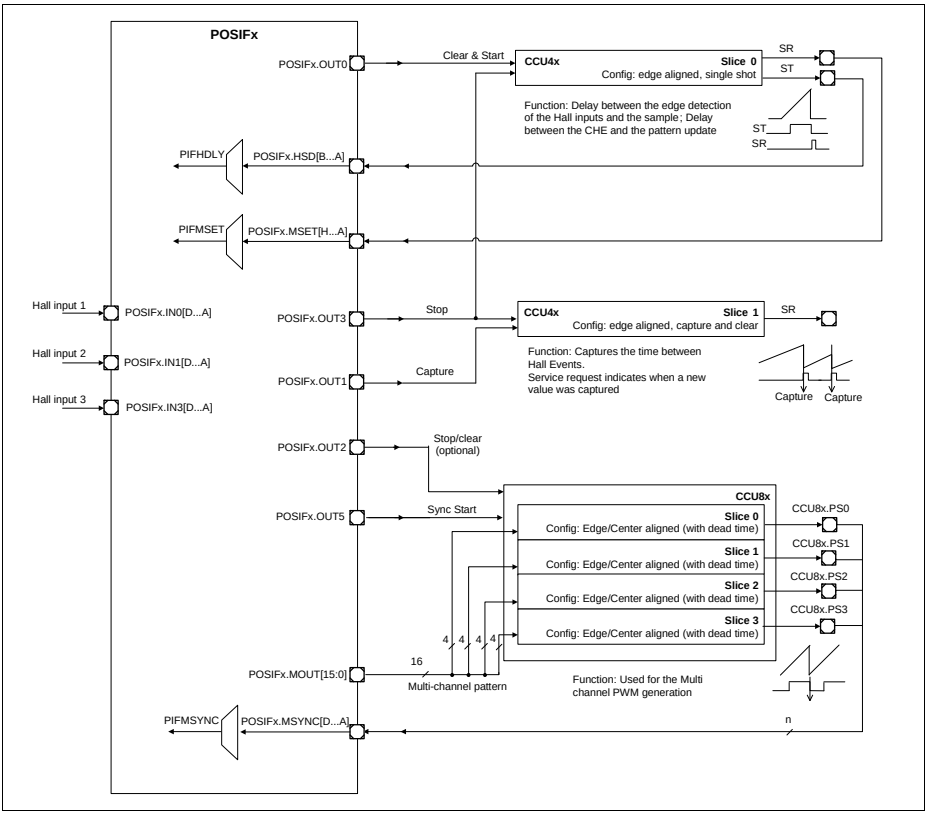


Figure 24-15 Hall Sensor Mode usage - profile 1

Hall Sensor Mode Usage - Flexible Time Control

On **Figure 24-16**, another profile is demonstrated. In this case instead of 2 CCU4 slices, the Hall Sensor Mode is using 3 CCU4 slices. This profile gives more flexibility in terms of delay configuration. It uses two timer slices to control independently the delay between the transition of the hall inputs and sampling, and the delay between a Correct Hall Event and the update of the Multi-Channel pattern. At the same time it also removes the need of using a service request to control the pattern update delay.

Slice 0 is used to control the delay between a transition at the hall inputs and the actual sampling. Slice 2 is used to control the delay between a Correct Hall Event and the update of the Multi-Channel pattern.

Slice 1 is used as keeper of the time stamp between Correct Hall events (the same function as Slice 1 in profile 1).

The synchronism between the PWM signal and the update of the Multi-Channel pattern is again done with one of the CCU8x.PSy outputs.

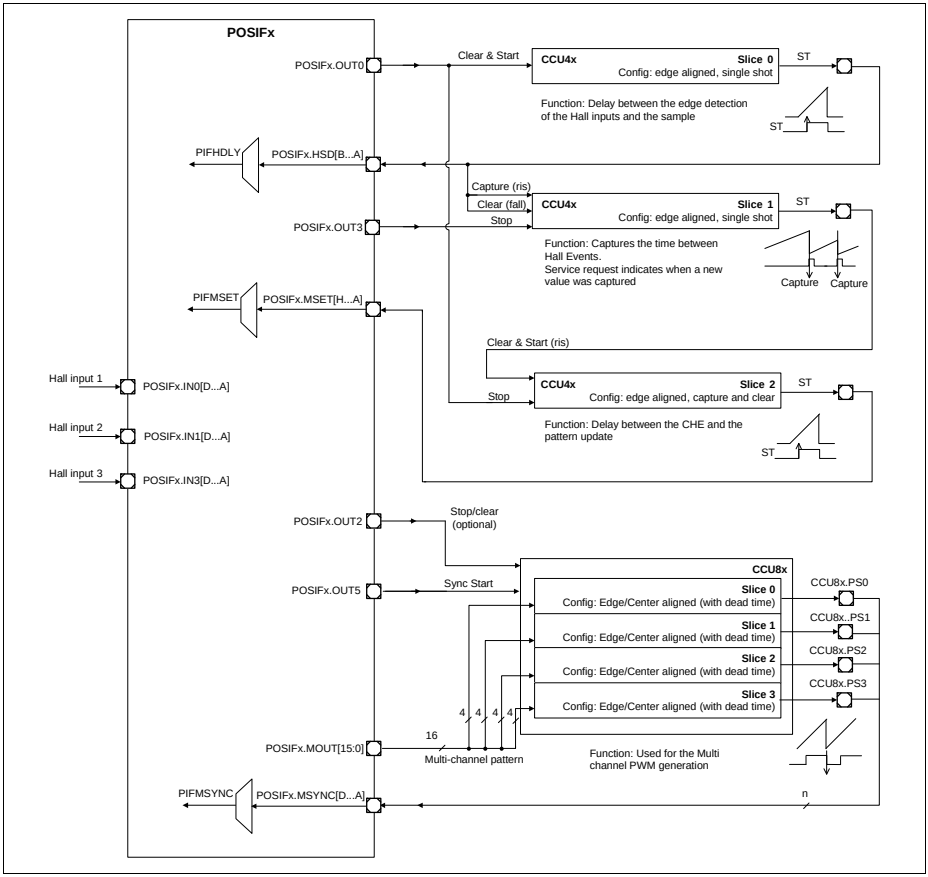


Figure 24-16 Hall Sensor Mode usage - profile 2

24.2.7.2 Quadrature Decoder Mode usage

The Quadrature Decoder Mode can be used in a very flexible way when connected with a CCU4 unit. The connection profile depends on the amount of functions that the user wants to perform in parallel: position control, revolution control and velocity measurement.

Quadrature Decoder Usage - Tick and Revolution Compare plus Velocity Between N Ticks

On **Figure 24-17**, the POSIF is linked with a CCU4, using all the module timer slices. In this profile, the user in Quadrature Decoder Mode has a slice being used for position control (comparison), one for revolutions counter and two aggregated slices used for velocity measurement.

Slice 0 is connected to the POSIFx.OUT0 and POSIFx.OUT1 outputs, which means that one has to configure POSIFx.OUT0 as counting functionality and POSIFx.OUT1 as Up/Down counting function. This slice is then used to track the actual position of the system and the compare channel can be configured to trigger the required actions, when the position reaches a certain value.

Slice 1 is connected to POSIFx.OUT4 pin, which means that it is used as a motor revolution counter. The compare channel can be programmed to trigger an interrupt every time that the motor performs N revolutions.

Slice 2 and Slice 3 are used to perform velocity measurements. Slice 2 receives the Quadrature Decoder period clock via POSIFx.OUT2, that is mapped to a counting functionality. The compare channel of this slice is then used to trigger a capture event in Slice3. The last one is using the module clock and therefore in every capture event, the actual system ticks are captured. This way the user knows how much time has elapsed between N phase periods and can calculate the corresponding velocity profiles.

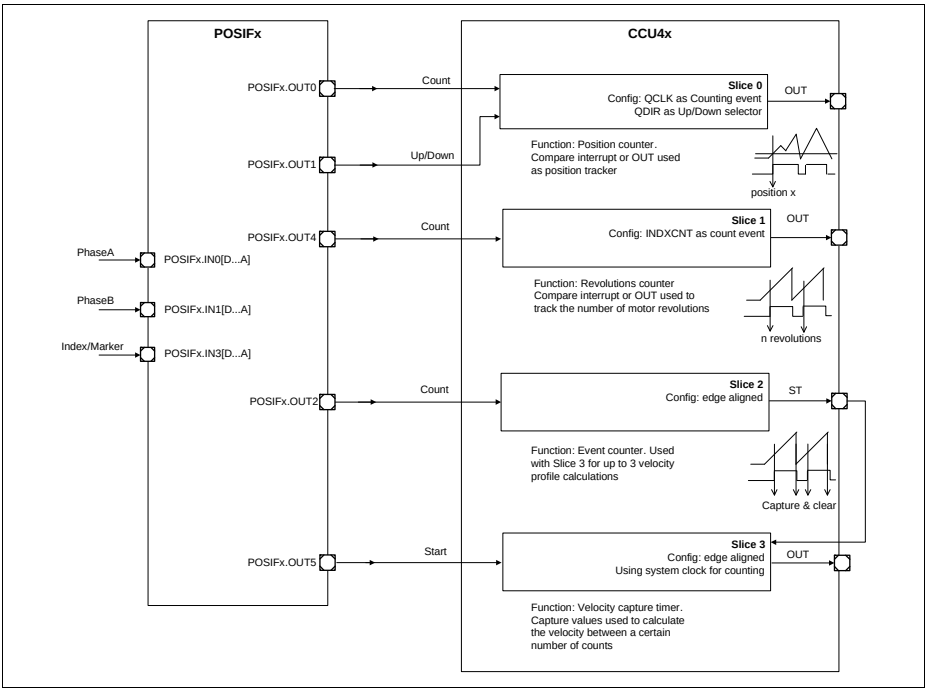


Figure 24-17 Quadrature Decoder Mode usage - profile 1

Quadrature Decoder Usage - Extended Tick Comparison plus Velocity Between N Ticks

The profile demonstrated in **Figure 24-18** enables the usage of 2 compare channels for the position control. This profile is especially useful for multi operations during just one motor revolution.

Slice 0 and Slice 1 are using the POSIFx.OUT0 as counting function and the POSIFxOUT1 as up/down control. The compare channels in each slice are then programmed with different compare values.

Slice 2 and Slice 3 are used in the same manner as profile 1, **Figure 24-18**.

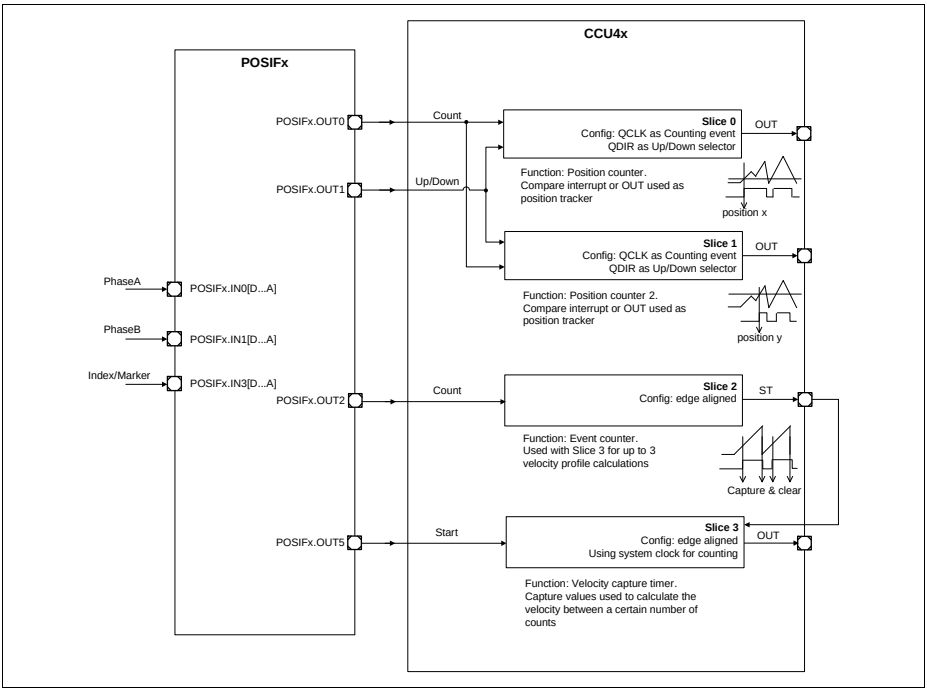


Figure 24-18 Quadrature Decoder Mode usage - profile 2

Quadrature Decoder Usage - Tick and Revolution Comparison with Index Clear plus Velocity Between N Ticks

In some applications it is useful to use the index marker as a clear signal for the position and velocity control. This clear action is linked with the POSIFx.OUT3 pin, that can be programmed to be asserted only at the first index marker or at all maker hits. This pin is then connected to the specific CCU4 slices and used as clear functionality. **Figure 24-19** shows the adaptation of profile 1 with the index marker signal used as clear signal.

The same procedure can be used in profile 2, so that we can have the 2 compare channels plus the velocity measurement and the index used as clear signal.

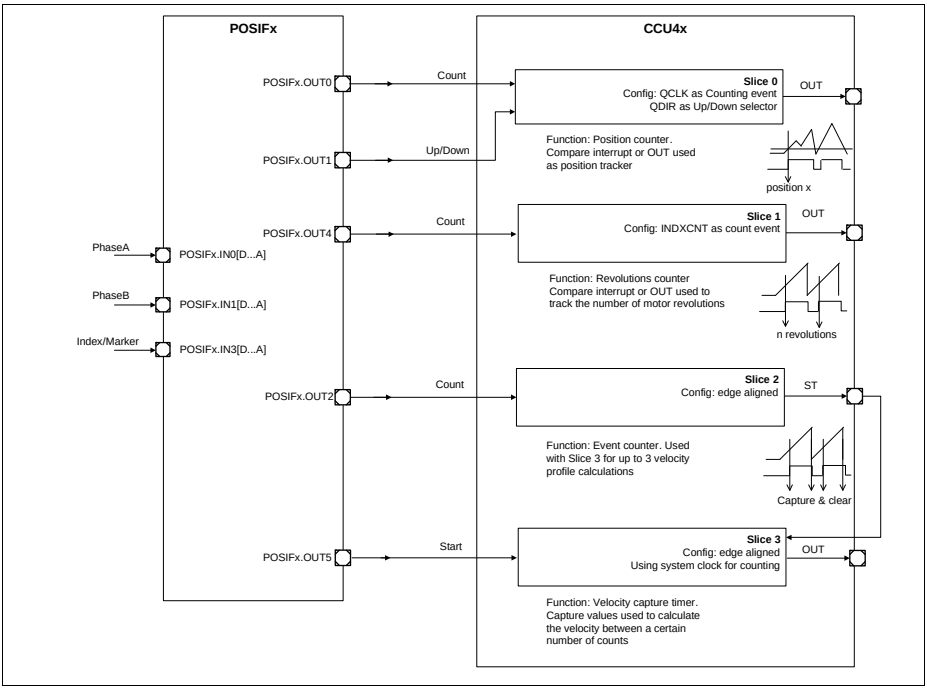


Figure 24-19 Quadrature Decoder Mode usage - profile 3

Quadrature Decoder Usage - Tick Comparison plus Micro Tick Velocity for Slow Rotating Speeds

When the motor rotating speed is slow, it may not be suitable to discard the time between each occurrence of a rotary encoder tick.

In a fast rotating system, the time between ticks is normally discarded and the velocity calculation can be done, by taking into account the number of ticks that have been elapsed since the last ISR. But in a slow rotating system, taking into account the number of ticks may not be enough (because of the associated error), and the software needs also to take into consideration, the time between the last tick and the actual ISR trigger.

Figure 24-20 shows a slow rotating system, where the ISR for the velocity calculation is triggered in a periodic way. In this case, because of the small amount of ticks between each ISR trigger, the software needs to know not only the amount of elapsed ticks but also the elapsed time between the last tick and the ISR occurrence.

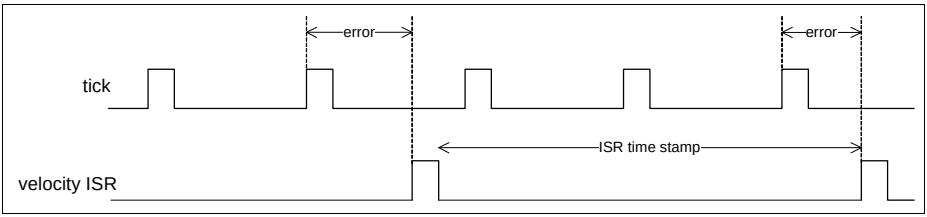


Figure 24-20 Slow rotating system example

It is possible to built a profile with the POSIF and one CCU4 module to perform a control loop that is immune to this slow velocity calculation pitfalls. The resource usage is exemplified on **Figure 24-21**.

One timer slice of a CCU4 module, Slice 0, is used to monitor the current position of the motor shaft.

Three additional timer slices are needed to built the slow velocity calculation loop: Slice 1, Slice 2 and Slice 3.

Timer Slice 1, is counting the number of ticks (PCLK) that are elapsed between each velocity ISR occurrence.

Timer Slice 2, is counting the time between each tick (PCLK) occurrence. This timer slice is cleared and started within every tick and therefore it always keeps the timing info between the last and the current tick.

Timer Slice 3, is controlling the periodicity of the velocity ISR. Every time that a velocity ISR is triggered, Slice 3 also triggers a capture in Slice 2 and a capture & clear in Slice 1.

With this mechanism, every time that the software reads back the captured values from Slice 1 and Slice 2, it can calculate the speed based on:

- the amount of ticks elapsed since the last ISR
- plus the amount of time elapsed between the last tick and the ISR

This control loop offers therefore a very accurate way to calculate the motor speed within systems with low velocity.

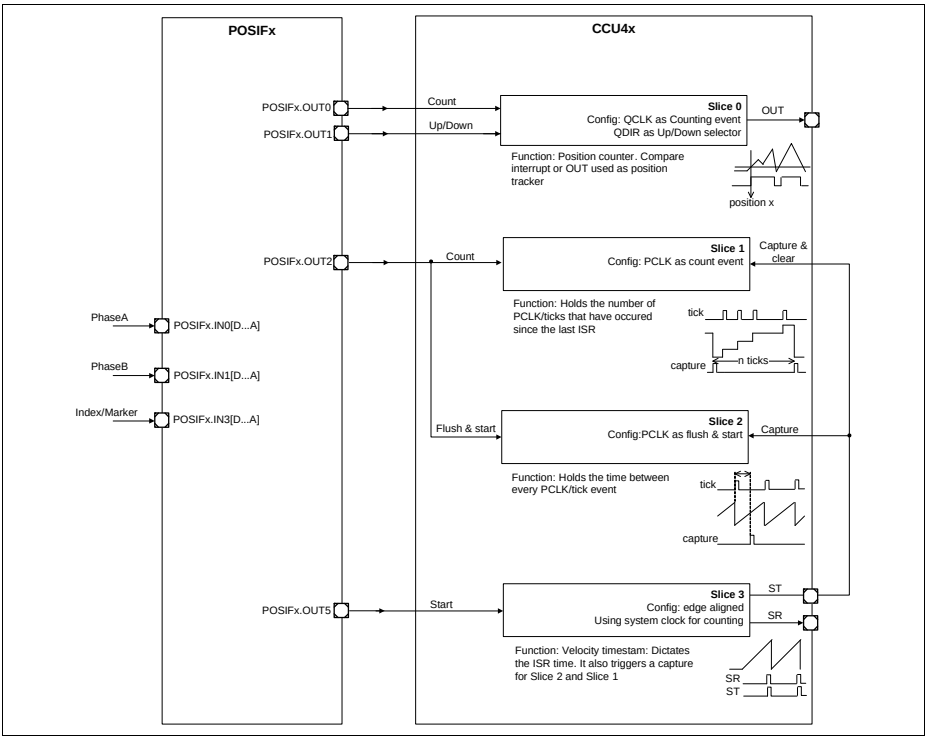


Figure 24-21 Quadrature Decoder Mode usage - profile 4

24.2.7.3 Stand-alone Multi-Channel Mode

The Multi-Channel Mode can be used in the POSIF module without being linked with the Hall Sensor Mode. This is especially useful for performing generic control loops that need to be perfectly synchronized.

The stand-alone Multi-Channel Mode is very generic and depends very much on the control loop specified by the user. A generic profile for the Multi-Channel mode is to use a CCU4/CCU8 module to perform the signal generation and a slice from a CCU4 module linked with an external pin that is used as trigger for updating the Multi-Channel pattern.

On **Figure 24-22**, a slice from a CCU4 unit is used to control the delay between the update of an external signal (e.g. external sensor) and the update of the Multi-Channel Pattern. Notice that this external signal can also be connected to the POSIF module if no timing delay is needed or it can be just controlled by SW, by writing an one into the **MCMS**.MNPS field.

One output can then be chosen from the CCU4/CCU8 unit that is generating the PWM signals, to act as synchronization trigger between the generated signal and the update of the Multi-Channel pattern (CCU4x.PSy in case of CCU4 and CCU8x.PSy in case of CCU8).

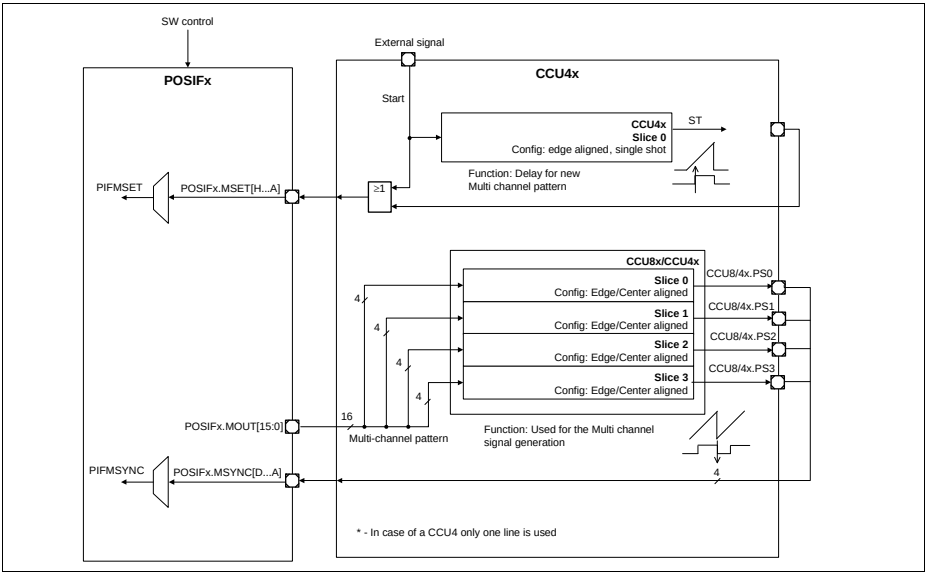


Figure 24-22 Stand-alone Multi-Channel Mode usage

24.3 Service Request Generation

The POSIF has several interrupt sources that are linked to the different operation modes: Hall Sensor Mode, Quadrature Decoder Mode and stand-alone Multi-Channel Mode.

The interrupt flags for the Hall Sensor Mode are described in **Section 24.3.1** while the interrupt flags of the Quadrature Decoder Mode are described in **Section 24.3.2**.

The flags associated with the Multi-Channel function are available in all the modes. This is due to the fact that the Multi-Channel Mode can operate in parallel with the Quadrature Decoder Mode and is needed whenever the Hall Sensor Mode is activated.

Each of the interrupt sources, can be routed to the POSIFx.SR0 or POSIFx.SR1 output, depending on the value programmed in the **PFLGE** register.

24.3.1 Hall Sensor Mode flags

The Hall Sensor Control contains four flags that can be configured to generate an interrupt request pulse, see **Figure 24-23**.

The four interrupt sources are:

- Transition at the Hall Inputs (**PFLG**.HIES)
- occurrence of a correct hall event (**PFLG**.CHES)
- occurrence of a wrong hall event (**PFLG**.WHES)
- shadow transfer of the Multi-Channel pattern (**PFLG**.MSTS)

The last one is triggered every time the Multi-Channel pattern is updated (PIFMST), which means that the POSIFx.MOUT[15:0] output was updated with a new value (see also **Figure 24-6**).

Each of this interrupt sources can be enabled/disabled individually. The SW also has the possibility to set and clear the specific flags by writing a 1_B into the specific fields of **SPFLG** and **RPFLG** registers. By enabling an interrupt source, an interrupt pulse is generated every time that a flag set operation occurs, independently if the flag is already set or not.

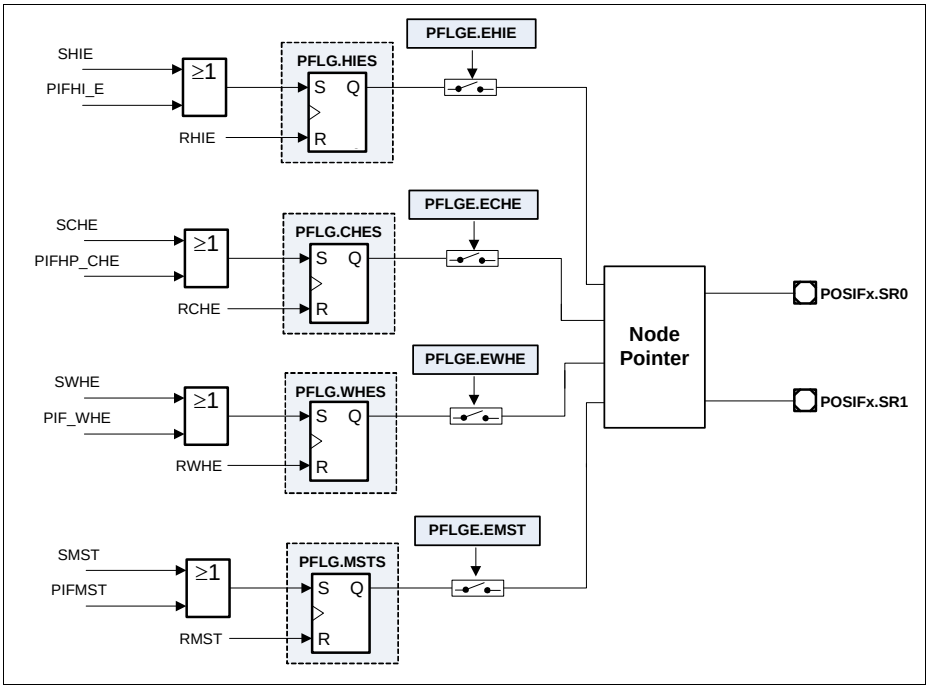


Figure 24-23 Hall Sensor Mode flags

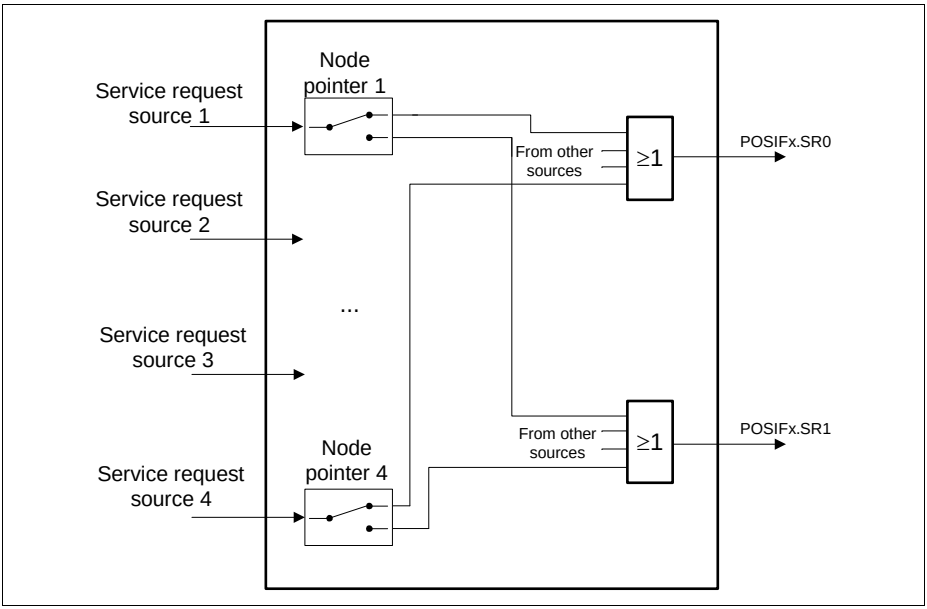


Figure 24-24 Interrupt node pointer overview - hall sensor mode

24.3.2 Quadrature Decoder Flags

The Quadrature Decoder Mode has five flags that can be enabled individually as interrupt sources (besides these five flags, the ones that are linked to the usage of Multi-Channel mode are also available: Multi-Channel Pattern update (**PFLG**.MSTS) and Wrong Hall event (**PFLG**.WHES). This can be useful if one selects the Quadrature Mode and the Multi-Channel stand-alone functionality.

These five flags are:

- Index event detection (**PFLG**.INDXS)
- phase detection error (**PFLG**.ERRS)
- quadrature clock generation (**PFLG**.CNTS)
- period clock generation (**PFLG**.PCLKS)
- direction change (**PFLG**.DIRS)

By enabling an interrupt source, an interrupt pulse is generated every time that a flag set operation occurs, independently if the flag is already set or not.

The index event detection flag is set every time that a index is detected.

The phase error flag is set when one invalid transition on the phase signals is detected, see **Figure 24-11**.

The quadrature clock and period clock flags are generated accordingly with the timings shown in **Figure 24-12**.

The direction change flag is set, every time that an inversion of the motor rotation happens.

Each flag can be set/cleared individually by SW, by writing into the specific field on the registers **SPFLG** and **RPFLG**, see **Figure 24-25**.

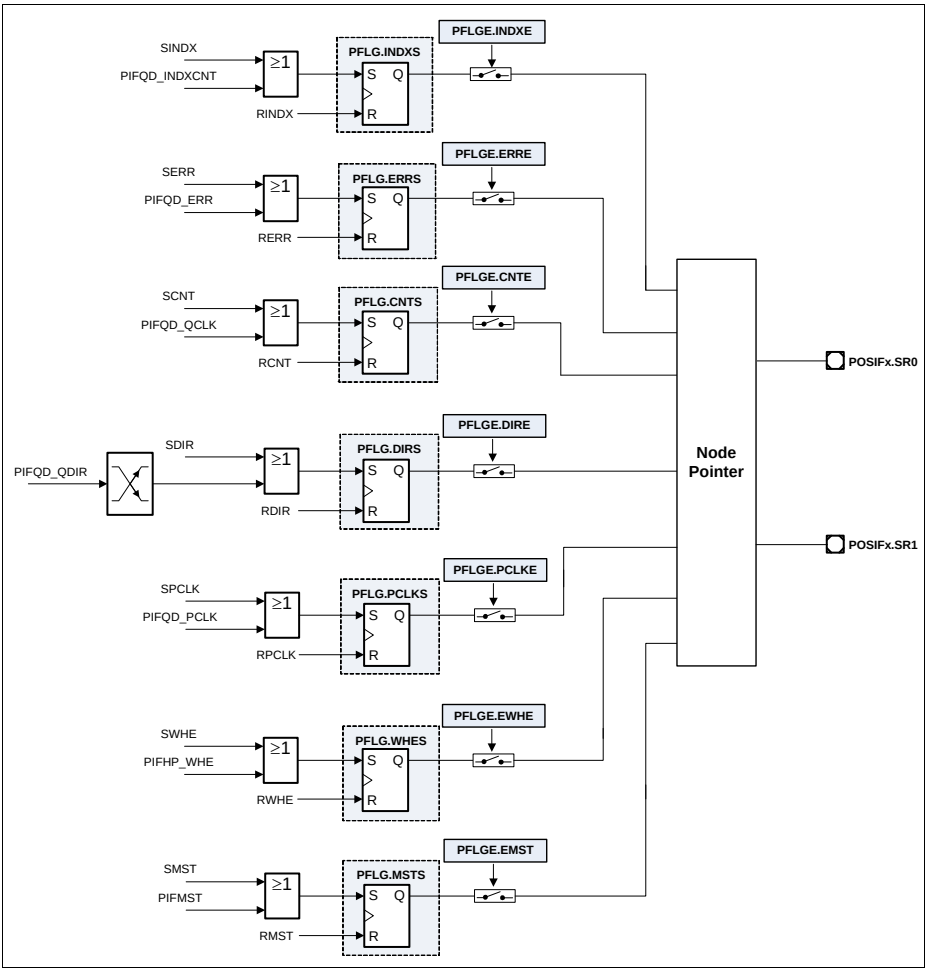


Figure 24-25 Quadrature Decoder flags

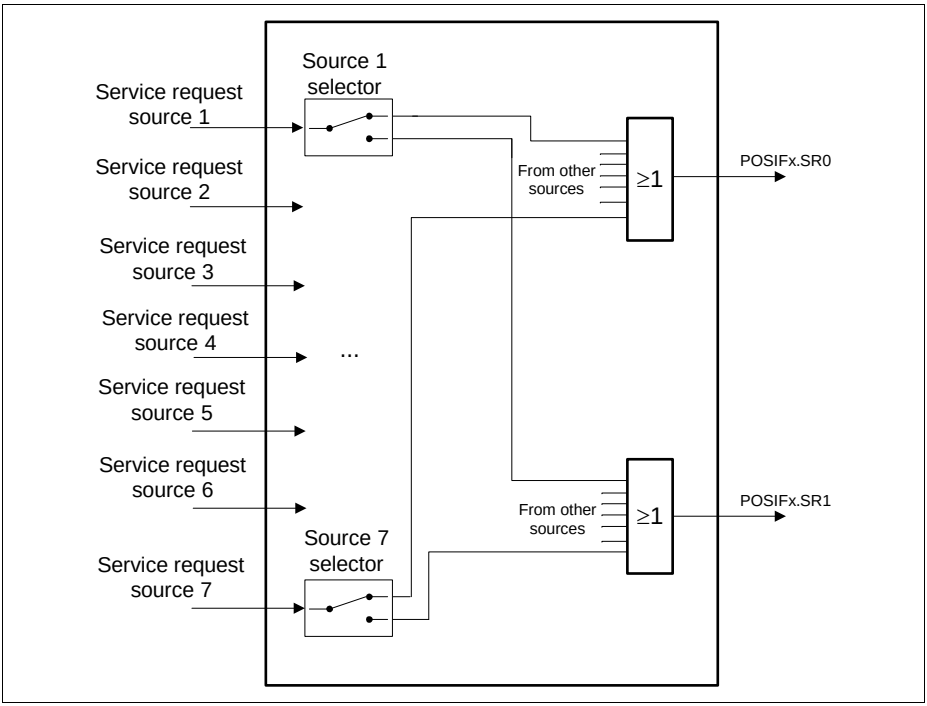


Figure 24-26 Interrupt node pointer overview - quadrature decoder mode

24.4 Debug Behavior

In suspend mode, the entire block is halted. The registers can still be accessed by the CPU (read only). This mode is useful for debugging purposes, e.g., where the current device status should be frozen in order to get a snapshot of the internal values. In suspend mode, all counters are stopped. The suspend mode is non-intrusive concerning the register bits. This means register bits are not modified by hardware when entering or leaving the suspend mode.

Entry into suspend mode can be configured at the kernel level by means of the register **PSUS**.

The module is only functional after the suspend signal becomes inactive.

24.5 Power, Reset and Clock

The following sections describe the operating conditions, characteristics and timing requirements for the POSIF. All the timing information is related to the module clock, f_{posif}.

24.5.1 Clocks

Module Clock

The module clock of the POSIF module is described in the SCU chapter as f_{CCU}.

The bus interface clock of the POSIF module is described in the SCU chapter as f_{PERIPH}.

The module clock for the POSIF is controlled via a specific control bit inside the SCU (System Control Unit), register CLKSET.

It is possible to disable the module clock for the POSIF, via a specific system control bit, nevertheless, there may be a dependency on the f_{posif} through the different POSIF instances or Capture/Compare Units. One should address the SCU Chapter for a complete description of the product clock scheme.

External Signals

The maximum frequency for the external signals, linked with the Hall Sensor and Rotary Encoder inputs can be seen in **Table 24-4**.

Table 24-4 External Hall/Rotary signals operating conditions

Parameter	Symbol	Values			Unit	Note / Test Condition
		Min.	Typ.	Max.		
Frequency	f_{esig}	–	–	$f_{posif}/4$	MHz	
ON time	ton_{esig}	$2T_{posif}$	–	–	ns	
OFF time	$toff_{esig}$	$2T_{posif}$	–	–	ns	

24.5.2 Module Reset

Each POSIF has one reset source. This reset source is handled at system level and it can be generated independently via a system control register, PRSET0 (address SCU chapter for a full description).

After reset release, the complete IP is set to default configuration. The default configuration for each register field is addressed on **Section 24.7**.

24.5.3 Power

The POSIF is inside the power core domain, therefore no special considerations about power up or power down sequences need to be taken. For a explanation about the different power domains, please address the SCU (System Control Unit) chapter.

24.6 Initialization and System Dependencies

24.6.1 Initialization

The initialization sequence for an application that is using the POSIF, should be the following:

1st Step: Apply reset to the POSIF, via the specific SCU register, PRSET0.

2nd Step: Release reset of the POSIF, via the specific SCU register, PRCLR0.

3rd Step: Enable the POSIF clock via the specific SCU register, CLKSET.

4th Step: Configure the POSIF operation mode, **PCONF** register.

5th Step: Configure the POSIF registers linked to the wanted operation mode:

- For Hall Sensor Mode: **HALPS**/**HALP**, **MCSM**/**MCM**
- For Quadrature Decoder Mode: **QDC**
- For stand-alone Multichannel Mode: **MCSM**/**MCM**

5th Step: Apply the pre programmed patterns into the **HALP** and **MCM** registers, by writing 1_B into the **MCMS**.STHR and **MCMS**.STMR fields.

6th Step: Configure the POSIF interrupts/service requests via the **PFLGE** register.

7th Step: Configure the Capture/Compare Units associated with the POSIF operation mode.

8th Step: If the synchronous start function of the POSIF is being used inside the associated Capture/Compare units, then one just needs to set the run bit of the module, by writing a 1_B into **PRUNS**.SRB.

9th Step: If the synchronous start function of the POSIF is not being used inside the associated Capture/Compare units, then start first the POSIF by writing a 1_B into **PRUNS**.SRB. After that, start the associated Capture/Compare Unit timer slices, by addressing the specific bit field inside each timer slice or by using the synchronous start function available on the SCU, CCUCON register.

Note: Due to different startup conditions of the motor system itself (as well SW control loop implementation) it is possible to receive some erroneous interrupt triggers before the SW and HW loop are completely locked. To overcome this, the SW can ignore the interrupts during start up or the interrupt sources can be enabled only after a proper lock between HW and SW has been sensed.

24.6.2 System Dependencies

Each POSIF may have different dependencies regarding module and bus clock frequencies. This dependencies should be addressed in the SCU and System Architecture Chapters.

Dependencies between several peripherals, regarding different clock operating frequencies may also exist. This should be addressed before configuring the connectivity between the POSIF and some other peripheral.

The following topics must be taken into consideration for good POSIF and system operation:

- POSIF module clock must be at maximum two times faster than the module bus interface clock
- Module input triggers for the POSIF must not exceed the module clock frequency (if the triggers are generated internally in the device)
- Module input triggers for the POSIF must not exceed the frequency dictated in **Section 24.5.1**
- Frequency of the POSIF outputs used as triggers/functions on other modules, must be crosschecked on the end point
- Applying and removing POSIF from reset, can cause unwanted operations in other modules. This can occur if the modules are using POSIF outputs as triggers/functions.

24.7 Registers

Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 24-5 Registers Address Space

Module	Base Address	End Address	Note
POSIF0	40028000_H	$4002BFFF_H$	
POSIF1	$4002C000_H$	$4002FFFF_H$	

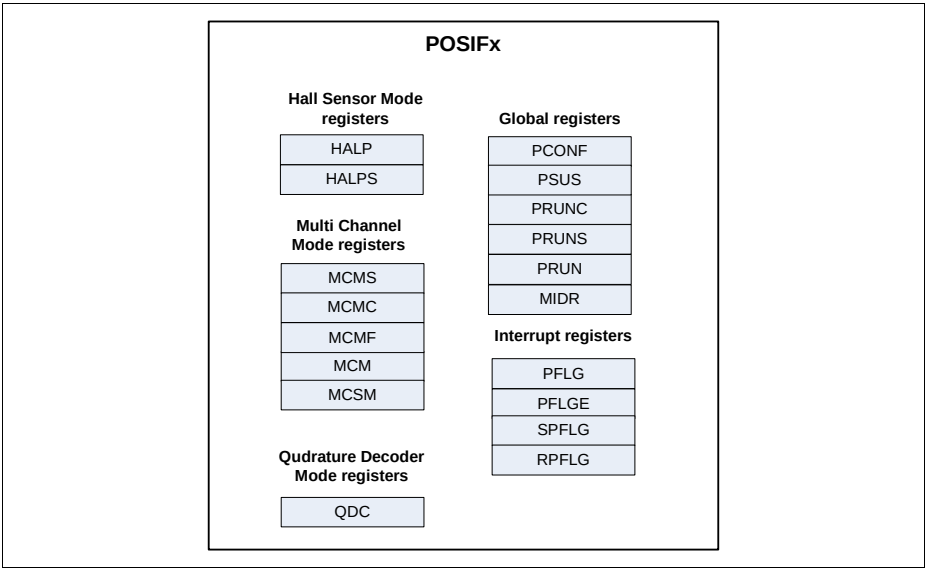


Figure 24-27 POSIF registers overview

Table 24-6 Register Overview of POSIF

Short Name	Description	Offset Addr.[1)]	Access Mode		Description See
			Read	Write	
POSIF Kernel Registers					
PCONF	Global control register	0000_H	U, PV	U, PV	Page 24-38
PSUS	Suspend Configuration	0004_H	U, PV	U, PV	Page 24-42
PRUNS	POSIF run bit set	0008_H	U, PV	U, PV	Page 24-43
PRUNC	POSIF run bit clear	$000C_H$	U, PV	U, PV	Page 24-43
PRUN	POSIF run bit status	0010_H	U, PV	BE	Page 24-44
MIDR	Module Identification register	0020_H	U, PV	BE	Page 24-45
Hall Sensor Mode Registers					
HALP	Hall Current and Expected patterns	0030_H	U, PV	BE	Page 24-46
HALPS	Hall Current and Expected shadow patterns	0034_H	U, PV	U, PV	Page 24-47
Multi-Channel Mode Register					
MCM	Multi-Channel Mode Pattern	0040_H	U, PV	BE	Page 24-48
MCSM	Multi-Channel Mode shadow Pattern	0044_H	U, PV	U, PV	Page 24-49
MCMS	Multi-Channel Mode Control set	0048_H	U, PV	U, PV	Page 24-49
MCMC	Multi-Channel Mode Control clear	$004C_H$	U, PV	U, PV	Page 24-50
MCMF	Multi-Channel Mode flag status	0050_H	U, PV	BE	Page 24-51
Quadrature Decoder Mode Register					
QDC	Quadrature Decoder Configuration	0060_H	U, PV	U, PV	Page 24-52
Interrupt Registers					
PFLG	POSIF interrupt status	0070_H	U, PV	BE	Page 24-53

Table 24-6 Register Overview of POSIF (cont'd)

Short Name	Description	Offset Addr.[1]	Access Mode		Description See
			Read	Write	
PFLGE	POSIF interrupt enable	0074_H	U, PV	U, PV	Page 24-55
SPFLG	Interrupt set register	0078_H	U, PV	U, PV	Page 24-58
RPFLG	Interrupt clear register	$007C_H$	U, PV	U, PV	Page 24-59

1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

24.7.1 Global registers

PCONF

The register contains the global configuration for the POSIF operation: operation mode, input selection, filter configuration.

PCONF
POSIF configuration **(0000_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		LPC		EWIL	EWIE	EWIS		MSYNS		MSES	MSETS			SPES	DSEL
r		rw		rw	rw	rw		rw		rw	rw			rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		INSEL2		INSEL1		INSEL0		0		MCUE	HIDG	0	QDCM	FSEL	
r		rw		rw		rw		r		rw	rw	r	rw	rw	

Field	Bits	Type	Description
FSEL	[1:0]	rw	**Function Selector** 00_B Hall Sensor Mode enabled 01_B Quadrature Decoder Mode enabled 10_B stand-alone Multi-Channel Mode enabled 11_B Quadrature Decoder and stand-alone Multi-Channel Mode enabled

Field	Bits	Type	Description
QDCM	2	rw	**Position Decoder Mode selection** This field selects if the Position Decoder block is in Quadrature Mode or Direction Count Mode. In Quadrature mode, the position encoder is providing the phase signals, while in Direction Count Mode is providing a clock and a direction signal. 0_B Position encoder is in Quadrature Mode 1_B Position encoder is in Direction Count Mode.
HIDG	4	rw	**Idle generation enable** Setting this field to 1_B disables the generation of the IDLE signal that forces a clear on the Multi-Channel pattern and run bit.
MCUE	5	rw	**Multi-Channel Pattern SW update enable** 0_B Multi-Channel pattern update is controlled via HW 1_B Multi-Channel pattern update is controlled via SW The update of the Multi-Channel pattern should only be controlled by SW when the Hall Sensor Mode is not being used
INSEL0	[9:8]	rw	**PhaseA/Hal input 1 selector** This fields selects which input is used for the Phase A or Hall input 1 function (dependent if the module is set for Quadrature Decoder or Hall Sensor Mode): 00_B POSIFx.IN0A 01_B POSIFx.IN0B 10_B POSIFx.IN0C 11_B POSIFx.IN0D
INSEL1	[11:10]	rw	**PhaseB/Hall input 2 selector** This fields selects which input is used for the Phase B or Hall input 2 function (dependent if the module is set for Quadrature Decoder or Hall Sensor Mode): 00_B POSIFx.IN1A 01_B POSIFx.IN1B 10_B POSIFx.IN1C 11_B POSIFx.IN1D

Field	Bits	Type	Description
INSEL2	[13:12]	rw	**Index/Hall input 3 selector** This fields selects which input is used for the Index or Hall input 3 function (dependent if the module is set for Quadrature Decoder or Hall Sensor Mode): 00_B POSIFx.IN2A 01_B POSIFx.IN2B 10_B POSIFx.IN2C 11_B POSIFx.IN2D
DSEL	16	rw	**Delay Pin selector** This field selects which input is used to trigger the end of the delay between the detection of an edge in the Hall inputs and the actual sample of the Hall inputs. 0_B POSIFx.HSDA 1_B POSIFx.HSDB
SPES	17	rw	**Edge selector for the sampling trigger** This field selects which edge is used of the selected POSIFx.HSD[B...A] signal to trigger a sample of the Hall inputs. 0_B Rising edge 1_B Falling edge
MSETS	[20:18]	rw	**Pattern update signal select** Selects the input signal that is used to enable a Multi-Channel pattern update. 000_B POSIFx.MSETA 001_B POSIFx.MSETB 010_B POSIFx.MSETC 011_B POSIFx.MSETD 100_B POSIFx.MSETE 101_B POSIFx.MSETF 110_B POSIFx.MSETG 111_B POSIFx.MSETH
MSES	21	rw	**Multi-Channel pattern update trigger edge** 0_B The signal used to enable a pattern update is active on the rising edge 1_B The signal used to enable a pattern update is active on the falling edge

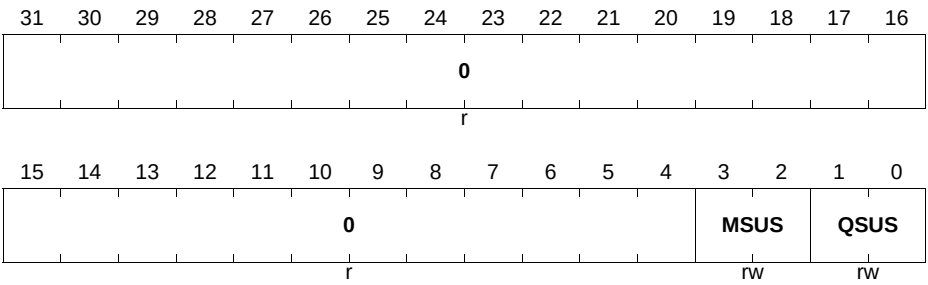
Field	Bits	Type	Description
MSYNS	[23:22]	rw	**PWM synchronization signal selector** This fields selects which input is used as trigger for Multi-Channel pattern update (synchronization with the PWM signal). 00_B POSIFx.MSYNCA 01_B POSIFx.MSYNCB 10_B POSIFx.MSYNCC 11_B POSIFx.MSYNCD
EWIS	[25:24]	rw	**Wrong Hall Event selection** 00_B POSIFx.EWHEA 01_B POSIFx.EWHEB 10_B POSIFx.EWHEC 11_B POSIFx.EWHED
EWIE	26	rw	**External Wrong Hall Event enable** 0_B External wrong hall event emulation signal, POSIFx.EWHE[D...A], is disabled 1_B External wrong hall event emulation signal, POSIFx.EWHE[D...A], is enabled.
EWIL	27	rw	**External Wrong Hall Event active level** 0_B POSIFx.EWHE[D...A] signal is active HIGH 1_B POSIFx.EWHE[D...A] signal is active LOW
LPC	[30:28]	rw	**Low Pass Filters Configuration** 000_B Low pass filter disabled 001_B Low pass of 1 clock cycle 010_B Low pass of 2 clock cycles 011_B Low pass of 4 clock cycles 100_B Low pass of 8 clock cycles 101_B Low pass of 16 clock cycles 110_B Low pass of 32 clock cycles 111_B Low pass of 64 clock cycles
0	3, [7:6], [15:14], 31	r	**Reserved** Read always returns 0

PSUS

The register contains the suspend configuration for the POSIF.

PSUS

POSIF Suspend Config **(0004_H)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
QSUS	[1:0]	rw	**Quadrature Mode Suspend Config** This field controls the entering in suspend for the quadrature decoder mode. 00_B Suspend request ignored 01_B Stop immediately 10_B Suspend in the next index occurrence 11_B Suspend in the next phase (PhaseA or PhaseB) occurrence
MSUS	[3:2]	rw	**Multi-Channel Mode Suspend Config** This field controls the entering in suspend for the Multi-Channel mode. The Hall sensor mode is also covered by this configuration. 00_B Suspend request ignored 01_B Stop immediately. Multi-Channel pattern is not set to the reset value. 10_B Stop immediately. Multi-Channel pattern is set to the reset value. 11_B Suspend with the synchronization of the PWM signal. Multi-Channel pattern is set to the reset value at the same time of the synchronization.
0	[31:4]	r	**Reserved** Read always returns 0

PRUNS

Via this register it is possible to set the run bit of the module.

PRUNS

POSIF Run Bit Set **(0008_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0															SRB
r															w

Field	Bits	Type	Description
SRB	0	w	**Set Run bit** Writing an 1_B into this bit sets the run bit of the module. Read always returns 0.
0	[31:1]	r	**Reserved** Read always returns 0

PRUNC

Via this register it is possible to clear the run bit and the internal state machines of the module.

PRUNC

POSIF Run Bit Clear **($000C_H$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0														CSM	CRB
r														w	w

Field	Bits	Type	Description
CRB	0	w	**Clear Run bit** Writing an 1_B into this bit clears the run bit of the module. The module is stopped. Read always returns 0.
CSM	1	w	**Clear Current internal status** Writing an 1_B into this bit resets the state machine of the quadrature decoder and the current status of the Hall sensor and Multi-Channel mode registers. The flags and static configuration bit fields are not cleared. Read always returns 0.
0	[31:2]	r	**Reserved** Read always returns 0

PRUN

The register contains the run bit status of the POSIF.

PRUN
POSIF Run Bit Status **(0010_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0															**RB**
r															rh

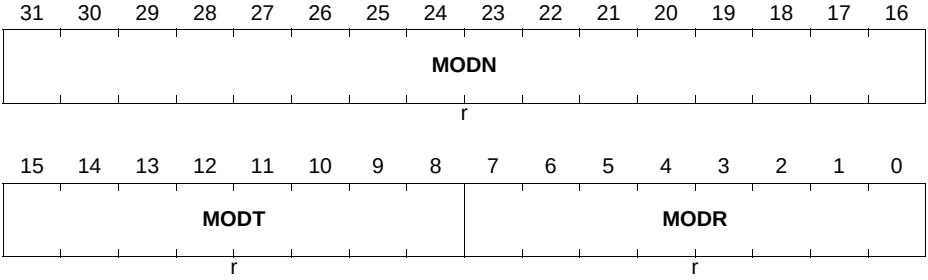
Field	Bits	Type	Description
RB	0	rh	**Run Bit** This field indicates if the module is in running or IDLE state. 0_B IDLE 1_B Running

Field	Bits	Type	Description
0	[31:1]	r	**Reserved** Read always returns 0

MIDR

This register contains the module identification number.

MIDR
Module Identification register (0020$_H$) Reset Value: 00A8C0XX$_H$



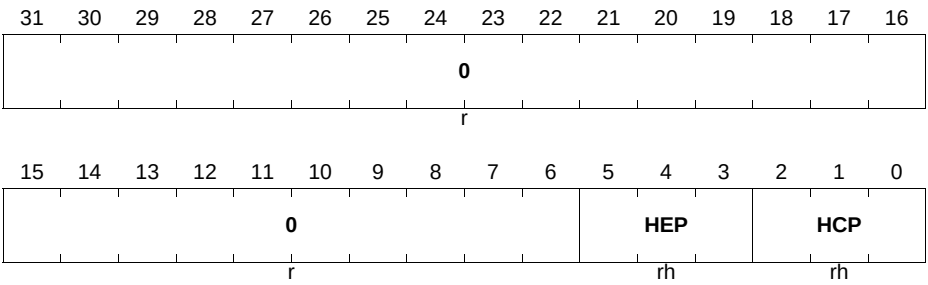
Field	Bits	Type	Description
MODR	[7:0]	r	**Module Revision** This bit field indicates the revision number of the module implementation (depending on the design step).
MODT	[15:8]	r	**Module Type**
MODN	[31:16]	r	**Module Number**

24.7.2 Hall Sensor Mode Registers

HALP

The register contains the values for the Hall Expected Pattern and Hall Current Pattern.

HALP
Hall Sensor Patterns **(0030_H)** **Reset Value: 00000000_H**

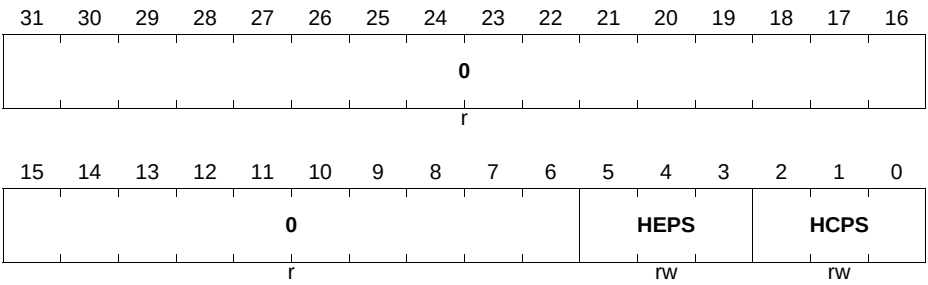


Field	Bits	Type	Description
HCP	[2:0]	rh	**Hall Current Pattern** This field contains the Hall Current pattern. This field is updated with the **HALPS**.HCPS value every time that a correct hall event occurs. HCP[0] - Hall Input 1 HCP[1] - Hall Input 2 HCP[2] - Hall Input 3
HEP	[5:3]	rh	**Hall Expected Pattern** This field contains the Hall Expected pattern. This field is updated with the **HALPS**.HEPS values every time that a correct hall event occurs. HEP[0] - Hall Input 1 HEP[1] - Hall Input 2 HEP[2] - Hall Input 3
0	[31:6]	r	**Reserved** Read always returns 0

HALPS

The register contains the values that are going to be loaded into the **HALP** register when the next correct hall event occurs.

HALPS
Hall Sensor Shadow Patterns **(0034_H)** **Reset Value: 00000000_H**



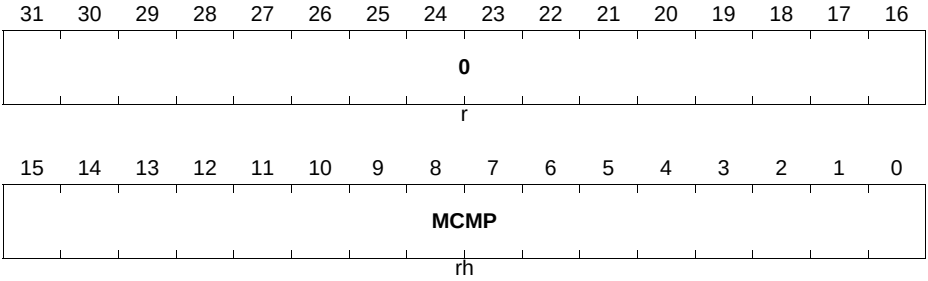
Field	Bits	Type	Description
HCPS	[2:0]	rw	**Shadow Hall Current Pattern** This field contains the next Hall Current pattern value. This field is set on the **HALP**.HCP field every time that a correct hall event occurs. HCPS[0] - Hall Input 1 HCPS[1] - Hall Input 2 HCPS[2] - Hall Input 3
HEPS	[5:3]	rw	**Shadow Hall expected Pattern** This field contains the next Hall Expected pattern. This field is set on the **HALP**.HEP field every time that a correct hall event occurs. HEPS[0] - Hall Input 1 HEPS[1] - Hall Input 2 HEPS[2] - Hall Input 3
0	[31:6]	r	**Reserved** Read always returns 0

24.7.3 Multi-Channel Mode Registers

MCM

The register contains the value of the Multi-Channel pattern that is applied to the outputs POSIFx.OUT[15:0].

MCM
Multi-Channel Pattern **(0040_H)** **Reset Value: 00000000_H**

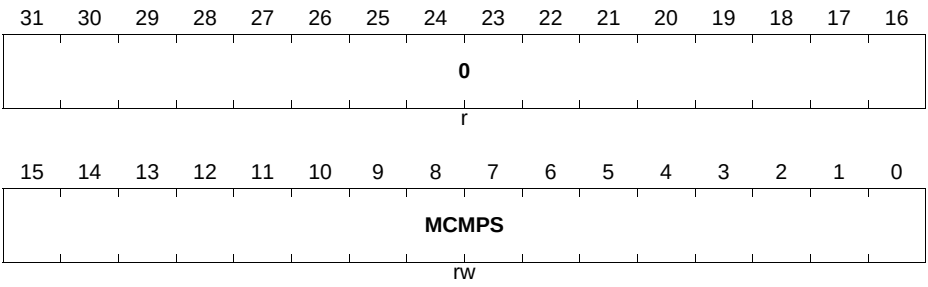


Field	Bits	Type	Description
MCMP	[15:0]	rh	**Multi-Channel Pattern** This field contains the Multi-Channel Pattern that is going to be applied to the Multi-Channel outputs, POSIFx.MOUT[15:0]. This field is updated with the value of the **MCSM**.MCMPS every time that a Multi-Channel pattern update is triggered.
0	[31:16]	r	**Reserved** Read always returns 0

MCSM

The register contains the value that is going to be loaded into the **MCM** register when the next Multi-Channel update trigger occurs.

MCSM
Multi-Channel Shadow Pattern **(0044_H)** **Reset Value: 00000000_H**

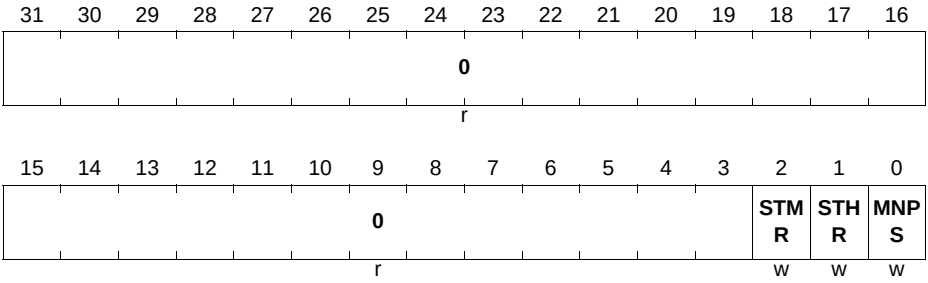


Field	Bits	Type	Description
MCMPS	[15:0]	rw	**Shadow Multi-Channel Pattern** This field contains the next Multi-Channel Pattern. Every time that a Multi-Channel pattern transfer is triggered, this value is passed into the field **MCM**.MCMP.
0	[31:16]	r	**Reserved** Read always returns 0

MCMS

Through this register it is possible to request a Multi-Channel pattern update. It is also possible through this register to request an immediate update of the Multi-Channel and Hall Sensor patterns without waiting for the hardware trigger.

MCMS
Multi-Channel Pattern Control set **(0048_H)** **Reset Value: 00000000_H**



Field	Bits	Type	Description
MNPS	0	w	**Multi-Channel Pattern Update Enable Set** Writing a 1_B into this field enables the Multi-Channel pattern update (sets the **MCMF**.MSS bit). The update is not done immediately due to the fact that the trigger that synchronizes the update with the PWM is still needed. A read always returns 0.
STHR	1	w	**Hall Pattern Shadow Transfer Request** Writing a 1_B into this field leads to an immediate update of the fields **HALP**.HCP and **HALP**.HEP. A read always returns 0.
STMR	2	w	**Multi-Channel Shadow Transfer Request** Writing a 1_B into this field leads to an immediate update of the field **MCM**.MCMP. A read always returns 0.
0	[31:3]	r	**Reserved** Read always returns 0

MCMC

Through this register is possible to cancel a Multi-Channel pattern update and to clear the Multi-Channel pattern to the default value.

MCMC
Multi-Channel Pattern Control clear (004C$_H$) **Reset Value: 00000000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0														MPC	MNPC
r														w	w

Field	Bits	Type	Description
MNPC	0	w	**Multi-Channel Pattern Update Enable Clear** Writing a 1_B into this field clears the **MCMF**.MSS bit. A read always returns 0.
MPC	1	w	**Multi-Channel Pattern clear** Writing a 1_B into this field clears the Multi-Channel Pattern value to 0000_H. A read always returns 0.
0	[31:2]	r	**Reserved** Read always returns 0

MCMF

The register contains the status of the Multi-Channel update request.

MCMF
Multi-Channel Pattern Control flag (0050_H) **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0															**MSS**
r															rh

Field	Bits	Type	Description
MSS	0	rh	**Multi-Channel Pattern update status** This field indicates if the Multi-Channel pattern is ready to be updated or not. When this field is set, the Multi-Channel pattern is updated when the triggering signal, selected from the POSIFx.MSYNC[D...A], becomes active. 0_B Update of the Multi-Channel pattern is set 1_B Update of the Multi-Channel pattern is not set
0	[31:1]	r	**Reserved** Read always returns 0

24.7.4 Quadrature Decoder Registers

QDC

The register contains the configuration for the operation of the Quadrature Decoder Mode.

QDC
Quadrature Decoder Control **(0060_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							DVAL	0		ICM		0	PHS	PBLS	PALS
r							rh	r		rw		r	rw	rw	rw

Field	Bits	Type	Description
PALS	0	rw	**Phase A Level selector** 0_B Phase A is active HIGH 1_B Phase A is active LOW
PBLS	1	rw	**Phase B Level selector** 0_B Phase B is active HIGH 1_B Phase B is active LOW
PHS	2	rw	**Phase signals swap** 0_B Phase A is the leading signal for clockwise rotation 1_B Phase B is the leading signal for clockwise rotation

Field	Bits	Type	Description
ICM	[5:4]	rw	**Index Marker generations control** This field controls the generation of the index marker that is linked to the output pin POSIFx.OUT3. 00_B No index marker generation on POSIFx.OUT3 01_B Only first index occurrence generated on POSIFx.OUT3 10_B All index occurrences generated on POSIFx.OUT3 11_B Reserved
DVAL	8	rh	**Current rotation direction** 0_B Counterclockwise rotation 1_B Clockwise rotation
0	3, [7:6], [31:9]	r	**Reserved** Read always returns 0

24.7.5 Interrupt Registers

PFLG

The register contains the status of all the interrupt flags of the module.

PFLG
POSIF Interrupt Flags **(0070_H)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			PCLKS	DIRS	CNTS	ERRS	INDXS	0			MSTS	0	HIES	WHES	CHES
r			rh	rh	rh	rh	rh	r			rh	r	rh	rh	rh

Field	Bits	Type	Description
CHES	0	rh	**Correct Hall Event Status** 0_B Correct Hall Event not detected 1_B Correct Hall Event detected

Field	Bits	Type	Description
WHES	1	rh	**Wrong Hall Event Status** 0_B Wrong Hall Event not detected 1_B Wrong Hall Event detected
HIES	2	rh	**Hall Inputs Update Status** 0_B Transition on the Hall Inputs not detected 1_B Transition on the Hall Inputs detected
MSTS	4	rh	**Multi-Channel pattern shadow transfer status** 0_B Shadow transfer not done 1_B Shadow transfer done
INDXS	8	rh	**Quadrature Index Status** 0_B Index event not detected 1_B Index event detected
ERRS	9	rh	**Quadrature Phase Error Status** 0_B Phase Error event not detected 1_B Phase Error event detected
CNTS	10	rh	**Quadrature CLK Status** 0_B Quadrature clock not generated 1_B Quadrature clock generated
DIRS	11	rh	**Quadrature Direction Change** 0_B Change on direction not detected 1_B Change on direction detected
PCLKS	12	rh	**Quadrature Period Clk Status** 0_B Period clock not generated 1_B Period clock generated
0	3, [7:5], [31:13]	r	**Reserved** Read always returns 0

PFLGE

Through this register it is possible to enable or disable each of the available interrupt sources. It is also possible to select to which service request line an interrupt is forward.

PFLGE
POSIF Interrupt Enable **(0074$_H$)** **Reset Value: 00000000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0			PCLSEL	DIRSEL	CNTSEL	ERRSEL	INDSEL	0			MSTSEL	0	HIESEL	WHESEL	CHESEL
r			rw	rw	rw	rw	rw	r			rw	r	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			EPCLK	EDIR	ECNT	EERR	EINDX	0			EMST	0	EHIE	EWHE	ECHE
r			rw	rw	rw	rw	rw	r			rw	r	rw	rw	rw

Field	Bits	Type	Description
ECHE	0	rw	**Correct Hall Event Enable** 0_B Correct Hall Event interrupt disabled 1_B Correct Hall Event interrupt enabled
EWHE	1	rw	**Wrong Hall Event Enable** 0_B Wrong Hall Event interrupt disabled 1_B Wrong Hall Event interrupt enabled
EHIE	2	rw	**Hall Input Update Enable** 0_B Update of the Hall Inputs interrupt is disabled 1_B Update of the Hall Inputs interrupt is enabled
EMST	4	rw	**Multi-Channel pattern shadow transfer enable** 0_B Shadow transfer event interrupt disabled 1_B Shadow transfer event interrupt enabled
EINDX	8	rw	**Quadrature Index Event Enable** 0_B Index event interrupt disabled 1_B Index event interrupt enabled
EERR	9	rw	**Quadrature Phase Error Enable** 0_B Phase error event interrupt disabled 1_B Phase error event interrupt enabled
ECNT	10	rw	**Quadrature CLK interrupt Enable** 0_B Quadrature CLK event interrupt disabled 1_B Quadrature CLK event interrupt enabled
EDIR	11	rw	**Quadrature direction change interrupt Enable** 0_B Direction change event interrupt disabled 1_B Direction change event interrupt enabled

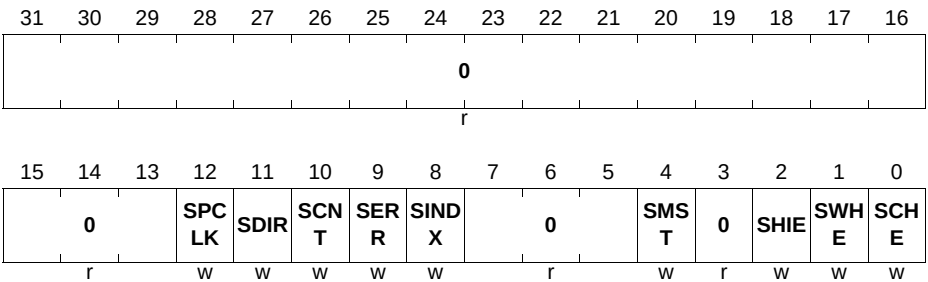
Field	Bits	Type	Description
EPCLK	12	rw	**Quadrature Period CLK interrupt Enable** 0_B Quadrature Period CLK event interrupt disabled 1_B Quadrature Period CLK event interrupt enabled
CHESEL	16	rw	**Correct Hall Event Service Request Selector** 0_B Correct Hall Event interrupt forward to POSIFx.SR0 1_B Correct Hall Event interrupt forward to POSIFx.SR1
WHESEL	17	rw	**Wrong Hall Event Service Request Selector** 0_B Wrong Hall Event interrupt forward to POSIFx.SR0 1_B Wrong Hall Event interrupt forward to POSIFx.SR1
HIESEL	18	rw	**Hall Inputs Update Event Service Request Selector** 0_B Hall Inputs Update Event interrupt forward to POSIFx.SR0 1_B Hall Inputs Update Event interrupt forward to POSIFx.SR1
MSTSEL	20	rw	**Multi-Channel pattern Update Event Service Request Selector** 0_B Multi-Channel pattern Update Event interrupt forward to POSIFx.SR0 1_B Multi-Channel pattern Update Event interrupt forward to POSIFx.SR1
INDSEL	24	rw	**Quadrature Index Event Service Request Selector** 0_B Quadrature Index Event interrupt forward to POSIFx.SR0 1_B Quadrature Index Event interrupt forward to POSIFx.SR1
ERRSEL	25	rw	**Quadrature Phase Error Event Service Request Selector** 0_B Quadrature Phase error Event interrupt forward to POSIFx.SR0 1_B Quadrature Phase error Event interrupt forward to POSIFx.SR1

Field	Bits	Type	Description
CNTSEL	26	rw	**Quadrature Clock Event Service Request Selector** 0_B Quadrature Clock Event interrupt forward to POSIFx.SR0 1_B Quadrature Clock Event interrupt forward to POSIFx.SR1
DIRSEL	27	rw	**Quadrature Direction Update Event Service Request Selector** 0_B Quadrature Direction Update Event interrupt forward to POSIFx.SR0 1_B Quadrature Direction Update Event interrupt forward to POSIFx.SR1
PCLSEL	28	rw	**Quadrature Period clock Event Service Request Selector** 0_B Quadrature Period clock Event interrupt forward to POSIFx.SR0 1_B Quadrature Period clock Event interrupt forward to POSIFx.SR1
0	3, [7:5], [15:13], 19, [23:21], [31:29]	r	**Reserved** Read always returns 0

SPFLG

Through this register it is possible for the SW to set a specific interrupt status flag.

SPFLG
POSIF Interrupt Set **(0078$_H$)** **Reset Value: 00000000$_H$**



Field	Bits	Type	Description
SCHE	0	w	**Correct Hall Event flag set** Writing a 1_B to this field sets the **PFLG**.CHES bit field. An interrupt pulse is generated. A read always returns 0.
SWHE	1	w	**Wrong Hall Event flag set** Writing a 1_B to this field sets the **PFLG**.WHES bit field. An interrupt pulse is generated. A read always returns 0.
SHIE	2	w	**Hall Inputs Update Event flag set** Writing a 1_B to this field sets the **PFLG**.HIES bit field. An interrupt pulse is generated. A read always returns 0.
SMST	4	w	**Multi-Channel Pattern shadow transfer flag set** Writing a 1_B to this field sets the **PFLG**.MSTS bit field. An interrupt pulse is generated. A read always returns 0.
SINDX	8	w	**Quadrature Index flag set** Writing a 1_B to this field sets the **PFLG**.INDXS bit field. An interrupt pulse is generated. A read always returns 0.
SERR	9	w	**Quadrature Phase Error flag set** Writing a 1_B to this field sets the **PFLG**.ERRS bit field. An interrupt pulse is generated. A read always returns 0.

Field	Bits	Type	Description
SCNT	10	w	**Quadrature CLK flag set** Writing a 1_B to this field sets the **PFLG**.CNTS bit field. An interrupt pulse is generated. A read always returns 0.
SDIR	11	w	**Quadrature Direction flag set** Writing a 1_B to this field sets the **PFLG**.DIRS bit field. An interrupt pulse is generated. A read always returns 0.
SPCLK	12	w	**Quadrature period clock flag set** Writing a 1_B to this field sets the **PFLG**.PCLKS bit field. An interrupt pulse is generated. A read always returns 0.
0	3, [7:5], [31:13]	r	**Reserved** Read always returns 0

RPFLG

Through this register it is possible for the SW to reset/clear a specific interrupt status flag.

RPFLG
POSIF Interrupt Clear **($007C_H$)** **Reset Value: 00000000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			RPCLK	RDIR	RCNT	RERR	RINDX	0			RMST	0	RHIE	RWHE	RCHE
r			w	w	w	w	w	r			w	r	w	w	w

Field	Bits	Type	Description
RCHE	0	w	**Correct Hall Event flag clear** Writing a 1_B to this field clears the **PFLG**.CHES bit field. A read always returns 0.

Field	Bits	Type	Description
RWHE	1	w	**Wrong Hall Event flag clear** Writing a 1_B to this field clears the **PFLG**.WHES bit field. A read always returns 0.
RHIE	2	w	**Hall Inputs Update Event flag clear** Writing a 1_B to this field clears the **PFLG**.HIES bit field. A read always returns 0.
RMST	4	w	**Multi-Channel Pattern shadow transfer flag clear** Writing a 1_B to this field clears the **PFLG**.MSTS bit field. A read always returns 0.
RINDX	8	w	**Quadrature Index flag clear** Writing a 1_B to this field clears the **PFLG**.INDXS bit field. A read always returns 0.
RERR	9	w	**Quadrature Phase Error flag clear** Writing a 1_B to this field clears the **PFLG**.ERRS bit field. A read always returns 0.
RCNT	10	w	**Quadrature CLK flag clear** Writing a 1_B to this field clears the **PFLG**.CNTS bit field. A read always returns 0.
RDIR	11	w	**Quadrature Direction flag clear** Writing a 1_B to this field clears the **PFLG**.DIRS bit field. A read always returns 0.
RPCLK	12	w	**Quadrature period clock flag clear** Writing a 1_B to this field clears the **PFLG**.PCLKS bit field. A read always returns 0.
0	3, [7:5], [31:13]	r	**Reserved** Read always returns 0

24.8 Interconnects

The following tables, describe the connectivity present on the device for each POSIF module.

The GPIO connections are available at the Ports chapter.

24.8.1 POSIF0 Pins

Table 24-7 POSIF0 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
POSIF0.CLK	I	SCU.CCUCLK	Module clock is the same one used by the capcoms
POSIF0.IN0A	I	GPIO	Shared connection for rotary encoder and hall sensor
POSIF0.IN0B	I	GPIO	Shared connection for rotary encoder and hall sensor
POSIF0.IN0C	I	VADC.G1BFL0	Shared connection for rotary encoder and hall sensor
POSIF0.IN0D	I	ERU1.PDOUT0	Shared connection for rotary encoder and hall sensor
POSIF0.IN1A	I	GPIO	Shared connection for rotary encoder and hall sensor
POSIF0.IN1B	I	GPIO	Shared connection for rotary encoder and hall sensor
POSIF0.IN1C	I	VADC.G1BFL1	Shared connection for rotary encoder and hall sensor
POSIF0.IN1D	I	ERU1.PDOUT1	Shared connection for rotary encoder and hall sensor
POSIF0.IN2A	I	GPIO	Shared connection for rotary encoder and hall sensor
POSIF0.IN2B	I	GPIO	Shared connection for rotary encoder and hall sensor
POSIF0.IN2C	I	VADC.C0SR0	Shared connection for rotary encoder and hall sensor
POSIF0.IN2D	I	ERU1.PDOUT2	Shared connection for rotary encoder and hall sensor
POSIF0.HSDA	I	CCU40.ST0	Used for the Hall pattern sample delay.

Table 24-7 POSIF0 Pin Connections (cont'd)

Global Inputs/Outputs	I/O	Connected To	Description
POSIF0.HSDB	I	0	Used for the Hall pattern sample delay.
POSIF0.EWHEA	I	VADC.G1BFL2	Wrong Hall event emulation, Trap, etc
POSIF0.EWHEB	I	ERU1.IOUT0	Wrong Hall event emulation, Trap, etc
POSIF0.EWHEC	I	ERU1.IOUT1	Wrong Hall event emulation, Trap, etc
POSIF0.EWHED	I	0	Wrong Hall event emulation, Trap, etc
POSIF0.MSETA	I	CCU40.SR0	Multi Pattern updated set. Requests a new shadow transfer
POSIF0.MSETB	I	CCU40.ST1	Multi Pattern updated set. Requests a new shadow transfer
POSIF0.MSETC	I	CCU42.SR0	Multi Pattern updated set. Requests a new shadow transfer
POSIF0.MSETD	I	CCU42.ST0	Multi Pattern updated set. Requests a new shadow transfer
POSIF0.MSETE	I	CCU80.SR1	Multi Pattern updated set. Requests a new shadow transfer
POSIF0.MSETF	I	ERU1.IOUT2	Multi Pattern updated set. Requests a new shadow transfer
POSIF0.MSETG	I	0	Multi Pattern updated set. Requests a new shadow transfer
POSIF0.MSETH	I	0	Multi Pattern updated set. Requests a new shadow transfer

Table 24-7 POSIF0 Pin Connections (cont'd)

Global Inputs/Outputs	I/O	Connected To	Description
POSIF0.MSYNCA	I	CCU80.PS1	Sync for updating the multi channel pattern with the shadow transfer
POSIF0.MSYNCB	I	CCU80.PS3	Sync for updating the multi channel pattern with the shadow transfer
POSIF0.MSYNCC	I	CCU40.PS1	Sync for updating the multi channel pattern with the shadow transfer
POSIF0.MSYNCD	I	CCU42.PS1	Sync for updating the multi channel pattern with the shadow transfer
POSIF0.OUT0	O	CCU40.IN0E; CCU40.IN1E; CCU40.IN2E	Quadrature mode: Quadrature clock; Hall sensor mode: Hall input edge detection
POSIF0.OUT1	O	CCU40.IN0F; CCU40.IN1F;	Quadrature mode: Shaft direction; Hall sensor mode: Correct Hall Event
POSIF0.OUT2	O	CCU40.IN1L; CCU40.IN2F; CCU42.IN0E; CCU42.IN1E; CCU42.IN2E; CCU42.IN3E; CCU80.IN0D; CCU80.IN1D; CCU80.IN2D; CCU80.IN3D;	Quadrature mode: Pclk for velocity; Hall sensor mode: Idle/wrong hall event
POSIF0.OUT3	O	CCU40.IN0G; CCU40.IN1G; CCU40.IN2G; CCU40.IN3E	Quadrature mode: Index event used for Clear/capt; Hall sensor mode: stop in Hall sensor mode
POSIF0.OUT4	O	CCU40.IN1H; CCU40.IN2H;	Quadrature mode:Index event; Hall sensor mode: Multi channel pattern update done

Table 24-7 POSIF0 Pin Connections (cont'd)

Global Inputs/Outputs	I/O	Connected To	Description
POSIF0.OUT5	O	CCU40.IN3F; CCU42.IN0F; CCU42.IN1F; CCU42.IN2F; CCU42.IN3F; CCU80.IN0E; CCU80.IN1E; CCU80.IN2E; CCU80.IN3E;	Sync start
POSIF0.OUT6	O	CCU42.MCSS; CCU80.MCSS;	Multi channel pattern update request
POSIF0.MOUT[0]	O	CCU42.MCI0; CCU80.MCI00;	Multi channel pattern
POSIF0.MOUT[1]	O	CCU42.MCI1; CCU80.MCI01;	Multi channel pattern
POSIF0.MOUT[2]	O	CCU42.MCI2; CCU80.MCI02;	Multi channel pattern
POSIF0.MOUT[3]	O	CCU42.MCI3; CCU80.MCI03;	Multi channel pattern
POSIF0.MOUT[4]	O	CCU80.MCI10;	Multi channel pattern
POSIF0.MOUT[5]	O	CCU80.MCI11;	Multi channel pattern
POSIF0.MOUT[6]	O	CCU80.MCI12;	Multi channel pattern
POSIF0.MOUT[7]	O	CCU80.MCI13;	Multi channel pattern
POSIF0.MOUT[8]	O	CCU80.MCI20;	Multi channel pattern
POSIF0.MOUT[9]	O	CCU80.MCI21;	Multi channel pattern
POSIF0.MOUT[10]	O	CCU80.MCI22;	Multi channel pattern
POSIF0.MOUT[11]	O	CCU80.MCI23;	Multi channel pattern
POSIF0.MOUT[12]	O	CCU80.MCI30;	Multi channel pattern
POSIF0.MOUT[13]	O	CCU80.MCI31;	Multi channel pattern
POSIF0.MOUT[14]	O	CCU80.MCI32;	Multi channel pattern
POSIF0.MOUT[15]	O	CCU80.MCI33;	Multi channel pattern

Table 24-7 POSIF0 Pin Connections (cont'd)

Global Inputs/Outputs	I/O	Connected To	Description
POSIF0.SR0	O	NVIC	Service request line 0
POSIF0.SR1	O	NVIC; VADC.G0REQTRO; VADC.G2REQTRO; VADC.BGREQTRO; ERU1.0A1; ERU1.1A1;	Service request line 1

24.8.2 POSIF1 Pins

Table 24-8 POSIF1 Pin Connections

Global Inputs/Outputs	I/O	Connected To	Description
POSIF1.CLK	I	SCU.CCUCLK	Module clock is the same one used by the capcoms
POSIF1.IN0A	I	GPIO	Shared connection for rotary encoder and hall sensor
POSIF1.IN0B	I	GPIO	Shared connection for rotary encoder and hall sensor
POSIF1.IN0C	I	VADC.G1BFL0	Shared connection for rotary encoder and hall sensor
POSIF1.IN0D	I	ERU1.PDOUT0	Shared connection for rotary encoder and hall sensor
POSIF1.IN1A	I	GPIO	Shared connection for rotary encoder and hall sensor
POSIF1.IN1B	I	GPIO	Shared connection for rotary encoder and hall sensor
POSIF1.IN1C	I	VADC.G1BFL1	Shared connection for rotary encoder and hall sensor
POSIF1.IN1D	I	ERU1.PDOUT1	Shared connection for rotary encoder and hall sensor
POSIF1.IN2A	I	GPIO	Shared connection for rotary encoder and hall sensor

Table 24-8 POSIF1 Pin Connections (cont'd)

Global Inputs/Outputs	I/O	Connected To	Description
POSIF1.IN2B	I	GPIO	Shared connection for rotary encoder and hall sensor
POSIF1.IN2C	I	VADC.C0SR1	Shared connection for rotary encoder and hall sensor
POSIF1.IN2D	I	ERU1.PDOUT2	Shared connection for rotary encoder and hall sensor
POSIF1.HSDA	I	CCU41.ST0	Used for the Hall pattern sample delay.
POSIF1.HSDB	I	0	Used for the Hall pattern sample delay.
POSIF1.EWHEA	I	VADC.G1BFL2	Wrong Hall event emulation, Trap, etc
POSIF1.EWHEB	I	ERU1.IOUT0	Wrong Hall event emulation, Trap, etc
POSIF1.EWHEC	I	ERU1.IOUT1	Wrong Hall event emulation, Trap, etc
POSIF1.EWHED	I	0	Wrong Hall event emulation, Trap, etc
POSIF1.MSETA	I	CCU41.SR0	Multi Pattern updated set. Requests a new shadow transfer
POSIF1.MSETB	I	CCU41.ST1	Multi Pattern updated set. Requests a new shadow transfer
POSIF1.MSETC	I	CCU43.SR0	Multi Pattern updated set. Requests a new shadow transfer
POSIF1.MSETD	I	CCU43.ST0	Multi Pattern updated set. Requests a new shadow transfer
POSIF1.MSETE	I	CCU81.SR1	Multi Pattern updated set. Requests a new shadow transfer

Table 24-8 POSIF1 Pin Connections (cont'd)

Global Inputs/Outputs	I/O	Connected To	Description
POSIF1.MSETF	I	ERU1.IOUT2	Multi Pattern updated set. Requests a new shadow transfer
POSIF1.MSETG	I	0	Multi Pattern updated set. Requests a new shadow transfer
POSIF1.MSETH	I	0	Multi Pattern updated set. Requests a new shadow transfer
POSIF1.MSYNCA	I	CCU81.PS1	Sync for updating the multi channel pattern with the shadow transfer
POSIF1.MSYNCB	I	CCU81.PS3	Sync for updating the multi channel pattern with the shadow transfer
POSIF1.MSYNCC	I	CCU41.PS1	Sync for updating the multi channel pattern with the shadow transfer
POSIF1.MSYNCD	I	CCU43.PS1	Sync for updating the multi channel pattern with the shadow transfer
POSIF1.OUT0	O	CCU41.IN0E; CCU41.IN1E; CCU41.IN2E	Quadrature mode: Quadrature clock; Hall sensor mode: Hall input edge detection
POSIF1.OUT1	O	CCU41.IN0F; CCU41.IN1F;	Quadrature mode: Shaft direction; Hall sensor mode: Correct Hall Event

Table 24-8 POSIF1 Pin Connections (cont'd)

Global Inputs/Outputs	I/O	Connected To	Description
POSIF1.OUT2	O	CCU41.IN1L; CCU41.IN2F; CCU43.IN0E; CCU43.IN1E; CCU43.IN2E; CCU43.IN3E; CCU81.IN0D; CCU81.IN1D; CCU81.IN2D; CCU81.IN3D;	Quadrature mode: Pclk for velocity; Hall sensor mode: Idle/wrong hall event
POSIF1.OUT3	O	CCU41.IN0G; CCU41.IN1G; CCU41.IN2G; CCU41.IN3E	Quadrature mode: Index event used for Clear/capt; Hall sensor mode: stop in Hall sensor mode
POSIF1.OUT4	O	CCU41.IN1H; CCU41.IN2H;	Quadrature mode:Index event; Hall sensor mode: Multi channel pattern update done
POSIF1.OUT5	O	CCU41.IN3F; CCU43.IN0F; CCU43.IN1F; CCU43.IN2F; CCU43.IN3F; CCU81.IN0E; CCU81.IN1E; CCU81.IN2E; CCU81.IN3E;	Sync start
POSIF1.OUT6	O	CCU43.MCSS; CCU81.MCSS;	Multi channel pattern update request
POSIF1.MOUT[0]	O	CCU43.MCI0; CCU81.MCI00;	Multi channel pattern
POSIF1.MOUT[1]	O	CCU43.MCI1; CCU81.MCI01;	Multi channel pattern
POSIF1.MOUT[2]	O	CCU43.MCI2; CCU81.MCI02;	Multi channel pattern
POSIF1.MOUT[3]	O	CCU43.MCI3; CCU81.MCI03;	Multi channel pattern

Table 24-8 POSIF1 Pin Connections (cont'd)

Global Inputs/Outputs	I/O	Connected To	Description
POSIF1.MOUT[4]	O	CCU81.MCI10;	Multi channel pattern
POSIF1.MOUT[5]	O	CCU81.MCI11;	Multi channel pattern
POSIF1.MOUT[6]	O	CCU81.MCI12;	Multi channel pattern
POSIF1.MOUT[7]	O	CCU81.MCI13;	Multi channel pattern
POSIF1.MOUT[8]	O	CCU81.MCI20;	Multi channel pattern
POSIF1.MOUT[9]	O	CCU81.MCI21;	Multi channel pattern
POSIF1.MOUT[10]	O	CCU81.MCI22;	Multi channel pattern
POSIF1.MOUT[11]	O	CCU81.MCI23;	Multi channel pattern
POSIF1.MOUT[12]	O	CCU81.MCI30;	Multi channel pattern
POSIF1.MOUT[13]	O	CCU81.MCI31;	Multi channel pattern
POSIF1.MOUT[14]	O	CCU81.MCI32;	Multi channel pattern
POSIF1.MOUT[15]	O	CCU81.MCI33;	Multi channel pattern
POSIF1.SR0	O	NVIC	Service request line 0
POSIF1.SR1	O	NVIC; VADC.G1REQTRO; VADC.G3REQTRO; ERU1.2A1; ERU1.3A1;	Service request line 1

General Purpose I/O Ports

25 General Purpose I/O Ports (PORTS)

The XMC4500 has many digital port pins which can be used as General Purpose I/Os (GPIO) and are connected to the on-chip peripheral units.

25.1 Overview

The PORTS provide a generic and flexible software and hardware interface for all standard digital I/Os. Each port slice has the same software interfaces for the operation as General Purpose I/O and it further provides the connectivity to the on-chip periphery and the control for the pad characteristics. **Table 25-1** gives an overview of the available PORTS and other pins in the different packages of the XMC4500:

Table 25-1 Port/Pin Overview

Function	LQFP-144	LFBG-144	LQFP-100	Note
P0	16	16	13	
P1	16	16	16	
P2	16	16	13	
P3	16	16	7	
P4	8	8	2	
P5	12	12	4	
P6	7	7	–	
P14	14	14	14	Analog/Digital Input only
P15	12	12	4	Analog/Digital Input only
Dedicated I/Os	12	12	12	HIB_IO, TMS, TCK, USB, VBUS, XTAL, RTC_XTAL, PORST
Analog Supply, Reference and Ground	4	4	4	VDDA, VSSA, VAREF, VAGND
Digital Supply	9	7	9	VDDP, VDDC, VBAT
Digital Ground	2	4	2	VSS, VSSO
Exposed Die Pad	1	–	1	Must be connected to V_{SS}

25.1.1 Features

This is a list of the main features of the PORTS:

- same generic register interface for each port pin, **Chapter 25.8**
- simple and robust software access for General Purpose I/O functionality, **Chapter 25.2**
- separate set and clear output control to avoid read-modify-write operations, **Chapter 25.8.5**
- direct input connections to on-chip peripherals, **Chapter 25.2.1**
- parallel input of the same pin to different peripherals possible, for example triggering a capture event in a CAPCOM unit and a Service Request via the ERU
- up to four alternate output connections from peripherals selectable, **Chapter 25.2.2**
- separate input and output path, which allows to evaluate the input while the output is active (feedback, plausability check)
- dedicated hardware-controlled interface for EBU, SDMMC, LEDTS and QSPI with select option, **Chapter 25.3**
- programmable open-drain or push-pull output driver stage, **Chapter 25.8.1**
- programmable driver strength and slew rate, **Page 25-6**
- programmable weak pull-up and pull-down devices, **Chapter 25.8.1**
- programmable input inverter, **Chapter 25.2.1**
- programmable power-save behavior in Deep-Sleep mode, **Chapter 25.4**
- defined power-up/power-fail behavior, **Chapter 25.6**
- Privilege Mode restricted access to configuration registers to avoid accidential modification
- disabling of digital input stage on shared analog input ports, **Chapter 25.5**

25.1.2 Block Diagram

Below is a figure with the generic structure of a digital port pin, split into the port slice with the control logic and the pad with the pull devices and the input and output stages, **Figure 25-1**.

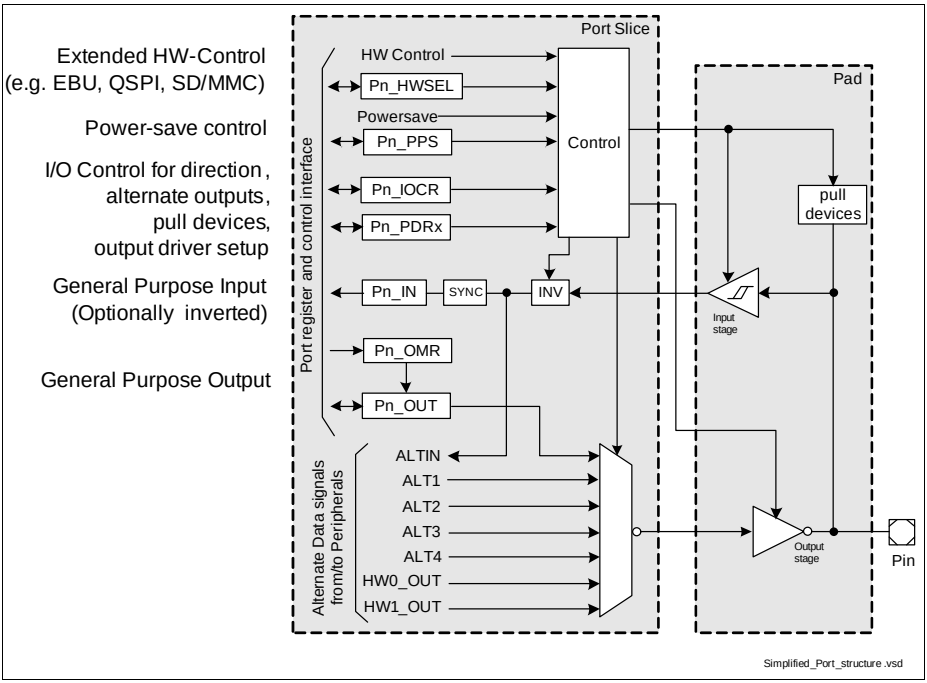


Figure 25-1 General Structure of a digital Port Pin

25.1.3 Definition of Terms

Some specific terms are used throughout this chaper:

- **Pin/Ball:** External connection of the device to the PCB.
- **Dedicated Pin:** A Pin with a dedicated function that is not under the control of the port logic (i.e. supply pins, $\overline{\text{PORST}}$).
- **Port Pin:** A pin under the control of the port logic (P0.1).
- **Port:** A group of up to 16 Port Pins sharing the same generic register set (P0).
- **Port Slice:** The “sum” of register bits and control logic used to control a port pin.
- **Pad:** Analog component containing the output driver, pull devices and input Schmitt-Trigger. Also interfaces the internal logic operating on V_{DDC} to the pad supply domain V_{DDP}.
- **GPIO:** General Purpose Input/Output. A port pin with the input and/or output function controlled by the application software.
- **Alternate Function:** Direct connection of a port pin with an on-chip peripheral.

25.2 GPIO and Alternate Function

The Ports can be operated as General Purpose Input/Outputs (GPIO) and with Alternate Functions of the on-chip periphery, configured by the Port Input/Output Control Register (Pn_IOCR, **Chapter 25.8.1**). It selects between

- Direct or Inverted Input
  - with or without pull device
- Push-pull or Open-Drain Output driven by
  - Pn_OUT (GPIO)
  - selected peripheral output connections.

As GPIO the port pin is controlled by the application software, reading the input value by the Port Input register Pn_IN (**Chapter 25.8.6**) and/or defining the output value by the Output Modification Register Pn_OMR (**Chapter 25.8.5**). Output modification by Pn_OMR is preferred over the direct change of the output value with the Output register Pn_OUT (**Chapter 25.8.4**), as Pn_OMR allows the manipulation of individual port pins in a single access without “disturbing” other pins controlled by the same Pn_OUT register. If an application uses a GPIO as a bi-directional I/O line, register Pn_IOCR has to be written to switch between input and output functionality.

For the operation with Alternate Functions, the port pins are directly connected to input or output functions of the on-chip periphery. This allows the peripheral to directly evaluate the input value or drive the output value of the port pin without further application software interaction after the initial configuration. The connection of alternate functions is used for control and communication interfaces, like a PWM from a CAPCOM unit or a SPI communication of a USIC channel. A detailed connectivity list of the peripherals to the port pins is given in the **Port I/O Functions** chapter. For specific functions, certain peripherals may also take direct control of “their” port pins, see **Hardware Controlled I/Os**.

25.2.1 Input Operation

As an input, the actual voltage level at the port pin is translated into a logical 0_B or 1_B via a Schmitt-Trigger device within the pad. The resulting input value can be optionally inverted. As general purpose input the signal is synchronized and can be read with the Input register (Pn_IN, **Chapter 25.8.6**). Alternatively, the input can be connected to multiple on-chip peripherals via the ALTIN signal. Where neccessary, these peripherals have internal controls to select the appropriate port pin with an input multiplexer stage, and will take care of synchronization and the further processing of the input signals (for more details on the input selection and handling see the respective peripheral chapters). With the Pn_IOCR register (**Chapter 25.8.1**) it is also possible to activate an internal weak pull-up or pull-down device in the pad.
The input register Pn_IN and the ALTIN signal always represent the state of the input, independent whether the port pin is configured as input or output. So, even if the port is

in output mode, the level of the pin can be read by software via Pn_IN and/or a peripheral can use the pin level as an input.

The ALTIN input signal of a port pin can be evaluated by multiple on-chip peripherals at the same time. For example, a pin used as slave select input of a USIC channel configured as SPI slave can also be used as trigger input of the ERU to trigger a service request or a wake-up event when the connected SPI master starts a communication.

25.2.2 Output Operation

In output mode, the output driver is activated and drives the value supplied through the output multiplexer to the port pin. Switching between input and output mode is accomplished through the Pn_IOCR register (**Chapter 25.8.1**), which

- enables or disables the output driver,
- selects between open-drain and push-pull mode,
- selects the general purpose or alternate function outputs.

The output multiplexer selects the signal source of the output with

- Pn_IOCR
  - general purpose output (Pn_OUT, **Chapter 25.8.4**)
  - alternate peripheral functions, ALT1..ALT4
- hardware control, Pn_HWSEL
  - HW0_OUT
  - HW1_OUT

Note: It is recommended to complete the Port and peripheral configuration with respect to driver strength, operating mode and inital values before the port pin is switched to output mode.

The output function is exclusive, meaning that always only exactly one peripheral has control of the output path.

Used as general purpose output, software can directly modify the content of Pn_OUT to define the output value on the pin. A write operation to Pn_OUT updates all port pins of that port (e.g. P0) that are configured as general purpose output. Updating just one or a selected few general purpose output pins via Pn_OUT requires a masked read-modify-write operation to avoid disturbing pins that shall not be changed. Direct writes to Pn_OUT will also affect Pn_OUT bits configured for use with the Pin Power-save function, **Chapter 25.4**.
Because of that, it is preferred to modify Pn_OUT bits by the Output Modification Register Pn_OMR (**Chapter 25.8.5**). The bits in Pn_OMR allow to individually set, clear or toggle the bits in the Pn_OUT register and only update the “addressed” Pn_OUT bits.

The data written by software into the output register Pn_OUT can also be used as input data to an on-chip peripheral. This enables, for example, peripheral tests and simulation via software without external circuitry.

Output lines of on-chip peripherals can directly control the output value of the output driver if selected via ALT1 to ALT4 as well as HW0_OUT and HW1_OUT. After initialization, this allows the connected peripherals to directly drive complex control and communication patterns without further software interaction with the ports.

The actual logic level at the pin can be examined through reading Pn_IN and compared against the applied output level (either applied by the output register Pn_OUT, or via an alternate output function of a peripheral unit). This can be used to detect some electrical failures at the pin caused through external circuitry. In addition, software-supported arbitration schemes between different “masters” can be implemented in this way, using the open-drain configuration and an external wired-AND circuitry. Collisions on the external communication lines can be detected when a high level (1_B) is output, but a low level (0_B) is seen when reading the pin value via the input register Pn_IN or directly by a peripheral (via ALTIN, for example a USIC channel in IIC mode).

Driver Mode

Before activating the push-pull driver, it is recommended to configure its driver strength and slew rate according to its pad class and the application need by the Pad Driver Mode register Pn_PDR (**Chapter 25.8.2**). Selecting the appropriate driver strength allows to optimize the outputs for the needed interface performance, can help to reduce power consumption, and limits noise, crosstalk and electromagnetic emissions.

There are three classes of GPIO output drivers:

- Class A1 pads (low speed 3.3V LVTTL outputs)
- Class A1+ pads (medium speed 3.3V LVTTL outputs)
- Class A2 pads (high speed 3.3V LVTTL outputs, e.g. for EBU or fast serial interfaces)

Class A1 pins provide the choice between medium and weak output drivers.
Class A1+ pins provide the choice between strong/medium/weak output drivers. For the strong driver, the signal transition edge can be additionally selected as soft or slow.
Class A2 pins provide the choice between strong/medium/weak output drivers. For the strong driver, the signal transition edge can be additionally selected as sharp/medium/soft.

The assignment of each port pin to one of these pad classes is listed in the **Package Pin Summary** table. Further details about pad properties in the XMC4500 are summarized in the Data Sheet.

25.3 Hardware Controlled I/Os

Some ports pins are overlaid with peripheral functions for which the connected peripheral needs direct hardware control, e.g. for the direction of a bi-directional data bus. There is a dedicated hardware control interface for these functions. As multiple peripherals need access to this interface, the Pn_HWSEL register (**Chapter 25.8.8**) allows to select between the hardware “masters”.

Depending on the operating mode, the peripheral can take control of various functions:

- Pin direction, input or output, e.g. for bi-directional signals
- Driver type, open-drain or push-pull
- Pull devices under peripheral control or under standard control via Pn_IOCR

Some configurations remain under control by the standard configuration interface, the output driver strength by Pn_PDR and the direct or inverted input path by Pn_IOCR. With active hardware control the Pin Power Save configuration is ignored.

Pn_HWSEL.HWx just pre-assigns the hardware-control of the pin to a certain peripheral, but the peripheral itself decides to actually take control over it. As long as the peripheral does not take control of a given pin via HWx_EN, the configuration of this pin is still defined by the configuration registers and it is available as GPIO or for other alternate functions. This might be because the selected peripheral has controls to just activate a subset of its pins, or because the peripheral is not active at all. E.g. unused address lines of the EBU are free for use as GPIO.

This mechanism can also be used to prohibit the hardware control of certain pins to a peripheral, in case the application does not need the respective functionality and the peripheral has no controls to disable the hardware control selectively.

If not specified differently, hardware outputs activate the push-pull output driver and the strength is defined by Pn_PDR.
Similarly, the default hardware input configuration and the pull devices are controlled by Pn_IOCR.

If the JTAG interface is selected by Pn_HWSEL of the respective port pins, pull device and output driver configuration is overridden by hardware.

If configured accordingly, the LEDTS module can also control the internal pull devices and change between push-pull and open-drain output drivers.

25.4 Power Saving Mode Operation

In Deep-Sleep mode, the behavior of a pin depends on the setting of the Pin Power Save register Pn_PPS (**Chapter 25.8.7**). Basically, each pin can be configured to react to the Power Save Mode Request or to ignore it. In case a pin is configured to react to a Power Save Mode Request, the output driver is switched to tri-state, the input Schmitt-Trigger and the pull devices are switched off (see **Figure 25-2**). The input signal to the on-chip peripherals is optionally driven statically high or low, software-defined by a value stored in Pn_OUT or by the last input value sampled to the Pn_OUT register during normal operation. The actual reaction is configured with the Pn_IOCR register under power save conditions, see **Table 25-8**.

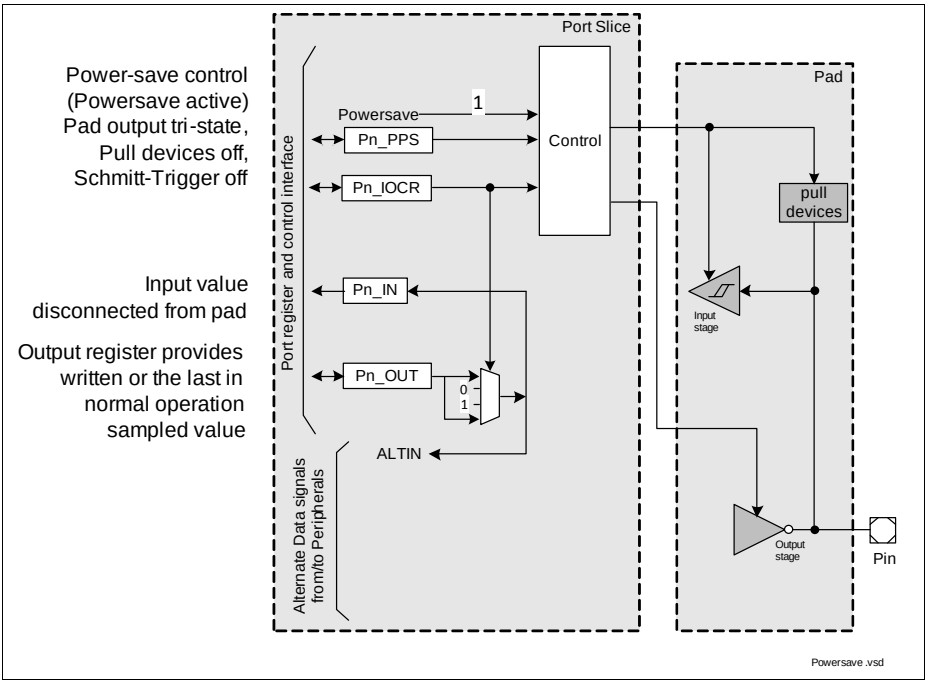


Figure 25-2 Port Pin in Power Save State

25.5 Analog Ports

P14 and P15 are analog and digital input ports with a simplified port and pad structure, see **Figure 25-3**. The analog pads have no output drivers and the digital input Schmitt-Trigger can be controlled by the Pn_PDISC (**Chapter 25.8.3**) register. Accordingly, the port control interface is reduced in its functionality. The Pn_IOCR register controls the pull devices, the optional input inversion and the input source in power-save mode. The Pn_OUT has only its power-save functionality, as described in **Chapter 25.4**.

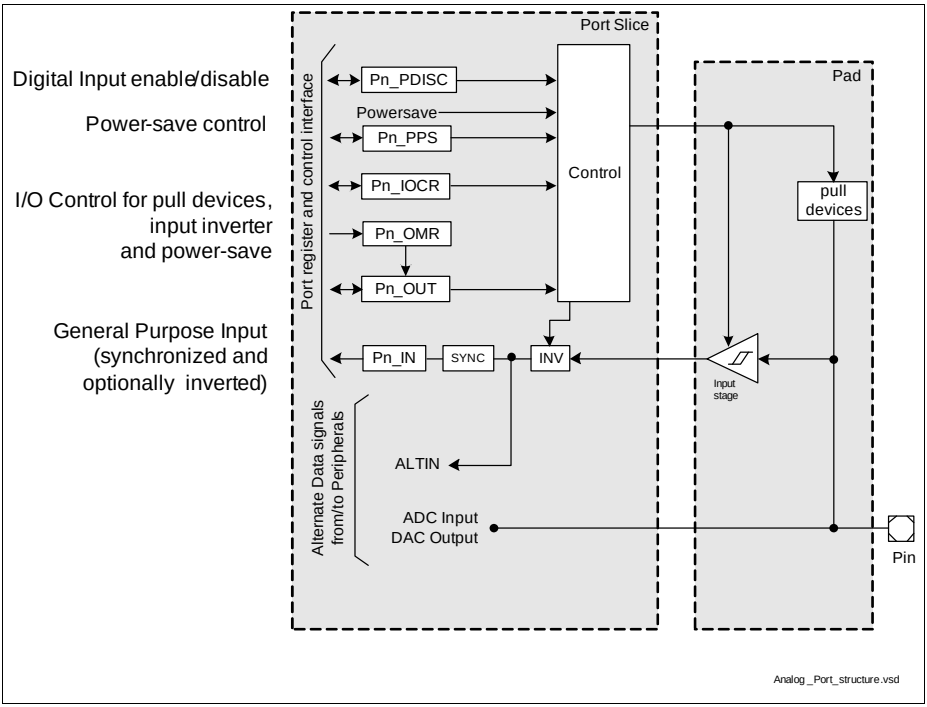


Figure 25-3 Analog Port Structure

25.6 Power, Reset and Clock

During power-up, until V_{DDC} and V_{DDP} voltage levels are stable and within defined limits, or during power-fail, while one/some voltage levels are outside the defined limits, the digital I/O pads are held in a defined state, which is tri-state input, output driver disabled and no pull devices active. The JTAG interface activates pull devices as default configuration.

The pins of the battery-buffered Hibernate Domain are independent to the supply monitoring of V_{DDC} and V_{DDP}, but are reset with the Standby Reset (see Reset Control chapter in the System Control Unit).

See the SCU Power Management chapter and the Data Sheet for details on the power-up, supply monitoring and voltage limits.

All Port registers are reset with the System Reset (see Reset Control chapter in the System Control Unit). The standard reset values are defined such that the port pins are configured as tri-state inputs, output driver disabled and no pull devices active.

Exceptions from these standard values are related to special interfaces (for example JTAG) or the analog input channels.

The Ports register interface is connected to Peripheral Bridge 1 (PBA1) and all registers of the Ports are clocked with f_{PERIPH}.

25.7 Initialization and System Dependencies

It is recommended to follow pre-defined routines for the initialization of the port pins.

Input

When a peripheral shall use a port pin as input, the actual pin levels may immediately trigger an unexpected peripheral event (e.g. clock edge at SPI). This can be avoided by forcing the "passive" level via pull-up/down programming.

The following steps are required to configure a port pin as an input:

- Pn_IOCR
  input configuration with pull device and/or power-save mode configuration
- Hardware Control (if applicable)
  - Pn_HWSEL
    switch hardware control to peripheral
- Pin Power Save (if applicable)
  - Pn_OMR/Pn_OUT
    default value in power save mode (if applicable)
  - Pn_PPS
    enable power save control

Output

When a port pin is configured as output for an on-chip peripheral, it is important that the peripheral is configured before the port switches the control to the peripheral in order to avoid spikes on the output.
The following steps are required to configure a port pin as an output:

- Pn_OMR/Pn_OUT
  Initial output value (as general purpose output)
- Pn_PDR
  Pad Driver Strength configuration
- GPIO or Alternate Output
  - Pn_IOCR
    Output multiplexer select
    Push-pull or open-drain output driver mode
    Activates the output driver!
- Hardware Control

– Pn_IOCR
depending on the hardware function Pn_IOCR can enable the internal pull devices
– Pn_HWSEL
Switch hardware control to peripheral

Transitions

If a port pin is used for different functions that require a reconfiguration of the port registers, it is recommended to do this transition via an intermediate “neutral” tri-state input configuration.

- Pn_HWSEL
disable hardware selection; can be omitted if no hardware control is used on the port pin
- Pn_PPS
disable power save mode control of the pin; can be ommited if no power save configuration is used on the port pin
- Pn_IOCR
tri-state input and no pull device active

25.8 Registers

Registers Overview

The absolute register address is calculated by adding:

Module Base Address + Offset Address

Table 25-2 Registers Address Space

Module	Base Address	End Address	Note
P0	$4802\ 8000_H$	$4802\ 80FF_H$	
P1	$4802\ 8100_H$	$4802\ 81FF_H$	
P2	$4802\ 8200_H$	$4802\ 82FF_H$	
P3	$4802\ 8300_H$	$4802\ 83FF_H$	
P4	$4802\ 8400_H$	$4802\ 84FF_H$	
P5	$4802\ 8500_H$	$4802\ 85FF_H$	
P6	$4802\ 8600_H$	$4802\ 86FF_H$	
P14	$4802\ 8E00_H$	$4802\ 8EFF_H$	Analog/Digital Input only
P15	$4802\ 8F00_H$	$4802\ 8FFF_H$	Analog/Digital Input only

Table 25-3 Register Overview

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
Pn_OUT	Port n Output Register	0000_H	U, PV	U, PV	Page 25-30
Pn_OMR	Port n Output Modification Register	0004_H	U, PV	U, PV	Page 25-31
–	Reserved	0008_H-$000C_H$	BE	BE	–
Pn_IOCR0	Port n Input/Output Control Register 0	0010_H	U, PV	PV	Page 25-14
Pn_IOCR4	Port n Input/Output Control Register 4	0014_H	U, PV	PV	Page 25-15
Pn_IOCR8	Port n Input/Output Control Register 8	0018_H	U, PV	PV	Page 25-15
Pn_IOCR12	Port n Input/Output Control Register 12	$001C_H$	U, PV	PV	Page 25-16
–	Reserved	0020_H	BE	BE	–
Pn_IN	Port n Input Register	0024_H	U, PV	R	Page 25-32
–	Reserved	0028_H-$003C_H$	BE	BE	–
Pn_PDR0	Port n Pad Driver Mode 0 Register	0040_H	U, PV	PV	Page 25-20
Pn_PDR1	Port n Pad Driver Mode 1 Register	0044_H	U, PV	PV	Page 25-21
–	Reserved	0048_H-$005C_H$	BE	BE	–
Pn_PDISC	Port n Pin Function Decision Control Register (non-ADC ports)	0060_H	U, PV	BE	Page 25-22
P14_PDISC P15_PDISC	Port n Pin Function Decision Control Register (ADC ports)	0060_H	U, PV	PV	Page 25-23 Page 25-26
–	Reserved	0064_H-$006C_H$	BE	BE	–
Pn_PPS	Port n Pin Power Save Register	0070_H	U, PV	PV	Page 25-33

Table 25-3 Register Overview (cont'd)

Short Name	Description	Offset Addr.	Access Mode		Description See
			Read	Write	
Pn_HWSEL	Port n Hardware Select Register	0074_H	U, PV	PV	Page 25-34
–	Reserved	0078_H-$00FC_H$	BE	BE	–

Table 25-4 Registers Access Rights and Reset Classes

Register Short Name	Access Rights		Reset Class
	Read	Write	
Pn_IN	U, PV	R	System Reset
Pn_OUT		U, PV	
Pn_OMR			
Pn_IOCR0		PV	
Pn_IOCR4			
Pn_IOCR8			
Pn_IOCR12			
Pn_PDISC (ADC ports)			
Pn_PDR0			
Pn_PDR1			
Pn_PPS			
Pn_HWSEL			
Pn_PDISC (non-ADC ports)		BE	

25.8.1 Port Input/Output Control Registers

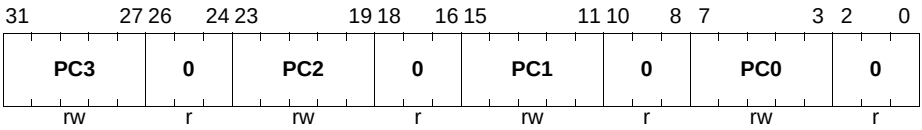
The port input/output control registers select the digital output and input driver functionality and characteristics of a GPIO port pin. Port direction (input or output), pull-up or pull-down devices for inputs, and push-pull or open-drain functionality for outputs can be selected by the corresponding bit fields PCx (x = 0-15). Each 32-bit wide port input/output control register controls four GPIO port lines:

Register Pn_IOCR0 controls the Pn.[3:0] port lines
Register Pn_IOCR4 controls the Pn.[7:4] port lines
Register Pn_IOCR8 controls the Pn.[11:8] port lines
Register Pn_IOCR12 controls the Pn.[15:12] port lines

The diagrams below show the register layouts of the port input/output control registers with the PCx bit fields. One PCx bit field controls exactly one port line Pn.x.

Pn_IOCR0 (n=0-6)
Port n Input/Output Control Register 0
($4802\ 8010_H + n*100_H$) **Reset Value: $0000\ 0000_H$**

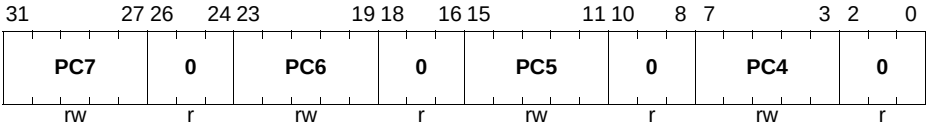
Pn_IOCR0 (n=14-15)
Port n Input/Output Control Register 0
($4802\ 8010_H + n*100_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
PC0, PC1, PC2, PC3	[7:3], [15:11], [23:19], [31:27]	rw	**Port Control for Port n Pin 0 to 3** This bit field determines the Port n line x functionality (x = 0-3) according to the coding table (see **Table 25-5**).
0	[2:0], [10:8], [18:16], [26:24]	r	**Reserved** Read as 0; should be written with 0.

Pn_IOCR4 (n=0-6)
Port n Input/Output Control Register 4
($4802\ 8014_H + n*100_H$) **Reset Value: $0000\ 0000_H$**

Pn_IOCR4 (n=14-15)
Port n Input/Output Control Register 4
($4802\ 8014_H + n*100_H$) **Reset Value: $0000\ 0000_H$**

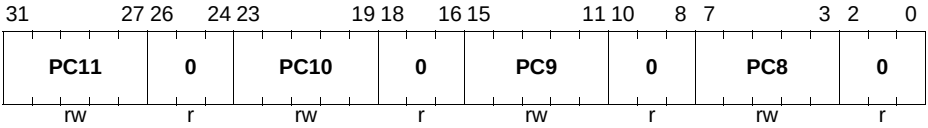


Field	Bits	Type	Description
PC4, PC5, PC6, PC7	[7:3], [15:11], [23:19], [31:27]	rw	**Port Control for Port n Pin 4 to 7** This bit field determines the Port n line x functionality (x = 4-7) according to the coding table (see **Table 25-5**).
0	[2:0], [10:8], [18:16], [26:24]	r	**Reserved** Read as 0; should be written with 0.

Pn_IOCR8 (n=0-3)
Port n Input/Output Control Register 8
($4802\ 8018_H + n*100_H$) **Reset Value: $0000\ 0000_H$**

P5_IOCR8
Port 5 Input/Output Control Register 8
(0018_H) **Reset Value: $0000\ 0000_H$**

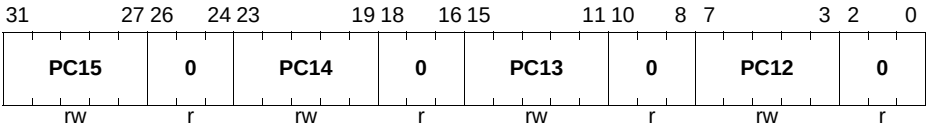
Pn_IOCR8 (n=14-15)
Port n Input/Output Control Register 8
($4802\ 8018_H + n*100_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
PC8, PC9, PC10, PC11	[7:3], [15:11], [23:19], [31:27]	rw	**Port Control for Port n Pin 8 to 11** This bit field determines the Port n line x functionality (x = 8-11) according to the coding table (see **Table 25-5**).
0	[2:0], [10:8], [18:16], [26:24]	r	**Reserved** Read as 0; should be written with 0.

Pn_IOCR12 (n=0-3)
Port n Input/Output Control Register 12
(4802 801C$_H$ + n*100$_H$) **Reset Value: 0000 0000$_H$**

Pn_IOCR12 (n=14-15)
Port n Input/Output Control Register 12
(4802 801C$_H$ + n*100$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
PC12, PC13, PC14, PC15	[7:3], [15:11], [23:19], [31:27]	rw	**Port Control for Port n Pin 12 to 15** This bit field determines the Port n line x functionality (x = 12-15) according to the coding table (see **Table 25-5**).
0	[2:0], [10:8], [18:16], [26:24]	r	**Reserved** Read as 0; should be written with 0.

Depending on the GPIO port functionality (number of GPIO lines of a port), not all of the port input/output control registers are implemented.

The structure with one control bit field for each port pin located in different register bytes offers the possibility to configure the port pin functionality of a single pin with byte-oriented accesses without accessing the other PCx bit fields.

Port Control Coding

Table 25-5 describes the coding of the PCx bit fields that determine the port line functionality.

The Pn_IOCRy PCx bit field is also used to contol the pin behavior in Deep-Sleep mode if the Pin Power Save option is enabled, see **Chapter 25.8.7**.

Table 25-5 Standard PCx Coding[1)]

PCx[4:0]	I/O	Output Characteristics	Selected Pull-up / Pull-down / Selected Output Function
$0X000_B$	Direct Input	–	No input pull device connected
$0X001_B$			Input pull-down device connected
$0X010_B$			Input pull-up device connected
$0X011_B$			No input pull device connected; Pn_OUTx continuously samples the input value
$0X100_B$	Inverted Input	–	No input pull device connected
$0X101_B$			Input pull-down device connected
$0X110_B$			Input pull-up device connected
$0X111_B$			No input pull device connected; Pn_OUTx continuously samples the input value

Table 25-5 Standard PCx Coding[1)] (cont'd)

PCx[4:0]	I/O	Output Characteristics	Selected Pull-up / Pull-down / Selected Output Function
10000_B	Output (Direct Input)	Push-pull	General-purpose output
10001_B			Alternate output function 1
10010_B			Alternate output function 2
10011_B			Alternate output function 3
10100_B			Alternate output function 4
10101_B			Reserved.
10110_B			Reserved.
10111_B			Reserved.
11000_B		Open-drain	General-purpose output
11001_B			Alternate output function 1
11010_B			Alternate output function 2
11011_B			Alternate output function 3
11100_B			Alternate output function 4
11101_B			Reserved.
11110_B			Reserved.
11111_B			Reserved.

1) For the analog and digital input ports P14 and P15 the combinations with PCx[4]=1_B are reserved.

25.8.2 Pad Driver Mode Register

The pad structure of the XMC4500 GPIO lines offers the possibility to select the output driver strength and the slew rate. These two parameters are controlled by the bit fields in the pad driver mode registers Pn_PDR0/1, independently from input/output and pull-up/pull-down control functionality as programmed in the Pn_IOCR register. Pn_PDR0 and Pn_PDR1 registers are assigned to each port.

Depending on the assigned pad class, the 3-bit wide pad driver mode selection bit fields PDx in the pad driver mode registers Pn_PDR make it possible to select the port line functionality as shown in **Table 25-6**. Note that the pad driver mode registers are specific for each port.

Table 25-6 Pad Driver Mode Selection

Pad Class	PDx.2	PDx.1	PDx.0	Functionality
A1	X	X	0	Medium driver
			1	Weak driver
A1+	0	X	0	Strong driver soft edge
	0	X	1	Strong driver slow edge
	1	X	0	Medium driver
	1	X	1	Weak driver
A2	0	0	0	Strong driver, sharp edge
	0	0	1	Strong driver, medium edge
	0	1	0	Strong driver, soft edge
	0	1	1	Reserved
	1	0	0	Medium driver
	1	0	1	
	1	1	0	Reserved
	1	1	1	Weak driver

Note: The XMC4500 Data Sheet describes the DC characteristics of all pad classes.

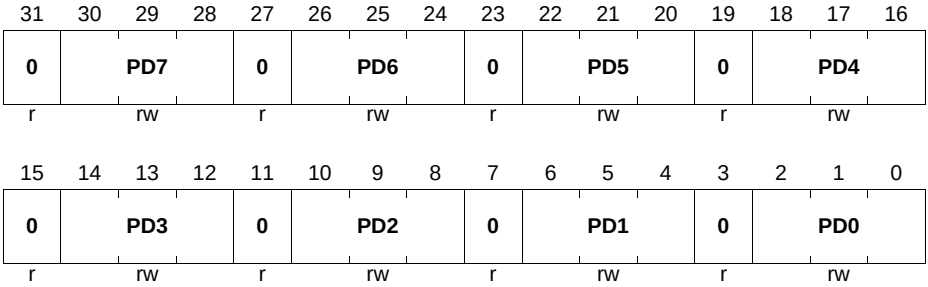
Pad Driver Mode Registers

This is the general description of the PDR registers. Each port contains its own specific PDR registers, described additionally at each port, that can contain between one and eight PDx fields for PDR0 and PDR1 registers, respectively. Each field controls 1 pin. For coding of PDx, see **Table 25-6**.

The analog and digital input ports P14 and P15 don’t have Pn_PDR registers.

Pn_PDR0 (n=0-6)

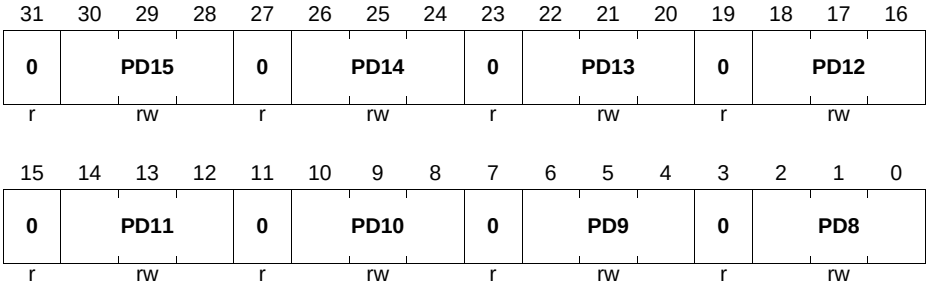
Port n Pad Driver Mode 0 Register(4802 8040$_H$ + n*100$_H$) Reset Value: 2222 2222$_H$



Field	Bits	Type	Description
PD0	[2:0]	rw	**Pad Driver Mode for Pn.0**
PD1	[6:4]	rw	**Pad Driver Mode for Pn.1**
PD2	[10:8]	rw	**Pad Driver Mode for Pn.2**
PD3	[14:12]	rw	**Pad Driver Mode for Pn.3**
PD4	[18:16]	rw	**Pad Driver Mode for Pn.4**
PD5	[22:20]	rw	**Pad Driver Mode for Pn.5**
PD6	[26:24]	rw	**Pad Driver Mode for Pn.6**
PD7	[30:28]	rw	**Pad Driver Mode for Pn.7**
0	3, 7, 11, 15, 19, 23, 27, 31	r	**Reserved** Read as 0; should be written with 0.

Pn_PDR1 (n=0-3)
Port n Pad Driver Mode 1 Register
(4802 8044$_H$ + n*100$_H$) **Reset Value: 2222 2222$_H$**

P5_PDR1
Port 5 Pad Driver Mode 1 Register (0044$_H$) **Reset Value: 2222 2222$_H$**



Field	Bits	Type	Description
PD8	[2:0]	rw	**Pad Driver Mode for Pn.8**
PD9	[6:4]	rw	**Pad Driver Mode for Pn.9**
PD10	[10:8]	rw	**Pad Driver Mode for Pn.10**
PD11	[14:12]	rw	**Pad Driver Mode for Pn.11**
PD12	[18:16]	rw	**Pad Driver Mode for Pn.12**
PD13	[22:20]	rw	**Pad Driver Mode for Pn.13**
PD14	[26:24]	rw	**Pad Driver Mode for Pn.14**
PD15	[30:28]	rw	**Pad Driver Mode for Pn.15**
0	3, 7, 11, 15, 19, 23, 27, 31	r	**Reserved** Read as 0; should be written with 0.

25.8.3 Pin Function Decision Control Register

Pin Function Decision Control Register

The primary use for this register is to disable/enable the digital pad structure in shared analog and digital ports, see the dedicated description for **P14_PDISC** and **P15_PDISC**.

For “normal” digital I/O ports (P0-P6) this register is read-only and the read value corresponds to the available pins in the given package.

Pn_PDISC (n=0-6)
Port n Pin Function Decision Control Register
($4802\ 8060_H + n*100_H$) Reset Value: XXXX $XXXX_H$[1)]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PDIS 15	**PDIS 14**	**PDIS 13**	**PDIS 12**	**PDIS 11**	**PDIS 10**	**PDIS 9**	**PDIS 8**	**PDIS 7**	**PDIS 6**	**PDIS 5**	**PDIS 4**	**PDIS 3**	**PDIS 2**	**PDIS 1**	**PDIS 0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

1) The reset value is package dependent.

Field	Bits	Type	Description
PDISx (x = 0-15)	x	r	**Pad Disable for Port n Pin x** 0_B Pad Pn.x is enabled. 1_B Pad Pn.x is disabled.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

P14_PDISC
Port 14 Pin Function Decision Control Register
(0060_H) **Reset Value: 0000 F3FF$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PDIS15	PDIS14	PDIS13	PDIS12	0	0	PDIS9	PDIS8	PDIS7	PDIS6	PDIS5	PDIS4	PDIS3	PDIS2	PDIS1	PDIS0
rw	rw	rw	rw	r	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
PDIS0	0	rw	**Pad Disable for Port 14 Pin 0** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 0 analog input 0 selected.
PDIS1	1	rw	**Pad Disable for Port 14 Pin 1** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 0 analog input 1 selected.
PDIS2	2	rw	**Pad Disable for Port 14 Pin 2** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 0 and ADC 1 analog input 2 selected.

Field	Bits	Type	Description
PDIS3	3	rw	**Pad Disable for Port 14 Pin 3** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 0 and ADC 1 analog input 3 selected.
PDIS4	4	rw	**Pad Disable for Port 14 Pin 4** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 0 analog input 4 and ADC 2 analog input 0 and DAC Reference selected.
PDIS5	5	rw	**Pad Disable for Port 14 Pin 5** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 0 analog input 5 and ADC 2 analog input 1 selected.
PDIS6	6	rw	**Pad Disable for Port 14 Pin 6** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 0 analog input 6 selected.

Field	Bits	Type	Description
PDIS7	7	rw	**Pad Disable for Port 14 Pin 7** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC0 analog input 7 selected.
PDIS8	8	rw	**Pad Disable for Port 14 Pin 8** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 1 analog input 0 and ADC 2 analog input 4 and DAC output 0 selected.
PDIS9	9	rw	**Pad Disable for Port 14 Pin 9** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 1 analog input 1 and ADC 2 analog input 5 and DAC output 1 selected.
PDIS12	12	rw	**Pad Disable for Port 14 Pin 12** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 1 analog input 4 selected.

Field	Bits	Type	Description
PDIS13	13	rw	**Pad Disable for Port 14 Pin 13** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 1 analog input 5 selected.
PDIS14	14	rw	**Pad Disable for Port 14 Pin 14** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 1 analog input 6 selected.
PDIS15	15	rw	**Pad Disable for Port 14 Pin 15** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 1 analog input 7 selected.
0	10, 11, [31:16]	r	**Reserved** Read as 0; should be written with 0.

P15_PDISC
Port 15 Pin Function Decision Control Register

(0060_H) **Reset Value: 0000 F3FC$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PDIS15	PDIS14	PDIS13	PDIS12	0		PDIS9	PDIS8	PDIS7	PDIS6	PDIS5	PDIS4	PDIS3	PDIS2	0	
rw	rw	rw	rw	r		rw	rw	rw	rw	rw	rw	rw	rw	r	

Field	Bits	Type	Description
PDIS2	2	rw	**Pad Disable for Port 15 Pin 2** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 2 analog input 2.
PDIS3	3	rw	**Pad Disable for Port 15 Pin 3** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 2 analog input 3.
PDIS4	4	rw	**Pad Disable for Port 15 Pin 4** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 2 analog input 4.
PDIS5	5	rw	**Pad Disable for Port 15 Pin 5** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 2 analog input 5.
PDIS6	6	rw	**Pad Disable for Port 15 Pin 6** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 2 analog input 6.

Field	Bits	Type	Description
PDIS7	7	rw	**Pad Disable for Port 15 Pin 7** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 2 analog input 7.
PDIS8	8	rw	**Pad Disable for Port 15 Pin 8** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 3 analog input 0.
PDIS9	9	rw	**Pad Disable for Port 15 Pin 9** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 3 analog input 1.
PDIS12	12	rw	**Pad Disable for Port 15 Pin 12** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 3 analog input 4.
PDIS13	13	rw	**Pad Disable for Port 15 Pin 13** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 3 analog input 5.

Field	Bits	Type	Description
PDIS14	14	rw	**Pad Disable for Port 15 Pin 14** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 3 analog input 6.
PDIS15	15	rw	**Pad Disable for Port 15 Pin 15** This bit disables or enables the digital pad function. The disabled (default) state of the pad selects the ADC analog input. Upon enabling, the functionality switches to the digital input. 0_B Pad is enabled, digital input selected. 1_B Pad is disabled, ADC 3 analog input 7.
0	0, 1, 10, 11, [31:16]	r	**Reserved** Read as 0; should be written with 0.

25.8.4 Port Output Register

The port output register determines the value of a GPIO pin when it is selected by Pn_IOCRx as output. Writing a 0 to a Pn_OUT.Px (x = 0-15) bit position delivers a low level at the corresponding output pin. A high level is output when the corresponding bit is written with a 1. Note that the bits of Pn_OUT.Px can be individually set/reset by writing appropriate values into the port output modification register Pn_OMR, avoiding read-modify-write operations on the Pn_OUT, which might affect other pins of the port.

The Pn_OUT is also used to store/drive a defined value for the input in Deep-Sleep mode. For details on this see the **Port Pin Power Save Register**. That is also the only use of the Pn_OUT register in the analog and digital input ports P14 and P15.

Pn_OUT (n=0-6)
Port n Output Register **(4802 8000$_H$ + n*100$_H$)** **Reset Value: 0000 0000$_H$**
Pn_OUT (n=14-15)
Port n Output Register **(4802 8000$_H$ + n*100$_H$)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
Px **(x = 0-15)**	x	rwh	**Port n Output Bit x** This bit determines the level at the output pin Pn.x if the output is selected as GPIO output. 0_B The output level of Pn.x is 0. 1_B The output level of Pn.x is 1. Pn.x can also be set/reset by control bits of the Pn_OMR register.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

25.8.5 Port Output Modification Register

The port output modification register contains control bits that make it possible to individually set, reset, or toggle the logic state of a single port line by manipulating the output register.

Pn_OMR (n=0-6)
Port n Output Modification Register
(4802 8004$_H$ + n*100$_H$) **Reset Value: 0000 0000$_H$**

Pn_OMR (n=14-15)
Port n Output Modification Register
(4802 8004$_H$ + n*100$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PR 15	**PR 14**	**PR 13**	**PR 12**	**PR 11**	**PR 10**	**PR 9**	**PR 8**	**PR 7**	**PR 6**	**PR 5**	**PR 4**	**PR 3**	**PR 2**	**PR 1**	**PR 0**
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PS 15	**PS 14**	**PS 13**	**PS 12**	**PS 11**	**PS 10**	**PS 9**	**PS 8**	**PS 7**	**PS 6**	**PS 5**	**PS 4**	**PS 3**	**PS 2**	**PS 1**	**PS 0**
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
PSx (x = 0-15)	x	w	**Port n Set Bit x** Setting this bit will set or toggle the corresponding bit in the port output register Pn_OUT. The function of this bit is shown in **Table 25-7**.
PRx (x = 0-15)	x + 16	w	**Port n Reset Bit x** Setting this bit will reset or toggle the corresponding bit in the port output register Pn_OUT. The function of this bit is shown in **Table 25-7**.

Note: Register Pn_OMR is virtual and does not contain any flip-flop. A read action delivers the value of 0. A 8 or 16-bits write behaves like a 32-bit write padded with zeros.

Table 25-7 Function of the Bits PRx and PSx

PRx	PSx	Function
0	0	Bit Pn_OUT.Px is not changed.
0	1	Bit Pn_OUT.Px is set.
1	0	Bit Pn_OUT.Px is reset.
1	1	Bit Pn_OUT.Px is toggled.

25.8.6 Port Input Register

The logic level of a GPIO pin can be read via the read-only port input register Pn_IN. Reading the Pn_IN register always returns the current logical value at the GPIO pin, synchronized to avoid meta-stabilities, independently whether the pin is selected as input or output.

Pn_IN (n=0-6)
Port n Input Register **($4802\ 8024_H + n*100_H$)** **Reset Value: $0000\ XXXX_H$**
Pn_IN (n=14-15)
Port n Input Register **($4802\ 8024_H + n*100_H$)** **Reset Value: $0000\ XXXX_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P15	**P14**	**P13**	**P12**	**P11**	**P10**	**P9**	**P8**	**P7**	**P6**	**P5**	**P4**	**P3**	**P2**	**P1**	**P0**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
Px **(x = 0-15)**	x	rh	**Port n Input Bit x** This bit indicates the level at the input pin Pn.x. 0_B The input level of Pn.x is 0. 1_B The input level of Pn.x is 1.
0	[31:16]	r	**Reserved** Read as 0.

25.8.7 Port Pin Power Save Register

When the XMC4500 enters Deep-Sleep mode, pins with enabled Pin Power Save option are set to a defined state and the input Schmitt-Trigger as well as the output driver stage are switched off.

Pn_PPS (n=0-6)
Port n Pin Power Save Register
($4802\ 8070_H + n*100_H$) **Reset Value: $0000\ 0000_H$**

Pn_PPS (n=14-15)
Port n Pin Power Save Register
($4802\ 8070_H + n*100_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PPS 15	**PPS 14**	**PPS 13**	**PPS 12**	**PPS 11**	**PPS 10**	**PPS 9**	**PPS 8**	**PPS 7**	**PPS 6**	**PPS 5**	**PPS 4**	**PPS 3**	**PPS 2**	**PPS 1**	**PPS 0**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
PPSx (x = 0-15)	x	rw	**Port n Pin Power Save Bit x** 0_B Pin Power Save of Pn.x is disabled. 1_B Pin Power Save of Pn.x is enabled.
0	[31:16]	r	**Reserved** Read as 0.

Deep-Sleep Pin Power Save behavior

The actual behavior in Deep-Sleep mode with enabled Pin Power Save is controlled by the Pn_IOCRy.PCx bit field (**Page 25-14**) of the respective pin. **Table 25-8** shows the coding.

Table 25-8 PCx Coding in Deep-Sleep mode

PCx[4:0]	I/O	Normal Operation or PPSx=0_B	Deep-Sleep mode and PPSx=1_B
$0X000_B$	Direct Input	See **Table 25-5**	Input value=Pn_OUTx
$0X001_B$			Input value=0_B; pull-down deactivated
$0X010_B$			Input value=1_B; pull-up deactivated
$0X011_B$			Input value=Pn_OUTx, storing the last sampled input value
$0X100_B$	Inverted Input	See **Table 25-5**	Input value=$\overline{\text{Pn\_OUTx}}$
$0X101_B$			Input value=1_B; pull-down deactivated
$0X110_B$			Input value=0_B; pull-up deactivated
$0X111_B$			Input value=$\overline{\text{Pn\_OUTx}}$, storing the last sampled input value
$1XXXX_B$	Output	See **Table 25-5**	Output driver off, Input Schmitt-Trigger off, no pull device active, Input value=Pn_OUTx

25.8.8 Port Pin Hardware Select Register

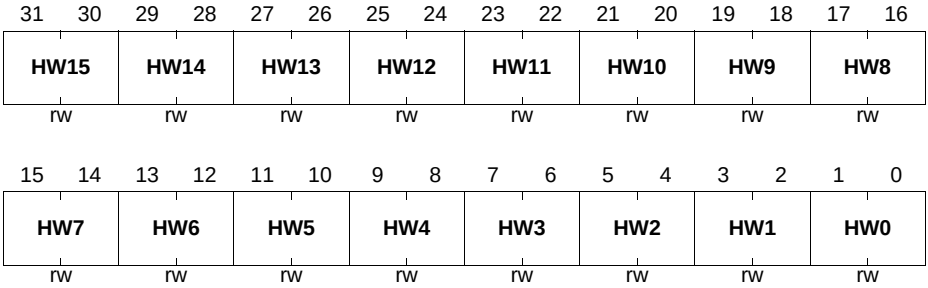
Some peripherals require direct hardware control of their I/Os. As on some pins multiple such peripheral I/Os are mapped, the register Pn_HWSEL is used to select which peripheral has the control over the pin.

Note: Pn_HWSEL.HWx just pre-assigns the hardware-control of the pin to a certain peripheral, but the peripheral itself decides to actually take control over it. As long as the peripheral does not take control of a given pin via HWx_EN, the configuration of this pin is still defined by the configuration registers and it is available as GPIO or for other alternate functions. This might be because the selected peripheral has controls to just activate a subset of its pins, or because the peripheral is not active at all.

This mechanism can also be used to prohibit the hardware control of certain pins to a peripheral, in case the application does not need the respective functionality and the peripheral has no controls to disable the hardware control selectively.

The shared analog and digital input ports P14 and P15 do not support the hardware select feature.

P0_HWSEL
Port 0 Pin Hardware Select Register (0074$_H$) Reset Value: 0001 4000$_H$
P1_HWSEL
Port 1 Pin Hardware Select Register (0074$_H$) Reset Value: 0000 0000$_H$
P2_HWSEL
Port 2 Pin Hardware Select Register (0074$_H$) Reset Value: 0000 0004$_H$
P3_HWSEL
Port 3 Pin Hardware Select Register (0074$_H$) Reset Value: 0000 0000$_H$
P4_HWSEL
Port 4 Pin Hardware Select Register (0074$_H$) Reset Value: 0000 0000$_H$
P5_HWSEL
Port 5 Pin Hardware Select Register (0074$_H$) Reset Value: 0000 0000$_H$
P6_HWSEL
Port 6 Pin Hardware Select Register (0074$_H$) Reset Value: 0000 0000$_H$
P14_HWSEL
Port 14 Pin Hardware Select Register (0074$_H$) Reset Value: 0000 0000$_H$
P15_HWSEL
Port 15 Pin Hardware Select Register (0074$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
HWx (x = 0-15)	[2*x+1: 2*x]	rw	**Port n Pin Hardware Select Bit x** 00_B Software control only. 01_B HW0 control path can override the software configuration. 10_B HW1 control path can override the software configuration. 11_B Reserved.

25.9 Package Pin Summary

The following general building block is used to describe each pin:

Table 25-9 Package Pin Mapping Description

Function	Package A	Package B	...	Pad Type	Notes
Name	N	Ax	...	A2	

The table is sorted by the “Function” column, starting with the regular Port pins (Px.y), followed by the dedicated pins (i.e. PORST) and supply pins.

The following columns, titled with the supported package variants, lists the package pin number to which the respective function is mapped in that package.

The “Pad Type” indicates the employed pad type (A1, A1+, A2, special=special pad, In=input pad, AN/DIG IN=analog and digital input, Power=power supply). Details about the pad properties are defined in the Data Sheet.

In the “Notes”, special information to the respective pin/function is given, i.e. deviations from the default configuration after reset.

Table 25-10 Package Pin Mapping

Function	LQFP 144	LFBGA 144	LQFP 100	Pad Type	Notes
P0.0	2	C4	2	A1+	
P0.0	2	C4	2	A1+	
P0.1	1	C3	1	A1+	
P0.2	144	A3	100	A2	
P0.3	143	A4	99	A2	
P0.4	142	B5	98	A2	
P0.5	141	A5	97	A2	
P0.6	140	A6	96	A2	
P0.7	128	B7	89	A2	After a system reset, this pin selects HW0.
P0.8	127	A8	88	A2	After a system reset, this pin selects HW0 with a weak pull-down active.
P0.9	4	D4	4	A2	
P0.10	3	B4	3	A1+	
P0.11	139	E5	95	A1+	

Table 25-10 Package Pin Mapping (cont'd)

Function	LQFP 144	LFBGA 144	LQFP 100	Pad Type	Notes
P0.12	138	D5	94	A1+	
P0.13	137	C5	-	A1+	
P0.14	136	E6	-	A1+	
P0.15	135	C6	-	A1+	
P1.0	112	D9	79	A1+	
P1.1	111	E9	78	A1+	
P1.2	110	C11	77	A2	
P1.3	109	C12	76	A2	
P1.4	108	C10	75	A1+	
P1.5	107	D10	74	A1+	
P1.6	116	B9	83	A2	
P1.7	115	B10	82	A2	
P1.8	114	A10	81	A2	
P1.9	113	B11	80	A2	
P1.10	106	D12	73	A1+	
P1.11	105	D11	72	A1+	
P1.12	104	E11	71	A2	
P1.13	103	E12	70	A2	
P1.14	102	E10	69	A2	
P1.15	94	G12	68	A2	
P2.0	74	J11	52	A2	
P2.1	73	K12	51	A2	After a system reset, this pin selects HW0.
P2.2	72	K11	50	A2	
P2.3	71	L11	49	A2	
P2.4	70	L10	48	A2	
P2.5	69	M10	47	A2	
P2.6	76	J9	54	A1+	
P2.7	75	K9	53	A1+	
P2.8	68	L9	46	A2	
P2.9	67	M9	45	A2	
P2.10	66	L8	44	A2	

Table 25-10 Package Pin Mapping (cont'd)

Function	LQFP 144	LFBGA 144	LQFP 100	Pad Type	Notes
P2.11	65	M8	-	A2	
P2.12	64	L7	-	A2	
P2.13	63	M7	-	A2	
P2.14	60	K7	41	A2	
P2.15	59	J6	40	A2	
P3.0	7	C1	7	A2	
P3.1	6	B2	6	A2	
P3.2	5	B3	5	A2	
P3.3	132	F7	93	A1+	
P3.4	131	E7	92	A1+	
P3.5	130	B6	91	A2	
P3.6	129	A7	90	A2	
P3.7	14	E4	-	A1+	
P3.8	13	E3	-	A1+	
P3.9	12	F5	-	A1+	
P3.10	11	F6	-	A1+	
P3.11	10	D3	-	A1+	
P3.12	9	D2	-	A2	
P3.13	8	C2	-	A2	
P3.14	134	D6	-	A1+	
P3.15	133	D7	-	A1+	
P4.0	124	B8	85	A2	
P4.1	123	A9	84	A2	
P4.2	122	E8	-	A1+	
P4.3	121	F8	-	A1+	
P4.4	120	C7	-	A1+	
P4.5	119	D8	-	A1+	
P4.6	118	C8	-	A1+	
P4.7	117	C9	-	A1+	
P5.0	84	H9	58	A1+	
P5.1	83	H8	57	A1+	
P5.2	82	H7	56	A1+	

Table 25-10 Package Pin Mapping (cont'd)

Function	LQFP 144	LFBGA 144	LQFP 100	Pad Type	Notes
P5.3	81	J10	-	A2	
P5.4	80	K10	-	A2	
P5.5	79	J8	-	A2	
P5.6	78	K8	-	A2	
P5.7	77	J7	55	A1+	
P5.8	58	H6	-	A2	
P5.9	57	K6	-	A2	
P5.10	56	H5	-	A1+	
P5.11	55	J5	-	A1+	
P6.0	101	G10	-	A2	
P6.1	100	F9	-	A2	
P6.2	99	H10	-	A2	
P6.3	98	G9	-	A1+	
P6.4	97	F10	-	A2	
P6.5	96	F11	-	A2	
P6.6	95	F12	-	A2	
P14.0	42	L3	31	AN/DIG_IN	
P14.1	41	L2	30	AN/DIG_IN	
P14.2	40	K3	29	AN/DIG_IN	
P14.3	39	J4	28	AN/DIG_IN	
P14.4	38	K1	27	AN/DIG_IN	
P14.5	37	K2	26	AN/DIG_IN	
P14.6	36	J3	25	AN/DIG_IN	
P14.7	35	J2	24	AN/DIG_IN	
P14.8	52	M5	37	AN/DAC/DIG_IN	
P14.9	51	L5	36	AN/DAC/DIG_IN	
P14.12	34	J1	23	AN/DIG_IN	
P14.13	33	H4	22	AN/DIG_IN	
P14.14	32	H3	21	AN/DIG_IN	
P14.15	31	H2	20	AN/DIG_IN	
P15.2	30	H1	19	AN/DIG_IN	

Table 25-10 Package Pin Mapping (cont'd)

Function	LQFP 144	LFBGA 144	LQFP 100	Pad Type	Notes
P15.3	29	G2	18	AN/DIG_IN	
P15.4	28	G4	-	AN/DIG_IN	
P15.5	27	G3	-	AN/DIG_IN	
P15.6	26	G5	-	AN/DIG_IN	
P15.7	25	G6	-	AN/DIG_IN	
P15.8	54	M6	39	AN/DIG_IN	
P15.9	53	L6	38	AN/DIG_IN	
P15.12	50	K5	-	AN/DIG_IN	
P15.13	49	M4	-	AN/DIG_IN	
P15.14	44	L4	-	AN/DIG_IN	
P15.15	43	K4	-	AN/DIG_IN	
USB_DP	16	E1	9	special	
USB_DM	15	D1	8	special	
HIB_IO_0	21	F4	14	A1 special	At the first power-up and with every reset of the hibernate domain this pin is configured as open-drain output and drives "0".
HIB_IO_1	20	F3	13	A1 special	At the first power-up and with every reset of the hibernate domain this pin is configured as input with no pull device active.
TCK	93	G8	67	A1	Weak pull-down active.
TMS	92	G7	66	A1+	Weak pull-up active.
$\overline{\text{PORST}}$	91	G11	65	special	Weak pull-up permanently active, strong pull-down controlled by EVR.
XTAL1	87	H11	61	clock_IN	
XTAL2	88	H12	62	clock_O	
RTC_XTAL1	22	F2	15	clock_IN	
RTC_XTAL2	23	F1	16	clock_O	
VBAT	24	G1	17	Power	
VBUS	17	E2	10	special	
VAREF	46	M3	33	AN_Ref	

Table 25-10 Package Pin Mapping (cont'd)

Function	LQFP 144	LFBGA 144	LQFP 100	Pad Type	Notes
VAGND	45	M2	32	AN_Ref	
VDDA	48	L1	35	AN_Power	
VSSA	47	M1	34	AN_Power	
VDDC	19	-	12	Power	
VDDC	61	-	42	Power	
VDDC	90	-	64	Power	
VDDC	125	-	86	Power	
VDDC	-	A2	-	Power	
VDDC	-	B12	-	Power	
VDDC	-	M11	-	Power	
VDDP	18	-	11	Power	
VDDP	62	-	43	Power	
VDDP	86	-	60	Power	
VDDP	126	-	87	Power	
VDDP	-	A11	-	Power	
VDDP	-	B1	-	Power	
VDDP	-	L12	-	Power	
VSS	85	-	59	Power	
VSS	-	A1	-	Power	
VSS	-	A12	-	Power	
VSS	-	M12	-	Power	
VSSO	89	J12	63	Power	
VSS	Exp. Pad	-	Exp. Pad	Power	**Exposed Die Pad** The exposed die pad is connected internally to VSS. For proper operation, it is mandatory to connect the exposed pad to the board ground. For thermal aspects, please refer to the Data Sheet. Board layout examples are given in an application note.

25.10 Port I/O Functions

The following general building block is used to describe each PORT pin:

Table 25-11 Port I/O Function Description

Function	Outputs			Inputs		
	ALT1	ALTn	HWO0	HWI0	Input	Input
P0.0		MODA.OUT	MODB.OUT	MODB.INA	MODC.INA	
Pn.y	MODA.OUT				MODA.INA	MODC.INB

Pn.y is the port pin name, defining the control and data bits/registers associated with it. As GPIO, the port is under software control. Its input value is read via Pn_IN.y, Pn_OUT defines the output value.

Up to four alternate output functions (ALT1/2/3/4) can be mapped to a single port pin, selected by Pn_IOCR.PC. The output value is directly driven by the respective module, with the pin characteristics controlled by the port registers (within the limits of the connected pad).

The port pin input can be connected to multiple peripherals. Most peripherals have an input multiplexer to select between different possible input sources.
The input path is also active while the pin is configured as output. This allows to feedback an output to on-chip resources without wasting an additional external pin.

By Pn_HWSEL (**Chapter 25.8.8**) it is possible to select between different hardware “masters” (HWO0/HWI0, HWO1/HWI1). The selected peripheral can take control of the pin(s). Hardware control overrules settings in the respective port pin registers.

25.10.1 Port I/O Function Table

Table 25-12 Port I/O Functions

Function	Outputs						Inputs									
	ALT1	ALT2	ALT3	ALT4	HWO0	HWO1	HWI0	HWI1	Input	Input	Input	Input	Input	Input	Input	Input
P0.0		CAN.N0_TXD	CCU80.OUT21	LEDTS0.COL2					U1C1.DX0D	ETH0. CLK_RMIIB	ERU0.0B0					
P0.1	USB. DRIVEVBUS	U1C1.DOUT0	CCU80.OUT11	LEDTS0.COL3						ETH0. CRS_DVB	ERU0.0A0					
P0.2		U1C1.SELO1	CCU80.OUT01		U1C0.DOUT3	EBU.AD0	U1C0.HWIN3	EBU.D0	ETH0.RXD0B		ERU0.3B3					
P0.3			CCU80.OUT20		U1C0.DOUT2	EBU.AD1	U1C0.HWIN2	EBU.D1	ETH0.RXD1B			ERU1.3B0				
P0.4	ETH0.TX_EN		CCU80.OUT10		U1C0.DOUT1	EBU.AD2	U1C0.HWIN1	EBU.D2		U1C0.DX0A	ERU0.2B3					
P0.5	ETH0.TXD0	U1C0.DOUT0	CCU80.OUT00		U1C0.DOUT0	EBU.AD3	U1C0.HWIN0	EBU.D3		U1C0.DX0B		ERU1.3A0				
P0.6	ETH0.TXD1	U1C0.SELO0	CCU80.OUT30			$\overline{\text{EBU.ADV}}$				U1C0.DX2A	ERU0.3B2		CCU80.IN2B			
P0.7	WWDT. SERVICE_OUT	U0C0.SELO0				EBU.AD6	DB.TDI	EBU.D6	U0C0.DX2B	DSD.DIN1A	ERU0.2B1		CCU80.IN0A	CCU80.IN1A	CCU80.IN2A	CCU80.IN3A
P0.8	SCU.EXTCLK	U0C0. SCLKOUT				EBU.AD7	$\overline{\text{DB.TRST}}$	EBU.D7	U0C0.DX1B	DSD.DIN0A	ERU0.2A1		CCU80.IN1B			
P0.9		U1C1.SELO0	CCU80.OUT12	LEDTS0.COL0	ETH0.MDO	$\overline{\text{EBU.CS1}}$	ETH0.MDIA		U1C1.DX2A	USB.ID	ERU0.1B0					
P0.10	ETH0.MDC	U1C1. SCLKOUT	CCU80.OUT02	LEDTS0.COL1					U1C1.DX1A		ERU0.1A0					
P0.11		U1C0. SCLKOUT	CCU80.OUT31			$\overline{\text{EBU.BREQ}}$			ETH0.RXERB	U1C0.DX1A	ERU0.3A2					
P0.12		U1C1.SELO0	CCU40.OUT3			$\overline{\text{EBU.HLDA}}$		$\overline{\text{EBU.HLDA}}$		U1C1.DX2B	ERU0.2B2					
P0.13		U1C1. SCLKOUT	CCU40.OUT2							U1C1.DX1B	ERU0.2A2					
P0.14		U1C0.SELO1	CCU40.OUT1		U1C1.DOUT3		U1C1.HWIN3						CCU42.IN3C			
P0.15		U1C0.SELO2	CCU40.OUT0		U1C1.DOUT2		U1C1.HWIN2						CCU42.IN2C			
P1.0	DSD.CGPWMN	U0C0.SELO0	CCU40.OUT3	ERU1.PDOUT3					U0C0.DX2A		ERU0.3B0		CCU40.IN3A			
P1.1	DSD.CGPWMP	U0C0. SCLKOUT	CCU40.OUT2	ERU1.PDOUT2			SDMMC.SDWC		U0C0.DX1A	POSIF0.IN2A	ERU0.3A0		CCU40.IN2A			
P1.2			CCU40.OUT1	ERU1.PDOUT1	U0C0.DOUT3	EBU.AD14	U0C0.HWIN3	EBU.D14		POSIF0.IN1A		ERU1.2B0	CCU40.IN1A			
P1.3		U0C0. MCLKOUT	CCU40.OUT0	ERU1.PDOUT0	U0C0.DOUT2	EBU.AD15	U0C0.HWIN2	EBU.D15		POSIF0.IN0A		ERU1.2A0	CCU40.IN0A			
P1.4	WWDT. SERVICE_OUT	CAN.N0_TXD	CCU80.OUT33	CCU81.OUT20	U0C0.DOUT1		U0C0.HWIN1		U0C0.DX0B	CAN.N1_RXDD	ERU0.2B0		CCU41.IN0C			
P1.5	CAN.N1_TXD	U0C0.DOUT0	CCU80.OUT23	CCU81.OUT10	U0C0.DOUT0		U0C0.HWIN0		U0C0.DX0A	CAN.N0_RXDA	ERU0.2A0	ERU1.0A0	CCU41.IN1C	DSD.DIN2B		
P1.6		U0C0. SCLKOUT			SDMMC. DATA1_OUT	EBU.AD10	SDMMC. DATA1_IN	EBU.D10	DSD.DIN2A							
P1.7		U0C0.DOUT0	DSD.MCLK2		SDMMC. DATA2_OUT	EBU.AD11	SDMMC. DATA2_IN	EBU.D11		DSD.MCLK2A						

Table 25-12 Port I/O Functions (cont'd)

Function	Outputs						Inputs									
	ALT1	ALT2	ALT3	ALT4	HWO0	HWO1	HWI0	HWI1	Input	Input	Input	Input	Input	Input	Input	Input
P1.8		U0C0.SELO1	DSD.MCLK1		SDMMC. DATA4_OUT	EBU.AD12	SDMMC. DATA4_IN	EBU.D12	CAN.N2_RXDA	DSD.MCLK1A						
P1.9		CAN.N2_TXD			SDMMC. DATA5_OUT	EBU.AD13	SDMMC. DATA5_IN	EBU.D13		DSD.MCLK0A						
P1.10	ETH0.MDC	U0C0. SCLKOUT	CCU81.OUT21				$\overline{\text{SDMMC.SDCD}}$						CCU41.IN2C			
P1.11		U0C0.SELO0	CCU81.OUT11		ETH0.MDO		ETH0.MDIC						CCU41.IN3C			
P1.12	ETH0.TX_EN	CAN.N1_TXD	CCU81.OUT01		SDMMC. DATA6_OUT	EBU.AD16	SDMMC. DATA6_IN	EBU.D16								
P1.13	ETH0.TXD0	U0C1.SELO3	CCU81.OUT20		SDMMC. DATA7_OUT	EBU.AD17	SDMMC. DATA7_IN	EBU.D17	CAN.N1_RXDC							
P1.14	ETH0.TXD1	U0C1.SELO2	CCU81.OUT10			EBU.AD18		EBU.D18								
P1.15	SCU.EXTCLK	DSD.MCLK2	CCU81.OUT00			EBU.AD19		EBU.D19		DSD.MCLK2B		ERU1.1A0				
P2.0		CCU81.OUT21	DSD.CGPWMN	LEDTS0.COL1	ETH0.MDO	EBU.AD20	ETH0.MDIB	EBU.D20			ERU0.0B3		CCU40.IN1C			
P2.1		CCU81.OUT11	DSD.CGPWMP	LEDTS0.COL0	DB.TDO/ TRACESWO	EBU.AD21		EBU.D21	ETH0. CLK_RMIIA			ERU1.0B0	CCU40.IN0C			
P2.2	VADC.EMUX00	CCU81.OUT01	CCU41.OUT3	LEDTS0. LINE0	LEDTS0. EXTENDED0	EBU.AD22	LEDTS0. TSIN0A	EBU.D22	ETH0.RXD0A	U0C1.DX0A	ERU0.1B2		CCU41.IN3A			
P2.3	VADC.EMUX01	U0C1.SELO0	CCU41.OUT2	LEDTS0. LINE1	LEDTS0. EXTENDED1	EBU.AD23	LEDTS0. TSIN1A	EBU.D23	ETH0.RXD1A	U0C1.DX2A	ERU0.1A2	POSIF1.IN2A	CCU41.IN2A			
P2.4	VADC.EMUX02	U0C1. SCLKOUT	CCU41.OUT1	LEDTS0. LINE2	LEDTS0. EXTENDED2	EBU.AD24	LEDTS0. TSIN2A	EBU.D24	ETH0.RXERA	U0C1.DX1A	ERU0.0B2	POSIF1.IN1A	CCU41.IN1A			
P2.5	ETH0.TX_EN	U0C1.DOUT0	CCU41.OUT0	LEDTS0. LINE3	LEDTS0. EXTENDED3	EBU.AD25	LEDTS0. TSIN3A	EBU.D25	ETH0. CRS_DVA	U0C1.DX0B	ERU0.0A2	POSIF1.IN0A	CCU41.IN0A			
P2.6	U2C0.SELO4		CCU80.OUT13	LEDTS0.COL3	U2C0.DOUT3		U2C0.HWIN3		DSD.DIN1B	CAN.N1_RXDA	ERU0.1B3		CCU40.IN3C			
P2.7	ETH0.MDC	CAN.N1_TXD	CCU80.OUT03	LEDTS0.COL2					DSD.DIN0B			ERU1.1B0	CCU40.IN2C			
P2.8	ETH0.TXD0		CCU80.OUT32	LEDTS0. LINE4	LEDTS0. EXTENDED4	EBU.AD26	LEDTS0. TSIN4A	EBU.D26	DAC. TRIGGER5				CCU40.IN0B	CCU40.IN1B	CCU40.IN2B	CCU40.IN3B
P2.9	ETH0.TXD1		CCU80.OUT22	LEDTS0. LINE5	LEDTS0. EXTENDED5	EBU.AD27	LEDTS0. TSIN5A	EBU.D27	DAC. TRIGGER4				CCU41.IN0B	CCU41.IN1B	CCU41.IN2B	CCU41.IN3B
P2.10	VADC.EMUX10				DB. ETM_TRACED ATA3	EBU.AD28		EBU.D28								
P2.11	ETH0.TXER		CCU80.OUT22		DB. ETM_TRACED ATA2	EBU.AD29		EBU.D29								
P2.12	ETH0.TXD2		CCU81.OUT33	ETH0.TXD0	DB. ETM_TRACED ATA1	EBU.AD30		EBU.D30					CCU43.IN3C			
P2.13	ETH0.TXD3			ETH0.TXD1	DB. ETM_TRACED ATA0	EBU.AD31		EBU.D31					CCU43.IN2C			
P2.14	VADC.EMUX11	U1C0.DOUT0	CCU80.OUT21		DB. ETM_TRACEC LK	$\overline{\text{EBU.BC0}}$				U1C0.DX0D			CCU43.IN0B	CCU43.IN1B	CCU43.IN2B	CCU43.IN3B

Table 25-12 Port I/O Functions (cont'd)

Function	Outputs						Inputs									
	ALT1	ALT2	ALT3	ALT4	HWO0	HWO1	HWI0	HWI1	Input	Input	Input	Input	Input	Input	Input	Input
P2.15	VADC.EMUX12		CCU80.OUT11	LEDTS0. LINE6	LEDTS0. EXTENDED6	EBU.$\overline{BC1}$	LEDTS0. TSIN6A		ETH0.COLA	U1C0.DX0C			CCU42.IN0B	CCU42.IN1B	CCU42.IN2B	CCU42.IN3B
P3.0	U2C1.SELO0	U0C1. SCLKOUT	CCU42.OUT0			EBU.$\overline{RD}$			U0C1.DX1B				CCU80.IN2C	CCU81.IN0C		
P3.1		U0C1.SELO0				EBU.$\overline{RD}$/ EBU.WR			U0C1.DX2B		ERU0.0B1		CCU80.IN1C			
P3.2	USB. DRIVEVBUS	CAN.N0_TXD		LEDTS0.COLA		EBU.$\overline{CS0}$					ERU0.0A1		CCU80.IN0C			
P3.3		U1C1.SELO1	CCU42.OUT3		SDMMC.LED			EBU.$\overline{WAIT}$		DSD.DIN3B			CCU42.IN3A	CCU80.IN3B		
P3.4	U2C1. MCLKOUT	U1C1.SELO2	CCU42.OUT2	DSD.MCLK3	SDMMC. BUS_POWER			EBU.$\overline{HOLD}$	U2C1.DX0B	DSD.MCLK3B			CCU42.IN2A	CCU80.IN0B		
P3.5	U2C1.DOUT0	U1C1.SELO3	CCU42.OUT1	U0C1.DOUT0	SDMMC.CMD_ OUT	EBU.AD4	SDMMC.CMD_I N	EBU.D4	U2C1.DX0A		ERU0.3B1		CCU42.IN1A			
P3.6	U2C1. SCLKOUT	U1C1.SELO4	CCU42.OUT0	U0C1. SCLKOUT	SDMMC.CLK_ OUT	EBU.AD5	SDMMC.CLK_I N	EBU.D5	U2C1.DX1B		ERU0.3A1		CCU42.IN0A			
P3.7		CAN.N2_TXD	CCU41.OUT3	LEDTS0. LINE0					U2C0.DX0C							
P3.8	U2C0.DOUT0	U0C1.SELO3	CCU41.OUT2	LEDTS0. LINE1					CAN.N2_RXDB				POSIF1.IN2B			
P3.9	U2C0. SCLKOUT	CAN.N1_TXD	CCU41.OUT1	LEDTS0. LINE2									POSIF1.IN1B			
P3.10	U2C0.SELO0	CAN.N0_TXD	CCU41.OUT0	LEDTS0. LINE3	U0C1.DOUT3		U0C1.HWIN3						POSIF1.IN0B			
P3.11	U2C1.DOUT0	U0C1.SELO2	CCU42.OUT3	LEDTS0. LINE4	U0C1.DOUT2		U0C1.HWIN2		CAN.N1_RXDB					CCU81.IN3C		
P3.12		U0C1.SELO1	CCU42.OUT2	LEDTS0. LINE5	U0C1.DOUT1		U0C1.HWIN1		CAN.N0_RXDC	U2C1.DX0D				CCU81.IN2C		
P3.13	U2C1. SCLKOUT	U0C1.DOUT0	CCU42.OUT1	LEDTS0. LINE6	U0C1.DOUT0		U0C1.HWIN0		U0C1.DX0D				CCU80.IN3C	CCU81.IN1C		
P3.14		U1C0.SELO3			U1C1.DOUT1		U1C1.HWIN1			U1C1.DX0B			CCU42.IN1C			
P3.15		U1C1.DOUT0			U1C1.DOUT0		U1C1.HWIN0			U1C1.DX0A			CCU42.IN0C			
P4.0			DSD.MCLK1		SDMMC. DATA0_OUT	EBU.AD8	SDMMC. DATA0_IN	EBU.D8	U1C1.DX1C	DSD.MCLK1B	U0C1.DX0E	U2C1.DX0C				
P4.1	U2C1.SELO0	U1C1. MCLKOUT	DSD.MCLK0	U0C1.SELO0	SDMMC. DATA3_OUT	EBU.AD9	SDMMC. DATA3_IN	EBU.D9	U2C1.DX2B	DSD.MCLK0B		U2C1.DX2A				
P4.2	U2C1.SELO1	U1C1.DOUT0		U2C1. SCLKOUT					U1C1.DX0C			U2C1.DX1A	CCU43.IN1C			
P4.3	U2C1.SELO2	U0C0.SELO5	CCU43OUT3										CCU43.IN3A			
P4.4		U0C0.SELO4	CCU43OUT2		U2C1.DOUT3		U2C1.HWIN3						CCU43.IN2A			
P4.5		U0C0.SELO3	CCU43OUT1		U2C1.DOUT2		U2C1.HWIN2						CCU43.IN1A			
P4.6		U0C0.SELO2	CCU43OUT0		U2C1.DOUT1		U2C1.HWIN1		CAN.N2_RXDC				CCU43.IN0A			
P4.7		CAN.N2_TXD			U2C1.DOUT0		U2C1.HWIN0		U0C0.DX0C				CCU43.IN0C			

Table 25-12 Port I/O Functions (cont'd)

Function	Outputs						Inputs									
	ALT1	ALT2	ALT3	ALT4	HWO0	HWO1	HWI0	HWI1	Input	Input	Input	Input	Input	Input	Input	Input
P5.0	U2C0.DOUT0	DSD.CGPWMN	CCU81.OUT33		U2C0.DOUT0		U2C0.HWIN0		U2C0.DX0B	ETH0.RXD0D	U0C0.DX0D		CCU81.IN0A	CCU81.IN1A	CCU81.IN2A	CCU81.IN3A
P5.1	U0C0.DOUT0	DSD.CGPWMP	CCU81.OUT32		U2C0.DOUT1		U2C0.HWIN1		U2C0.DX0A	ETH0.RXD1D			CCU81.IN0B			
P5.2	U2C0. SCLKOUT		CCU81.OUT23						U2C0.DX1A	ETH0. CRS_DVD			CCU81.IN1B			
P5.3	U2C0.SELO0		CCU81.OUT22		EBU.CKE	EBU.A20			U2C0.DX2A	ETH0.RXERD			CCU81.IN2B			
P5.4	U2C0.SELO1		CCU81.OUT13		$\overline{\text{EBU.RAS}}$	EBU.A21				ETH0.CRSD			CCU81.IN3B			
P5.5	U2C0.SELO2		CCU81.OUT12		$\overline{\text{EBU.CAS}}$	EBU.A22				ETH0.COLD						
P5.6	U2C0.SELO3		CCU81.OUT03		EBU.BFCLKO	EBU.A23			EBU.BFCLKI							
P5.7			CCU81.OUT02	LEDTS0.COLA	U2C0.DOUT2		U2C0.HWIN2									
P5.8		U1C0. SCLKOUT	CCU80.OUT01		EBU.SDCLKO	$\overline{\text{EBU.CS2}}$			ETH0.RXD2A	U1C0.DX1B						
P5.9		U1C0.SELO0	CCU80.OUT20	ETH0.TX_EN	EBU.BFCLKO	$\overline{\text{EBU.CS3}}$			ETH0.RXD3A	U1C0.DX2B						
P5.10		U1C0. MCLKOUT	CCU80.OUT10	LEDTS0. LINE7	LEDTS0. EXTENDED7		LEDTS0. TSIN7A		ETH0. CLK_TXA							
P5.11		U1C0.SELO1	CCU80.OUT00						ETH0.CRSA							
P6.0	ETH0.TXD2	U0C1.SELO1	CCU81.OUT31		DB. ETM_TRACECLK	EBU.A16										
P6.1	ETH0.TXD3	U0C1.SELO0	CCU81.OUT30		DB. ETM_TRACEDATA3	EBU.A17			U0C1.DX2C							
P6.2	ETH0.TXER	U0C1. SCLKOUT	CCU43OUT3		DB. ETM_TRACEDATA2	EBU.A18			U0C1.DX1C							
P6.3			CCU43OUT2						U0C1.DX0C	ETH0.RXD3B						
P6.4		U0C1.DOUT0	CCU43OUT1		EBU.SDCLKO	EBU.A19			EBU.SDCLKI	ETH0.RXD2B						
P6.5		U0C1. MCLKOUT	CCU43OUT0		DB. ETM_TRACEDATA1	EBU.BC2			DSD.DIN3A	ETH0. CLK_RMIID						
P6.6			DSD.MCLK3		DB. ETM_TRACEDATA0	EBU.BC3			DSD.MCLK3A	ETH0. CLK_TXB						
P14.0									VADC.G0CH0							
P14.1									VADC.G0CH1							
P14.2									VADC.G0CH2	VADC.G1CH2						
P14.3									VADC.G0CH3	VADC.G1CH3			CAN.N0_RXDB			
P14.4									VADC.G0CH4		VADC.G2CH0					
P14.5									VADC.G0CH5		VADC.G2CH1		POSIF0.IN2B			
P14.6									VADC.G0CH6				POSIF0.IN1B		G0ORC6	
P14.7									VADC.G0CH7				POSIF0.IN0B		G0ORC7	

Table 25-12 Port I/O Functions (cont'd)

Function	Outputs						Inputs									
	ALT1	ALT2	ALT3	ALT4	HWO0	HWO1	HWI0	HWI1	Input	Input	Input	Input	Input	Input	Input	Input
P14.8					DAC.OUT_0					VADC.G1CH0		VADC.G3CH2	ETH0.RXD0C			
P14.9					DAC.OUT_1					VADC.G1CH1		VADC.G3CH3	ETH0.RXD1C			
P14.12										VADC.G1CH4						
P14.13										VADC.G1CH5						
P14.14										VADC.G1CH6					G1ORC6	
P14.15										VADC.G1CH7					G1ORC7	
P15.2											VADC.G2CH2					
P15.3											VADC.G2CH3					
P15.4											VADC.G2CH4					
P15.5											VADC.G2CH5					
P15.6											VADC.G2CH6					
P15.7											VADC.G2CH7					
P15.8												VADC.G3CH0	ETH0. CLK_RMIIC			
P15.9												VADC.G3CH1	ETH0. CRS_DVC			
P15.12												VADC.G3CH4				
P15.13												VADC.G3CH5				
P15.14												VADC.G3CH6				
P15.15												VADC.G3CH7				
USB_DP																
USB_DM																
HIB_IO_0	HIBOUT	WWDT. SERVICE_OUT							WAKEUPA							
HIB_IO_1	HIBOUT	WWDT. SERVICE_OUT							WAKEUPB							
TCK							DB.TCK/ SWCLK									
TMS					DB.TMS/ SWDIO											
$\overline{\text{PORST}}$																
XTAL1									U0C0.DX0F	U0C1.DX0F	U1C0.DX0F	U1C1.DX0F	U2C0.DX0F	U2C1.DX0F		
XTAL2																
RTC_XTAL1											ERU0.1B1					
RTC_XTAL2																

Startup Modes

26 Startup modes

This chapter describes the various startup modes supported by the device along with actions that must be performed by the end user.

26.1 Overview

The on-chip firmware resides in a non volatile memory namely the BootROM (also known as MaskROM) and is the first software of any kind to be executed by the CPU right after reset.

The on-chip firmware has two parts:

- Startup Software (SSW) which provisions the various boot modes and is the main thread of execution
- Test Firmware (Testware, not described in this document) which deals with test related routines that can be invoked by ATE in test mode only

The terms startup mode and boot mode mean the same and are used interchangeably throughout this chapter.

26.1.1 Features

Supported boot modes are summarized briefly. Desired boot mode can be enabled by driving the boot mode pins (JTAG TCK and TMS) with appropriate logic levels and issuing a power on reset (PORST). Some boot modes can only be selected by configuring STCON.SWCON bit field (of SCU module) and applying a system reset (such as a watchdog reset or CPU software reset).

Normal Boot mode

Startup type:***Internal start***

Required Reset type:*PORST and System reset*

An application located at the start of flash is given control after SSW has finished its execution.

Alternative Boot mode (ABM-0/ABM-1)

Startup type:*Internal start*

Required Reset type:*System reset*

An application located at user defined location on the flash is given control by SSW. The SSW after completing its execution evaluates a header hereafter known as ABM header kept at a well known address on the flash which in turn provides the location of application placed at user defined address. Two such applications can be programmed into the flash and thus two ABMs are supported. An invalid header results in the SSW

aborting further execution and launching the CPU into safe mode. A PORST is required to exit the safe mode of operation.

Fallback ABM

Startup type:*Internal start*

Required Reset type:*System reset*

SSW evaluates the two ABM headers in succession. The first ABM header found valid by SSW results in application referenced by that header given control to. Should both the headers be found unusable, SSW aborts further execution and places the CPU into a safe mode. A PORST is required to exit the safe mode of operation.

PSRAM boot

Startup type:*Internal start*

Required Reset type:*System reset*

An application loaded into PSRAM is given control after SSW finishes its execution. The start address of this application is deduced from an ABM like header placed in the last 32 bytes of PSRAM.

UART BSL

Startup type:*External start*

Required Reset type:*PORST and System reset*

An application can be downloaded into the start of PSRAM over the USIC ASC interface and executed. The size of the application downloaded is limited to the size of PSRAM on the device.

CAN BSL (CAN Bootstrap loading)

Startup type:*External start*

Required Reset type:*PORST and System reset*

This is same as ASC BSL mode except that the application is downloaded over CAN interface and executed.

Boot mode Index (BMI)

Startup type:*Not applicable*

Required Reset type:*PORST and System reset*

A user defined bootmode is offered via following mechanism. With initial factory programming (at customer site), a so called BMI string can be programmed into user configuration block (UCB). The BMI string describes actions that must be performed by SSW before lending into one of the internal or external startup modes.

26.2 Startup modes

These chapters describe the various device startup modes summarized in **Chapter 26.1.1**.

26.2.1 Reset types and corresponding boot modes

XMC4000 platforms supports 2 categories of reset namely Power On Reset (PORST) and System reset. Every cause of reset which is not a PORST is a system reset.

When the SSW executes after a PORST, it gets to choose from one of Normal boot mode, ASC BSL, CAN BSL and BMI based on what is read off the boot pins (JTAG TCK and TMS).

When the SSW executes after a system reset, it chooses one of the following boot modes - Normal, ASC BSL, BMI, CAN BSL, PSRAM boot, ABM-0, ABM-1 and Fallback ABM.

Following table enlists the boot mode pin encoding and associated boot modes.

Table 26-1 Boot mode pin encoding for PORST

TCK	TMS	HWCON[1]	HWCON[0]	Boot mode
0	0	0	1	ASC BSL
0	1	0	0	Normal
1	0	1	1	CAN BSL
1	1	1	0	BMI

TCK and TMS are cached into HWCON[1:0] bit field of the SCU STCON register after PORST. HWCON bits are read by the SSW upon emergence from PORST.

SCU STCON.HWCON is mirrored by STCON.SWCON[1:0] after a PORST.

STCON contents can only be modified by PORST.

SWCON bitfield is read by firmware when the device emerges from a system reset. The following table enlists the encoding of the SWCON bitfield and boot modes. In the event when software does not explicitly program SWCON and a system reset is experienced, values on TCK and TMS decide the boot mode.

Table 26-2 System reset boot modes

SWCON[3:0]	Boot mode
0000_B	Normal
0001_B	ASC BSL
0010_B	BMI

Table 26-2 System reset boot modes (cont'd)

SWCON[3:0]	Boot mode
0011_B	CAN BSL
0100_B	PSRAM boot
1000_B	ABM-0
1100_B	ABM-1
1110_B	Fallback ABM

26.2.2 Initial boot sequence

There are several tasks which the SSW performs before it gets to the point where the user requested boot mode must be identified and launched. This is to ensure that user applications have a stable execution environment when program control is eventually ceded to them.

The SSW ensures that the flash subsystem has initialized before any user program can be executed out of it (flash).

It identifies the user requested boot mode and launches the same. It is important to state at this point that a few DSRAM1 locations are used by SSW for staging information read out of FCS. **Figure 26-1** depicts usage of DSRAM1 used by SSW.

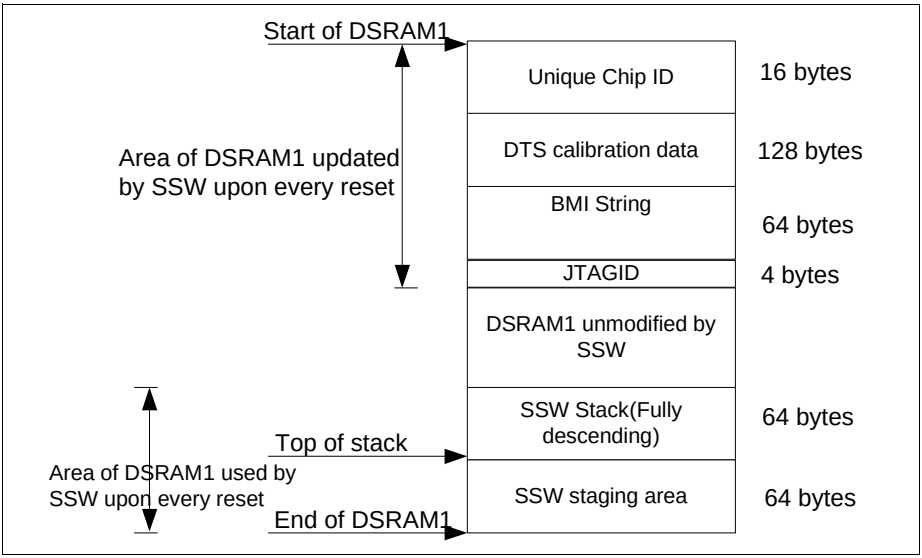


Figure 26-1 DSRAM1 usage by SSW

26.2.3 Boot mode selection

HWCON bit field is read only for PORST (Power ON Reset). For every other reset type (available in SCU_RSTSTAT) register, the SWCON field is assessed.

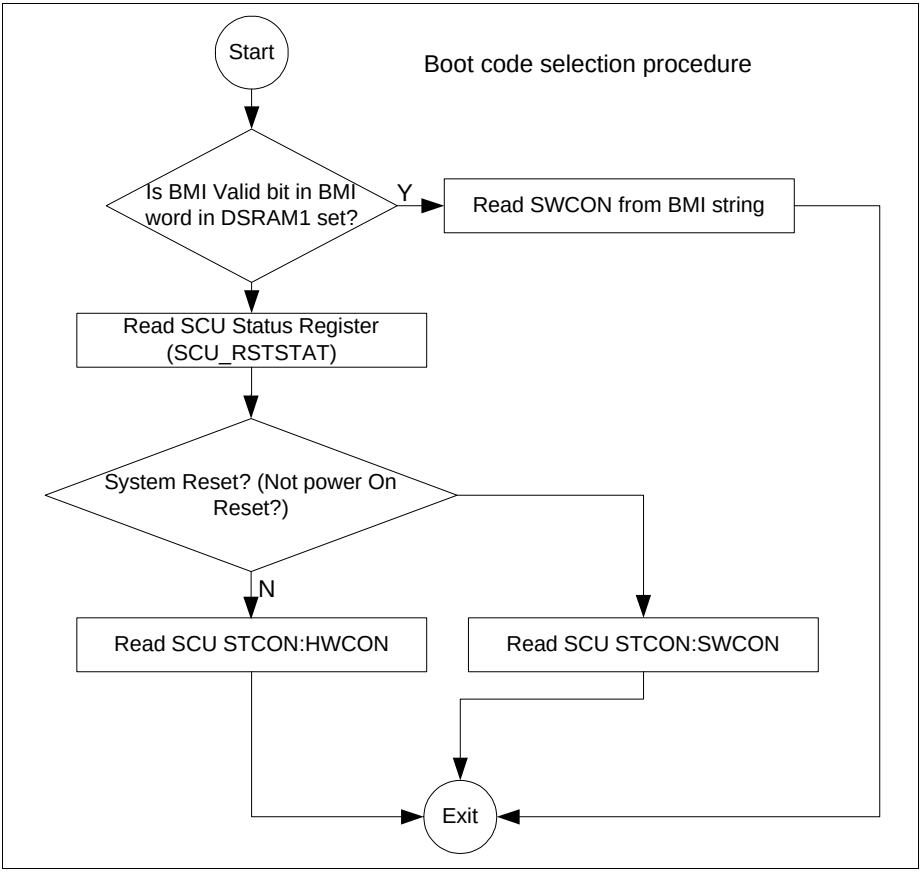


Figure 26-2 Reading Bootcode

Figure 26-3 depicts the decision tree that the SSW has to traverse in order to select the desired boot mode.

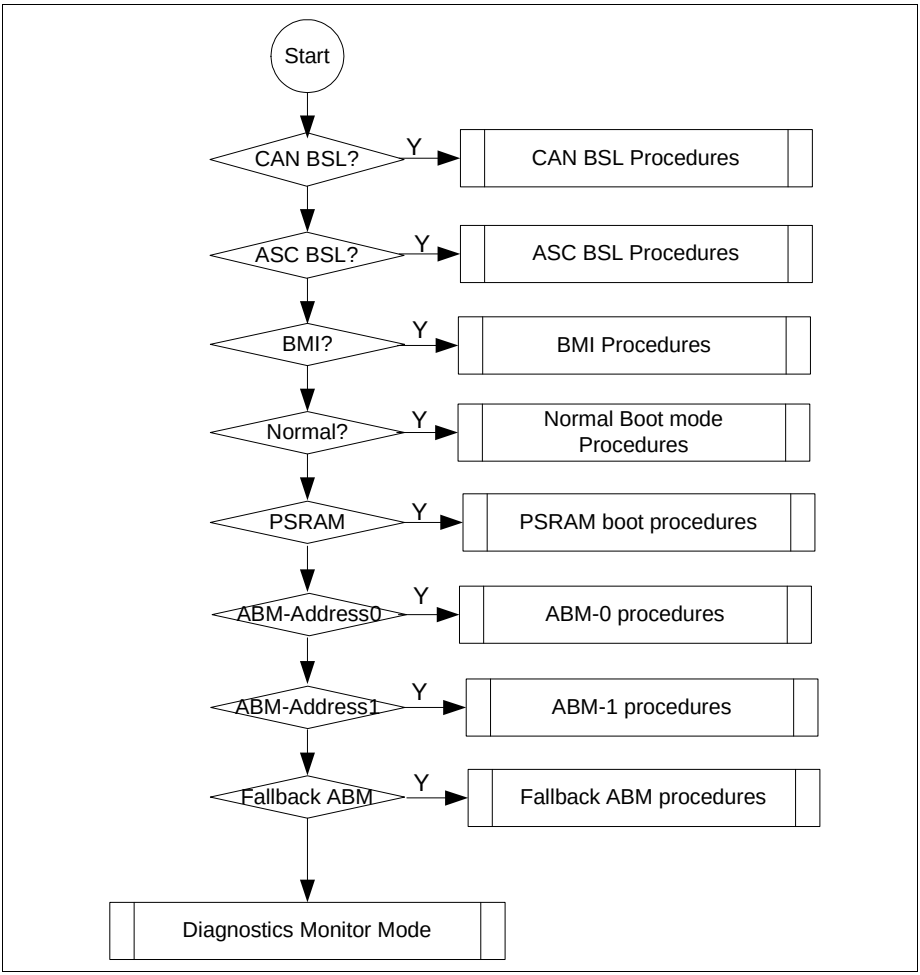


Figure 26-3 Boot mode identification

26.2.4 Normal boot mode

This is a boot mode in which user application available at the start of flash ($0C000000_H$) is given control to after SSW execution.

SSW enables access to coresight system before ceding control to the first user instruction. If the HALT after RESET feature were requested for by a connected hardware debugger, SSW configures a breakpoint on the first user instruction.

Debugger access is prohibited if the global flash read protection were found enabled. Before program control can be ceded to user application (described next), SSW turns on the Startup protection feature. This restricts access to certain registers (described in various chapters as startup protected registers) and memories such as the Flash Configuration Sector (FCS).

It is expected that the vector table of user application is available at the start of the flash. Firmware essentially reprograms the Cortex M4's SCB VTOR register with the start address of flash ($0C000000_H$) and cedes control to user application by programing register R15 (Program Counter) with reset vector contents. The reset vector contents point to a routine that could be in either the cached or the uncached address space of the flash.

$0C000000_H$ is the uncached start address of the flash.

Users have the option of linking their applications (Vector table, Text and Constant Data) to the cahed address space (08000000_H - $080FFFFF_H$). Thus the cached address space becomes Virtual Memory Address (VMA). The Load Memory Address (LMA) for the cached space is the range $0C000000_H$ - $0C0FFFFF_H$.The linking mechanism of the software toolchain must ensure that the following equation is maintained.

VMA = LMA & 04000000_H

Users may also choose to link their applications to uncached space in which case the VMA and the LMA are the same.

When an application has been linked to cached address space, user code should reprogram Cortex M4's SCB VTOR register with the VMA of the vector table.

Figure 26-4, **Figure 26-5** and **Figure 26-6** depict commonly followed application memory layout.

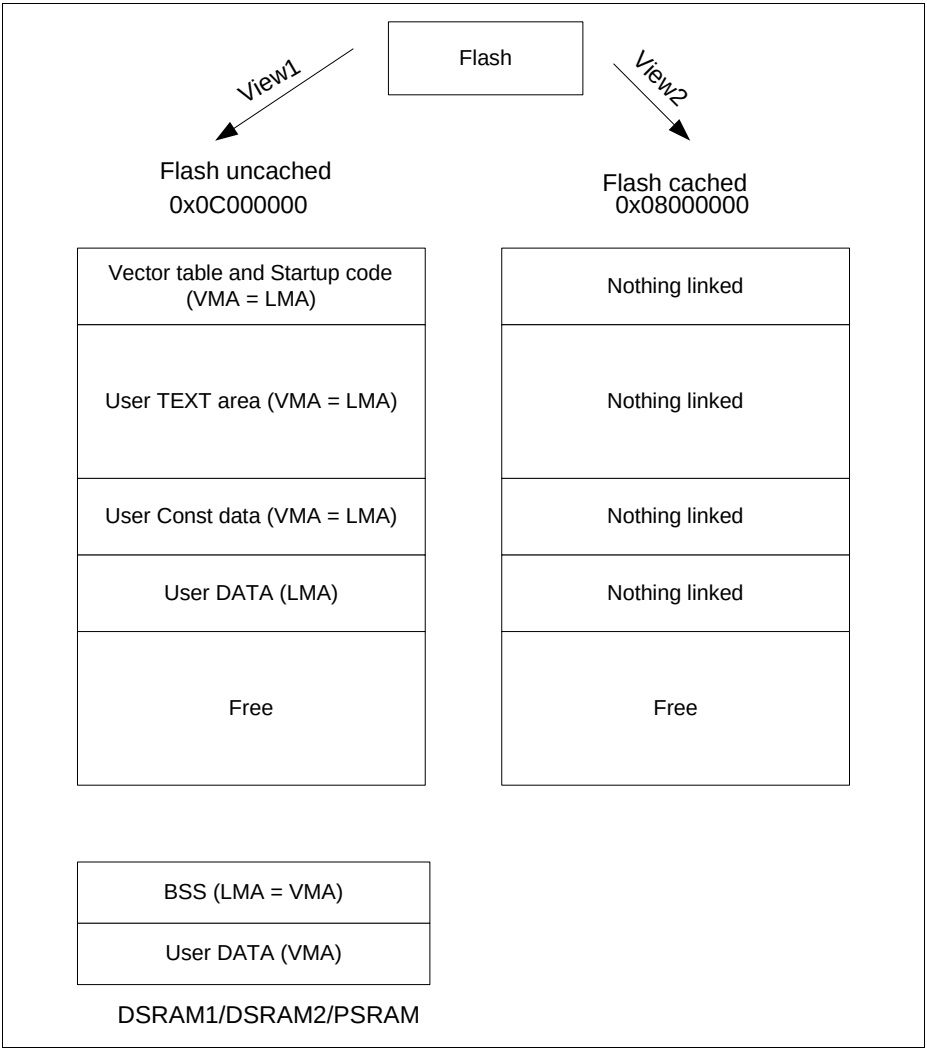


Figure 26-4 Memory layout1

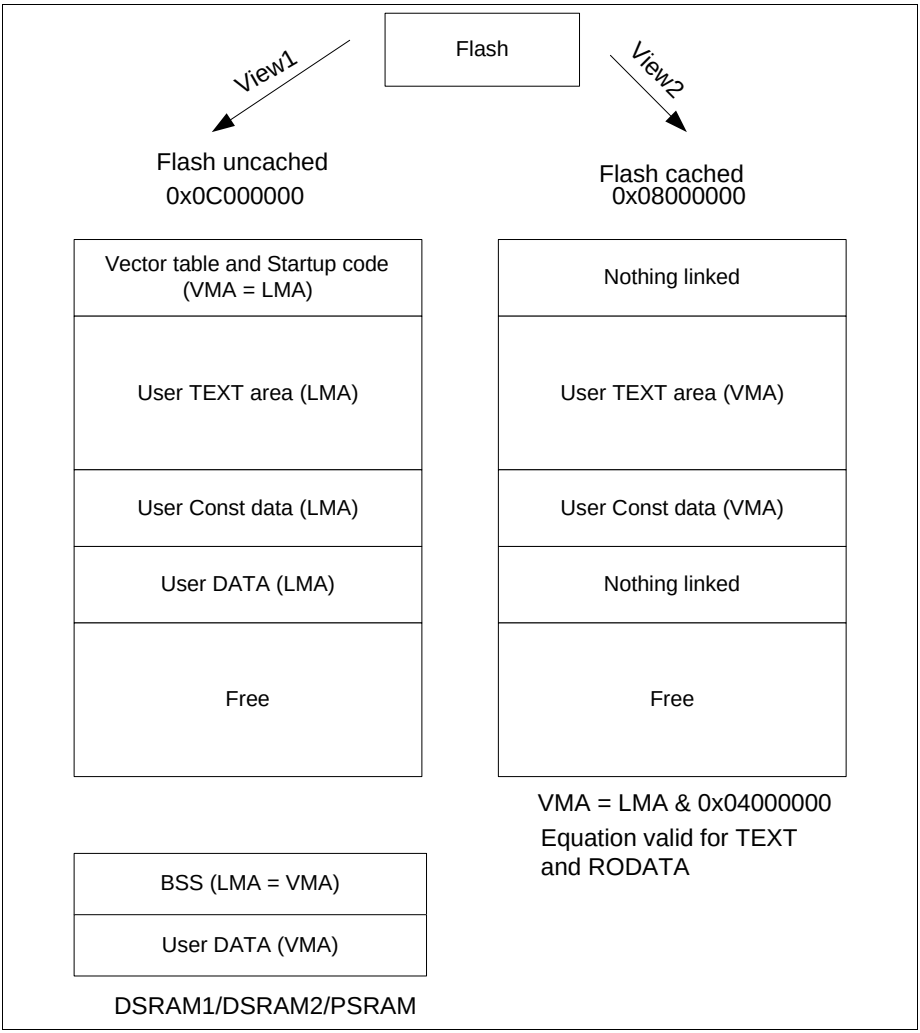


Figure 26-5 Memory layout2

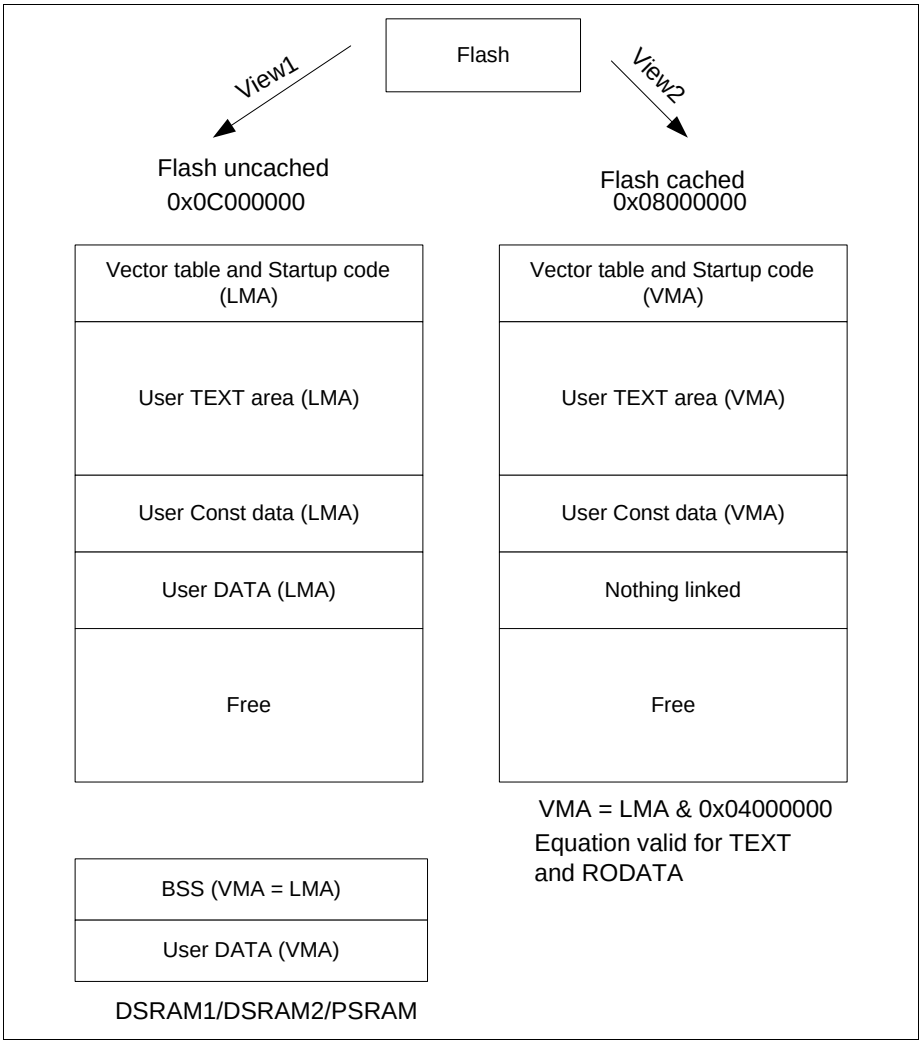


Figure 26-6 Memory layout3

User actions for normal boot mode

User must:

- Flash the application at the start of flash
- Drive TCK and TMS as per **Table 26-1**

- Issue a PORST
- Alternatively, a currently running application can write to STCON.SWCON and issue a system reset (SWCON encoding available in **Table 26-2**)

26.2.5 Boot from PSRAM

This boot mode option requires user code to be downloaded into Program SRAM (PSRAM) first.

The SWCON bit field of the SCU STCON register is then expected to be programmed with the Boot from PSRAM boot code. This must be followed by initiation of any of the system resets.

PORST leaves SRAM contents undefined. Any other reset results in previous PSRAM contents retained intact. Hence in order for this boot mode to function as desired, the reset type simply cannot be PORST. Application initiated software reset or a watchdog reset are two examples of system reset.

For SSW to branch to user application in PSRAM, it must first be assured of integrity of user application. This is done by means of a magic key ($A5C3E10F_H$) and CRC audit. It is therefore required that the PSRAM boot header be placed at the last 32 bytes of the PSRAM. The layout of the header is depicted in **Figure 26-7**. Polynomial used for checksum calculation is of CRC-32 type ($04C11DB7_H$).

Magic Key (32 bits)
Start address of code (32 bits)
Length of the code (32 bits)
CRC-32 code for code range (32 bits)
CRC-32 code for above 4 (32 bits)

Figure 26-7 PSRAM header layout

This layout is reused in the ABM boot modes. A pictorial representation PSRAM usage for this boot mode is presented in **Figure 26-8**.

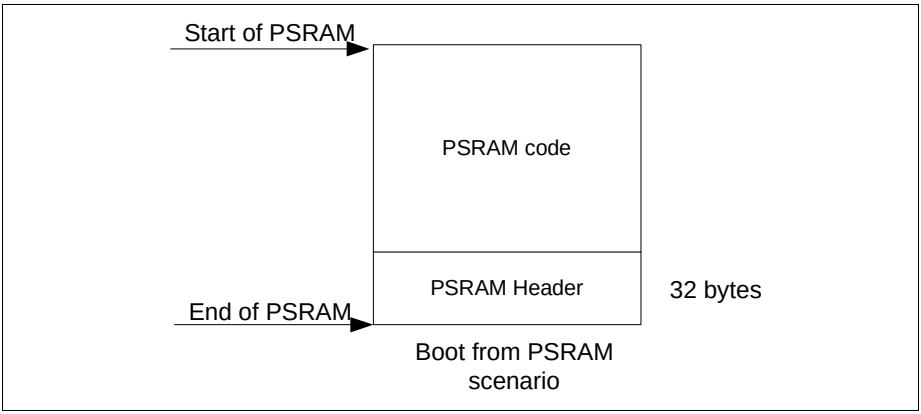


Figure 26-8 PSRAM layout for PSRAM boot

After audits confirm integrity of the code, SSW installs startup protection and cedes control to user application. SCB VTOR is programmed with PSRAM start address 10000000_H and CPU register R15 with application reset vector.

User actions for PSRAM boot mode

User must:

- Download application (Vector table + code) into PSRAM (Currently running program launched by any of the other internal boot modes can do this)
- Create PSRAM header and program the last 32 bytes of PSRAM with this header (Currently running program can either download header from host or create a header after application has been downloaded)
- Program SWCON bit field in the SCU STCON register
- Issue a system reset

26.2.6 Alternative boot mode - Address0 (ABM-0)

SSW can cede control to user application residing at an user defined flash address. As described in **Figure 26-7**, an identical header is expected to be present however at a fixed location on the flash. This fixed address for the header is the last 32bytes (Example $0C00FFE0_H$ on XMC4500) of the first 64KB physical sector.

Should the SSW find a corrupted header, execution is aborted and a diagnostics monitor mode is entered into.

As a norm, the address range of an application is typically linear without any holes. As an exception, an application may be scattered on the flash (with the help of scatter linker scripts) thus leaving holes on the flash. In such as cases, SSW only audits the ABM header and decides if control may be ceded to the reset vector. Distinction between

linear and scattered application is made by evaluating application CRC and application length fields of the header. Both of these fields are set to $FFFFFFFF_H$ in the case of scattered application. Polynomial used for CRC calculations is CRC-32 ($04C11DB7_H$). A pictorial representation of this concept is presented in **Figure 26-9**.

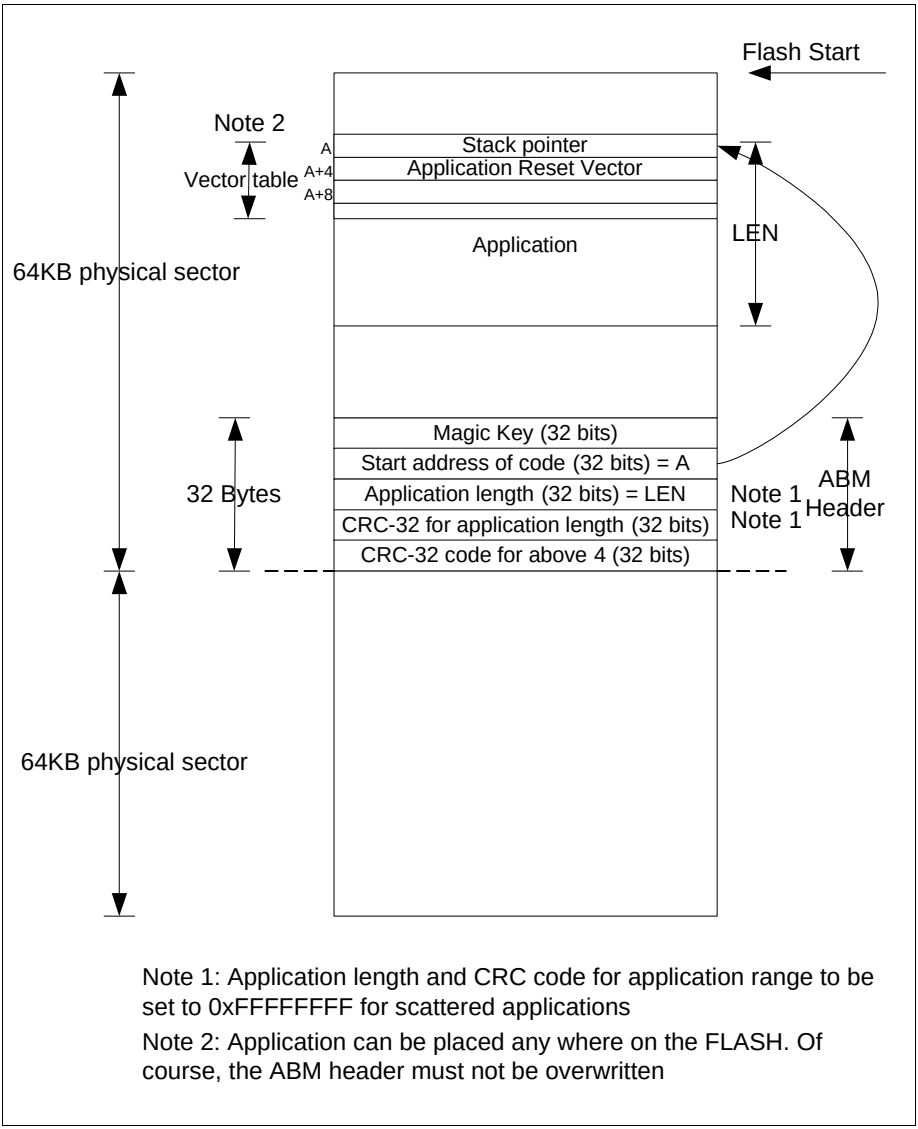


Figure 26-9 ABM concept

User actions

User must:

- Program flash with application and ABM header
- Drive TCK and TMS to launch Normal boot mode
- Configure STCON.SWCON per **Table 26-2**
- Issue a system reset (Example : Watchdog reset, Software reset)
- Alternatively, program the flash with application and ABM header
- Encode the SWCON field in the BMI word with ABM boot code
- Drive TCK and TMS for BMI and issue PORST

26.2.7 Alternative boot mode - Address1 (ABM-1)

This is same as ABM-Address0. Address for the header is the last 32bytes (Example $0C01FFE0_H$ for XMC4500) of the second 64KB physical sector.

26.2.8 Fallback ABM

When this boot mode is selected, ABM Address-0 header is audited first. A positive audit results in SSW ceding control to user application pointed to by the header. A negative audit results in evaluation of ABM Address-1. Should the audit of ABM-Address1 header fail, SSW launched diagnostics monitor mode.

26.2.9 ASC BSL mode

SSW supports bootstrap loader modes. The name though is a misnomer. When configured, any user application limited to the size of PSRAM on the device can be downloaded into PSRAM over the USIC channel (U0C0) and immediately executed.

As an example, this application may be a secondary flash loader that can download a larger application and write the latter into program flash.

Data and code fetches are disabled if the global flash read protection is found installed. Initial preparation and generic procedures are described next.

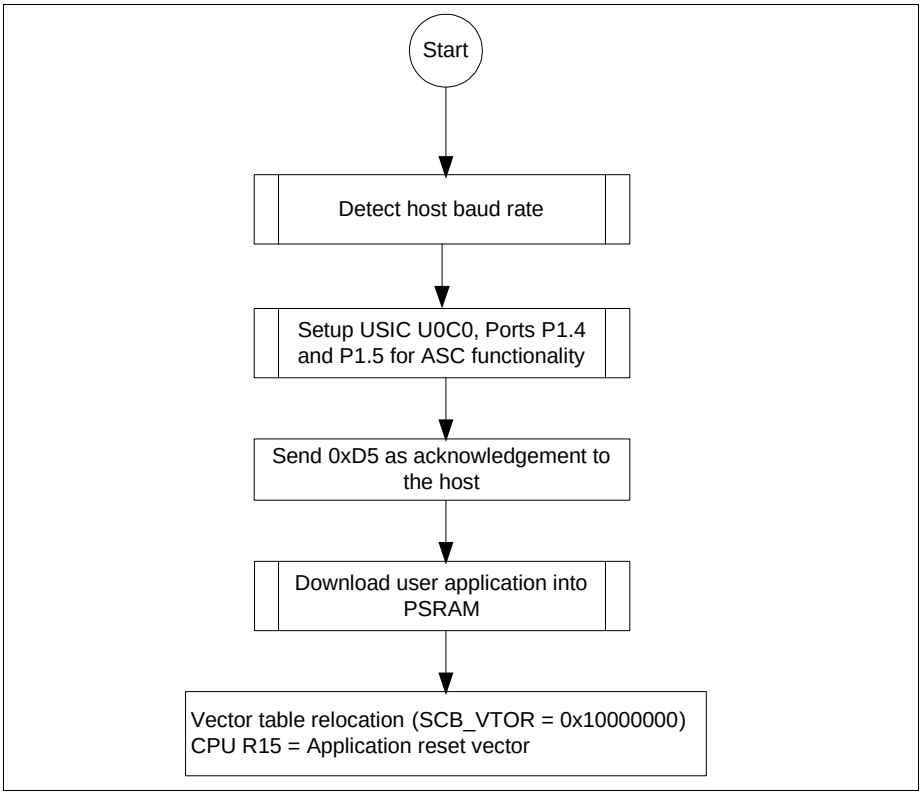


Figure 26-10 ASC BSL mode procedures

Full duplex ASC functionality of U0C0 is used for the BSL mode.

Port pins used are P1.4 (U0C0_DX0B) for USIC RX and P1.5 (U0C0_DOUT0) for USIC TX functionality.

The host starts by transmitting a zero byte to help the device detect the baud rate. After the baud rate has been detected by the device, a download protocol specified next helps download of application of any size only limited by the size of PSRAM on the device.

After the baud rate has been detected, SSW transmits an acknowledgement byte $D5_H$ back to the host. It then awaits 4 bytes describing the length of the application from the host. The least significant byte is received first.

If application length is found acceptable by SSW, an OK (1_H) byte is sent to the host following which the latter sends the byte stream of the application. After the byte stream has been received, SSW terminates the protocol by sending a final OK byte and then cedes control to the downloaded application.

If application length is found to be in error (application length greater than device PSRAM size), a N_OK byte is transmitted back to the host and the SSW resumes awaiting the length bytes.

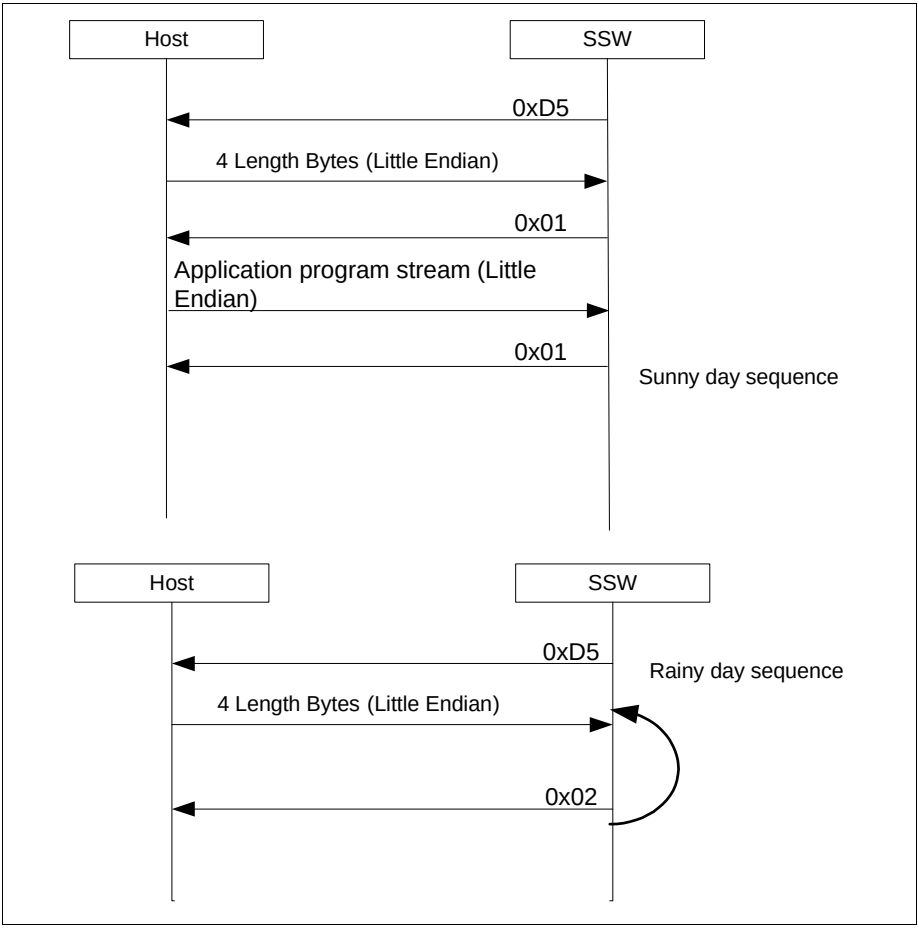


Figure 26-11 Application download protocol

User actions for ASC BSL mode

User must:

- Configure boot mode pins as described in **Table 26-1**
- Reset the device
- Ensure host sends 00_H to the device (to help device detect the baud rate)

- Ensure host receives $D5_H$ as acknowledgement from device
- Ensure host then sends data length of application and receive a positive acknowledgement
- Ensure host sends the complete application byte stream

26.2.10 CAN BSL mode

The CAN bootstrap loader mode transfers user application via Node-0 of the MultiCAN module into PSRAM.

A stable external cock is mandatory. SSW uses an iterative algorithm to determine the external clock frequency and switches to it. For transfer rates of 1mbps, it must be ensured by the end user that the external clock at least 10MHz.

A protocol comprising three phases described next leads to user application downloaded into PSRAM. Size of downloaded application is only limited by the size of PSRAM on the device.

Figure 26-12 depicts aforementioned CAN BSL mode procedures.

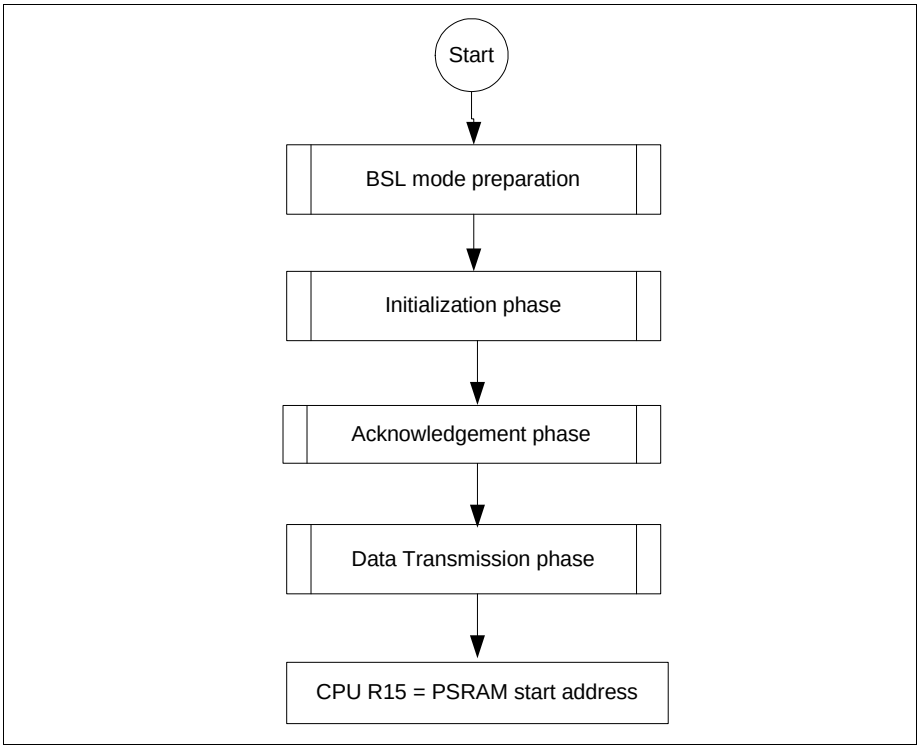


Figure 26-12 CAN BSL procedures

Initialization phase

The baud rate of the host is determined.

SSW switches to external clock source. PLL is brought out of power down mode and configured for pre-scalar mode of operation. The VCO is bypassed and powered down.

A standard CAN base frame comprising eight data bytes is transmitted continuously by the host. The 11 bit message ID of the frame (555_H) is used by SSW for baud rate detection. Data bytes 2 and 3 contain the acknowledgement identifier which the SSW must use which acknowledging the completion of initialization phase to the host. The next two bytes (4 and 5) indicate the number of eight byte data frames of user application which must be received by SSW and placed into PSRAM. The last two bytes (6 and 7) indicate the message identifier that would be used by the host while transmitting the user application.

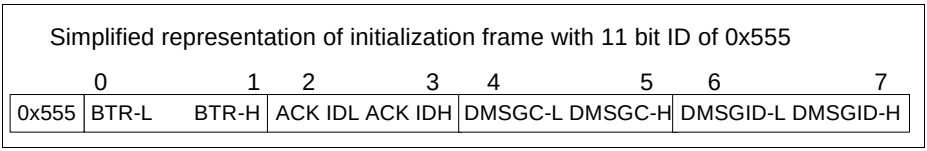


Figure 26-13 Data field of CAN BSL Initialization frame

Port pins used are P1.4 (MCAN_N0_TXD) for TX and P1.5 (MCAN_N0_RXDA) for RX functionality. Reversal of MultiCAN I/O functionality vis-vis ASC BSL may be noted here.

Acknowledgement phase

An acknowledgement frame is sent to the host indicating completion of initialization phase.

After SSW computes the baud rate of the host and reconfigures the NBTR register of node-0, it waits until the initialization frame is correctly and fully received. SSW signals its intent to use the bus by transmitting a dominant (0) bit in its ACK slot.

After the dominant bit has been transmitted, an acknowledgement frame using the ACK-ID extracted from the initialization frame is sent to the host. If the size of application intended to be downloaded is greater than the size of PSRAM on the device, a negative acknowledgement as depicted below is sent. SSW enters acknowledgement phase again. **Figure 26-14** depicts data part of acknowledgement frame.

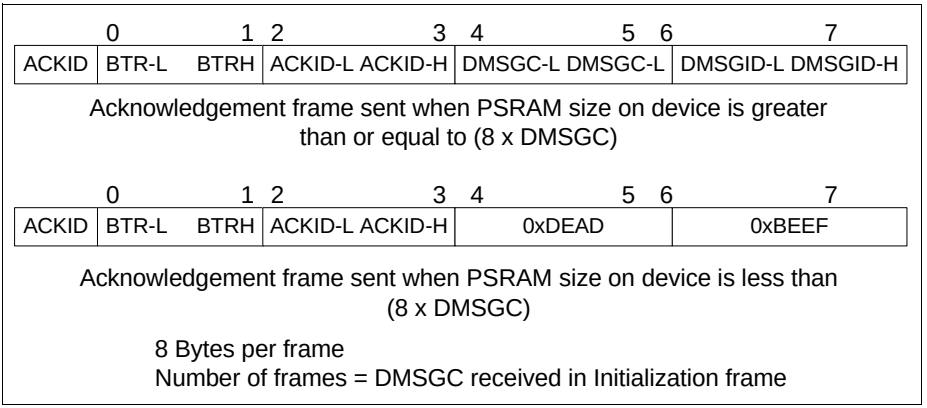


Figure 26-14 CAN Acknowledgement frame

Data transmission phase

Host transmits user application in several CAN frames to the device.

After the SSW transmits the acknowledgement frame, it prepares to receive user application over several CAN frames. Each CAN frame carries eight data bytes and the number of CAN frames is limited to the value retrieved from the DMSGC (Data Message Count) field of the initialization frame. Message identifier is essentially the DMSGID extracted from the initialization frame.

Data received in a frame is placed into PSRAM sequentially.After all of the frames have been received, SSW cedes control to the first user instruction at the start of PSRAM.

User actions for CAN BSL mode

User must:

- Configure the boot mode pins as described in **Table 26-1**
- Reset the device
- Ensure CAN host continuously transmits initial frame described in **Table 26-2**
- Ensure host receives acknowledgement frame and transmits the application stream

26.2.11 Boot Mode Index (BMI)

BMI provides a provision for end user to customize boot sequence. A 32bit BMI word describes a set of activities that must be performed by SSW. Such a BMI word is available in page-1 of User Configuration Block-2 (UCB2-Page1).

BMI word along with associated parameters is known as the BMI string. **Figure 26-15** depicts this pictorially.

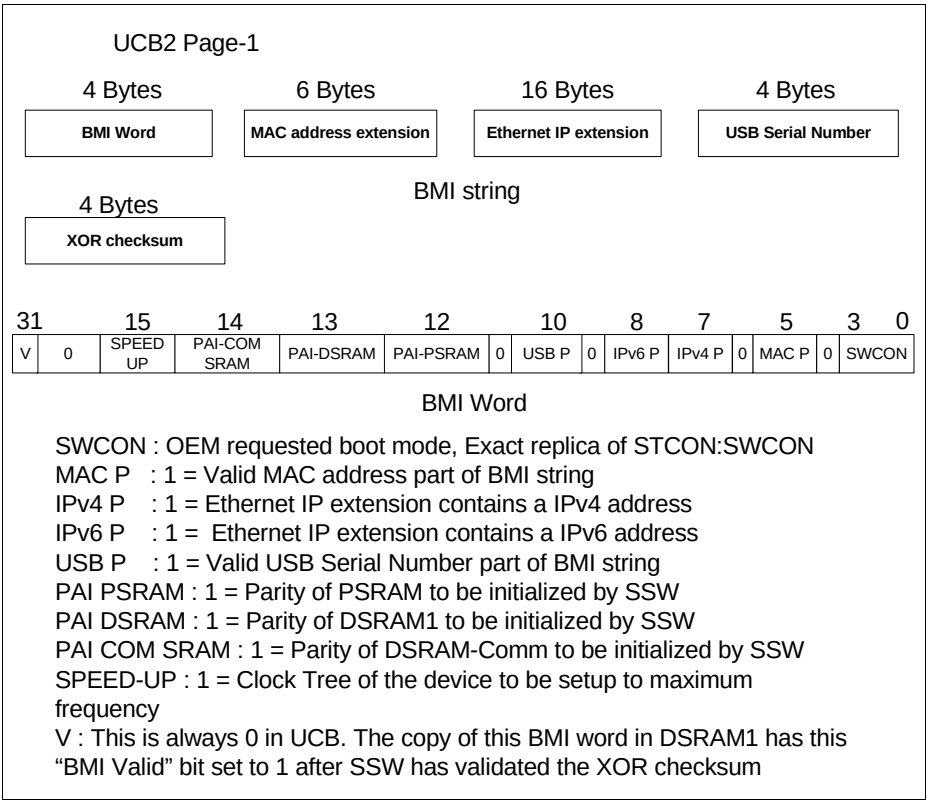


Figure 26-15 BMI String layout

The BMI string is written into UCB2-Page1 by OEM. SSW upon a reset (and any reset) copies the BMI string into a 64 byte reserved location on DSRAM-1. A 4 byte XOR checksum is appended to the string.

Of the 64 bytes, the BMI string is stored starting from the lowest address with the BMI word stored first followed by the MAC address, IP address and USB serial number.

All elements of the BMI string and the XOR checksum are stored in little-endian format.

Details of actions taken by SSW are listed in the **Figure 26-16**.

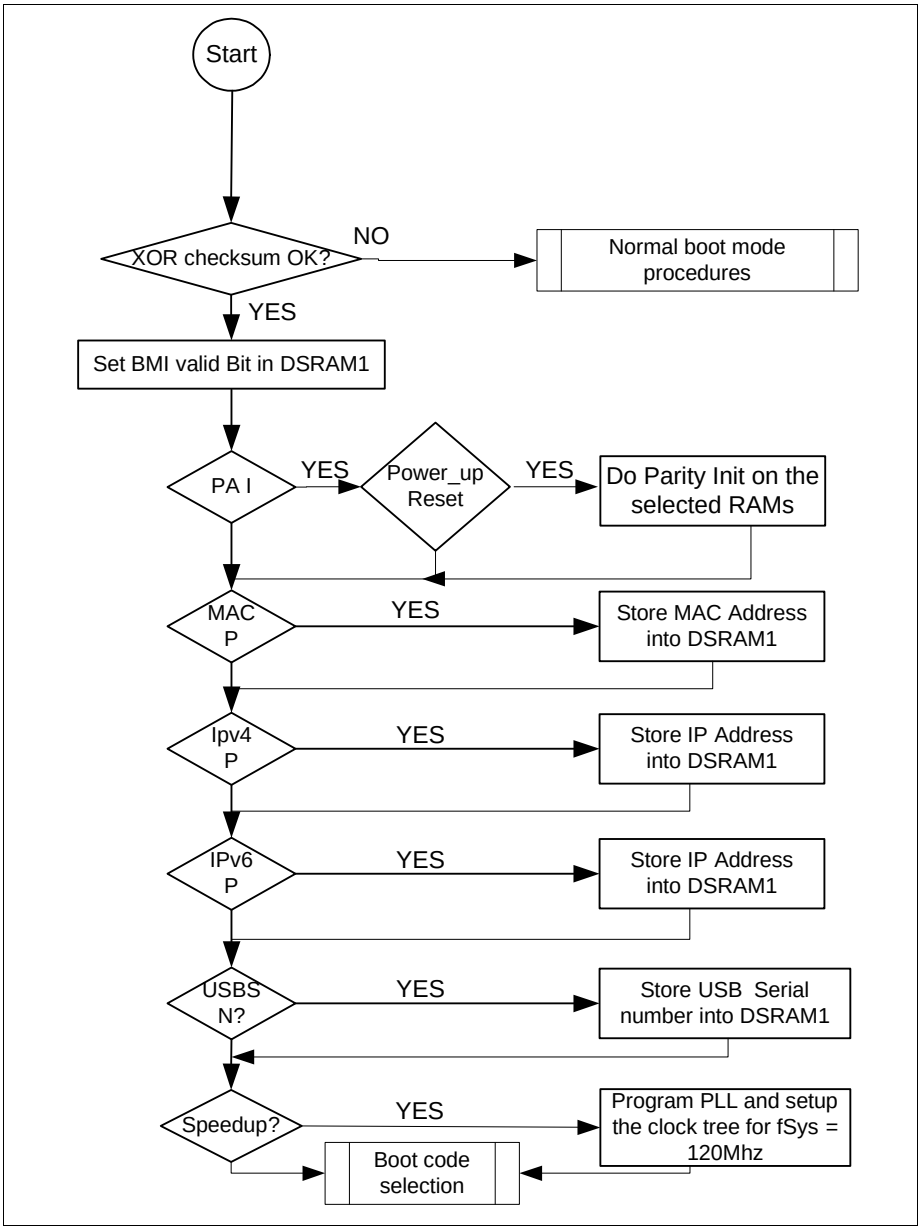


Figure 26-16 BMI actions

User actions for BMI

User must:

- Flash BMI string into the User Configuration Block (UCB2-Page1)
- Configure boot mode pins as described in **Table 26-1**
- Reset the device

26.3 Debug behavior

This chapter describes the Halt after reset (HAR), flash protection, debugger access and diagnostics monitor mode features.

26.3.1 Boot modes and hardware debugger support

The SSW cannot be debugged. It is not possible to halt the CPU after a reset until the SSW has finished its execution. The SSW disables the coresight module thus inhibiting installation of breakpoints and watchpoints. All memory acceses that can otherwise be requested over the debug bus are also inhibited. The coresight module is enabled subject to certain conditions at the time when SSW has finished its activities and is about to cede control to user applications.

Halt after reset feature

Halt after reset feature results in halting of the CPU when the first instruction from the user program enters the CPU pipeline. This is implemented only for Normal boot mode, ABM-0 and ABM-1. It is further applicable only for a PORST case. A breakpoint is installed at the address retrieved from reset vector location of the application exception vector table. However for this to happen, the hardware debugger is expected to have registered its request with coresight for halting the CPU.

The CPU simply cannot be halted while the SSW executes. The hardware debugger can only register a halt request with the coresight while the SSW is executing. This request is picked up by SSW towards the end of its execution resulting in installation of breakpoint on the first user application instruction.

Debugger support after SSW execution

Debugger support is enabled after SSW execution. But this is subject to the state of flash protection. As long as the user flash is not protected, user programs can be debugged. Should the SSW detect that flash has been password protected, it disables the coresight (ARM debug) interface. Hence debugging is not possible on flash protected devices.

Flash access after SSW execution

Special attention must be paid to the table below. Flash access is unconditionally provided for Normal and ABM boot modes regardless of flash protection. CPU can fetch CODE and RO-DATA from the program flash for the two boot modes. Flash access is however only conditionally permitted for BSL and PSRAM boot modes. **Table 26-3** lists the criteria for flash and debugger accesses.

Table 26-3 Flash and Debug access policy

HWCON/SWCON	Flash protected?	Flash access	Debugger access
BSL	N	Y	Y
BSL	Y	N	N
Normal/ABM	N	Y	Y
Normal/ABM	Y	Y	N
Boot from PSRAM	N	Y	Y
Boot from PSRAM	Y	N	N

Empty flash and debugger behavior

SSW executes a tight loop upon detecting empty flash. A value of 00000000_H at $0C000004_H$ (Reset vector) indicates empty flash. A hardware debugger can be attached subsequently and appropriate measures taken.

26.3.2 Failures and handling

It is possible to find out the progress made by SSW during its execution and also the reason for boot failures.

Diagnostics Monitor Mode (DMM)

During its course of execution, SSW has either something normal to trace or an error to report. As access to the Coresight system is disabled during SSW execution, it is imperative that a backchannel mechanism is provisioned for aforesaid monologue from the SSW.

The SCU provides a dedicated register SCU_TSW0 which the SSW can write to. SCU_TSW0 is in fact mirrored by the TCU (Test Control Unit) and the latter's contents can be conveniently scanned out.

This register is known as the TRACE_ERROR_REG and the JTAG instruction for accessing it is the FW_TRACE_ERR (63_H). Please contact your Infineon representative for more details on TCU.

Errors classified as recoverable result in a watchdog reset. Fatal errors require a PORST. The power consumption is reduced to a minimum level.

Figure 26-17 represents the concept pictorially.

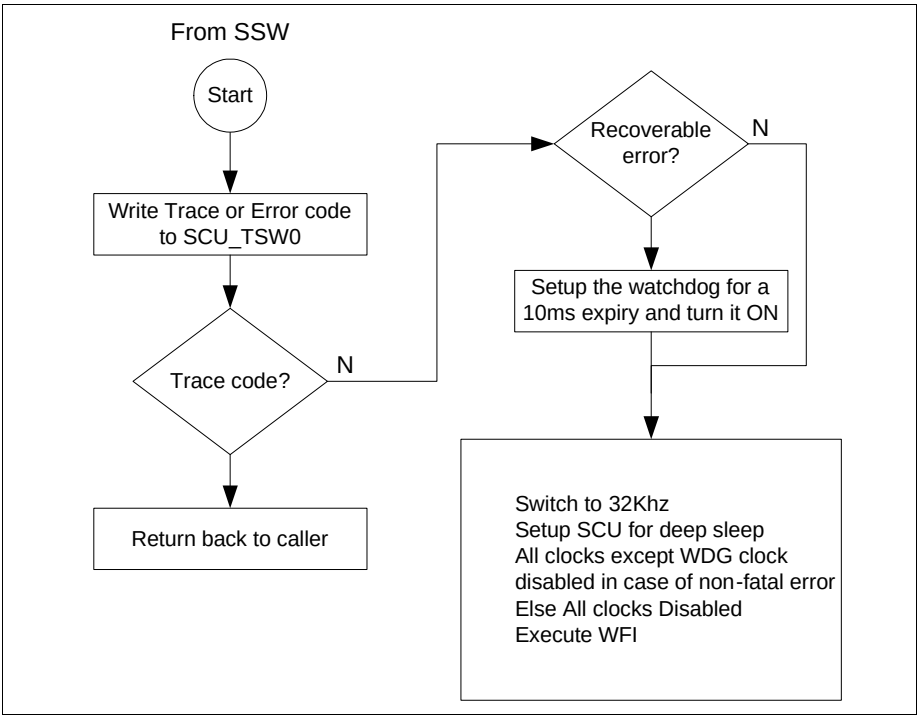


Figure 26-17 Diagnostics monitor mode

Encoding of the trace word

SSW can write a 32 bit word representing either a trace code or an error code. TSW0[15:0] represents the code. TSW0[17:16] represent the code type.

A code type of 0b00 represents invalid code contents, 0b01 represents a trace code, 0b10 represents a fatal error and 0b11 represents a non fatal error.

TSW0[31:18] bits are currently reserved.

List of errors and their classification

Table 26-4 lists errors that lead to SSW launching DMM.

Table 26-4 Error events and codes

Event	Severity	16 bit Code
Flash rampup error	Fatal	1_H
FCS CRC mismatch	Fatal	2_H
Invalid SWCON	Fatal	3_H
MBIST error	Fatal	4_H
PSRAM boot header CRC mismatch	Fatal	5_H
ABM header CRC mismatch	Fatal	6_H
Invalid bootcode	Fatal	7_H
NMI exception	Fatal	8_H
Hardfault exception	Fatal	9_H
Memory management exception	Fatal	A_H
Busfault exception	Fatal	B_H
Usage fault exception	Fatal	C_H
ASC baudrate calculation error	Fatal	D_H
Invalid BMI password error	Fatal	E_H
EVR rampup error	Fatal	F_H
ASC BSL receive error	Fatal	10_H
CAN BSL errors	Fatal	11_H

26.4 Power, Reset and Clock

SSW operates at 24MHz. The fast internal oscillator provides the system clock frequency (f_{OFI}). Frequency scaling is conditionally performed in BMI boot mode of operation based on the BMI word.

Registers modified by SSW

SSW during its course of execution modifies default register settings of a few registers. They are listed in **Table 26-5**.

Table 26-5 Registers modified by SSW

Startup mode	Register	Bitfield	Value
All	CPU_CPACR	23:20	1111_b
All	CPU_SHCR	18:16	111_b
All	CPU_CCR	4:3	11_b
All	SCU_PRSTAT2	6	0_b
All	FCE_CLC	1:0	00_b
All	FCE_CFG0	10:8	000_b
All	FCE_IR0	31:0	Dependent on calculations
All	FCE_CRC0	31:0	Dependent on calculations
All	FCE_RES0	31:0	Dependent on calculations
All	P1_IOCR4	15:11	10010_b
ASC BSL	SCU_PRSTAT0	11	0_b
ASC BSL	U0C0_KSCFG	0	1_b
ASC BSL	U0C0_BRG	14:10	11111_b
ASC BSL	U0C0_BRG	25:16	Dependent on detected baud rate
ASC BSL	U0C0_FDR	15:14	01_b
ASC BSL	U0C0_FDR	9:0	Dependent on detected baud rate
ASC BSL	U0C0_FDR	25:16	Dependent on detected baud rate
ASC BSL	U0C0_DX0CR	2:0 9	001_b 1_b
ASC BSL	U0C0_CCR	0 12:8	1_b 00111_b
ASC BSL	U0C0_SCTR	1 9:8 21:16 27:24	1_b 01b 000111b 0111b
ASC BSL	U0C0_TCSR	11:10 8	01_b 1_b

Table 26-5 Registers modified by SSW (cont'd)

Startup mode	Register	Bitfield	Value
ASC BSL	U0C0_CCR	9:8 3:0	01_b 0010_b
ASC BSL with Frequency scaling (FS)	SCU_OSCHPCTRL	5:4	00_b
ASC BSL with FS	SCU_PLLCON0	1:0 16	00_b 0_b
ASC BSL with FS	SCU_PLLCON1	22:16 14:8	1_b 1001_b
ASC BSL with FS	SCU_PLLCON2	0	0_b
ASC BSL with FS	SCU_SYSCLKCR	17:16	01_b
ASC BSL with FS	SCU_PLLSTAT	9:0	1110100100_b
CAN BSL	SCU_OSCHPCTRL	5:4 20:16	00_b Dependent on external clock frequency
CAN BSL	SCU_PLLCON0	17:16	00_b
CAN BSL	SCU_SYSCLKCR	17:16	01_b
CAN BSL	SCU_PLLSTAT	9:0	1110010101_b
CAN BSL	SCU_PRSTAT1	4	0_b
CAN BSL	CAN_CLC	1:0	00_b
CAN BSL	CAN_FDR	9:0 15:14 25:16	1111111111_b 01b Dependent on baud rate
CAN BSL	CAN_NPCR1	2:0	011_b
CAN BSL	CAN_MOAR0/1	28:0	Dependent on data
CAN BSL	CAN_MOAMR0/1	27:0	$1FFFFFFF_H$
CAN BSL	CAN_MOFCR0/1	27:24	1000_b
CAN BSL	CAN_MODATAL0/1, CAN_MODATAH0/1	31:0	Dependent on data
CAN BSL	P1_IOCR4	7:3	10010_b
CAN BSL	CAN_NCR0/1	0	0_b

Table 26-5 Registers modified by SSW (cont'd)

Startup mode	Register	Bitfield	Value
CAN BSL	CAN_NBTR0/1	5:0,14:8 7:6	Frequency/Baud rate dependent 11b
CAN BSL	CAN_NFCR0/1	20:19	10_b
CAN BSL	CAN_LIST1/2	24 23:16 15:8	0_b 1H 1H
CAN BSL	P1_IOCR4	15:11	10001_b

Debug and Trace System

27 Debug and Trace System (DBG)

The XMC4500 Series Microcontrollers provide a large variety of debug, trace and test features. They are implemented with a standard configuration of the ARM CoreSight™ module together with a daisy chained standard TAP controller. Debug and trace functions are integrated into the ARM Cortex-M4. The debug system supports serial wire debug (SWD) and trace functions in addition to standard JTAG debug and parallel trace.

References

[19] Cortex-M4 Technical Reference Manual

[20] CoreSight™ ETM-M4 Technical Reference Manual

[21] CoreSight™ Technology System Design Guide

[22] CoreSight™ Components Technical Reference Manual

[23] ARM Debug Interface v5 Architecture Specification

[24] Embedded Trace Macrocell Architecture Specification

[25] ARMv7-M Architecture Reference Manual

27.1 Overview

The Debug and Trace System implements ARM CoreSight™ debug and trace features with the objective of debugging the entire SoC. The CoreSight™ infrastructure includes a debug subsystem and a trace subsystem. The debug functionality includes processor halt, single-step, processor core register access, Vector Catch, unlimited software break points and full system memory access. The debug function includes a breakpoint unit supporting 2 literal comparators and 6 instruction comparators and a watchpoint unit supporting 4 watchpoints. The processing element (CPU) is paired with an instruction/data ETM (ETM-M4). CoreSight™ enables different trace sources to be enabled into one stream. The unique trace stream, marked with suitable identifiers and timestamps. Trace can be done using either a 4-bit parallel or a Serial Wire interface. Less data can be traced with Serial Wire interface, but only one output pin is required for application. Parallel trace has a greater bandwidth, but uses 5 more pins.

Features

The accurate Debug and Trace System provides the following functionality:

- Serial Wire Debug Port (SW-DP)
- JTAG Debug Port (SWJ-DP)
- Flash Patch Breakpoint (FPB)
- Data Watchpoint and Trace (DWT)
- Embedded Trace Module (ETM)

- Instrumentation Trace Macrocell (ITM)
- Trace Port Interface Unit (TPIU)
- Halt after reset (HAR)

Note: Please refer to ARM Reference Documentation Cortex-M4-r0p0 for more detailed information on the debug and trace functionality.

Application Mapping

Table 27-1 Debug System available features mapped to functions

SW-DP	Provides Serial Wire Debug, which allows to debug via 2 pins. Instrumentation Trace is provided via a third pin.
SWJ-DP	This debug port provides native JTAG debug capabilities.
FPB	The FPB implements hardware breakpoints and patches code and data from code space to system space
DWT	Implemented watch points, trigger resources, and system profiling. The DWT contains four comparators that can be configured as a hardware watchpoint, an ETM trigger, a PC sampler event trigger or a data address sampler event trigger, data sampler, interrupt trace and CPU statistics
ETM	The ETM provides Instruction Trace capabilities
ITM	Application driven trace source, supports printf style debugging. The ITM generates trace information as packets out of four sources (Software Trace, Hardware Trace, Time Stamping and Global System Time Stamping).
TPIU	The TPIU encodes and provides trace information to the debugger. As ports the single wire viewer (TRACESWO) or 4-bit Trace Port (TRACEDATA[3:0], TRACECLK) can be used.
HAR	Allows to halt the CPU before application code is entered.

Block Diagram

The Debug and Trace system block diagram is shown in **Figure 27-1**.

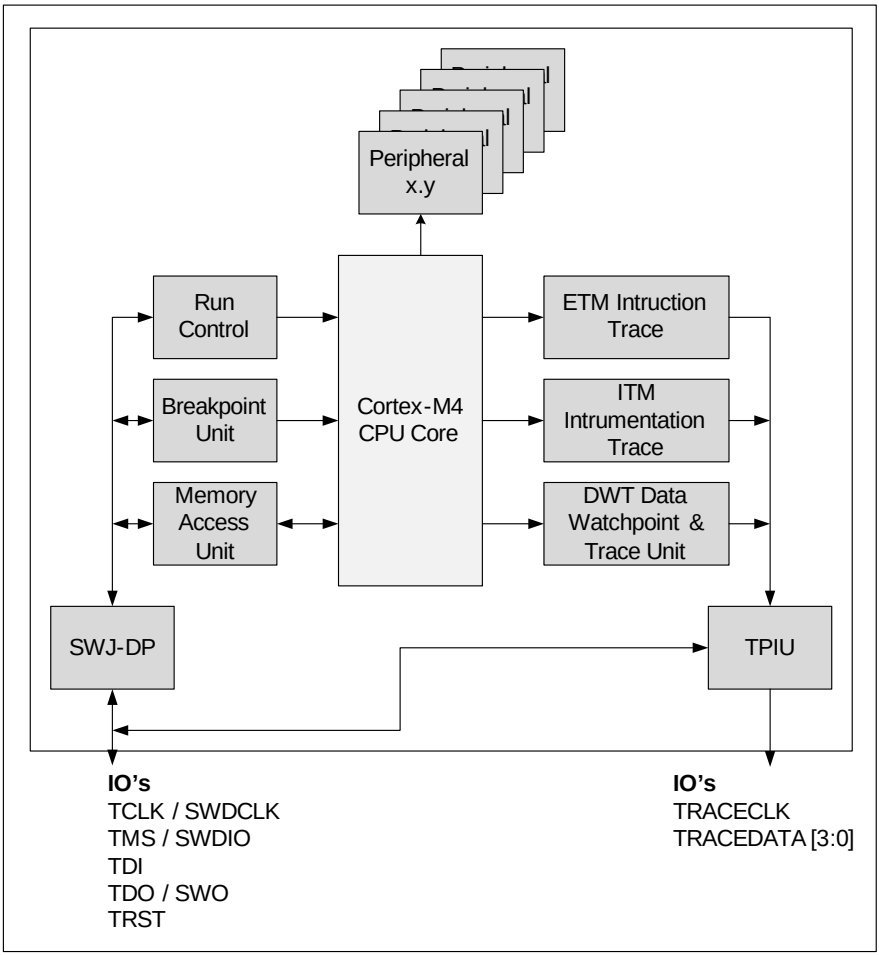


Figure 27-1 Debug and Trace System block diagram

27.2 Debug System Operation

The Debug System provides general debug options and additional trace functions. Debug options are based on break points and CPU halt. The trace capability supports data access trace and instruction execution trace.

27.2.1 Flash Patch Breakpoint (FPB)

The FPB implements hardware breakpoints. Six instruction comparators can be configured to generate a breakpoint instruction to the CPU on a match. The original M4 code patch function is not available.

27.2.2 Data Watchpoint and Trace (DWT)

The four DWT comparators can be configured to generate PC sampling packets at defined intervals, PC or Data watch point packets and a Watch point event to halt the CPU. To enable the features, the DWT provides counters with clock cycle, folded instructions, load store unit operation, sleep cycles, clock per instruction and interrupt overhead count.

27.2.3 Instrumentation Trace Macrocell (ITM)

The ITM supports printf style debugging and is an application trace source. The ITM is available to trace application software execution, and allows to emit diagnostic system information. Three different sources are supported to emit trace information as packet, which are software trace, hardware trace and time stamping. The software trace allows software to write directly to ITM stimulus register using printf function. For hardware trace the ITM emits packets generated by the DWT. Relative to packets the ITM emits timestamps, which are generated by a 48-bit counter. The packets emitted by the ITM are output to the trace port interface (TPIU). The TPIU formatter adds some extra packets and then outputs the complete packet sequence to the debugger. The ITM function can be activated by the TRCEN register bit, which is located in the Debug Exception and Monitor control register. ITM data can also be transferred using the Serial Wire interface. The ITM data are the only Trace source data which can be transferred via the Serial Wire interface.

27.2.4 Embedded Trace Macrocell (ETM)

The ETM enables program execution reconstruction. As a short description, data are traced using the DWT component or the ITM, whereas instructions are traced using ETM. The ETM transmits the information as packets and is triggered by internal resources. These internal trigger resources must be programmed independently and the trigger source is selected using the available Trigger Event Register. Available events are address match, provided by an address comparator or a logic equation between two events. The trigger source are one of the DWT module provided comparators (four are available). This allows to monitor clock cycle matching events and data address matching events. The packets which are generated by the ETM are transmitted on the Trace Port output Unit (TPIU). The TPIU adds some extra packets and transmits the complete packet sequence to the debug tool.

27.2.5 Trace Port Interface Unit (TPIU)

The TPIU collects on-chip trace data from ITM and ETM and sends this debug data to the external trace capture hardware. The TPIU uses dedicated trace ports. Maximum available trace ports are four.

27.3 Power, Reset and Clock

For requirements based on power, reset and clock signaling consult CoreSight™ Technology System Design Guide **[21]** for detailed information. Note, there is no power management implemented for the debug system.

27.3.1 Reset

Reset and implementation is provided according to the ARM reference documentation **[19]**.

27.3.1.1 CoreSight™ resets

The SWJ-DP and SW-DP register are in the power on reset domain. Besides this reset a tool controlled Debug reset can be generated based on a debug register configuration. Activating the reset request in the debug control and status register results in an activation of the CTRL/STAT.CDBGRSTREQ. The reset allows a debugger to reset the debug logic in a CoreSight™ system without affecting the functional behavior of the target system. The Debug logic is reset by system reset, if no tool is registered at the debug system.

System Reset (Warm Reset)

A System or warm reset initializes the majority of the processor, excluding NVIC and debug logic, (FPB, DWT, and ITM). The System reset affects the Debug system only, if the tool is not registered (CTRL/STAT.CDBGPWRUPREQ not set).

SWJ-DP reset

nTRST reset initializes the state of the JTAG SWJ-DP TAP controller. Normal processor operation is not affected.

SW-DP reset

Only the PORESETn reset initializes the SW-DP and the other debug logic.

Normal operation

During normal operation the resets PORESETn SYSRESETn and DAPRESETn are deasserted. If the SWJ-DP or SWDP ports are not in use, nTRST must be tied to 0 or 1.

27.3.1.2 Serial Wire interface driven system reset

The CTRL/STAT.SYSRESETREQ allows to reset the CPU core in Serial Wire interface mode. The CTRL/STAT.SYSRESETREQ is asserted when the CTRL/STAT.SYSRESETREQ bit of the Application Interrupt and Reset Control register is set. This causes a reset, intended to force a large system reset of all major components, except for debug logic. **[22]**.

27.4 Initialization and System Dependencies

This chapter provides information about Debug access and Flash protection, Halt After Reset sequences, Halting Debug and Peripheral suspend, Timestamping for Trace and the available tool interfaces and how to enable the tool interface. Additionally the ID Codes and the ROM Table is presented, including the values a debug tool uses to identify the device and available debug functionality. The last section shows the JTAG Debug instruction code definition.

27.4.1 Debug accesses and Flash protection

The Flash has integral measures to protect its content from unauthorized access and modification, see the section Read and Write Protection in the PMU chapter and startup chapter. Special care is taken, that the debugger can't bypass this protection. Because of this, per default and after a system reset the debug interface is disabled. Depending on the boot scenario and the Flash protection setup, the Startup Software enables the debug interface. If it is left disabled, the user can define a protocol, e.g. a password-protected unlock sequence via an SPI port, to enable the debug interface.

27.4.2 Halt after reset

The XMC4500 product supports two different possibilities of halt after reset. One after a power on reset (PORST), which is called Cold Reset Halt situation or Halt after reset (HAR) and the second one after a system reset, which is called Warm Reset Halt situation. For a HAR the debug tool has to register during the start up software (SSW) execution time. The SSW halts at the end of SSW on a successful tool registration, before application code is entered. The Warm Reset Halt is based on brake point setting on the very first application code line and is entered by a system reset.

For security reasons it is required to prevent a debug access to the processor before and while the boot firmware code from ROM (SSW) is being executed. A bit DAPSA, (DAP has system access) in the SCU is implemented, allowing the access from CoreSight™ debug system to the processor core. The default value of this bit is disabled debug access. The register is reset by System Reset. The System Reset disables the debug

access each time SSW is being executed. At the end of the SSW the DAPSA is enabled always (independent of any other register setting or signaling value), to allow debug access to the CPU. A tool accessing the SoC during the SSW execution time reads back a zero and a write is going to a virtual, none existing address.

In a HAR situation the system always comes from a PORST. A tool can register for a HAR by sending a pattern enabling the CTRL/STAT.CDBGPWRUPREQ and the DHCSR.C_DEBUGEN **[25]** register inside the CoreSight™ module. The registration has to be done after the PORST in a time interval smaller then the time the SSW is executed. This registration time is called tssw and the bits must be set before tssw is expired. The timing value for tssw needs to be handled by the debug tool software (no hardware timer available) and informs the tool software during a Cold Reset about the time frame available to set the HAR conditions.

The following figure (**Figure** HAR - Halt After Reset) shows the software flow based on the modules participating to the HAR.

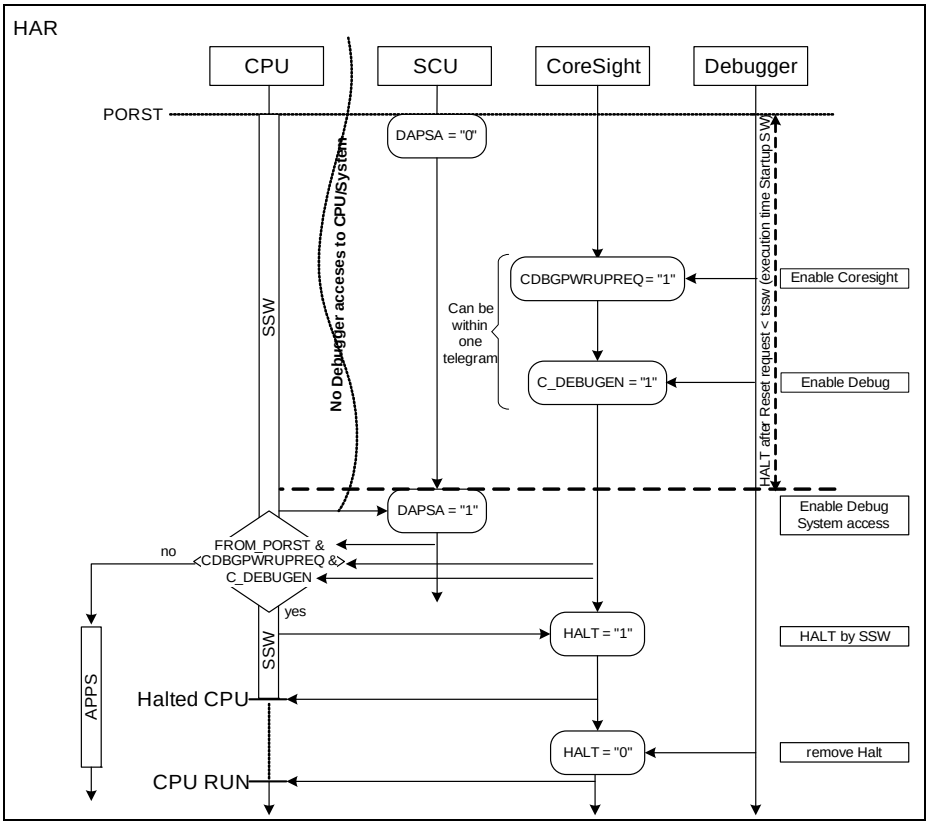


Figure 27-2 HAR - Halt After Reset

A Halt after system reset (Warm Reset) can be achieved by programming a break point at the first instruction of the application code. Before a Warm Reset the CTRL/STAT.CDBGPWRUPREQ and the DHCSR.C_DEBUGEN setting has to be ensured. After a system reset, the HAR situation is not considered, as the reset is not coming from PORST.

Note: The CTRL/STAT.*CDBGPWRUPREQ and* DHCSR.*C_DEBUGEN does not have to be set after a system reset, if they have already been set before.*

A tool hot plug condition allows to debug the system starting with a tool registration (setting CTRL/STAT.CDBGPWRUPREQ register) and a debug system enable (setting the DHCSR.C_DEBUGEN register). Afterwards break points can be set or the CPU can directly by HALTED by an enable of C_HALT.

The following Figure (**Figure** Hot Plug and Warm Reset) illustrates the debug tool HOT PLUG situation or the Halt after Warm Reset (system reset) and how to proceed to come to a Halt situation.

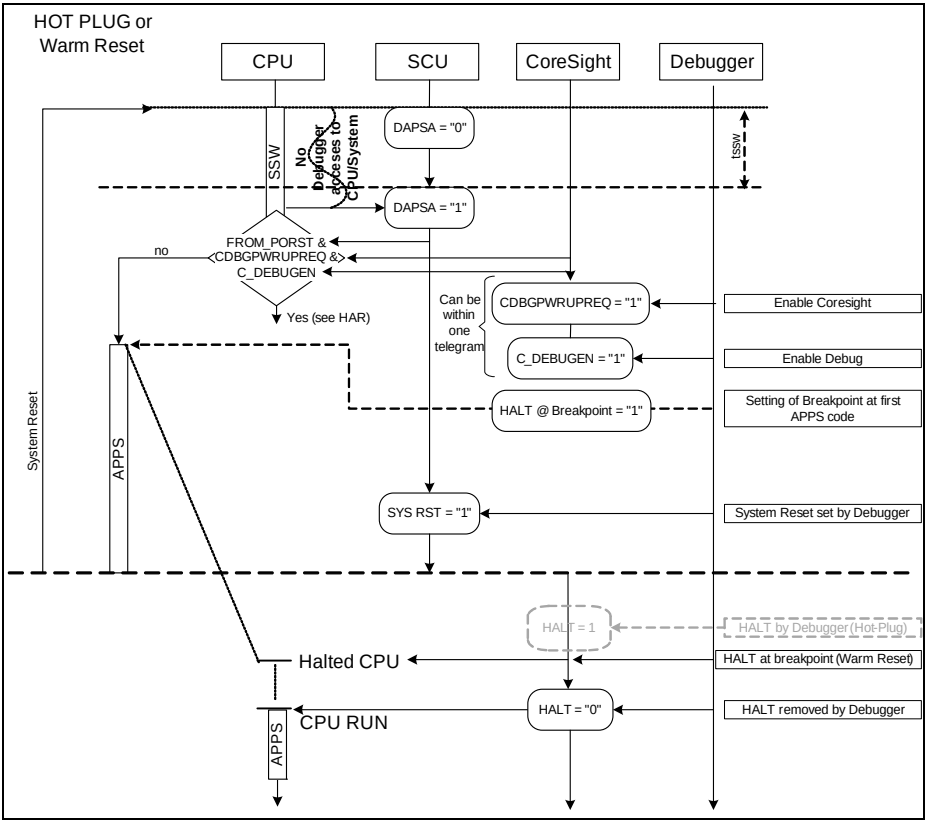


Figure 27-3 HOT PLUG or Warm Reset

27.4.3 Halting Debug and Peripheral Suspend

If the program execution of the CPU is stopped by the debugger, e.g. with a breakpoint, it is possible to suspend the peripherals as well. This allows to debug critical states of the whole microcontroller. It is particularly useful, e.g. to suspend the Watchdog Timer as it can't be serviced by a halted CPU.

In other cases it is important to keep some peripherals running, e.g. a PWM or a CAN node, to avoid system errors or even critical damage to the application. Because of this,

the peripherals allow to configure how they behave when the CPU enters the halting debug mode.

It can be decided at the peripheral to support a Hard Suspend or a Soft Suspend. At a Hard Suspend situation the clock at the peripheral is switched off immediately, without waiting on acknowledge from the module. At a soft suspend the peripheral can decide when to suspend, usually at the end of the actual active transfer.

A Watchdog timer is only running when the suspend bus is not active. This is particularly useful as it can't be serviced by a halted CPU. A configuration option is available, which allows to enable the Watchdog timer also during suspend. This allows to debug Watchdog behavior, if a debugger is connected.

The user has to ensure, that always only those peripherals are sensitive to suspend, which are intended to be. To address this, each peripheral supporting suspend does have an enable register which allows to enable the suspend feature. The following table (**Table 27-2**) shows the peripherals, supporting or not supporting peripheral suspend or detailed information on the peripheral suspend behavior during soft suspend can be found at the respective peripheral chapter.

Table 27-2 Peripheral Suspend support

Peripheral	Supported	Default mode	Hard Suspend	Soft Suspend	Suspend Reset
RTC	no	- - -	- - -	- - -	- - -
WDT[1)]	yes	active	yes	no	system reset
LEDTS	yes	not active	yes	no	debug reset
SDMMC	no	- - -	- - -	- - -	- - -
EBU	no	- - -	- - -	- - -	- - -
ETH	no	- - -	- - -	- - -	- - -
USB	no	- - -	- - -	- - -	- - -
USIC	yes	not active	no	yes	debug reset
MultiCAN	yes	not active	yes	yes	debug reset
VADC	yes	not active	yes	yes	debug reset
DSD	yes	not active	yes	yes	debug reset
DAC	no	- - -	- - -	- - -	- - -
CCU4[1)]	yes	not active	yes	yes	system reset
CCU8[1)]	yes	not active	yes	yes	system reset
POSIF[1)]	yes	not active	yes	yes	system reset

1) A system reset results in a suspend configuration loss. If it is required to have suspend configuration available after system reset, a HW breakpoint has to be set at the first instruction of use code and reconfiguration of suspend behavior at the peripheral has to be performed again.

27.4.4 Timestamping

A 48-bit timestamping capability is required by the debug system in order to get accurate correlation between the ETM trace and trace data from ITM and DWT. This is also named global timestamping. Timestamp packets encode timestamp information, generic control and synchronization information. The Timestamp counter is a free running global counter.

A synchronization packet is a timestamp packet control. It is emitted at each DWT trigger (DWT must be configured to trigger the ITM).

27.4.5 Debug tool interface access (SWJ-DP)

Debug capabilities can be accessed by a debug tool via Serial Wire (SW) or JTAG interface (JTAG - Debug Port). By default, the JTAG interface is active. The User might switch to SW interface as the full JTAG pins are not available to the user. To enable SW interface a dedicated JTAG sequence on TMS/SWDIO and TCK/SWCLK is required to to switch to the Serial Wire Debug interface. A successful sequence disables JTAG interface and enables SW interface.

The sequences to do this are described in **Chapter 27.4.5.1** and **Chapter 27.4.5.2**

27.4.5.1 Switch from JTAG to SWD

The sequence for switching from JTAG to SWD is:

- Send 50 or more TCK cycles with TMS = 1
- Send the 16-bit sequence on TMS = 1110011110011110 (0xE79E LSB first)
- Send 50 or more TCK cycles with TMS = 1

27.4.5.2 Switch from SWD to JTAG

The sequence for switching from SWD to JTAG is:

- Send 50 or more TCK cycles with TMS = 1
- Send the 16-bit sequence on TMS = 1110011100111100 (0xE73C LSB first)
- Send 50 or more TCK cycles with TMS = 1

27.4.6 ID Codes

Available ID Codes are used by a debug tool to identify the available debug components during tool setup.

Table 27-3 ARM CoreSight™ Component ID codes

ID	Value	Description
CPUID	410F C240$_H$	CPUID to identify the Cortex M4 core
AHBAPID	2477 0011$_H$	Identify the AHB-AP is available
IDCONFIG	101D B083$_H$	The ARM TAP IDCONFIG
SWJ-DP	4BA0 0477$_H$	The ARM JTAG ID
SW_DP	2BA0 1477$_H$	The ARM SW-DP ID

27.4.7 ROM Table

To identify Infineon as manufacturer and XMC4500 as device, the ROM table has to be read out.

Table 27-4 PID Values of XMC4500 **ROM Table**

Name	Offset	Value	Description	Reference
PID0	FE0$_H$	11011011$_B$	Peripheral ID0	Part Number [7:0]
PID1	FE4$_H$	00010001$_B$	Peripheral ID1	bits [7:4] JEP106 ID code [3:0] bits [3:0] Part Number [11:8]
PID2	FE8$_H$	00011100$_B$	Peripheral ID2	bits [7:4] Revision bit [3] == 1: JEDEC assigned ID fields bits [2:0] JEP106 ID code [6:4]
PID3	FEC$_H$	00000000$_B$	Peripheral ID3	bits [7:4] RevAnd, minor revision field bits [3:0] if non-zero indicate a customer-modified block
PID4	FD0$_H$	00000000$_B$	Peripheral ID4	bits [7:4] 4KB count bits [3:0] JEP106 continuation code

27.4.8 JTAG debug port

A standard JTAG IEEE1149 Boundary-Scan statemachine is implemented. It includes all mandatory instructions (SAMPLE/PRELOAD, EXTEST) and also some optional instructions (IDCODE, CLAMP, HIGHZ), and some custom/optional instructions are implemented. The optional RUNBIST instruction is not used and will be treated as BYPASS. No USERCODE-instruction is implemented, it will show only 0 values

Table 27-5 JTAG INSTRUCTIONS

Opcode	Range	Type	Instruction
0000 0000	00H	Reserved	
0000 0001	01H - 08H (8 instr.)	IEEE1149 Boundary-Scan	INTEST
0000 0010			SAMPLE/PRELOAD
0000 0011			RUNBIST
0000 0100			IDCODE
0000 0101			USERCODE
0000 0110			CLAMP
0000 0111			EXTEST
0000 1001- 0000 1111	Reserved		
0001 0001 - 0100 1111	11H - 4FH 63instr.		
1111 1111	FFH	IEEE 1149.1	BYPASS

JTAG Instruction Definition

BYPASS

The BYPASS instruction bypasses all serial register path, which are accessed over the JTAG TAP port. In the Capture-DR state the rising edge of the TCK clock sets the bypass register to zero. The bypass instruction is primarily used to allow a shorter access path to cascaded devices when the JTAG interface connects a number of devices in series. All unused instructions must connect the TAP controller bypass register output to the Test Access Port output TDO. The BYPASS instruction has no effect on the operation of the device.

27.5 Debug System Registers

For CoreSight™ register overview and detailed register definitions, please refer to ARM documentation CoreSight™ Components Technical Reference Manual **[22]** and ARMv7-M Architecture Reference Manual **[25]**.

27.6 Debug and Trace Signals

XMC4500 MC Product family provides debug capability using ARM CoreSight™ Debug port SWJ-DP. SWJ-DP includes two debug interfaces namely JTAG Debug Port (JTAG-DP) and Serial Wire Debug Port (SW-DP).

The JTAG-DP interface has 4 (without Reset pin TRST for low pin package) or 5 Pins, see **Table 27-6**.

The serial wire Debug Port has 2 (Clock + Bidirectional data) or 3 pins (Clock + Bidirectional data + Asynchronous Trace output). They are overlaid on the JTAG-DP pins (TCK, TMS and TDO) for efficient use of package pins, see **Table 27-7**.

Additionally 5 ETM trace port output signals (TRACECLK,TRACEDATA[3:0]) are available, see **Table 27-8**.

Sub chapters below additionally describe pull resistors to the IO and the suggested debug connector pin assignment.

Table 27-6 JTAG Debug signal description

Signal	Direction	Function
TCK	I	JTAG Test Clock. This pin is the clock for debug module when running in JTAG debug mode.
TMS	I	JTAG Test Mode Select. The TMS pin selects the next state in the TAP state machine.
TDI	I	JTAG Test Data In. This is the serial data input for the shift register
TDO	O	JTAG Test Data Output. This is the serial data output from the shift register. Data is shifted out of the device on the negative edge of the TCK signal
$\overline{\text{TRST}}$	I	JTAG Test reset.

Table 27-7 Serial Wire Debug signal description

Signal	Direction	Function
SWDCLK	I	Serial Wire Clock. This pin is the clock for debug module when running in Serial Wire debug mode.
SWDIO	I / O	Serial Wire debug data IO. Used by an external debug tool to communicate with and control the Cortex-M4 CPU.
SWO	O	Serial Wire Output. The SWO pin provides data from the ITM and/or ETM for an external debug tool to evaluate the instrumentation trace.

Table 27-8 ETM Trace Port signal description

Signal	Direction	Function
TRACECLK	O	Trace Clock. Provides the sample clock for trace data on the TRACEDATA pins when tracing is enabled by an external tracing tool.
TRACEDATA[3:0]	O	Trace Data bits 3 to 0. Provide ETM trace data when tracing is enabled by an external debug tool. The debug tool can interpret the compressed information and make it available to the user.

ETM Trace port output enable

The ETM module allows to control the trace port signal on a shared GPIO at the IO port level. The enabling is done by the tool software, by configuring in the ETM main Control register (ETMCR) **[20]**.

27.6.1 Internal pull-up and pull-down on JTAG pins

It is a requirement to ensure none floating JTAG input pins, as they are directly connected to flip-flops controlling the debug function. To avoid any uncontrolled I/O voltage levels internal pull-up and pull-downs on JTAG input pins are provided.

- TRST: Internal pull-down
- TMS/SWDIO: Internal pull-up
- TCK/SWCLK: Internal pull-down

27.6.2 Debug Connector

The suggested connector is the Cortex Debug and ETM connector, which is a 20-pin connector. The connector supports JTAG debug, Serial-Wire debug, Serial Wire viewer (via SWO connection when Serial Wire debug mode is used) and instruction trace operations. The following **Figure 27-4** shows the 20-pin connector for debug and trace.

There may be systems, which required HW to detect debugger presence. This can be enabled by using the GNDDetect pin of the Debug and Trace connector. The pin is driven low by the tool, when the tool connector is plugged into the connector at the PCB. GNDDetect for tool detection requires to have a weak pull-up on the PCB, connected to this pin. At the connector it is suggested not to have the KEY pin available as the KEY is used to identify the correct connector pugging.

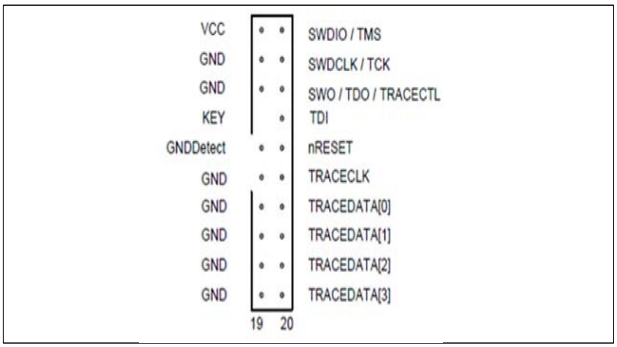


Figure 27-4 Debug and Trace connector

Lists of Figures and Tables

List of Figures

List of Figures

List of Figures

List of Figures

List of Figures

List of Figures

List of Figures

List of Figures

List of Figures

List of Figures

List of Figures

List of Figures

List of Figures

List of Tables

List of Tables

List of Tables

List of Tables

List of Tables

List of Tables

List of Tables

List of Tables

List of Tables

List of Tables

w w w . i n f i n e o n . c o m

Published by Infineon Technologies AG