

C161CS-32R/-L

C161JC-32R/-L

C161JI-32R/-L

16-Bit Single-Chip Microcontroller

16bit

Microcontrollers



Never stop thinking.

**Edition 2001-02**

**Published by Infineon Technologies AG,  
St.-Martin-Strasse 53,  
D-81541 München, Germany**

**© Infineon Technologies AG 2001.  
All Rights Reserved.**

**Attention please!**

The information herein is given to describe certain components and shall not be considered as warranted characteristics.

Terms of delivery and rights to technical change reserved.

We hereby disclaim any and all warranties, including but not limited to warranties of non-infringement, regarding circuits, descriptions and charts stated herein.

Infineon Technologies is an approved CECC manufacturer.

**Information**

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office in Germany or our Infineon Technologies Representatives worldwide.

**Warnings**

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

Infineon Technologies Components may only be used in life-support devices or systems 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.

C161CS-32R/-L

C161JC-32R/-L

C161JI-32R/-L

16-Bit Single-Chip Microcontroller

Microcontrollers



Never stop thinking.

**C161CS/JC/JI****Revision History: V3.0, 2001-02**Previous Version: 2001-01 V2.0 (intermediate version)  
1999-05 V1.0

<b>Page</b>	<b>Subjects (major changes since last revision)<sup>1)</sup></b>
all	Converted to new company layout, figures have been redrawn
<a href="#">1-2</a>	List of derivatives enhanced
<a href="#">2-21</a>	List of protected bits enhanced
<a href="#">3-4</a>	XBUS areas corrected
<a href="#">4-2</a>	Sleep mode added
<a href="#">5-29</a>	Interrupt source control enhanced
<a href="#">6-9</a>	Frequency table adapted
<a href="#">6-10</a>	Description of PLL base frequency improved
<a href="#">7-2</a>	Bit P4LIN added
<a href="#">7-8</a>	Description of temperature compensation removed
<a href="#">7-14</a>	Description of PORT0 control corrected
<a href="#">7-31</a>	Description of $\overline{\text{BHE}}$ during bus hold corrected
<a href="#">7-33</a>	ODP4 enhanced
<a href="#">7-34</a>	Description of alternate Port 4 functions corrected
<a href="#">7-48</a>	CAN/SDLM interface functions added
<a href="#">9-6</a>	Note reworked
<a href="#">9-22, 9-23</a>	Bits BSWCx and EWENx added to register BUSCONx
<a href="#">9-28</a>	Note corrected
<a href="#">9-31, 9-32</a>	Description of bus arbitration improved
<a href="#">9-33</a>	Description of $\overline{\text{BHE}}$ during bus hold corrected
<a href="#">9-35, 9-36</a>	Section "Connecting Bus Masters" improved
<a href="#">9-37ff</a>	XBUS interface description improved
<a href="#">10-6, 10-27</a>	Table enhanced
<a href="#">10-16, 10-33</a>	Figure corrected
<a href="#">10-30</a>	Description of T5M corrected
<a href="#">10-36</a>	CT3 function added to figure
<a href="#">11-13, 11-14</a>	Tables enhanced
<a href="#">13-5</a>	Description of transmission timing improved
<a href="#">13-13</a>	Baudrate tables improved
<a href="#">14-2</a>	Clock path in figure corrected
<a href="#">14-5, 14-6</a>	Time range table and reset source table improved
<a href="#">16-2, 16-3</a>	Description of BSL entry improved
<a href="#">16-7</a>	Baudrate table added

**C161CS/JC/JI****Revision History: V3.0, 2001-02 (cont'd)**Previous Version: 2001-01 V2.0 (intermediate version)  
1999-05 V1.0

<b>Page</b>	<b>Subjects (major changes since last revision)<sup>1)</sup></b>
<b>17-7</b>	Frequency table enhanced
<b>17-14</b>	Description improved (2nd paragraph)
<b>18-4</b>	Sample time control added
<b>19-1</b>	Port 7 added
<b>19-11</b>	Bit timing section rearranged
<b>20-5, 20-9</b>	Description improved
<b>20-10</b>	Description of TXINCE corrected
<b>20-12</b>	Description improved (lower half)
<b>20-25ff</b>	Several bit-descriptions improved
<b>20-43</b>	Location of RXCNTB corrected
<b>21-1</b>	Port 9 added
<b>21-3</b>	Clock count n and BRP value corrected
<b>21-4</b>	Figure improved
<b>22-13</b>	Figure corrected
<b>22-18</b>	Software configuration introduced (see notes)
<b>22-19</b>	Table enhanced for 33 MHz
<b>22-22</b>	Code example corrected
<b>22-23</b>	Address space for RSTCON corrected
<b>23-2</b>	SYSCON1 added (3 <sup>rd</sup> paragraph)
<b>23-21</b>	Frequency range table improved
<b>23-22ff</b>	Description of security mechanism and SW examples reworked
<b>24-5</b>	Linear stack size corrected
<b>25-3</b>	Offset of RH7 corrected
<b>25-4ff</b>	P5DIDIS, RSTCON added, SDLM registers added

<sup>1)</sup> These changes refer to version V1.0, 1999-05.

Controller Area Network (CAN): License of Robert Bosch GmbH

**We Listen to Your Comments**

Any information within this document that you feel is wrong, unclear or missing at all?  
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](mailto:mcdocu.comments@infineon.com)**



<b>Table of Contents</b>		<b>Page</b>
<b>1</b>	<b>Introduction</b> .....	1-1
1.1	The Members of the 16-bit Microcontroller Family .....	1-3
1.2	Summary of Basic Features .....	1-5
1.3	Abbreviations .....	1-8
<b>2</b>	<b>Architectural Overview</b> .....	2-1
2.1	Basic CPU Concepts and Optimizations .....	2-2
2.1.1	High Instruction Bandwidth/Fast Execution .....	2-3
2.1.2	Programmable Multiple Priority Interrupt System .....	2-7
2.2	The On-Chip System Resources .....	2-8
2.3	The On-Chip Peripheral Blocks .....	2-11
2.4	Power Management Features .....	2-19
2.5	Protected Bits .....	2-21
<b>3</b>	<b>Memory Organization</b> .....	3-1
3.1	Internal ROM Area .....	3-3
3.2	Internal RAM and SFR Area .....	3-4
3.3	The On-Chip XRAM .....	3-9
3.4	External Memory Space .....	3-11
3.5	Crossing Memory Boundaries .....	3-12
3.6	Protection of the On-Chip Mask ROM .....	3-13
<b>4</b>	<b>The Central Processing Unit (CPU)</b> .....	4-1
4.1	Instruction Pipelining .....	4-3
4.2	Particular Pipeline Effects .....	4-6
4.3	Bit-Handling and Bit-Protection .....	4-10
4.4	Instruction State Times .....	4-11
4.5	CPU Special Function Registers .....	4-12
<b>5</b>	<b>Interrupt and Trap Functions</b> .....	5-1
5.1	Interrupt System Structure .....	5-2
5.1.1	Interrupt Control Registers .....	5-6
5.2	Operation of the PEC Channels .....	5-12
5.3	Prioritization of Interrupt and PEC Service Requests .....	5-16
5.4	Saving the Status During Interrupt Service .....	5-18
5.5	Interrupt Response Times .....	5-20
5.6	PEC Response Times .....	5-23
5.7	Interrupt Node Sharing .....	5-25
5.8	External Interrupts .....	5-26
5.9	Trap Functions .....	5-31
<b>6</b>	<b>Clock Generation</b> .....	6-1
6.1	Oscillators .....	6-3
6.2	Frequency Control .....	6-7

<b>Table of Contents</b>		<b>Page</b>
6.3	Oscillator Watchdog .....	6-11
6.4	Clock Drivers .....	6-12
<b>7</b>	<b>Parallel Ports</b> .....	<b>7-1</b>
7.1	Input Threshold Control .....	7-2
7.2	Output Driver Control .....	7-4
7.3	Alternate Port Functions .....	7-9
7.4	PORT0 .....	7-12
7.5	PORT1 .....	7-16
7.6	Port 2 .....	7-21
7.7	Port 3 .....	7-26
7.8	Port 4 .....	7-32
7.9	Port 5 .....	7-37
7.10	Port 6 .....	7-40
7.11	Port 7 .....	7-46
7.12	Port 9 .....	7-50
<b>8</b>	<b>Dedicated Pins</b> .....	<b>8-1</b>
<b>9</b>	<b>The External Bus Interface</b> .....	<b>9-1</b>
9.1	Single Chip Mode .....	9-2
9.2	External Bus Modes .....	9-3
9.3	Programmable Bus Characteristics .....	9-12
9.4	$\overline{\text{READY}}$ Controlled Bus Cycles .....	9-18
9.5	Controlling the External Bus Controller .....	9-20
9.6	EBC Idle State .....	9-30
9.7	External Bus Arbitration .....	9-31
9.8	The XBUS Interface .....	9-37
9.8.1	Accessing the On-chip XBUS Peripherals .....	9-38
9.8.2	External Accesses to XBUS Peripherals .....	9-39
<b>10</b>	<b>The General Purpose Timer Units</b> .....	<b>10-1</b>
10.1	Timer Block GPT1 .....	10-1
10.1.1	GPT1 Core Timer T3 .....	10-3
10.1.2	GPT1 Auxiliary Timers T2 and T4 .....	10-12
10.1.3	Interrupt Control for GPT1 Timers .....	10-21
10.2	Timer Block GPT2 .....	10-22
10.2.1	GPT2 Core Timer T6 .....	10-24
10.2.2	GPT2 Auxiliary Timer T5 .....	10-30
10.2.3	Interrupt Control for GPT2 Timers and CAPREL .....	10-38
<b>11</b>	<b>The Asynchronous/Synchronous Serial Interface</b> .....	<b>11-1</b>
11.1	Asynchronous Operation .....	11-5
11.2	Synchronous Operation .....	11-8

<b>Table of Contents</b>		<b>Page</b>
11.3	Hardware Error Detection Capabilities .....	11-10
11.4	ASC0 Baud Rate Generation .....	11-11
11.5	ASC0 Interrupt Control .....	11-15
<b>12</b>	<b>The Async./Synchronous Serial Interface ASC1 .....</b>	<b>12-1</b>
12.1	Asynchronous Operation .....	12-5
12.2	Synchronous Operation .....	12-5
12.3	Hardware Error Detection Capabilities .....	12-6
12.4	ASC1 Baud Rate Generation .....	12-6
12.5	ASC1 Port Control .....	12-6
12.6	ASC1 Interrupt Control .....	12-7
<b>13</b>	<b>The High-Speed Synchronous Serial Interface .....</b>	<b>13-1</b>
13.1	Full-Duplex Operation .....	13-7
13.2	Half-Duplex Operation .....	13-10
13.3	Continuous Transfers .....	13-11
13.4	Port Control .....	13-12
13.5	Baud Rate Generation .....	13-13
13.6	Error Detection Mechanisms .....	13-15
13.7	SSC Interrupt Control .....	13-17
<b>14</b>	<b>The Watchdog Timer (WDT) .....</b>	<b>14-1</b>
14.1	Operation of the Watchdog Timer .....	14-3
14.2	Reset Source Indication .....	14-6
<b>15</b>	<b>The Real Time Clock .....</b>	<b>15-1</b>
<b>16</b>	<b>The Bootstrap Loader .....</b>	<b>16-1</b>
16.1	Entering the Bootstrap Loader .....	16-2
16.2	Loading the Startup Code .....	16-5
16.3	Exiting Bootstrap Loader Mode .....	16-5
16.4	Choosing the Baudrate for the BSL .....	16-6
<b>17</b>	<b>The Capture/Compare Units .....</b>	<b>17-1</b>
17.1	The CAPCOM Timers .....	17-4
17.2	CAPCOM Unit Timer Interrupts .....	17-9
17.3	Capture/Compare Registers .....	17-10
17.4	Capture Mode .....	17-13
17.5	Compare Modes .....	17-14
17.6	Capture/Compare Interrupts .....	17-19
<b>18</b>	<b>The Analog/Digital Converter .....</b>	<b>18-1</b>
18.1	Mode Selection and Operation .....	18-3
18.2	Conversion Timing Control .....	18-13
18.3	A/D Converter Interrupt Control .....	18-15



<b>Table of Contents</b>		<b>Page</b>
<b>19</b>	<b>The On-Chip CAN Interface</b> .....	19-1
19.1	Functional Blocks of the CAN Module .....	19-2
19.2	General Functional Description .....	19-7
19.2.1	CAN Interrupt Handling .....	19-9
19.2.2	Configuration of the Bit Timing .....	19-11
19.2.3	Mask Registers .....	19-15
19.3	The Message Object .....	19-18
19.4	Controlling the CAN Module .....	19-30
19.5	Configuration Examples for Message Objects .....	19-34
19.6	The Second CAN Module CAN2 .....	19-36
19.7	The CAN Application Interface .....	19-37
<b>20</b>	<b>The Serial Data Link Module (SDLM)</b> .....	20-1
20.1	Frame Format Basics .....	20-2
20.2	The Structure of the SDLM .....	20-6
20.3	Message Operating Mode .....	20-8
20.3.1	Receive Operation .....	20-8
20.3.2	Transmit Operation .....	20-10
20.3.3	In-Frame Response (IFR) Operation .....	20-11
20.3.4	Block Mode .....	20-12
20.4	Flowcharts .....	20-14
20.5	Interrupt Handling .....	20-21
20.6	Port Control .....	20-23
20.7	SDLM Register Description .....	20-25
20.8	Interrupt Node Control .....	20-48
<b>21</b>	<b>The IIC Bus Module</b> .....	21-1
21.1	IIC Bus Conditions .....	21-2
21.2	The Physical IIC Bus Interface .....	21-4
21.3	Operating the IIC Bus .....	21-6
21.3.1	Operation in Master Mode .....	21-6
21.3.2	Operation in Multimaster Mode .....	21-7
21.3.3	Operation in Slave Mode .....	21-7
21.4	IIC Interrupt Control .....	21-12
21.5	Programming Example .....	21-14
<b>22</b>	<b>System Reset</b> .....	22-1
22.1	Reset Sources .....	22-2
22.2	Status After Reset .....	22-5
22.3	Application-Specific Initialization Routine .....	22-9
22.4	System Startup Configuration .....	22-12
22.4.1	System Startup Configuration upon an External Reset .....	22-13
22.4.2	System Startup Configuration upon a Single-Chip Mode Reset ...	22-20
22.5	System Configuration via Software .....	22-22

<b>Table of Contents</b>		<b>Page</b>
<b>23</b>	<b>Power Management</b> .....	23-1
23.1	Idle Mode .....	23-3
23.2	Sleep Mode .....	23-5
23.3	Power Down Mode .....	23-6
23.3.1	Status of Output Pins During Power Reduction Modes .....	23-8
23.4	Slow Down Operation .....	23-10
23.5	Flexible Peripheral Management .....	23-14
23.6	Programmable Frequency Output Signal .....	23-17
23.7	Security Mechanism .....	23-22
<b>24</b>	<b>System Programming</b> .....	24-1
24.1	Stack Operations .....	24-4
24.2	Register Banking .....	24-9
24.3	Procedure Call Entry and Exit .....	24-9
24.4	Table Searching .....	24-12
24.5	Floating Point Support .....	24-12
24.6	Peripheral Control and Interface .....	24-13
24.7	Trap/Interrupt Entry and Exit .....	24-13
24.8	Unseparable Instruction Sequences .....	24-14
24.9	Overriding the DPP Addressing Mechanism .....	24-14
24.10	Handling the Internal Code Memory .....	24-16
24.11	Pits, Traps and Mines .....	24-18
<b>25</b>	<b>The Register Set</b> .....	25-1
25.1	Register Description Format .....	25-1
25.2	CPU General Purpose Registers (GPRs) .....	25-2
25.3	Special Function Registers Ordered by Name .....	25-4
25.4	Special Function Registers Ordered by Address .....	25-14
25.5	Special Notes .....	25-24
<b>26</b>	<b>Instruction Set Summary</b> .....	26-1
<b>27</b>	<b>Device Specification</b> .....	27-1
<b>28</b>	<b>Keyword Index</b> .....	28-1

## **1 Introduction**

The rapidly growing area of embedded control applications is representing one of the most time-critical operating environments for today's microcontrollers. Complex control algorithms have to be processed based on a large number of digital as well as analog input signals, and the appropriate output signals must be generated within a defined maximum response time. Embedded control applications also are often sensitive to board space, power consumption, and overall system cost.

Embedded control applications therefore require microcontrollers, which:

- offer a high level of system integration
- eliminate the need for additional peripheral devices and the associated software overhead
- provide system security and fail-safe mechanisms
- provide effective means to control (and reduce) the device's power consumption.

With the increasing complexity of embedded control applications, a significant increase in CPU performance and peripheral functionality over conventional 8-bit controllers is required from microcontrollers for high-end embedded control systems. In order to achieve this high performance goal Infineon has decided to develop its family of 16-bit CMOS microcontrollers without the constraints of backward compatibility.

Of course the architecture of the 16-bit microcontroller family pursues successful hardware and software concepts, which have been established in Infineon's popular 8-bit controller families.

## About this Manual

This manual describes the functionality of a number of 16-bit microcontrollers of the Infineon C166 Family.

As these microcontrollers provide a great extent of identical functionality it makes sense to describe a superset of the provided features. For this reason some sections of this manual do not refer to all the C161 derivatives that are offered (e.g. devices without on-chip program memory). These sections contain respective notes wherever possible.

The descriptions in this manual refer to the following C161 derivatives:

- **C161CS-32RF** 256 KByte ROM, 2 CAN modules
- **C161CS-LF** No program memory, 2 CAN modules
- **C161JC-32RF** 256 KByte ROM, 1 CAN module, SDLM module (J1850)
- **C161JC-LF** No program memory, 1 CAN module, SDLM module (J1850)
- **C161JI-32RF** 256 KByte ROM, SDLM module (J1850)
- **C161JI-LF** No program memory, SDLM module (J1850)

This manual is valid for the versions with on-chip ROM of the mentioned derivatives as well as for the ROMless versions. Of course it refers to all devices of the different available temperature ranges and packages.

For simplicity all these various versions are referred to by the term **C161CS/JC/JI** throughout this manual. The complete pro-electron conforming designations are listed in the respective data sheets.

## 1.1 The Members of the 16-bit Microcontroller Family

The microcontrollers of the Infineon 16-bit family have been designed to meet the high performance requirements of real-time embedded control applications. The architecture of this family has been optimized for high instruction throughput and minimum response time to external stimuli (interrupts). Intelligent peripheral subsystems have been integrated to reduce the need for CPU intervention to a minimum extent. This also minimizes the need for communication via the external bus interface. The high flexibility of this architecture allows to serve the diverse and varying needs of different application areas such as automotive, industrial control, or data communications.

The core of the 16-bit family has been developed with a modular family concept in mind. All family members execute an efficient control-optimized instruction set (additional instructions for members of the second generation). This allows an easy and quick implementation of new family members with different internal memory sizes and technologies, different sets of on-chip peripherals and/or different numbers of IO pins.

The XBUS concept opens a straight forward path for the integration of application specific peripheral modules in addition to the standard on-chip peripherals in order to build application specific derivatives.

As programs for embedded control applications become larger, high level languages are favored by programmers, because high level language programs are easier to write, to debug and to maintain.

The 80C166-type microcontrollers were the **first generation** of the 16-bit controller family. These devices have established the C166 architecture.

The C165-type and C167-type devices are members of the **second generation** of this family. This second generation is even more powerful due to additional instructions for HLL support, an increased address space, increased internal RAM and highly efficient management of various resources on the external bus.

**Enhanced derivatives** of this second generation provide additional features like additional internal high-speed RAM, an integrated CAN-Module, an on-chip PLL, etc.

Utilizing integration to design efficient systems may require the integration of application specific peripherals to boost system performance, while minimizing the part count. These efforts are supported by the so-called XBUS, defined for the Infineon 16-bit microcontrollers (second generation). This XBUS is an internal representation of the external bus interface that opens and simplifies the integration of peripherals by standardizing the required interface. One representative taking advantage of this technology is the integrated CAN module.

The C165-type devices are reduced versions of the C167 which provide a smaller package and reduced power consumption at the expense of the A/D converter, the CAPCOM units and the PWM module.

The C164-type devices, the C167CS derivatives, and some of the C161-type devices are further enhanced by a flexible power management and form the **third generation** of the 16-bit controller family. This power management mechanism provides effective means to control the power that is consumed in a certain state of the controller and thus allows the minimization of the overall power consumption with respect to a given application.

A variety of different versions is provided which offer various kinds of on-chip program memory:

- Mask-programmable ROM
- Flash memory
- OTP memory
- ROMless with no non-volatile memory at all.

Also there are devices with specific functional units.

The devices may be offered in different packages, temperature ranges and speed classes.

More standard and application-specific derivatives are planned and in development.

*Note: Not all derivatives will be offered in any temperature range, speed class, package or program memory variation.*

Information about specific versions and derivatives will be made available with the devices themselves. Contact your Infineon representative for up-to-date material.

*Note: As the architecture and the basic features (i.e. CPU core and built in peripherals) are identical for most of the currently offered versions of the C161CS/JC/JI, the descriptions within this manual that refer to the "C161CS/JC/JI" also apply to the other variations, unless otherwise noted.*

## **1.2 Summary of Basic Features**

The C161CS/JC/JI is an improved representative of the Infineon family of full featured 16-bit single-chip CMOS microcontrollers. It combines high CPU performance (up to 12.5/16.5 million instructions per second) with high peripheral functionality and means for power reduction.

Several key features contribute to the high performance of the C161CS/JC/JI (the indicated timings refer to a CPU clock of 25/33 MHz).

### **High Performance 16-bit CPU with Four-Stage Pipeline**

- 80/60 ns minimum instruction cycle time, with most instructions executed in 1 cycle
- 400/300 ns multiplication (16-bit × 16-bit), 800/600 ns division (32-bit / 16-bit)
- Multiple high bandwidth internal data buses
- Register based design with multiple variable register banks
- Single cycle context switching support
- 16 MBytes linear address space for code and data (Von Neumann architecture)
- System stack cache support with automatic stack overflow/underflow detection

### **Control Oriented Instruction Set with High Efficiency**

- Bit, byte, and word data types
- Flexible and efficient addressing modes for high code density
- Enhanced boolean bit manipulation with direct addressability of 6 Kbits for peripheral control and user defined flags
- Hardware traps to identify exception conditions during runtime
- HLL support for semaphore operations and efficient data access

### **Integrated On-Chip Memory**

- 2 KByte internal RAM for variables, register banks, system stack and code
- 8 KByte on-chip high-speed XRAM for variables, user stack and code
- 256 KByte on-chip Program ROM (not for ROMless devices)

### **External Bus Interface**

- Multiplexed or demultiplexed bus configurations
- Segmentation capability and chip select signal generation
- 8-bit or 16-bit data bus
- Bus cycle characteristics selectable for five programmable address areas

### 16-Priority-Level Interrupt System

- 59 interrupt nodes with separate interrupt vectors
- 240/180 ns typical interrupt latency (400/300 ns maximum) in case of internal program execution
- Fast external interrupts

### 8-Channel Peripheral Event Controller (PEC)

- Interrupt driven single cycle data transfer
- Transfer count option (std. CPU interrupt after programmable number of PEC transfers)
- Eliminates overhead of saving and restoring system state for interrupt requests

### Intelligent On-Chip Peripheral Subsystems

- 12-channel 10-bit A/D Converter with programmable conversion time (7.76  $\mu$ s minimum), auto scan modes, channel injection mode
- Two 16-channel Capture/Compare Units with 2 independent time bases each, very flexible PWM unit/event recording unit with different operating modes, includes four 16-bit timers/counters, maximum resolution  $f_{CPU}/8$
- Two Multifunctional General Purpose Timer Units  
GPT1: Three 16-bit timers/counters, maximum resolution  $f_{CPU}/8$   
GPT2: Two 16-bit timers/counters, maximum resolution  $f_{CPU}/4$
- Two Asynchronous/Synchronous Serial Channels (USART) with baud rate generator, parity, framing, and overrun error detection
- High Speed Synchronous Serial Channel programmable data length and shift direction
- IIC Bus module with 10-bit addressing and 400 kbit/s
- One or two on-chip CAN Bus Modules, Rev. 2.0B active
- Serial Data Link Module (SDLM), compliant with J1850, supporting Class 2
- Real Time Clock
- Watchdog Timer with programmable time intervals
- Bootstrap Loader for flexible system initialization

### 93 IO Lines with Individual Bit Addressability

- Tri-stated in input mode
- Selectable input thresholds (not on all pins)
- Push/pull or open drain output mode
- Programmable port driver control

### Different Temperature Ranges

- 0 to +70 °C, -40 to +85 °C, -40 to +125 °C



**Infineon CMOS Process**

- Low power CMOS technology including power saving Idle and Power Down modes.

**128-Pin Plastic Thin Quad Flat Pack (TQFP) Package**

- P-TQFP, 20 × 20 mm body, 0.5 mm (19.7 mil) lead spacing, surface mount technology

**Complete Development Support**

For the development tool support of its microcontrollers, Infineon follows a clear third party concept. Currently around 120 tool suppliers world-wide, ranging from local niche manufacturers to multinational companies with broad product portfolios, offer powerful development tools for the Infineon C500 and C166 microcontroller families, guaranteeing a remarkable variety of price-performance classes as well as early availability of high quality key tools such as compilers, assemblers, simulators, debuggers or in-circuit emulators.

Infineon incorporates its strategic tool partners very early into the product development process, making sure embedded system developers get reliable, well-tuned tool solutions, which help them unleash the power of Infineon microcontrollers in the most effective way and with the shortest possible learning curve.

The tool environment for the Infineon 16-bit microcontrollers includes the following tools:

- Compilers (C, MODULA2, FORTH)
- Macro-assemblers, linkers, locators, library managers, format-converters
- Architectural simulators
- HLL debuggers
- Real-time operating systems
- VHDL chip models
- In-circuit emulators (based on bondout or standard chips)
- Plug-in emulators
- Emulation and clip-over adapters, production sockets
- Logic analyzer disassemblers
- Starter kits
- Evaluation boards with monitor programs
- Industrial boards (also for CAN, FUZZY, PROFIBUS, FORTH applications)
- Network driver software (CAN, PROFIBUS)

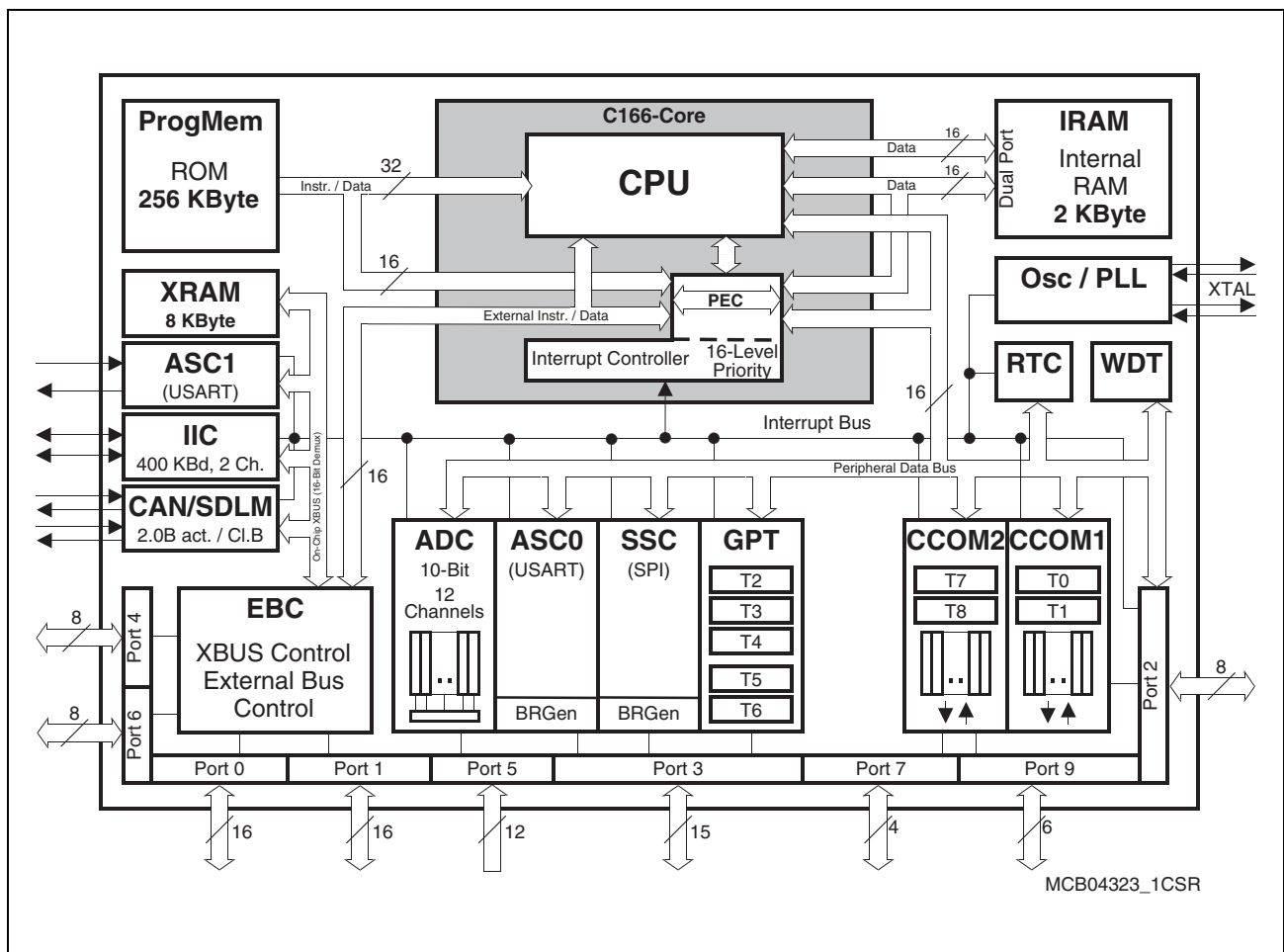
### **1.3 Abbreviations**

The following acronyms and terms are used within this document:

ADC	Analog Digital Converter
ALE	Address Latch Enable
ALU	Arithmetic and Logic Unit
ASC	Asynchronous/synchronous Serial Controller
CAN	Controller Area Network (License Bosch)
CAPCOM	CAPture and COMpare unit
CISC	Complex Instruction Set Computing
CMOS	Complementary Metal Oxide Silicon
CPU	Central Processing Unit
EBC	External Bus Controller
ESFR	Extended Special Function Register
Flash	Non-volatile memory that may be electrically erased
GPR	General Purpose Register
GPT	General Purpose Timer unit
HLL	High Level Language
IIC	Inter Integrated Circuit (Bus)
IO	Input/Output
OTP	One Time Programmable memory
PEC	Peripheral Event Controller
PLA	Programmable Logic Array
PLL	Phase Locked Loop
PWM	Pulse Width Modulation
RAM	Random Access Memory
RISC	Reduced Instruction Set Computing
ROM	Read Only Memory
RTC	Real Time Clock
SDD	Slow Down Divider
SFR	Special Function Register
SSC	Synchronous Serial Controller
XBUS	Internal representation of the External Bus
XRAM	On-chip extension RAM

## 2 Architectural Overview

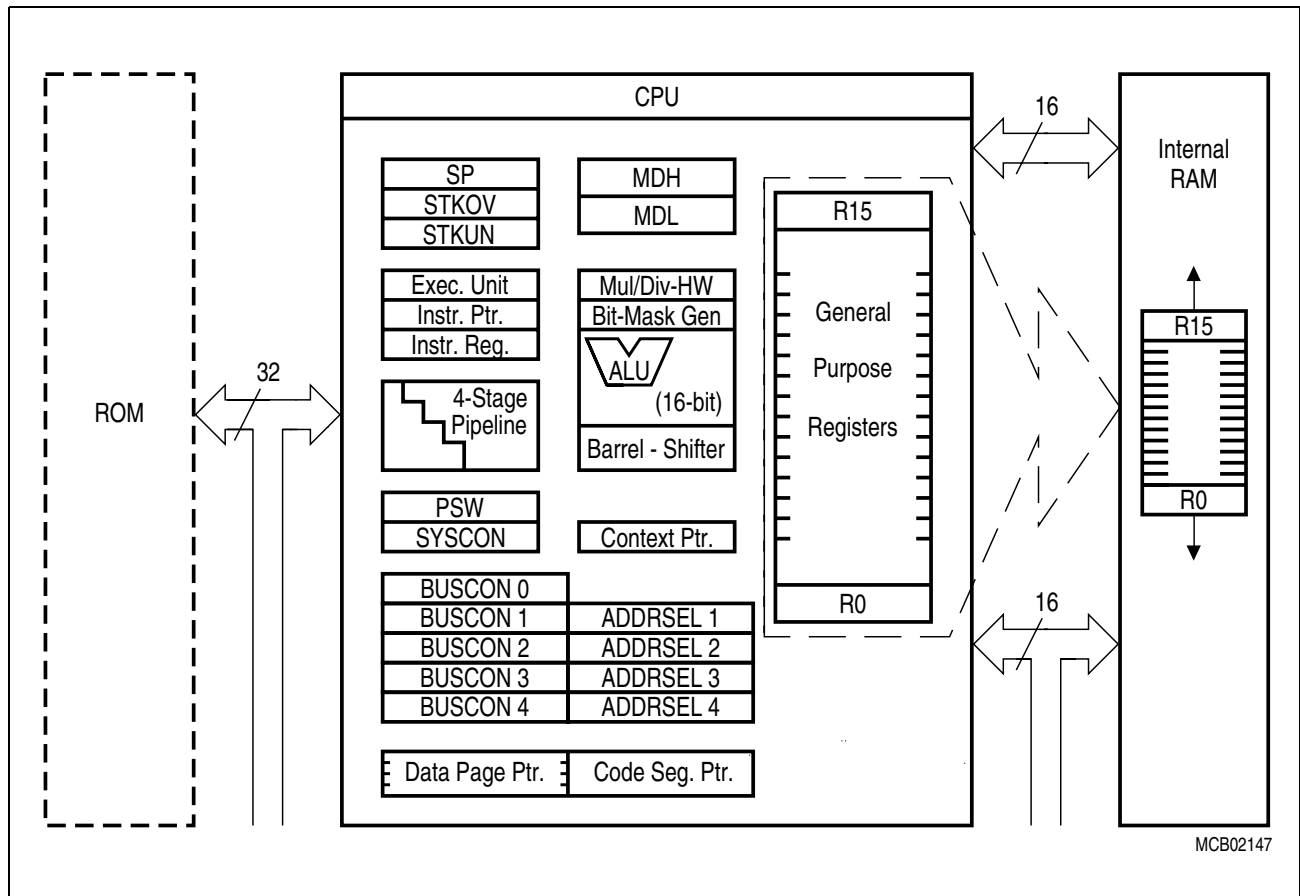
The architecture of the C161CS/JC/JI combines the advantages of both RISC and CISC processors in a very well-balanced way. The sum of the features which are combined result in a high performance microcontroller, which is the right choice not only for today's applications, but also for future engineering challenges. The C161CS/JC/JI not only integrates a powerful CPU core and a set of peripheral units into one chip, but also connects the units in a very efficient way. One of the four buses used concurrently on the C161CS/JC/JI is the XBUS, an internal representation of the external bus interface. This bus provides a standardized method of integrating application-specific peripherals to produce derivatives of the standard C161CS/JC/JI.



**Figure 2-1 C161CS/JC/JI Functional Block Diagram**

## 2.1 Basic CPU Concepts and Optimizations

The main core of the CPU consists of a 4-stage instruction pipeline, a 16-bit arithmetic and logic unit (ALU) and dedicated SFRs. Additional hardware is provided for a separate multiply and divide unit, a bit-mask generator and a barrel shifter.



**Figure 2-2 CPU Block Diagram**

To meet the demand for greater performance and flexibility, a number of areas has been optimized in the processor core. Functional blocks in the CPU core are controlled by signals from the instruction decode logic. These are summarized below, and described in detail in the following sections:

1. High Instruction Bandwidth/Fast Execution
2. High Function 8-bit and 16-bit Arithmetic and Logic Unit
3. Extended Bit Processing and Peripheral Control
4. High Performance Branch-, Call-, and Loop Processing
5. Consistent and Optimized Instruction Formats
6. Programmable Multiple Priority Interrupt Structure

### 2.1.1 High Instruction Bandwidth/Fast Execution

Based on the hardware provisions, most of the C161CS/JC/JI's instructions can be executed in just one machine cycle, which requires 2 CPU clock cycles ( $2 \times 1 / f_{\text{CPU}} = 4 \text{ TCL}$ ). For example, shift and rotate instructions are always processed within one machine cycle, independent of the number of bits to be shifted.

Branch-, multiply- and divide instructions normally take more than one machine cycle. These instructions, however, have also been optimized. For example, branch instructions only require an additional machine cycle, when a branch is taken, and most branches taken in loops require no additional machine cycles at all, due to the so-called 'Jump Cache'.

A 32-bit/16-bit division takes 20 CPU clock cycles, a 16-bit  $\times$  16-bit multiplication takes 10 CPU clock cycles.

The instruction cycle time has been dramatically reduced through the use of instruction pipelining. This technique allows the core CPU to process portions of multiple sequential instruction stages in parallel. The following four stage pipeline provides the optimum balancing for the CPU core:

**FETCH:** In this stage, an instruction is fetched from the internal ROM or RAM or from the external memory, based on the current IP value.

**DECODE:** In this stage, the previously fetched instruction is decoded and the required operands are fetched.

**EXECUTE:** In this stage, the specified operation is performed on the previously fetched operands.

**WRITE BACK:** In this stage, the result is written to the specified location.

If this technique were not used, each instruction would require four machine cycles. This increased performance allows a greater number of tasks and interrupts to be processed.

#### Instruction Decoder

Instruction decoding is primarily generated from PLA outputs based on the selected opcode. No microcode is used and each pipeline stage receives control signals staged in control registers from the decode stage PLAs. Pipeline holds are primarily caused by wait states for external memory accesses and cause the holding of signals in the control registers. Multiple-cycle instructions are performed through instruction injection and simple internal state machines which modify required control signals.

### **High Function 8-bit and 16-bit Arithmetic and Logic Unit**

All standard arithmetic and logical operations are performed in a 16-bit ALU. In addition, for byte operations, signals are provided from bits six and seven of the ALU result to correctly set the condition flags. Multiple precision arithmetic is provided through a 'CARRY-IN' signal to the ALU from previously calculated portions of the desired operation.

Most internal execution blocks have been optimized to perform operations on either 8-bit or 16-bit quantities. Once the pipeline has been filled, one instruction is completed per machine cycle, except for multiply and divide. An advanced Booth algorithm has been incorporated to allow four bits to be multiplied and two bits to be divided per machine cycle. Thus, these operations use two coupled 16-bit registers, MDL and MDH, and require four and nine machine cycles, respectively, to perform a 16-bit by 16-bit (or 32-bit by 16-bit) calculation plus one machine cycle to setup and adjust the operands and the result. Even these longer multiply and divide instructions can be interrupted during their execution to allow for very fast interrupt response. Instructions have also been provided to allow byte packing in memory while providing sign extension of bytes for word wide arithmetic operations. The internal bus structure also allows transfers of bytes or words to or from peripherals based on the peripheral requirements.

A set of consistent flags is automatically updated in the PSW after each arithmetic, logical, shift, or movement operation. These flags allow branching on specific conditions. Support for both signed and unsigned arithmetic is provided through user-specifiable branch tests. These flags are also preserved automatically by the CPU upon entry into an interrupt or trap routine.

All targets for branch calculations are also computed in the central ALU.

A 16-bit barrel shifter provides multiple bit shifts in a single cycle. Rotates and arithmetic shifts are also supported.

### **Extended Bit Processing and Peripheral Control**

A large number of instructions has been dedicated to bit processing. These instructions provide efficient control and testing of peripherals while enhancing data manipulation. Unlike other microcontrollers, these instructions provide direct access to two operands in the bit-addressable space without requiring to move them into temporary flags.

The same logical instructions available for words and bytes are also supported for bits. This allows the user to compare and modify a control bit for a peripheral in one instruction. Multiple bit shift instructions have been included to avoid long instruction streams of single bit shift operations. These are also performed in a single machine cycle.

In addition, bit field instructions have been provided, which allow the modification of multiple bits from one operand in a single instruction.

### **High Performance Branch-, Call-, and Loop Processing**

Due to the high percentage of branching in controller applications, branch instructions have been optimized to require one extra machine cycle only when a branch is taken. This is implemented by precalculating the target address while decoding the instruction. To decrease loop execution overhead, three enhancements have been provided:

- The first solution provides single cycle branch execution after the first iteration of a loop. Thus, only one machine cycle is lost during the execution of the entire loop. In loops which fall through upon completion, no machine cycles are lost when exiting the loop. No special instructions are required to perform loops, and loops are automatically detected during execution of branch instructions.
- The second loop enhancement allows the detection of the end of a table and avoids the use of two compare instructions embedded in loops. One simply places the lowest negative number at the end of the specific table, and specifies branching if neither this value nor the compared value have been found. Otherwise the loop is terminated if either condition has been met. The terminating condition can then be tested.
- The third loop enhancement provides a more flexible solution than the Decrement and Skip on Zero instruction which is found in other microcontrollers. Through the use of Compare and Increment or Decrement instructions, the user can make comparisons to any value. This allows loop counters to cover any range. This is particularly advantageous in table searching.

Saving of system state is automatically performed on the internal system stack avoiding the use of instructions to preserve state upon entry and exit of interrupt or trap routines. Call instructions push the value of the IP on the system stack, and require the same execution time as branch instructions.

Instructions have also been provided to support indirect branch and call instructions. This supports implementation of multiple CASE statement branching in assembler macros and high level languages.

### **Consistent and Optimized Instruction Formats**

To obtain optimum performance in a pipelined design, an instruction set has been designed which incorporates concepts from Reduced Instruction Set Computing (RISC). These concepts primarily allow fast decoding of the instructions and operands while reducing pipeline holds. These concepts, however, do not preclude the use of complex instructions, which are required by microcontroller users. The following goals were used to design the instruction set:

1. Provide powerful instructions to perform operations which currently require sequences of instructions and are frequently used. Avoid transfer into and out of temporary registers such as accumulators and carry bits. Perform tasks in parallel such as saving state upon entry into interrupt routines or subroutines.
2. Avoid complex encoding schemes by placing operands in consistent fields for each instruction. Also avoid complex addressing modes which are not frequently used. This decreases the instruction decode time while also simplifying the development of compilers and assemblers.
3. Provide most frequently used instructions with one-word instruction formats. All other instructions are placed into two-word formats. This allows all instructions to be placed on word boundaries, which alleviates the need for complex alignment hardware. It also has the benefit of increasing the range for relative branching instructions.

The high performance offered by the hardware implementation of the CPU can efficiently be utilized by a programmer via the highly functional C161CS/JC/JI instruction set which includes the following instruction classes:

- Arithmetic Instructions
- Logical Instructions
- Boolean Bit Manipulation Instructions
- Compare and Loop Control Instructions
- Shift and Rotate Instructions
- Prioritize Instruction
- Data Movement Instructions
- System Stack Instructions
- Jump and Call Instructions
- Return Instructions
- System Control Instructions
- Miscellaneous Instructions

Possible operand types are bits, bytes and words. Specific instruction support the conversion (extension) of bytes to words. A variety of direct, indirect or immediate addressing modes are provided to specify the required operands.



### **2.1.2 Programmable Multiple Priority Interrupt System**

The following enhancements have been included to allow processing of a large number of interrupt sources:

1. **Peripheral Event Controller (PEC):** This processor is used to off-load many interrupt requests from the CPU. It avoids the overhead of entering and exiting interrupt or trap routines by performing single-cycle interrupt-driven byte or word data transfers between any two locations in segment 0 with an optional increment of either the PEC source or the destination pointer. Just one cycle is 'stolen' from the current CPU activity to perform a PEC service.
2. **Multiple Priority Interrupt Controller:** This controller allows all interrupts to be placed at any specified priority. Interrupts may also be grouped, which provides the user with the ability to prevent similar priority tasks from interrupting each other. For each of the possible interrupt sources there is a separate control register, which contains an interrupt request flag, an interrupt enable flag and an interrupt priority bitfield. Once having been accepted by the CPU, an interrupt service can only be interrupted by a higher prioritized service request. For standard interrupt processing, each of the possible interrupt sources has a dedicated vector location.
3. **Multiple Register Banks:** This feature allows the user to specify up to sixteen general purpose registers located anywhere in the internal RAM. A single one-machine-cycle instruction allows to switch register banks from one task to another.
4. **Interruptable Multiple Cycle Instructions:** Reduced interrupt latency is provided by allowing multiple-cycle instructions (multiply, divide) to be interruptable.

With an interrupt response time within a range from just 5 to 10 CPU clock cycles (in case of internal program execution), the C161CS/JC/JI is capable of reacting very fast on non-deterministic events.

Its fast external interrupt inputs are sampled every CPU clock cycle and allow to recognize even very short external signals.

The C161CS/JC/JI also provides an excellent mechanism to identify and to process exceptions or error conditions that arise during run-time, so called 'Hardware Traps'. Hardware traps cause an immediate non-maskable system reaction which is similar to a standard interrupt service (branching to a dedicated vector table location). The occurrence of a hardware trap is additionally signified by an individual bit in the trap flag register (TFR). Except for another higher prioritized trap service being in progress, a hardware trap will interrupt any current program execution. In turn, hardware trap services can normally not be interrupted by standard or PEC interrupts.

Software interrupts are supported by means of the 'TRAP' instruction in combination with an individual trap (interrupt) number.

## **2.2 The On-Chip System Resources**

The C161CS/JC/JI controllers provide a number of powerful system resources designed around the CPU. The combination of CPU and these resources results in the high performance of the members of this controller family.

### **Peripheral Event Controller (PEC) and Interrupt Control**

The Peripheral Event Controller allows to respond to an interrupt request with a single data transfer (word or byte) which only consumes one instruction cycle and does not require to save and restore the machine status. Each interrupt source is prioritized every machine cycle in the interrupt control block. If PEC service is selected, a PEC transfer is started. If CPU interrupt service is requested, the current CPU priority level stored in the PSW register is tested to determine whether a higher priority interrupt is currently being serviced. When an interrupt is acknowledged, the current state of the machine is saved on the internal system stack and the CPU branches to the system specific vector for the peripheral.

The PEC contains a set of SFRs which store the count value and control bits for eight data transfer channels. In addition, the PEC uses a dedicated area of RAM which contains the source and destination addresses. The PEC is controlled similar to any other peripheral through SFRs containing the desired configuration of each channel.

An individual PEC transfer counter is implicitly decremented for each PEC service except forming in the continuous transfer mode. When this counter reaches zero, a standard interrupt is performed to the vector location related to the corresponding source. PEC services are very well suited, for example, to move register contents to/from a memory table. The C161CS/JC/JI has 8 PEC channels each of which offers such fast interrupt-driven data transfer capabilities.

### **Memory Areas**

The memory space of the C161CS/JC/JI is configured in a Von Neumann architecture which means that code memory, data memory, registers and IO ports are organized within the same linear address space which covers up to 16 MBytes. The entire memory space can be accessed bitwise or wordwise. Particular portions of the on-chip memory have additionally been made directly bit addressable.

**A 2 KByte 16-bit wide internal RAM (IRAM)** provides fast access to General Purpose Registers (GPRs), user data (variables) and system stack. The internal RAM may also be used for code. A unique decoding scheme provides flexible user register banks in the internal memory while optimizing the remaining RAM for user data.

## Architectural Overview

The CPU has an actual register context consisting of up to 16 wordwide and/or bytewise GPRs at its disposal, which are physically located within the on-chip RAM area. A Context Pointer (CP) register determines the base address of the active register bank to be accessed by the CPU at a time. The number of register banks is only restricted by the available internal RAM space. For easy parameter passing, a register bank may overlap others.

A system stack of up to 1024 words is provided as a storage for temporary data. The system stack is also located within the on-chip RAM area, and it is accessed by the CPU via the stack pointer (SP) register. Two separate SFRs, STKOV and STKUN, are implicitly compared against the stack pointer value upon each stack access for the detection of a stack overflow or underflow.

Hardware detection of the selected memory space is placed at the internal memory decoders and allows the user to specify any address directly or indirectly and obtain the desired data without using temporary registers or special instructions.

**An 8 KByte 16-bit wide on-chip XRAM** provides fast access to user data (variables), user stacks and code. The on-chip XRAM is realized as an X-Peripheral and appears to the software as an external RAM. Therefore it cannot store register banks and is not bitaddressable. The XRAM allows 16-bit accesses with maximum speed.

**For Special Function Registers** 1024 Bytes of the address space are reserved. The standard Special Function Register area (SFR) uses 512 Bytes, while the Extended Special Function Register area (ESFR) uses the other 512 Bytes. (E)SFRs are wordwide registers which are used for controlling and monitoring functions of the different on-chip units. Unused (E)SFR addresses are reserved for future members of the C166 Family with enhanced functionality.

**An optional on-chip ROM memory (256 KByte)** provides for both code and constant data storage. This memory area is connected to the CPU via a 32-bit-wide bus. Thus, an entire double-word instruction can be fetched in just one machine cycle. Program execution from on-chip program memory is the fastest of all possible alternatives.

The type of the on-chip program memory depends on the chosen derivative.

## **External Bus Interface**

In order to meet the needs of designs where more memory is required than is provided on chip, up to 16 MBytes of external RAM and/or ROM can be connected to the microcontroller via its external bus interface. The integrated External Bus Controller (EBC) allows to access external memory and/or peripheral resources in a very flexible way. For up to five address areas the bus mode (multiplexed/demultiplexed), the data bus width (8-bit/16-bit) and even the length of a bus cycle (waitstates, signal delays) can be selected independently. This allows to access a variety of memory and peripheral components directly and with maximum efficiency. If the device does not run in Single Chip Mode, where no external memory is required, the EBC can control external accesses in one of the following external access modes:

- 16-/18-/20-/24-bit Addresses, 16-bit Data, Demultiplexed
- 16-/18-/20-/24-bit Addresses, 8-bit Data, Demultiplexed
- 16-/18-/20-/24-bit Addresses, 16-bit Data, Multiplexed
- 16-/18-/20-/24-bit Addresses, 8-bit Data, Multiplexed

The demultiplexed bus modes use PORT1 for addresses and PORT0 for data input/output. The multiplexed bus modes use PORT0 for both addresses and data input/output. Port 4 is used for the upper address lines (A16 ...) if selected.

Important timing characteristics of the external bus interface (waitstates, ALE length and Read/Write Delay) have been made programmable to allow the user the adaption of a wide range of different types of memories and/or peripherals. Access to very slow memories or peripherals is supported via a particular 'Ready' function.

For applications which require less than 64 KBytes of address space, a non-segmented memory model can be selected, where all locations can be addressed by 16-bits, and thus Port 4 is not needed as an output for the upper address bits (Axx ... A16), as is the case when using the segmented memory model.

**The on-chip XBUS** is an internal representation of the external bus and allows to access integrated application-specific peripherals/modules in the same way as external components. It provides a defined interface for these customized peripherals.

The on-chip XRAM and the on-chip CAN-Modules are examples for these X-Peripherals.

## **2.3 The On-Chip Peripheral Blocks**

The C166 Family clearly separates peripherals from the core. This structure permits the maximum number of operations to be performed in parallel and allows peripherals to be added or deleted from family members without modifications to the core. Each functional block processes data independently and communicates information over common buses. Peripherals are controlled by data written to the respective Special Function Registers (SFRs). These SFRs are located either within the standard SFR area (00'FE00<sub>H</sub> ... 00'FFFF<sub>H</sub>) or within the extended ESFR area (00'F000<sub>H</sub> ... 00'F1FF<sub>H</sub>).

These built in peripherals either allow the CPU to interface with the external world, or provide functions on-chip that otherwise were to be added externally in the respective system.

The C161CS/JC/JI generic peripherals are:

- Two General Purpose Timer Blocks (GPT1 and GPT2)
- Two Serial Interfaces (ASC0 and SSC)
- A Watchdog Timer
- Two 16-channel Capture/Compare units (CAPCOM1 and CAPCOM2)
- A 10-bit Analog/Digital Converter
- A Real Time Clock
- Nine IO ports with a total of 93 IO lines

Each peripheral also contains a set of Special Function Registers (SFRs), which control the functionality of the peripheral and temporarily store intermediate data results. Each peripheral has an associated set of status flags. Individually selected clock signals are generated for each peripheral from binary multiples of the CPU clock.

### **Peripheral Interfaces**

The on-chip peripherals generally have two different types of interfaces, an interface to the CPU and an interface to external hardware. Communication between CPU and peripherals is performed through Special Function Registers (SFRs) and interrupts. The SFRs serve as control/status and data registers for the peripherals. Interrupt requests are generated by the peripherals based on specific events which occur during their operation (e.g. operation complete, error, etc.).

For interfacing with external hardware, specific pins of the parallel ports are used, when an input or output function has been selected for a peripheral. During this time, the port pins are controlled by the peripheral (when used as outputs) or by the external hardware which controls the peripheral (when used as inputs). This is called the 'alternate (input or output) function' of a port pin, in contrast to its function as a general purpose IO pin.

## Peripheral Timing

Internal operation of CPU and peripherals is based on the CPU clock ( $f_{\text{CPU}}$ ). The on-chip oscillator derives the CPU clock from the crystal or from the external clock signal. The clock signal which is gated to the peripherals is independent from the clock signal which feeds the CPU. During Idle mode the CPU's clock is stopped while the peripherals continue their operation. Peripheral SFRs may be accessed by the CPU once per state. When an SFR is written to by software in the same state where it is also to be modified by the peripheral, the software write operation has priority. Further details on peripheral timing are included in the specific sections about each peripheral.

## Programming Hints

### Access to SFRs

All SFRs reside in data page 3 of the memory space. The following addressing mechanisms allow to access the SFRs:

- Indirect or direct addressing with **16-bit (mem) addresses** must guarantee that the used data page pointer (DPP0 ... DPP3) selects data page 3.
- Accesses via the Peripheral Event Controller (**PEC**) use the SRCPx and DSTPx pointers instead of the data page pointers.
- **Short 8-bit (reg) addresses** to the standard SFR area do not use the data page pointers but directly access the registers within this 512 Byte area.
- **Short 8-bit (reg) addresses** to the extended **ESFR** area require switching to the 512 Byte extended SFR area. This is done via the EXTension instructions EXTR, EXTP(R), EXTS(R).

**Byte write operations** to word wide SFRs via indirect or direct 16-bit (mem) addressing or byte transfers via the PEC force zeros in the non-addressed byte. Byte write operations via short 8-bit (reg) addressing can only access the low byte of an SFR and force zeros in the high byte. It is therefore recommended, to use the bit field instructions (BFLDL and BFLDH) to write to any number of bits in either byte of an SFR without disturbing the non-addressed byte and the unselected bits.

### Reserved Bits

Some of the bits which are contained in the C161CS/JC/JI's SFRs are marked as 'Reserved'. User software should never write '1's to reserved bits. These bits are currently not implemented and may be used in future products to invoke new functions. In this case, the active state for these functions will be '1', and the inactive state will be '0'. Therefore writing only '0's to reserved locations provides portability of the current software to future devices. After read accesses reserved bits should be ignored or masked out.



## Serial Channels

Serial communication with other microcontrollers, processors, terminals or external peripheral components is provided by three serial interfaces with different functionality, two Asynchronous/Synchronous Serial Channels (**ASC0**, **ASC1**) and a High-Speed Synchronous Serial Channel (**SSC**).

**The ASC0** is upward compatible with the serial ports of the Infineon 8-bit microcontroller families. It supports full-duplex asynchronous communication at up to 780/1030 kBaud and half-duplex synchronous communication at up to 3.1/4.1 MBaud @ 25/33 MHz CPU clock.

A dedicated baud rate generator allows to set up all standard baud rates without oscillator tuning. For transmission, reception and error handling 4 separate interrupt vectors are provided. In asynchronous mode, 8- or 9-bit data frames are transmitted or received, preceded by a start bit and terminated by one or two stop bits. For multiprocessor communication, a mechanism to distinguish address from data bytes has been included (8-bit data plus wake up bit mode).

In synchronous mode, the ASC0 transmits or receives bytes (8 bits) synchronously to a shift clock which is generated by the ASC0. The ASC0 always shifts the LSB first. A loop back option is available for testing purposes.

A number of optional hardware error detection capabilities has been included to increase the reliability of data transfers. A parity bit can automatically be generated on transmission or be checked on reception. Framing error detection allows to recognize data frames with missing stop bits. An overrun error will be generated, if the last character received has not been read out of the receive buffer register at the time the reception of a new character is complete.

**The ASC1** is another USART which is functionally identical with the ASC0. The ASC1 is accessed via the XBUS (no bit handling) and supports 3 interrupt vectors. The port line handling is slightly different from that of the ASC0.

**The SSC** supports full-duplex synchronous communication at up to 6.25/8.25 Mbaud @ 25/33 MHz CPU clock. It may be configured so it interfaces with serially linked peripheral components. A dedicated baud rate generator allows to set up all standard baud rates without oscillator tuning. For transmission, reception and error handling 3 separate interrupt vectors are provided.

The SSC transmits or receives characters of 2 ... 16-bits length synchronously to a shift clock which can be generated by the SSC (master mode) or by an external master (slave mode). The SSC can start shifting with the LSB or with the MSB and allows the selection of shifting and latching clock edges as well as the clock polarity.

A number of optional hardware error detection capabilities has been included to increase the reliability of data transfers. Transmit and receive error supervise the correct handling of the data buffer. Phase and baudrate error detect incorrect serial data.

### **The On-Chip CAN Modules**

The integrated CAN Modules (CAN1, CAN2) handle the completely autonomous transmission and reception of CAN frames in accordance with the CAN specification V2.0 part B (active), i.e. the on-chip CAN Module can receive and transmit standard frames with 11-bit identifiers as well as extended frames with 29-bit identifiers.

The modules provide Full CAN functionality on up to 15 message objects (up to 30 objects if both modules are connected to the same physical bus). Message object 15 may be configured for Basic CAN functionality. Both modes provide separate masks for acceptance filtering which allows to accept a number of identifiers in Full CAN mode and also allows to disregard a number of identifiers in Basic CAN mode. All message objects can be updated independent from the other objects and are equipped for the maximum message length of 8 Bytes.

The bit timing is derived from the XCLK and is programmable up to a data rate of 1 MBaud. Each CAN Module uses two pins (configurable, both modules may use the same pair of pins) to interface to a bus transceiver.

*Note: The C161JC provides one CAN module, the C161CS provides two CAN modules.*

### **The On-chip IIC Bus Module**

The integrated IIC Module handles the transmission and reception of frames over the two-line IIC bus in accordance with the IIC Bus specification. The on-chip IIC Module can receive and transmit data using 7-bit or 10-bit addressing and it can operate in slave mode, in master mode or in multi-master mode.

Several physical interfaces (port pins) can be established under software control. Data can be transferred at speeds up to 400 kbit/s.

Two interrupt nodes dedicated to the IIC module allow efficient interrupt service and also support operation via PEC transfers.

*Note: The port pins associated with the IIC interfaces feature open drain drivers only, as required by the IIC specification.*



**Serial Data Link Module (SDLM)**

The Serial Data Link Module (SDLM) provides serial communication via a J1850 type multiplexed serial bus via an external J1850 bus transceiver. The module conforms to the SAE Class B J1850 specification for variable pulse width modulation (VPW). The SDLM is integrated as an on-chip peripheral and is connected to the CPU via the XBUS.

**General SDLM Features:**

- Compliant to the SAE Class B J1850 specification (VPW), Class 2 protocol fully supported
- Variable Pulse Width (VPW) operation at 10.4 kBaud, High Speed 4X operation at 41.6 kBaud
- Programmable Normalization Bit, programmable delay for transceiver interface
- Digital Noise Filter
- Power Down mode with automatic wakeup support upon bus activity
- Single Byte Header and Consolidated Header supported
- CRC generation and checking
- Receive and transmit Block Mode

**Data Link Operation Features:**

- 11 Byte Transmit Buffer and double buffered 11 Byte receive buffer (optional overwrite enable)
- Support for In Frame Response (IFR) types 1, 2 and 3
- Transmit and Receiver Message Buffers configurable for either FIFO or Byte mode
- Advanced Interrupt Handling with 8 separately enabled sources
- Automatic IFR transmission (Types 1 and 2) for 3-Byte consolidated headers
- User configurable clock divider
- Bus status flags (IDLE, EOF, EOD, SOF, Tx and Rx in progress)

## Parallel Ports

The C161CS/JC/JI provides up to 93 IO lines which are organized into eight input/output ports and one input port. All port lines are bit-addressable, and all input/output lines are individually (bit-wise) programmable as inputs or outputs via direction registers. The IO ports are true bidirectional ports which are switched to high impedance state when configured as inputs. The output drivers of five IO ports can be configured (pin by pin) for push/pull operation or open-drain operation via control registers. During the internal reset, all port pins are configured as inputs.

All port lines have programmable alternate input or output functions associated with them. PORT0 and PORT1 may be used as address and data lines when accessing external memory, while Port 4 outputs the additional segment address bits A23/19/17 ... A16 in systems where segmentation is used to access more than 64 KBytes of memory. Port 6 provides the optional bus arbitration signals ( $\overline{\text{BREQ}}$ ,  $\overline{\text{HLDA}}$ ,  $\overline{\text{HOLD}}$ ) and the chip select signals  $\overline{\text{CS4}}$  ...  $\overline{\text{CS0}}$ . Port 2 accepts the fast external interrupt inputs and provides inputs/outputs for the CAPCOM1 unit. Port 3 includes alternate functions of timers, serial interfaces, the optional bus control signal  $\overline{\text{BHE}}$ , and the system clock output (CLKOUT/ FOUT). Port 5 is used for timer control signals and for the analog inputs to the A/D Converter. Port 7 provides inputs/outputs for the CAPCOM2 unit (more on P1H). Port 9 provides the IIC Bus lines. Four pins of PORT1 may also be used as inputs/outputs for the CAPCOM2 unit. All port lines that are not used for these alternate functions may be used as general purpose IO lines.

## Watchdog Timer

The Watchdog Timer represents one of the fail-safe mechanisms which have been implemented to prevent the controller from malfunctioning for longer periods of time.

The Watchdog Timer is always enabled after a reset of the chip, and can only be disabled in the time interval until the EINIT (end of initialization) instruction has been executed. Thus, the chip's start-up procedure is always monitored. The software has to be designed to service the Watchdog Timer before it overflows. If, due to hardware or software related failures, the software fails to do so, the Watchdog Timer overflows and generates an internal hardware reset and pulls the  $\overline{\text{RSTOUT}}$  pin low in order to allow external hardware components to reset.

The Watchdog Timer is a 16-bit timer, clocked with the CPU clock divided by 2, 4, 128, or 256. The high byte of the Watchdog Timer register can be set to a prespecified reload value (stored in WDTREL) in order to allow further variation of the monitored time interval. Each time it is serviced by the application software, the high byte of the Watchdog Timer is reloaded. Thus, time intervals between 21  $\mu\text{s}$  and 671 ms can be monitored @ 25 MHz (16  $\mu\text{s}$  and 335 ms @ 33 MHz). The default Watchdog Timer interval after reset is 5.2/4.0 ms (@ 25/33 MHz).

## **A/D Converter**

For analog signal measurement, a 10-bit A/D converter with 12 multiplexed input channels and a sample and hold circuit has been integrated on-chip. It uses the method of successive approximation. The sample time (for loading the capacitors) and the conversion time is programmable and can so be adjusted to the external circuitry.

Overrun error detection/protection is provided for the conversion result register (ADDAT): either an interrupt request will be generated when the result of a previous conversion has not been read from the result register at the time the next conversion is complete, or the next conversion is suspended in such a case until the previous result has been read.

For applications which require less analog input channels, the remaining channel inputs can be used as digital input (or IO) port pins.

The A/D converter of the C161CS/JC/JI supports four different conversion modes. In the standard Single Channel conversion mode, the analog level on a specified channel is sampled once and converted to a digital result. In the Single Channel Continuous mode, the analog level on a specified channel is repeatedly sampled and converted without software intervention. In the Auto Scan mode, the analog levels on a prespecified number of channels are sequentially sampled and converted. In the Auto Scan Continuous mode, the number of prespecified channels is repeatedly sampled and converted. In addition, the conversion of a specific channel can be inserted (injected) into a running sequence without disturbing this sequence. This is called Channel Injection Mode.

The Peripheral Event Controller (PEC) may be used to automatically store the conversion results into a table in memory for later evaluation, without requiring the overhead of entering and exiting interrupt routines for each data transfer.

## **Real Time Clock**

The C161CS/JC/JI contains a real time clock (RTC) which serves for different purposes:

- System clock to determine the current time and date, even during idle mode and power down mode (optionally)
- Cyclic time based interrupt, e.g. to provide a system time tick independent of the CPU frequency without loading the general purpose timers, or to wake up regularly from idle mode.
- 48-bit timer for long term measurements, the maximum usable timespan is more than 100 years.

The RTC module consists of a chain of 3 divider blocks, a fixed 8:1 divider, the reloadable 16-bit timer T14 and the 32-bit RTC timer (accessible via registers RTCH and RTCL). Both timers count up.

## **General Purpose Timer (GPT) Unit**

The GPT units represent a very flexible multifunctional timer/counter structure which may be used for many different time related tasks such as event timing and counting, pulse width and duty cycle measurements, pulse generation, or pulse multiplication.

The five 16-bit timers are organized in two separate modules, GPT1 and GPT2. Each timer in each module may operate independently in a number of different modes, or may be concatenated with another timer of the same module.

Each timer can be configured individually for one of four basic modes of operation, which are Timer, Gated Timer, Counter Mode and Incremental Interface Mode (GPT1 timers). In Timer Mode the input clock for a timer is derived from the internal CPU clock divided by a programmable prescaler, while Counter Mode allows a timer to be clocked in reference to external events (via TxIN).

Pulse width or duty cycle measurement is supported in Gated Timer Mode where the operation of a timer is controlled by the 'gate' level on its external input pin TxIN.

In Incremental Interface Mode the GPT1 timers can be directly connected to the incremental position sensor signals A and B via the respective inputs TxIN and TxEUD. Direction and count signals are internally derived from these two input signals, so the contents of timer Tx corresponds to the sensor position. The third position sensor signal TOP0 can be connected to an interrupt input.

The count direction (up/down) for each timer is programmable by software or may additionally be altered dynamically by an external signal (TxEUD) to facilitate e.g. position tracking.

The core timers T3 and T6 have output toggle latches (TxOTL) which change their state on each timer over-flow/underflow. The state of these latches may be output on port pins (TxOUT) or may be used internally to concatenate the core timers with the respective auxiliary timers resulting in 32/33-bit timers/counters for measuring long time periods with high resolution.

Various reload or capture functions can be selected to reload timers or capture a timer's contents triggered by an external signal or a selectable transition of toggle latch TxOTL.

The maximum resolution of the timers in module GPT1 is 8 CPU clock cycles (= 16 TCL). With their maximum resolution of 4 CPU clock cycles (= 8 TCL) the GPT2 timers provide precise event control and time measurement.

## **Capture/Compare (CAPCOM) Units**

The two CAPCOM units support generation and control of timing sequences on up to 32 channels with a maximum resolution of 8 CPU clock cycles. The CAPCOM units are typically used to handle high speed IO tasks such as pulse and waveform generation, pulse width modulation (PWM), Digital to Analog (D/A) conversion, software timing, or time recording relative to external events.

Four 16-bit timers (T0/T1, T7/T8) with reload registers provide two independent time bases for the capture/compare register array.

The input clock for the timers is programmable to several prescaled values of the internal CPU clock, or may be derived from an overflow/underflow of timer T6 in module GPT2. This provides a wide range of variation for the timer period and resolution and allows precise adjustments to the application specific requirements. In addition, external count inputs for CAPCOM timers T0 and T7 allow event scheduling for the capture/compare registers relative to external events.

Both of the two capture/compare register arrays contain 16 dual purpose capture/compare registers, each of which may be individually allocated to either CAPCOM timer T0 or T1 (T7 or T8, respectively), and programmed for capture or compare function. Eight register of each unit have one port pin associated with it which serves as an input pin for triggering the capture function, or as an output pin to indicate the occurrence of a compare event.

When a capture/compare register has been selected for capture mode, the current contents of the allocated timer will be latched (captured) into the capture/compare register in response to an external event at the port pin which is associated with this register. In addition, a specific interrupt request for this capture/compare register is generated. Either a positive, a negative, or both a positive and a negative external signal transition at the pin can be selected as the triggering event. The contents of all registers which have been selected for one of the five compare modes are continuously compared with the contents of the allocated timers. When a match occurs between the timer value and the value in a capture/compare register, specific actions will be taken based on the selected compare mode.

## **2.4 Power Management Features**

The known basic power reduction modes (Idle and Power Down) are enhanced by a number of additional power management features (see below). These features can be combined to reduce the controller's power consumption to the respective application's possible minimum.

- Flexible clock generation
- Flexible peripheral management (peripherals can be enabled/disabled separately or in groups)
- Periodic wakeup from Idle mode via RTC timer

The listed features provide effective means to realize standby conditions for the system with an optimum balance between power reduction (i.e. standby time) and peripheral operation (i.e. system functionality).

### **Flexible Clock Generation**

The flexible clock generation system combines a variety of improved mechanisms (partly user controllable) to provide the C161CS/JC/JI modules with clock signals. This is especially important in power sensitive modes like standby operation.

**The power optimized oscillator** generally reduces the amount of power which is consumed in order to generate the clock signal within the C161CS/JC/JI.

**The clock system** efficiently controls the amount of power which is consumed in order to distribute the clock signal within the C161CS/JC/JI.

**Slowdown operation** is achieved by dividing the oscillator clock by a programmable factor (1 ... 32) resulting in a low frequency device operation which significantly reduces the overall power consumption.

### **Flexible Peripheral Management**

The flexible peripheral management provides a mechanism to enable and disable each peripheral module separately. In each situation (e.g. several system operating modes, standby, etc.) only those peripherals may be kept running which are required for the respective functionality. All others can be switched off. It also allows the operation control of whole groups of peripherals including the power required for generating and distributing their clock input signal. Other peripherals may remain active, e.g. in order to maintain communication channels. The registers of separately disabled peripherals (not within a disabled group) can still be accessed.

### **Periodic Wakeup from Idle Mode**

Periodic wakeup from Idle mode combines the drastically reduced power consumption in Idle mode (in conjunction with the additional power management features) with a high level of system availability. External signals and events can be scanned (at a lower rate) by periodically activating the CPU and selected peripherals which then return to powersave mode after a short time. This greatly reduces the system's average power consumption.

## 2.5 Protected Bits

The C161CS/JC/JI provides a special mechanism to protect bits which can be modified by the on-chip hardware from being changed unintentionally by software accesses to related bits (see also [Chapter 4](#)).

The following bits are protected.

**Table 2-1 C161CS/JC/JI Protected Bits**

Register	Bit Name	Notes
T2IC, T3IC, T4IC	<b>T2IR, T3IR, T4IR</b>	GPT1 timer interrupt request flags
T5IC, T6IC	<b>T5IR, T6IR</b>	GPT2 timer interrupt request flags
CRIC	<b>CRIR</b>	GPT2 CAPREL interrupt request flag
T3CON, T6CON	<b>T3OTL, T6OTL</b>	GPTx timer output toggle latches
T0IC, T1IC	<b>T0IR, T1IR</b>	CAPCOM1 timer interrupt request flags
T7IC, T8IC	<b>T7IR, T8IR</b>	CAPCOM2 timer interrupt request flags
S0TIC, S0TBIC	<b>S0TIR, S0TBIR</b>	ASC0 transmit(buffer) interrupt request flags
S0RIC, S0EIC	<b>S0RIR, S0EIR</b>	ASC0 receive/error interrupt request flags
S0CON	<b>S0REN</b>	ASC0 receiver enable flag
SSCTIC, SSCRIC	<b>SSCTIR, SSCRIR</b>	SSC transmit/receive interrupt request flags
SSCEIC	<b>SSCEIR</b>	SSC error interrupt request flag
SSCCON	<b>SSCBSY</b>	SSC busy flag
SSCCON	<b>SSCBE, SSCPE</b>	SSC error flags
SSCCON	<b>SSCRE, SSCTE</b>	SSC error flags
ADCIC, ADEIC	<b>ADCIR, ADEIR</b>	ADC end-of-conv./overrun intr. request flag
ADCON	<b>ADST, ADCRQ</b>	ADC start flag/injection request flag
CC31IC ... CC16IC	<b>CC31IR ... CC16IR</b>	CAPCOM2 interrupt request flags
CC15IC ... CC0IC	<b>CC15IR ... CC0IR</b>	CAPCOM1 interrupt request flags
TFR	<b>TFR.15,14,13</b>	Class A trap flags
TFR	<b>TFR.7,3,2,1,0</b>	Class B trap flags
P2	<b>P2.15 ... P2.8</b>	All bits of Port 2
P7	<b>P7.7 ... P7.4</b>	All bits of Port 7
XP7IC ... XP0IC	<b>XP7IR ... XP0IC</b>	X-Peripheral interrupt request flags



**Table 2-1 C161CS/JC/JI Protected Bits (cont'd)**

<b>Register</b>	<b>Bit Name</b>	<b>Notes</b>
ISNC	<b>RTCIR, PLLIR</b>	Interrupt node sharing request flags
FOCON	<b>FOTL, FOCNT.5 ... 0</b>	Frequency output toggle latch and counter

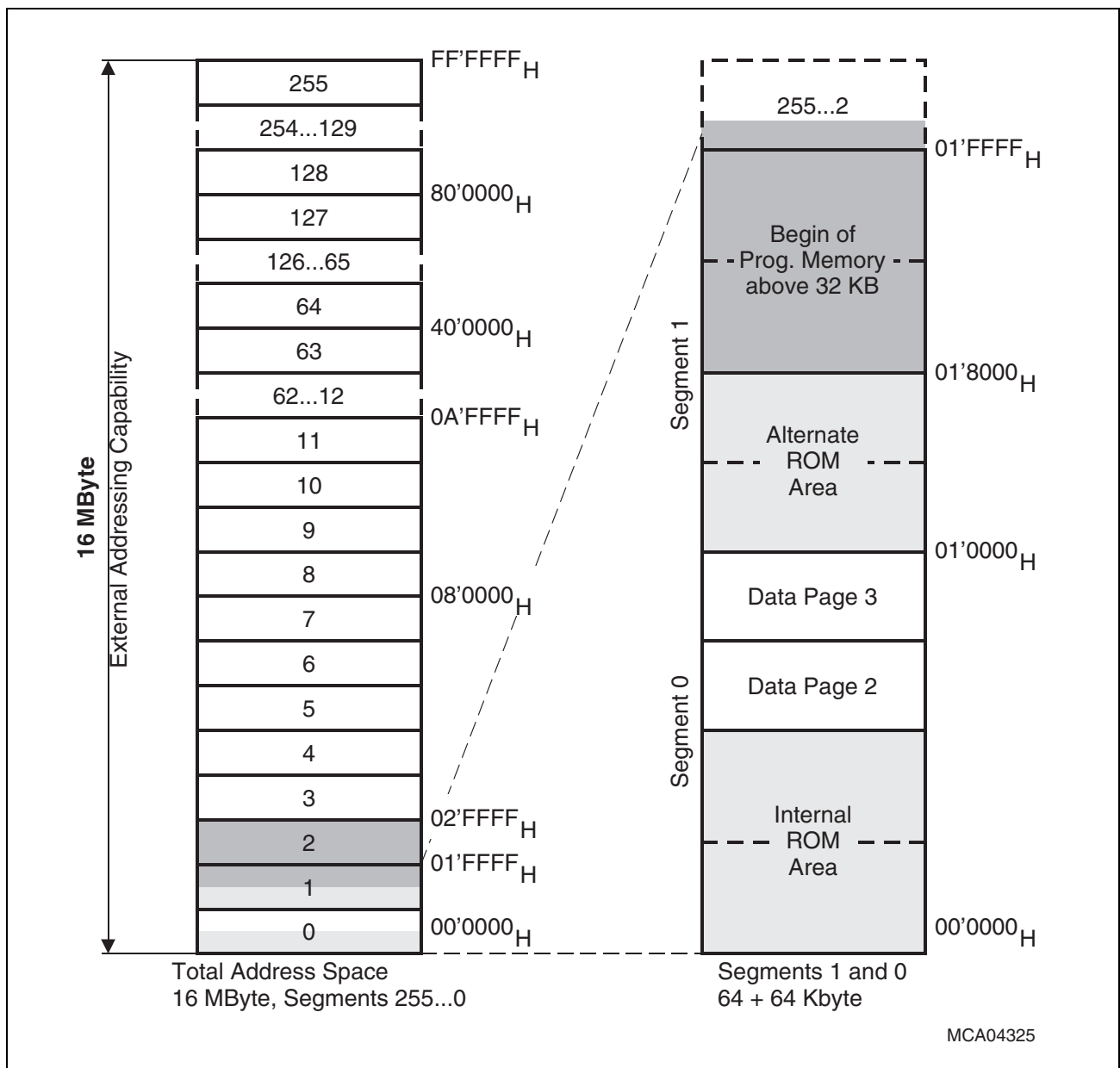
$\Sigma = 98$  protected bits.



### 3 Memory Organization

The memory space of the C161CS/JC/JI is configured in a “Von Neumann” architecture. This means that code and data are accessed within the same linear address space. All of the physically separated memory areas, including internal ROM/Flash/OTP (where integrated), internal RAM, the internal Special Function Register Areas (SFRs and ESFRs), the address areas for integrated XBUS peripherals and external memory are mapped into one common address space.

The C161CS/JC/JI provides a total addressable memory space of 16 MBytes. This address space is arranged as 256 segments of 64 KBytes each, and each segment is again subdivided into four data pages of 16 KBytes each (see **Figure 3-1**).



**Figure 3-1 Address Space Overview**

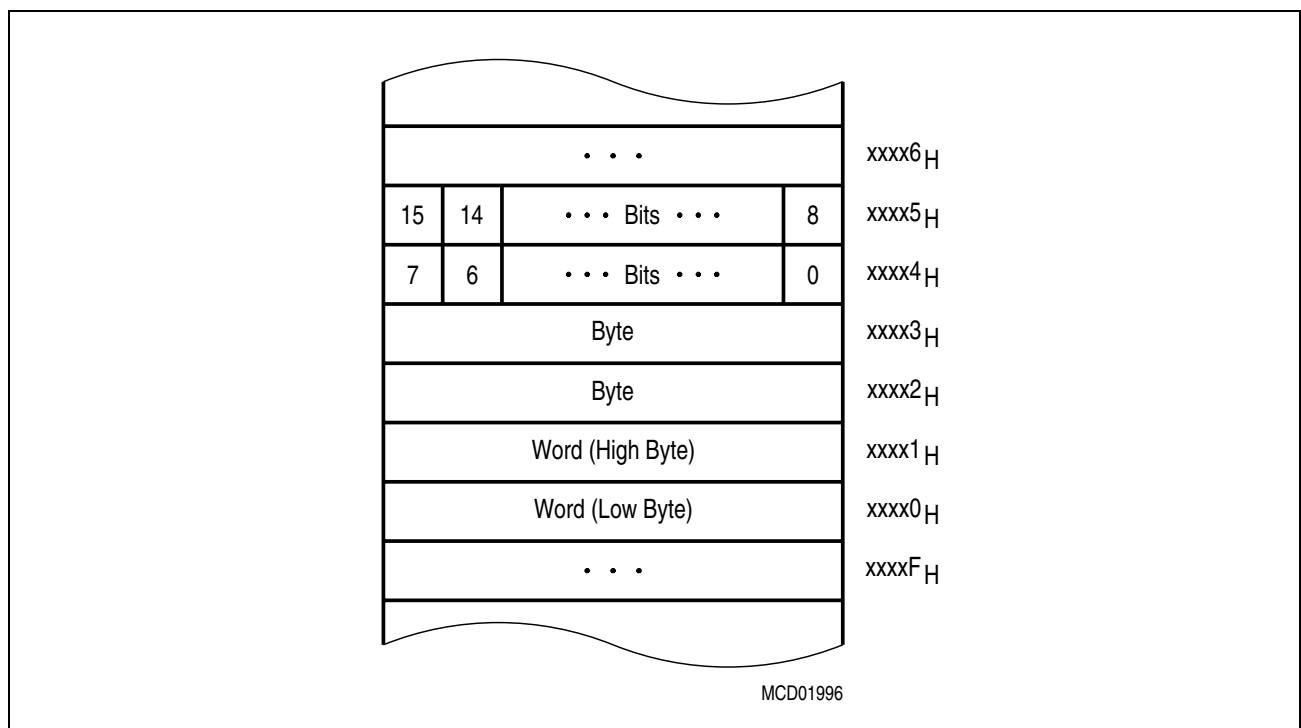
**Memory Organization**

Most internal memory areas are mapped into segment 0, the system segment. The upper 4 KByte of segment 0 (00'F000<sub>H</sub> ... 00'FFFF<sub>H</sub>) hold the Internal RAM and Special Function Register Areas (SFR and ESFR). The lower 32 KByte of segment 0 (00'0000<sub>H</sub> ... 00'7FFF<sub>H</sub>) may be occupied by a part of the on-chip ROM/Flash/OTP memory and is called the Internal ROM area. This ROM area can be remapped to segment 1 (01'0000<sub>H</sub> ... 01'7FFF<sub>H</sub>), to enable external memory access in the lower half of segment 0, or the internal ROM may be disabled at all.

Code and data may be stored in any part of the internal memory areas, except for the SFR blocks, which may be used for control/data, but not for instructions.

*Note: Accesses to the internal ROM area on ROMless devices will produce unpredictable results.*

Bytes are stored at even or odd byte addresses. Words are stored in ascending memory locations with the low byte at an even byte address being followed by the high byte at the next odd byte address. Double words (code only) are stored in ascending memory locations as two subsequent words. Single bits are always stored in the specified bit position at a word address. Bit position 0 is the least significant bit of the byte at an even byte address, and bit position 15 is the most significant bit of the byte at the next odd byte address. Bit addressing is supported for a part of the Special Function Registers, a part of the internal RAM and for the General Purpose Registers.



**Figure 3-2 Storage of Words, Bytes, and Bits in a Byte Organized Memory**

*Note: Byte units forming a single word or a double word must always be stored within the same physical (internal, external, ROM, RAM) and organizational (page, segment) memory area.*

### 3.1 Internal ROM Area

The C161CS/JC/JI may reserve an address area of variable size (depending on the version) for on-chip mask-programmable ROM/Flash/OTP memory (organized as  $X \times 32$ ). The lower 32 KByte of this on-chip memory block are referred to as “Internal ROM Area”. Internal ROM accesses are globally enabled or disabled via bit ROMEN in register SYSCON. This bit is set during reset according to the level on pin  $\overline{EA}$ , or may be altered via software. If enabled, the internal ROM area occupies the lower 32 KByte of either segment 0 or segment 1 (alternate ROM area). This mapping is controlled by bit ROMS1 in register SYSCON.

*Note: The size of the internal ROM area is independent of the size of the actual implemented Program Memory. Also devices with less than 32 KByte of Program Memory or with no Program Memory at all will have this 32 KByte area occupied, if the Program Memory is enabled. Devices with a larger Program Memory provide the mapping option only for the internal ROM area.*

Devices with a Program Memory size above 32 KByte expand the ROM area from the middle of segment 1, i.e. starting at address  $01'8000_H$ .

The internal Program Memory can be used for both code (instructions) and data (constants, tables, etc.) storage.

Code fetches are always made on even byte addresses. The highest possible code storage location in the internal Program Memory is either  $xx'xxFE_H$  for single word instructions, or  $xx'xxFC_H$  for double word instructions. The respective location must contain a branch instruction (unconditional), because sequential boundary crossing from internal Program Memory to external memory is not supported and causes erroneous results.

Any word and byte data read accesses may use the indirect or long 16-bit addressing modes. There is no short addressing mode for internal ROM operands. Any word data access is made to an even byte address. The highest possible word data storage location in the internal ROM is  $xx'xxFE_H$ . For PEC data transfers the internal Program Memory can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.

The internal Program Memory is not provided for single bit storage, and therefore it is not bit addressable.

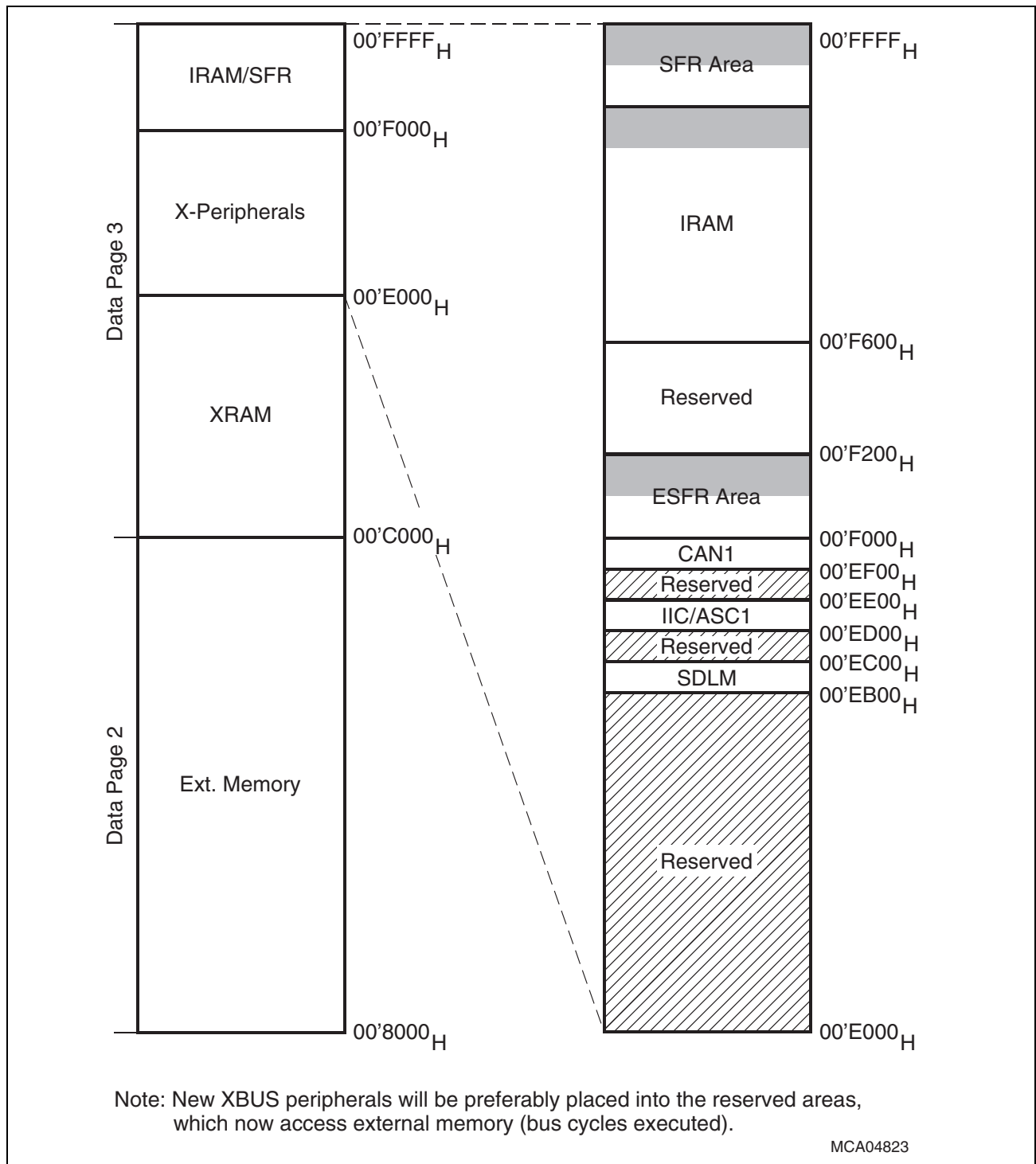
*Note: The ‘x’ in the locations above depend on the available Program Memory and on the mapping.*

The internal ROM may be enabled, disabled or mapped into segment 0 or segment 1 under software control. **Chapter 24** shows how to do this and reminds of the precautions that must be taken in order to prevent the system from crashing.

### 3.2 Internal RAM and SFR Area

The RAM/SFR area is located within data page 3 and provides access to the internal RAM (IRAM, organized as  $X \times 16$ ) and to two 512 Byte blocks of Special Function Registers (SFRs).

The C161CS/JC/JI provides 2 KByte of IRAM.



**Figure 3-3 System Memory Map**

**Memory Organization**

*Note: The upper 256 Bytes of SFR area, ESFR area and internal RAM are bit-addressable (see shaded blocks in [Figure 3-3](#)).*

Code accesses are always made on even byte addresses. The highest possible code storage location in the internal RAM is either 00'FDFF<sub>H</sub> for single word instructions or 00'FDFF<sub>H</sub> for double word instructions. The respective location must contain a branch instruction (unconditional), because sequential boundary crossing from internal RAM to the SFR area is not supported and causes erroneous results.

Any word and byte data in the internal RAM can be accessed via indirect or long 16-bit addressing modes, if the selected DPP register points to data page 3. Any word data access is made on an even byte address. The highest possible word data storage location in the internal RAM is 00'FDFF<sub>H</sub>. For PEC data transfers, the internal RAM can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.

The upper 256 Byte of the internal RAM (00'FD00<sub>H</sub> through 00'FDFF<sub>H</sub>) and the GPRs of the current bank are provided for single bit storage, and thus they are bitaddressable.

**System Stack**

The system stack may be defined within the internal RAM. The size of the system stack is controlled by bitfield STKSZ in register SYSCON (see [Table 3-1](#)).

**Table 3-1 System Stack Size Encoding**

<b>&lt;STKSZ&gt;</b>	<b>Stack Size (words)</b>	<b>Internal RAM Addresses (words)</b>
0 0 0 <sub>B</sub>	256	00'FBFF <sub>H</sub> ... 00'FA00 <sub>H</sub> (Default after Reset)
0 0 1 <sub>B</sub>	128	00'FBFF <sub>H</sub> ... 00'FB00 <sub>H</sub>
0 1 0 <sub>B</sub>	64	00'FBFF <sub>H</sub> ... 00'FB80 <sub>H</sub>
0 1 1 <sub>B</sub>	32	00'FBFF <sub>H</sub> ... 00'FBC0 <sub>H</sub>
1 0 0 <sub>B</sub>	512	00'FBFF <sub>H</sub> ... 00'F800 <sub>H</sub>
1 0 1 <sub>B</sub>	–	Reserved. Do not use this combination.
1 1 0 <sub>B</sub>	–	Reserved. Do not use this combination.
1 1 1 <sub>B</sub>	1024	00'FDFF <sub>H</sub> ... 00'F600 <sub>H</sub> (Note: No circular stack)

For all system stack operations the on-chip RAM is accessed via the Stack Pointer (SP) register. The stack grows downward from higher towards lower RAM address locations. Only word accesses are supported to the system stack. A stack overflow (STKOV) and a stack underflow (STKUN) register are provided to control the lower and upper limits of the selected stack area. These two stack boundary registers can be used not only for protection against data destruction, but also allow to implement a circular stack with hardware supported system stack flushing and filling (except for option '111'). The technique of implementing this circular stack is described in [Chapter 24](#).

### General Purpose Registers

The General Purpose Registers (GPRs) use a block of 16 consecutive words within the internal RAM. The Context Pointer (CP) register determines the base address of the currently active register bank. This register bank may consist of up to 16 Word-GPRs (R0, R1, ... R15) and/or of up to 16 Byte-GPRs (RL0, RH0, ... RL7, RH7). The sixteen Byte-GPRs are mapped onto the first eight Word-GPRs (see [Table 3-2](#)).

In contrast to the system stack, a register bank grows from lower towards higher address locations and occupies a maximum space of 32 Byte. The GPRs are accessed via short 2-, 4-, or 8-bit addressing modes using the Context Pointer (CP) register as base address (independent of the current DPP register contents). Additionally, each bit in the currently active register bank can be accessed individually.

**Table 3-2 Mapping of General Purpose Registers to RAM Addresses**

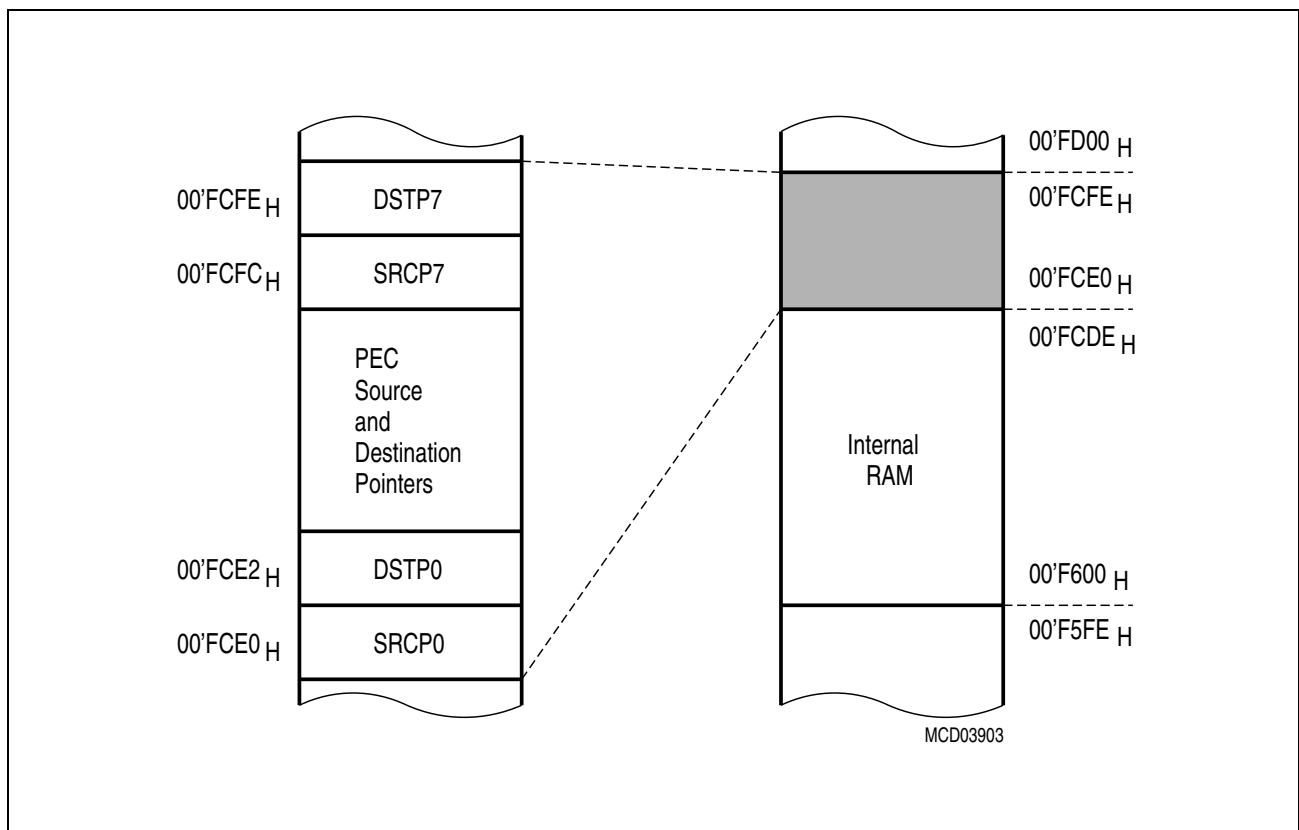
Internal RAM Address	Byte Registers		Word Register
<CP> + 1E <sub>H</sub>	–		R15
<CP> + 1C <sub>H</sub>	–		R14
<CP> + 1A <sub>H</sub>	–		R13
<CP> + 18 <sub>H</sub>	–		R12
<CP> + 16 <sub>H</sub>	–		R11
<CP> + 14 <sub>H</sub>	–		R10
<CP> + 12 <sub>H</sub>	–		R9
<CP> + 10 <sub>H</sub>	–		R8
<CP> + 0E <sub>H</sub>	RH7	RL7	R7
<CP> + 0C <sub>H</sub>	RH6	RL6	R6
<CP> + 0A <sub>H</sub>	RH5	RL5	R5
<CP> + 08 <sub>H</sub>	RH4	RL4	R4
<CP> + 06 <sub>H</sub>	RH3	RL3	R3
<CP> + 04 <sub>H</sub>	RH2	RL2	R2
<CP> + 02 <sub>H</sub>	RH1	RL1	R1
<CP> + 00 <sub>H</sub>	RH0	RL0	R0

The C161CS/JC/JI supports fast register bank (context) switching. Multiple register banks can physically exist within the internal RAM at the same time. Only the register bank selected by the Context Pointer register (CP) is active at a given time, however. Selecting a new active register bank is simply done by updating the CP register. A particular Switch Context (SCXT) instruction performs register bank switching and an

automatic saving of the previous context. The number of implemented register banks (arbitrary sizes) is only limited by the size of the available internal RAM. Details on using, switching and overlapping register banks are described in [Chapter 24](#).

### PEC Source and Destination Pointers

The 16 word locations in the internal RAM from 00'FCE0<sub>H</sub> to 00'FCFE<sub>H</sub> (just below the bit-addressable section) are provided as source and destination address pointers for data transfers on the eight PEC channels. Each channel uses a pair of pointers stored in two subsequent word locations with the source pointer (SRCP<sub>x</sub>) on the lower and the destination pointer (DSTP<sub>x</sub>) on the higher word address (x = 7 ... 0).



**Figure 3-4 Location of the PEC Pointers**

Whenever a PEC data transfer is performed, the pair of source and destination pointers, which is selected by the specified PEC channel number, is accessed independent of the current DPP register contents and also the locations referred to by these pointers are accessed independent of the current DPP register contents. If a PEC channel is not used, the corresponding pointer locations area available and can be used for word or byte data storage.

For more details about the use of the source and destination pointers for PEC data transfers see [Chapter 5](#).

## Special Function Registers

The functions of the CPU, the bus interface, the IO ports and the on-chip peripherals of the C161CS/JC/JI are controlled via a number of so-called Special Function Registers (SFRs). These SFRs are arranged within two areas of 512 Byte size each. The first register block, the SFR area, is located in the 512 Bytes above the internal RAM (00'FFFF<sub>H</sub> ... 00'FE00<sub>H</sub>), the second register block, the Extended SFR (ESFR) area, is located in the 512 Bytes below the internal RAM (00'F1FF<sub>H</sub> ... 00'F000<sub>H</sub>).

Special function registers can be addressed via indirect and long 16-bit addressing modes. Using an 8-bit offset together with an implicit base address allows to address word SFRs and their respective low bytes. However, this **does not work** for the respective high bytes!

*Note: Writing to any byte of an SFR causes the non-addressed complementary byte to be cleared!*

The upper half of each register block is bit-addressable, so the respective control/status bits can directly be modified or checked using bit addressing.

When accessing registers in the ESFR area using 8-bit addresses or direct bit addressing, an Extend Register (EXTR) instruction is required before, to switch the short addressing mechanism from the standard SFR area to the Extended SFR area. This is not required for 16-bit and indirect addresses. The GPRs R15 ... R0 are duplicated, i.e. they are accessible within both register blocks via short 2-, 4- or 8-bit addresses without switching.

ESFR\_SWITCH\_EXAMPLE:

```
EXTR    #4                ;Switch to ESFR area for next 4 instr.
MOV     ODP2, #data16     ;ODP2 uses 8-bit reg addressing
BFLDL  DP6, #mask, #data8 ;Bit addressing for bit fields
BSET   DP1H.7           ;Bit addressing for single bits
MOV     T8REL, R1        ;T8REL uses 16-bit mem address,
                        ;R1 is duplicated into the ESFR space
                        ;(EXTR is not required for this access)
;---- ;-----          ;The scope of the EXTR #4 instruction...
                        ; ... ends here!
MOV     T8REL, R1        ;T8REL uses 16-bit mem address,
                        ;R1 is accessed via the SFR space
```

In order to minimize the use of the EXTR instructions the ESFR area mostly holds registers which are mainly required for initialization and mode selection. Registers that need to be accessed frequently are allocated to the standard SFR area, wherever possible.

*Note: The tools are equipped to monitor accesses to the ESFR area and will automatically insert EXTR instructions, or issue a warning in case of missing or excessive EXTR instructions.*



### 3.3 The On-Chip XRAM

The C161CS/JC/JI provides access to 8 KByte of on-chip extension RAM. The XRAM is located within data page 3 (organized as  $4\text{ K} \times 16$ ). As the XRAM is connected to the internal XBUS it is accessed like external memory, however, no external bus cycles are executed for these accesses. XRAM accesses are globally enabled or disabled via bit XPEN in register SYSCON. This bit is cleared after reset and may be set via software during the initialization to allow accesses to the on-chip XRAM. When the XRAM is disabled (default after reset) all accesses to the XRAM area are mapped to external locations. The XRAM may be used for both code (instructions) and data (variables, user stack, tables, etc.) storage.

Code fetches are always made on even byte addresses. The highest possible code storage location in the XRAM is either  $00'DFFE_H$  for single word instructions, or  $00'DFFC_H$  for double word instructions. The respective location must contain a branch instruction (unconditional), because sequential boundary crossing from XRAM to external memory is not supported and causes erroneous results.

Any word and byte data read accesses may use the indirect or long 16-bit addressing modes. There is no short addressing mode for XRAM operands. Any word data access is made to an even byte address. The highest possible word data storage location in the XRAM is  $00'DFFE_H$ . For PEC data transfers the XRAM can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.

*Note: As the XRAM appears like external memory it cannot be used for the C161CS/JC/JI's system stack or register banks. The XRAM is not provided for single bit storage and therefore is not bitaddressable.*

The on-chip XRAM is accessed with the following bus cycles:

- Normal ALE
- No cycle time waitstates (no  $\overline{\text{READY}}$  control)
- No tristate time waitstate
- No Read/Write delay
- 16-bit demultiplexed bus cycles (4 TCL)

Even if the XRAM is used like external memory it does not occupy BUSCONx/ADDRSELx registers but rather is selected via additional dedicated XBCON/XADRS registers. These registers are mask-programmed and are not user accessible. With these registers the address area  $00'C000_H$  to  $00'DFFF_H$  are reserved for XRAM accesses.

**XRAM Access via External Masters**

In X-Peripheral Share mode (bit XPER-SHARE in register SYSCON is set) the on-chip XRAM of the C161CS/JC/JI can be accessed by an external master during hold mode via the C161CS/JC/JI's bus interface. These external accesses must use the same configuration as internally programmed (see above). No waitstates are required. In X-Peripheral Share mode the C161CS/JC/JI bus interface reverses its direction, i.e. address lines (PORT1, Port 4), control signals ( $\overline{RD}$ ,  $\overline{WR}$ ), and  $\overline{BHE}$  must be driven by the external master.

*Note: The configuration in register SYSCON cannot be changed after the execution of the EINIT instruction.*

### 3.4 External Memory Space

The C161CS/JC/JI is capable of using an address space of up to 16 MByte. Only parts of this address space are occupied by internal memory areas. All addresses which are not used for on-chip memory (ROM/Flash/OTP or RAM) or for registers may reference external memory locations. This external memory is accessed via the C161CS/JC/JI's external bus interface.

**Four memory bank sizes** are supported:

- Non-segmented mode: 64 KByte with A15 ... A0 on PORT0 or PORT1
- 2-bit segmented mode: 256 KByte with A17 ... A16 on Port 4  
and A15 ... A0 on PORT0 or PORT1
- 4-bit segmented mode: 1 MByte with A19 ... A16 on Port 4  
and A15 ... A0 on PORT0 or PORT1
- 8-bit segmented mode: 16 MByte with A22 ... A16 on Port 4  
and A15 ... A0 on PORT0 or PORT1

Each bank can be directly addressed via the address bus, while the programmable chip select signals can be used to select various memory banks.

The C161CS/JC/JI also supports **four different bus types**:

- Multiplexed 16-bit Bus with address and data on PORT0 (Default after Reset)
- Multiplexed 8-bit Bus with address and data on PORT0/P0L
- Demultiplexed 16-bit Bus with address on PORT1 and data on PORT0
- Demultiplexed 8-bit Bus with address on PORT1 and data on P0L

Memory model and bus mode are selected during reset by pin  $\overline{EA}$  and PORT0 pins. For further details about the external bus configuration and control please refer to [Chapter 9](#).

External word and byte data can only be accessed via indirect or long 16-bit addressing modes using one of the four DPP registers. There is no short addressing mode for external operands. Any word data access is made to an even byte address.

For PEC data transfers the external memory in segment 0 can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.

The external memory is not provided for single bit storage and therefore it is not bitaddressable.

### 3.5 Crossing Memory Boundaries

The address space of the C161CS/JC/JI is implicitly divided into equally sized blocks of different granularity and into logical memory areas. Crossing the boundaries between these blocks (code or data) or areas requires special attention to ensure that the controller executes the desired operations.

**Memory Areas** are partitions of the address space that represent different kinds of memory (if provided at all). These memory areas are the internal RAM/SFR area, the internal ROM/Flash/OTP (if available), the on-chip X-Peripherals (if integrated) and the external memory.

Accessing subsequent data locations that belong to different memory areas is no problem. However, when executing code, the different memory areas must be switched explicitly via branch instructions. Sequential boundary crossing is not supported and leads to erroneous results.

*Note: Changing from the external memory area to the internal RAM/SFR area takes place within segment 0.*

**Segments** are contiguous blocks of 64 KByte each. They are referenced via the code segment pointer CSP for code fetches and via an explicit segment number for data accesses overriding the standard DPP scheme.

During code fetching segments are not changed automatically, but rather must be switched explicitly. The instructions JMPS, CALLS and RETS will do this.

In larger sequential programs make sure that the highest used code location of a segment contains an unconditional branch instruction to the respective following segment, to prevent the prefetcher from trying to leave the current segment.

**Data Pages** are contiguous blocks of 16 KByte each. They are referenced via the data page pointers DPP3 ... 0 and via an explicit data page number for data accesses overriding the standard DPP scheme. Each DPP register can select one of the possible 1024 data pages. The DPP register that is used for the current access is selected via the two upper bits of the 16-bit data address. Subsequent 16-bit data addresses that cross the 16 KByte data page boundaries therefore will use different data page pointers, while the physical locations need not be subsequent within memory.

### 3.6 Protection of the On-Chip Mask ROM

The on-chip mask ROM of the C161CS/JC/JI can be protected against read accesses of both code and data. ROM protection is established during the production process of the device (a ROM mask can be ordered with a ROM protection or without it). No software control is possible, i.e. the ROM protection cannot be disabled or enabled by software.

When a device has been produced with ROM protection active, the ROM contents are protected against unauthorized access by the following measures:

- **No data read accesses** to the internal ROM by any instruction which is executed from any location outside the on-chip mask ROM (including IRAM, XRAM, and external memory).

A program cannot read any data out of the protected ROM from outside.

The read data will be replaced by the default value 009B<sub>H</sub> for any read access to any location.

- **No codes fetches** from the internal ROM by any instruction which is executed from any location outside the on-chip mask ROM (including IRAM, XRAM, and external memory).

A program cannot branch to a location within the protected ROM from outside. This applies to JUMPs as well as to RETURNS, i.e. a called routine within RAM or external memory can never return to the protected ROM.

The fetched code will be replaced by the default value 009BH for any access to any location. This default value will be decoded as the instruction "TRAP #00" which will restart program execution at location 00'0000<sub>H</sub>.

*Note: ROM protection may be used for applications where the complete software fits into the on-chip ROM, or where the on-chip ROM holds an initialization software which is then replaced by an external (e.g.) application software. In the latter case no data (constants, tables, etc.) can be stored within the ROM. The ROM itself should be mapped to segment 1 before branching outside, so an interrupt vector table can be established in external memory.*

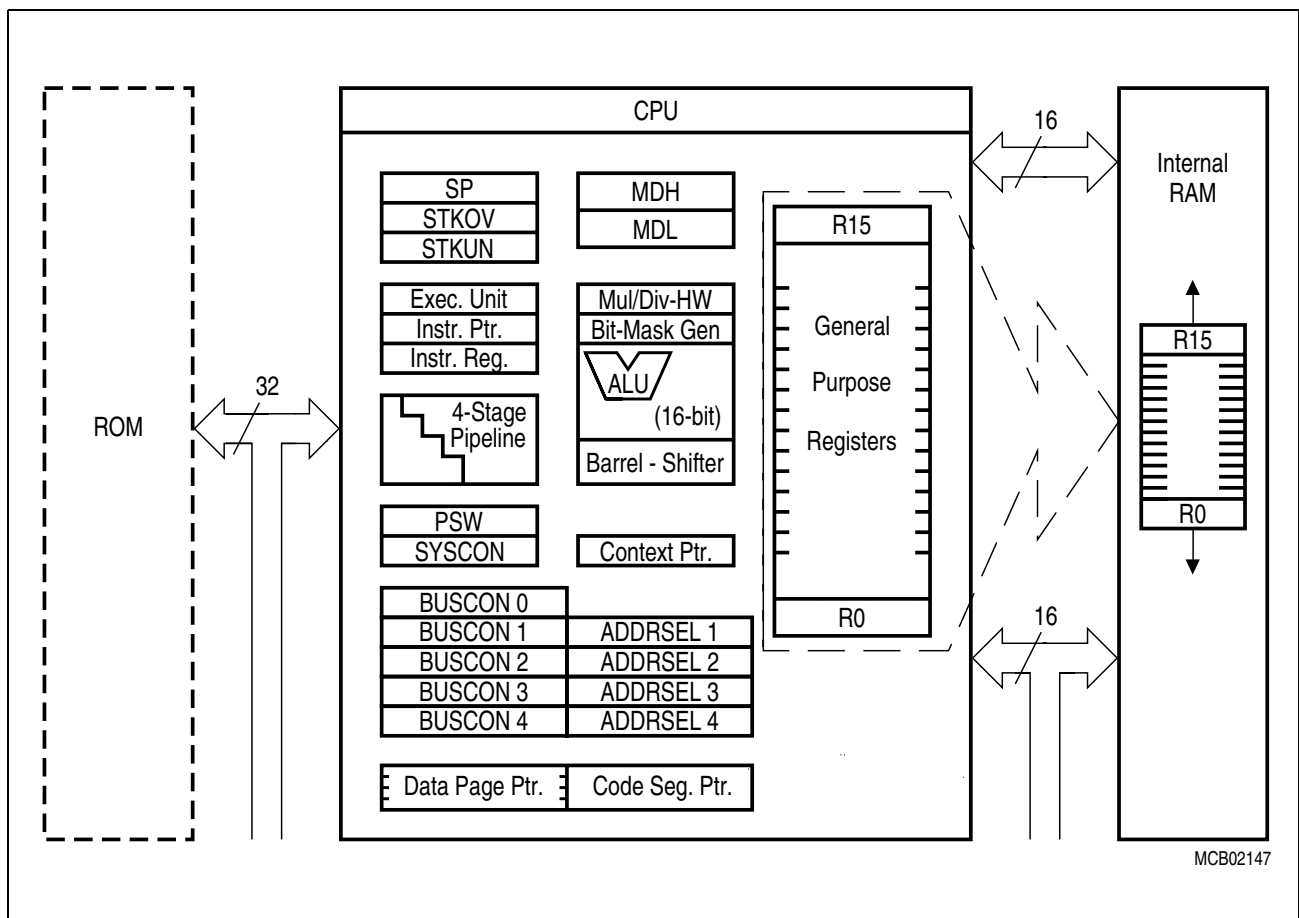
## 4 The Central Processing Unit (CPU)

Basic tasks of the CPU are to fetch and decode instructions, to supply operands for the arithmetic and logic unit (ALU), to perform operations on these operands in the ALU, and to store the previously calculated results. As the CPU is the main engine of the C161CS/JC/JI controller, it is also affected by certain actions of the peripheral subsystem.

Since a four stage pipeline is implemented in the C161CS/JC/JI, up to four instructions can be processed in parallel. Most instructions of the C161CS/JC/JI are executed in one machine cycle (2 CPU clock periods) due to this parallelism.

This chapter describes how the pipeline works for sequential and branch instructions in general, and which hardware provisions have been made to speed the execution of jump instructions in particular. The general instruction timing is described including standard and exceptional timing.

While internal memory accesses are normally performed by the CPU itself, external peripheral or memory accesses are performed by a particular on-chip External Bus Controller (EBC), which is automatically invoked by the CPU whenever a code or data address refers to the external address space.



**Figure 4-1 CPU Block Diagram**

**The Central Processing Unit (CPU)**

If possible, the CPU continues operating while an external memory access is in progress. If external data are required but are not yet available, or if a new external memory access is requested by the CPU, before a previous access has been completed, the CPU will be held by the EBC until the request can be satisfied. The EBC is described in a dedicated chapter.

The on-chip peripheral units of the C161CS/JC/JI work nearly independent of the CPU with a separate clock generator. Data and control information is interchanged between the CPU and these peripherals via Special Function Registers (SFRs).

Whenever peripherals need a non-deterministic CPU action, an on-chip Interrupt Controller compares all pending peripheral service requests against each other and prioritizes one of them. If the priority of the current CPU operation is lower than the priority of the selected peripheral request, an interrupt will occur.

Basically, there are two types of interrupt processing:

- **Standard interrupt processing** forces the CPU to save the current program status and the return address on the stack before branching to the interrupt vector jump table.
- **PEC interrupt processing** steals just one machine cycle from the current CPU activity to perform a single data transfer via the on-chip Peripheral Event Controller (PEC).

System errors detected during program execution (so-called hardware traps) or an external non-maskable interrupt are also processed as standard interrupts with a very high priority.

In contrast to other on-chip peripherals, there is a closer conjunction between the watchdog timer and the CPU. If enabled, the watchdog timer expects to be serviced by the CPU within a programmable period of time, otherwise it will reset the chip. Thus, the watchdog timer is able to prevent the CPU from going totally astray when executing erroneous code. After reset, the watchdog timer starts counting automatically, but it can be disabled via software, if desired.

Beside its normal operation there are the following particular CPU states:

- **Reset state:** Any reset (hardware, software, watchdog) forces the CPU into a predefined active state.
- **IDLE state:** The clock signal to the CPU itself is switched off, while the clocks for the on-chip peripherals keep running.
- **SLEEP state:** All of the on-chip clocks are switched off (RTC clock selectable), wake-up via external interrupts or RTC.
- **POWER DOWN state:** All of the on-chip clocks are switched off (RTC clock selectable), all inputs are disregarded.

A transition into an active CPU state is forced by an interrupt (if being in IDLE or SLEEP) or by a reset (if being in POWER DOWN mode).

The IDLE, SLEEP, POWER DOWN, and RESET states can be entered by particular C161CS/JC/JI system control instructions.



**The Central Processing Unit (CPU)**

A set of Special Function Registers is dedicated to the functions of the CPU core:

- General System Configuration: **SYSCON (RP0H)**
- CPU Status Indication and Control: **PSW**
- Code Access Control: **IP, CSP**
- Data Paging Control: **DPP0, DPP1, DPP2, DPP3**
- GPRs Access Control: **CP**
- System Stack Access Control: **SP, STKUN, STKOV**
- Multiply and Divide Support: **MDL, MDH, MDC**
- ALU Constants Support: **ZEROS, ONES**

**4.1 Instruction Pipelining**

The instruction pipeline of the C161CS/JC/JI partitions instruction processing into four stages of which each one has its individual task:

**1<sup>st</sup> → FETCH:** In this stage the instruction selected by the Instruction Pointer (IP) and the Code Segment Pointer (CSP) is fetched from either the internal ROM, internal RAM, or external memory.

**2<sup>nd</sup> → DECODE:** In this stage the instructions are decoded and, if required, the operand addresses are calculated and the respective operands are fetched. For all instructions, which implicitly access the system stack, the SP register is either decremented or incremented, as specified. For branch instructions the Instruction Pointer and the Code Segment Pointer are updated with the desired branch target address (provided that the branch is taken).

**3<sup>rd</sup> → EXECUTE:** In this stage an operation is performed on the previously fetched operands in the ALU. Additionally, the condition flags in the PSW register are updated as specified by the instruction. All explicit writes to the SFR memory space and all auto-increment or auto-decrement writes to GPRs used as indirect address pointers are performed during the execute stage of an instruction, too.

**4<sup>th</sup> → WRITE BACK:** In this stage all external operands and the remaining operands within the internal RAM space are written back.

A particularity of the C161CS/JC/JI are the so-called injected instructions. These injected instructions are generated internally by the machine to provide the time needed to process instructions, which cannot be processed within one machine cycle. They are automatically injected into the decode stage of the pipeline, and then they pass through the remaining stages like every standard instruction. Program interrupts are performed by means of injected instructions, too. Although these internally injected instructions will not be noticed in reality, they are introduced here to ease the explanation of the pipeline in the following.

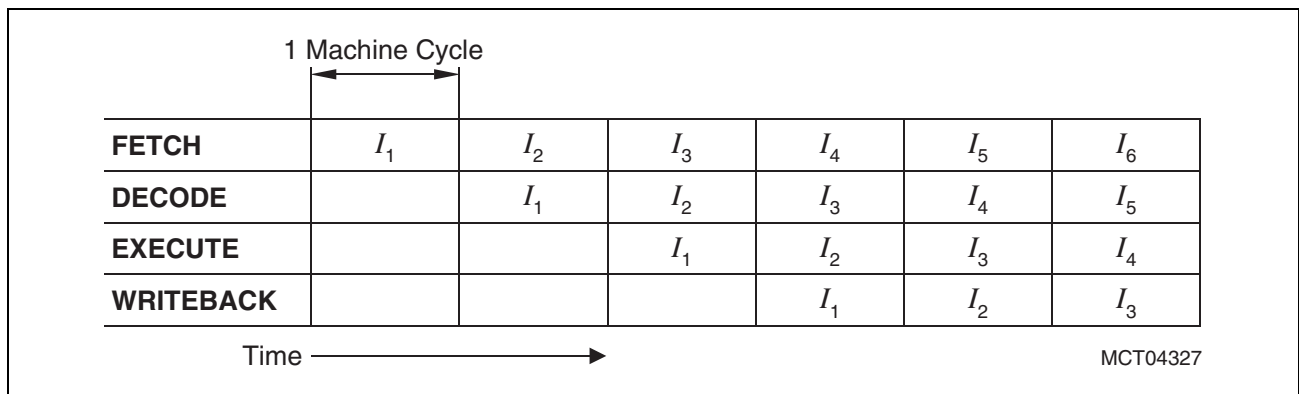


The Central Processing Unit (CPU)

**Sequential Instruction Processing**

Each single instruction has to pass through each of the four pipeline stages regardless of whether all possible stage operations are really performed or not. Since passing through one pipeline stage takes at least one machine cycle, any isolated instruction takes at least four machine cycles to be completed. Pipelining, however, allows parallel (i.e. simultaneous) processing of up to four instructions. Thus, most of the instructions seem to be processed during one machine cycle as soon as the pipeline has been filled once after reset (see [Figure 4-2](#)).

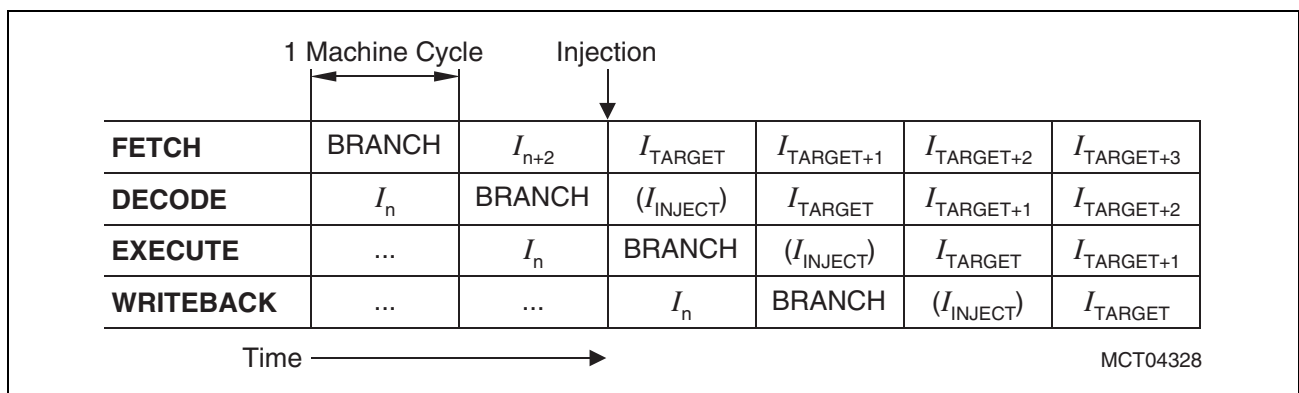
Instruction pipelining increases the average instruction throughput considered over a certain period of time. In the following, any execution time specification of an instruction always refers to the average execution time due to pipelined parallel instruction processing.



**Figure 4-2 Sequential Instruction Pipelining**

**Standard Branch Instruction Processing**

Instruction pipelining helps to speed sequential program processing. In the case that a branch is taken, the instruction which has already been fetched providently is mostly not the instruction which must be decoded next. Thus, at least one additional machine cycle is normally required to fetch the branch target instruction. This extra machine cycle is provided by means of an injected instruction (see [Figure 4-3](#)).



**Figure 4-3 Standard Branch Instruction Pipelining**

**The Central Processing Unit (CPU)**

If a conditional branch is not taken, there is no deviation from the sequential program flow, and thus no extra time is required. In this case the instruction after the branch instruction will enter the decode stage of the pipeline at the beginning of the next machine cycle after decode of the conditional branch instruction.

**Cache Jump Instruction Processing**

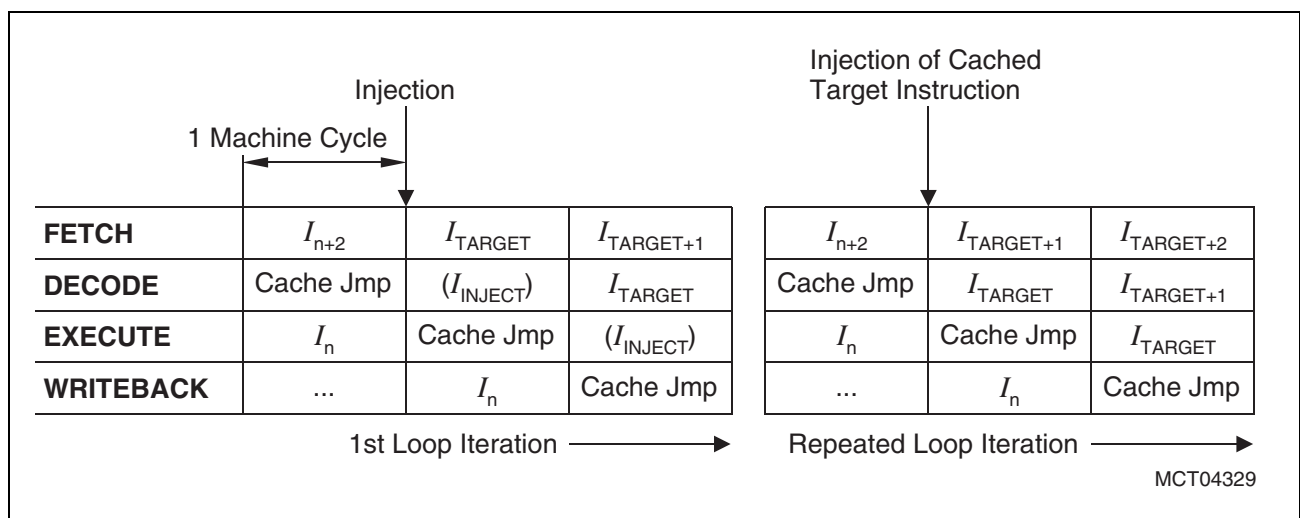
The C161CS/JC/JI incorporates a jump cache to optimize conditional jumps, which are processed repeatedly within a loop. Whenever a jump on cache is taken, the extra time to fetch the branch target instruction can be saved and thus the corresponding cache jump instruction in most cases takes only one machine cycle.

This performance is achieved by the following mechanism:

Whenever a cache jump instruction passes through the decode stage of the pipeline for the first time (and provided that the jump condition is met), the jump target instruction is fetched as usual, causing a time delay of one machine cycle. In contrast to standard branch instructions, however, the target instruction of a cache jump instruction (JMPA, JMPR, JB, JBC, JNB, JNBS) is additionally stored in the cache after having been fetched.

After each repeatedly following execution of the same cache jump instruction, the jump target instruction is not fetched from program memory but taken from the cache and immediately injected into the decode stage of the pipeline (see **Figure 4-4**).

A time saving jump on cache is always taken after the second and any further occurrence of the same cache jump instruction, unless an instruction which, has the fundamental capability of changing the CSP register contents (JMPS, CALLS, RETS, TRAP, RETI), or any standard interrupt has been processed during the period of time between two following occurrences of the same cache jump instruction.



**Figure 4-4 Cache Jump Instruction Pipelining**

## 4.2 Particular Pipeline Effects

Since up to four different instructions are processed simultaneously, additional hardware has been spent in the C161CS/JC/JI to consider all causal dependencies which may exist on instructions in different pipeline stages without a loss of performance. This extra hardware (i.e. for 'forwarding' operand read and write values) resolves most of the possible conflicts (e.g. multiple usage of buses) in a time optimized way and thus avoids that the pipeline becomes noticeable for the user in most cases. However, there are some very rare cases, where the circumstance that the C161CS/JC/JI is a pipelined machine requires attention by the programmer. In these cases the delays caused by pipeline conflicts can be used for other instructions in order to optimize performance.

### Context Pointer Updating

An instruction, which calculates a physical GPR operand address via the CP register, is mostly not capable of using a new CP value, which is to be updated by an immediately preceding instruction. Thus, to make sure that the new CP value is used, at least one instruction must be inserted between a CP-changing and a subsequent GPR-using instruction, as shown in the following example:

```
In      :SCXT CP,#0FC00h ;select a new context
In + 1: ...           ;must not be an instruction using a GPR
In + 2:MOV R0,#dataX  ;write to GPR 0 in the new context
```

### Data Page Pointer Updating

An instruction, which calculates a physical operand address via a particular DPP<sub>n</sub> (n = 0 to 3) register, is mostly not capable of using a new DPP<sub>n</sub> register value, which is to be updated by an immediately preceding instruction. Thus, to make sure that the new DPP<sub>n</sub> register value is used, at least one instruction must be inserted between a DPP<sub>n</sub>-changing instruction and a subsequent instruction which implicitly uses DPP<sub>n</sub> via a long or indirect addressing mode, as shown in the following example:

```
In      :MOV DPP0,#4      ;select data page 4 via DPP0
In + 1: ...           ;must not be an instruction using DPP0
In + 2:MOV DPP0:0000H,R1;move contents of R1 to address location
                          ;01'0000H (in data page 4) supposed segment.
                          ;is enabled
```

### Explicit Stack Pointer Updating

None of the RET, RETI, RETS, RETP or POP instructions is capable of correctly using a new SP register value, which is to be updated by an immediately preceding instruction. Thus, in order to use the new SP register value without erroneously performed stack accesses, at least one instruction must be inserted between an explicitly SP-writing and any subsequent of the just mentioned implicitly SP-using instructions, as shown in the following example:

```
In      :MOV SP,#0FA40H ;select a new top of stack
In + 1:...           ;must not be an instruction popping operands
                          ;from the system stack
In + 2:POP R0       ;pop word value from new top of stack
                          ;into R0
```

*Note: Conflicts with instructions writing to the stack (PUSH, CALL, SCXT) are solved internally by the CPU logic.*

### Controlling Interrupts

Software modifications (implicit or explicit) of the PSW are done in the execute phase of the respective instructions. In order to maintain fast interrupt responses, however, the current interrupt prioritization round does not consider these changes, i.e. an interrupt request may be acknowledged after the instruction that disables interrupts via IEN or ILVL or after the following instructions. Timecritical instruction sequences therefore should not begin directly after the instruction disabling interrupts, as shown in the following examples:

```
INTERRUPTS_OFF:
BCLR IEN           ;globally disable interrupts
<Instr non-crit>  ;non-critical instruction
<Instr 1st-crit> ;begin of uninterruptable critical sequence
...
<Instr last-crit> ;end of uninterruptable critical sequence
INTERRUPTS_ON:
BSET IEN           ;globally re-enable interrupts
CRITICAL_SEQUENCE:
ATOMIC #3         ;immediately block interrupts
BCLR IEN           ;globally disable interrupts
...               ;here is the uninterruptable sequence
BSET IEN           ;globally re-enable interrupts
```

*Note: The described delay of 1 instruction also applies for enabling the interrupts system i.e. no interrupt requests are acknowledged until the instruction following the enabling instruction.*

## The Central Processing Unit (CPU)

### External Memory Access Sequences

The effect described here will only become noticeable, when watching the external memory access sequences on the external bus (e.g. by means of a Logic Analyzer). Different pipeline stages can simultaneously put a request on the External Bus Controller (EBC). The sequence of instructions processed by the CPU may diverge from the sequence of the corresponding external memory accesses performed by the EBC, due to the predefined priority of external memory accesses:

1<sup>st</sup>      Write Data  
2<sup>nd</sup>      Fetch Code  
3<sup>rd</sup>      Read Data.

### Initialization of Port Pins

Modifications of the direction of port pins (input or output) become effective only after the instruction following the modifying instruction. As bit instructions (BSET, BCLR) use internal read-modify-write sequences accessing the whole port, instructions modifying the port direction should be followed by an instruction that does not access the same port (see example below).

```
PORT_INIT_WRONG:
BSET   DP3.13           ;change direction of P3.13 to output
BSET   P3.9             ;P3.13 is still input,
                       ;rd-mod-wr reads pin P3.13

PORT_INIT_RIGHT:
BSET   DP3.13           ;change direction of P3.13 to output
NOP    ;any instruction not accessing port 3
BSET   P3.9             ;P3.13 is now output,
                       ;rd-mod-wr reads P3.13's output latch
```

*Note: Special attention must be paid to interrupt service routines that modify the same port as the software they have interrupted.*

### Changing the System Configuration

The instruction following an instruction that changes the system configuration via register SYSCON (e.g. the mapping of the internal ROM, segmentation, stack size) cannot use the new resources (e.g. ROM or stack). In these cases an instruction that does not access these resources should be inserted. Code accesses to the new ROM area are only possible after an absolute branch to this area.

*Note: As a rule, instructions that change ROM mapping should be executed from internal RAM or external memory.*

---

**The Central Processing Unit (CPU)****BUSCON/ADDRSEL**

The instruction following an instruction that changes the properties of an external address area cannot access operands within the new area. In these cases an instruction that does not access this address area should be inserted. Code accesses to the new address area should be made after an absolute branch to this area.

*Note: As a rule, instructions that change external bus properties should not be executed from the respective external memory area.*

**Timing**

Instruction pipelining reduces the average instruction processing time in a wide scale (from four to one machine cycles, mostly). However, there are some rare cases, where a particular pipeline situation causes the processing time for a single instruction to be extended either by a half or by one machine cycle. Although this additional time represents only a tiny part of the total program execution time, it might be of interest to avoid these pipeline-caused time delays in time critical program modules.

Besides a general execution time description, [Section 4.3](#) provides some hints on how to optimize time-critical program parts with regard to such pipeline-caused timing particularities.

### 4.3 Bit-Handling and Bit-Protection

The C161CS/JC/JI provides several mechanisms to manipulate bits. These mechanisms either manipulate software flags within the internal RAM, control on-chip peripherals via control bits in their respective SFRs or control IO functions via port pins. The instructions BSET, BCLR, BAND, BOR, BXOR, BMOV, BMOVN explicitly set or clear specific bits. The instructions BFLDL and BFLDH allow to manipulate up to 8 bits of a specific byte at one time. The instructions JBC and JNBS implicitly clear or set the specified bit when the jump is taken. The instructions JB and JNB (also conditional jump instructions that refer to flags) evaluate the specified bit to determine if the jump is to be taken.

*Note: Bit operations on undefined bit locations will always read a bit value of '0', while the write access will not effect the respective bit location.*

All instructions that manipulate single bits or bit groups internally use a read-modify-write sequence that accesses the whole word, which contains the specified bit(s).

This method has several consequences:

- Bits can only be modified within the internal address areas, i.e. internal RAM and SFRs. External locations cannot be used with bit instructions.

The upper 256 Bytes of the SFR area, the ESFR area and the internal RAM are bit-addressable (see [Chapter 3](#)), i.e. those register bits located within the respective sections can be directly manipulated using bit instructions. The other SFRs must be accessed byte/word wise.

*Note: All GPRs are bit-addressable independent of the allocation of the register bank via the context pointer CP. Even GPRs which are allocated to not bit-addressable RAM locations provide this feature.*

- The read-modify-write approach may be critical with hardware-effected bits. In these cases the hardware may change specific bits while the read-modify-write operation is in progress, where the writeback would overwrite the new bit value generated by the hardware. The solution is either the implemented hardware protection (see below) or realized through special programming (see [Chapter 4.2](#)).

**Protected bits** are not changed during the read-modify-write sequence, i.e. when hardware sets e.g. an interrupt request flag between the read and the write of the read-modify-write sequence. The hardware protection logic guarantees that only the intended bit(s) is/are effected by the write-back operation.

*Note: If a conflict occurs between a bit manipulation generated by hardware and an intended software access the software access has priority and determines the final value of the respective bit.*

A summary of the protected bits implemented in the C161CS/JC/JI can be found at the end of [Chapter 2](#).



#### 4.4 Instruction State Times

Basically, the time to execute an instruction depends on where the instruction is fetched from, and where possible operands are read from or written to. The fastest processing mode of the C161CS/JC/JI is to execute a program fetched from the internal code memory. In that case most of the instructions can be processed within just one machine cycle, which is also the general minimum execution time.

All external memory accesses are performed by the C161CS/JC/JI's on-chip External Bus Controller (EBC), which works in parallel with the CPU.

This section summarizes the execution times in a very condensed way. A detailed description of the execution times for the various instructions and the specific exceptions can be found in the “**C166 Family Instruction Set Manual**”.

**Table 4-1** shows the minimum execution times required to process a C161CS/JC/JI instruction fetched from the internal code memory, the internal RAM or from external memory. These execution times apply to most of the C161CS/JC/JI instructions - except some of the branches, the multiplication, the division and a special move instruction. In case of internal ROM program execution there is no execution time dependency on the instruction length except for some special branch situations. The numbers in the table are in units of CPU clock cycles and assume no waitstates.

**Table 4-1 Minimum Execution Times**

Memory Area	Instruction Fetch		Word Operand Access	
	Word Instruction	Doubleword Instruction	Read from	Write to
Internal code memory	2	2	2	–
Internal RAM	6	8	0/1	0
16-bit Demux Bus	2	4	2	2
16-bit Mux Bus	3	6	3	3
8-bit Demux Bus	4	8	4	4
8-bit Mux Bus	6	12	6	6

Execution from the internal RAM provides flexibility in terms of loadable and modifiable code on the account of execution time.

Execution from external memory strongly depends on the selected bus mode and the programming of the bus cycles (waitstates).



---

**The Central Processing Unit (CPU)**

The operand and instruction accesses listed below can extend the execution time of an instruction:

- Internal code memory operand reads (same for byte and word operand reads)
- Internal RAM operand reads via indirect addressing modes
- Internal SFR operand reads immediately after writing
- External operand reads
- External operand writes
- Jumps to non-aligned double word instructions in the internal ROM space
- Testing Branch Conditions immediately after PSW writes

#### **4.5 CPU Special Function Registers**

The core CPU requires a set of Special Function Registers (SFRs) to maintain the system state information, to supply the ALU with register-addressable constants and to control system and bus configuration, multiply and divide ALU operations, code memory segmentation, data memory paging, and accesses to the General Purpose Registers and the System Stack.

The access mechanism for these SFRs in the CPU core is identical to the access mechanism for any other SFR. Since all SFRs can simply be controlled by means of any instruction, which is capable of addressing the SFR memory space, a lot of flexibility has been gained, without the need to create a set of system-specific instructions.

Note, however, that there are user access restrictions for some of the CPU core SFRs to ensure proper processor operations. The instruction pointer IP and code segment pointer CSP cannot be accessed directly at all. They can only be changed indirectly via branch instructions.

The PSW, SP, and MDC registers can be modified not only explicitly by the programmer, but also implicitly by the CPU during normal instruction processing. Note that any explicit write request (via software) to an SFR supersedes a simultaneous modification by hardware of the same register.

*Note: Any write operation to a single byte of an SFR clears the non-addressed complementary byte within the specified SFR.*

*Non-implemented (reserved) SFR bits cannot be modified, and will always supply a read value of '0'.*

**The Central Processing Unit (CPU)**

**The System Configuration Register SYSCON**

This bit-addressable register provides general system configuration and control functions. The reset value for register SYSCON depends on the state of the PORT0 pins during reset (see hardware effectable bits).

**SYSCON**

**System Control Register**

**SFR (FF12<sub>H</sub>/89<sub>H</sub>)**

**Reset Value: 0XX0<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STKSZ		ROM S1	SGT DIS	ROM EN	BYT DIS	CLK EN	WR CFG	CS CFG	-	OW D DIS	BD RST EN	XP EN	VISI BLE	XPER-SHARE	
rw		rw	rw	rwh	rwh	rw	rwh	rw	-	rwh	rw	rw	rw	rw	

Bit	Function
XPER-SHARE	<b>XBUS Peripheral Share Mode Control</b> 0: External accesses to XBUS peripherals are disabled. 1: XBUS peripherals are accessible via the external bus during hold mode.
VISIBLE	<b>Visible Mode Control</b> 0: Accesses to XBUS peripherals are done internally. 1: XBUS peripheral accesses are made visible on the external pins.
XPEN	<b>XBUS Peripheral Enable Bit</b> 0: Accesses to the on-chip X-Peripherals and their functions are disabled. 1: The on-chip X-Peripherals are enabled and can be accessed.
BDRSTEN	<b>Bidirectional Reset Enable Bit</b> 0: Pin $\overline{RSTIN}$ is an input only. 1: Pin $\overline{RSTIN}$ is pulled low during the internal reset sequence after any reset.
OWDDIS	<b>Oscillator Watchdog Disable Bit</b> (Depending on reset configuration) 0: The on-chip oscillator watchdog is enabled and active. 1: The on-chip oscillator watchdog is disabled and the CPU clock is always fed from the oscillator input.
CSCFG	<b>Chip Select Configuration Control</b> (Cleared after reset) 0: Latched $\overline{CS}$ mode. The $\overline{CS}$ signals are latched internally and driven to the (enabled) port pins synchronously. 1: Unlatched $\overline{CS}$ mode. The $\overline{CS}$ signals are directly derived from the address and driven to the (enabled) port pins.
WRCFG	<b>Write Configuration Control</b> (Set according to pin P0H.0 during reset) 0: Pins $\overline{WR}$ and $\overline{BHE}$ retain their normal function. 1: Pin $\overline{WR}$ acts as $\overline{WRL}$ , pin $\overline{BHE}$ acts as $\overline{WRH}$ .

**The Central Processing Unit (CPU)**

<b>Bit</b>	<b>Function</b>
<b>CLKEN</b>	<b>System Clock Output Enable</b> (CLKOUT, cleared after reset) 0: CLKOUT disabled: pin may be used for general purpose IO. 1: CLKOUT enabled: pin outputs the system clock signal.
<b>BYTDIS</b>	<b>Disable/Enable Control for Pin <math>\overline{\text{BHE}}</math></b> (Set according to data bus width) 0: Pin $\overline{\text{BHE}}$ enabled. 1: Pin $\overline{\text{BHE}}$ disabled, pin may be used for general purpose IO.
<b>ROMEN</b>	<b>Internal ROM Enable</b> (Set according to pin $\overline{\text{EA}}$ during reset) 0: Internal program memory disabled, accesses to the ROM area use the external bus. 1: Internal program memory enabled.
<b>SGTDIS</b>	<b>Segmentation Disable/Enable Control</b> (Cleared after reset) 0: Segmentation enabled. (CSP is saved/restored during interrupt entry/exit) 1: Segmentation disabled (Only IP is saved/restored).
<b>ROMS1</b>	<b>Internal ROM Mapping</b> 0: Internal ROM area mapped to segment 0 (00'0000 <sub>H</sub> ... 00'7FFF <sub>H</sub> ). 1: Internal ROM area mapped to segment 1 (01'0000 <sub>H</sub> ... 01'7FFF <sub>H</sub> ).
<b>STKSZ</b>	<b>System Stack Size</b> Selects the size of the system stack (in the internal RAM) from 32 to 512 words.

*Note: Register SYSCON cannot be changed after execution of the EINIT instruction. The function of bits XPER-SHARE, VISIBLE, WRCFG, BYTDIS, ROMEN and ROMS1 is described in more detail in [Chapter 9](#).*

## The Central Processing Unit (CPU)

### System Clock Output Enable (CLKEN)

The system clock output function is enabled by setting bit CLKEN in register SYSCON to '1'. If enabled, port pin P3.15 takes on its alternate function as CLKOUT output pin. The clock output is a 50% duty cycle clock (except for direct drive operation where CLKOUT reflects the clock input signal, and for slowdown operation where CLKOUT mirrors the CPU clock signal) whose frequency equals the CPU operating frequency ( $f_{OUT} = f_{CPU}$ ).

*Note: The output driver of port pin P3.15 is switched on automatically, when the CLKOUT function is enabled. The port direction bit is disregarded. After reset, the clock output function is disabled (CLKEN = '0'). In emulation mode the CLKOUT function is enabled automatically.*

While signal CLKOUT is tightly coupled to the CPU clock signal, the programmable frequency signal FOUT (controlled by register FOCON) may be output on this pin. Please refer to [Chapter 23](#).

### Segmentation Disable/Enable Control (SGTDIS)

Bit SGTDIS allows to select either the segmented or non-segmented memory mode.

**In non-segmented memory mode** (SGTDIS = '1') it is assumed that the code address space is restricted to 64 KBytes (segment 0) and thus 16 bits are sufficient to represent all code addresses. For implicit stack operations (CALL or RET) the CSP register is totally ignored and only the IP is saved to and restored from the stack.

**In segmented memory mode** (SGTDIS = '0') it is assumed that the whole address space is available for instructions. For implicit stack operations (CALL or RET) the CSP register and the IP are saved to and restored from the stack. After reset the segmented memory mode is selected.

*Note: Bit SGTDIS controls if the CSP register is pushed onto the system stack in addition to the IP register before an interrupt service routine is entered, and it is repopped when the interrupt service routine is left again.*

### System Stack Size (STKSZ)

This bitfield defines the size of the physical system stack, which is located in the internal RAM of the C161CS/JC/JI. An area of 32 ... 512 words or all of the internal RAM may be dedicated to the system stack. A so-called "circular stack" mechanism allows to use a bigger virtual stack than this dedicated RAM area.

These techniques as well as the encoding of bitfield STKSZ are described in more detail in [Chapter 24](#).

The Central Processing Unit (CPU)

The Processor Status Word PSW

This bit-addressable register reflects the current state of the microcontroller. Two groups of bits represent the current ALU status, and the current CPU interrupt status. A separate bit (USR0) within register PSW is provided as a general purpose user flag.

PSW

Program Status Word                      SFR (FF10<sub>H</sub>/88<sub>H</sub>)                      Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ILVL			IEN	HLD EN	-	-	-	USR0	MUL IP	E	Z	V	C	N	
rwh			rw	rw	-	-	-	rw	rwh	rwh	rwh	rwh	rwh	rwh	

Bit	Function
N	<b>Negative Result</b> Set, when the result of an ALU operation is negative.
C	<b>Carry Flag</b> Set, when the result of an ALU operation produces a carry bit.
V	<b>Overflow Result</b> Set, when the result of an ALU operation produces an overflow.
Z	<b>Zero Flag</b> Set, when the result of an ALU operation is zero.
E	<b>End of Table Flag</b> Set, when the source operand of an instruction is 8000 <sub>H</sub> or 80 <sub>H</sub> .
MULIP	<b>Multiplication/Division In Progress</b> 0: There is no multiplication/division in progress. 1: A multiplication/division has been interrupted.
USR0	<b>User General Purpose Flag</b> May be used by the application software.
HLDEN, ILVL, IEN	<b>Interrupt and EBC Control Fields</b> Define the response to interrupt requests and enable external bus arbitration. (Described in <a href="#">Chapter 5</a> )

ALU Status (N, C, V, Z, E, MULIP)

The condition flags (N, C, V, Z, E) within the PSW indicate the ALU status due to the last recently performed ALU operation. They are set by most of the instructions due to specific rules, which depend on the ALU or data movement operation performed by an instruction.

**The Central Processing Unit (CPU)**

After execution of an instruction which explicitly updates the PSW register, the condition flags cannot be interpreted as described in the following, because any explicit write to the PSW register supersedes the condition flag values, which are implicitly generated by the CPU. Explicitly reading the PSW register supplies a read value which represents the state of the PSW register after execution of the immediately preceding instruction.

*Note: After reset, all of the ALU status bits are cleared.*

**N-Flag:** For most of the ALU operations, the N-flag is set to '1', if the most significant bit of the result contains a '1', otherwise it is cleared. In the case of integer operations the N-flag can be interpreted as the sign bit of the result (negative: N = '1', positive: N = '0'). Negative numbers are always represented as the 2's complement of the corresponding positive number. The range of signed numbers extends from '-8000<sub>H</sub>' to '+7FFF<sub>H</sub>' for the word data type, or from '-80<sub>H</sub>' to '+7F<sub>H</sub>' for the byte data type. For Boolean bit operations with only one operand the N-flag represents the previous state of the specified bit. For Boolean bit operations with two operands the N-flag represents the logical XORing of the two specified bits.

**C-Flag:** After an addition the C-flag indicates that a carry from the most significant bit of the specified word or byte data type has been generated. After a subtraction or a comparison the C-flag indicates a borrow, which represents the logical negation of a carry for the addition.

This means that the C-flag is set to '1', if **no** carry from the most significant bit of the specified word or byte data type has been generated during a subtraction, which is performed internally by the ALU as a 2's complement addition, and the C-flag is cleared when this complement addition caused a carry.

The C-flag is always cleared for logical, multiply and divide ALU operations, because these operations cannot cause a carry anyhow.

For shift and rotate operations the C-flag represents the value of the bit shifted out last. If a shift count of zero is specified, the C-flag will be cleared. The C-flag is also cleared for a prioritize ALU operation, because a '1' is never shifted out of the MSB during the normalization of an operand.

For Boolean bit operations with only one operand the C-flag is always cleared. For Boolean bit operations with two operands the C-flag represents the logical ANDing of the two specified bits.

**V-Flag:** For addition, subtraction and 2's complementation the V-flag is always set to '1', if the result overflows the maximum range of signed numbers, which are representable by either 16 bits for word operations ('-8000<sub>H</sub>' to '+7FFF<sub>H</sub>'), or by 8 bits for byte operations ('-80<sub>H</sub>' to '+7F<sub>H</sub>'), otherwise the V-flag is cleared. Note that the result of an integer addition, integer subtraction, or 2's complement is not valid, if the V-flag indicates an arithmetic overflow.

For multiplication and division the V-flag is set to '1', if the result cannot be represented in a word data type, otherwise it is cleared. Note that a division by zero will always cause an overflow. In contrast to the result of a division, the result of a multiplication is valid

The Central Processing Unit (CPU)

regardless of whether the V-flag is set to '1' or not.

Since logical ALU operations cannot produce an invalid result, the V-flag is cleared by these operations.

The V-flag is also used as 'Sticky Bit' for rotate right and shift right operations. With only using the C-flag, a rounding error caused by a shift right operation can be estimated up to a quantity of one half of the LSB of the result. In conjunction with the V-flag, the C-flag allows evaluating the rounding error with a finer resolution (see [Table 4-2](#)).

For Boolean bit operations with only one operand the V-flag is always cleared. For Boolean bit operations with two operands the V-flag represents the logical ORing of the two specified bits.

**Table 4-2 Shift Right Rounding Error Evaluation**

C-flag	V-flag	Rounding Error Quantity		
0	0	–	No rounding error	–
0	1	0 <	Rounding error	< 1/2 LSB
1	0		Rounding error	= 1/2 LSB
1	1		Rounding error	> 1/2 LSB

**Z-Flag:** The Z-flag is normally set to '1', if the result of an ALU operation equals zero, otherwise it is cleared.

For the addition and subtraction with carry the Z-flag is only set to '1', if the Z-flag already contains a '1' and the result of the current ALU operation additionally equals zero. This mechanism is provided for the support of multiple precision calculations.

For Boolean bit operations with only one operand the Z-flag represents the logical negation of the previous state of the specified bit. For Boolean bit operations with two operands the Z-flag represents the logical NORing of the two specified bits. For the prioritize ALU operation the Z-flag indicates, if the second operand was zero or not.

**E-Flag:** The E-flag can be altered by instructions, which perform ALU or data movement operations. The E-flag is cleared by those instructions which cannot be reasonably used for table search operations. In all other cases the E-flag is set depending on the value of the source operand to signify whether the end of a search table is reached or not. If the value of the source operand of an instruction equals the lowest negative number, which is representable by the data format of the corresponding instruction ('8000<sub>H</sub>' for the word data type, or '80<sub>H</sub>' for the byte data type), the E-flag is set to '1', otherwise it is cleared.

**MULIP-Flag:** The MULIP-flag will be set to '1' by hardware upon the entrance into an interrupt service routine, when a multiply or divide ALU operation was interrupted before completion. Depending on the state of the MULIP bit, the hardware decides whether a multiplication or division must be continued or not after the end of an interrupt service. The MULIP bit is overwritten with the contents of the stacked MULIP-flag when the return-from-interrupt-instruction (RETI) is executed. This normally means that the MULIP-flag is cleared again after that.



**The Central Processing Unit (CPU)**

*Note: The MULIP flag is a part of the task environment! When the interrupting service routine does not return to the interrupted multiply/divide instruction (i.e. in case of a task scheduler that switches between independent tasks), the MULIP flag must be saved as part of the task environment and must be updated accordingly for the new task before this task is entered.*

**CPU Interrupt Status (IEN, ILVL)**

The Interrupt Enable bit allows to globally enable (IEN = '1') or disable (IEN = '0') interrupts. The four-bit Interrupt Level field (ILVL) specifies the priority of the current CPU activity. The interrupt level is updated by hardware upon entry into an interrupt service routine, but it can also be modified via software to prevent other interrupts from being acknowledged. In case an interrupt level '15' has been assigned to the CPU, it has the highest possible priority, and thus the current CPU operation cannot be interrupted except by hardware traps or external non-maskable interrupts. For details please refer to [Chapter 5](#).

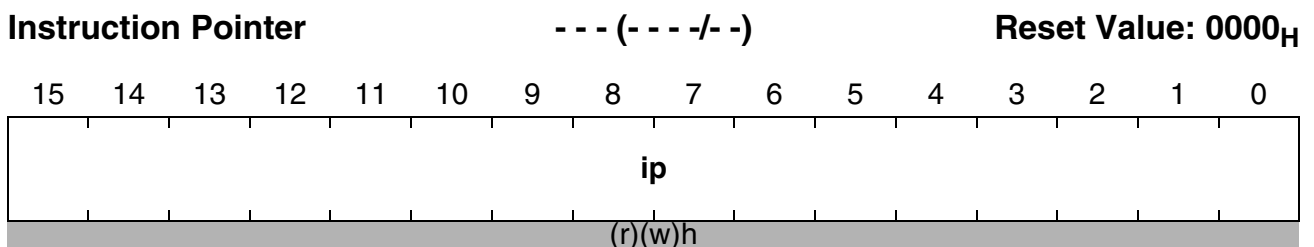
After reset all interrupts are globally disabled, and the lowest priority (ILVL = 0) is assigned to the initial CPU activity.

**The Instruction Pointer IP**

This register determines the 16-bit intra-segment address of the currently fetched instruction within the code segment selected by the CSP register. The IP register is not mapped into the C161CS/JC/JI's address space, and thus it is not directly accessible by the programmer. The IP can, however, be modified indirectly via the stack by means of a return instruction.

The IP register is implicitly updated by the CPU for branch instructions and after instruction fetch operations.

**IP**



Bit	Function
ip	Specifies the intra segment offset, from where the current instruction is to be fetched. IP refers to the current segment <SEGNR>.



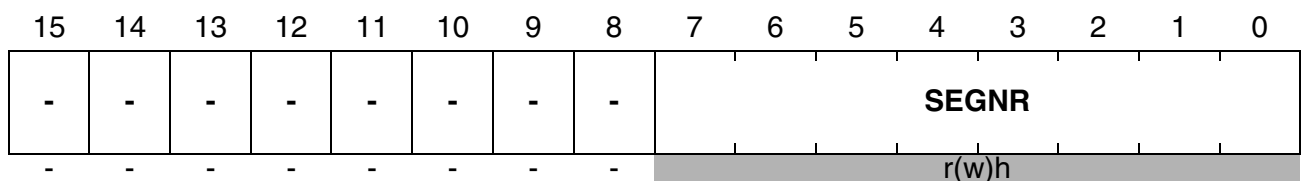
**The Central Processing Unit (CPU)**

**The Code Segment Pointer CSP**

This non-bit addressable register selects the code segment being used at run-time to access instructions. The lower 8 bits of register CSP select one of up to 256 segments of 64 KBytes each, while the upper 8 bits are reserved for future use.

**CSP**

**Code Segment Pointer**                      **SFR (FE08<sub>H</sub>/04<sub>H</sub>)**                      **Reset Value: 0000<sub>H</sub>**



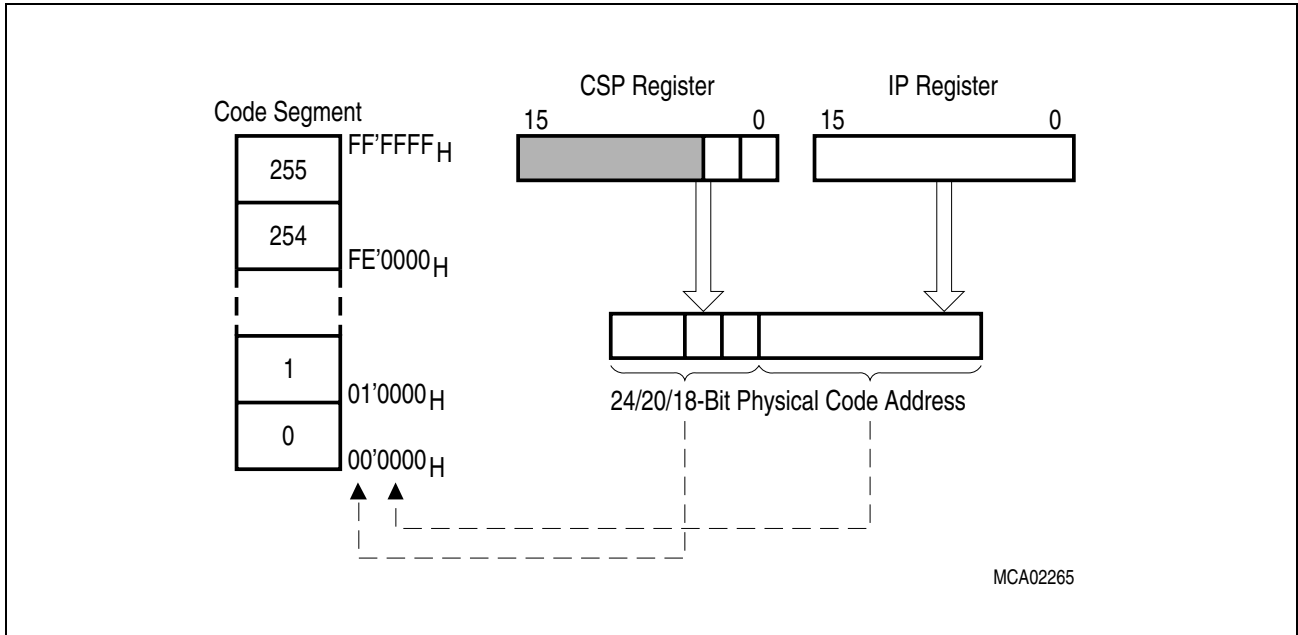
Bit	Function
<b>SEGNR</b>	<b>Segment Number</b> Specifies the code segment, from where the current instruction is to be fetched. SEGNR is ignored, when segmentation is disabled.

Code memory addresses are generated by directly extending the 16-bit contents of the IP register by the contents of the CSP register as shown in [Figure 4-5](#).

In case of the segmented memory mode the selected number of segment address bits (via bitfield SALSEL) of register CSP is output on the respective segment address pins of Port 4 for all external code accesses. For non-segmented memory mode or Single Chip Mode the content of this register is not significant, because all code accesses are automatically restricted to segment 0.

*Note: The CSP register can only be read but not written by data operations. It is, however, modified either directly by means of the JMPS and CALLS instructions, or indirectly via the stack by means of the RETS and RETI instructions. Upon the acceptance of an interrupt or the execution of a software TRAP instruction, the CSP register is automatically set to zero.*

The Central Processing Unit (CPU)



**Figure 4-5 Addressing via the Code Segment Pointer**

*Note: When segmentation is disabled, the IP value is used directly as the 16-bit address.*

**The Central Processing Unit (CPU)**

**The Data Page Pointers DPP0, DPP1, DPP2, DPP3**

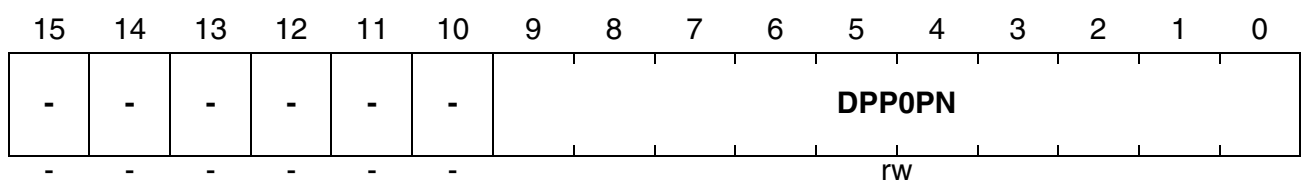
These four non-bit addressable registers select up to four different data pages being active simultaneously at run-time. The lower 10 bits of each DPP register select one of the 1024 possible 16-KByte data pages while the upper 6 bits are reserved for future use. The DPP registers allow to access the entire memory space in pages of 16 KBytes each.

**DPP0**

**Data Page Pointer 0**

**SFR (FE00<sub>H</sub>/00<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

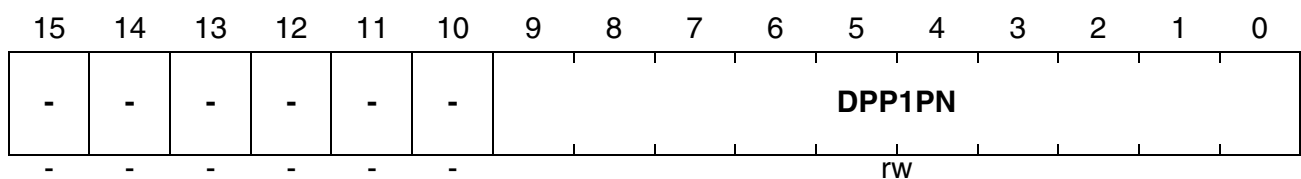


**DPP1**

**Data Page Pointer 1**

**SFR (FE02<sub>H</sub>/01<sub>H</sub>)**

**Reset Value: 0001<sub>H</sub>**

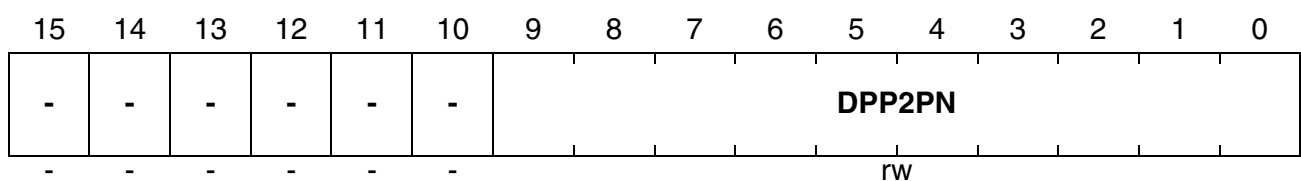


**DPP2**

**Data Page Pointer 2**

**SFR (FE04<sub>H</sub>/02<sub>H</sub>)**

**Reset Value: 0002<sub>H</sub>**

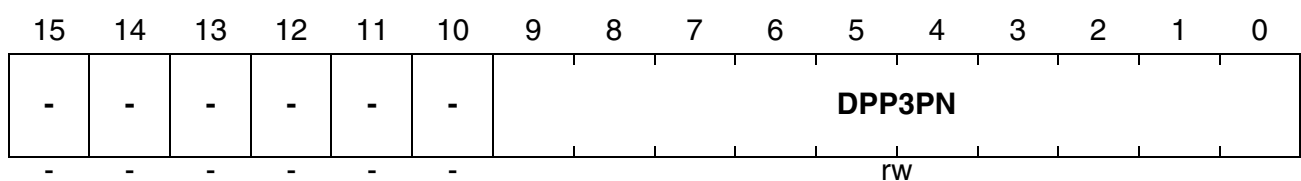


**DPP3**

**Data Page Pointer 3**

**SFR (FE06<sub>H</sub>/03<sub>H</sub>)**

**Reset Value: 0003<sub>H</sub>**



**The Central Processing Unit (CPU)**

<b>Bit</b>	<b>Function</b>
<b>DPPxPN</b>	<b>Data Page Number of DPPx</b> Specifies the data page selected via DPPx. Only the least significant two bits of DPPx are significant, when segmentation is disabled.

The DPP registers are implicitly used, whenever data accesses to any memory location are made via indirect or direct long 16-bit addressing modes (except for override accesses via EXTended instructions and PEC data transfers). After reset, the Data Page Pointers are initialized in a way that all indirect or direct long 16-bit addresses result in identical 18-bit addresses. This allows to access data pages 3 ... 0 within segment 0 as shown in **Figure 4-6**. If the user does not want to use any data paging, no further action is required.

Data paging is performed by concatenating the lower 14 bits of an indirect or direct long 16-bit address with the contents of the DPP register selected by the upper two bits of the 16-bit address. The contents of the selected DPP register specify one of the 1024 possible data pages. This data page base address together with the 14-bit page offset forms the physical 24-bit address (selectable part is driven to the address pins).

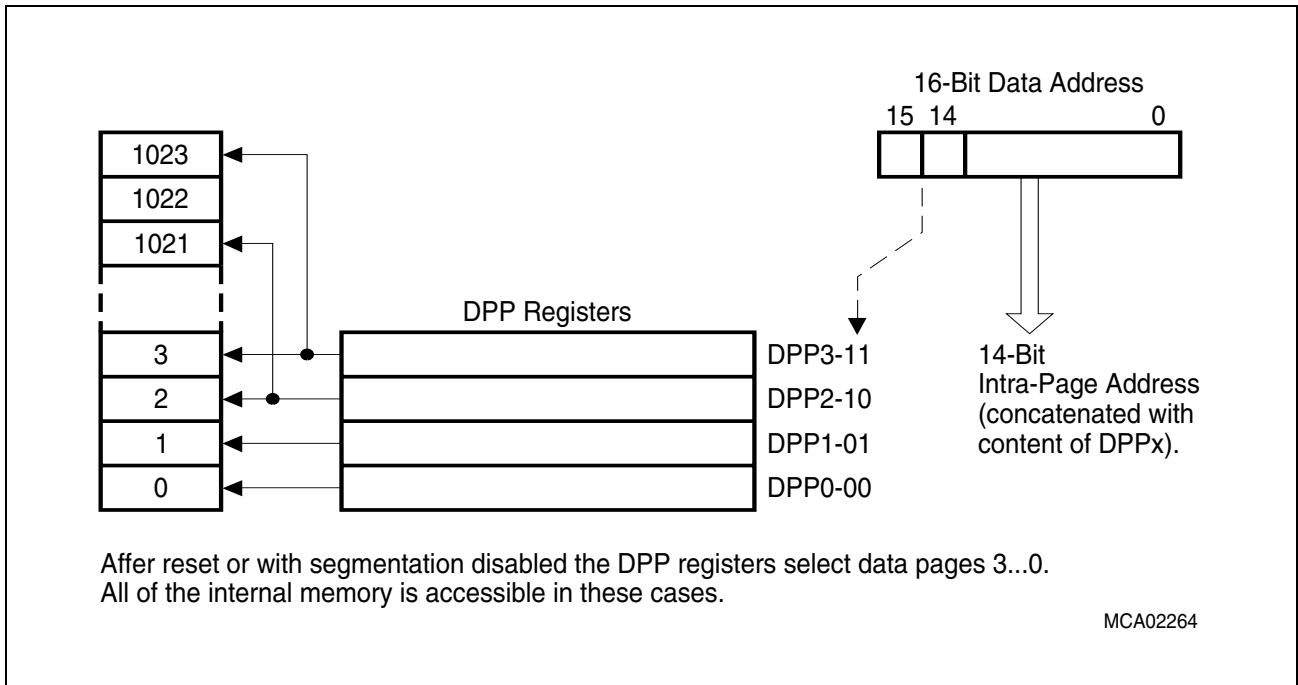
In case of non-segmented memory mode, only the two least significant bits of the implicitly selected DPP register are used to generate the physical address. Thus, extreme care should be taken when changing the content of a DPP register, if a non-segmented memory model is selected, because otherwise unexpected results could occur.

In case of the segmented memory mode the selected number of segment address bits (via bitfield SALSEL) of the respective DPP register is output on the respective segment address pins of Port 4 for all external data accesses.

A DPP register can be updated via any instruction, which is capable of modifying an SFR.

*Note: Due to the internal instruction pipeline, a new DPP value is not yet usable for the operand address calculation of the instruction immediately following the instruction updating the DPP register.*

**The Central Processing Unit (CPU)**



**Figure 4-6 Addressing via the Data Page Pointers**

**The Central Processing Unit (CPU)**

**The Context Pointer CP**

This non-bit addressable register is used to select the current register context. This means that the CP register value determines the address of the first General Purpose Register (GPR) within the current register bank of up to 16 wordwide and/or bytewise GPRs.

**CP**

**Context Pointer**

**SFR (FE10<sub>H</sub>/08<sub>H</sub>)**

**Reset Value: FC00<sub>H</sub>**

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

Bit	Function
cp	<p><b>Modifiable portion of register CP</b></p> <p>Specifies the (word) base address of the current register bank. When writing a value to register CP with bits CP.11 ... CP.9 = '000', bits CP.11 ... CP.10 are set to '11' by hardware, in all other cases all bits of bit field "cp" receive the written value.</p>

*Note: It is the user's responsibility that the physical GPR address specified via CP register plus short GPR address must always be an internal RAM location. If this condition is not met, unexpected results may occur.*

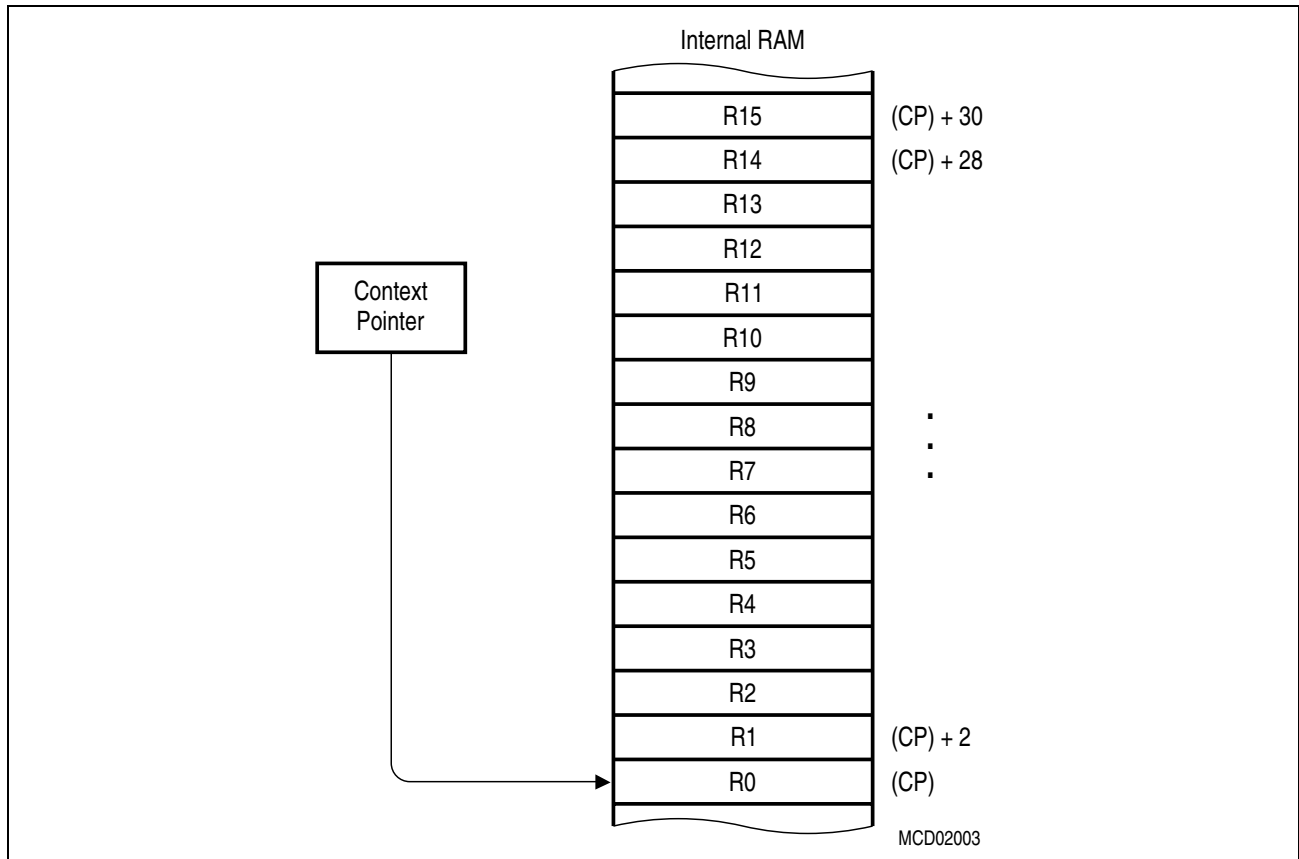
- Do not set CP below the IRAM start address, i.e. 00'FA00<sub>H</sub>/00'F600<sub>H</sub>/00'F200<sub>H</sub> (referring to an IRAM size of 1/2/3 KByte)
- Do not set CP above 00'FDFF<sub>H</sub>
- Be careful using the upper GPRs with CP above 00'FDE0<sub>H</sub>

The CP register can be updated via any instruction which is capable of modifying an SFR.

*Note: Due to the internal instruction pipeline, a new CP value is not yet usable for GPR address calculations of the instruction immediately following the instruction updating the CP register.*

The Switch Context instruction (SCXT) allows to save the content of register CP on the stack and updating it with a new value in just one machine cycle.

The Central Processing Unit (CPU)



**Figure 4-7 Register Bank Selection via Register CP**

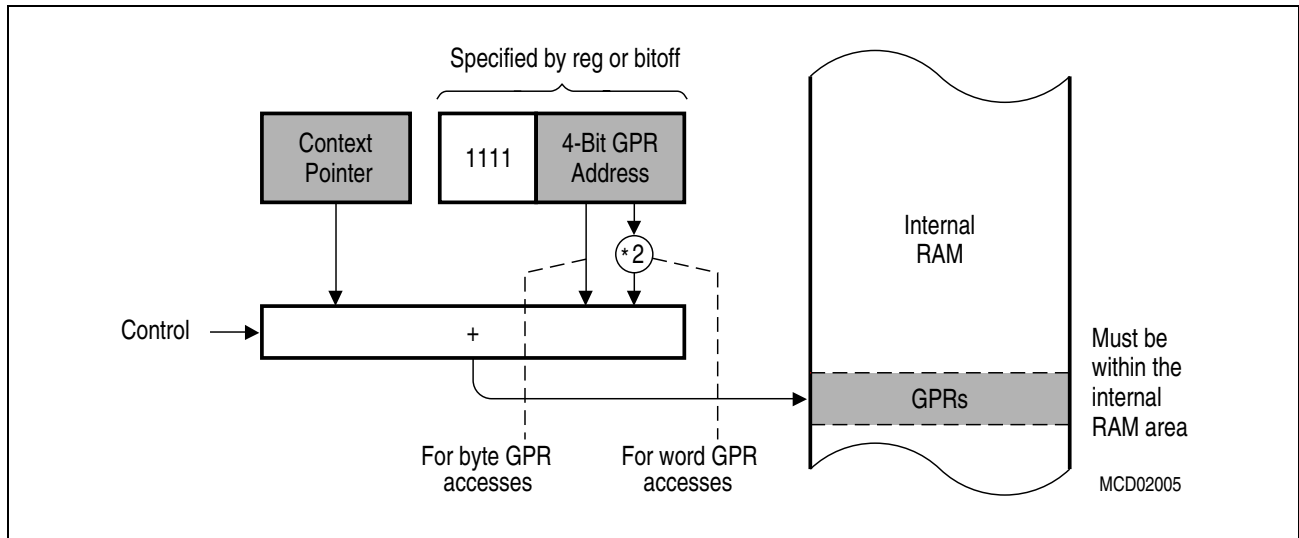
Several addressing modes use register CP implicitly for address calculations. The addressing modes mentioned below are described in [Chapter 26](#).

**Short 4-Bit GPR Addresses** (mnemonic: Rw or Rb) specify an address relative to the memory location specified by the contents of the CP register, i.e. the base of the current register bank.

Depending on whether a relative word (Rw) or byte (Rb) GPR address is specified, the short 4-bit GPR address is either multiplied by two or not before it is added to the content of register CP (see [Figure 4-8](#)). Thus, both byte and word GPR accesses are possible in this way.

GPRs used as indirect address pointers are always accessed wordwise. For some instructions only the first four GPRs can be used as indirect address pointers. These GPRs are specified via short 2-bit GPR addresses. The respective physical address calculation is identical to that for the short 4-bit GPR addresses.

**Short 8-Bit Register Addresses** (mnemonic: reg or bitoff) within a range from F0<sub>H</sub> to FF<sub>H</sub> interpret the four least significant bits as short 4-bit GPR address, while the four most significant bits are ignored. The respective physical GPR address calculation is identical to that for the short 4-bit GPR addresses. For single bit accesses on a GPR, the GPR's word address is calculated as just described, but the position of the bit within the word is specified by a separate additional 4-bit value.



**Figure 4-8 Implicit CP Use by Short GPR Addressing Modes**

### The Stack Pointer SP

This non-bit addressable register is used to point to the top of the internal system stack (TOS). The SP register is pre-decremented whenever data is to be pushed onto the stack, and it is post-incremented whenever data is to be popped from the stack. Thus, the system stack grows from higher toward lower memory locations.

Since the least significant bit of register SP is tied to '0' and bits 15 through 12 are tied to '1' by hardware, the SP register can only contain values from F000<sub>H</sub> to FFFE<sub>H</sub>. This allows to access a physical stack within the internal RAM of the C161CS/JC/JI. A virtual stack (usually bigger) can be realized via software. This mechanism is supported by registers STKOV and STKUN (see respective descriptions below).

The SP register can be updated via any instruction, which is capable of modifying an SFR.

*Note: Due to the internal instruction pipeline, a POP or RETURN instruction must not immediately follow an instruction updating the SP register.*

### SP

**Stack Pointer Register**

**SFR (FE12<sub>H</sub>/09<sub>H</sub>)**

**Reset Value: FC00<sub>H</sub>**

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

Bit	Function
sp	<b>Modifiable portion of register SP</b> Specifies the top of the internal system stack.



**The Central Processing Unit (CPU)**

**The Stack Overflow Pointer STKOV**

This non-bit addressable register is compared against the SP register after each operation, which pushes data onto the system stack (e.g. PUSH and CALL instructions or interrupts) and after each subtraction from the SP register. If the content of the SP register is less than the content of the STKOV register, a stack overflow hardware trap will occur.

Since the least significant bit of register STKOV is tied to '0' and bits 15 through 12 are tied to '1' by hardware, the STKOV register can only contain values from F000<sub>H</sub> to FFFE<sub>H</sub>.

**STKOV**

<b>Stack Overflow Reg.</b>				<b>SFR (FE14<sub>H</sub>/0A<sub>H</sub>)</b>						<b>Reset Value: FA00<sub>H</sub></b>					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1						stkov						0
r	r	r	r						rw						r

Bit	Function
stkov	<b>Modifiable portion of register STKOV</b> Specifies the lower limit of the internal system stack.

The Stack Overflow Trap (entered when (SP) < (STKOV)) may be used in two different ways:

- **Fatal error indication** treats the stack overflow as a system error through the associated trap service routine. Under these circumstances data in the bottom of the stack may have been overwritten by the status information stacked upon servicing the stack overflow trap.
- **Automatic system stack flushing** allows to use the system stack as a 'Stack Cache' for a bigger external user stack. In this case register STKOV should be initialized to a value, which represents the desired lowest Top of Stack address plus 12 according to the selected maximum stack size. This considers the worst case that will occur, when a stack overflow condition is detected just during entry into an interrupt service routine. Then, six additional stack word locations are required to push IP, PSW, and CSP for both the interrupt service routine and the hardware trap service routine.

More details about the stack overflow trap service routine and virtual stack management are given in [Chapter 24](#).

The Central Processing Unit (CPU)

**The Stack Underflow Pointer STKUN**

This non-bit addressable register is compared against the SP register after each operation, which pops data from the system stack (e.g. POP and RET instructions) and after each addition to the SP register. If the content of the SP register is greater than the content of the STKUN register, a stack underflow hardware trap will occur.

Since the least significant bit of register STKUN is tied to '0' and bits 15 through 12 are tied to '1' by hardware, the STKUN register can only contain values from F000<sub>H</sub> to FFFE<sub>H</sub>.

**STKUN**

<b>Stack Underflow Reg.</b>				<b>SFR (FE16<sub>H</sub>/0B<sub>H</sub>)</b>						<b>Reset Value: FC00<sub>H</sub></b>					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1						stkun						0
r	r	r	r						rw						r

Bit	Function
stkun	<b>Modifiable portion of register STKUN</b> Specifies the upper limit of the internal system stack.

The Stack Underflow Trap (entered when (SP) > (STKUN)) may be used in two different ways:

- **Fatal error indication** treats the stack underflow as a system error through the associated trap service routine.
- **Automatic system stack refilling** allows to use the system stack as a 'Stack Cache' for a bigger external user stack. In this case register STKUN should be initialized to a value, which represents the desired highest Bottom of Stack address.

More details about the stack underflow trap service routine and virtual stack management are given in [Chapter 24](#).

**Scope of Stack Limit Control**

The stack limit control realized by the register pair STKOV and STKUN detects cases where the stack pointer SP is moved outside the defined stack area either by ADD or SUB instructions or by PUSH or POP operations (explicit or implicit, i.e. CALL or RET instructions).

This control mechanism is not triggered, i.e. no stack trap is generated, when

- the stack pointer SP is directly updated via MOV instructions
- the limits of the stack area (STKOV, STKUN) are changed, so that SP is outside of the new limits.

The Central Processing Unit (CPU)

**The Multiply/Divide High Register MDH**

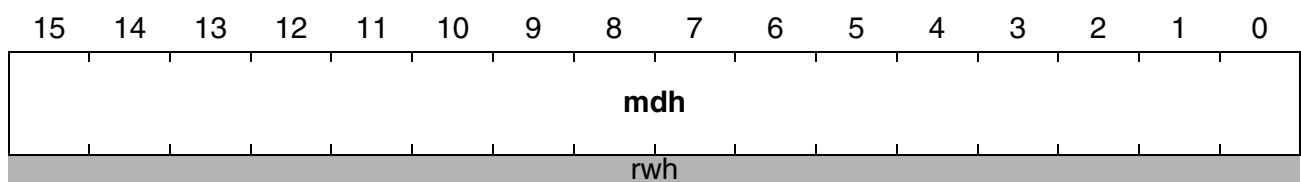
This register is a part of the 32-bit multiply/divide register, which is implicitly used by the CPU, when it performs a multiplication or a division. After a multiplication, this non-bit addressable register represents the high order 16 bits of the 32-bit result. For long divisions, the MDH register must be loaded with the high order 16 bits of the 32-bit dividend before the division is started. After any division, register MDH represents the 16-bit remainder.

**MDH**

**Multiply/Divide High Reg.**

**SFR (FE0C<sub>H</sub>/06<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Bit	Function
mdh	Specifies the high order 16 bits of the 32-bit multiply and divide reg. MD.

Whenever this register is updated via software, the Multiply/Divide Register In Use (MDRIU) flag in the Multiply/Divide Control register (MDC) is set to '1'.

When a multiplication or division is interrupted before its completion and when a new multiply or divide operation is to be performed within the interrupt service routine, register MDH must be saved along with registers MDL and MDC to avoid erroneous results.

A detailed description of how to use the MDH register for programming multiply and divide algorithms can be found in [Chapter 24](#).

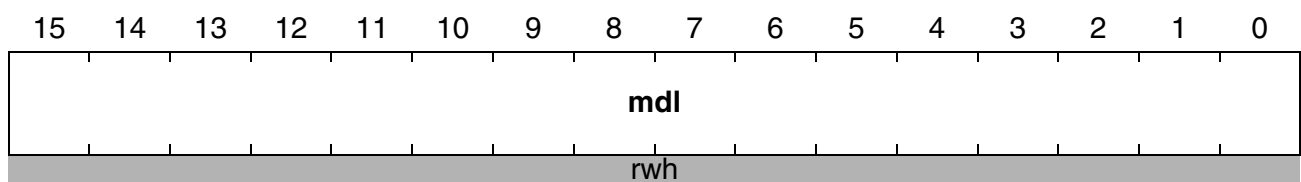
The Central Processing Unit (CPU)

**The Multiply/Divide Low Register MDL**

This register is a part of the 32-bit multiply/divide register, which is implicitly used by the CPU, when it performs a multiplication or a division. After a multiplication, this non-bit addressable register represents the low order 16 bits of the 32-bit result. For long divisions, the MDL register must be loaded with the low order 16 bits of the 32-bit dividend before the division is started. After any division, register MDL represents the 16-bit quotient.

**MDL**

**Multiply/Divide Low Reg.                      SFR (FE0E<sub>H</sub>/07<sub>H</sub>)                      Reset Value: 0000<sub>H</sub>**



Bit	Function
mdl	Specifies the low order 16 bits of the 32-bit multiply and divide reg. MD.

Whenever this register is updated via software, the Multiply/Divide Register In Use (MDRIU) flag in the Multiply/Divide Control register (MDC) is set to '1'. The MDRIU flag is cleared, whenever the MDL register is read via software.

When a multiplication or division is interrupted before its completion and when a new multiply or divide operation is to be performed within the interrupt service routine, register MDL must be saved along with registers MDH and MDC to avoid erroneous results.

A detailed description of how to use the MDL register for programming multiply and divide algorithms can be found in [Chapter 24](#).

The Central Processing Unit (CPU)

The Multiply/Divide Control Register MDC

This bit addressable 16-bit register is implicitly used by the CPU, when it performs a multiplication or a division. It is used to store the required control information for the corresponding multiply or divide operation. Register MDC is updated by hardware during each single cycle of a multiply or divide instruction.

MDC

Multiply/Divide Control Reg.      SFR (FF0E<sub>H</sub>/87<sub>H</sub>)      Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	!!	!!	!!	MDR IU	!!	!!	!!	!!
-	-	-	-	-	-	-	-	r(w)h	r(w)h	r(w)h	r(w)h	r(w)h	r(w)h	r(w)h	r(w)h

Bit	Function
MDRIU	<p><b>Multiply/Divide Register In Use</b></p> <p>0: Cleared, when register MDL is read via software.</p> <p>1: Set when register MDL or MDH is written via software, or when a multiply or divide instruction is executed.</p>
!!	<p><b>Internal Machine Status</b></p> <p>The multiply/divide unit uses these bits to control internal operations. Never modify these bits without saving and restoring register MDC.</p>

When a division or multiplication was interrupted before its completion and the multiply/divide unit is required, the MDC register must first be saved along with registers MDH and MDL (to be able to restart the interrupted operation later), and then it must be cleared prepare it for the new calculation. After completion of the new division or multiplication, the state of the interrupted multiply or divide operation must be restored.

The MDRIU flag is the only portion of the MDC register which might be of interest for the user. The remaining portions of the MDC register are reserved for dedicated use by the hardware, and should never be modified by the user in another way than described above. Otherwise, a correct continuation of an interrupted multiply or divide operation cannot be guaranteed.

A detailed description of how to use the MDC register for programming multiply and divide algorithms can be found in [Chapter 24](#).

**The Central Processing Unit (CPU)**

**The Constant Zeros Register ZEROS**

All bits of this bit-addressable register are fixed to '0' by hardware. This register can be read only. Register ZEROS can be used as a register-addressable constant of all zeros, i.e. for bit manipulation or mask generation. It can be accessed via any instruction, which is capable of addressing an SFR.

**ZEROS**

Zeros Register		SFR (FF1C <sub>H</sub> /8E <sub>H</sub> )		Reset Value: 0000 <sub>H</sub>											
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

**The Constant Ones Register ONES**

All bits of this bit-addressable register are fixed to '1' by hardware. This register can be read only. Register ONES can be used as a register-addressable constant of all ones, i.e. for bit manipulation or mask generation. It can be accessed via any instruction, which is capable of addressing an SFR.

**ONES**

Ones Register		SFR (FF1E <sub>H</sub> /8F <sub>H</sub> )		Reset Value: FFFF <sub>H</sub>											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

## **5 Interrupt and Trap Functions**

The architecture of the C161CS/JC/JI supports several mechanisms for fast and flexible response to service requests that can be generated from various sources internal or external to the microcontroller.

These mechanisms include:

### **Normal Interrupt Processing**

The CPU temporarily suspends the current program execution and branches to an interrupt service routine in order to service an interrupt requesting device. The current program status (IP, PSW, in segmentation mode also CSP) is saved on the internal system stack. A prioritization scheme with 16 priority levels allows the user to specify the order in which multiple interrupt requests are to be handled.

### **Interrupt Processing via the Peripheral Event Controller (PEC)**

A faster alternative to normal software controlled interrupt processing is servicing an interrupt requesting device with the C161CS/JC/JI's integrated Peripheral Event Controller (PEC). Triggered by an interrupt request, the PEC performs a single word or byte data transfer between any two locations in segment 0 (data pages 0 through 3) through one of eight programmable PEC Service Channels. During a PEC transfer the normal program execution of the CPU is halted for just 1 instruction cycle. No internal program status information needs to be saved. The same prioritization scheme is used for PEC service as for normal interrupt processing. PEC transfers share the 2 highest priority levels.

### **Trap Functions**

Trap functions are activated in response to special conditions that occur during the execution of instructions. A trap can also be caused externally by the Non-Maskable Interrupt pin  $\overline{\text{NMI}}$ . Several hardware trap functions are provided for handling erroneous conditions and exceptions that arise during the execution of an instruction. Hardware traps always have highest priority and cause immediate system reaction. The software trap function is invoked by the TRAP instruction, which generates a software interrupt for a specified interrupt vector. For all types of traps the current program status is saved on the system stack.

### **External Interrupt Processing**

Although the C161CS/JC/JI does not provide dedicated interrupt pins, it allows to connect external interrupt sources and provides several mechanisms to react on external events, including standard inputs, non-maskable interrupts and fast external interrupts. These interrupt functions are alternate port functions, except for the non-maskable interrupt and the reset input.

## 5.1 Interrupt System Structure

The C161CS/JC/JI provides 56 separate interrupt nodes that may be assigned to 16 priority levels. In order to support modular and consistent software design techniques, most sources of an interrupt or PEC request are supplied with a separate interrupt control register and interrupt vector. The control register contains the interrupt request flag, the interrupt enable bit, and the interrupt priority of the associated source. Each source request is then activated by one specific event, depending on the selected operating mode of the respective device. For efficient usage of the resources also multi-source interrupt nodes are incorporated. These nodes can be activated by several source requests, e.g. as different kinds of errors in the serial interfaces. However, specific status flags which identify the type of error are implemented in the serial channels' control registers.

The C161CS/JC/JI provides a vectored interrupt system. In this system specific vector locations in the memory space are reserved for the reset, trap, and interrupt service functions. Whenever a request occurs, the CPU branches to the location that is associated with the respective interrupt source. This allows direct identification of the source that caused the request. The only exceptions are the class B hardware traps, which all share the same interrupt vector. The status flags in the Trap Flag Register (TFR) can then be used to determine which exception caused the trap. For the special software TRAP instruction, the vector address is specified by the operand field of the instruction, which is a seven bit trap number.

The reserved vector locations build a jump table in the low end of the C161CS/JC/JI's address space (segment 0). The jump table is made up of the appropriate jump instructions that transfer control to the interrupt or trap service routines, which may be located anywhere within the address space. The entries of the jump table are located at the lowest addresses in code segment 0 of the address space. Each entry occupies 2 words, except for the reset vector and the hardware trap vectors, which occupy 4 or 8 words.

**Table 5-1** lists all sources that are capable of requesting interrupt or PEC service in the C161CS/JC/JI, the associated interrupt vectors, their locations and the associated trap numbers. It also lists the mnemonics of the affected Interrupt Request flags and their corresponding Interrupt Enable flags. The mnemonics are composed of a part that specifies the respective source, followed by a part that specifies their function (IR = Interrupt Request flag, IE = Interrupt Enable flag).

*Note: Each entry of the interrupt vector table provides room for two word instructions or one doubleword instruction. The respective vector location results from multiplying the trap number by 4 (4 Bytes per entry).*

*All interrupt nodes that are currently not used by their associated modules or are not connected to a module in the actual derivative may be used to generate software controlled interrupt requests by setting the respective IR flag.*



**Interrupt and Trap Functions**
**Table 5-1 C161CS/JC/JI Interrupt Notes and Vectors**

Source of Interrupt or PEC Service Request	Request Flag	Enable Flag	Interrupt Vector	Vector Location	Trap Number
CAPCOM Register 0	CC0IR	CC0IE	CC0INT	00'0040 <sub>H</sub>	10 <sub>H</sub> /16 <sub>D</sub>
CAPCOM Register 1	CC1IR	CC1IE	CC1INT	00'0044 <sub>H</sub>	11 <sub>H</sub> /17 <sub>D</sub>
CAPCOM Register 2	CC2IR	CC2IE	CC2INT	00'0048 <sub>H</sub>	12 <sub>H</sub> /18 <sub>D</sub>
CAPCOM Register 3	CC3IR	CC3IE	CC3INT	00'004C <sub>H</sub>	13 <sub>H</sub> /19 <sub>D</sub>
CAPCOM Register 4	CC4IR	CC4IE	CC4INT	00'0050 <sub>H</sub>	14 <sub>H</sub> /20 <sub>D</sub>
CAPCOM Register 5	CC5IR	CC5IE	CC5INT	00'0054 <sub>H</sub>	15 <sub>H</sub> /21 <sub>D</sub>
CAPCOM Register 6	CC6IR	CC6IE	CC6INT	00'0058 <sub>H</sub>	16 <sub>H</sub> /22 <sub>D</sub>
CAPCOM Register 7	CC7IR	CC7IE	CC7INT	00'005C <sub>H</sub>	17 <sub>H</sub> /23 <sub>D</sub>
CAPCOM Register 8	CC8IR	CC8IE	CC8INT	00'0060 <sub>H</sub>	18 <sub>H</sub> /24 <sub>D</sub>
CAPCOM Register 9	CC9IR	CC9IE	CC9INT	00'0064 <sub>H</sub>	19 <sub>H</sub> /25 <sub>D</sub>
CAPCOM Register 10	CC10IR	CC10IE	CC10INT	00'0068 <sub>H</sub>	1A <sub>H</sub> /26 <sub>D</sub>
CAPCOM Register 11	CC11IR	CC11IE	CC11INT	00'006C <sub>H</sub>	1B <sub>H</sub> /27 <sub>D</sub>
CAPCOM Register 12	CC12IR	CC12IE	CC12INT	00'0070 <sub>H</sub>	1C <sub>H</sub> /28 <sub>D</sub>
CAPCOM Register 13	CC13IR	CC13IE	CC13INT	00'0074 <sub>H</sub>	1D <sub>H</sub> /29 <sub>D</sub>
CAPCOM Register 14	CC14IR	CC14IE	CC14INT	00'0078 <sub>H</sub>	1E <sub>H</sub> /30 <sub>D</sub>
CAPCOM Register 15	CC15IR	CC15IE	CC15INT	00'007C <sub>H</sub>	1F <sub>H</sub> /31 <sub>D</sub>
CAPCOM Register 16	CC16IR	CC16IE	CC16INT	00'00C0 <sub>H</sub>	30 <sub>H</sub> /48 <sub>D</sub>
CAPCOM Register 17	CC17IR	CC17IE	CC17INT	00'00C4 <sub>H</sub>	31 <sub>H</sub> /49 <sub>D</sub>
CAPCOM Register 18	CC18IR	CC18IE	CC18INT	00'00C8 <sub>H</sub>	32 <sub>H</sub> /50 <sub>D</sub>
CAPCOM Register 19	CC19IR	CC19IE	CC19INT	00'00CC <sub>H</sub>	33 <sub>H</sub> /51 <sub>D</sub>
CAPCOM Register 20	CC20IR	CC20IE	CC20INT	00'00D0 <sub>H</sub>	34 <sub>H</sub> /52 <sub>D</sub>
CAPCOM Register 21	CC21IR	CC21IE	CC21INT	00'00D4 <sub>H</sub>	35 <sub>H</sub> /53 <sub>D</sub>
CAPCOM Register 22	CC22IR	CC22IE	CC22INT	00'00D8 <sub>H</sub>	36 <sub>H</sub> /54 <sub>D</sub>
CAPCOM Register 23	CC23IR	CC23IE	CC23INT	00'00DC <sub>H</sub>	37 <sub>H</sub> /55 <sub>D</sub>
CAPCOM Register 24	CC24IR	CC24IE	CC24INT	00'00E0 <sub>H</sub>	38 <sub>H</sub> /56 <sub>D</sub>
CAPCOM Register 25	CC25IR	CC25IE	CC25INT	00'00E4 <sub>H</sub>	39 <sub>H</sub> /57 <sub>D</sub>
CAPCOM Register 26	CC26IR	CC26IE	CC26INT	00'00E8 <sub>H</sub>	3A <sub>H</sub> /58 <sub>D</sub>
CAPCOM Register 27	CC27IR	CC27IE	CC27INT	00'00EC <sub>H</sub>	3B <sub>H</sub> /59 <sub>D</sub>
CAPCOM Register 28	CC28IR	CC28IE	CC28INT	00'00F0 <sub>H</sub>	3C <sub>H</sub> /60 <sub>D</sub>
CAPCOM Register 29	CC29IR	CC29IE	CC29INT	00'0110 <sub>H</sub>	44 <sub>H</sub> /68 <sub>D</sub>

**Interrupt and Trap Functions**
**Table 5-1 C161CS/JC/JI Interrupt Notes and Vectors (cont'd)**

Source of Interrupt or PEC Service Request	Request Flag	Enable Flag	Interrupt Vector	Vector Location	Trap Number
CAPCOM Register 30	CC30IR	CC30IE	CC30INT	00'0114 <sub>H</sub>	45 <sub>H</sub> /69 <sub>D</sub>
CAPCOM Register 31	CC31IR	CC31IE	CC31INT	00'0118 <sub>H</sub>	46 <sub>H</sub> /70 <sub>D</sub>
CAPCOM Timer 0	T0IR	T0IE	T0INT	00'0080 <sub>H</sub>	20 <sub>H</sub> /32 <sub>D</sub>
CAPCOM Timer 1	T1IR	T1IE	T1INT	00'0084 <sub>H</sub>	21 <sub>H</sub> /33 <sub>D</sub>
CAPCOM Timer 7	T7IR	T7IE	T7INT	00'00F4 <sub>H</sub>	3D <sub>H</sub> /61 <sub>D</sub>
CAPCOM Timer 8	T8IR	T8IE	T8INT	00'00F8 <sub>H</sub>	3E <sub>H</sub> /62 <sub>D</sub>
GPT1 Timer 2	T2IR	T2IE	T2INT	00'0088 <sub>H</sub>	22 <sub>H</sub> /34 <sub>D</sub>
GPT1 Timer 3	T3IR	T3IE	T3INT	00'008C <sub>H</sub>	23 <sub>H</sub> /35 <sub>D</sub>
GPT1 Timer 4	T4IR	T4IE	T4INT	00'0090 <sub>H</sub>	24 <sub>H</sub> /36 <sub>D</sub>
GPT2 Timer 5	T5IR	T5IE	T5INT	00'0094 <sub>H</sub>	25 <sub>H</sub> /37 <sub>D</sub>
GPT2 Timer 6	T6IR	T6IE	T6INT	00'0098 <sub>H</sub>	26 <sub>H</sub> /38 <sub>D</sub>
GPT2 CAPREL Register	CRIR	CRIE	CRINT	00'009C <sub>H</sub>	27 <sub>H</sub> /39 <sub>D</sub>
A/D Conversion Complete	ADCIR	ADCIE	ADCINT	00'00A0 <sub>H</sub>	28 <sub>H</sub> /40 <sub>D</sub>
A/D Overrun Error	ADEIR	ADEIE	ADEINT	00'00A4 <sub>H</sub>	29 <sub>H</sub> /41 <sub>D</sub>
ASC0 Transmit	S0TIR	S0TIE	S0TINT	00'00A8 <sub>H</sub>	2A <sub>H</sub> /42 <sub>D</sub>
ASC0 Transmit Buffer	S0TBIR	S0TBIE	S0TBINT	00'011C <sub>H</sub>	47 <sub>H</sub> /71 <sub>D</sub>
ASC0 Receive	S0RIR	S0RIE	S0RINT	00'00AC <sub>H</sub>	2B <sub>H</sub> /43 <sub>D</sub>
ASC0 Error	S0EIR	S0EIE	S0EINT	00'00B0 <sub>H</sub>	2C <sub>H</sub> /44 <sub>D</sub>
SSC Transmit	SCTIR	SCTIE	SCTINT	00'00B4 <sub>H</sub>	2D <sub>H</sub> /45 <sub>D</sub>
SSC Receive	SCRIR	SCRIE	SCRINT	00'00B8 <sub>H</sub>	2E <sub>H</sub> /46 <sub>D</sub>
SSC Error	SCEIR	SCEIE	SCEINT	00'00BC <sub>H</sub>	2F <sub>H</sub> /47 <sub>D</sub>
IIC Data Transfer Event	XP0IR	XP0IE	XP0INT	00'0100 <sub>H</sub>	40 <sub>H</sub> /64 <sub>D</sub>
IIC Protocol Event	XP1IR	XP1IE	XP1INT	00'0104 <sub>H</sub>	41 <sub>H</sub> /65 <sub>D</sub>
CAN1 (C161CS/JC)	XP2IR	XP2IE	XP2INT	00'0108 <sub>H</sub>	42 <sub>H</sub> /66 <sub>D</sub>
PLL/RTC (via ISNC)	XP3IR	XP3IE	XP3INT	00'010C <sub>H</sub>	43 <sub>H</sub> /67 <sub>D</sub>
ASC1 Transmit	XP4IR	XP4IE	XP4INT	00'0120 <sub>H</sub>	48 <sub>H</sub> /72 <sub>D</sub>
ASC1 Receive	XP5IR	XP5IE	XP5INT	00'0124 <sub>H</sub>	49 <sub>H</sub> /73 <sub>D</sub>
ASC1 Error	XP6IR	XP6IE	XP6INT	00'0128 <sub>H</sub>	4A <sub>H</sub> /74 <sub>D</sub>
CAN2 (C161CS) SDLM (C161JC/JI)	XP7IR	XP7IE	XP7INT	00'012C <sub>H</sub>	4B <sub>H</sub> /75 <sub>D</sub>

**Interrupt and Trap Functions**

**Table 5-2** lists the vector locations for hardware traps and the corresponding status flags in register TFR. It also lists the priorities of trap service for cases, where more than one trap condition might be detected within the same instruction. After any reset (hardware reset, software reset instruction SRST, or reset by watchdog timer overflow) program execution starts at the reset vector at location 00'0000<sub>H</sub>. Reset conditions have priority over every other system activity and therefore have the highest priority (trap priority III). Software traps may be initiated to any vector location between 00'0000<sub>H</sub> and 00'01FC<sub>H</sub>. A service routine entered via a software TRAP instruction is always executed on the current CPU priority level which is indicated in bit field ILVL in register PSW. This means that routines entered via the software TRAP instruction can be interrupted by all hardware traps or higher level interrupt requests.

**Table 5-2 Hardware Trap Summary**

<b>Exception Condition</b>	<b>Trap Flag</b>	<b>Trap Vector</b>	<b>Vector Location</b>	<b>Trap Number</b>	<b>Trap Prio</b>
<b>Reset Functions</b>	–				
Hardware Reset		RESET	00'0000 <sub>H</sub>	00 <sub>H</sub>	III
Software Reset		RESET	00'0000 <sub>H</sub>	00 <sub>H</sub>	III
Watchdog Timer Overflow		RESET	00'0000 <sub>H</sub>	00 <sub>H</sub>	III
<b>Class A Hardware Traps</b>					
Non-Maskable Interrupt	NMI	NMITRAP	00'0008 <sub>H</sub>	02 <sub>H</sub>	II
Stack Overflow	STKOF	STOTRAP	00'0010 <sub>H</sub>	04 <sub>H</sub>	II
Stack Underflow	STKUF	STUTRAP	00'0018 <sub>H</sub>	06 <sub>H</sub>	II
<b>Class B Hardware Traps</b>					
Undefined Opcode	UNDOPC	BTRAP	00'0028 <sub>H</sub>	0A <sub>H</sub>	I
Protected Instruction Fault	PRTFLT	BTRAP	00'0028 <sub>H</sub>	0A <sub>H</sub>	I
Illegal Word Operand Access	ILLOPA	BTRAP	00'0028 <sub>H</sub>	0A <sub>H</sub>	I
Illegal Instruction Access	ILLINA	BTRAP	00'0028 <sub>H</sub>	0A <sub>H</sub>	I
Illegal External Bus Access	ILLBUS	BTRAP	00'0028 <sub>H</sub>	0A <sub>H</sub>	I
Reserved	–	–	[2C <sub>H</sub> -3C <sub>H</sub> ]	[0B <sub>H</sub> -0F <sub>H</sub> ]	
<b>Software Traps</b>					
TRAP Instruction	–	–	Any [00'0000 <sub>H</sub> - 00'01FC <sub>H</sub> ] in steps of 4 <sub>H</sub>	Any [00 <sub>H</sub> -7F <sub>H</sub> ]	Current CPU Priority

## **Normal Interrupt Processing and PEC Service**

During each instruction cycle one out of all sources which require PEC or interrupt processing is selected according to its interrupt priority. This priority of interrupts and PEC requests is programmable in two levels. Each requesting source can be assigned to a specific priority. A second level (called “group priority”) allows to specify an internal order for simultaneous requests from a group of different sources on the same priority level. At the end of each instruction cycle the one source request with the highest current priority will be determined by the interrupt system. This request will then be serviced, if its priority is higher than the current CPU priority in register PSW.

## **Interrupt System Register Description**

Interrupt processing is controlled globally by register PSW through a general interrupt enable bit (IEN) and the CPU priority field (ILVL). Additionally the different interrupt sources are controlled individually by their specific interrupt control registers (... IC). Thus, the acceptance of requests by the CPU is determined by both the individual interrupt control registers and the PSW. PEC services are controlled by the respective PECCx register and the source and destination pointers, which specify the task of the respective PEC service channel.

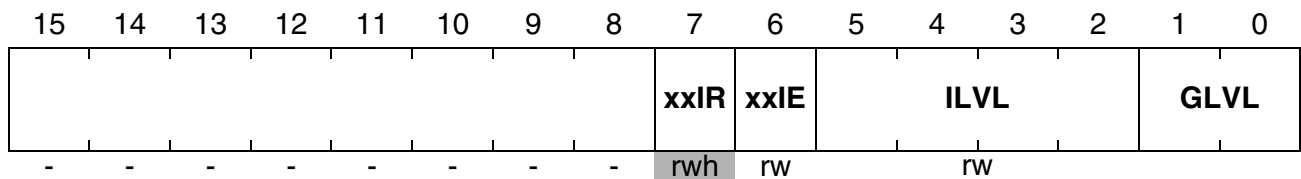
### **5.1.1 Interrupt Control Registers**

All interrupt control registers are organized identically. The lower 8 bits of an interrupt control register contain the complete interrupt status information of the associated source, which is required during one round of prioritization, the upper 8 bits of the respective register are reserved. All interrupt control registers are bit-addressable and all bits can be read or written via software. This allows each interrupt source to be programmed or modified with just one instruction. When accessing interrupt control registers through instructions which operate on word data types, their upper 8 bits (15 ... 8) will return zeros, when read, and will discard written data.

The layout of the Interrupt Control registers shown below applies to each xxIC register, where xx stands for the mnemonic for the respective source.

Interrupt and Trap Functions

xxIC (E)SFR (yyyy<sub>H</sub>/zz<sub>H</sub>) Reset Value: - - 00<sub>H</sub>



Bit	Function
<b>GLVL</b>	<b>Group Level</b> Defines the internal order for simultaneous requests of the same priority. 3: Highest group priority 0: Lowest group priority
<b>ILVL</b>	<b>Interrupt Priority Level</b> Defines the priority level for the arbitration of requests. F <sub>H</sub> : Highest priority level 0 <sub>H</sub> : Lowest priority level
<b>xxIE</b>	<b>Interrupt Enable Control Bit</b> (individually enables/disables a specific source) '0': Interrupt request is disabled '1': Interrupt Request is enabled
<b>xxIR</b>	<b>Interrupt Request Flag</b> '0': No request pending '1': This source has raised an interrupt request

The **Interrupt Request Flag** is set by hardware whenever a service request from the respective source occurs. It is cleared automatically upon entry into the interrupt service routine or upon a PEC service. In the case of PEC service the Interrupt Request flag remains set, if the COUNT field in register PECCx of the selected PEC channel decrements to zero. This allows a normal CPU interrupt to respond to a completed PEC block transfer.

*Note: Modifying the Interrupt Request flag via software causes the same effects as if it had been set or cleared by hardware.*

The **Interrupt Enable Control Bit** determines whether the respective interrupt node takes part in the arbitration cycles (enabled) or not (disabled). The associated request flag will be set upon a source request in any case. The occurrence of an interrupt request can so be polled via xxIR even while the node is disabled.

*Note: In this case the interrupt request flag xxIR is not cleared automatically but must be cleared via software.*

### **Interrupt Priority Level and Group Level**

The four bits of bit field ILVL specify the priority level of a service request for the arbitration of simultaneous requests. The priority increases with the numerical value of ILVL, so 0000<sub>B</sub> is the lowest and 1111<sub>B</sub> is the highest priority level.

When more than one interrupt request on a specific level gets active at the same time, the values in the respective bit fields GLVL are used for second level arbitration to select one request for being serviced. Again the group priority increases with the numerical value of GLVL, so 00<sub>B</sub> is the lowest and 11<sub>B</sub> is the highest group priority.

*Note: All interrupt request sources that are enabled and programmed to the same priority level must always be programmed to different group priorities. Otherwise an incorrect interrupt vector will be generated.*

Upon entry into the interrupt service routine, the priority level of the source that won the arbitration and who's priority level is higher than the current CPU level, is copied into bit field ILVL of register PSW after pushing the old PSW contents on the stack.

The interrupt system of the C161CS/JC/JI allows nesting of up to 15 interrupt service routines of different priority levels (level 0 cannot be arbitrated).

Interrupt requests that are programmed to priority levels 15 or 14 (i.e. ILVL = 111X<sub>B</sub>) will be serviced by the PEC, unless the COUNT field of the associated PECC register contains zero. In this case the request will instead be serviced by normal interrupt processing. Interrupt requests that are programmed to priority levels 13 through 1 will always be serviced by normal interrupt processing.

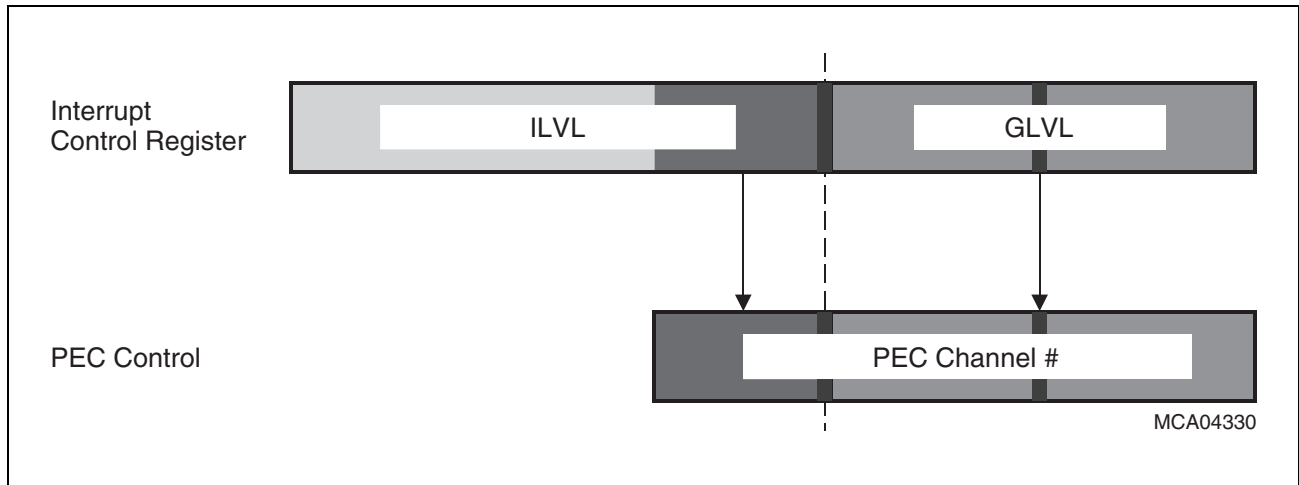
*Note: Priority level 0000<sub>B</sub> is the default level of the CPU. Therefore a request on level 0 will never be serviced, because it can never interrupt the CPU. However, an enabled interrupt request on level 0000<sub>B</sub> will terminate the C161CS/JC/JI's Idle mode and reactivate the CPU.*

For interrupt requests which are to be serviced by the PEC, the associated PEC channel number is derived from the respective ILVL (LSB) and GLVL (see [Figure 5-1](#)). So programming a source to priority level 15 (ILVL = 1111<sub>B</sub>) selects the PEC channel group 7 ... 4, programming a source to priority level 14 (ILVL = 1110<sub>B</sub>) selects the PEC channel group 3 ... 0. The actual PEC channel number is then determined by the group priority field GLVL.

Simultaneous requests for PEC channels are prioritized according to the PEC channel number, where channel 0 has lowest and channel 8 has highest priority.

*Note: All sources that request PEC service must be programmed to different PEC channels. Otherwise an incorrect PEC channel may be activated.*

Interrupt and Trap Functions



**Figure 5-1 Priority Levels and PEC Channels**

**Table 5-3** shows in a few examples, which action is executed with a given programming of an interrupt control register.

**Table 5-3 Interrupt Priority Examples**

Priority Level		Type of Service	
ILVL	GLVL	COUNT = 00H	COUNT ≠ 00H
1 1 1 1	1 1	CPU interrupt, level 15, group priority 3	PEC service, channel 7
1 1 1 1	1 0	CPU interrupt, level 15, group priority 2	PEC service, channel 6
1 1 1 0	1 0	CPU interrupt, level 14, group priority 2	PEC service, channel 2
1 1 0 1	1 0	CPU interrupt, level 13, group priority 2	CPU interrupt, level 13, group priority 2
0 0 0 1	1 1	CPU interrupt, level 1, group priority 3	CPU interrupt, level 1, group priority 3
0 0 0 1	0 0	CPU interrupt, level 1, group priority 0	CPU interrupt, level 1, group priority 0
0 0 0 0	X X	No service!	No service!

*Note: All requests on levels 13 ... 1 cannot initiate PEC transfers. They are always serviced by an interrupt service routine. No PECC register is associated and no COUNT field is checked.*

Interrupt and Trap Functions

**Interrupt Control Functions in the PSW**

The Processor Status Word (PSW) is functionally divided into 2 parts: the lower byte of the PSW basically represents the arithmetic status of the CPU, the upper byte of the PSW controls the interrupt system of the C161CS/JC/JI and the arbitration mechanism for the external bus interface.

*Note: Pipeline effects have to be considered when enabling/disabling interrupt requests via modifications of register PSW (see [Chapter 4](#)).*

**PSW**

Processor Status Word				SFR (FF10 <sub>H</sub> /88 <sub>H</sub> )				Reset Value: 0000 <sub>H</sub>							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ILVL				IEN	HLD EN	-	-	-	USR 0	MUL IP	E	Z	V	C	N
rw				rw	rw	-	-	-	rw	rwh	rwh	rwh	rwh	rwh	rwh

Bit	Function
<b>N, C, V, Z, E, MULIP, USR0</b>	<b>CPU status flags</b> (described in <a href="#">Chapter 4</a> ) Define the current status of the CPU (ALU, multiplication unit).
<b>HLDEN</b>	<b>HOLD Enable</b> (Enables External Bus Arbitration) 0: Bus arbitration disabled, P6.7 ... P6.5 may be used for general purpose IO 1: Bus arbitration enabled, P6.7 ... P6.5 serve as $\overline{\text{BREQ}}$ , $\overline{\text{HLDA}}$ , $\overline{\text{HOLD}}$ , resp.
<b>IEN</b>	<b>Interrupt Enable Control Bit</b> (Globally enables/disables interrupt requests) 0: Interrupt requests are disabled 1: Interrupt requests are enabled
<b>ILVL</b>	<b>CPU Priority Level</b> Defines the current priority level for the CPU F <sub>H</sub> : Highest priority level 0 <sub>H</sub> : Lowest priority level



---

## Interrupt and Trap Functions

**CPU priority ILVL** defines the current level for the operation of the CPU. This bit field reflects the priority level of the routine that is currently executed. Upon the entry into an interrupt service routine this bit field is updated with the priority level of the request that is being serviced. The PSW is saved on the system stack before. The CPU level determines the minimum interrupt priority level that will be serviced. Any request on the same or a lower level will not be acknowledged.

The current CPU priority level may be adjusted via software to control which interrupt request sources will be acknowledged.

PEC transfers do not really interrupt the CPU, but rather “steal” a single cycle, so PEC services do not influence the ILVL field in the PSW.

Hardware traps switch the CPU level to maximum priority (i.e. 15) so no interrupt or PEC requests will be acknowledged while an exception trap service routine is executed.

*Note: The TRAP instruction does not change the CPU level, so software invoked trap service routines may be interrupted by higher requests.*

**Interrupt Enable bit IEN** globally enables or disables PEC operation and the acceptance of interrupts by the CPU. When IEN is cleared, no new interrupt requests are accepted by the CPU. Requests that already have entered the pipeline at that time will process, however. When IEN is set to ‘1’, all interrupt sources, which have been individually enabled by the interrupt enable bits in their associated control registers, are globally enabled.

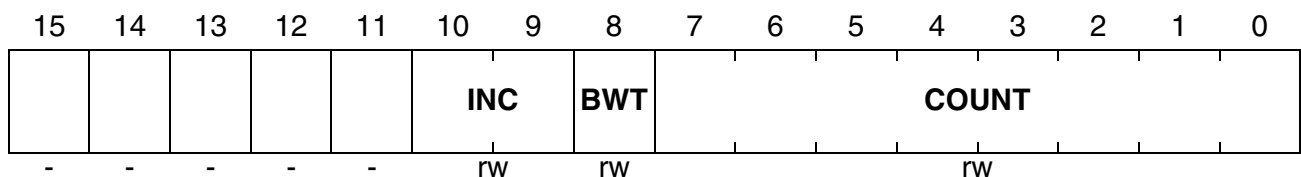
*Note: Traps are non-maskable and are therefore not affected by the IEN bit.*

## 5.2 Operation of the PEC Channels

The C161CS/JC/JI's Peripheral Event Controller (PEC) provides 8 PEC service channels, which move a single byte or word between two locations in segment 0 (data pages 3 ... 0). This is the fastest possible interrupt response and in many cases is sufficient to service the respective peripheral request (e.g. serial channels, etc.). Each channel is controlled by a dedicated PEC Channel Counter/Control register (PECCx) and a pair of pointers for source (SRCPx) and destination (DSTPx) of the data transfer. The PECC registers control the action that is performed by the respective PEC channel.

### PECCx

**PEC Control Reg.**                      **SFR (FECy<sub>H</sub>/6z<sub>H</sub>, see [Table 5-4](#))**                      **Reset Value: 0000<sub>H</sub>**



Bit	Function
<b>COUNT</b>	<b>PEC Transfer Count</b> Counts PEC transfers and influences the channel's action (see <a href="#">Table 5-5</a> )
<b>BWT</b>	<b>Byte/Word Transfer Selection</b> 0: Transfer a Word 1: Transfer a Byte
<b>INC</b>	<b>Increment Control</b> (Modification of SRCPx or DSTPx) 0 0: Pointers are not modified 0 1: Increment DSTPx by 1 or 2 (BWT) 1 0: Increment SRCPx by 1 or 2 (BWT) 1 1: Reserved. Do not use this combination. (changed to '10' by hardware)

**Table 5-4 PEC Control Register Addresses**

Register	Address	Reg. Space	Register	Address	Reg. Space
PECC0	FEC0 <sub>H</sub> /60 <sub>H</sub>	SFR	PECC4	FEC8 <sub>H</sub> /64 <sub>H</sub>	SFR
PECC1	FEC2 <sub>H</sub> /61 <sub>H</sub>	SFR	PECC5	FECA <sub>H</sub> /65 <sub>H</sub>	SFR
PECC2	FEC4 <sub>H</sub> /62 <sub>H</sub>	SFR	PECC6	FECC <sub>H</sub> /66 <sub>H</sub>	SFR
PECC3	FEC6 <sub>H</sub> /63 <sub>H</sub>	SFR	PECC7	FECE <sub>H</sub> /67 <sub>H</sub>	SFR

**Interrupt and Trap Functions**

**Byte/Word Transfer bit BWT** controls, if a byte or a word is moved during a PEC service cycle. This selection controls the transferred data size and the increment step for the modified pointer.

**Increment Control field INC** controls, if one of the PEC pointers is incremented after the PEC transfer. It is not possible to increment both pointers, however. If the pointers are not modified (INC = '00'), the respective channel will always move data from the same source to the same destination.

*Note: The reserved combination '11' is changed to '10' by hardware. However, it is not recommended to use this combination.*

The PEC Transfer Count Field COUNT controls the action of a respective PEC channel, where the content of bit field COUNT at the time the request is activated selects the action. COUNT may allow a specified number of PEC transfers, unlimited transfers or no PEC service at all.

**Table 5-5** summarizes, how the COUNT field itself, the interrupt requests flag IR and the PEC channel action depends on the previous content of COUNT.

**Table 5-5 Influence of Bitfield COUNT**

Previous COUNT	Modified COUNT	IR after PEC Service	Action of PEC Channel and Comments
FF <sub>H</sub>	FF <sub>H</sub>	'0'	Move a Byte/Word Continuous transfer mode, i.e. COUNT is not modified
FE <sub>H</sub> ... 02 <sub>H</sub>	FD <sub>H</sub> ... 01 <sub>H</sub>	'0'	Move a Byte/Word and decrement COUNT
01 <sub>H</sub>	00 <sub>H</sub>	'1'	Move a Byte/Word Leave request flag set, which triggers another request
00 <sub>H</sub>	00 <sub>H</sub>	('1')	<b>No action!</b> Activate interrupt service routine rather than PEC channel

The PEC transfer counter allows to service a specified number of requests by the respective PEC channel, and then (when COUNT reaches 00<sub>H</sub>) activate the interrupt service routine, which is associated with the priority level. After each PEC transfer the COUNT field is decremented and the request flag is cleared to indicate that the request has been serviced.

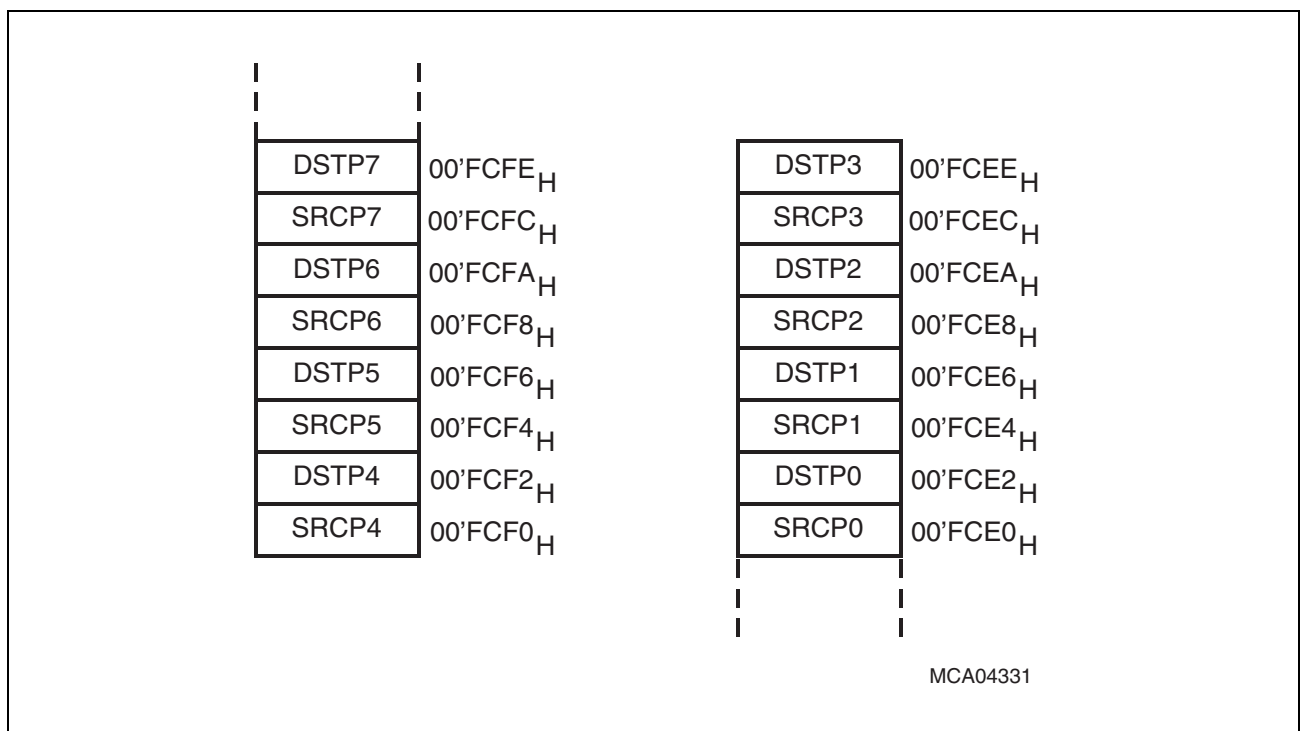
**Interrupt and Trap Functions**

**Continuous transfers** are selected by the value FF<sub>H</sub> in bit field COUNT. In this case COUNT is not modified and the respective PEC channel services any request until it is disabled again.

When COUNT is decremented from 01<sub>H</sub> to 00<sub>H</sub> after a transfer, the request flag is not cleared, which generates another request from the same source. When COUNT already contains the value 00<sub>H</sub>, the respective PEC channel remains idle and the associated interrupt service routine is activated instead. This allows to choose, if a level 15 or 14 request is to be serviced by the PEC or by the interrupt service routine.

*Note: PEC transfers are only executed, if their priority level is higher than the CPU level, i.e. only PEC channels 7 ... 4 are processed, while the CPU executes on level 14. All interrupt request sources that are enabled and programmed for PEC service should use different channels. Otherwise only one transfer will be performed for all simultaneous requests. When COUNT is decremented to 00<sub>H</sub>, and the CPU is to be interrupted, an incorrect interrupt vector will be generated.*

**The source and destination pointers** specify the locations between which the data is to be moved. A pair of pointers (SRCP<sub>x</sub> and DSTP<sub>x</sub>) is associated with each of the 8 PEC channels. These pointers do not reside in specific SFRs, but are mapped into the internal RAM of the C161CS/JC/JI just below the bit-addressable area (see [Figure 5-2](#)).



**Figure 5-2 Mapping of PEC Pointers into the Internal RAM**

---

## Interrupt and Trap Functions

PEC data transfers do not use the data page pointers DPP3 ... DPP0. The PEC source and destination pointers are used as 16-bit intra-segment addresses within segment 0, so data can be transferred between any two locations within the first four data pages 3 ... 0.

The pointer locations for inactive PEC channels may be used for general data storage. Only the required pointers occupy RAM locations.

*Note: If word data transfer is selected for a specific PEC channel (i.e. BWT = '0'), the respective source and destination pointers must both contain a valid word address which points to an even byte boundary. Otherwise the Illegal Word Access trap will be invoked, when this channel is used.*

### 5.3 Prioritization of Interrupt and PEC Service Requests

Interrupt and PEC service requests from all sources can be enabled, so they are arbitrated and serviced (if they win), or they may be disabled, so their requests are disregarded and not serviced.

**Enabling and disabling interrupt requests** may be done via three mechanisms:

**Control bits** allow to switch each individual source “ON” or “OFF”, so it may generate a request or not. The control bits (xxIE) are located in the respective interrupt control registers. All interrupt requests may be enabled or disabled generally via bit IEN in register PSW. This control bit is the “main switch” that selects, if requests from any source are accepted or not.

For a specific request to be arbitrated the respective source’s enable bit and the global enable bit must both be set.

**The priority level** automatically selects a certain group of interrupt requests that will be acknowledged, disclosing all other requests. The priority level of the source that won the arbitration is compared against the CPU’s current level and the source is only serviced, if its level is higher than the current CPU level. Changing the CPU level to a specific value via software blocks all requests on the same or a lower level. An interrupt source that is assigned to level 0 will be disabled and never be serviced.

**The ATOMIC and EXTend instructions** automatically disable all interrupt requests for the duration of the following 1 ... 4 instructions. This is useful e.g. for semaphore handling and does not require to re-enable the interrupt system after the inseparable instruction sequence (see [Chapter 24](#)).

#### Interrupt Class Management

An interrupt class covers a set of interrupt sources with the same importance, i.e. the same priority from the system’s viewpoint. Interrupts of the same class must not interrupt each other. The C161CS/JC/JI supports this function with two features:

Classes with up to 4 members can be established by using the same interrupt priority (ILVL) and assigning a dedicated group level (GLVL) to each member. This functionality is built-in and handled automatically by the interrupt controller.

Classes with more than 4 members can be established by using a number of adjacent interrupt priorities (ILVL) and the respective group levels (4 per ILVL). Each interrupt service routine within this class sets the CPU level to the highest interrupt priority within the class. All requests from the same or any lower level are blocked now, i.e. no request of this class will be accepted.

**Interrupt and Trap Functions**

The example below establishes 3 interrupt classes which cover 2 or 3 interrupt priorities, depending on the number of members in a class. A level 6 interrupt disables all other sources in class 2 by changing the current CPU level to 8, which is the highest priority (ILVL) in class 2. Class 1 requests or PEC requests are still serviced in this case.

The 24 interrupt sources (excluding PEC requests) are so assigned to 3 classes of priority rather than to 7 different levels, as the hardware support would do.

**Table 5-6 Software Controlled Interrupt Classes (Example)**

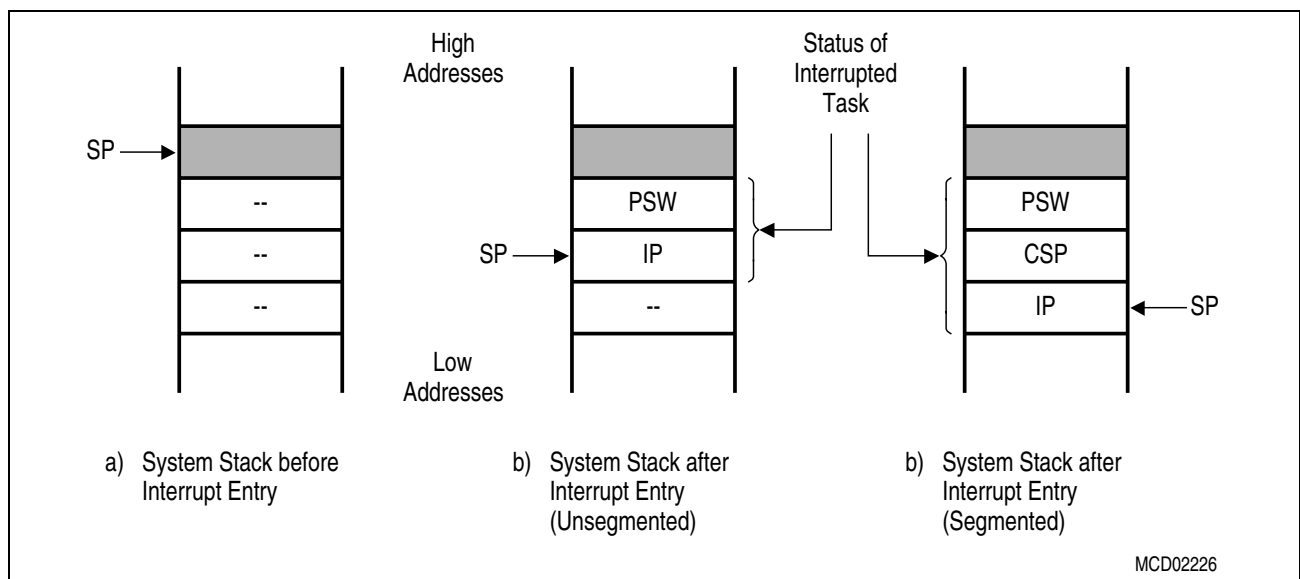
ILVL (Priority)	GLVL				Interpretation
	3	2	1	0	
15					PEC service on up to 8 channels
14					
13					
12	X	X	X	X	Interrupt Class 1 5 sources on 2 levels
11	X				
10					
9					
8	X	X	X	X	Interrupt Class 2 9 sources on 3 levels
7	X	X	X	X	
6	X				
5	X	X	X	X	Interrupt Class 3 5 sources on 2 levels
4	X				
3					
2					
1					
0					No service!

## 5.4 Saving the Status During Interrupt Service

Before an interrupt request that has been arbitrated is actually serviced, the status of the current task is automatically saved on the system stack. The CPU status (PSW) is saved along with the location, where the execution of the interrupted task is to be resumed after returning from the service routine. This return location is specified through the Instruction Pointer (IP) and, in case of a segmented memory model, the Code Segment Pointer (CSP). Bit SGTDIS in register SYSCON controls, how the return location is stored.

The system stack receives the PSW first, followed by the IP (unsegmented) or followed by CSP and then IP (segmented mode). This optimizes the usage of the system stack, if segmentation is disabled.

The CPU priority field (ILVL in PSW) is updated with the priority of the interrupt request that is to be serviced, so the CPU now executes on the new level. If a multiplication or division was in progress at the time the interrupt request was acknowledged, bit MULIP in register PSW is set to '1'. In this case the return location that is saved on the stack is not the next instruction in the instruction flow, but rather the multiply or divide instruction itself, as this instruction has been interrupted and will be completed after returning from the service routine.



**Figure 5-3 Task Status Saved on the System Stack**

The interrupt request flag of the source that is being serviced is cleared. The IP is loaded with the vector associated with the requesting source (the CSP is cleared in case of segmentation) and the first instruction of the service routine is fetched from the respective vector location, which is expected to branch to the service routine itself. The data page pointers and the context pointer are not affected.

When the interrupt service routine is left (RETI is executed), the status information is popped from the system stack in the reverse order, taking into account the value of bit SGTDIS.



## Context Switching

An interrupt service routine usually saves all the registers it uses on the stack, and restores them before returning. The more registers a routine uses, the more time is wasted with saving and restoring. The C161CS/JC/JI allows to switch the complete bank of CPU registers (GPRs) with a single instruction, so the service routine executes within its own, separate context.

The instruction “SCXT CP, #New\_Bank” pushes the content of the context pointer (CP) on the system stack and loads CP with the immediate value “New\_Bank”, which selects a new register bank. The service routine may now use its “own registers”. This register bank is preserved, when the service routine terminates, i.e. its contents are available on the next call.

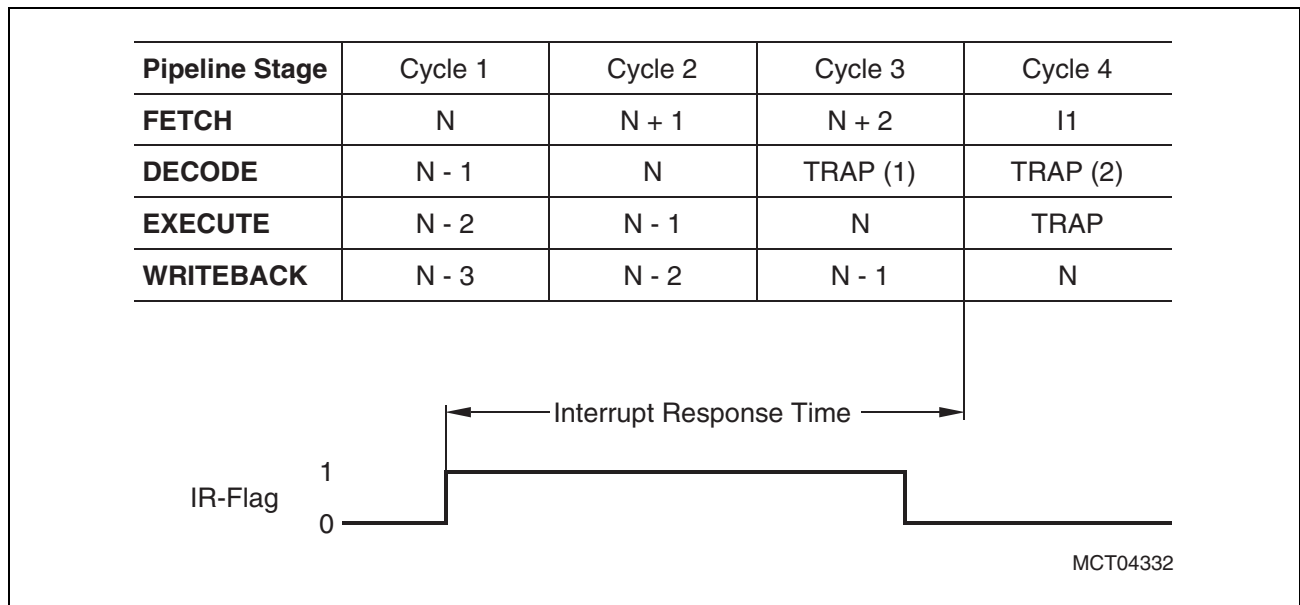
Before returning (RETI) the previous CP is simply POPped from the system stack, which returns the registers to the original bank.

*Note: The first instruction following the SCXT instruction must not use a GPR.*

Resources that are used by the interrupting program must eventually be saved and restored, e.g. the DPPs and the registers of the MUL/DIV unit.

## 5.5 Interrupt Response Times

The interrupt response time defines the time from an interrupt request flag of an enabled interrupt source being set until the first instruction (I1) being fetched from the interrupt vector location. The basic interrupt response time for the C161CS/JC/JI is 3 instruction cycles.



**Figure 5-4 Pipeline Diagram for Interrupt Response Time**

All instructions in the pipeline including instruction N (during which the interrupt request flag is set) are completed before entering the service routine. The actual execution time for these instructions (e.g. waitstates) therefore influences the interrupt response time.

In **Figure 5-4** the respective interrupt request flag is set in cycle 1 (fetching of instruction N). The indicated source wins the prioritization round (during cycle 2). In cycle 3 a TRAP instruction is injected into the decode stage of the pipeline, replacing instruction N + 1 and clearing the source's interrupt request flag to '0'. Cycle 4 completes the injected TRAP instruction (save PSW, IP and CSP, if segmented mode) and fetches the first instruction (I1) from the respective vector location.

All instructions that entered the pipeline after setting of the interrupt request flag (N + 1, N + 2) will be executed after returning from the interrupt service routine.

The minimum interrupt response time is 5 states (10 TCL). This requires program execution from the internal code memory, no external operand read requests and setting the interrupt request flag during the last state of an instruction cycle. When the interrupt request flag is set during the first state of an instruction cycle, the minimum interrupt response time under these conditions is 6 state times (12 TCL).

The interrupt response time is increased by all delays of the instructions in the pipeline that are executed before entering the service routine (including N).

**Interrupt and Trap Functions**

- When internal hold conditions between instruction pairs  $N - 2/N - 1$  or  $N - 1/N$  occur, or instruction  $N$  explicitly writes to the PSW or the SP, the minimum interrupt response time may be extended by 1 state time for each of these conditions.
- When instruction  $N$  reads an operand from the internal code memory, or when  $N$  is a call, return, trap, or MOV  $R_n, [R_m + \#data16]$  instruction, the minimum interrupt response time may additionally be extended by 2 state times during internal code memory program execution.
- In case instruction  $N$  reads the PSW and instruction  $N - 1$  has an effect on the condition flags, the interrupt response time may additionally be extended by 2 state times.

The worst case interrupt response time during internal code memory program execution adds to 12 state times (24 TCL).

Any reference to external locations increases the interrupt response time due to pipeline related access priorities. The following conditions have to be considered:

- Instruction fetch from an external location
- Operand read from an external location
- Result write-back to an external location

Depending on where the instructions, source and destination operands are located, there are a number of combinations. Note, however, that only access conflicts contribute to the delay.

A few examples illustrate these delays:

- The worst case interrupt response time including external accesses will occur, when instructions  $N, N + 1$  and  $N + 2$  are executed out of external memory, instructions  $N - 1$  and  $N$  require external operand read accesses, instructions  $N - 3$  through  $N$  write back external operands, and the interrupt vector also points to an external location. In this case the interrupt response time is the time to perform 9 word bus accesses, because instruction  $I_1$  cannot be fetched via the external bus until all write, fetch and read requests of preceding instructions in the pipeline are terminated.
- When the above example has the interrupt vector pointing into the internal code memory, the interrupt response time is 7 word bus accesses plus 2 states, because fetching of instruction  $I_1$  from internal code memory can start earlier.
- When instructions  $N, N + 1$  and  $N + 2$  are executed out of external memory and the interrupt vector also points to an external location, but all operands for instructions  $N - 3$  through  $N$  are in internal memory, then the interrupt response time is the time to perform 3 word bus accesses.
- When the above example has the interrupt vector pointing into the internal code memory, the interrupt response time is 1 word bus access plus 4 states.

---

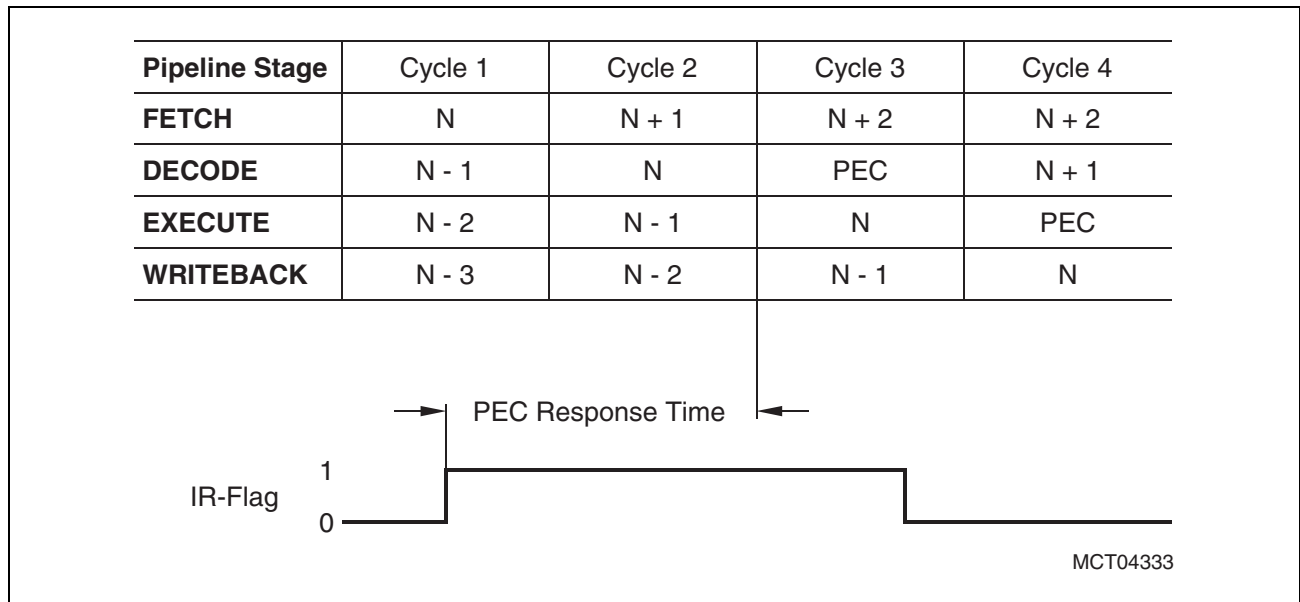
## Interrupt and Trap Functions

After an interrupt service routine has been terminated by executing the RETI instruction, and if further interrupts are pending, the next interrupt service routine will not be entered until at least two instruction cycles have been executed of the program that was interrupted. In most cases two instructions will be executed during this time. Only one instruction will typically be executed, if the first instruction following the RETI instruction is a branch instruction (without cache hit), or if it reads an operand from internal code memory, or if it is executed out of the internal RAM.

*Note: A bus access in this context includes all delays which can occur during an external bus cycle.*

## 5.6 PEC Response Times

The PEC response time defines the time from an interrupt request flag of an enabled interrupt source being set until the PEC data transfer being started. The basic PEC response time for the C161CS/JC/JI is 2 instruction cycles.



**Figure 5-5 Pipeline Diagram for PEC Response Time**

In **Figure 5-5** above the respective interrupt request flag is set in cycle 1 (fetching of instruction N). The indicated source wins the prioritization round (during cycle 2). In cycle 3 a PEC transfer “instruction” is injected into the decode stage of the pipeline, suspending instruction N + 1 and clearing the source’s interrupt request flag to ‘0’. Cycle 4 completes the injected PEC transfer and resumes the execution of instruction N + 1.

All instructions that entered the pipeline after setting of the interrupt request flag (N + 1, N + 2) will be executed after the PEC data transfer.

*Note: When instruction N reads any of the PEC control registers PECC7 ... PECC0, while a PEC request wins the current round of prioritization, this round is repeated and the PEC data transfer is started one cycle later.*

The minimum PEC response time is 3 states (6 TCL). This requires program execution from the internal code memory, no external operand read requests and setting the interrupt request flag during the last state of an instruction cycle. When the interrupt request flag is set during the first state of an instruction cycle, the minimum PEC response time under these conditions is 4 state times (8 TCL).

The PEC response time is increased by all delays of the instructions in the pipeline that are executed before starting the data transfer (including N).

**Interrupt and Trap Functions**

- When internal hold conditions between instruction pairs  $N - 2/N - 1$  or  $N - 1/N$  occur, the minimum PEC response time may be extended by 1 state time for each of these conditions.
- When instruction  $N$  reads an operand from the internal code memory, or when  $N$  is a call, return, trap, or `MOV Rn, [Rm+ #data16]` instruction, the minimum PEC response time may additionally be extended by 2 state times during internal code memory program execution.
- In case instruction  $N$  reads the PSW and instruction  $N - 1$  has an effect on the condition flags, the PEC response time may additionally be extended by 2 state times.

The worst case PEC response time during internal code memory program execution adds to 9 state times (18 TCL).

Any reference to external locations increases the PEC response time due to pipeline related access priorities. The following conditions have to be considered:

- Instruction fetch from an external location
- Operand read from an external location
- Result write-back to an external location

Depending on where the instructions, source and destination operands are located, there are a number of combinations. Note, however, that only access conflicts contribute to the delay.

A few examples illustrate these delays:

- The worst case interrupt response time including external accesses will occur, when instructions  $N$  and  $N + 1$  are executed out of external memory, instructions  $N - 1$  and  $N$  require external operand read accesses and instructions  $N - 3$ ,  $N - 2$  and  $N - 1$  write back external operands. In this case the PEC response time is the time to perform 7 word bus accesses.
- When instructions  $N$  and  $N + 1$  are executed out of external memory, but all operands for instructions  $N - 3$  through  $N - 1$  are in internal memory, then the PEC response time is the time to perform 1 word bus access plus 2 state times.

Once a request for PEC service has been acknowledged by the CPU, the execution of the next instruction is delayed by 2 state times plus the additional time it might take to fetch the source operand from internal code memory or external memory and to write the destination operand over the external bus in an external program environment.

*Note: A bus access in this context includes all delays which can occur during an external bus cycle.*

## 5.7 Interrupt Node Sharing

Interrupt nodes may be shared between several module requests either if the requests are generated mutually exclusive or if the requests are generated at a low rate. If more than one source is enabled in this case the interrupt handler will first have to determine the requesting source. However, this overhead is not critical for low rate requests.

This node sharing is controlled via the sub-node interrupt control register ISNC which provides a separate request flag and enable bit for each supported request source. The interrupt level used for arbitration is determined by the node control register (... IC).

### ISNC

Interrupt Sub-Node Ctrl. Reg. **ESFR (F1DE<sub>H</sub>/EF<sub>H</sub>)** Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	PLL IE	PLL IR	RTC IE	RTC IR
-	-	-	-	-	-	-	-	-	-	-	-	rw	rw	rw	rw

Bit	Function
<b>xxIR</b>	<b>Interrupt Request Flag for Source xx</b> 0: No request from source xx pending. 1: Source xx has raised an interrupt request.
<b>xxIE</b>	<b>Interrupt Enable Control Bit for Source xx</b> 0: Source xx interrupt request is disabled. 1: Source xx interrupt request is enabled.

**Table 5-7 Sub-Node Control Bit Allocation**

Bit pos.	Interrupt Source	Associated Node
15 ... 4	<i>Reserved.</i>	<i>Reserved.</i>
3 2	PLL / OWD	XP3IC
1 0	RTC	XP3IC

*Note: In order to ensure compatibility with other derivatives application software should never set reserved bits within register ISNC.*

## 5.8 External Interrupts

Although the C161CS/JC/JI has no dedicated INTR input pins, it provides many possibilities to react on external asynchronous events by using a number of IO lines for interrupt input. The interrupt function may either be combined with the pin's main function or may be used instead of it, i.e. if the main pin function is not required.

Interrupt signals may be connected to:

- CC31IO ... CC16IO, the capture input/compare output lines of the CAPCOM2 unit,
- CC15IO ... CC0IO, the capture input/compare output lines of the CAPCOM1 unit,
- T4IN, T2IN, the timer input pins,
- CAPIN, the capture input of GPT2.

For each of these pins either a positive, a negative, or both a positive and a negative external transition can be selected to cause an interrupt or PEC service request. The edge selection is performed in the control register of the peripheral device associated with the respective port pin. The peripheral must be programmed to a specific operating mode to allow generation of an interrupt by the external signal. The priority of the interrupt request is determined by the interrupt control register of the respective peripheral interrupt source, and the interrupt vector of this source will be used to service the external interrupt request.

*Note: In order to use any of the listed pins as external interrupt input, it must be switched to input mode via its direction control bit DPx.y in the respective port direction control register DPx.*

**Table 5-8 Pins to be Used as External Interrupt Inputs**

Port Pin	Original Function	Control Register
P7.7-4/CC31-28IO	CAPCOM register 31-28 capture input	CC31-CC28
P1H.7-4/CC27-24IO	CAPCOM register 27-24 capture input	CC27-CC24
P2.15-8/CC15-8IO	CAPCOM register 15-8 capture input	CC15-CC8
P3.7/T2IN	Auxiliary timer T2 input pin	T2CON
P3.5/T4IN	Auxiliary timer T4 input pin	T4CON
P3.2/CAPIN	GPT2 capture input pin	T5CON

When port pins CCxIO are to be used as external interrupt input pins, bit field CCMODx in the control register of the corresponding capture/compare register CCx must select capture mode. When CCMODx is programmed to 001<sub>B</sub>, the interrupt request flag CCxIR in register CCxIC will be set on a positive external transition at pin CCxIO. When CCMODx is programmed to 010<sub>B</sub>, a negative external transition will set the interrupt request flag. When CCMODx = 011<sub>B</sub>, both a positive and a negative transition will set the request flag. In all three cases, the contents of the allocated CAPCOM timer will be



**Interrupt and Trap Functions**

latched into capture register CCx, independent whether the timer is running or not. When the interrupt enable bit CCxIE is set, a PEC request or an interrupt request for vector CCxINT will be generated.

Pins T2IN or T4IN can be used as external interrupt input pins when the associated auxiliary timer T2 or T4 in block GPT1 is configured for capture mode. This mode is selected by programming the mode control fields T2M or T4M in control registers T2CON or T4CON to 101<sub>B</sub>. The active edge of the external input signal is determined by bit fields T2I or T4I. When these fields are programmed to X01<sub>B</sub>, interrupt request flags T2IR or T4IR in registers T2IC or T4IC will be set on a positive external transition at pins T2IN or T4IN, respectively. When T2I or T4I are programmed to X10<sub>B</sub>, then a negative external transition will set the corresponding request flag. When T2I or T4I are programmed to X11<sub>B</sub>, both a positive and a negative transition will set the request flag. In all three cases, the contents of the core timer T3 will be captured into the auxiliary timer registers T2 or T4 based on the transition at pins T2IN or T4IN. When the interrupt enable bits T2IE or T4IE are set, a PEC request or an interrupt request for vector T2INT or T4INT will be generated.

Pin CAPIN differs slightly from the timer input pins as it can be used as external interrupt input pin without affecting peripheral functions. When the capture mode enable bit T5SC in register T5CON is cleared to '0', signal transitions on pin CAPIN will only set the interrupt request flag CRIR in register CRIC, and the capture function of register CAPREL is not activated.

So register CAPREL can still be used as reload register for GPT2 timer T5, while pin CAPIN serves as external interrupt input. Bit field CI in register T5CON selects the effective transition of the external interrupt input signal. When CI is programmed to 01<sub>B</sub>, a positive external transition will set the interrupt request flag. CI = 10<sub>B</sub> selects a negative transition to set the interrupt request flag, and with CI = 11<sub>B</sub>, both a positive and a negative transition will set the request flag. When the interrupt enable bit CRIE is set, an interrupt request for vector CRINT or a PEC request will be generated.

*Note: The non-maskable interrupt input pin  $\overline{NMI}$  (sample rate 2 TCL) and the reset input  $\overline{RSTIN}$  provide another possibility for the CPU to react on an external input signal.  $\overline{NMI}$  and  $\overline{RSTIN}$  are dedicated input pins, which cause hardware traps.*

**Interrupt and Trap Functions**

**Fast External Interrupts**

The input pins that may be used for external interrupts are sampled every 16 TCL, i.e. external events are scanned and detected in timeframes of 16 TCL (8 TCL for CAPIN). The C161CS/JC/JI provides 8 interrupt inputs that are sampled every 2 TCL, so external events are captured faster than with standard interrupt inputs.

The upper 8 pins of Port 2 (P2.15-P2.8) can individually be programmed to this fast interrupt mode, where also the trigger transition (rising, falling or both) can be selected. The External Interrupt Control register EXICON controls this feature for all 8 pins.

**EXICON**

**Ext. Interrupt Control Reg.      ESFR (F1C0<sub>H</sub>/E0<sub>H</sub>)      Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>EXI7ES</b>		<b>EXI6ES</b>		<b>EXI5ES</b>		<b>EXI4ES</b>		<b>EXI3ES</b>		<b>EXI2ES</b>		<b>EXI1ES</b>		<b>EXI0ES</b>	
rw		rw		rw		rw		rw		rw		rw		rw	

Bit	Function
<b>EXIxES</b>	<b>External Interrupt x Edge Selection Field (x = 7 ... 0)</b> 0 0: Fast external interrupts disabled: standard mode 0 1: Interrupt on positive edge (rising) 1 0: Interrupt on negative edge (falling) 1 1: Interrupt on any edge (rising or falling)

*Note: The fast external interrupt inputs are sampled every 2 TCL. The interrupt request arbitration and processing, however, is executed every 8 TCL.*

These fast external interrupts use the interrupt nodes and vectors of the CAPCOM channels CC8-CC15, so the capture/compare function cannot be used on the respective Port 2 pins (with EXIxES ≠ 00<sub>B</sub>). However, general purpose IO is possible in all cases.

**Interrupt and Trap Functions**

**External Interrupt Source Control**

The input source for fast external interrupts (controlled via register EXICON) can be derived either from the associated port pin EXnIN or from an alternate source. This selection is controlled via register EXISEL.

Activating the alternate input source e.g. permits the detection of transitions on the interface lines of disabled interfaces. Upon this trigger the respective interface can be reactivated and respond to the detected activity.

**EXISEL**

**Ext. Interrupt Source Reg.      ESFR (F1DA<sub>H</sub>/ED<sub>H</sub>)      Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>EXI7SS</b>		<b>EXI6SS</b>		<b>EXI5SS</b>		<b>EXI4SS</b>		<b>EXI3SS</b>		<b>EXI2SS</b>		<b>EXI1SS</b>		<b>EXI0SS</b>	
rw		rw		rw		rw		rw		rw		rw		rw	

Bit	Function
<b>EXIxSS</b>	<b>External Interrupt x Source Selection Field (x = 7 ... 0)</b> 00: Input from associated EXzIN pin. 01: Input from alternate pin. 10: Input from pin EXzIN ORed with alternate pin. 11: Input from pin EXzIN ANDed with alternate pin.

**Table 5-9** summarizes the association of the bitfields of register EXISEL with the respective interface input lines.

**Table 5-9      Connection of Interface Inputs to External Interrupt Nodes**

Bitfield	Associated Interface Line	Notes
EXI0SS	CAN1_RxD ( <b>C161CS/JC</b> )	The used pin depends on the assignment for the respective module.
EXI1SS	CAN2_RxD ( <b>C161CS</b> )	
EXI2SS	RxD0	ASC0
EXI3SS	SCLK	SSC
EXI4SS	RxD1	ASC1
EXI7SS	SDL_RxD ( <b>C161JC/JI</b> )	The used pin depends on the assignment for the SDLM.

### External Interrupts During Sleep Mode

During Sleep mode all peripheral clock signals are deactivated which also disables the standard edge detection logic for the fast external interrupts. However, transitions on these interrupt inputs must be recognized in order to initiate the wakeup. Therefore during Sleep mode a special edge detection logic for the fast external interrupts (EXzIN) is activated, which requires no clock signal (therefore also works in Sleep mode) and is equipped with an analog noise filter. This filter suppresses spikes (generated by noise) up to 10 ns. Input pulses with a duration of 100 ns minimum are recognized and generate an interrupt request.

This filter delays the recognition of an external wakeup signal by approx. 100 ns, but the spike suppression ensures safe and robust operation of the sleep/wakeup mechanism in an active environment.

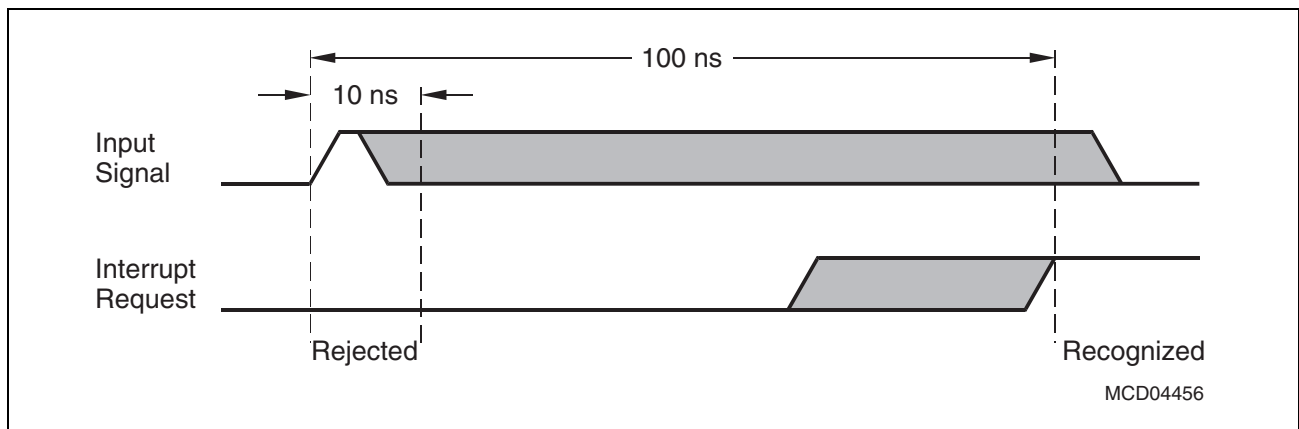


Figure 5-6 Input Noise Filter Operation

## 5.9 Trap Functions

Traps interrupt the current execution similar to standard interrupts. However, trap functions offer the possibility to bypass the interrupt system's prioritization process in cases where immediate system reaction is required. Trap functions are not maskable and always have priority over interrupt requests on any priority level.

The C161CS/JC/JI provides two different kinds of trapping mechanisms. **Hardware traps** are triggered by events that occur during program execution (e.g. illegal access or undefined opcode), **software traps** are initiated via an instruction within the current execution flow.

### Software Traps

The TRAP instruction is used to cause a software call to an interrupt service routine. The trap number that is specified in the operand field of the trap instruction determines which vector location in the address range from 00'0000<sub>H</sub> through 00'01FC<sub>H</sub> will be branched to. Executing a TRAP instruction causes a similar effect as if an interrupt at the same vector had occurred. PSW, CSP (in segmentation mode), and IP are pushed on the internal system stack and a jump is taken to the specified vector location. When segmentation is enabled and a trap is executed, the CSP for the trap service routine is set to code segment 0. No Interrupt Request flags are affected by the TRAP instruction. The interrupt service routine called by a TRAP instruction must be terminated with a RETI (return from interrupt) instruction to ensure correct operation.

*Note: The CPU level in register PSW is not modified by the TRAP instruction, so the service routine is executed on the same priority level from which it was invoked. Therefore, the service routine entered by the TRAP instruction can be interrupted by other traps or higher priority interrupts, other than when triggered by a hardware trap.*

### Hardware Traps

Hardware traps are issued by faults or specific system states that occur during runtime of a program (not identified at assembly time). A hardware trap may also be triggered intentionally, e.g. to emulate additional instructions by generating an Illegal Opcode trap. The C161CS/JC/JI distinguishes eight different hardware trap functions. When a hardware trap condition has been detected, the CPU branches to the trap vector location for the respective trap condition. Depending on the trap condition, the instruction which caused the trap is either completed or cancelled (i.e. it has no effect on the system state) before the trap handling routine is entered.

Hardware traps are non-maskable and always have priority over every other CPU activity. If several hardware trap conditions are detected within the same instruction cycle, the highest priority trap is serviced (see [Table 5-2](#)).

**Interrupt and Trap Functions**

PSW, CSP (in segmentation mode), and IP are pushed on the internal system stack and the CPU level in register PSW is set to the highest possible priority level (i.e. level 15), disabling all interrupts. The CSP is set to code segment zero, if segmentation is enabled. A trap service routine must be terminated with the RETI instruction.

The eight hardware trap functions of the C161CS/JC/JI are divided into two classes:

**Class A traps** are

- External Non-Maskable Interrupt ( $\overline{\text{NMI}}$ )
- Stack Overflow
- Stack Underflow Trap

These traps share the same trap priority, but have an individual vector address.

**Class B traps** are

- Undefined Opcode
- Protection Fault
- Illegal Word Operand Access
- Illegal Instruction Access
- Illegal External Bus Access Trap

These traps share the same trap priority, and the same vector address.

The bit-addressable Trap Flag Register (TFR) allows a trap service routine to identify the kind of trap which caused the exception. Each trap function is indicated by a separate request flag. When a hardware trap occurs, the corresponding request flag in register TFR is set to '1'.

The reset functions (hardware, software, watchdog) may be regarded as a type of trap. Reset functions have the highest system priority (trap priority III).

Class A traps have the second highest priority (trap priority II), on the 3rd rank are class B traps, so a class A trap can interrupt a class B trap. If more than one class A trap occur at a time, they are prioritized internally, with the NMI trap on the highest and the stack underflow trap on the lowest priority.

All class B traps have the same trap priority (trap priority I). When several class B traps get active at a time, the corresponding flags in the TFR register are set and the trap service routine is entered. Since all class B traps have the same vector, the priority of service of simultaneously occurring class B traps is determined by software in the trap service routine.

A class A trap occurring during the execution of a class B trap service routine will be serviced immediately. During the execution of a class A trap service routine, however, any class B trap occurring will not be serviced until the class A trap service routine is exited with a RETI instruction. In this case, the occurrence of the class B trap condition is stored in the TFR register, but the IP value of the instruction which caused this trap is lost.

**Interrupt and Trap Functions**

**TFR**

**Trap Flag Register**

**SFR (FFAC<sub>H</sub>/D6<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>NMI</b>	<b>STK OF</b>	<b>STK UF</b>	-	-	-	-	-	<b>UND OPC</b>	-	-	-	<b>PRT FLT</b>	<b>ILL OPA</b>	<b>ILL INA</b>	<b>ILL BUS</b>
rwh	rwh	rwh	-	-	-	-	-	rwh	-	-	-	rwh	rwh	rwh	rwh

Bit	Function
<b>ILLBUS</b>	<b>Illegal External Bus Access Flag</b> An external access has been attempted with no external bus defined.
<b>ILLINA</b>	<b>Illegal Instruction Access Flag</b> A branch to an odd address has been attempted.
<b>ILLOPA</b>	<b>Illegal Word Operand Access Flag</b> A word operand access (read or write) to an odd address has been attempted.
<b>PRTFLT</b>	<b>Protection Fault Flag</b> A protected instruction with an illegal format has been detected.
<b>UNDOPC</b>	<b>Undefined Opcode Flag</b> The currently decoded instruction has no valid C161CS/JC/JI opcode.
<b>STKUF</b>	<b>Stack Underflow Flag</b> The current stack pointer value exceeds the content of register STKUN.
<b>STKOF</b>	<b>Stack Overflow Flag</b> The current stack pointer value falls below the content of reg. STKOV.
<b>NMI</b>	<b>Non Maskable Interrupt Flag</b> A negative transition (falling edge) has been detected on pin $\overline{\text{NMI}}$ .

*Note: The trap service routine must clear the respective trap flag, otherwise a new trap will be requested after exiting the service routine. Setting a trap request flag by software causes the same effects as if it had been set by hardware.*

In the case where e.g. an Undefined Opcode trap (class B) occurs simultaneously with an NMI trap (class A), both the NMI and the UNDOPC flag is set, the IP of the instruction with the undefined opcode is pushed onto the system stack, but the NMI trap is executed. After return from the NMI service routine, the IP is popped from the stack and immediately pushed again because of the pending UNDOPC trap.

**External NMI Trap**

Whenever a high to low transition on the dedicated external  $\overline{\text{NMI}}$  pin (Non-Maskable Interrupt) is detected, the NMI flag in register TFR is set and the CPU will enter the NMI trap routine. The IP value pushed on the system stack is the address of the instruction following the one after which normal processing was interrupted by the NMI trap.

*Note: The  $\overline{\text{NMI}}$  pin is sampled with every CPU clock cycle to detect transitions.*

**Stack Overflow Trap**

Whenever the stack pointer is decremented to a value which is less than the value in the stack overflow register STKOV, the STKOF flag in register TFR is set and the CPU will enter the stack overflow trap routine. Which IP value will be pushed onto the system stack depends on which operation caused the decrement of the SP. When an implicit decrement of the SP is made through a PUSH or CALL instruction, or upon interrupt or trap entry, the IP value pushed is the address of the following instruction. When the SP is decremented by a subtract instruction, the IP value pushed represents the address of the instruction after the instruction following the subtract instruction.

For recovery from stack overflow it must be ensured that there is enough excess space on the stack for saving the current system state (PSW, IP, in segmented mode also CSP) twice. Otherwise, a system reset should be generated.

**Stack Underflow Trap**

Whenever the stack pointer is incremented to a value which is greater than the value in the stack underflow register STKUN, the STKUF flag is set in register TFR and the CPU will enter the stack underflow trap routine. Again, which IP value will be pushed onto the system stack depends on which operation caused the increment of the SP. When an implicit increment of the SP is made through a POP or return instruction, the IP value pushed is the address of the following instruction. When the SP is incremented by an add instruction, the pushed IP value represents the address of the instruction after the instruction following the add instruction.

**Undefined Opcode Trap**

When the instruction currently decoded by the CPU does not contain a valid C161CS/JC/JI opcode, the UNDOPC flag is set in register TFR and the CPU enters the undefined opcode trap routine. The IP value pushed onto the system stack is the address of the instruction that caused the trap.

This can be used to emulate unimplemented instructions. The trap service routine can examine the faulting instruction to decode operands for unimplemented opcodes based on the stacked IP. In order to resume processing, the stacked IP value must be incremented by the size of the undefined instruction, which is determined by the user, before a RETI instruction is executed.



**Protection Fault Trap**

Whenever one of the special protected instructions is executed where the opcode of that instruction is not repeated twice in the second word of the instruction and the byte following the opcode is not the complement of the opcode, the PRTFLT flag in register TFR is set and the CPU enters the protection fault trap routine. The protected instructions include DISWDT, EINIT, IDLE, PWRDN, SRST, and SRVWDT. The IP value pushed onto the system stack for the protection fault trap is the address of the instruction that caused the trap.

**Illegal Word Operand Access Trap**

Whenever a word operand read or write access is attempted to an odd byte address, the ILLOPA flag in register TFR is set and the CPU enters the illegal word operand access trap routine. The IP value pushed onto the system stack is the address of the instruction following the one which caused the trap.

**Illegal Instruction Access Trap**

Whenever a branch is made to an odd byte address, the ILLINA flag in register TFR is set and the CPU enters the illegal instruction access trap routine. The IP value pushed onto the system stack is the illegal odd target address of the branch instruction.

**Illegal External Bus Access Trap**

Whenever the CPU requests an external instruction fetch, data read or data write, and no external bus configuration has been specified, the ILLBUS flag in register TFR is set and the CPU enters the illegal bus access trap routine. The IP value pushed onto the system stack is the address of the instruction following the one which caused the trap.

## 6 Clock Generation

All activities of the C161CS/JC/JI's controller hardware and its on-chip peripherals are controlled via the system clock signal  $f_{\text{CPU}}$ .

This reference clock is generated in three stages (see also [Figure 6-1](#)):

### Oscillators

The on-chip Pierce oscillators (main oscillator and auxiliary oscillator) can either run with an external crystal and appropriate oscillator circuitry or they can be driven by an external oscillator or another clock source.

### Frequency Control

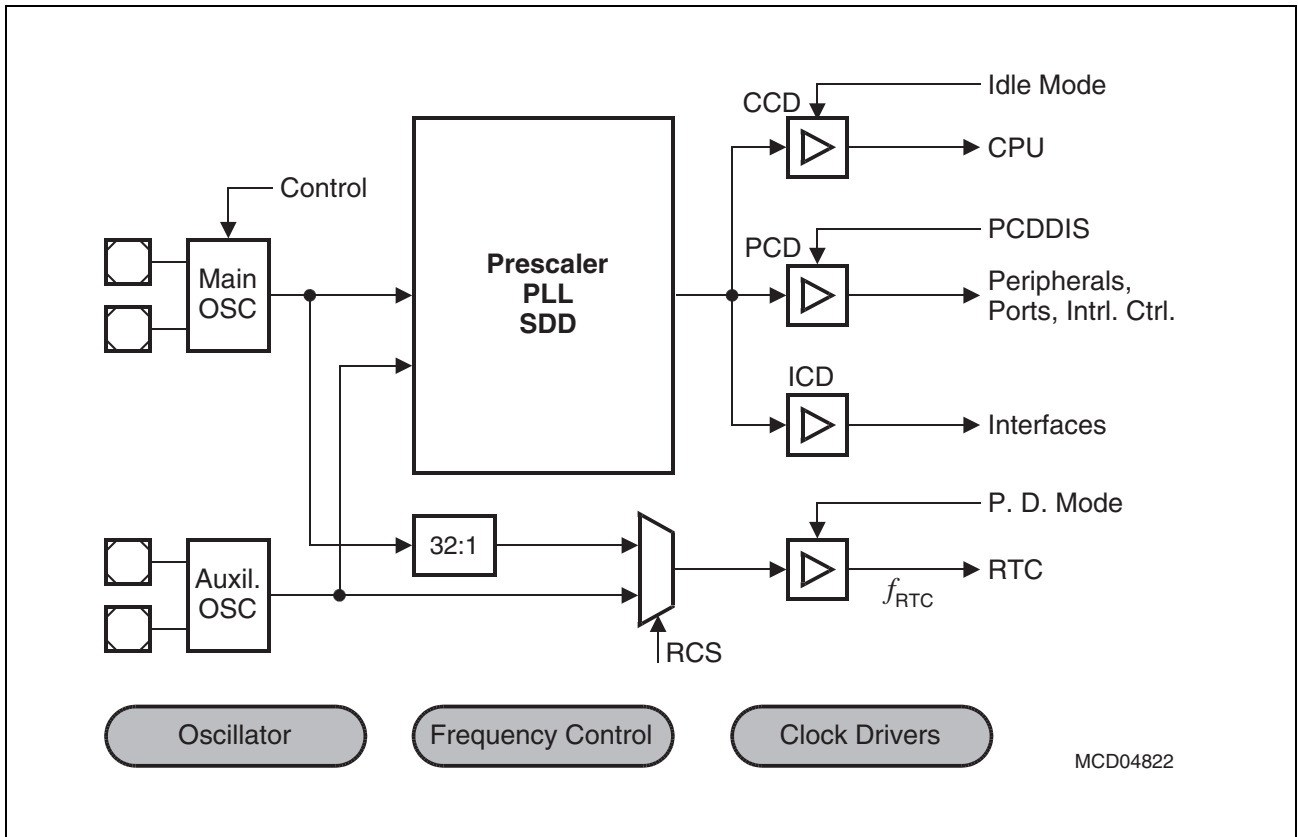
The input clock signal feeds the controller hardware:

- directly, providing phase coupled operation on not too high input frequency
- divided by 2 in order to get 50% duty cycle clock signal
- via an on-chip phase locked loop (PLL) providing max. performance on low input frequency
- via the Slow Down Divider (SDD) in order to reduce the power consumption.

The resulting internal clock signal is referred to as "CPU clock"  $f_{\text{CPU}}$ .

### Clock Drivers

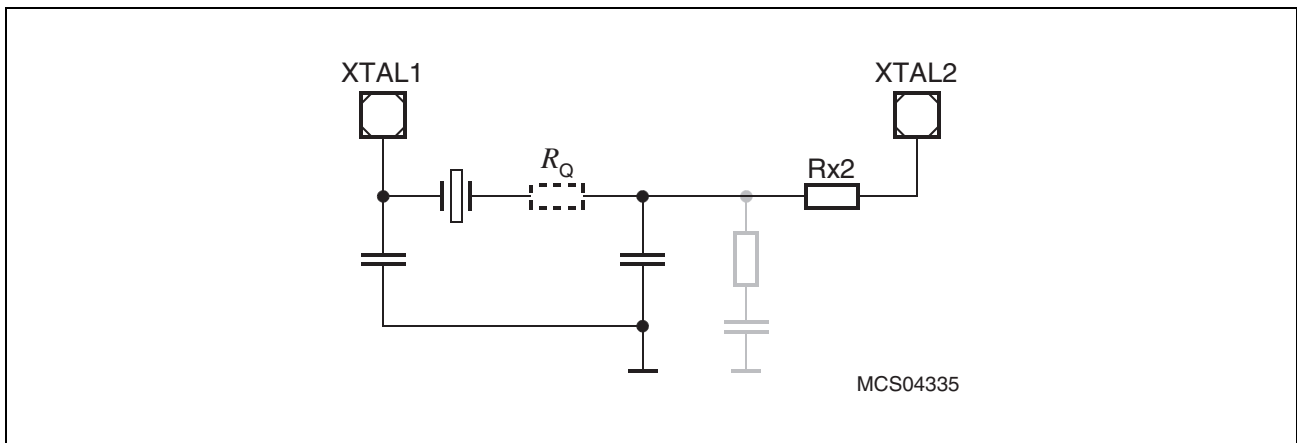
The CPU clock is distributed via separate clock drivers which feed the CPU itself and two groups of peripheral modules. The RTC is fed with the prescaled oscillator clock ( $f_{\text{RTC}}$ ) via a separate clock driver, so it is not affected by the clock control functions.



**Figure 6-1 CPU Clock Generation Stages**

## 6.1 Oscillators

The main oscillator of the C161CS/JC/JI is a power optimized Pierce oscillator providing an inverter and a feedback element. Pins XTAL1 and XTAL2 connect the inverter to the external crystal. The standard external oscillator circuitry (see [Figure 6-2](#)) comprises the crystal, two low end capacitors and series resistor ( $R_{x2}$ ) to limit the current through the crystal. The additional LC combination is only required for 3<sup>rd</sup> overtone crystals to suppress oscillation in the fundamental mode. A test resistor ( $R_Q$ ) may be temporarily inserted to measure the oscillation allowance of the oscillator circuitry.



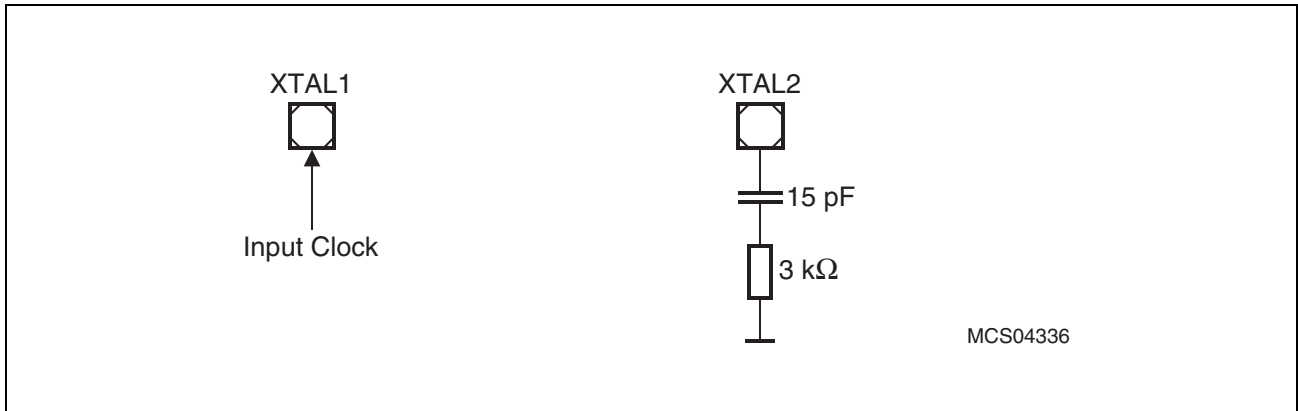
**Figure 6-2 External (Main) Oscillator Circuitry**

The on-chip oscillator is optimized for an input frequency range of 4 to 40 MHz.

An external clock signal (e.g. from an external oscillator or from a master device) may be fed to the input XTAL1. The Pierce oscillator then is not required to support the oscillation itself but is rather driven by the input signal. In this case the input frequency range may be 0 to 50 MHz (please note that the maximum applicable input frequency is limited by the device's maximum CPU frequency).

*Note: **Oscillator measurement** within the final target system is recommended to determine the actual oscillation allowance for the oscillator-crystal system. The measurement technique, examples for evaluated systems, and recommendations are provided in a specific application note about oscillators (available via your representative or [www](#)).*

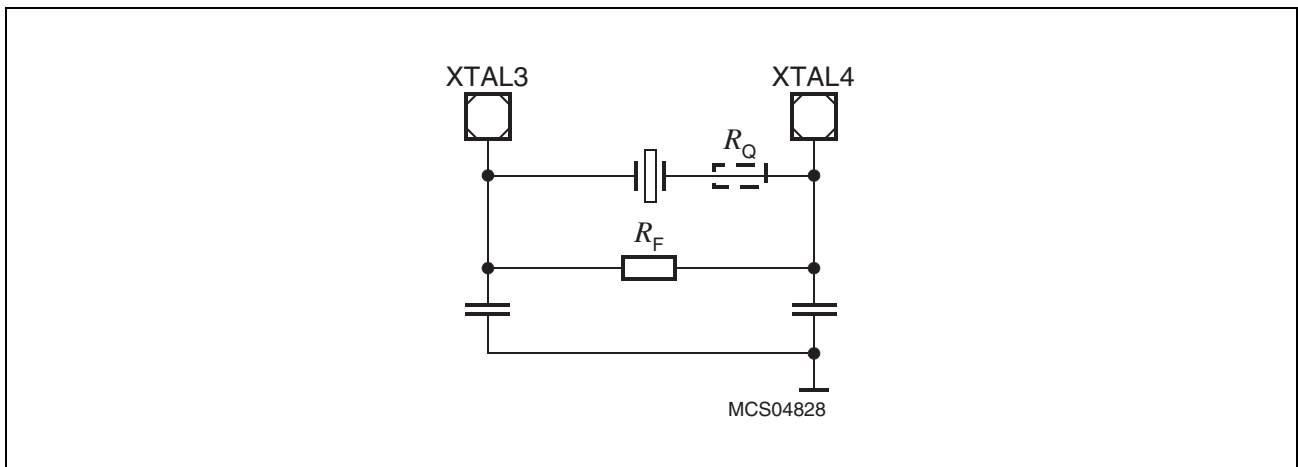
For input frequencies above 25 ... 30 MHz the oscillator's output should be terminated as shown in **Figure 6-3**, at lower frequencies it may be left open. This termination improves the operation of the oscillator by filtering out frequencies above the intended oscillator frequency.



**Figure 6-3 Oscillator Output Termination**

*Note: It is strongly recommended to measure the oscillation allowance (or margin) in the final target system (layout) to determine the optimum parameters for the oscillator operation.*

The **auxiliary oscillator** of the C161CS/JC/JI is a Pierce oscillator which is highly optimized for operation at a frequency of approximately 32 kHz. This narrow band optimization allows an extremely low power consumption of the auxiliary oscillator. Pins XTAL3 and XTAL4 connect the inverter to the external crystal. The recommendations given for the main oscillator apply accordingly.



**Figure 6-4 External (Auxiliary) Oscillator Circuitry**

The power consumption of the auxiliary oscillator is mainly influenced by the feedback resistance ( $R_F$ ). The feedback resistor is added externally in order to permit the usage of higher resistance values which result in a lower power consumption.

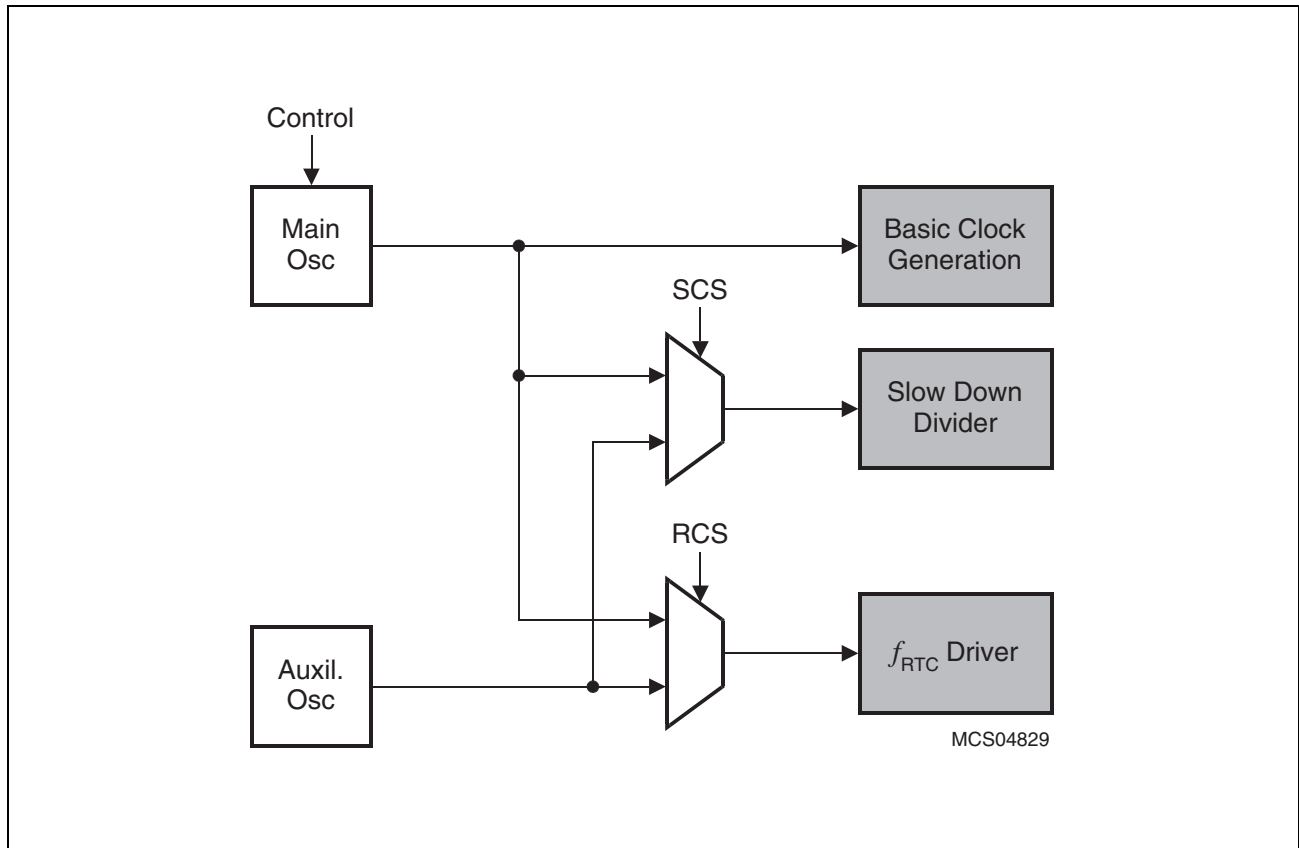
A typical range for  $R_F$  is 1 ... 50 M $\Omega$ , which equals a feedback current of 5 ... 0.1  $\mu$ A.

### Oscillator Selection

The input clock for the C161CS/JC/JI's clock system is derived from the main oscillator for the generation of the configured **basic clock**. The input clock for the Slow Down Divider may be derived either from the main oscillator or from the auxiliary oscillator.

Also the RTC may be clocked from either source.

Bits SCS (for the clocking system) and RCS (for the RTC) in register SYSCON2 select the active clock source. These bits are not changed by a reset so the selection remains valid after a warm reset.



**Figure 6-5 Oscillator Selection Mechanism**

*Note: When the auxiliary oscillator is not used, i.e. is not connected to an external clock signal or to a crystal, its input XTAL3 should be connected to GND.*

The main oscillator is automatically switched off (signal Control, see [Figure 6-5](#)) while none of the three potential consumers (Basic Clock Generation, Slow Down Divider, RTC Clock Driver, see [Figure 6-5](#)) is using its clock signal. This is the case if ...

- The Basic Clock Generation is off, because CPU clock is generated by SDD **AND**
- The Slow Down Divider is fed from the auxiliary oscillator (i.e. SCS = '1') **AND**
- The RTC Clock Driver is fed from the auxiliary oscillator (i.e. RCS = '1').

Switching off the main oscillator may be useful in order to further reduce the power consumption during phases where only minimum system life functions must be maintained

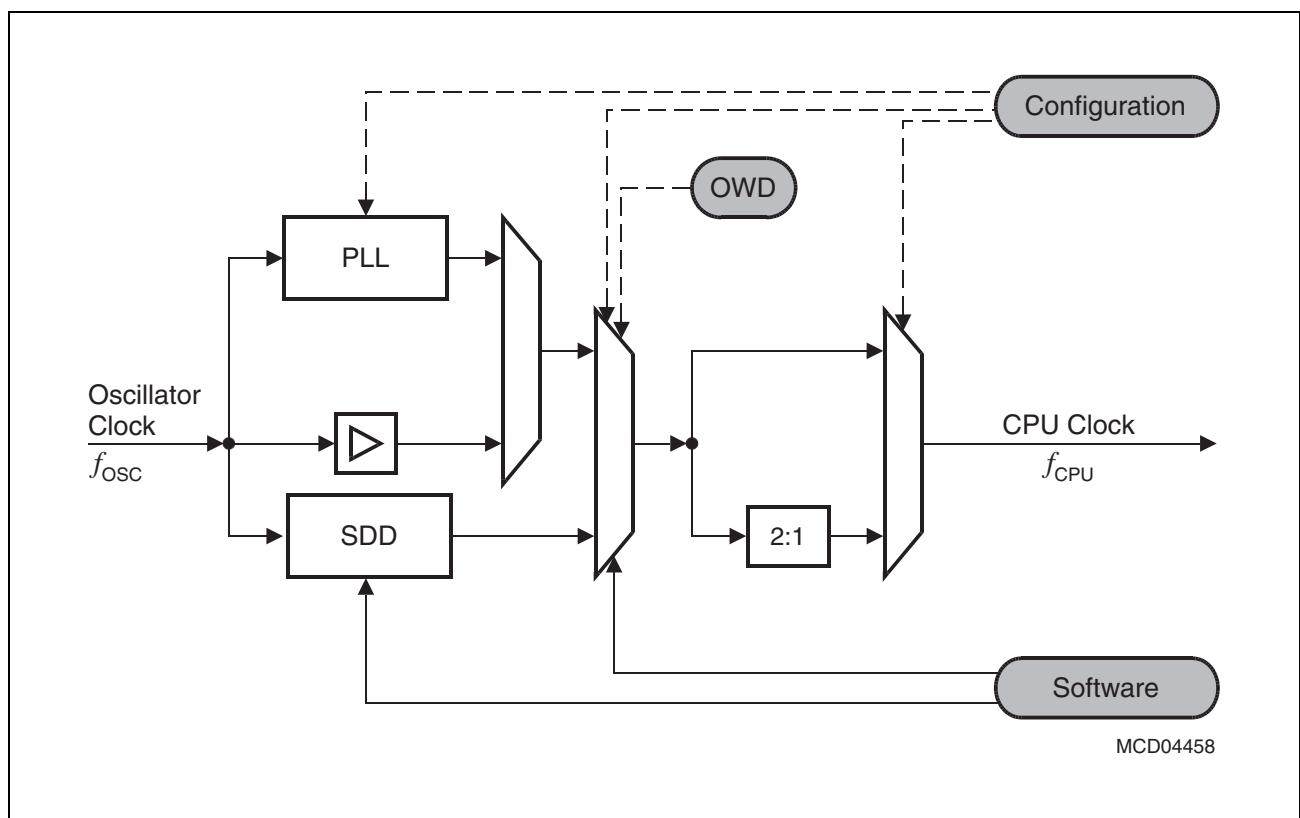
## 6.2 Frequency Control

The CPU clock is generated from the oscillator clock in either of two software selectable ways:

**The basic clock** is the standard operating clock for the C161CS/JC/JI and is required to deliver the intended maximum performance. The clock configuration in register RP0H (bitfield CLKCFG = RP0H.7-5) determines one of three possible basic clock generation modes:

- Direct Drive: the oscillator clock is directly fed to the controller hardware.
- Prescaler: the oscillator clock is divided by 2 to achieve a 50% duty cycle.
- PLL: the oscillator clock is multiplied by a configurable factor of  $F = 1.5 \dots 5$ .

**The Slow Down clock** is the oscillator clock divided by a programmable factor of 1 ... 32 (additional 2:1 divider in prescaler mode). This alternate possibility runs the C161CS/JC/JI at a lower frequency (depending on the programmed slow down factor) and thus greatly reduces its power consumption.

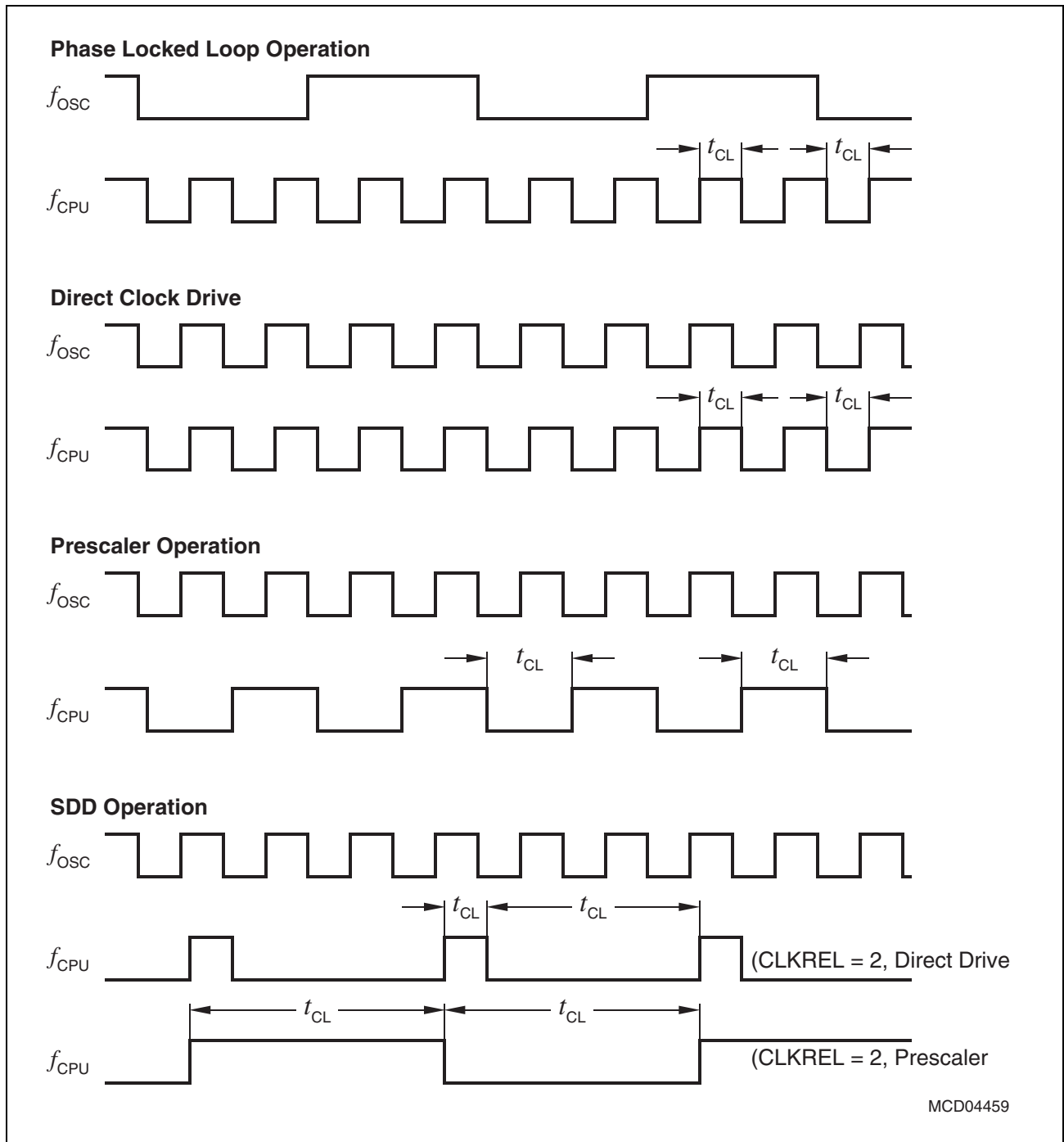


**Figure 6-6 Frequency Control Paths**

*Note: The configuration register RP0H is loaded with the logic levels present on the upper half of PORT0 (P0H) after a long hardware reset, i.e. bitfield CLKCFG represents the logic levels on pins P0.15-13 (P0H.7-5).*



The internal operation of the C161CS/JC/JI is controlled by the internal CPU clock  $f_{CPU}$ . Both edges of the CPU clock can trigger internal (e.g. pipeline) or external (e.g. bus cycles) operations (see [Figure 6-7](#)).



**Figure 6-7 Generation Mechanisms for the CPU Clock**

### Direct Drive

When direct drive is configured ( $\text{CLKCFG} = 011_{\text{B}}$ ) the C161CS/JC/JI's clock system is directly fed from the external clock input, i.e.  $f_{\text{CPU}} = f_{\text{OSC}}$ . This allows operation of the C161CS/JC/JI with a reasonably small fundamental mode crystal. The specified minimum values for the CPU clock phases (TCLs) must be respected. Therefore the maximum input clock frequency depends on the clock signal's duty cycle.

### Prescaler Operation

When prescaler operation is configured ( $\text{CLKCFG} = 001_{\text{B}}$ ) the C161CS/JC/JI's input clock is divided by 2 to generate then CPU clock signal, i.e.  $f_{\text{CPU}} = f_{\text{OSC}} / 2$ . This requires the oscillator (or input clock) to run on 2 times the intended operating frequency but guarantees a 50% duty cycle for the internal clock system independent of the input clock signal's waveform.

### PLL Operation

When PLL operation is configured (via  $\text{CLKCFG}$ ) the C161CS/JC/JI's input clock is fed to the on-chip phase locked loop circuit which multiplies its frequency by a factor of  $\mathbf{F} = 1.5 \dots 5$  (selectable via  $\text{CLKCFG}$ , see [Table 6-1](#)) and generates a CPU clock signal with 50% duty cycle, i.e.  $f_{\text{CPU}} = f_{\text{OSC}} \times \mathbf{F}$ .

The on-chip PLL circuit allows operation of the C161CS/JC/JI on a low frequency external clock while still providing maximum performance. The PLL also provides fail safe mechanisms which allow the detection of frequency deviations and the execution of emergency actions in case of an external clock failure.

When the PLL detects a missing input clock signal it generates an interrupt request. This warning interrupt indicates that the PLL frequency is no more locked, i.e. no more stable. This occurs when the input clock is unstable and especially when the input clock fails completely, e.g. due to a broken crystal. In this case the synchronization mechanism will reduce the PLL output frequency down to the PLL's base frequency (2 ... 5 MHz). The base frequency is still generated and allows the CPU to execute emergency actions in case of a loss of the external clock.

On power-up the PLL provides a stable clock signal within ca. 1 ms after  $V_{\text{DD}}$  has reached the specified valid range, even if there is no external clock signal (in this case the PLL will run on its base frequency of 2 ... 5 MHz). The PLL starts synchronizing with the external clock signal as soon as it is available. Within ca. 1 ms after stable oscillations of the external clock within the specified frequency range the PLL will be synchronous with this clock at a frequency of  $\mathbf{F} \times f_{\text{OSC}}$ , i.e. the PLL locks to the external clock.

When PLL operation is selected the CPU clock is a selectable multiple of the oscillator frequency, i.e. the input frequency.

[Table 6-1](#) lists the possible selections.

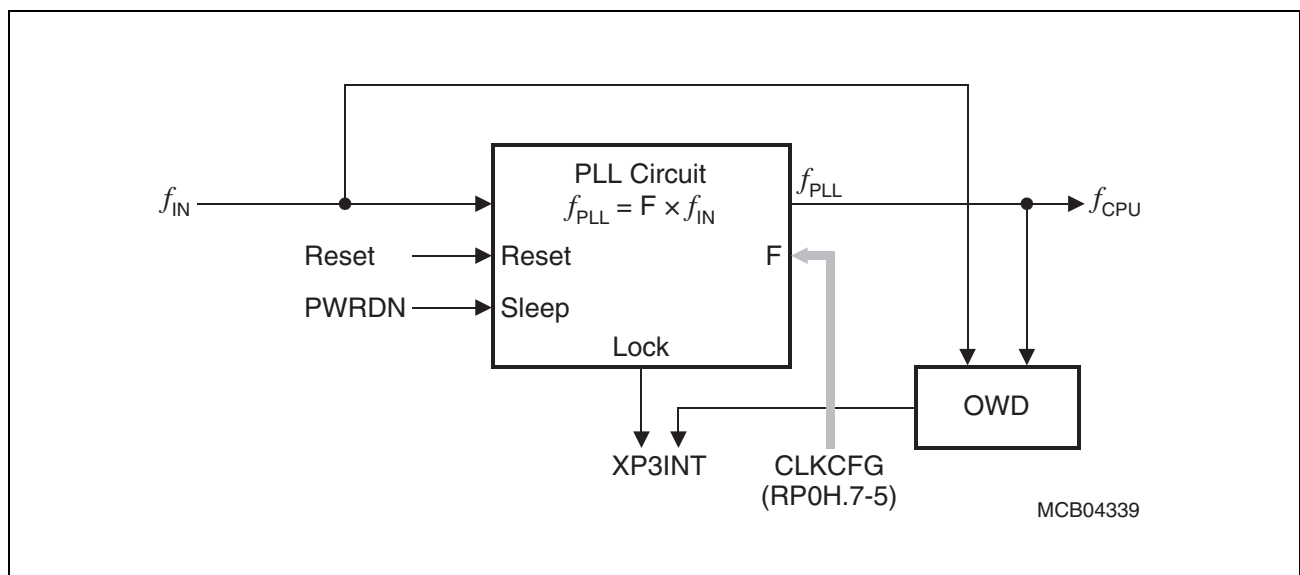
**Table 6-1 C161CS/JC/JI Clock Generation Modes**

RP0H.7-5 (P0H.7-5)	CPU Frequency $f_{CPU} = f_{OSC} \times F$	External Clock Input Range <sup>1)</sup>	Notes
1 1 1	$f_{OSC} \times 4$	2.5 to 8.25 MHz	Default configuration
1 1 0	$f_{OSC} \times 3$	3.33 to 11 MHz	–
1 0 1	$f_{OSC} \times 2$	5 to 16.5 MHz	–
1 0 0	$f_{OSC} \times 5$	2 to 6.6 MHz	–
0 1 1	$f_{OSC} \times 1$	1 to 33 MHz	Direct drive <sup>2)</sup>
0 1 0	$f_{OSC} \times 1.5$	6.66 to 22 MHz	–
0 0 1	$f_{OSC} / 2$	2 to 66 MHz	CPU clock via prescaler
0 0 0	$f_{OSC} \times 2.5$	4 to 13.2 MHz	–

<sup>1)</sup> The external clock input range refers to a CPU clock range of 10 ... 33 MHz.

<sup>2)</sup> The maximum frequency depends on the duty cycle of the external clock signal. In emulation mode pin P0.15 (P0H.7) is inverted, i.e. the configuration '111' would select direct drive in emulation mode.

The PLL constantly synchronizes to the external clock signal. Due to the fact that the external frequency is  $1/F$ 'th of the PLL output frequency the output frequency may be slightly higher or lower than the desired frequency. This jitter is irrelevant for longer time periods. For short periods (1 ... 4 CPU clock cycles) it remains below 4%.



**Figure 6-8 PLL Block Diagram**

### 6.3 Oscillator Watchdog

The C161CS/JC/JI provides an Oscillator Watchdog (OWD) which monitors the clock signal fed to input XTAL1 of the on-chip oscillator (either with a crystal or via external clock drive) in prescaler or direct drive mode (not if the PLL provides the basic clock). For this operation the PLL provides a clock signal (base frequency) which is used to supervise transitions on the oscillator clock. This PLL clock is independent from the XTAL1 clock. When the expected oscillator clock transitions are missing the OWD activates the PLL Unlock/OWD interrupt node and supplies the CPU with the PLL clock signal instead of the selected oscillator clock (see [Figure 6-6](#)). Under these circumstances the PLL will oscillate with its base frequency.

In direct drive mode the PLL base frequency is used directly ( $f_{\text{CPU}} = 2 \dots 5 \text{ MHz}$ ).

In prescaler mode the PLL base frequency is divided by 2 ( $f_{\text{CPU}} = 1 \dots 2.5 \text{ MHz}$ ).

If the oscillator clock fails while the PLL provides the basic clock the system will be supplied with the PLL base frequency anyway.

With this PLL clock signal the CPU can either execute a controlled shutdown sequence bringing the system into a defined and safe idle state, or it can provide an emergency operation of the system with reduced performance based on this (normally slower) emergency clock.

*Note: The CPU clock source is only switched back to the oscillator clock after a hardware reset.*

**The oscillator watchdog can be disabled** by setting bit OWDDIS in register SYSCON. In this case the PLL remains idle and provides no clock signal, while the CPU clock signal is derived directly from the oscillator clock or via prescaler or SDD. Also no interrupt request will be generated in case of a missing oscillator clock.

*Note: At the end of an external reset bit OWDDIS reflects the inverted level of pin  $\overline{RD}$  at that time. Thus the oscillator watchdog may also be disabled via hardware by (externally) pulling the  $\overline{RD}$  line low upon a reset, similar to the standard reset configuration via PORT0.*

**The oscillator watchdog cannot provide full security** while the CPU clock signal is generated by the SlowDown Divider, because the OWD cannot switch to the PLL clock in this case (see [Figure 6-6](#)). OWD interrupts are only recognizable if  $f_{\text{OSC}}$  is still available (e.g. input frequency too low or intermittent failure only).

A broken crystal cannot be detected by software (OWD interrupt server) as no SDD clock is available in such a case.

*Note: When the main oscillator is switched off the OWD should be disabled in any case in order to preserve the configured basic clock mode.*

## 6.4 Clock Drivers

The operating clock signal  $f_{CPU}$  is distributed to the controller hardware via several clock drivers which are disabled under certain circumstances. The real time clock RTC is clocked via a separate clock driver which delivers the prescaled oscillator clock (contrary to the other clock drivers). [Table 6-2](#) summarizes the different clock drivers and their function, especially in power reduction modes:

**Table 6-2 Clock Drivers Description**

<b>Clock Driver</b>	<b>Clock Signal</b>	<b>Active Mode</b>	<b>Idle Mode</b>	<b>Power Down and Sleep Mode</b>	<b>Connected Circuitry</b>
<b>CCD</b> CPU Clock Driver	$f_{CPU}$	ON	Off	Off	CPU, internal memory modules (IRAM, ROM/OTP/Flash)
<b>ICD</b> Interface Clock Driver	$f_{CPU}$	ON	ON	Off	ASC0, WDT, SSC, interrupt detection circuitry
<b>PCD</b> Peripheral Clock Driver	$f_{CPU}$	Control via PCDDIS	Control via PCDDIS	Off	(X)Peripherals (timers, etc.) except those driven by ICD, interrupt controller, ports
<b>RCD</b> RTC Clock Driver	$f_{RTC}$	ON	ON	Control via PDCON / SLEEP- CON	Realtime clock

*Note: Disabling PCD by setting bit PCDDIS stops the clock signal for all connected modules. Make sure that all these modules are in a safe state before stopping their clock signal.*

*The port input and output values will not change while PCD is disabled (ASC0 and SSC will still operate, if active),*

*CLKOUT will be high if enabled.*

*Please also respect the hints given in [Section 23.5](#).*

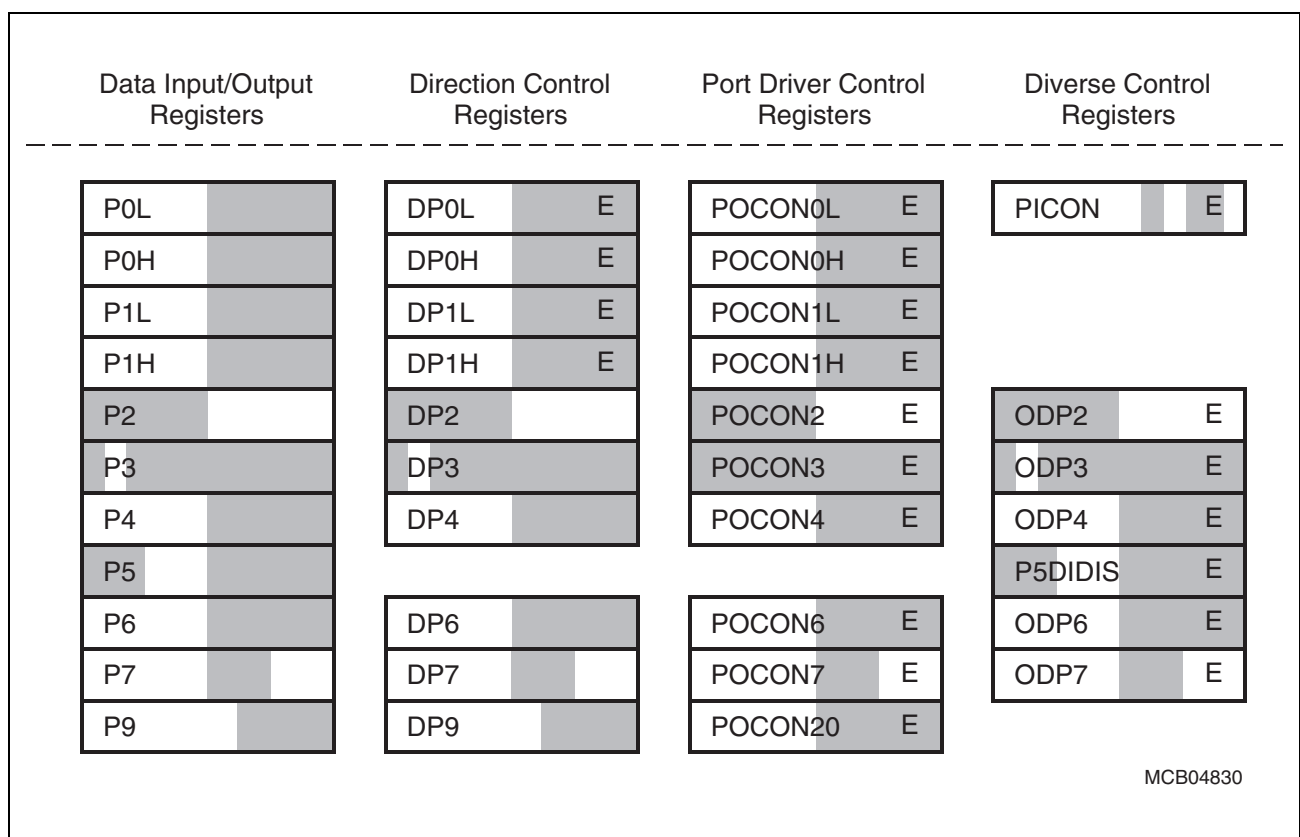
## 7 Parallel Ports

In order to accept or generate single external control signals or parallel data, the C161CS/JC/JI provides up to 93 parallel IO lines organized into seven 8-bit IO ports (PORT0 made of P0H and P0L, PORT1 made of P1H and P1L, Port 2, Port 4, Port 6), one 15-bit IO port (Port 3), one 12-bit input port (Port 5), one 4-bit IO port (Port 7), and one 6-bit open-drain IO port (Port 9).

These port lines may be used for general purpose Input/Output controlled via software or may be used implicitly by the C161CS/JC/JI's integrated peripherals or the External Bus Controller.

All port lines are bit addressable, and all input/output lines are individually (bit-wise) programmable as inputs or outputs via direction registers (except Port 5, of course). The IO ports are true bidirectional ports which are switched to high impedance state when configured as inputs. The output drivers of five IO ports (2, 3, 4, 6, 7) can be configured (pin by pin) for push/pull operation or open-drain operation via control registers. Port 9 provides open-drain-only drivers.

The logic level of a pin is clocked into the input latch once per state time, regardless whether the port is configured for input or output.



**Figure 7-1 SFRs and Pins associated with the Parallel Ports**

A write operation to a port pin configured as an input causes the value to be written into the port output latch, while a read operation returns the latched state of the pin itself. A read-modify-write operation reads the value of the pin, modifies it, and writes it back to the output latch.

Writing to a pin configured as an output ( $DP_{x.y} = '1'$ ) causes the output latch and the pin to have the written value, since the output buffer is enabled. Reading this pin returns the value of the output latch. A read-modify-write operation reads the value of the output latch, modifies it, and writes it back to the output latch, thus also modifying the level at the pin.

## 7.1 Input Threshold Control

The standard inputs of the C161CS/JC/JI determine the status of input signals according to TTL levels. In order to accept and recognize noisy signals, CMOS-like input thresholds can be selected instead of the standard TTL thresholds for all pins of specific ports. These special thresholds are defined above the TTL thresholds and feature a defined hysteresis to prevent the inputs from toggling while the respective input signal level is near the thresholds.

The Port Input Control register PICON allows to select these thresholds for each byte of the indicated ports, i.e. 8-bit ports are controlled by one bit each while 16-bit ports are controlled by two bits each.

### PICON

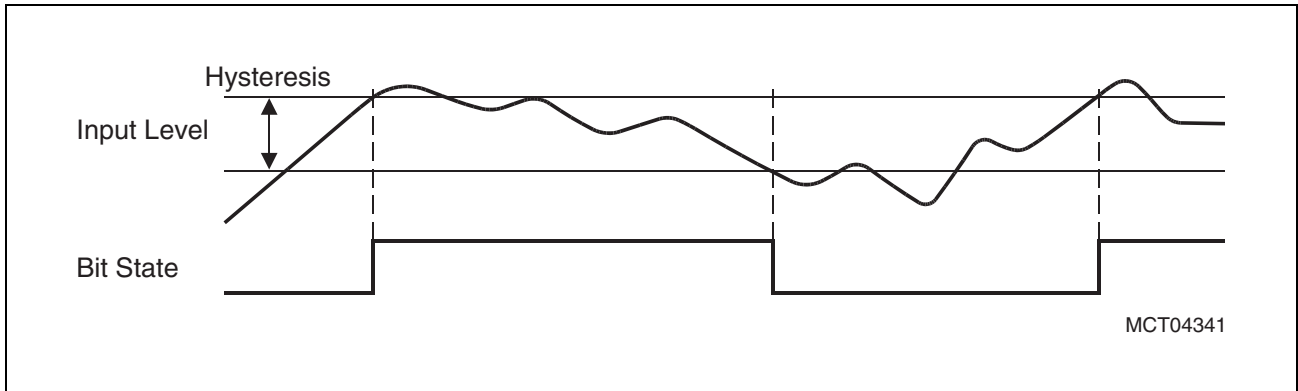
**Port Input Control Reg.**                      **SFR (F1C4<sub>H</sub>/E2<sub>H</sub>)**                      **Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								-	P7L IN	P6L IN	P4L IN	P3H IN	P3L IN	P2H IN	-
-	-	-	-	-	-	-	-	-	rW	rW	rW	rW	rW	rW	-

Bit	Function
<b>PxLIN</b>	<b>Port x Low Byte Input Level Selection</b> 0: Pins Px.7 ... Px.0 switch on standard TTL input levels 1: Pins Px.7 ... Px.0 switch on special threshold input levels
<b>PxHIN</b>	<b>Port x High Byte Input Level Selection</b> 0: Pins Px.15 ... Px.8 switch on standard TTL input levels 1: Pins Px.15 ... Px.8 switch on special threshold input levels

All options for individual direction and output mode control are available for each pin independent from the selected input threshold.

The input hysteresis provides stable inputs from noisy or slowly changing external signals.



**Figure 7-2 Hysteresis for Special Input Thresholds**



## 7.2 Output Driver Control

The output driver of a port pin is activated by switching the respective pin to output, i.e.  $DPx.y = '1'$ . The value that is driven to the pin is determined by the port output latch or by the associated alternate function (e.g. address, peripheral IO, etc.). The user software can control the characteristics of the output driver via the following mechanisms:

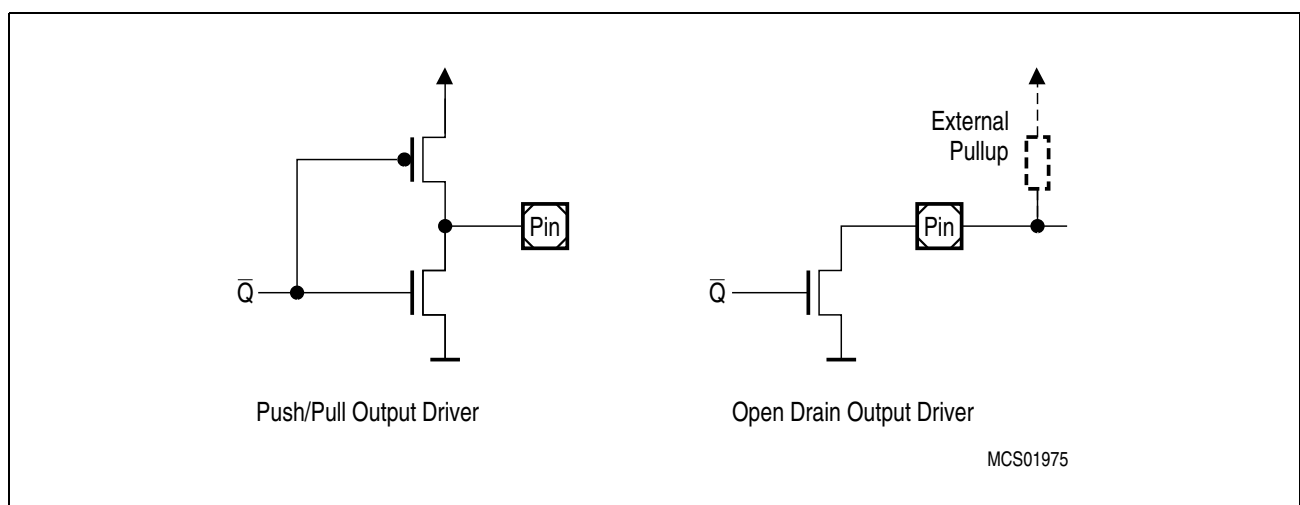
- **Open Drain Mode:** The upper (push) transistor is always disabled. Only '0' is driven actively, an external pullup is required.
- **Driver Characteristic:** The driver strength (static/dynamic) can be selected.
- **Edge Characteristic:** The rise/fall time of an output signal can be selected.

### Open Drain Mode

In the C161CS/JC/JI certain ports provide Open Drain Control, which allows to switch the output driver of a port pin from a push/pull configuration to an open drain configuration. In push/pull mode a port output driver has an upper and a lower transistor, thus it can actively drive the line either to a high or a low level. In open drain mode the upper transistor is always switched off, and the output driver can only actively drive the line to a low level. When writing a '1' to the port latch, the lower transistor is switched off and the output enters a high-impedance state. The high level must then be provided by an external pullup device. With this feature, it is possible to connect several port pins together to a Wired-AND configuration, saving external glue logic and/or additional software overhead for enabling/disabling output signals.

This feature is controlled through the respective Open Drain Control Registers ODPx which are provided for each port that has this feature implemented. These registers allow the individual bit-wise selection of the open drain mode for each port line.

If the respective control bit  $ODPx.y$  is '0' (default after reset), the output driver is in the push/pull mode. If  $ODPx.y$  is '1', the open drain configuration is selected. Note that all ODPx registers are located in the ESFR space.



**Figure 7-3 Output Drivers in Push/Pull Mode and in Open Drain Mode**

### Driver Characteristic

This defines either the general driving capability of the respective driver, or if the driver strength is reduced after the target output level has been reached or not. Reducing the driver strength increases the output's internal resistance which attenuates noise that is imported/exported via the output line. For driving LEDs or power transistors, however, a stable high output current may still be required.

The controllable output drivers of the C161CS/JC/JI pins feature two differently sized transistors (strong and weak) for each direction (push and pull). The time of activating/deactivating these transistors determines the output characteristics of the respective port driver.

Three modes can be selected to adapt the driver characteristics to the application's requirements:

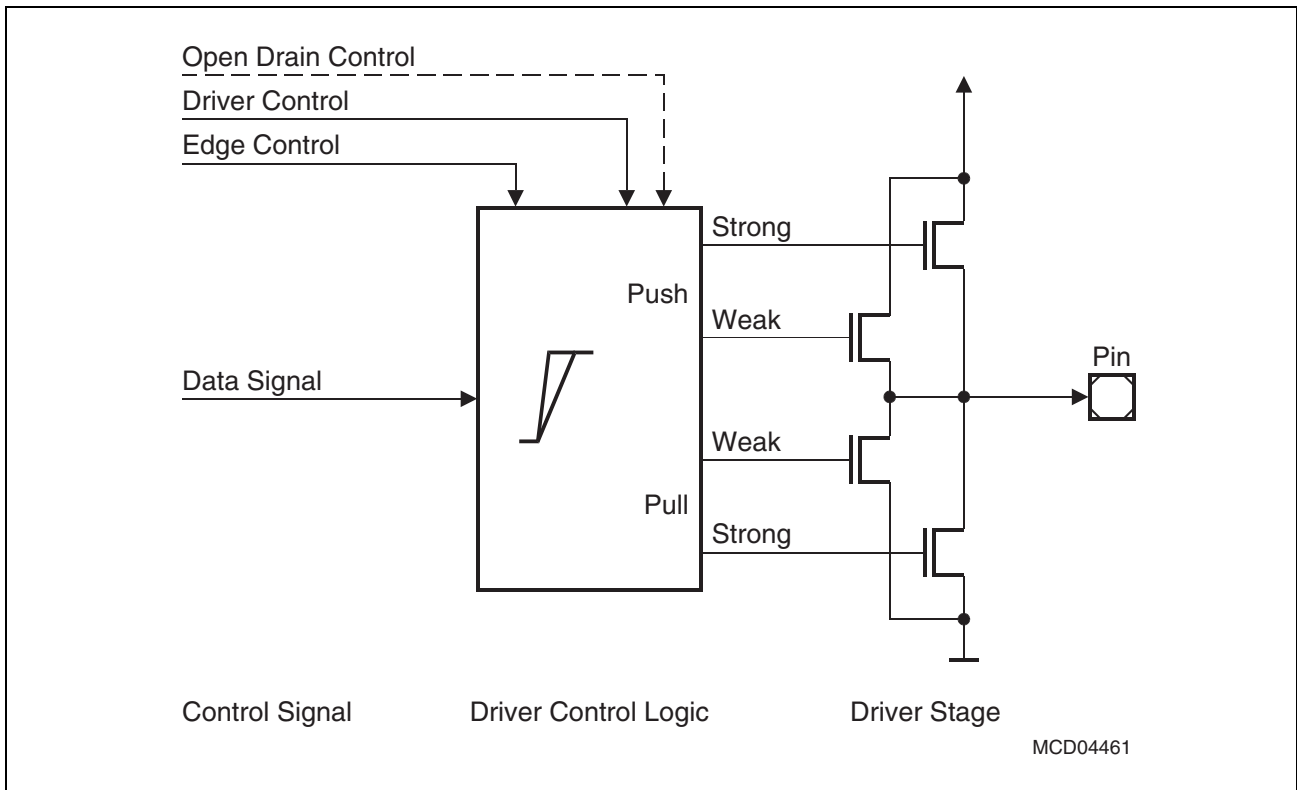
**In High Current Mode** both transistors are activated all the time. In this case the driver provides maximum output current even after the target signal level is reached.

**In Low Noise Mode** both transistors are activated at the beginning of a signal transition. When the target signal level is reached the driver strength is reduced by switching off the strong transistor. The weak transistor will keep the specified output level while the susceptibility for noise is reduced.

**In Low Current Mode** only the weak transistor is activated while the strong transistor remains off. This results in smooth transitions with low current peaks (and reduced susceptibility for noise) on the cost of increased transition times, i.e. slower edges, depending on the capacitive load.



### Edge Characteristic

This defines the rise/fall time for the respective output, i.e. the output transition time. Slow edges reduce the peak currents that are drawn when changing the voltage level of an external capacitive load. For a bus interface, however, fast edges may still be required. Edge characteristic effects the pre-driver which controls the final output driver stage.



**Figure 7-4 Structure of Two-Level Output Driver with Edge Control**

**Table 7-1 Output Transistor Operation**

Driver Mode		Low Current Mode		Dynamic Current Mode		High Current Mode	
		'0'	'1'	'0'	'1'	'0'	'1'
Push <sup>1)</sup> transistors	Strong	–	–	–		–	ON
	Weak	–	ON	–	ON	–	ON
Pull transistors	Strong	–	–		–	ON	–
	Weak	ON	–	ON	–	ON	–

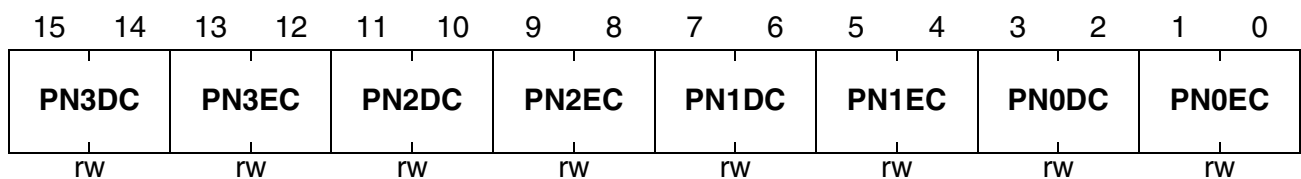
<sup>1)</sup> The upper (push) transistors are always off for output pins that operate in open drain mode.

The **Port Output Control registers** POCONx provide the corresponding control bits. For each feature (edge/driver characteristic and for each port nibble) a 2-bit control field is provided (i.e. 4 bits for each port nibble). Word ports consume four control nibbles each, byte ports consume two control nibbles each, where each control nibble controls 4 pins of the respective port.

The general register layout shown below is valid for all POCON registers. Please note that for byte ports only two pairs of bitfields are provided (see [Table 7-2](#)).

**POCON\***

**Port Output Control Reg. \***      **ESFR (F0xx<sub>H</sub>/<sub>yy<sub>H</sub>)</sub>**      **Reset Value: 0000<sub>H</sub>**



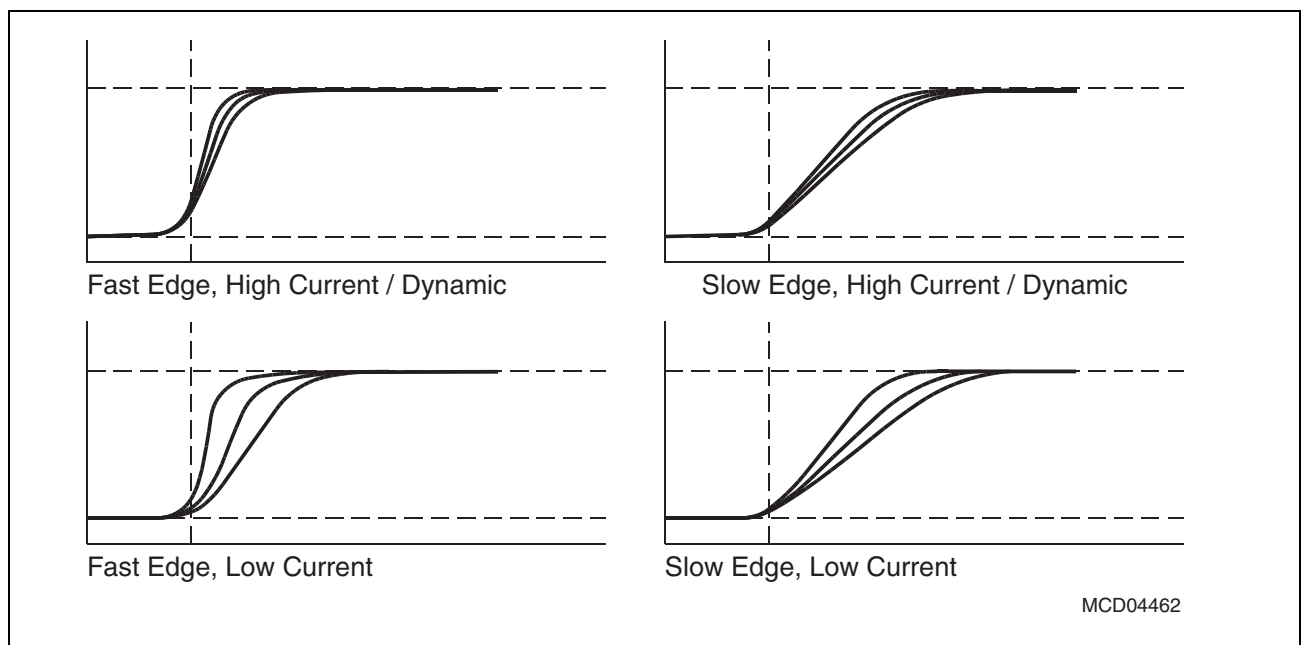
Bit	Function
<b>PNxEC</b>	<b>Port Nibble x Edge Characteristic</b> (Defines the output rise/fall time $t_{RF}$ ) 00: Fast edge mode, rise/fall times depend on the driver's dimensioning. 01: Reduced edge mode. 10: <i>Reserved.</i> 11: <i>Reserved.</i>
<b>PNxDC</b>	<b>Port Nibble x Driver Characteristic</b> (Defines the current delivered by the output) 00: High Current mode: Driver always operates with maximum strength. 01: Low Current mode: Driver always operates with reduced strength. 10: Dynamic Current mode: Driver strength is reduced after the target level has been reached. 11: <i>Reserved.</i>

[Table 7-2](#) lists the defined POCON registers and the allocation of control bitfields and port pins.

**Table 7-2 Port Output Control Register Allocation**

Control Register	Location	Controlled Pins (by POCONx.y-z)				Notes
		.15-12	.11-8	.7-4	.3-0	
POCON20	F0AA <sub>H</sub> / 55 <sub>H</sub>	$\overline{\text{RSTOUT}}$	CLKOUT / FOUT	ALE	$\overline{\text{WR}}$ , $\overline{\text{RD}}$ , $\overline{\text{BHE}}$ / $\overline{\text{WH}}$	No associated port
POCON7	F090 <sub>H</sub> / 48 <sub>H</sub>	---	---	P7.7-4	---	
POCON6	F08E <sub>H</sub> / 47 <sub>H</sub>	---	---	P6.7-4	P6.3-0	
POCON4	F08C <sub>H</sub> / 46 <sub>H</sub>	---	---	P4.7-4	P4.3-0	
POCON3	F08A <sub>H</sub> / 45 <sub>H</sub>	P3.15-12	P3.11-8	P3.7-4	P3.3-0	P3.14 is missing
POCON2	F088 <sub>H</sub> / 44 <sub>H</sub>	P2.15-12	P2.11-8	---	---	
POCON1H	F086 <sub>H</sub> / 43 <sub>H</sub>	---	---	P1H.7-4	P1H.3-0	
POCON1L	F084 <sub>H</sub> / 42 <sub>H</sub>	---	---	P1L.7-4	P1L.3-0	
POCON0H	F082 <sub>H</sub> / 41 <sub>H</sub>	---	---	P0H.7-4	P0H.3-0	
POCON0L	F080 <sub>H</sub> / 40 <sub>H</sub>	---	---	P0L.7-4	P0L.3-0	

**Figure 7-5** summarizes the effects of the driver characteristics: Edge characteristic generally influences the output signal's **shape**. Driver characteristic influences the signal shape's susceptibility to the external **capacitive load**.



**Figure 7-5 General Output Signal Waveforms**

### 7.3 Alternate Port Functions

In order to provide a maximum of flexibility for different applications and their specific IO requirements, port lines have programmable alternate input or output functions associated with them.

**Table 7-3 Summary of Alternate Port Functions**

Port	Alternate Function(s)	Alternate Signal(s)
PORT0	Address and data lines when accessing external resources (e.g. memory)	AD15 ... AD0
PORT1	Address lines when accessing ext. resources, Capture inputs or compare outputs of the CAPCOM units	A15 ... A0, CC27IO ... CC24IO
Port 2	Capture inputs or compare outputs of the CAPCOM units, CAPCOM timer input, Fast external interrupt inputs	CC15IO ... CC8IO, T7IN, EX7IN ... EX0IN
Port 3	System clock or programmable frequency output, Optional bus control signal, Input/output functions of serial interfaces, timers	CLKOUT/FOUT, $\overline{\text{BHE}}/\overline{\text{WRH}}$ , RxD0, TxD0, MTSR, MRST, SCLK, T2IN, T3IN, T4IN, T3EUD, T3OUT, CAPIN, T6OUT/RxD1, T0IN/TxD1
Port 4	Selected segment address lines in systems with more than 64 KBytes of ext. resources, CAN interface(s) when assigned, SDLM interface when assigned	A23 ... A16,  CAN1_TxD, (C161CS/JC) CAN1_RxD, (C161CS/JC) CAN2_TxD, (C161CS) CAN2_RxD, (C161CS) SDL_TxD, (C161JC/JI) SDL_RxD, (C161JC/JI)
Port 5	Analog input channels to the A/D converter, Timer control signal inputs	AN15 ... AN12, AN7 ... AN0, T2EUD, T4EUD, T5IN, T6IN
Port 6	Bus arbitration signals, Chip select output signals	$\overline{\text{BREQ}}$ , $\overline{\text{HLDA}}$ , $\overline{\text{HOLD}}$ , CS4 ... CS0
Port_7	Capture inputs or compare outputs of the CAPCOM units	CC31IO ... CC28IO
Port 9	IIC interface lines	SDA2, SCL1, SDA1, SCL0, SDA0

If an **alternate output function** of a pin is to be used, the direction of this pin must be programmed for output ( $DP_{x.y} = '1'$ ), except for some signals that are used directly after reset and are configured automatically. Otherwise the pin remains in the high-impedance state and is not effected by the alternate output function. The respective port latch should hold a '1', because its output is combined with the alternate output data. X-Peripherals (peripherals connected to the on-chip XBUS) control their associated IO pins directly via separate control lines.

If an **alternate input function** of a pin is used, the direction of the pin must be programmed for input ( $DP_{x.y} = '0'$ ) if an external device is driving the pin. The input direction is the default after reset. If no external device is connected to the pin, however, one can also set the direction for this pin to output. In this case, the pin reflects the state of the port output latch. Thus, the alternate input function reads the value stored in the port output latch. This can be used for testing purposes to allow a software trigger of an alternate input function by writing to the port output latch.

On most of the port lines, the user software is responsible for setting the proper direction when using an alternate input or output function of a pin. This is done by setting or clearing the direction control bit  $DP_{x.y}$  of the pin before enabling the alternate function. There are port lines, however, where the direction of the port line is switched automatically. For instance, in the multiplexed external bus modes of PORT0, the direction must be switched several times for an instruction fetch in order to output the addresses and to input the data. Obviously, this cannot be done through instructions. In these cases, the direction of the port line is switched automatically by hardware if the alternate function of such a pin is enabled.

To determine the appropriate level of the port output latches check how the alternate data output is combined with the respective port latch output.

There is one basic structure for all port lines with only an alternate input function. Port lines with only an alternate output function, however, have different structures due to the way the direction of the pin is switched and depending on whether the pin is accessible by the user software or not in the alternate function mode.

All port lines that are not used for these alternate functions may be used as general purpose IO lines. When using port pins for general purpose output, the initial output value should be written to the port latch prior to enabling the output drivers, in order to avoid undesired transitions on the output pins.

This applies to single pins as well as to pin groups (see examples below).

OUTPUT\_ENABLE\_SINGLE\_PIN:

```
BSET    P4.0                ;Initial output level is 'high'  
BSET    DP4.0              ;Switch on the output driver
```

OUTPUT\_ENABLE\_PIN\_GROUP:

```
BFLDL   P4, #05H, #05H     ;Initial output level is 'high'  
BFLDL   DP4, #05H, #05H   ;Switch on the output drivers
```

*Note: When using several BSET pairs to control more pins of one port, these pairs must be separated by instructions, which do not reference the respective port (see [Section 4.2](#)).*

Each of these ports and the alternate input and output functions are described in detail in the following subsections.



## 7.4 PORT0

The two 8-bit ports P0H and P0L represent the higher and lower part of PORT0, respectively. Both halves of PORT0 can be written (e.g. via a PEC transfer) without effecting the other half.

If this port is used for general purpose IO, the direction of each line can be configured via the corresponding direction registers DP0H and DP0L.

### P0L

**PORT0 Low Register**

**SFR (FF00<sub>H</sub>/80<sub>H</sub>)**

**Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								<b>P0L</b> <b>.7</b>	<b>P0L</b> <b>.6</b>	<b>P0L</b> <b>.5</b>	<b>P0L</b> <b>.4</b>	<b>P0L</b> <b>.3</b>	<b>P0L</b> <b>.2</b>	<b>P0L</b> <b>.1</b>	<b>P0L</b> <b>.0</b>
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

### P0H

**PORT0 High Register**

**SFR (FF02<sub>H</sub>/81<sub>H</sub>)**

**Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								<b>P0H</b> <b>.7</b>	<b>P0H</b> <b>.6</b>	<b>P0H</b> <b>.5</b>	<b>P0H</b> <b>.4</b>	<b>P0H</b> <b>.3</b>	<b>P0H</b> <b>.2</b>	<b>P0H</b> <b>.1</b>	<b>P0H</b> <b>.0</b>
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
P0X.y	Port data register P0H or P0L bit y

**DP0L**

**P0L Direction Ctrl. Register      ESFR (F100<sub>H</sub>/80<sub>H</sub>)      Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								DP0L .7	DP0L .6	DP0L .5	DP0L .4	DP0L .3	DP0L .2	DP0L .1	DP0L .0
-	-	-	-	-	-	-	-	rW	rW	rW	rW	rW	rW	rW	rW

**DP0H**

**P0H Direction Ctrl. Register      ESFR (F102<sub>H</sub>/81<sub>H</sub>)      Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								DP0H .7	DP0H .6	DP0H .5	DP0H .4	DP0H .3	DP0H .2	DP0H .1	DP0H .0
-	-	-	-	-	-	-	-	rW	rW	rW	rW	rW	rW	rW	rW

Bit	Function
DP0X.y	<b>Port direction register DP0H or DP0L bit y</b> DP0X.y = 0: Port line P0X.y is an input (high-impedance) DP0X.y = 1: Port line P0X.y is an output

**Alternate Functions of PORT0**

When an external bus is enabled, PORT0 is used as data bus or address/data bus. Note that an external 8-bit demultiplexed bus only uses P0L, while P0H is free for IO (provided that no other bus mode is enabled).

PORT0 is also used to select the system startup configuration. During reset, PORT0 is configured to input, and each line is held high through an internal pullup device. Each line can now be individually pulled to a low level (see DC-level specifications in the respective Data Sheets) through an external pulldown device. A default configuration is selected when the respective PORT0 lines are at a high level. Through pulling individual lines to a low level, this default can be changed according to the needs of the applications.

The internal pullup devices are designed such that an external pulldown resistors (see Data Sheet specification) can be used to apply a correct low level. These external pulldown resistors can remain connected to the PORT0 pins also during normal operation, however, care has to be taken such that they do not disturb the normal function of PORT0 (this might be the case, for example, if the external resistor is too strong).

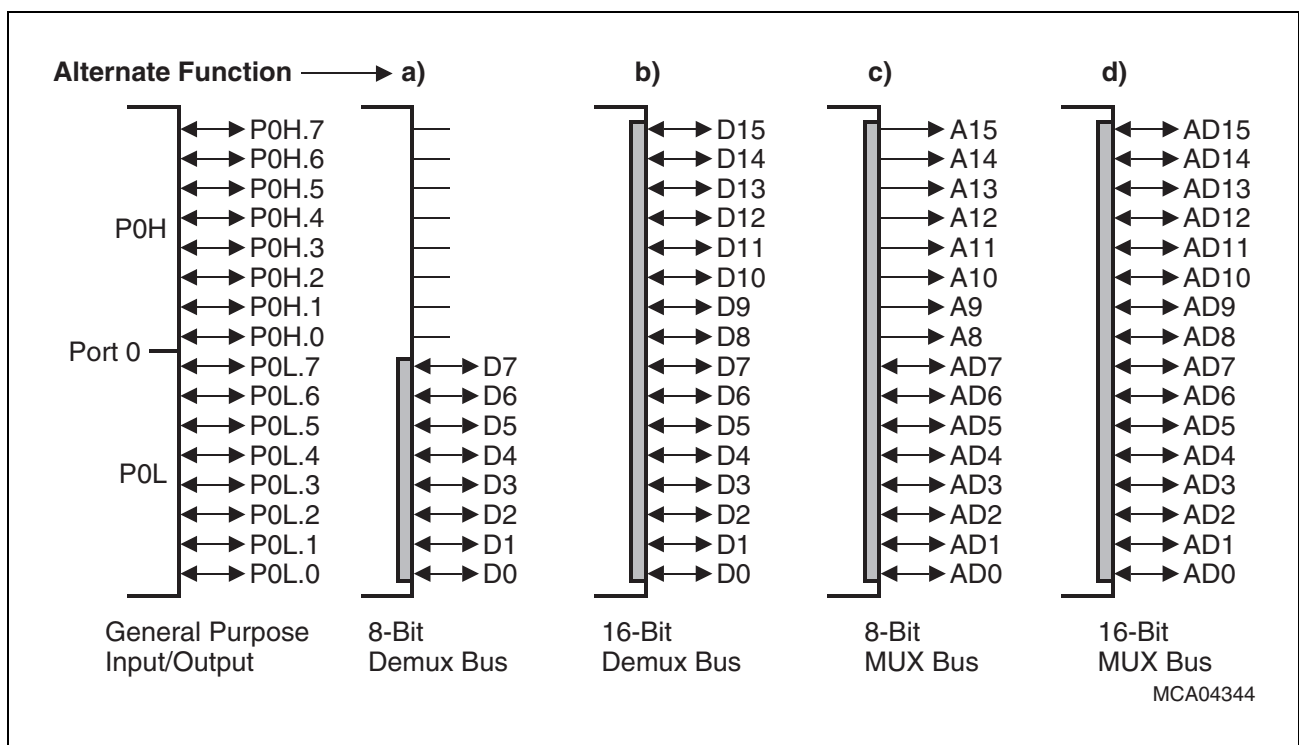
With the end of reset, the selected bus configuration will be written to the BUSCON0 register. The configuration of the high byte of PORT0 will be copied into the special register RP0H. This read-only register holds the selection for the number of chip selects

and segment addresses. Software can read this register in order to react according to the selected configuration, if required.

When the reset is terminated, the internal pullup devices are switched off, and PORT0 will be switched to the appropriate operating mode.

During external accesses in multiplexed bus modes PORT0 first outputs the 16-bit intra-segment address as an alternate output function. PORT0 is then switched to high-impedance input mode to read the incoming instruction or data. In 8-bit data bus mode, two memory cycles are required for word accesses, the first for the low byte and the second for the high byte of the word. During write cycles PORT0 outputs the data byte or word after outputting the address.

During external accesses in demultiplexed bus modes PORT0 reads the incoming instruction or data word or outputs the data byte or word.



**Figure 7-6 PORT0 IO and Alternate Functions**

While external bus cycles are executed, PORT0 is controlled by the bus controller. The port direction is determined by the type of the bus cycle, the data are transferred directly from/to the bus controller hardware. The alternate output data can be the 16-bit intrasegment address or the 8/16-bit data information. While PORT0 is not used by the bus controller, it is controlled by its direction and output latch registers. User software must therefore be very careful when writing to PORT0 registers while the external bus is enabled. In most cases keeping the reset values will be the best choice.

**Figure 7-7** shows the structure of a PORT0 pin.

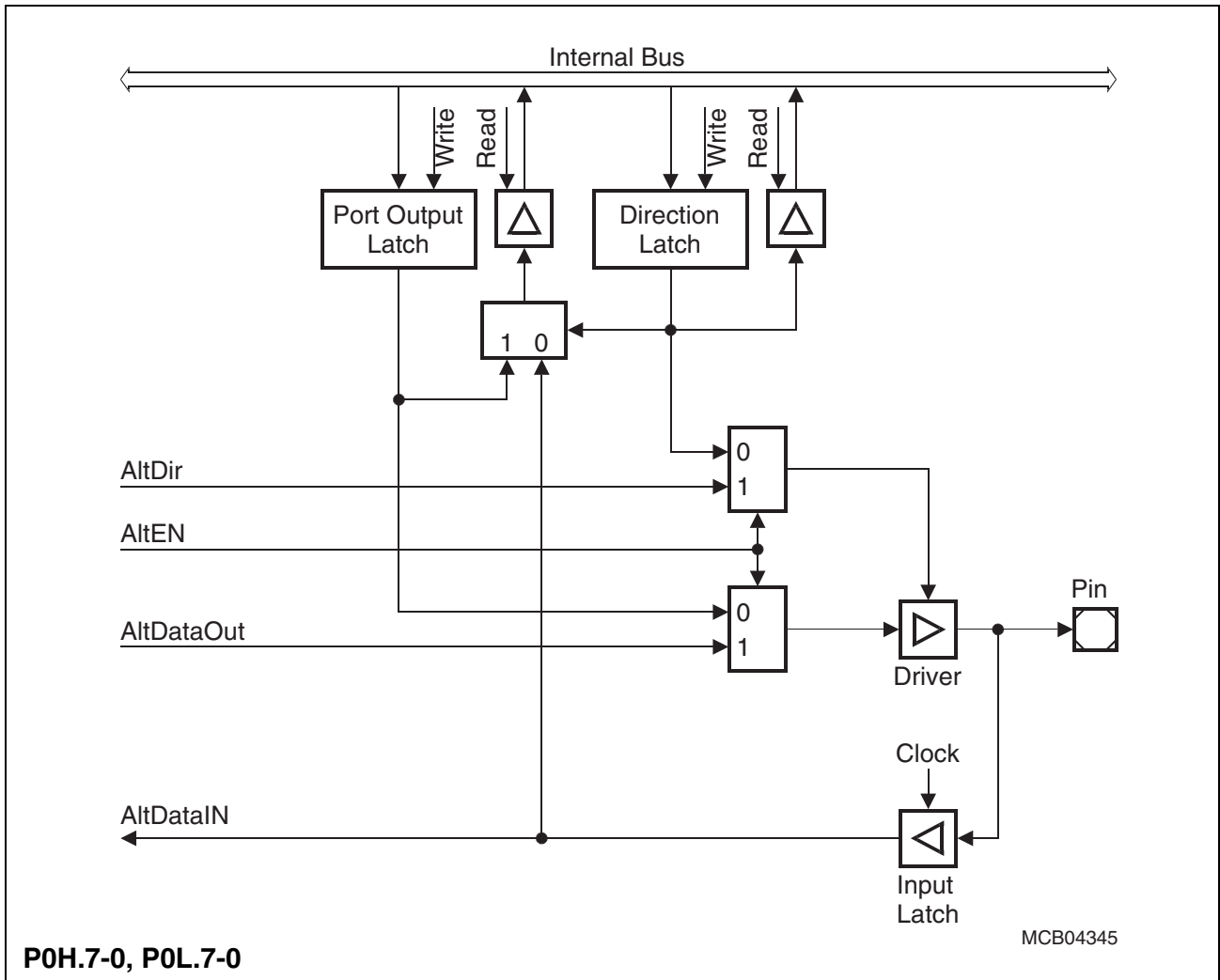


Figure 7-7 Block Diagram of a PORT0 Pin

## 7.5 PORT1

The two 8-bit ports P1H and P1L represent the higher and lower part of PORT1, respectively. Both halves of PORT1 can be written (e.g. via a PEC transfer) without effecting the other half.

If this port is used for general purpose IO, the direction of each line can be configured via the corresponding direction registers DP1H and DP1L.

### P1L

**PORT1 Low Register**

**SFR (FF04<sub>H</sub>/82<sub>H</sub>)**

**Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								<b>P1L .7</b>	<b>P1L .6</b>	<b>P1L .5</b>	<b>P1L .4</b>	<b>P1L .3</b>	<b>P1L .2</b>	<b>P1L .1</b>	<b>P1L .0</b>
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

### P1H

**PORT1 High Register**

**SFR (FF06<sub>H</sub>/83<sub>H</sub>)**

**Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								<b>P1H .7</b>	<b>P1H .6</b>	<b>P1H .5</b>	<b>P1H .4</b>	<b>P1H .3</b>	<b>P1H .2</b>	<b>P1H .1</b>	<b>P1H .0</b>
-	-	-	-	-	-	-	-	rwh	rwh	rwh	rwh	rw	rw	rw	rw

Bit	Function
P1X.y	Port data register P1H or P1L bit y

**DP1L**

**P1L Direction Ctrl. Register      ESFR (F104<sub>H</sub>/82<sub>H</sub>)      Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								DP1 L.7	DP1 L.6	DP1 L.5	DP1 L.4	DP1 L.3	DP1 L.2	DP1 L.1	DP1 L.0
-	-	-	-	-	-	-	-	rW	rW	rW	rW	rW	rW	rW	rW

**DP1H**

**P1H Direction Ctrl. Register      ESFR (F106<sub>H</sub>/83<sub>H</sub>)      Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								DP1 H.7	DP1 H.6	DP1 H.5	DP1 H.4	DP1 H.3	DP1 H.2	DP1 H.1	DP1 H.0
-	-	-	-	-	-	-	-	rW	rW	rW	rW	rW	rW	rW	rW

Bit	Function
DP1X.y	<b>Port direction register DP1H or DP1L bit y</b> DP1X.y = 0: Port line P1X.y is an input (high-impedance) DP1X.y = 1: Port line P1X.y is an output

**Alternate Functions of PORT1**

When a demultiplexed external bus is enabled, PORT1 is used as address bus. Note that demultiplexed bus modes use PORT1 as a 16-bit port. Otherwise all 16 port lines can be used for general purpose IO.

The upper four pins of PORT1 (P1H.7 ... P1H.4) also serve as capture inputs or compare outputs (CC27IO ... CC24IO) for the CAPCOM2 unit.

The usage of the port lines by the CAPCOM unit, its accessibility via software, and the precautions are the same as described for the Port 2 lines.

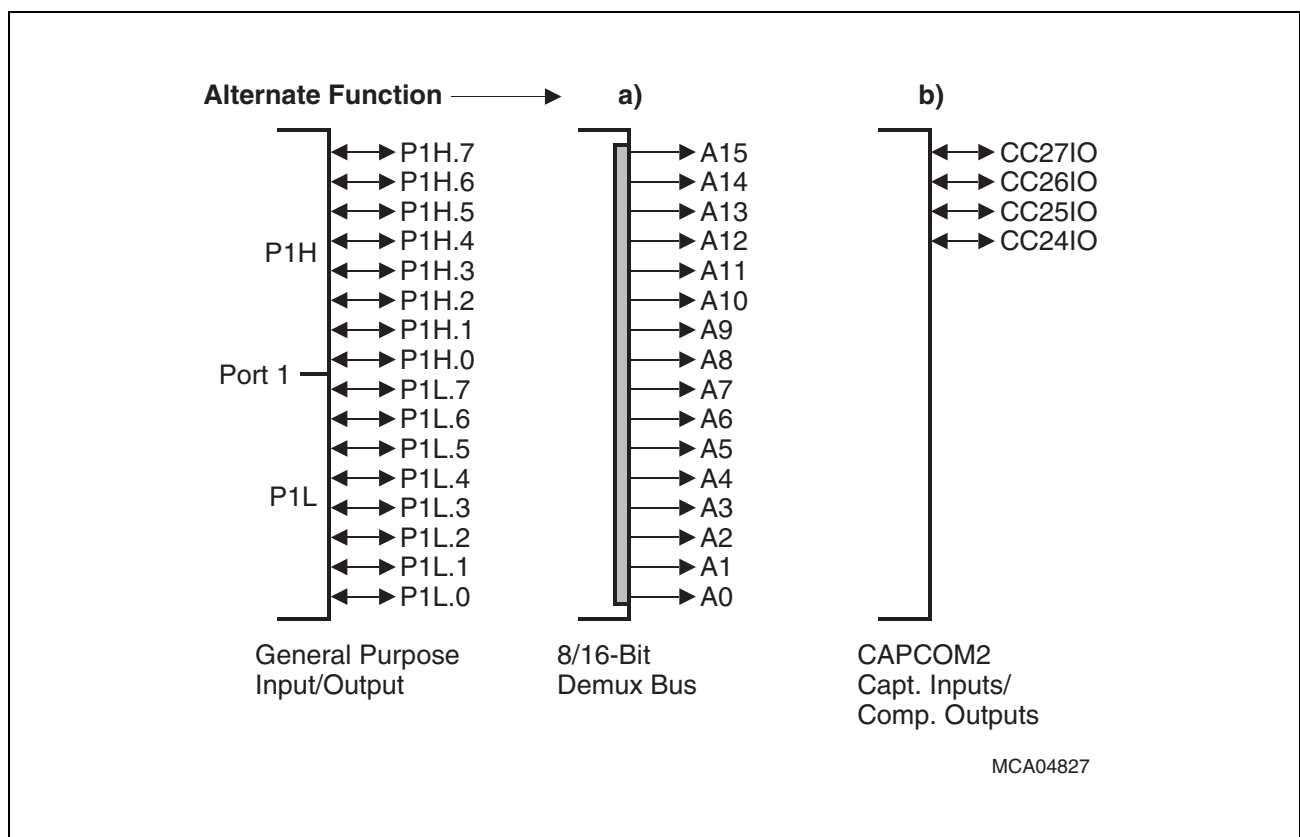
As all other capture inputs, the capture input function of pins P1H.7 ... P1H.4 can also be used as external interrupt inputs (sample rate 16 TCL).

As a side effect, the capture input capability of these lines can also be used in the address bus mode. Hereby changes of the upper address lines could be detected and trigger an interrupt request in order to perform some special service routines. External capture signals can only be applied if no address output is selected for PORT1.

During external accesses in demultiplexed bus modes PORT1 outputs the 16-bit intra-segment address as an alternate output function.

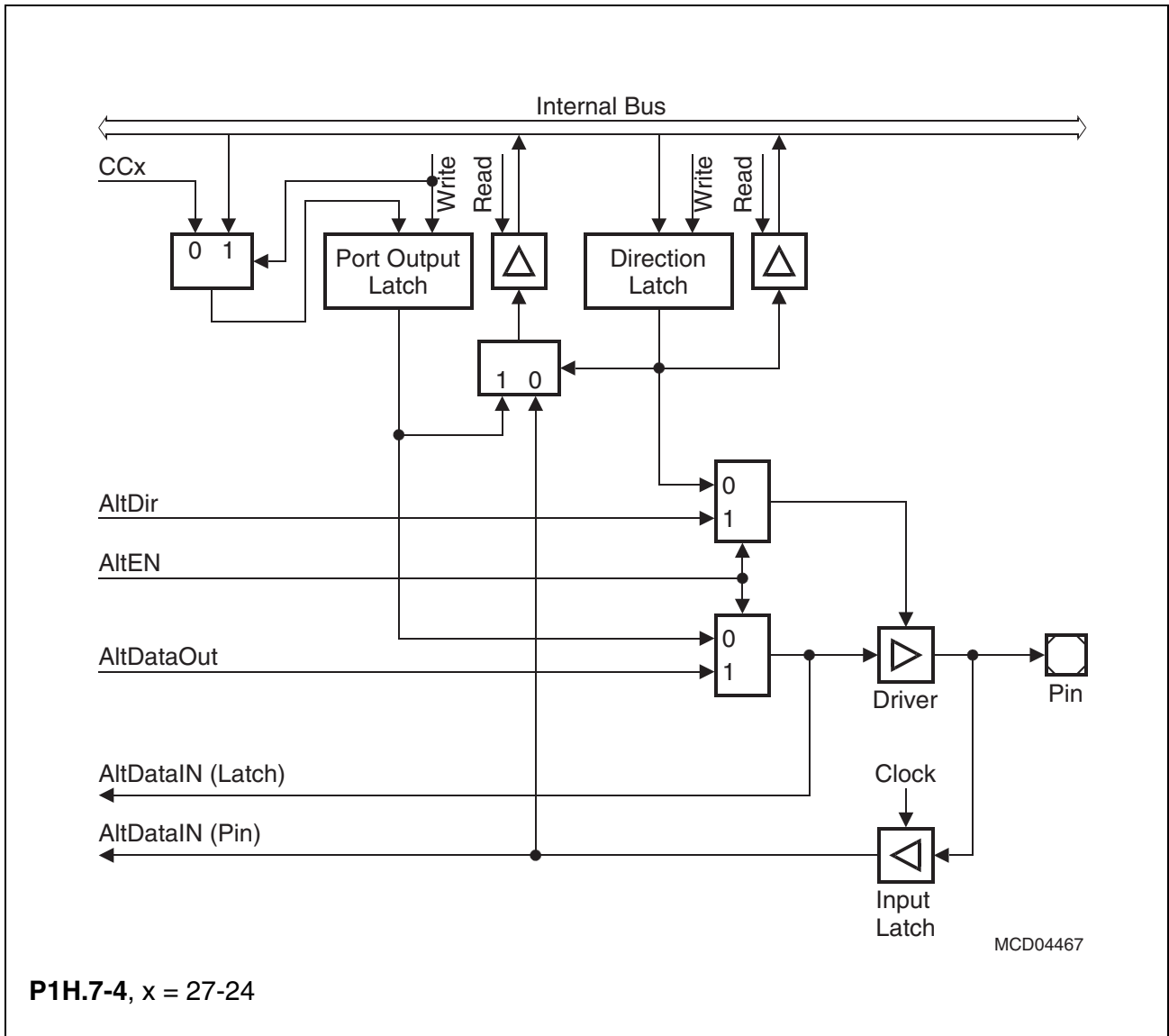
During external accesses in multiplexed bus modes, when **no** BUSCON register selects a demultiplexed bus mode, PORT1 is not used and is available for general purpose IO.

When an external bus mode is enabled, the direction of the port pin and the loading of data into the port output latch are controlled by the bus controller hardware. The input of the port output latch is disconnected from the internal bus and is switched to the line labeled “Alternate Data Output” via a multiplexer. The alternate data is the 16-bit intrasegment address. While an external bus mode is enabled, the user software should not write to the port output latch, otherwise unpredictable results may occur. When the external bus modes are disabled, the contents of the direction register last written by the user becomes active.



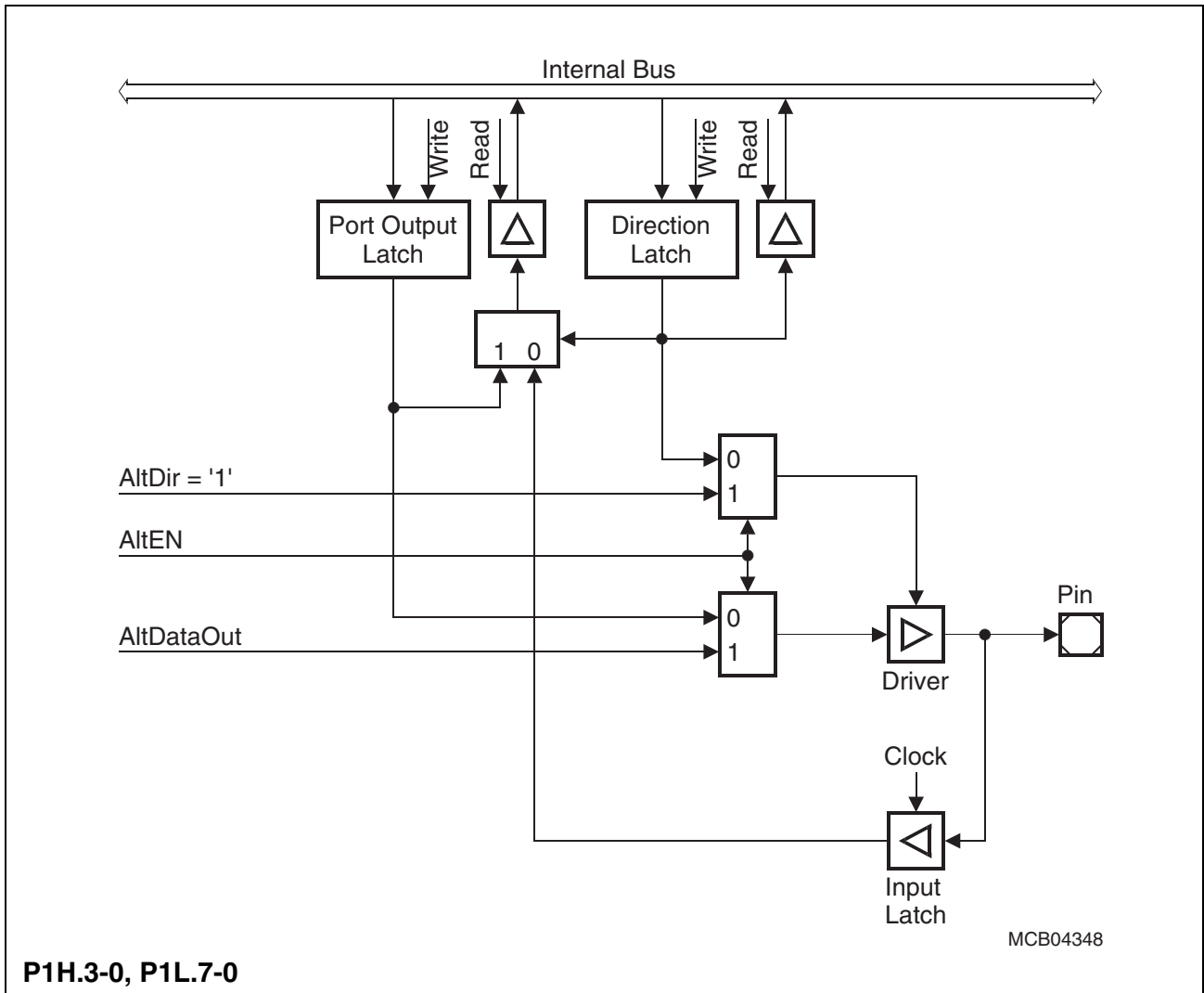
**Figure 7-8 PORT1 IO and Alternate Functions**

**Figure 7-9** shows the structure of PORT1 pins. The upper 4 pins of PORT1 combine internal bus data and alternate data output before the port latch input.



**Figure 7-9 Block Diagram of a PORT1 Pin with Address and CAPCOM Function**





P1H.3-0, P1L.7-0

**Figure 7-10 Block Diagram of a PORT1 Pin with Address Function**

## 7.6 Port 2

If this 8-bit port is used for general purpose IO, the direction of each line can be configured via the corresponding direction register DP2. Each port line can be switched into push/pull or open drain mode via the open drain control register ODP2.

### P2

**Port 2 Data Register**                      **SFR (FFC0<sub>H</sub>/E0<sub>H</sub>)**                      **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>P2</b> <b>.15</b>	<b>P2</b> <b>.14</b>	<b>P2</b> <b>.13</b>	<b>P2</b> <b>.12</b>	<b>P2</b> <b>.11</b>	<b>P2</b> <b>.10</b>	<b>P2.9</b>	<b>P2.8</b>	-	-	-	-	-	-	-	-
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	-	-	-	-	-	-	-	-

Bit	Function
<b>P2.y</b>	<b>Port data register P2 bit y</b>

### DP2

**P2 Direction Ctrl. Register**                      **SFR (FFC2<sub>H</sub>/E1<sub>H</sub>)**                      **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>DP2</b> <b>.15</b>	<b>DP2</b> <b>.14</b>	<b>DP2</b> <b>.13</b>	<b>DP2</b> <b>.12</b>	<b>DP2</b> <b>.11</b>	<b>DP2</b> <b>.10</b>	<b>DP2</b> <b>.9</b>	<b>DP2</b> <b>.8</b>	-	-	-	-	-	-	-	-
rw	rw	rw	rw	rw	rw	rw	rw	-	-	-	-	-	-	-	-

Bit	Function
<b>DP2.y</b>	<b>Port direction register DP2 bit y</b> DP2.y = 0: Port line P2.y is an input (high-impedance) DP2.y = 1: Port line P2.y is an output

**ODP2**

**P2 Open Drain Ctrl. Reg.**

**ESFR (F1C2<sub>H</sub>/E1<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODP2 .15	ODP2 .14	ODP2 .13	ODP2 .12	ODP2 .11	ODP2 .10	ODP2 .9	ODP2 .8	-	-	-	-	-	-	-	-
rw	rw	rw	rw	rw	rw	rw	rw	-	-	-	-	-	-	-	-

Bit	Function
ODP2.y	<p><b>Port 2 Open Drain control register bit y</b></p> <p>ODP2.y = 0: Port line P2.y output driver in push/pull mode</p> <p>ODP2.y = 1: Port line P2.y output driver in open drain mode</p>

### **Alternate Functions of Port 2**

All Port 2 lines (P2.15 ... P2.8) serve as capture inputs or compare outputs (CC15IO ... CC8IO) for the CAPCOM1 unit, or as external interrupt inputs EX7IN ... EX0IN (16 TCL sample rate).

When a Port 2 line is used as a capture input, the state of the input latch, which represents the state of the port pin, is directed to the CAPCOM unit via the line "Alternate Pin Data Input". If an external capture trigger signal is used, the direction of the respective pin must be set to input. If the direction is set to output, the state of the port output latch will be read since the pin represents the state of the output latch. This can be used to trigger a capture event through software by setting or clearing the port latch. Note that in the output configuration, no external device may drive the pin, otherwise conflicts would occur.

When a Port 2 line is used as a compare output (compare modes 1 and 3), the compare event (or the timer overflow in compare mode 3) directly effects the port output latch. In compare mode 1, when a valid compare match occurs, the state of the port output latch is read by the CAPCOM control hardware via the line "Alternate Latch Data Input", inverted, and written back to the latch via the line "Alternate Data Output". The port output latch is clocked by the signal "Compare Trigger" which is generated by the CAPCOM unit. In compare mode 3, when a match occurs, the value '1' is written to the port output latch via the line "Alternate Data Output". When an overflow of the corresponding timer occurs, a '0' is written to the port output latch. In both cases, the output latch is clocked by the signal "Compare Trigger". The direction of the pin should be set to output by the user, otherwise the pin will be in the high-impedance state and will not reflect the state of the output latch.

As can be seen from the port structure below, the user software always has free access to the port pin even when it is used as a compare output. This is useful for setting up the initial level of the pin when using compare mode 1 or the double-register mode. In these modes, unlike in compare mode 3, the pin is not set to a specific value when a compare match occurs, but is toggled instead.

When the user wants to write to the port pin at the same time a compare trigger tries to clock the output latch, the write operation of the user software has priority. Each time a CPU write access to the port output latch occurs, the input multiplexer of the port output latch is switched to the line connected to the internal bus. The port output latch will receive the value from the internal bus and the hardware triggered change will be lost.

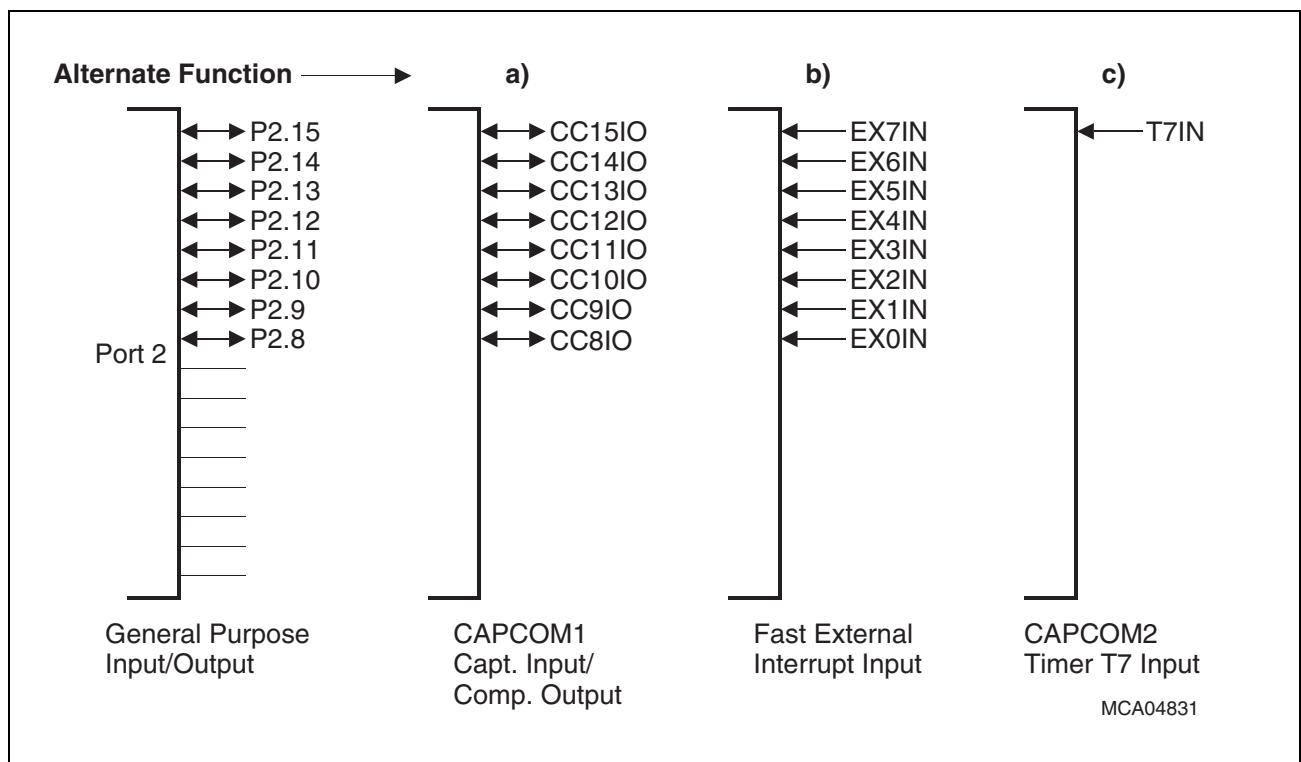
As all other capture inputs, the capture input function of pins P2.15 ... P2.8 can also be used as external interrupt inputs (sample rate 16 TCL) or as Fast External Interrupt inputs (sample rate 2 TCL).

P2.15 in addition serves as input for CAPCOM2 timer T7 (T7IN).

**Table 7-4** summarizes the alternate functions of Port 2.

**Table 7-4 Alternate Functions of Port 2**

Port 2 Pin	Alternate Function a)	Alternate Function b)	Alternate Function c)
P2.8	CC8IO	EX0IN Fast External Interrupt 0 Inp.	–
P2.9	CC9IO	EX1IN Fast External Interrupt 1 Inp.	–
P2.10	CC10IO	EX2IN Fast External Interrupt 2 Inp.	–
P2.11	CC11IO	EX3IN Fast External Interrupt 3 Inp.	–
P2.12	CC12IO	EX4IN Fast External Interrupt 4 Inp.	–
P2.13	CC13IO	EX5IN Fast External Interrupt 5 Inp.	–
P2.14	CC14IO	EX6IN Fast External Interrupt 6 Inp.	–
P2.15	CC15IO	EX7IN Fast External Interrupt 7 Inp.	T7IN Timer T7 Ext. Count Input



**Figure 7-11 Port 2 IO and Alternate Functions**

The pins of Port 2 combine internal bus data and alternate data output before the port latch input.

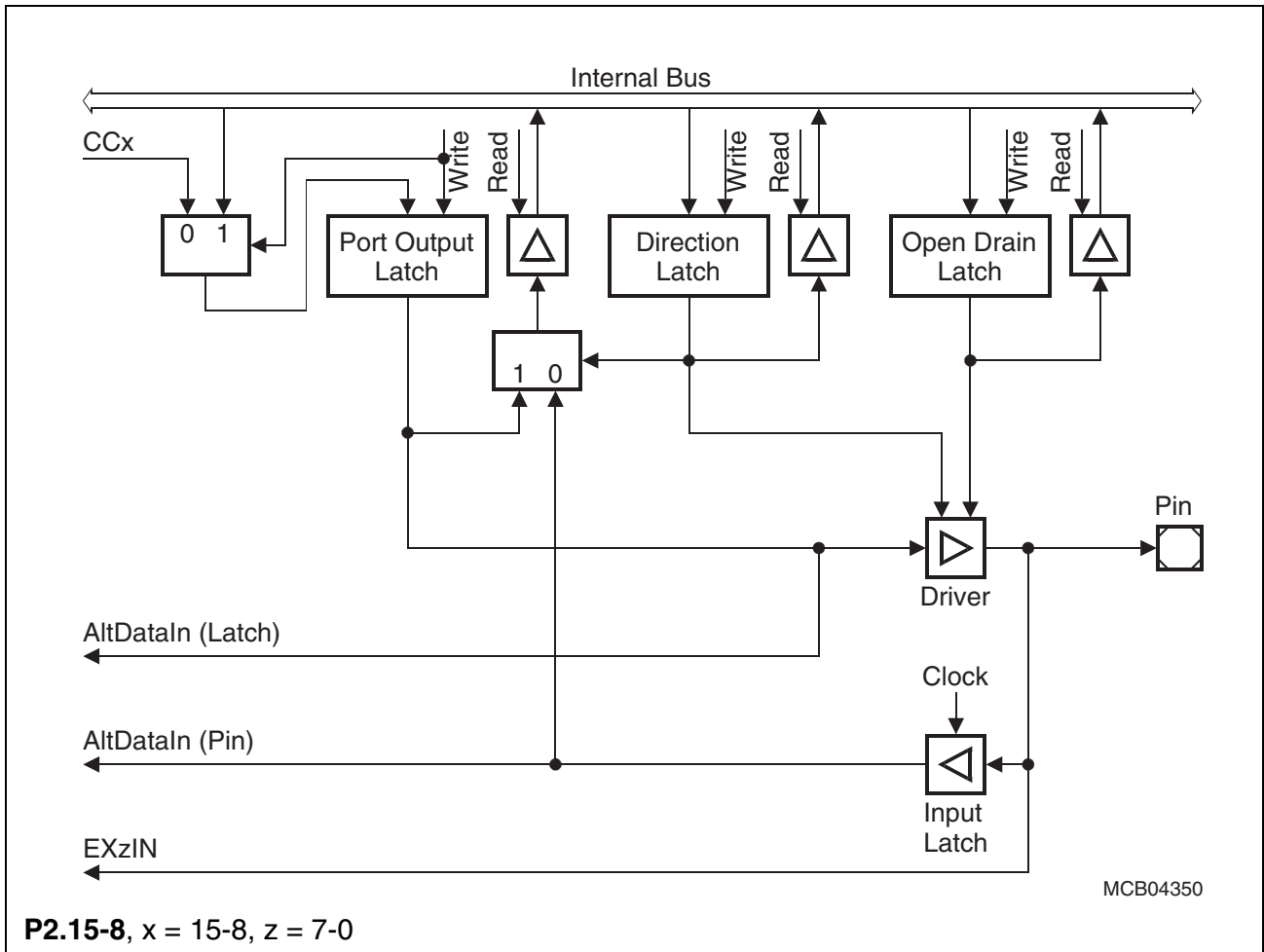


Figure 7-12 Block Diagram of a Port 2 Pin

## 7.7 Port 3

If this 15-bit port is used for general purpose IO, the direction of each line can be configured via the corresponding direction register DP3. Most port lines can be switched into push/pull or open drain mode via the open drain control register ODP3 (pins P3.15 and P3.12 do not support open drain mode!).

### P3

#### Port 3 Data Register

SFR (FFC4<sub>H</sub>/E2<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P3 .15	-	P3 .13	P3 .12	P3 .11	P3 .10	P3 .9	P3 .8	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
rw	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
P3.y	Port data register P3 bit y

### DP3

#### P3 Direction Ctrl. Register

SFR (FFC6<sub>H</sub>/E3<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DP3 .15	-	DP3 .13	DP3 .12	DP3 .11	DP3 .10	DP3 .9	DP3 .8	DP3 .7	DP3 .6	DP3 .5	DP3 .4	DP3 .3	DP3 .2	DP3 .1	DP3 .0
rw	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
DP3.y	Port direction register DP3 bit y DP3.y = 0: Port line P3.y is an input (high-impedance) DP3.y = 1: Port line P3.y is an output

**ODP3**

**P3 Open Drain Ctrl. Reg.**

**ESFR (F1C6<sub>H</sub>/E3<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	ODP 3.13	-	ODP 3.11	ODP 3.10	ODP 3.9	ODP 3.8	ODP 3.7	ODP 3.6	ODP 3.5	ODP 3.4	ODP 3.3	ODP 3.2	ODP 3.1	ODP 3.0
-	-	rw	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
ODP3.y	<p><b>Port 3 Open Drain control register bit y</b></p> <p>ODP3.y = 0: Port line P3.y output driver in push/pull mode</p> <p>ODP3.y = 1: Port line P3.y output driver in open drain mode</p>

*Note: Due to pin limitations register bit P3.14 is not connected to an IO pin.  
Pins P3.15 and P3.12 do not support open drain mode.*

**Alternate Functions of Port 3**

The pins of Port 3 serve for various functions which include external timer control lines, the two serial interfaces, and the control lines  $\overline{\text{BHE}}/\overline{\text{WRH}}$  and clock output.

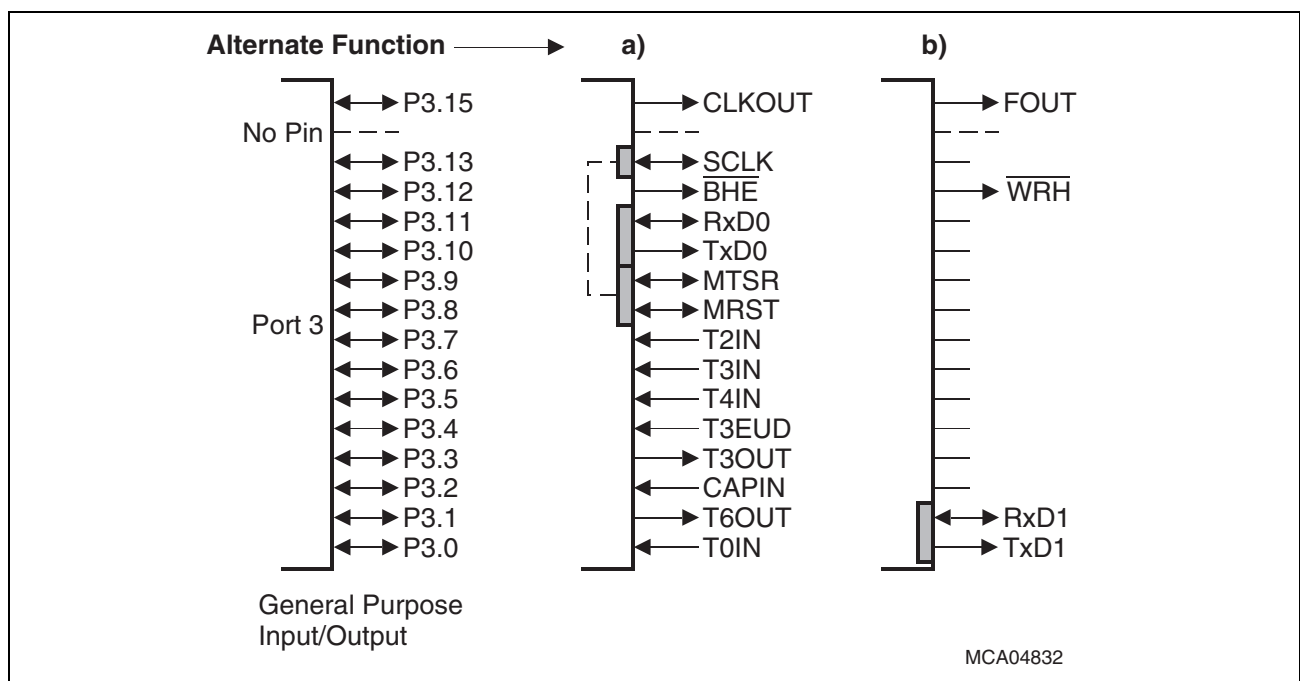
**Table 7-5** summarizes the alternate functions of Port 3.



**Table 7-5 Alternate Functions of Port 3**

Port 3 Pin	Alternate Function
P3.0	T0IN/ CAPCOM1 Timer T0 Count Input/ TxD1 ASC1 Transmit Data Output
P3.1	T6OUT/ GPT2 Timer T6 Toggle Latch Output/ RxD1 ASC1 Receive Data Input <sup>1)</sup>
P3.2	CAPIN GPT2 Capture Input
P3.3	T3OUT GPT1 Timer T3 Toggle Latch Output
P3.4	T3EUD GPT1 Timer T3 External Up/Down Input
P3.5	T4IN GPT1 Timer T4 Count Input
P3.6	T3IN GPT1 Timer T3 Count Input
P3.7	T2IN GPT1 Timer T2 Count Input
P3.8	MRST SSC Master Receive/Slave Transmit
P3.9	MTRSR SSC Master Transmit/Slave Receive
P3.10	TxD0 ASC0 Transmit Data Output
P3.11	RxD0 ASC0 Receive Data Input
P3.12	BHE/WRH Byte High Enable/Write High Output
P3.13	SCLK SSC Shift Clock Input/Output
–	–
P3.15	CLKOUT/ System Clock Output/ FOUT Programmable Frequency Output

<sup>1)</sup> If both alternate output signals (T6OUT and RxD1) are enabled, RxD1 will control pin P3.1.



**Figure 7-13 Port 3 IO and Alternate Functions**

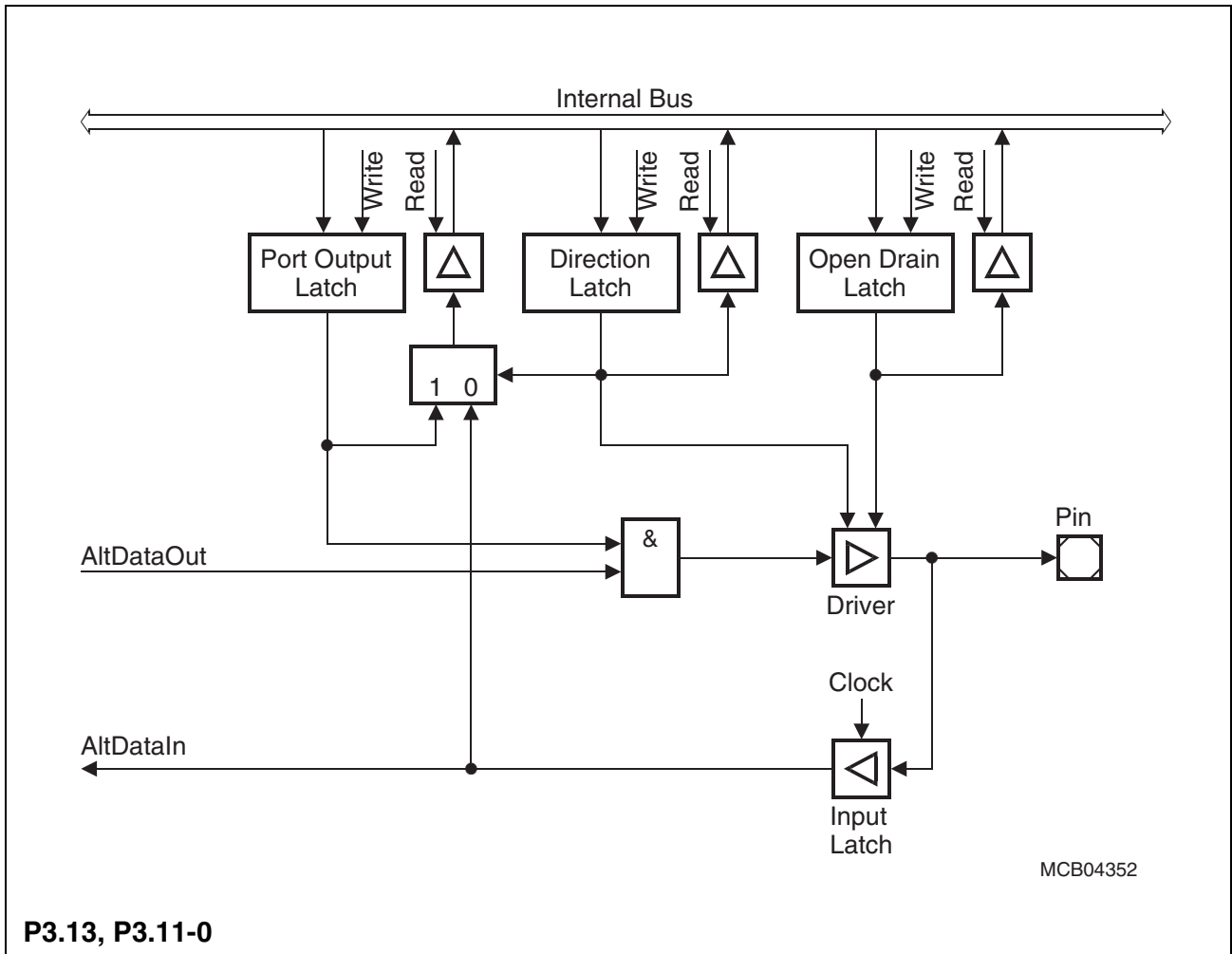
The port structure of the Port 3 pins depends on their alternate function (see [Figure 7-14](#)).

When the on-chip peripheral associated with a Port 3 pin is configured to use the alternate input function, it reads the input latch, which represents the state of the pin, via the line labeled “Alternate Data Input”. Port 3 pins with alternate input functions are: T2IN, T3IN, T4IN, T3EUD, and CAPIN.

When the on-chip peripheral associated with a Port 3 pin is configured to use the alternate output function, its “Alternate Data Output” line is ANDed with the port output latch line. When using these alternate functions, the user must set the direction of the port line to output ( $DP3.y = 1$ ) and must set the port output latch ( $P3.y = 1$ ). Otherwise the pin is in its high-impedance state (when configured as input) or the pin is stuck at ‘0’ (when the port output latch is cleared). When the alternate output functions are not used, the “Alternate Data Output” line is in its inactive state, which is a high level (‘1’). Port 3 pins with alternate output functions are: T3OUT, TxD0, and CLKOUT/FOUT.

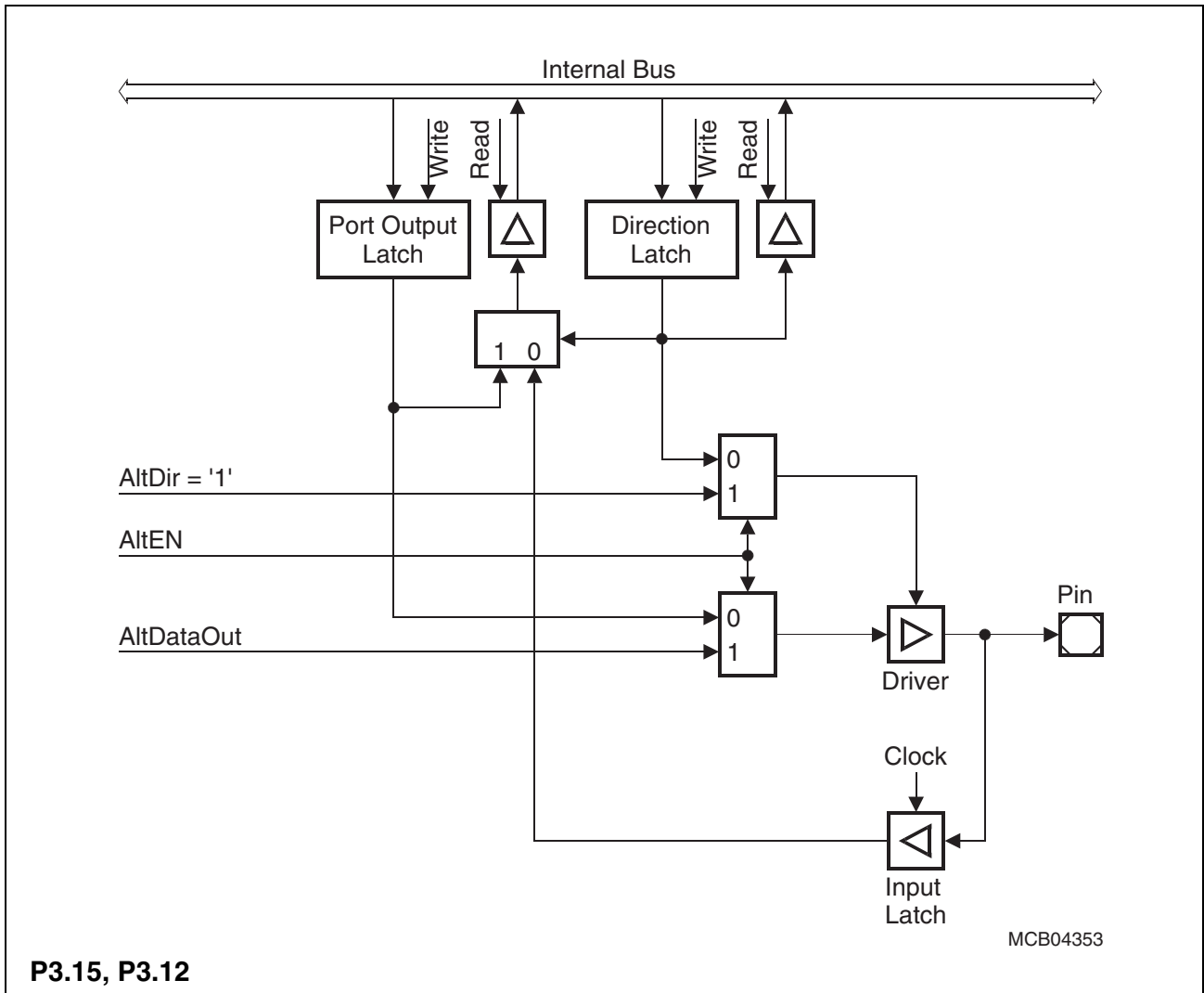
When the on-chip peripheral associated with a Port 3 pin is configured to use both the alternate input and output function, the descriptions above apply to the respective current operating mode. The direction must be set accordingly. Port 3 pins with alternate input/output functions are: T0IN/TxD1, T6OUT/RxD1, MTSR, MRST, RxD0, and SCLK.

*Note: Enabling the CLKOUT function automatically enables the P3.15 output driver. Setting bit DP3.15 = ‘1’ is not required. The CLKOUT function is automatically enabled in emulation mode.*



**Figure 7-14 Block Diagram of a Port 3 Pin with Alternate Input or Alternate Output Function**

Pin P3.12 ( $\overline{\text{BHE}}/\overline{\text{WRH}}$ ) is one more pin with an alternate output function. However, its structure is slightly different (see [Figure 7-15](#)), because after reset the  $\overline{\text{BHE}}$  or  $\overline{\text{WRH}}$  function must be used depending on the system startup configuration. In these cases there is no possibility to program any port latches before. Thus the appropriate alternate function is selected automatically. If  $\overline{\text{BHE}}/\overline{\text{WRH}}$  is not used in the system, this pin can be used for general purpose IO by disabling the alternate function ( $\text{BYTDIS} = '1'$  /  $\text{WRCFG} = '0'$ ).



**Figure 7-15 Block Diagram of Pins P3.15 (CLKOUT/FOUT) and P3.12 (BHE/WRH)**

*Note: Enabling the  $\overline{BHE}$  or  $\overline{WRH}$  function automatically enables the P3.12 output driver. Setting bit DP3.12 = '1' is not required.  
During bus hold pin  $\overline{BHE}$  (if enabled) is floating.  
Enabling the CLKOUT function automatically enables the P3.15 output driver. Setting bit DP3.15 = '1' is not required.*

## 7.8 Port 4

If this 8-bit port is used for general purpose IO, the direction of each line can be configured via the corresponding direction register DP4.

### P4

**Port 4 Data Register**

**SFR (FFC8<sub>H</sub>/E4<sub>H</sub>)**

**Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								<b>P4.7</b>	<b>P4.6</b>	<b>P4.5</b>	<b>P4.4</b>	<b>P4.3</b>	<b>P4.2</b>	<b>P4.1</b>	<b>P4.0</b>
-	-	-	-	-	-	-	-	rW	rW	rW	rW	rW	rW	rW	rW

Bit	Function
<b>P4.y</b>	<b>Port data register P4 bit y</b>

### DP4

**P4 Direction Ctrl. Register**

**SFR (FFCA<sub>H</sub>/E5<sub>H</sub>)**

**Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								<b>DP4 .7</b>	<b>DP4 .6</b>	<b>DP4 .5</b>	<b>DP4 .4</b>	<b>DP4 .3</b>	<b>DP4 .2</b>	<b>DP4 .1</b>	<b>DP4 .0</b>
-	-	-	-	-	-	-	-	rW	rW	rW	rW	rW	rW	rW	rW

Bit	Function
<b>DP4.y</b>	<b>Port direction register DP4 bit y</b> DP4.y = 0: Port line P4.y is an input (high-impedance) DP4.y = 1: Port line P4.y is an output

**ODP4**

**P4 Open Drain Ctrl. Reg.**

**ESFR (F1CA<sub>H</sub>/E5<sub>H</sub>)**

**Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								<b>ODP4</b>	<b>ODP4</b>	<b>ODP4</b>	<b>ODP4</b>	<b>ODP4</b>	<b>ODP4</b>	<b>ODP4</b>	<b>ODP4</b>
								<b>.7</b>	<b>.6</b>	<b>.5</b>	<b>.4</b>	<b>.3</b>	<b>.2</b>	<b>.1</b>	<b>.0</b>
-	-	-	-	-	-	-	-	rW	rW	rW	rW	rW	rW	rW	rW

Bit	Function
<b>ODP4.y</b>	<b>Port 4 Open Drain control register bit y</b> ODP4.y = 0: Port line P4.y output driver in push/pull mode ODP4.y = 1: Port line P4.y output driver in open drain mode

### Alternate Functions of Port 4

During external bus cycles that use segmentation (i.e. an address space above 64 KByte) a number of Port 4 pins may output the segment address lines. The number of pins that is used for segment address output determines the external address space which is directly accessible. The other pins of Port 4 (if any) may be used for general purpose IO or for the CAN/SDLM interface.

If segment address lines are selected, the alternate function of Port 4 may be necessary to access e.g. external memory directly after reset. For this reason Port 4 will be switched to this alternate function automatically.

The number of segment address lines is selected via bitfield SALSEL in register RP0H. During an external reset register RP0H is configured according to the levels of PORT0. Software can adjust the number of selected segment address lines via register RSTCON.

The CAN/SDLM interface(s) can use 2 or 4 pins of Port 4 to interface the CAN Module(s) or SDLM to an external transceiver. In this case the number of possible segment address lines is reduced.

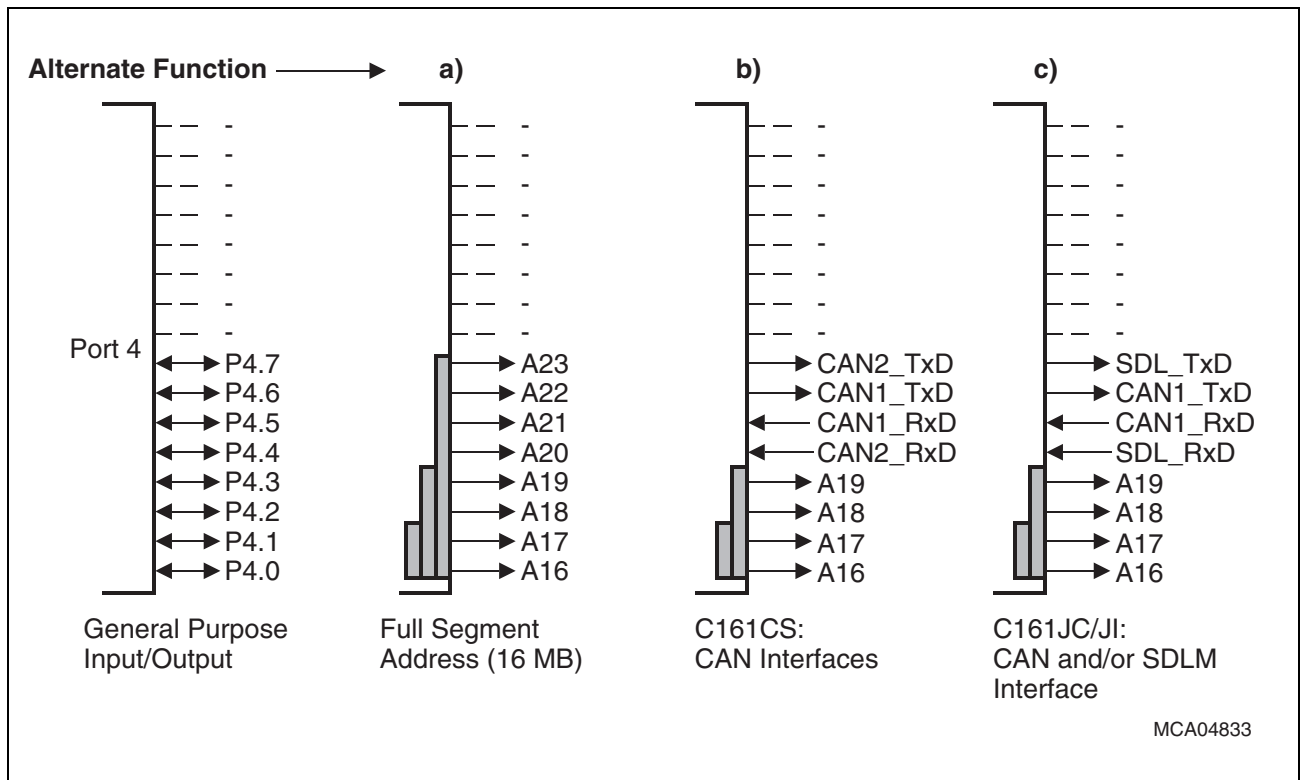
**Table 7-6** summarizes the alternate functions of Port 4 depending on the number of selected segment address lines (coded via bitfield SALSEL).

**Table 7-6 Alternate Functions of Port 4**

Port 4 Pin	Std. Function SALSEL = 01 64 KB	Altern. Function SALSEL = 11 256KB	Altern. Function SALSEL = 00 1 MB	Altern. Function SALSEL = 10 16 MB
P4.0	Gen. purpose IO	Seg. Address A16	Seg. Address A16	Seg. Address A16
P4.1	Gen. purpose IO	Seg. Address A17	Seg. Address A17	Seg. Address A17
P4.2	Gen. purpose IO	Gen. purpose IO	Seg. Address A18	Seg. Address A18
P4.3	Gen. purpose IO	Gen. purpose IO	Seg. Address A19	Seg. Address A19
P4.4	Gen. p. IO or Int.	Gen. p. IO or Int.	Gen. p. IO or Int.	S.A. A20 or Int.
P4.5	Gen. p. IO or CAN	Gen. p. IO or CAN	Gen. p. IO or CAN	S.A. A21 or CAN
P4.6	Gen. p. IO or Int.	Gen. p. IO or Int.	Gen. p. IO or Int.	S.A. A22 or Int.
P4.7	Gen. p. IO or Int.	Gen. p. IO or Int.	Gen. p. IO or Int.	S.A. A23 or Int.

*Note: Port 4 pins that are neither used for segment address output nor for the CAN/SDLM interface may be used for general purpose IO.*

*If more than one function is selected for a Port 4 pin, the CAN/SDLM interface lines will override general purpose IO and the segment address.*



**Figure 7-16 Port 4 IO and Alternate Functions**

*Note: The usage of Port 4 pins (especially P4.6 and P4.7) for CAN/SDLM interface lines depends on the chosen assignments for the CAN/SDLM module(s). CAN/SDLM interface lines will override general purpose IO and segment address lines.*



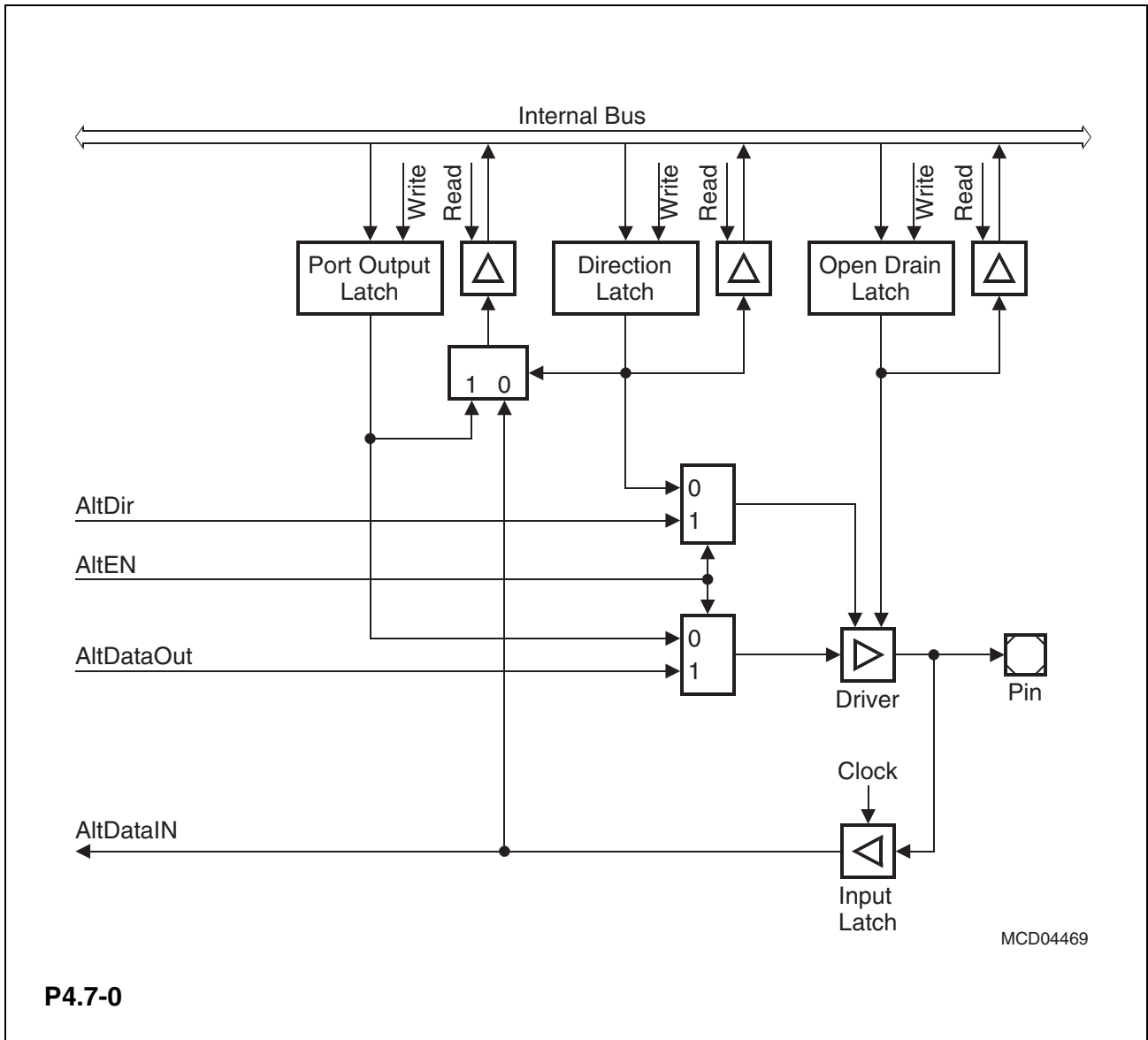


Figure 7-17 Block Diagram of a Port 4 Pin

## 7.9 Port 5

This 12-bit input port can only read data. There is no output latch and no direction register. Data written to P5 will be lost.

### P5

#### Port 5 Data Register

#### SFR (FFA2<sub>H</sub>/D1<sub>H</sub>)

#### Reset Value: XXXX<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P5 .15	P5 .14	P5 .13	P5 .12	-	-	-	-	P5.7	P5.6	P5.5	P5.4	P5.3	P5.2	P5.1	P5.0
r	r	r	r	-	-	-	-	r	r	r	r	r	r	r	r

Bit	Function
P5.y	Port data register P5 bit y (Read only)

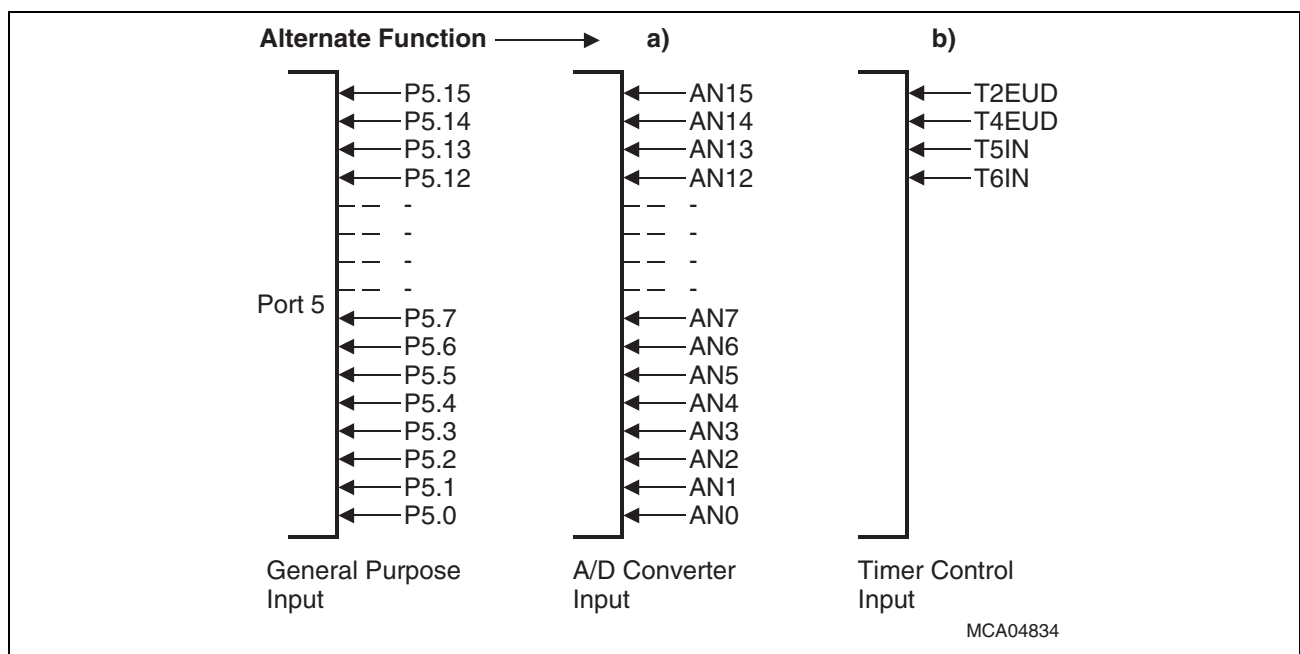
### Alternate Functions of Port 5

Each line of Port 5 is also connected to the input multiplexer of the Analog/Digital Converter. All port lines can accept analog signals (ANx) that can be converted by the ADC. For pins that shall be used as analog inputs it is recommended to disable the digital input stage via register P5DIDIS (see description below). This avoids undesired cross currents and switching noise while the (analog) input signal level is between  $V_{IL}$  and  $V_{IH}$ . Some pins of Port 5 also serve as external GPT timer control lines.

**Table 7-7** summarizes the alternate functions of Port 5.

**Table 7-7 Alternate Functions of Port 5**

Port 5 Pin	Alternate Function a)	Alternate Function b)
P5.0	Analog Input AN0	–
P5.1	Analog Input AN1	–
P5.2	Analog Input AN2	–
P5.3	Analog Input AN3	–
P5.4	Analog Input AN4	–
P5.5	Analog Input AN5	–
P5.6	Analog Input AN6	–
P5.7	Analog Input AN7	–
P5.12	Analog Input AN12	T6IN      Timer 6 Count Input
P5.13	Analog Input AN13	T5IN      Timer 5 Count Input
P5.14	Analog Input AN14	T4EUD     Timer 4 ext. Up/Down Input
P5.15	Analog Input AN15	T2EUD     Timer 2 ext. Up/Down Input



**Figure 7-18 Port 5 IO and Alternate Functions**

### Port 5 Digital Input Control

Port 5 pins may be used for both digital and analog input. By setting the respective bit in register P5DIDIS the digital input stage of the respective Port 5 pin can be disconnected from the pin. This is recommended when the pin is to be used as analog input, as it reduces the current through the digital input stage and prevents it from toggling while the (analog) input level is between the digital low and high thresholds. So the consumed power and the generated noise can be reduced.

After reset all digital inputs are enabled.

**P5DIDIS**

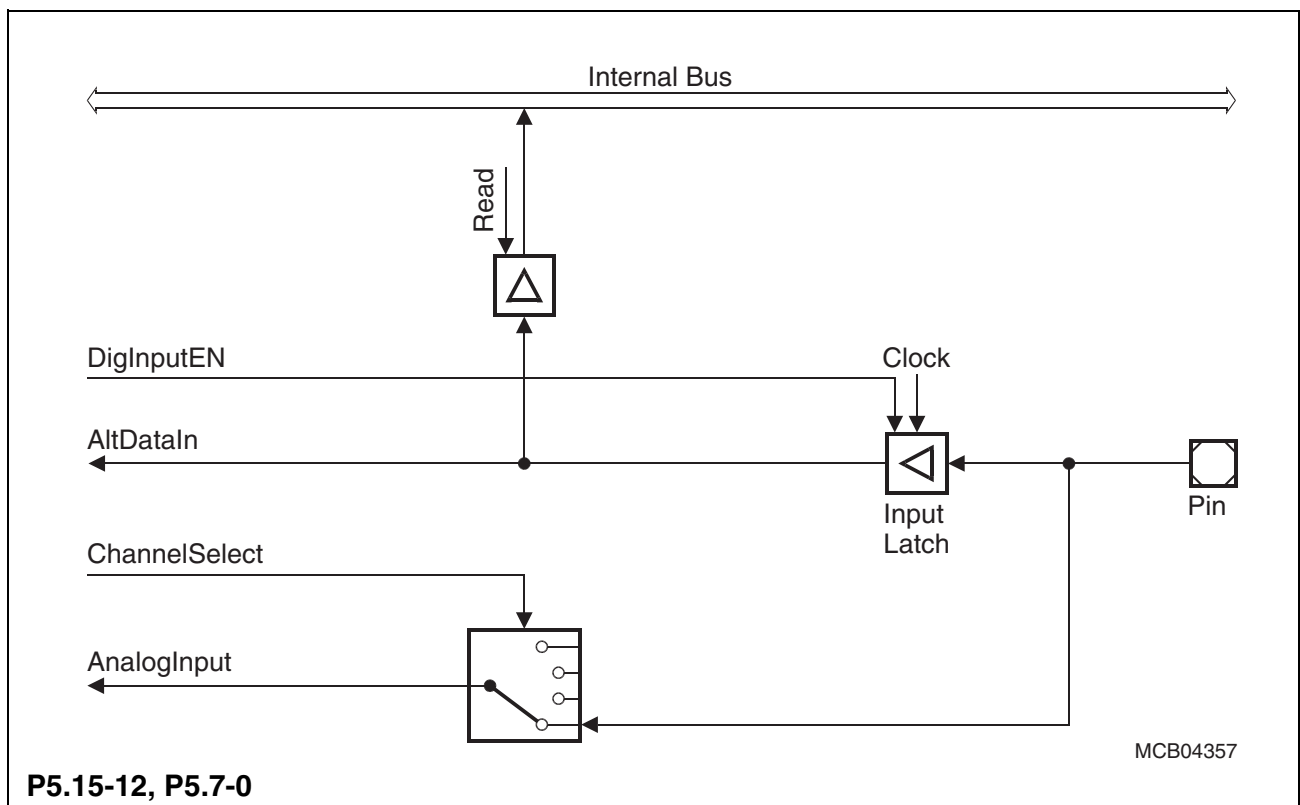
**Port 5 Digital Inp.Disable Reg. SFR (FFA4<sub>H</sub>/D2<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P5D .15	P5D .14	P5D .13	P5D .12	-	-	-	-	P5D .7	P5D .6	P5D .5	P5D .4	P5D .3	P5D .2	P5D .1	P5D .0
rw	rw	rw	rw	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
<b>P5D.y</b>	<p><b>Port 5 Bit y Digital Input Control</b></p> <p>P5D.y = 0: Digital input stage connected to port line P5.y</p> <p>P5D.y = 1: Digital input stage disconnected from port line P5.y When being read or used as alternate input this line appears as '1'.</p>

Port 5 pins have a special port structure (see [Figure 7-19](#)), first because it is an input only port, and second because the analog input channels are directly connected to the pins rather than to the input latches.



**Figure 7-19 Block Diagram of a Port 5 Pin**

*Note: The "AltDataIn" line does not exist on all Port 5 inputs.*

## 7.10 Port 6

If this 8-bit port is used for general purpose IO, the direction of each line can be configured via the corresponding direction register DP6. Each port line can be switched into push/pull or open drain mode via the open drain control register ODP6.

### P6

**Port 6 Data Register**

**SFR (FFCC<sub>H</sub>/E6<sub>H</sub>)**

**Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								<b>P6.7</b>	<b>P6.6</b>	<b>P6.5</b>	<b>P6.4</b>	<b>P6.3</b>	<b>P6.2</b>	<b>P6.1</b>	<b>P6.0</b>
-	-	-	-	-	-	-	-	rW	rW	rW	rW	rW	rW	rW	rW

Bit	Function
<b>P6.y</b>	<b>Port data register P6 bit y</b>

### DP6

**P6 Direction Ctrl. Register**

**SFR (FFCE<sub>H</sub>/E7<sub>H</sub>)**

**Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								<b>DP6 .7</b>	<b>DP6 .6</b>	<b>DP6 .5</b>	<b>DP6 .4</b>	<b>DP6 .3</b>	<b>DP6 .2</b>	<b>DP6 .1</b>	<b>DP6 .0</b>
-	-	-	-	-	-	-	-	rW	rW	rW	rW	rW	rW	rW	rW

Bit	Function
<b>DP6.y</b>	<b>Port direction register DP6 bit y</b> DP6.y = 0: Port line P6.y is an input (high-impedance) DP6.y = 1: Port line P6.y is an output

**ODP6**

**P6 Open Drain Ctrl. Reg.**

**ESFR (F1CE<sub>H</sub>/E7<sub>H</sub>)**

**Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
									ODP6 .7	ODP6 .6	ODP6 .5	ODP6 .4	ODP6 .3	ODP6 .2	ODP6 2.1	ODP6 .0
-	-	-	-	-	-	-	-	-	rW	rW	rW	rW	rW	rW	rW	rW

Bit	Function
<b>ODP6.y</b>	<b>Port 6 Open Drain control register bit y</b> ODP6.y = 0: Port line P6.y output driver in push/pull mode ODP6.y = 1: Port line P6.y output driver in open drain mode

**Alternate Functions of Port 6**

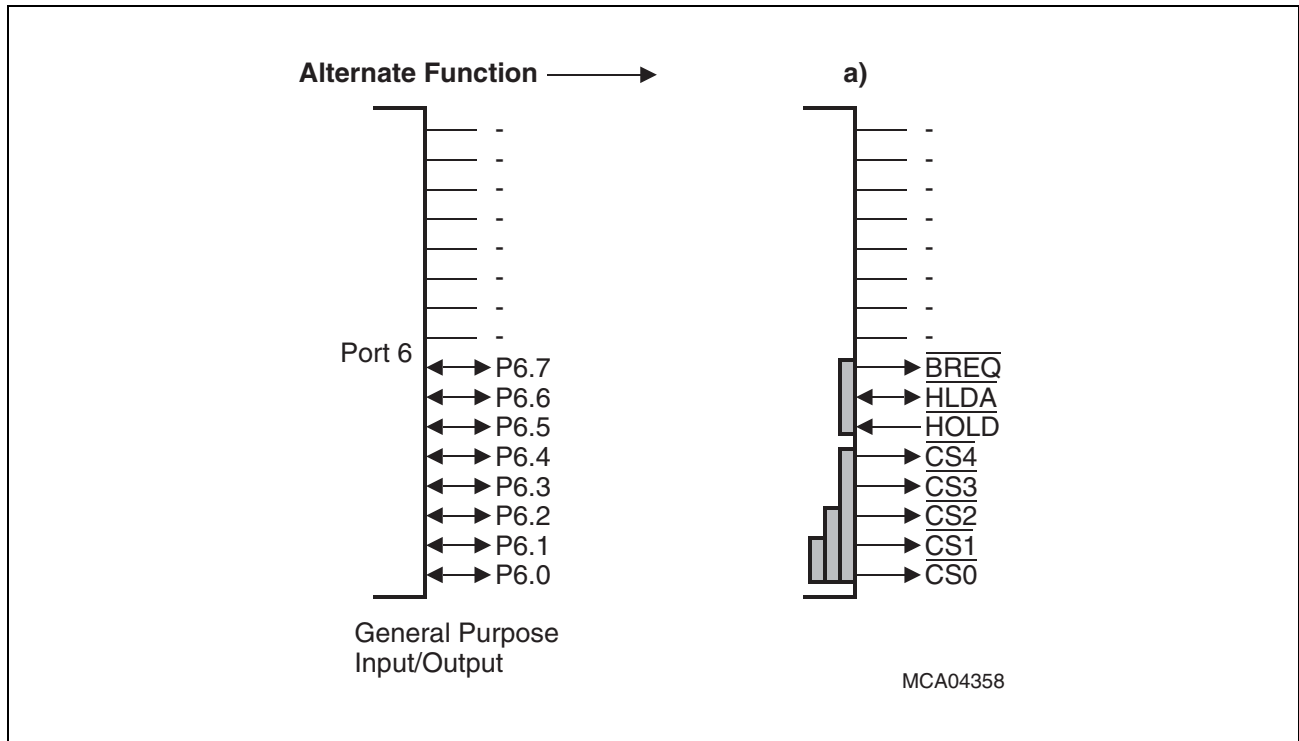
A programmable number of chip select signals (CS4 ... CS0) derived from the bus control registers (BUSCON4 ... BUSCON0) can be output on 5 pins of Port 6. The other 3 pins may be used for bus arbitration to accommodate additional masters in a C161CS/JC/JI system.

The number of chip select signals is selected via bitfield CSSEL in register RP0H. During an external reset register RP0H is configured according to the levels of PORT0. Software can adjust the number of selected chip select signals via register RSTCON.

**Table 7-8** summarizes the alternate functions of Port 6 depending on the number of selected chip select lines (coded via bitfield CSSEL).

**Table 7-8 Alternate Functions of Port 6**

Port 6 Pin	Altern. Function CSSEL = 10	Altern. Function CSSEL = 01	Altern. Function CSSEL = 00	Altern. Function CSSEL = 11
P6.0	Gen. purpose IO	Chip select $\overline{CS0}$	Chip select $\overline{CS0}$	Chip select $\overline{CS0}$
P6.1	Gen. purpose IO	Chip select $\overline{CS1}$	Chip select $\overline{CS1}$	Chip select $\overline{CS1}$
P6.2	Gen. purpose IO	Gen. purpose IO	Chip select $\overline{CS2}$	Chip select $\overline{CS2}$
P6.3	Gen. purpose IO	Gen. purpose IO	Gen. purpose IO	Chip select $\overline{CS3}$
P6.4	Gen. purpose IO	Gen. purpose IO	Gen. purpose IO	Chip select $\overline{CS4}$
P6.5	$\overline{HOLD}$	External hold request input		
P6.6	$\overline{HLDA}$	Hold acknowledge output (master mode) or input (slave mode)		
P6.7	$\overline{BREQ}$	Bus request output		



**Figure 7-20 Port 6 IO and Alternate Functions**

The chip select lines of Port 6 additionally have an internal weak pullup device. This device is switched on under the following conditions:

- always during reset for all potential  $\overline{CS}$  output pins
- if the Port 6 line is used as a chip select output, and the C161CS/JC/JI is in Hold mode (invoked through  $\overline{HOLD}$ ), and the respective pin driver is in push/pull mode ( $ODP6.x = '0'$ ).

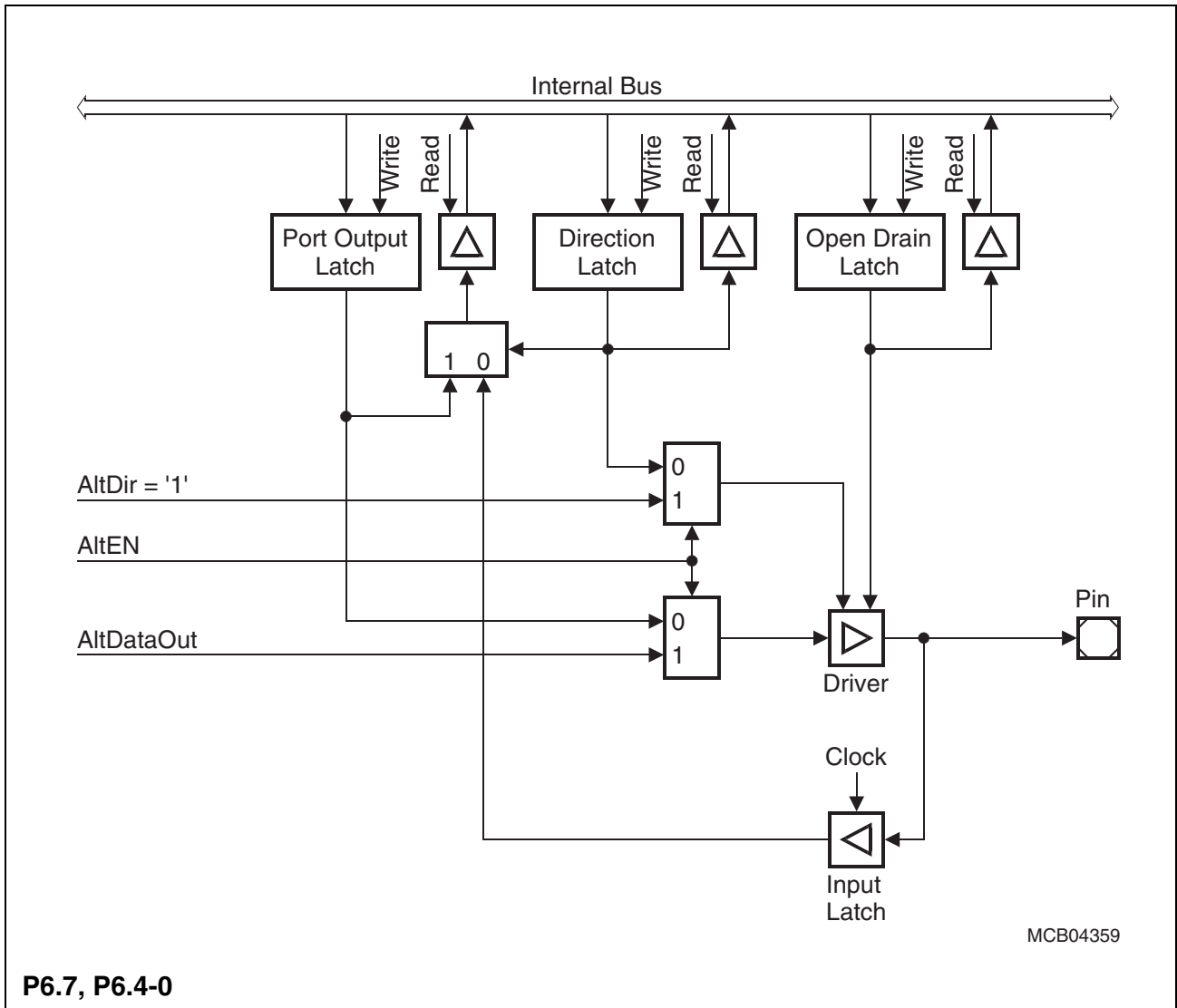
This feature is implemented to drive the chip select lines high during reset in order to avoid multiple chip selection, and to allow another master to access the external memory via the same chip select lines (Wired-AND), while the C161CS/JC/JI is in Hold mode.

With  $ODP6.x = '1'$  (open drain output selected), the internal pullup device will not be active during Hold mode; external pullup devices must be used in this case.

When entering Hold mode the  $\overline{CS}$  lines are actively driven high for one clock phase, then the output level is controlled by the pullup devices (if activated).

After reset the  $\overline{CS}$  function must be used, if selected so. In this case there is no possibility to program any port latches before. Thus the alternate function ( $\overline{CS}$ ) is selected automatically in this case.

*Note: The open drain output option can only be selected via software earliest during the initialization routine; the configured chip select lines (via CSSEL) will be in push/pull output driver mode directly after reset.*

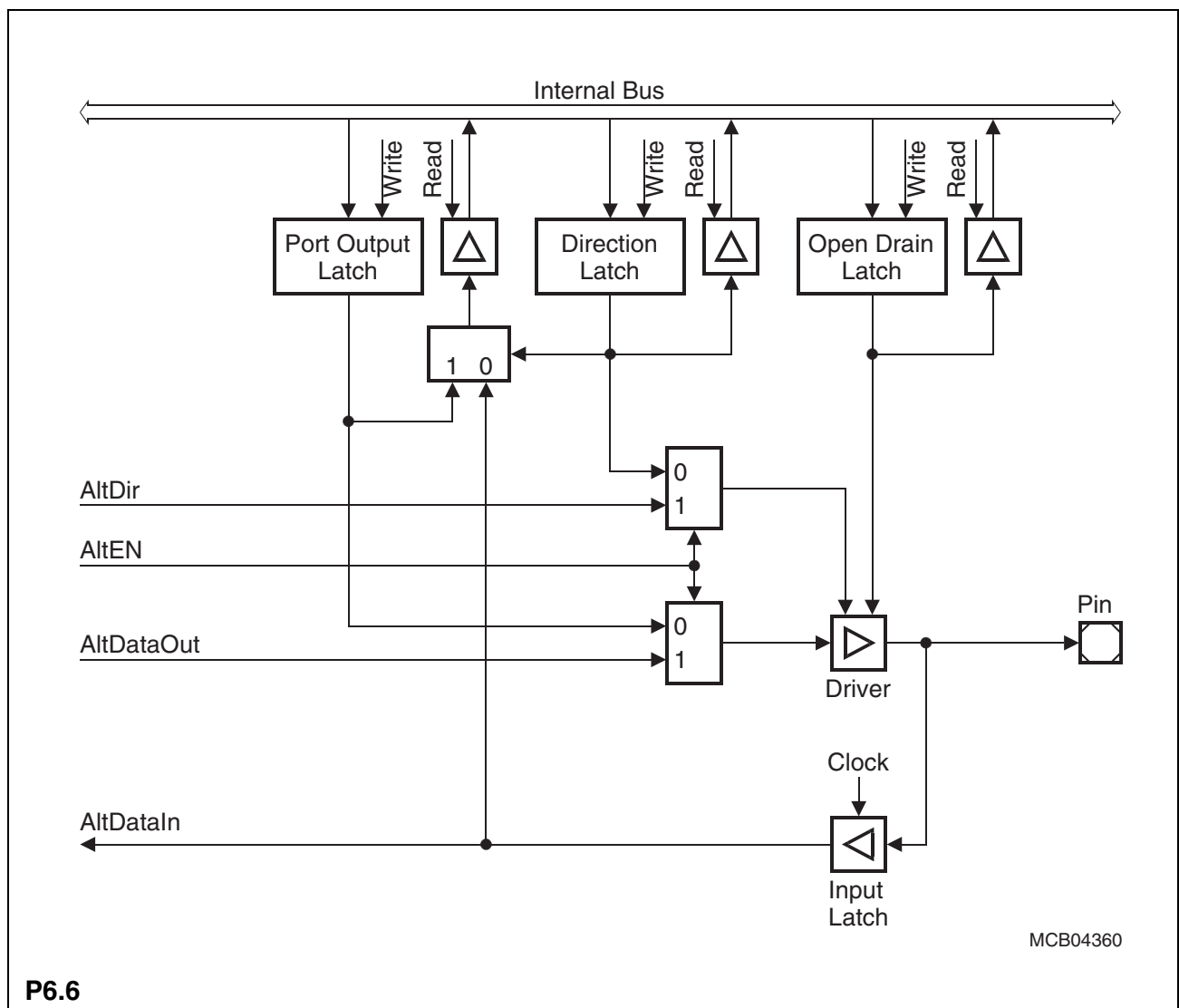


P6.7, P6.4-0

Figure 7-21 Block Diagram of Port 6 Pins with an Alternate Output Function



The bus arbitration signals  $\overline{\text{HOLD}}$ ,  $\overline{\text{HLDA}}$  and  $\overline{\text{BREQ}}$  are selected with bit  $\text{HLDEN}$  in register  $\text{PSW}$ . When the bus arbitration signals are enabled via  $\text{HLDEN}$ , also these pins are switched automatically to the appropriate direction. Note that the pin drivers for  $\overline{\text{HLDA}}$  and  $\overline{\text{BREQ}}$  are automatically controlled, while the pin driver for  $\overline{\text{HOLD}}$  is automatically disabled.



**Figure 7-22 Block Diagram of Pin P6.6 ( $\overline{\text{HLDA}}$ )**

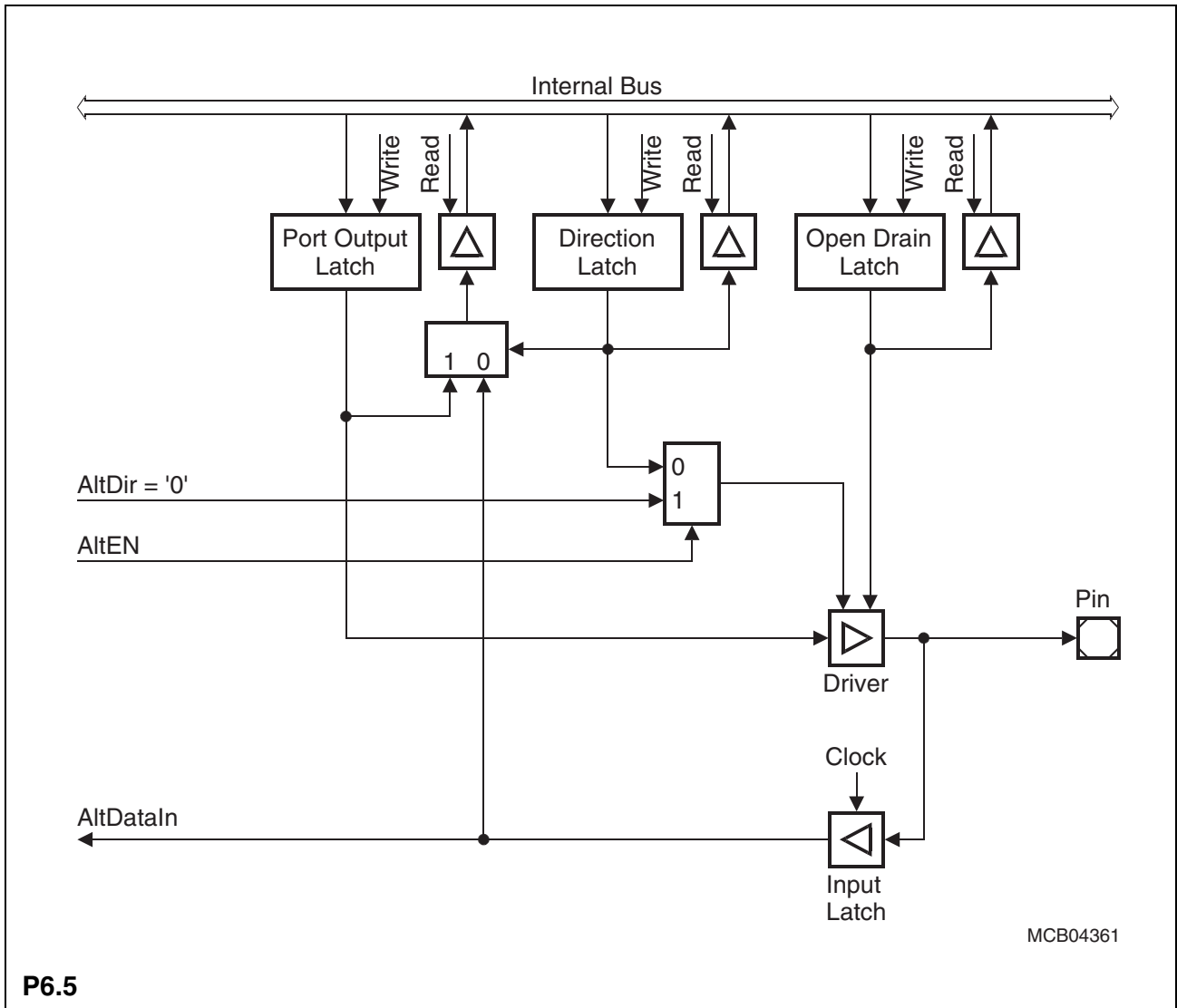


Figure 7-23 Block Diagram of Pin P6.5 (HOLD)

## 7.11 Port 7

If this 8-bit port is used for general purpose IO, the direction of each line can be configured via the corresponding direction register DP7. Each port line can be switched into push/pull or open drain mode via the open drain control register ODP7.

### P7

**Port 7 Data Register**                      **SFR (FFD0<sub>H</sub>/E8<sub>H</sub>)**                      **Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								<b>P7.7</b>	<b>P7.6</b>	<b>P7.5</b>	<b>P7.4</b>	-	-	-	-
-	-	-	-	-	-	-	-	rW	rW	rW	rW	-	-	-	-

Bit	Function
<b>P7.y</b>	<b>Port data register P7 bit y</b>

### DP7

**P7 Direction Ctrl. Register**                      **SFR (FFD2<sub>H</sub>/E9<sub>H</sub>)**                      **Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								<b>DP7.7</b>	<b>DP7.6</b>	<b>DP7.5</b>	<b>DP7.4</b>	-	-	-	-
-	-	-	-	-	-	-	-	rW	rW	rW	rW	-	-	-	-

Bit	Function
<b>DP7.y</b>	<b>Port direction register DP7 bit y</b> DP7.y = 0: Port line P7.y is an input (high-impedance) DP7.y = 1: Port line P7.y is an output

**ODP7**

**P7 Open Drain Ctrl. Reg.**

**ESFR (F1D2<sub>H</sub>/E9<sub>H</sub>)**

**Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								ODP7 .7	ODP7 .6	ODP7 .5	ODP7 .4	-	-	-	-
-	-	-	-	-	-	-	-	rW	rW	rW	rW	-	-	-	-

Bit	Function
<b>ODP7.y</b>	<b>Port 7 Open Drain control register bit y</b> ODP7.y = 0: Port line P7.y output driver in push/pull mode ODP7.y = 1: Port line P7.y output driver in open drain mode

**Alternate Functions of Port 7**

The Port 7 lines (P7.7-4) serve as capture inputs or compare outputs (CC31IO ... CC28IO) for the CAPCOM2 unit.

The usage of the port lines by the CAPCOM unit, its accessibility via software, and the precautions are the same as described for the Port 2 lines.

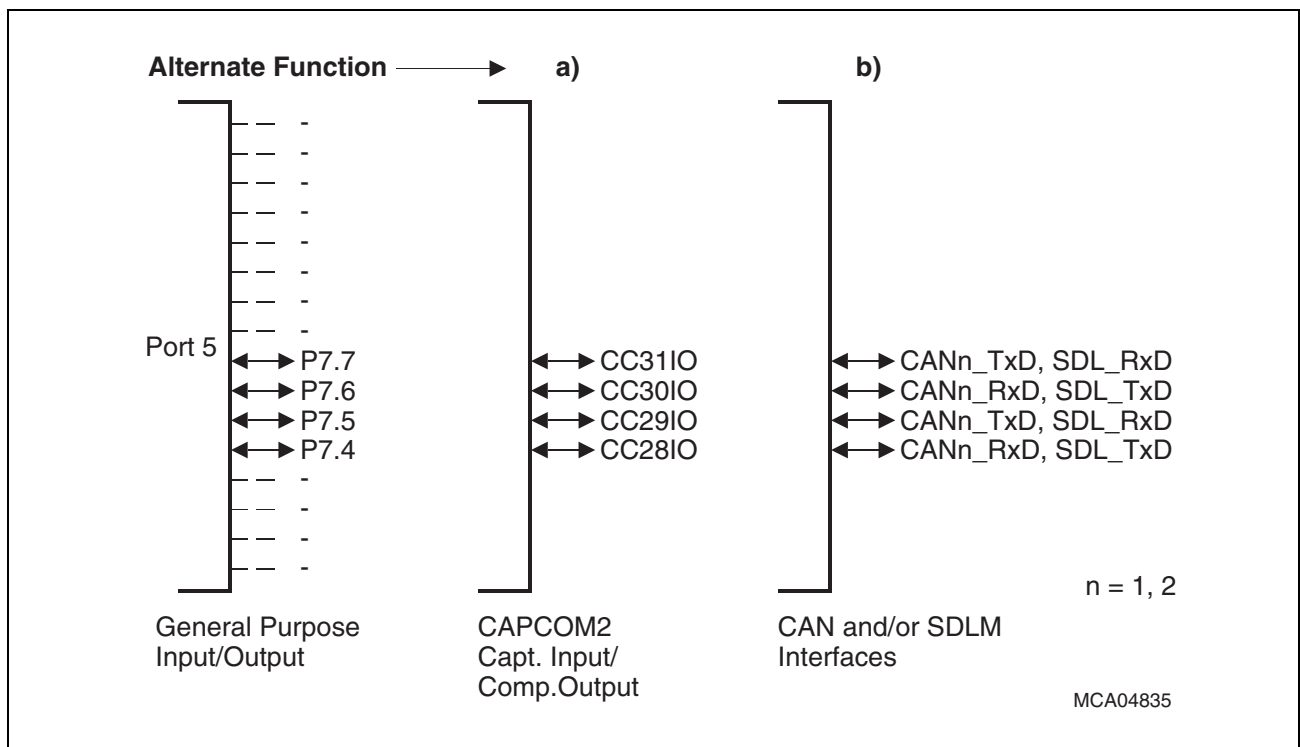
As all other capture inputs, the capture input function of pins P7.7-4 can also be used as external interrupt inputs (sample rate 16 TCL).

The CAN/SDLM interface(s) can use 2 or 4 pins of Port 7 to interface the CAN/SDLM module(s) to an external transceiver. In this case the number of possible CAPCOM IO lines is reduced.

**Table 7-9** summarizes the alternate functions of Port 7.

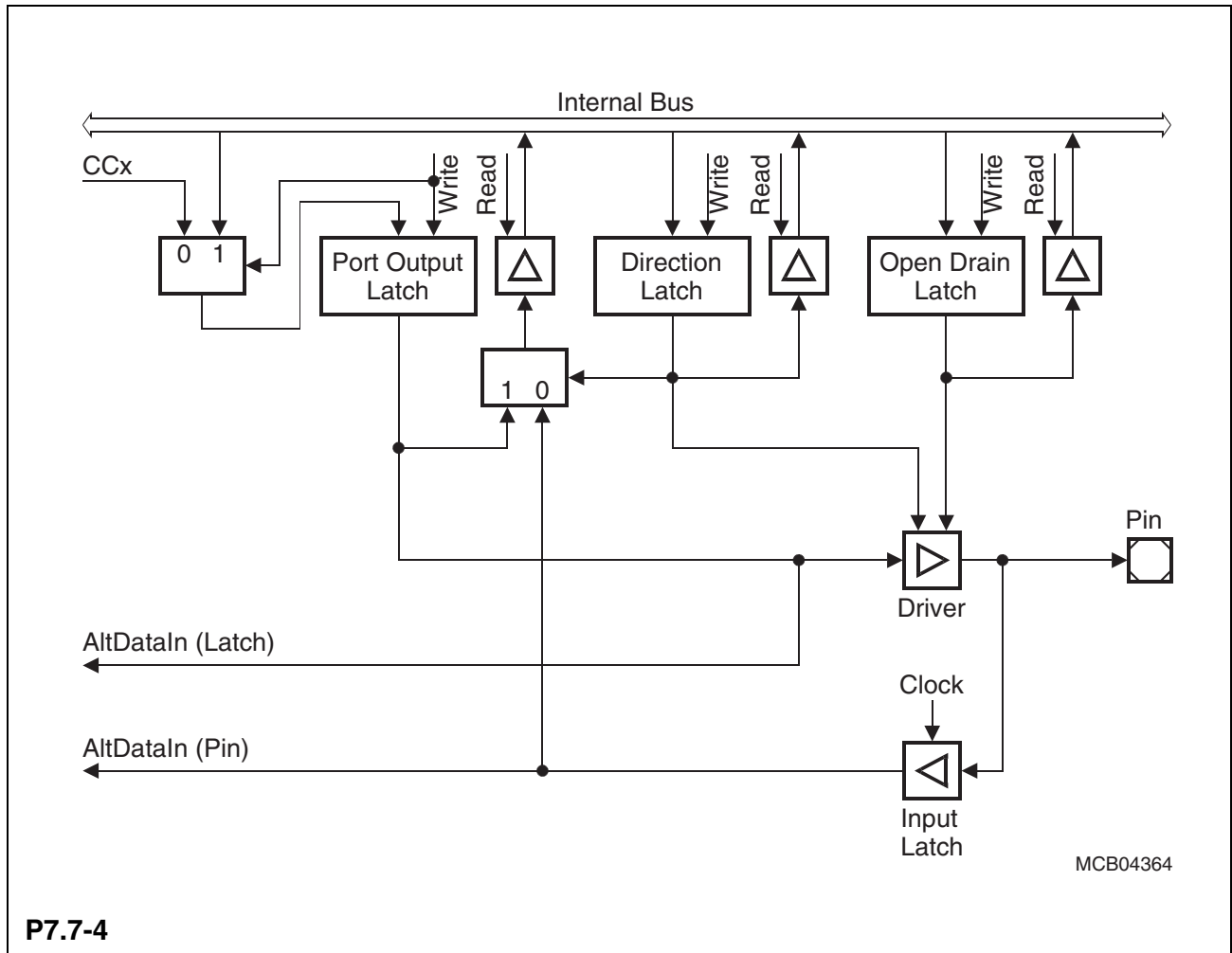
**Table 7-9 Alternate Functions of Port 7**

Port 7 Pin	Alternate Function	
P7.4	CC28IO	Capture input/compare output channel 28 or CAN/SDLM
P7.5	CC29IO	Capture input/compare output channel 29 or CAN/SDLM
P7.6	CC30IO	Capture input/compare output channel 30 or CAN/SDLM
P7.7	CC31IO	Capture input/compare output channel 31 or CAN/SDLM



**Figure 7-24 Port 7 IO and Alternate Functions**

The pins of Port 7 combine internal bus data and alternate data output before the port latch input, as do the Port 2 pins.



**Figure 7-25 Block Diagram of Port 7 Pins P7.7-4**

## 7.12 Port 9

If this 6-bit port is used for general purpose IO, the direction of each line can be configured via the corresponding direction register DP9.

Port 9 is an open drain mode I/O port (pullup resistors must be provided externally if required).

### P9

**Port 9 Data Register**                      **SFR (FFD8<sub>H</sub>/EC<sub>H</sub>)**                      **Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								-	-	P9.5	P9.4	P9.3	P9.2	P9.1	P9.0
-	-	-	-	-	-	-	-	-	-	rW	rW	rW	rW	rW	rW

Bit	Function
P9.y	Port data register P9 bit y

### DP9

**P9 Direction Ctrl. Register**                      **SFR (FFDA<sub>H</sub>/ED<sub>H</sub>)**                      **Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								-	-	DP9.5	DP9.4	DP9.3	DP9.2	DP9.1	DP9.0
-	-	-	-	-	-	-	-	-	-	rW	rW	rW	rW	rW	rW

Bit	Function
DP9.y	Port direction register DP9 bit y DP9.y = 0: Port line P9.y is an input (high-impedance) DP9.y = 1: Port line P9.y is an (open drain) output

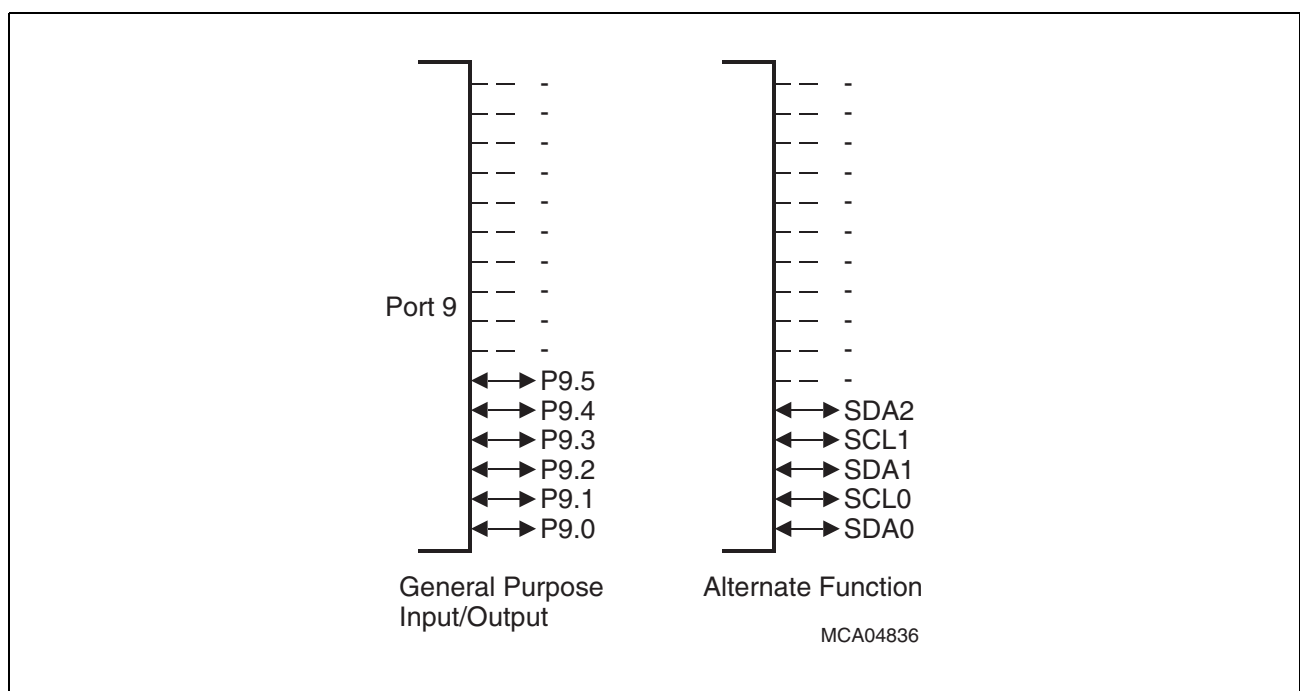
### Alternate Functions of Port 9

The pins of Port 9 are used to connect the on-chip IIC bus module to the external world.

**Table 7-10** summarizes the alternate functions of Port 9.

**Table 7-10 Alternate Functions of Port 9**

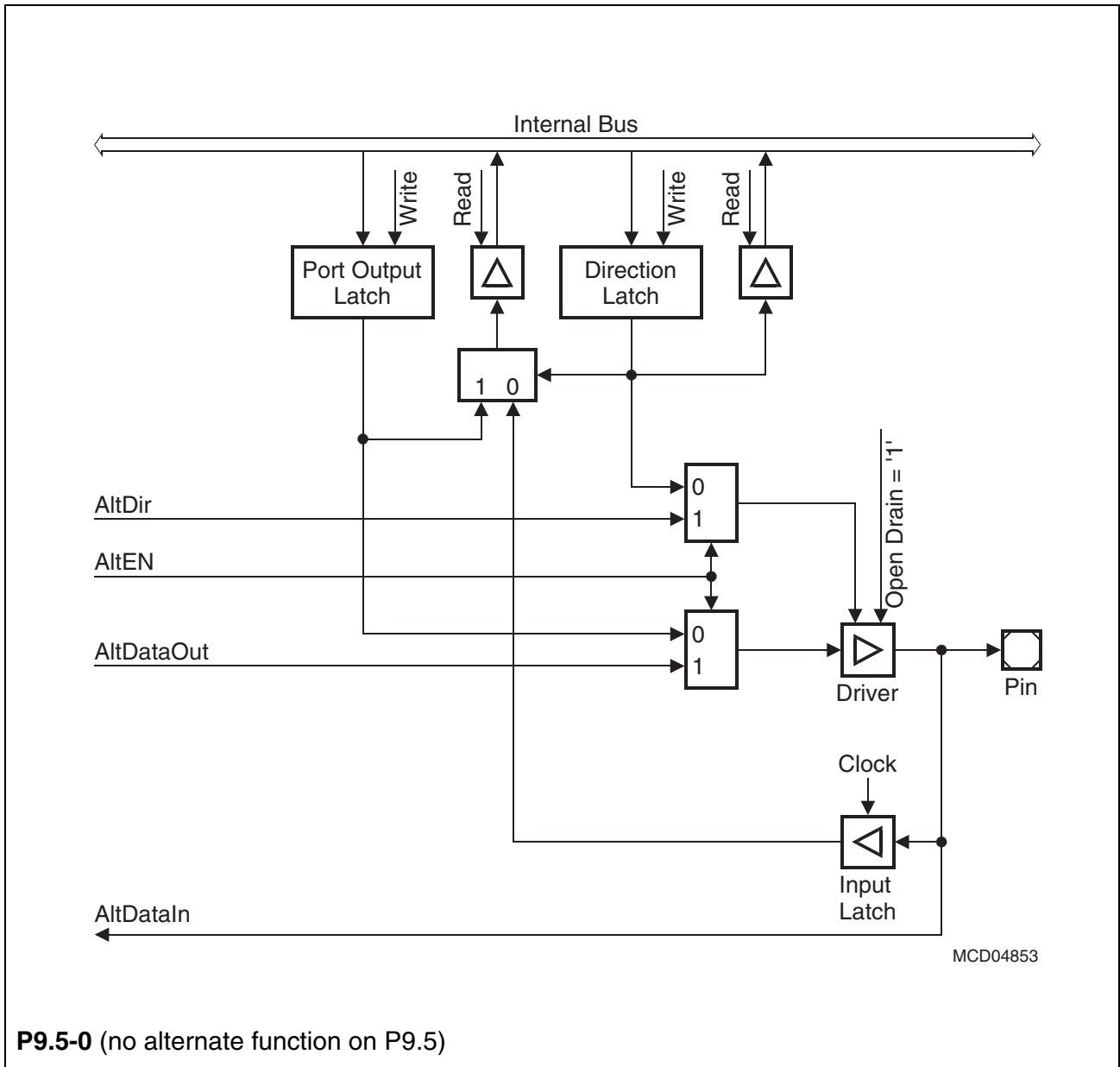
Port 9 Pin	Alternate Function
P9.0	SDA0 IIC serial data line 0
P9.1	SCL0 IIC serial clock line 0
P9.2	SDA1 IIC serial data line 1
P9.3	SCL1 IIC serial clock line 1
P9.4	SDA2 IIC serial data line 2
P9.5	–



**Figure 7-26 Port 9 IO and Alternate Functions**

*Note: The alternate input/output signals of Port 9 are enabled/disabled via software by the IIC Bus module.*





P9.5-0 (no alternate function on P9.5)

Figure 7-27 Block Diagram of a Port 9 Pin

## 8 Dedicated Pins

Most of the input/output or control signals of the functional the C161CS/JC/JI are realized as alternate functions of pins of the parallel ports. There is, however, a number of signals that use separate pins, including the oscillator, special control signals and, of course, the power supply.

**Table 8-1** summarizes the 33 dedicated pins of the C161CS/JC/JI.

**Table 8-1 C161CS/JC/JI Dedicated Pins**

Pin(s)	Function
ALE	Address Latch Enable
$\overline{RD}$	External Read Strobe
$\overline{WR/WRL}$	External Write/Write Low Strobe
$\overline{READY}$	Ready Input
$\overline{EA}$	External Access Enable
$\overline{NMI}$	Non-Maskable Interrupt Input
XTAL1, XTAL2	Oscillator Input/Output (main oscillator)
XTAL3, XTAL4	Oscillator Input/Output (auxiliary oscillator)
$\overline{RSTIN}$	Reset Input
$\overline{RSTOUT}$	Reset Output
$V_{AREF}, V_{AGND}$	Power Supply for Analog/Digital Converter
$V_{DD}$	Digital Power Supply (10 pins)
$V_{SS}$	Digital Reference Ground (10 pins)

**The Address Latch Enable signal ALE** controls external address latches that provide a stable address in multiplexed bus modes.

**ALE is activated** for every external bus cycle independent of the selected bus mode, i.e. it is also activated for bus cycles with a demultiplexed address bus. When an external bus is enabled (one or more of the BUSACT bits set) also X-Peripheral accesses will generate an active ALE signal.

**ALE is not activated** for internal accesses, i.e. accesses to ROM/OTP/Flash (if provided), the internal RAM and the special function registers. In single chip mode, i.e. when no external bus is enabled (no BUSACT bit set), ALE will also remain inactive for X-Peripheral accesses.

During reset an internal pulldown ensures an inactive (low) level on the ALE output.

At the end of a true single-chip mode reset ( $\overline{EA} = '1'$ ) the current level on pin ALE is latched and is used for configuration (together with pin  $\overline{RD}$ ). Pin ALE selects standard start/boot, when driven low (default) or alternate start/boot when driven high.

For standard configuration pin ALE should be low or not connected.

**The External Read Strobe  $\overline{RD}$**  controls the output drivers of external memory or peripherals when the C161CS/JC/JI reads data from these external devices. During accesses to on-chip X-Peripherals  $\overline{RD}$  remains inactive (high).

During reset an internal pullup ensures an inactive (high) level on the  $\overline{RD}$  output.

At the end of reset the current level on pin  $\overline{RD}$  is latched and is used for configuration.

For a reset with external access ( $\overline{EA} = '0'$ ) pin  $\overline{RD}$  controls the oscillator watchdog. The latched  $\overline{RD}$  level determines the reset value of bit OWDDIS in register SYSCON. The default high level on pin  $\overline{RD}$  leaves the oscillator watchdog active (OWDDIS = '0'), while a low level disables the watchdog (OWDDIS = '1') e.g. for testing purposes.

For a true single-chip mode reset ( $\overline{EA} = '1'$ ) pin  $\overline{RD}$  enables the bootstrap loader, when driven low (pin ALE is evaluated together with pin  $\overline{RD}$ ).

For standard configuration pin  $\overline{RD}$  should be high or not connected.

**The External Write Strobe  $\overline{WR/WRL}$**  controls the data transfer from the C161CS/JC/JI to an external memory or peripheral device. This pin may either provide an general  $\overline{WR}$  signal activated for both byte and word write accesses, or specifically control the low byte of an external 16-bit device ( $\overline{WRL}$ ) together with the signal  $\overline{WRH}$  (alternate function of P3.12/ $\overline{BHE}$ ). During accesses to on-chip X-Peripherals  $\overline{WR/WRL}$  remains inactive (high).

During reset an internal pullup ensures an inactive (high) level on the  $\overline{WR/WRL}$  output.

**The Ready Input  $\overline{READY}$**  receives a control signal from an external memory or peripheral device that is used to terminate an external bus cycle, provided that this function is enabled for the current bus cycle.  $\overline{READY}$  may be used as synchronous  $\overline{READY}$  or may be evaluated asynchronously. When waitstates are defined for a  $\overline{READY}$  controlled address window the  $\overline{READY}$  input is not evaluated during these waitstates.

An internal pullup ensures an inactive (high) level on the  $\overline{READY}$  input.

**The External Access Enable Pin  $\overline{EA}$**  determines if the C161CS/JC/JI after reset starts fetching code from the internal ROM area ( $\overline{EA} = '1'$ ) or via the external bus interface ( $\overline{EA} = '0'$ ). Be sure to hold this input low for ROMless devices. At the end of the internal reset sequence the  $\overline{EA}$  signal is latched together with the configuration (PORT0,  $\overline{RD}$ , ALE).

**The Non-Maskable Interrupt Input  $\overline{NMI}$**  allows to trigger a high priority trap via an external signal (e.g. a power-fail signal). It also serves to validate the PWRDN instruction that switches the C161CS/JC/JI into Power-Down mode. The  $\overline{NMI}$  pin is sampled with every CPU clock cycle to detect transitions.

**The Oscillator Input XTAL1 and Output XTAL2** connect the internal **Main Oscillator** to the external crystal. The oscillator provides an inverter and a feedback element. The standard external oscillator circuitry (see [Chapter 6](#)) comprises the crystal, two low end capacitors and series resistor to limit the current through the crystal. The main oscillator is intended for the generation of the basic operating clock signal of the C161CS/JC/JI.

An external clock signal may be fed to the input XTAL1, leaving XTAL2 open or terminating it for higher input frequencies.

**The Oscillator Input XTAL3 and Output XTAL4** connect the internal **Auxiliary Oscillator** to the external crystal. The oscillator provides an inverter and a feedback element. The standard external oscillator circuitry (see [Chapter 6](#)) comprises the crystal, two low end capacitors and series resistor to limit the current through the crystal. The auxiliary oscillator is intended for the generation of a power saving backup clock signal for the C161CS/JC/JI's real time clock, especially during power saving modes.

An external clock signal may be fed to the input XTAL3, leaving XTAL4 open.

*Note: In order to reduce its power consumption as much as possible the operating range of the auxiliary oscillator is optimized around 32 kHz (10 ... 50 kHz when driven externally).*

**The Reset Input  $\overline{\text{RSTIN}}$**  allows to put the C161CS/JC/JI into the well defined reset condition either at power-up or external events like a hardware failure or manual reset. The input voltage threshold of the  $\overline{\text{RSTIN}}$  pin is raised compared to the standard pins in order to minimize the noise sensitivity of the reset input.

In bidirectional reset mode the C161CS/JC/JI's line  $\overline{\text{RSTIN}}$  may be driven active by the chip logic e.g. in order to support external equipment which is required for startup (e.g. flash memory).

Bidirectional reset reflects internal reset sources (software, watchdog) also to the  $\overline{\text{RSTIN}}$  pin and converts short hardware reset pulses to a minimum duration of the internal reset sequence. Bidirectional reset is enabled by setting bit BDRSTEN in register SYSCON and changes  $\overline{\text{RSTIN}}$  from a pure input to an open drain IO line. When an internal reset is triggered by the SRST instruction or by a watchdog timer overflow or a low level is applied to the  $\overline{\text{RSTIN}}$  line, an internal driver pulls it low for the duration of the internal reset sequence. After that it is released and is then controlled by the external circuitry alone.

The bidirectional reset function is useful in applications where external devices require a defined reset signal but cannot be connected to the C161CS/JC/JI's  $\overline{\text{RSTOUT}}$  signal, e.g. an external flash memory which must come out of reset and deliver code well before  $\overline{\text{RSTOUT}}$  can be deactivated via EINIT.

The following behavior differences must be observed when using the bidirectional reset feature in an application:

- Bit BDRSTEN in register SYSCON cannot be changed after EINIT and is cleared automatically after a reset.
- The reset indication flags always indicate a long hardware reset.
- The PORT0 configuration is treated like on a hardware reset. Especially the bootstrap loader may be activated when P0L.4 is low.
- Pin  $\overline{\text{RSTIN}}$  may only be connected to external reset devices with an open drain output driver.
- A short hardware reset is extended to the duration of the internal reset sequence.

**The Reset Output  $\overline{\text{RSTOUT}}$**  provides a special reset signal for external circuitry.  $\overline{\text{RSTOUT}}$  is activated at the beginning of the reset sequence, triggered via  $\overline{\text{RSTIN}}$ , a watchdog timer overflow or by the SRST instruction.  $\overline{\text{RSTOUT}}$  remains active (low) until the EINIT instruction is executed. This allows to initialize the controller before the external circuitry is activated.

*Note: During emulation mode pin  $\overline{\text{RSTOUT}}$  is used as an input and therefore must be driven by the external circuitry.*

**The Power Supply pins for the Analog/Digital Converter  $V_{\text{AREF}}$  and  $V_{\text{AGND}}$**  provide a separate power supply (reference voltage) for the on-chip ADC. This reduces the noise that is coupled to the analog input signals from the digital logic sections and so improves the stability of the conversion results, when  $V_{\text{AREF}}$  and  $V_{\text{AGND}}$  are properly decoupled from  $V_{\text{DD}}$  and  $V_{\text{SS}}$ .

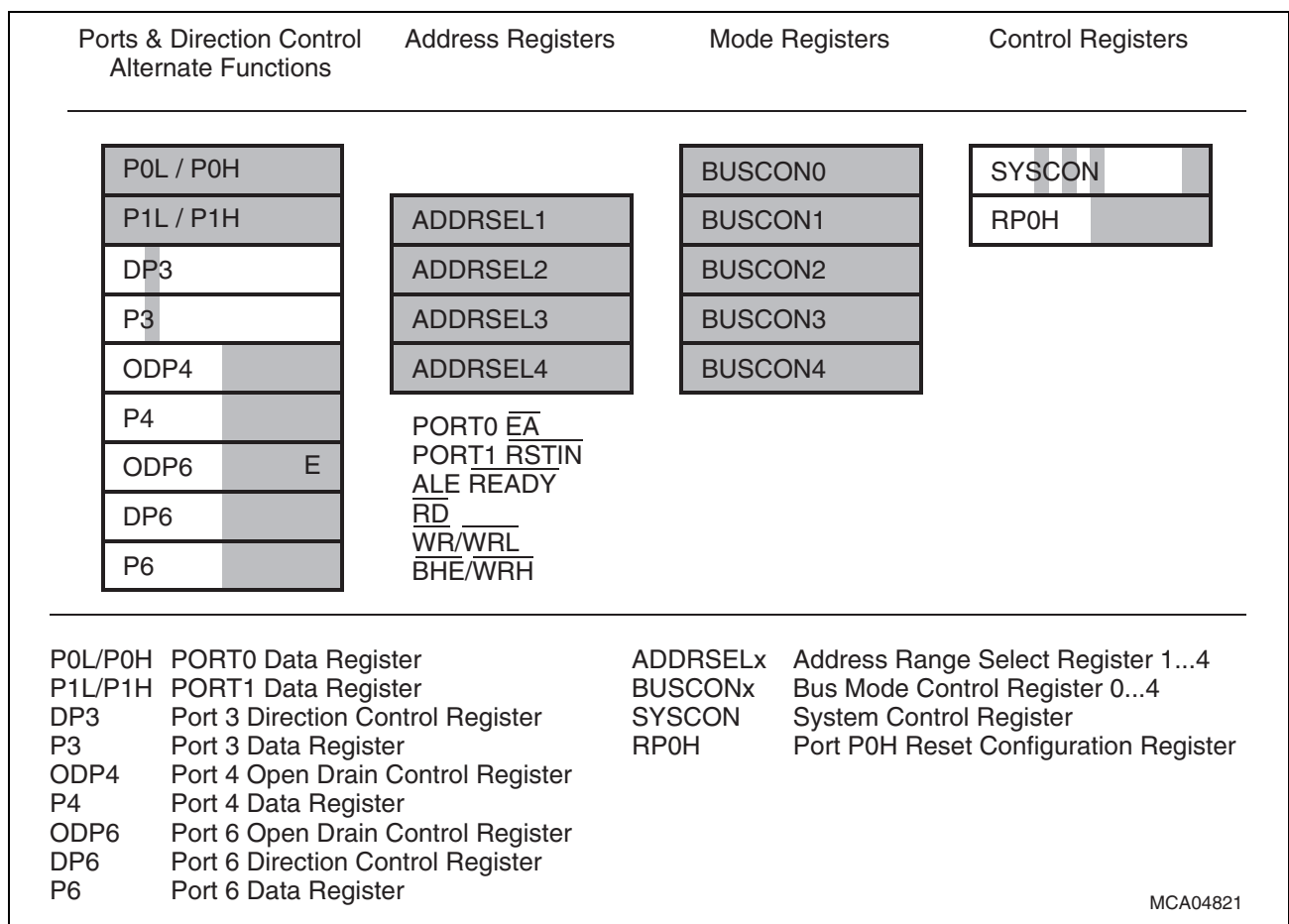
**The Power Supply pins  $V_{\text{DD}}$  and  $V_{\text{SS}}$**  provide the power supply for the digital logic of the C161CS/JC/JI. The respective  $V_{\text{DD}}/V_{\text{SS}}$  pairs should be decoupled as close to the pins as possible. For best results it is recommended to implement two-level decoupling, e.g. (the widely used) 100 nF in parallel with 30 ... 40 pF capacitors which deliver the peak currents.

The  $V_{\text{DD}}/V_{\text{SS}}$  pair on pins 26/25 provides the digital supply for the on-chip ADC, which may be especially protected against noise.

*Note: All  $V_{\text{DD}}$  pins and all  $V_{\text{SS}}$  pins must be connected to the power supply and ground, respectively.*

## 9 The External Bus Interface

Although the C161CS/JC/JI provides a powerful set of on-chip peripherals and on-chip RAM and ROM/OTP/Flash (except for ROMless versions) areas, these internal units only cover a small fraction of its address space of up to 16 MByte. The external bus interface allows to access external peripherals and additional volatile and non-volatile memory. The external bus interface provides a number of configurations, so it can be tailored to fit perfectly into a given application system.



**Figure 9-1 SFRs and Port Pins Associated with the External Bus Interface**

Accesses to external memory or peripherals are executed by the integrated External Bus Controller (EBC). The function of the EBC is controlled via the SYSCON register and the BUSCONx and ADDRSELx registers. The BUSCONx registers specify the external bus cycles in terms of address (mux/demux), data width (16-bit/8-bit), chip selects and length (waitstates/READY control/ALE/RW delay). These parameters are used for accesses within a specific address area which is defined via the corresponding register ADDRSELx.

The four pairs BUSCON1/ADDRSEL1 ... BUSCON4/ADDRSEL4 allow to define four independent “address windows”, while all external accesses outside these windows are controlled via register BUSCON0.

## 9.1 Single Chip Mode

Single chip mode is entered, when pin  $\overline{EA}$  is high during reset. In this case register BUSCON0 is initialized with 00C0<sub>H</sub>, which also resets bit BUSACT0, so no external bus is enabled.

In single chip mode the C161CS/JC/JI operates only with and out of internal resources. No external bus is configured and no external peripherals and/or memory can be accessed. Also no port lines are occupied for the bus interface. When running in single chip mode, however, external access may be enabled by configuring an external bus under software control. Single chip mode allows the C161CS/JC/JI to start execution out of the internal program memory (Mask-ROM, OTP or Flash memory).

*Note: Any attempt to access a location in the external memory space in single chip mode results in the hardware trap ILLBUS if no external bus has been explicitly enabled by software.*

## 9.2 External Bus Modes

When the external bus interface is enabled (bit BUSACT<sub>x</sub> = '1') and configured (bitfield BTYP), the C161CS/JC/JI uses a subset of its port lines together with some control lines to build the external bus.

**Table 9-1 Summary of External Bus Modes**

<b>BTYP Encoding</b>	<b>External Data Bus Width</b>	<b>External Address Bus Mode</b>
0 0	8-bit Data	Demultiplexed Addresses
0 1	8-bit Data	Multiplexed Addresses
1 0	16-bit Data	Demultiplexed Addresses
1 1	16-bit Data	Multiplexed Addresses

The bus configuration (BTYP) for the address windows (BUSCON4 ... BUSCON1) is selected via software typically during the initialization of the system.

The bus configuration (BTYP) for the default address range (BUSCON0) is selected via PORT0 during reset, provided that pin  $\overline{EA}$  is low during reset. Otherwise BUSCON0 may be programmed via software just like the other BUSCON registers.

The 16 MByte address space of the C161CS/JC/JI is divided into 256 segments of 64 KByte each. The 16-bit intra-segment address is output on PORT0 for multiplexed bus modes or on PORT1 for demultiplexed bus modes. When segmentation is disabled, only one 64 KByte segment can be used and accessed. Otherwise additional address lines may be output on Port 4 (addressing up to 16 MByte) and/or several chip select lines may be used to select different memory banks or peripherals. These functions are selected during reset via bitfields SALSEL and CSSEL of register RP0H, respectively.

*Note: Bit SGTDIS of register SYSCON defines, if the CSP register is saved during interrupt entry (segmentation active) or not (segmentation disabled).*



### Multiplexed Bus Modes

In the multiplexed bus modes the 16-bit intra-segment address as well as the data use PORT0. The address is time-multiplexed with the data and has to be latched externally. The width of the required latch depends on the selected data bus width, i.e. an 8-bit data bus requires a byte latch (the address bits A15 ... A8 on P0H do not change, while P0L multiplexes address and data), a 16-bit data bus requires a word latch (the least significant address line A0 is not relevant for word accesses).

The upper address lines (An ... A16) are permanently output on Port 4 (if segmentation is enabled) and do not require latches.

The EBC initiates an external access by generating the Address Latch Enable signal (ALE) and then placing an address on the bus. The falling edge of ALE triggers an external latch to capture the address. After a period of time during which the address must have been latched externally, the address is removed from the bus. The EBC now activates the respective command signal ( $\overline{RD}$ ,  $\overline{WR}$ ,  $\overline{WRL}$ ,  $\overline{WRH}$ ). Data is driven onto the bus either by the EBC (for write cycles) or by the external memory/peripheral (for read cycles). After a period of time, which is determined by the access time of the memory/peripheral, data become valid.

**Read cycles:** Input data is latched and the command signal is now deactivated. This causes the accessed device to remove its data from the bus which is then tri-stated again.

**Write cycles:** The command signal is now deactivated. The data remain valid on the bus until the next external bus cycle is started.

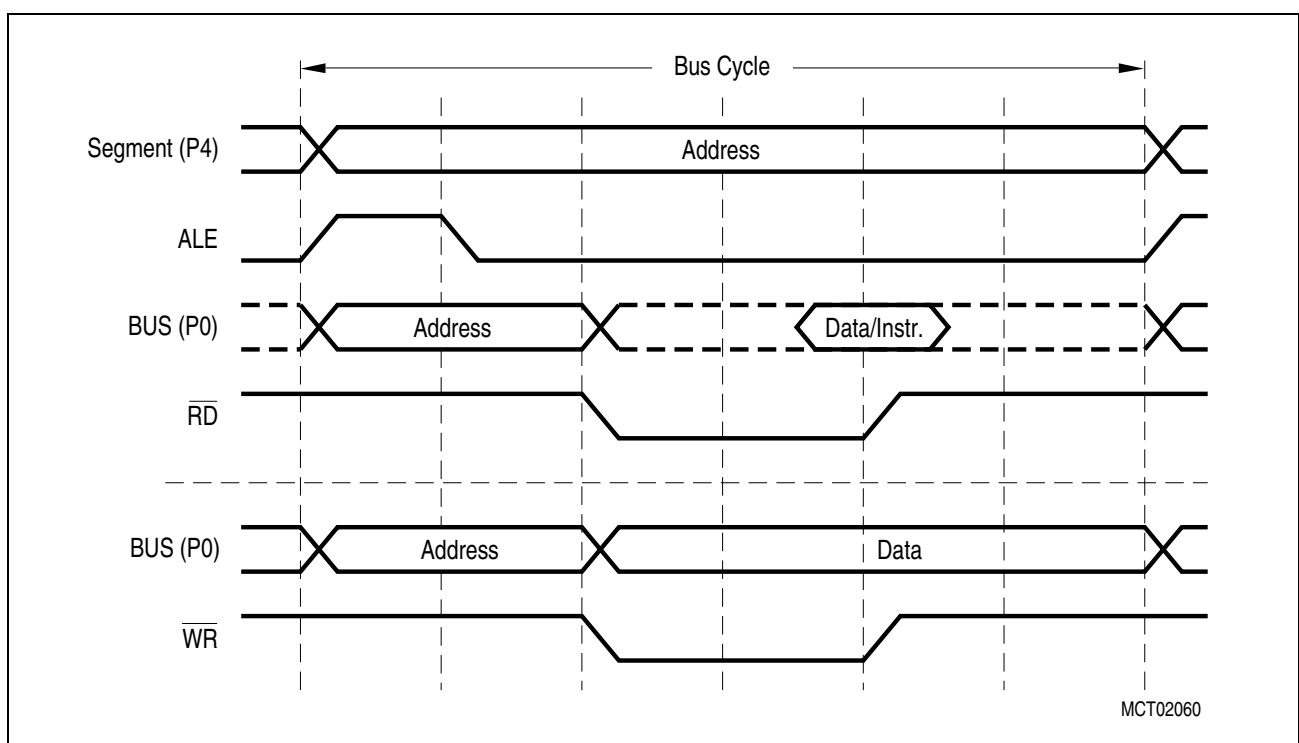


Figure 9-2 Multiplexed Bus Cycle

### Demultiplexed Bus Modes

In the demultiplexed bus modes the 16-bit intra-segment address is permanently output on PORT1, while the data uses PORT0 (16-bit data) or P0L (8-bit data).

The upper address lines are permanently output on Port 4 (if selected via SALSEL during reset). No address latches are required.

The EBC initiates an external access by placing an address on the address bus. After a programmable period of time the EBC activates the respective command signal ( $\overline{RD}$ ,  $\overline{WR}$ ,  $\overline{WRL}$ ,  $\overline{WRH}$ ). Data is driven onto the data bus either by the EBC (for write cycles) or by the external memory/peripheral (for read cycles). After a period of time, which is determined by the access time of the memory/peripheral, data become valid.

**Read cycles:** Input data is latched and the command signal is now deactivated. This causes the accessed device to remove its data from the data bus which is then tri-stated again.

**Write cycles:** The command signal is now deactivated. If a subsequent external bus cycle is required, the EBC places the respective address on the address bus. The data remain valid on the bus until the next external bus cycle is started.

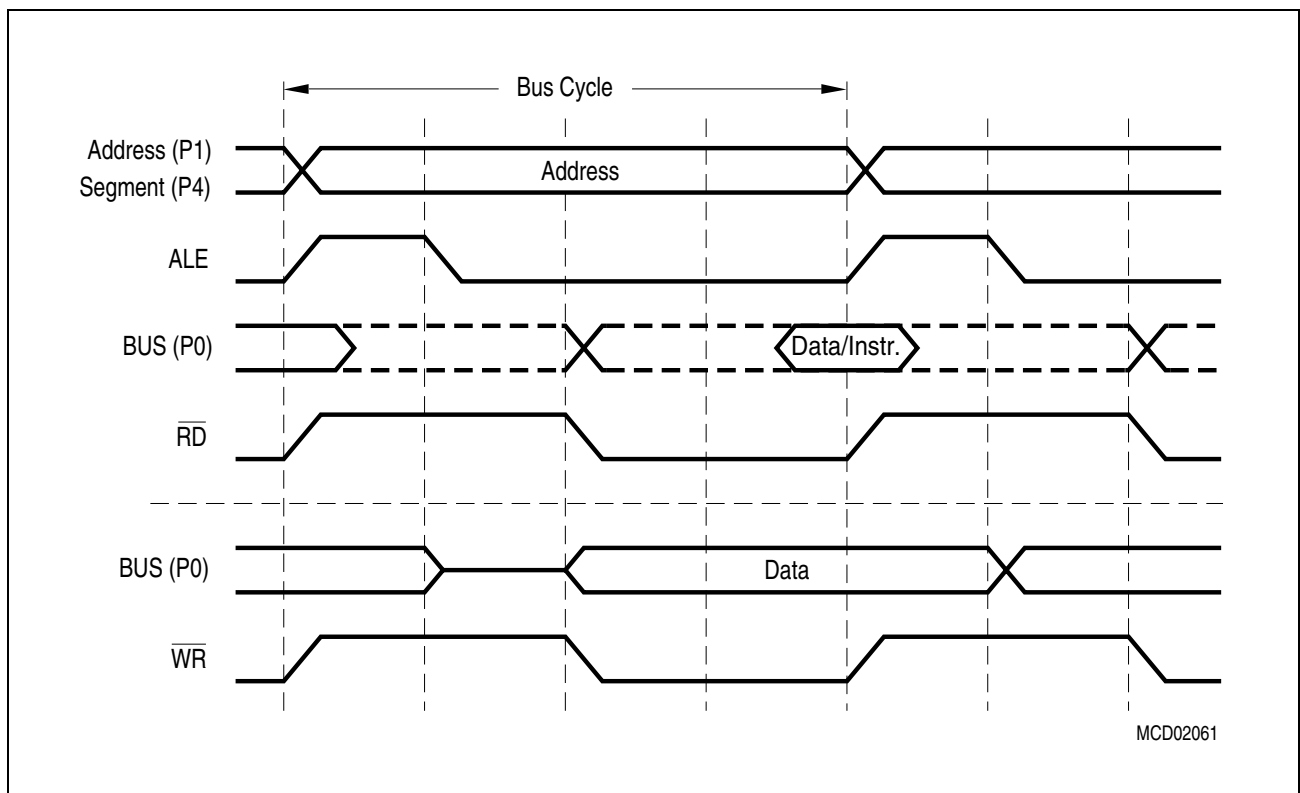


Figure 9-3 Demultiplexed Bus Cycle

### Switching Between the Bus Modes

The EBC allows to switch between different bus modes dynamically, i.e. subsequent external bus cycles may be executed in different ways. Certain address areas may use multiplexed or demultiplexed buses or use  $\overline{\text{READY}}$  control or predefined waitstates.

A change of the external bus characteristics can be initiated in two different ways:

**Switching between predefined address windows** automatically selects the bus mode that is associated with the respective window. Predefined address windows allow to use different bus modes without any overhead, but restrict their number to the number of BUSCONs. However, as BUSCON0 controls all address areas, which are not covered by the other BUSCONs, this allows to have gaps between these windows, which use the bus mode of BUSCON0.

PORT1 will output the intra-segment address, when any of the BUSCON registers selects a demultiplexed bus mode, even if the current bus cycle uses a multiplexed bus mode. This allows to have an external address decoder connected to PORT1 only, while using it for all kinds of bus cycles.

The usage of the BUSCON/ADDRSEL registers is controlled via the issued addresses. When an access (code fetch or data) is initiated, the respective generated physical address defines, if the access is made internally, uses one of the address windows defined by ADDRSEL4 ... 1, or uses the default configuration in BUSCON0. After initializing the active registers, they are selected and evaluated automatically by interpreting the physical address. No additional switching or selecting is necessary during run time, except when more than the four address windows plus the default is to be used.

**Reprogramming the BUSCON and/or ADDRSEL registers** allows to either change the bus mode for a given address window, or change the size of an address window that uses a certain bus mode. Reprogramming allows to use a great number of different address windows (more than BUSCONs are available) on the expense of the overhead for changing the registers and keeping appropriate tables.

*Note: Be careful when changing the configuration for an address area that currently supplies the instruction stream. Due to the internal pipelining, the first instruction fetch that will use the new configuration depends on the instructions prior to the configuration change. Special care is required when changing bits like BUSACT or RDYEN, in order not to cut the instruction stream inadvertently.*

*Only change the other configuration bits after checking that the respective application can cope with the intended modification(s).*

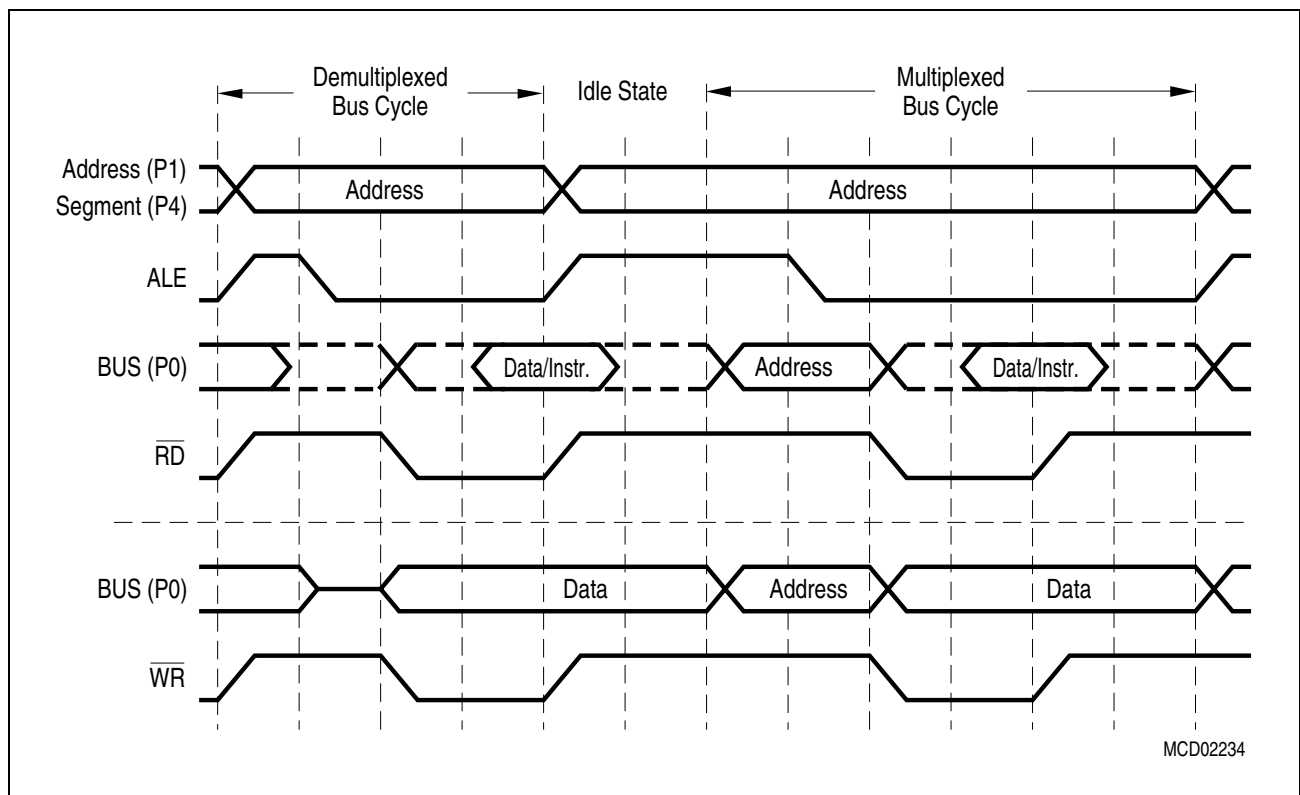
*It is recommended to change ADDRSEL registers only while the respective BUSACT bit in the associated BUSCON register is cleared.*

**Switching from demultiplexed to multiplexed bus mode** represents a special case. The bus cycle is started by activating ALE and driving the address to Port 4 and PORT1 as usual, if another BUSCON register selects a demultiplexed bus. However, in the

## The External Bus Interface

multiplexed bus modes the address is also required on PORT0. In this special case the address on PORT0 is delayed by one CPU clock cycle, which delays the complete (multiplexed) bus cycle and extends the corresponding ALE signal (see [Figure 9-4](#)).

This extra time is required to allow the previously selected device (via demultiplexed bus) to release the data bus, which would be available in a demultiplexed bus cycle.



**Figure 9-4 Switching from Demultiplexed to Multiplexed Bus Mode**

**Switching between external resources** (e.g. different peripherals) may incur a problem if the previously accessed resource needs some time to switch of its output drivers (after a read) and the resource to be accessed next switches on its output drivers very fast. In systems running on higher frequencies this may lead to a bus conflict (the switch off delays normally are independent from the clock frequency).

In such a case an additional waitstate can automatically be inserted when leaving a certain address window, i.e. when the next cycle accesses a different window. This waitstate is controlled in the same way as the waitstate when switching from demultiplexed to multiplexed bus mode, see [Figure 9-4](#).

BUSCON switch waitstates are enabled via bits BSWCx in the BUSCON registers. By enabling the automatic BUSCON switch waitstate (BSWCx = '1') there is no impact on the system performance as long as the external bus cycles access the same address window. Only if the following cycle accesses a different window a waitstate is inserted between the last access to the previous window and the first access to the new window. After reset no BUSCON switch waitstates are selected.

### External Data Bus Width

The EBC can operate on 8-bit or 16-bit wide external memory/peripherals. A 16-bit data bus uses PORT0, while an 8-bit data bus only uses P0L, the lower byte of PORT0. This saves on address latches, bus transceivers, bus routing and memory cost on the expense of transfer time. The EBC can control word accesses on an 8-bit data bus as well as byte accesses on a 16-bit data bus.

**Word accesses on an 8-bit data bus** are automatically split into two subsequent byte accesses, where the low byte is accessed first, then the high byte. The assembly of bytes to words and the disassembly of words into bytes is handled by the EBC and is transparent to the CPU and the programmer.

**Byte accesses on a 16-bit data bus** require that the upper and lower half of the memory can be accessed individually. In this case the upper byte is selected with the  $\overline{\text{BHE}}$  signal, while the lower byte is selected with the A0 signal. So the two bytes of the memory can be enabled independent from each other, or together when accessing words.

When writing bytes to an external 16-bit device, which has a single  $\overline{\text{CS}}$  input, but two  $\overline{\text{WR}}$  enable inputs (for the two bytes), the EBC can directly generate these two write control signals. This saves the external combination of the  $\overline{\text{WR}}$  signal with A0 or  $\overline{\text{BHE}}$ . In this case pin  $\overline{\text{WR}}$  serves as  $\overline{\text{WRL}}$  (write low byte) and pin  $\overline{\text{BHE}}$  serves as  $\overline{\text{WRH}}$  (write high byte). Bit WRCFG in register SYSCON selects the operating mode for pins  $\overline{\text{WR}}$  and  $\overline{\text{BHE}}$ . The respective byte will be written on both data bus halves.

When reading bytes from an external 16-bit device, whole words may be read and the C161CS/JC/JI automatically selects the byte to be input and discards the other. However, care must be taken when reading devices that change state when being read, like FIFOs, interrupt status registers, etc. In this case individual bytes should be selected using  $\overline{\text{BHE}}$  and A0.

**Table 9-2 Bus Mode versus Performance**

Bus Mode	Transfer Rate (Speed factor for byte/word/dword access)	System Requirements	Free IO Lines
8-bit Multiplexed	Very low (1.5/3/6)	Low (8-bit latch, byte bus)	P1H, P1L
8-bit Demultipl.	Low (1/2/4)	Very low (no latch, byte bus)	P0H
16-bit Multiplexed	High (1.5/1.5/3)	High (16-bit latch, word bus)	P1H, P1L
16-bit Demultipl.	Very high (1/1/2)	Low (no latch, word bus)	–

*Note: PORT1 becomes available for general purpose IO, when none of the BUSCON registers selects a demultiplexed bus mode.*

**Disable/Enable Control for Pin  $\overline{\text{BHE}}$  (BYTDIS)**

Bit BYTDIS is provided for controlling the active low Byte High Enable ( $\overline{\text{BHE}}$ ) pin. The function of the  $\overline{\text{BHE}}$  pin is enabled, if the BYTDIS bit contains a '0'. Otherwise, it is disabled and the pin can be used as standard IO pin. The  $\overline{\text{BHE}}$  pin is implicitly used by the External Bus Controller to select one of two byte-organized memory chips, which are connected to the C161CS/JC/JI via a word-wide external data bus. After reset the  $\overline{\text{BHE}}$  function is automatically enabled (BYTDIS = '0'), if a 16-bit data bus is selected during reset, otherwise it is disabled (BYTDIS = '1'). It may be disabled, if byte access to 16-bit memory is not required, and the  $\overline{\text{BHE}}$  signal is not used.

**Segment Address Generation**

During external accesses the EBC generates a (programmable) number of address lines on Port 4, which extend the 16-bit address output on PORT0 or PORT1 and so increase the accessible address space. The number of segment address lines is selected during reset and coded in bit field SALSEL in register RP0<sub>H</sub> (see [Table 9-3](#)).

**Table 9-3 Decoding of Segment Address Lines**

SALSEL	Segment Address Lines	Directly accessible Address Space
1 1	Two: A17 ... A16	256 KByte (Default without pull-downs)
1 0	Eight: A23 ... A16	16 MByte (Maximum)
0 1	None	64 KByte (Minimum)
0 0	Four: A19 ... A16	1 MByte

*Note: The total accessible address space may be increased by accessing several banks which are distinguished by individual chip select lines.*

*If Port 4 is used to output segment address lines, in most cases the drivers must operate in push/pull mode. Make sure that OPD4 does not select open drain mode in this case.*

### $\overline{CS}$ Signal Generation

During external accesses the EBC can generate a (programmable) number of  $\overline{CS}$  lines on Port 6, which allow to directly select external peripherals or memory banks without requiring an external decoder. The number of  $\overline{CS}$  lines is selected during reset and coded in bit field CSSEL in register RP0H (see [Table 9-4](#)).

**Table 9-4 Decoding of Chip Select Lines**

CSSEL	Chip Select Lines	Note
1 1	Five: $\overline{CS4} \dots \overline{CS0}$	Default without pull-downs
1 0	None	Port 6 pins free for IO
0 1	Two: $\overline{CS1} \dots \overline{CS0}$	–
0 0	Three: $\overline{CS2} \dots \overline{CS0}$	–

The  $\overline{CSx}$  outputs are associated with the BUSCONx registers and are driven active (low) for any access within the address area defined for the respective BUSCON register. For any access outside this defined address area the respective  $\overline{CSx}$  signal will go inactive (high). At the beginning of each external bus cycle the corresponding valid  $\overline{CS}$  signal is determined and activated. All other  $\overline{CS}$  lines are deactivated (driven high) at the same time.

*Note: The  $\overline{CSx}$  signals will not be updated for an access to any internal address area (i.e. when no external bus cycle is started), even if this area is covered by the respective ADDRSELx register. An access to an on-chip X-Peripheral deactivates all external  $\overline{CS}$  signals.*

*Upon accesses to address windows without a selected  $\overline{CS}$  line all selected  $\overline{CS}$  lines are deactivated.*

The chip select signals allow to be operated in four different modes (see [Table 9-5](#)) which are selected via bits CSWENx and CSRENx in the respective BUSCONx register.

**Table 9-5 Chip Select Generation Modes**

CSWENx	CSRENx	Chip Select Mode
0	0	Address Chip Select (Default after Reset)
0	1	Read Chip Select
1	0	Write Chip Select
1	1	Read/Write Chip Select



**The External Bus Interface**

**Read or Write Chip Select** signals remain active only as long as the associated control signal ( $\overline{RD}$  or  $\overline{WR}$ ) is active. This also includes the programmable read/write delay. Read chip select is only activated for read cycles, write chip select is only activated for write cycles, read/write chip select is activated for both read and write cycles (write cycles are assumed, if any of the signals  $\overline{WRH}$  or  $\overline{WRL}$  gets active). These modes save external glue logic, when accessing external devices like latches or drivers that only provide a single enable input.

**Address Chip Select** signals remain active during the complete bus cycle. For address chip select signals two generation modes can be selected via bit CSCFG in register SYSCON:

- A **latched** address chip select signal (CSCFG = '0') becomes active with the falling edge of ALE and becomes inactive at the beginning of an external bus cycle that accesses a different address window. No spikes will be generated on the chip select lines and no changes occur as long as locations within the same address window or within internal memory (excluding X-Peripherals and XRAM) are accessed.
- An **early** address chip select signal (CSCFG = '1') becomes active together with the address and  $\overline{BHE}$  (if enabled) and remains active until the end of the current bus cycle. Early address chip select signals are not latched internally and may toggle intermediately while the address is changing.

*Note:  $\overline{CS0}$  provides a latched address chip select directly after reset (except for single chip mode) when the first instruction is fetched.*

Internal pullup devices hold all  $\overline{CS}$  lines high during reset. After the end of a reset sequence the pullup devices are switched off and the pin drivers control the pin levels on the selected  $\overline{CS}$  lines. Not selected  $\overline{CS}$  lines will enter the high-impedance state and are available for general purpose IO.

The pullup devices are also active during bus hold on the selected  $\overline{CS}$  lines, while  $\overline{HLDA}$  is active and the respective pin is switched to push/pull mode. Open drain outputs will float during bus hold. In this case external pullup devices are required or the new bus master is responsible for driving appropriate levels on the  $\overline{CS}$  lines.

**Segment Address versus Chip Select**

The external bus interface of the C161CS/JC/JI supports many configurations for the external memory. By increasing the number of segment address lines the C161CS/JC/JI can address a linear address space of 256 KByte, 1 MByte or 16 MByte. This allows to implement a large sequential memory area, and also allows to access a great number of external devices, using an external decoder. By increasing the number of  $\overline{CS}$  lines the C161CS/JC/JI can access memory banks or peripherals without external glue logic. These two features may be combined to optimize the overall system performance.

*Note: Bit SGTDIS of register SYSCON defines, if the CSP register is saved during interrupt entry (segmentation active) or not (segmentation disabled).*



### 9.3 Programmable Bus Characteristics

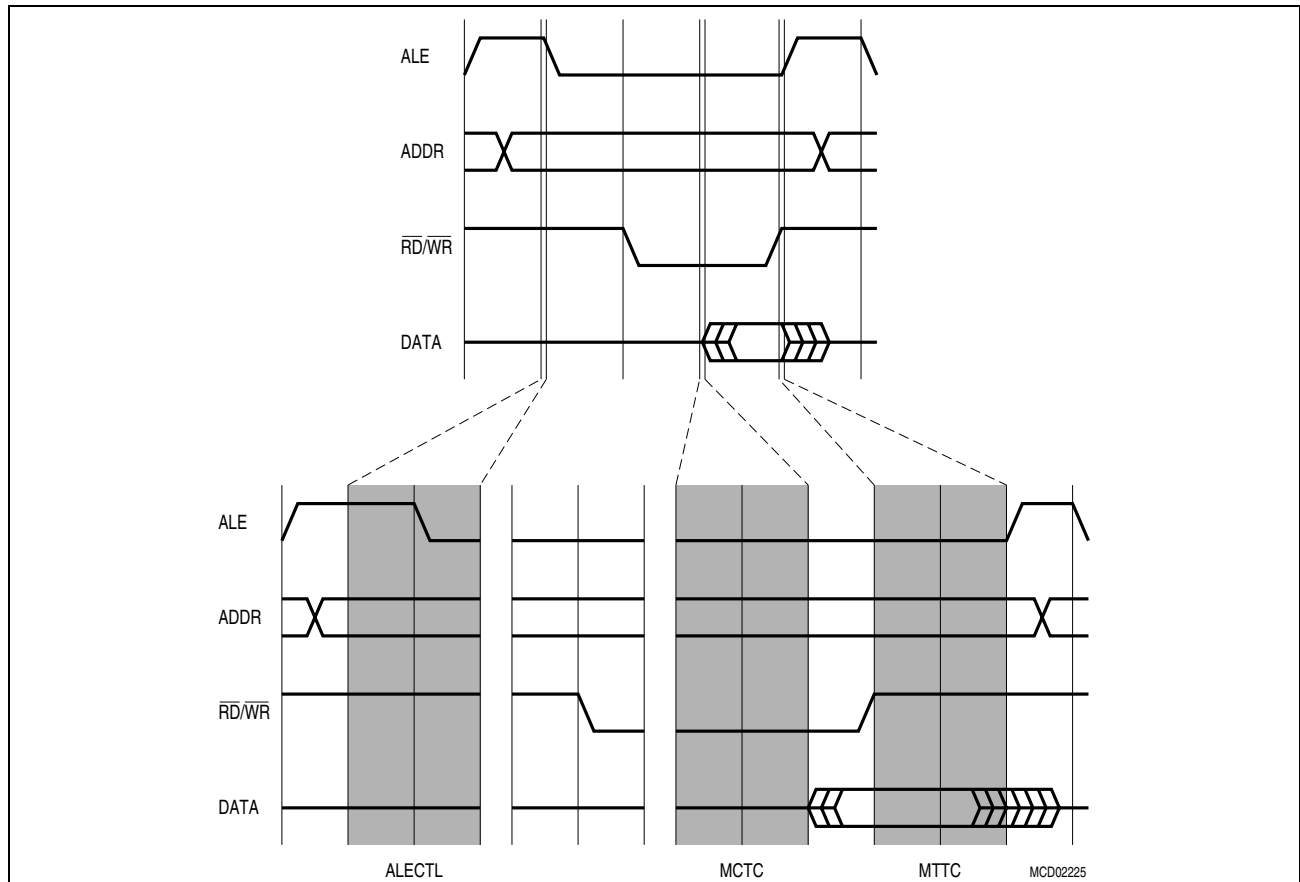
Important timing characteristics of the external bus interface have been made user programmable to allow to adapt it to a wide range of different external bus and memory configurations with different types of memories and/or peripherals.

The following parameters of an external bus cycle are programmable:

- **ALE Control** defines the ALE signal length and the address hold time after its falling edge
- **Memory Cycle Time** (extendable with 1 ... 15 waitstates) defines the allowable access time
- **Memory Tri-State Time** (extendable with 1 waitstate) defines the time for a data driver to float
- **Read/Write Delay Time** defines when a command is activated after the falling edge of ALE
- **READY Control** defines, if a bus cycle is terminated internally or externally

*Note: Internal accesses are executed with maximum speed and therefore are not programmable.*

*External accesses use the slowest possible bus cycle after reset. The bus cycle timing may then be optimized by the initialization software.*



**Figure 9-5 Programmable External Bus Cycle**

### ALE Length Control

The length of the ALE signal and the address hold time after its falling edge are controlled by the ALECTLx bits in the BUSCON registers. When bit ALECTL is set to '1', external bus cycles accessing the respective address window will have their ALE signal prolonged by half a CPU clock (1 TCL). Also the address hold time after the falling edge of ALE (on a multiplexed bus) will be prolonged by half a CPU clock, so the data transfer within a bus cycle refers to the same CLKOUT edges as usual (i.e. the data transfer is delayed by one CPU clock). This allows more time for the address to be latched.

*Note: ALECTL0 is '1' after reset to select the slowest possible bus cycle, the other ALECTLx are '0' after reset.*

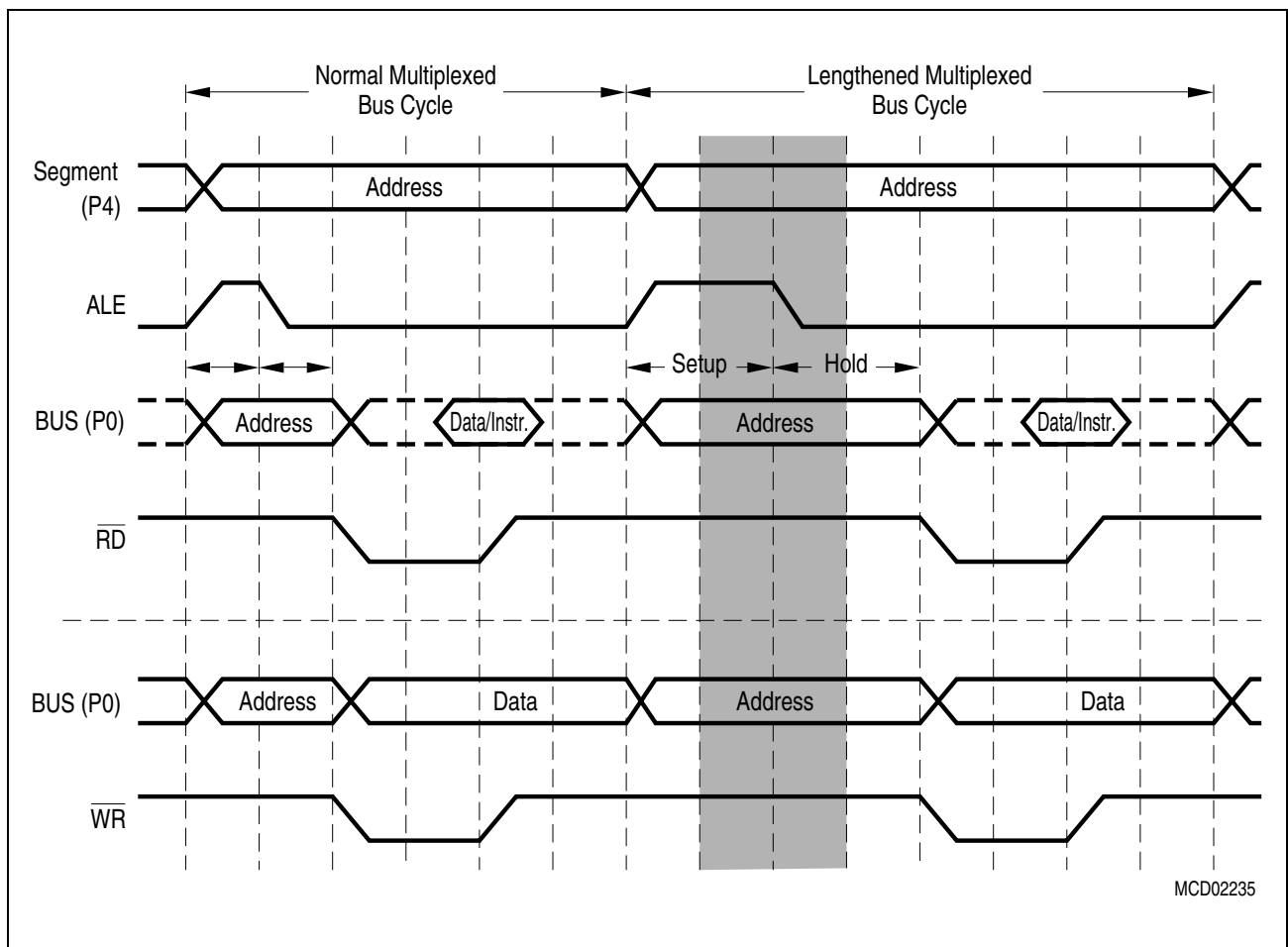
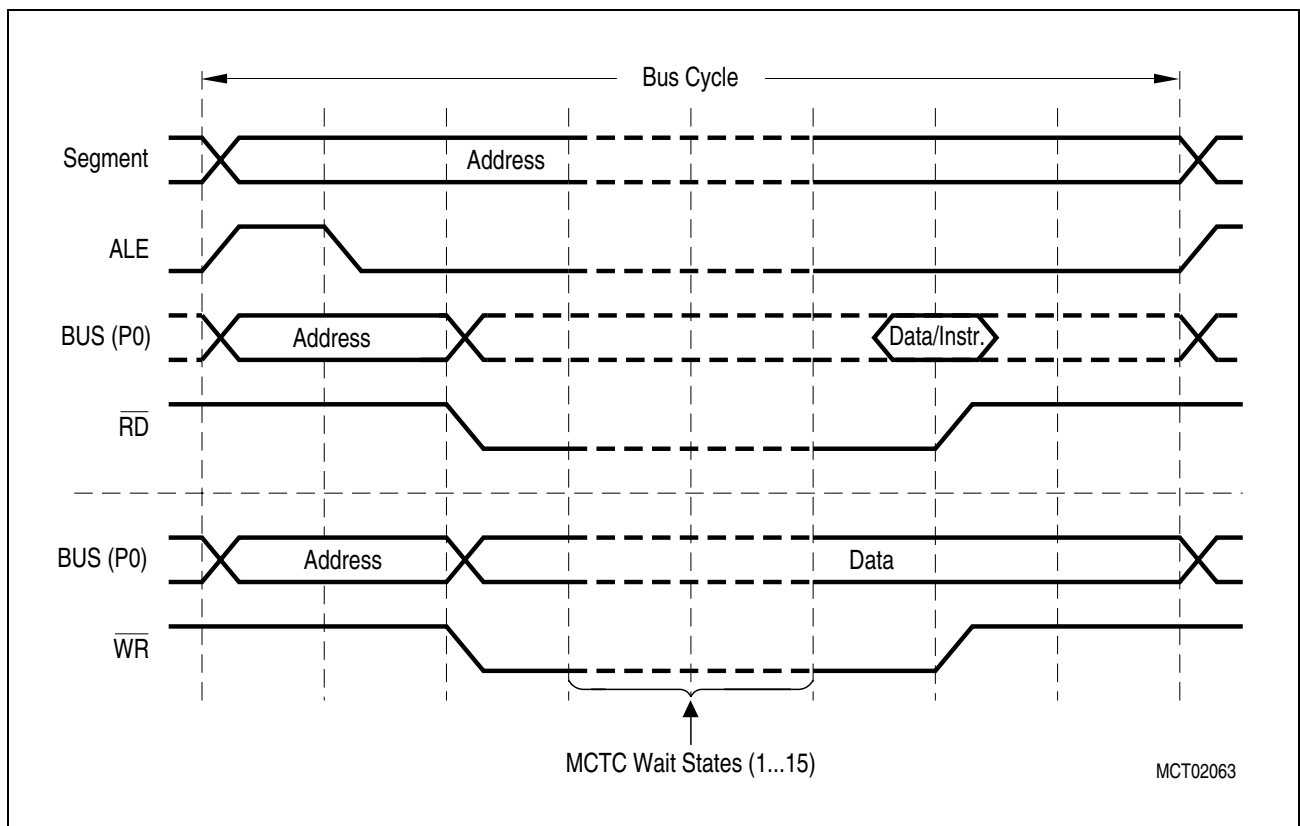


Figure 9-6 ALE Length Control

### Programmable Memory Cycle Time

The C161CS/JC/JI allows the user to adjust the controller's external bus cycles to the access time of the respective memory or peripheral. This access time is the total time required to move the data to the destination. It represents the period of time during which the controller's signals do not change.



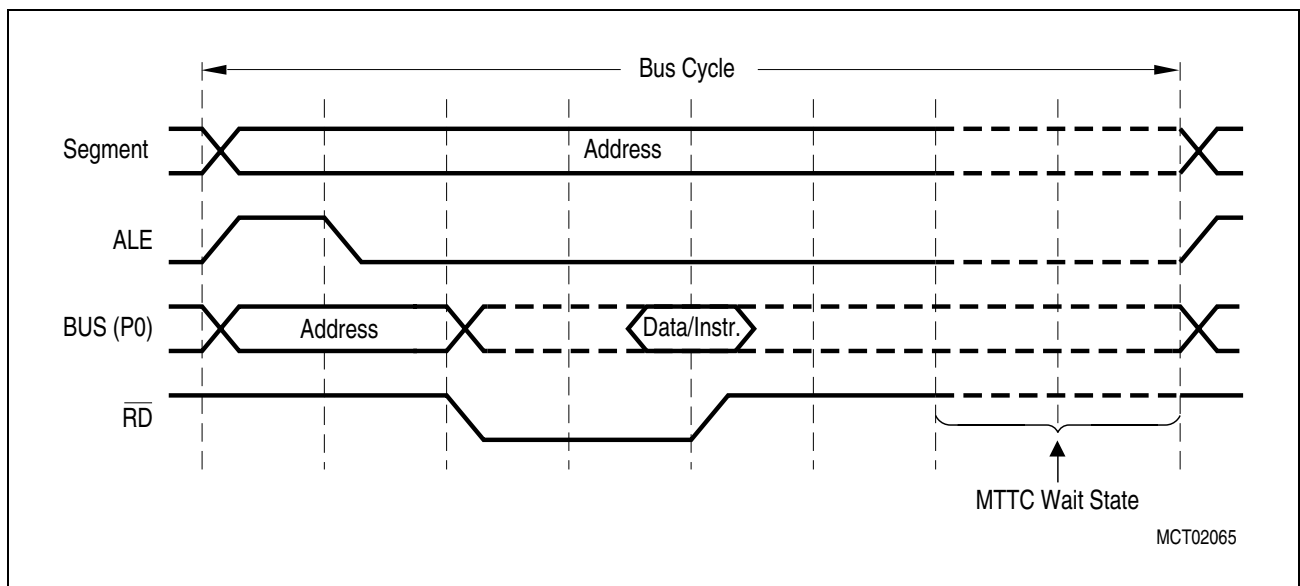
**Figure 9-7 Memory Cycle Time**

The external bus cycles of the C161CS/JC/JI can be extended for a memory or peripheral, which cannot keep pace with the controller's maximum speed, by introducing wait states during the access (see [Figure 9-7](#)). During these memory cycle time wait states, the CPU is idle, if this access is required for the execution of the current instruction.

The memory cycle time wait states can be programmed in increments of one CPU clock (2 TCL) within a range from 0 to 15 (default after reset) via the MCTC fields of the BUSCON registers. 15 - <MCTC> waitstates will be inserted.

### Programmable Memory Tri-state Time

The C161CS/JC/JI allows the user to adjust the time between two subsequent external accesses to account for the tri-state time of the external device. The tri-state time defines, when the external device has released the bus after deactivation of the read command ( $\overline{RD}$ ).



**Figure 9-8 Memory Tri-state Time**

The output of the next address on the external bus can be delayed for a memory or peripheral, which needs more time to switch off its bus drivers, by introducing a wait state after the previous bus cycle (see [Figure 9-8](#)). During this memory tri-state time wait state, the CPU is not idle, so CPU operations will only be slowed down if a subsequent external instruction or data fetch operation is required during the next instruction cycle.

The memory tri-state time waitstate requires one CPU clock (2 TCL) and is controlled via the MTTCx bits of the BUSCON registers. A waitstate will be inserted, if bit MTTCx is '0' (default after reset).

*Note: External bus cycles in multiplexed bus modes implicitly add one tri-state time waitstate in addition to the programmable MTTC waitstate.*

### **Read/Write Signal Delay**

The C161CS/JC/JI allows the user to adjust the timing of the read and write commands to account for timing requirements of external peripherals. The read/write delay controls the time between the falling edge of ALE and the falling edge of the command. Without read/write delay the falling edges of ALE and command(s) are coincident (except for propagation delays). With the delay enabled, the command(s) become active half a CPU clock (1 TCL) after the falling edge of ALE.

The read/write delay does not extend the memory cycle time, and does not slow down the controller in general. In multiplexed bus modes, however, the data drivers of an external device may conflict with the C161CS/JC/JI's address, when the early  $\overline{RD}$  signal is used. Therefore multiplexed bus cycles should always be programmed with read/write delay.

The read/write delay is controlled via the RWDCx bits in the BUSCON registers. The command(s) will be delayed, if bit RWDCx is '0' (default after reset).

### **Early $\overline{WR}$ Signal Deactivation**

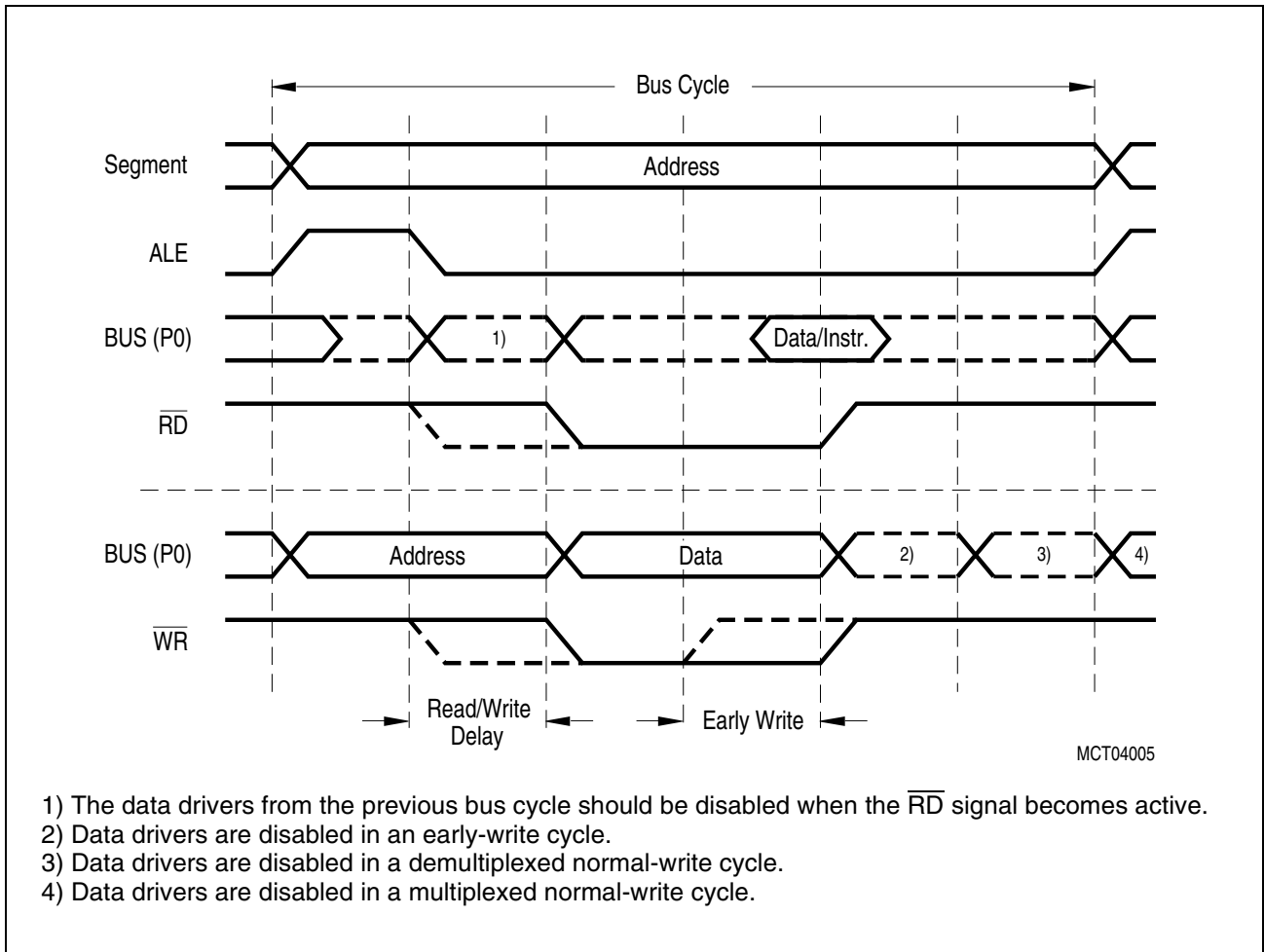
The duration of an external write access can be shortened by one TCL. The  $\overline{WR}$  signal is activated (driven low) in the standard way, but can be deactivated (driven high) one TCL earlier than defined in the standard timing. In this case also the data output drivers will be deactivated one TCL earlier.

This is especially useful in systems which operate on higher CPU clock frequencies and employ external modules (memories, peripherals, etc.) which switch on their own data drivers very fast in response to e.g. a chip select signal.

Conflicts between the C161CS/JC/JI's and the external peripheral's output drivers can be avoided then by selecting early  $\overline{WR}$  for the C161CS/JC/JI.

*Note: Make sure that the reduced  $\overline{WR}$  low time then still matches the requirements of the external peripheral/memory.*

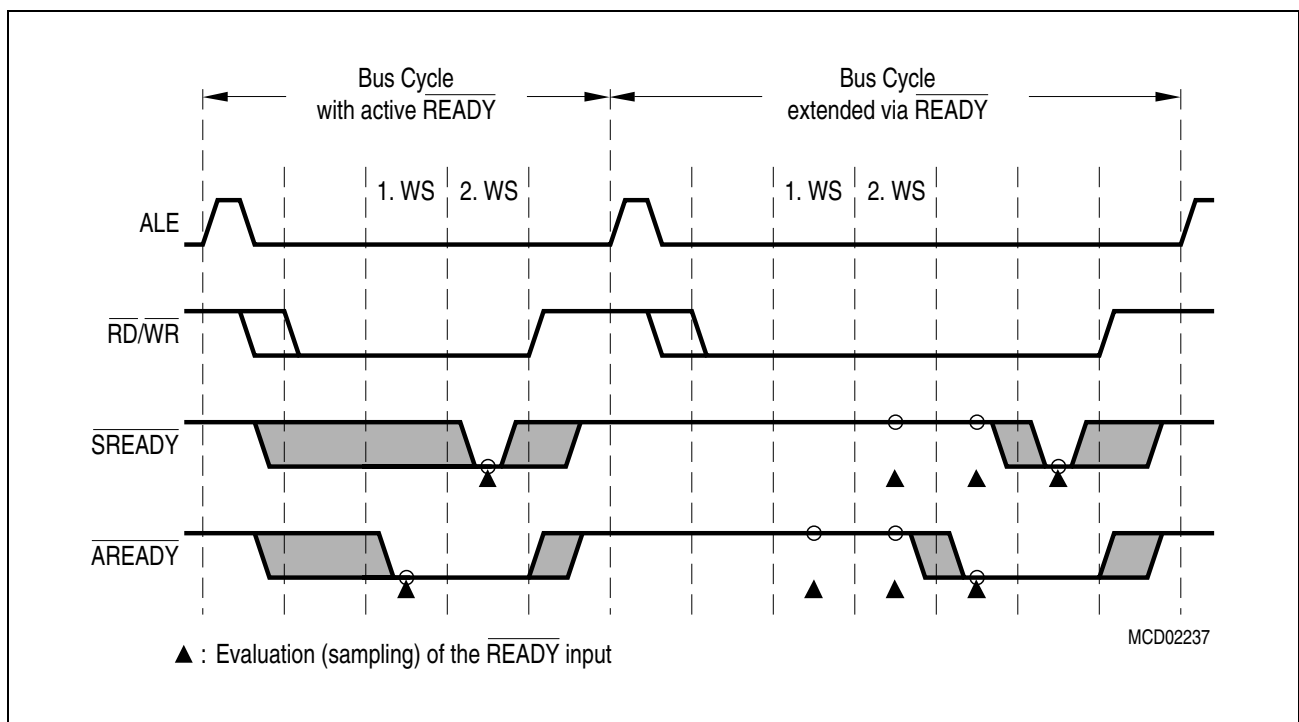
Early  $\overline{WR}$  deactivation is controlled via the EWENx bits in the BUSCON registers. The  $\overline{WR}$  signal will be shortened if bit EWENx is '1' (default after reset is a standard  $\overline{WR}$  signal, i.e. EWENx = '0').



**Figure 9-9 Read/Write Signal Duration Control**

## 9.4 $\overline{\text{READY}}$ Controlled Bus Cycles

For situations, where the programmable waitstates are not enough, or where the response (access) time of a peripheral is not constant, the C161CS/JC/JI provides external bus cycles that are terminated via a  $\overline{\text{READY}}$  input signal (synchronous or asynchronous). In this case the C161CS/JC/JI first inserts a programmable number of waitstates (0 ... 7) and then monitors the  $\overline{\text{READY}}$  line to determine the actual end of the current bus cycle. The external device drives  $\overline{\text{READY}}$  low in order to indicate that data have been latched (write cycle) or are available (read cycle).



**Figure 9-10  $\overline{\text{READY}}$  Controlled Bus Cycles**

The  $\overline{\text{READY}}$  function is enabled via the RDYENx bits in the BUSCON registers. When this function is selected (RDYENx = '1'), only the lower 3 bits of the respective MCTC bit field define the number of inserted waitstates (0 ... 7), while the MSB of bit field MCTC selects the  $\overline{\text{READY}}$  operation:

MCTC.3 = '0': Synchronous  $\overline{\text{READY}}$ , i.e. the  $\overline{\text{READY}}$  signal must meet setup and hold times.

MCTC.3 = '1': Asynchronous  $\overline{\text{READY}}$ , i.e. the  $\overline{\text{READY}}$  signal is synchronized internally.

## The External Bus Interface

**The Synchronous  $\overline{\text{READY}}$**  provides the fastest bus cycles, but requires setup and hold times to be met. The CLKOUT signal **should be enabled** and may be used by the peripheral logic to control the  $\overline{\text{READY}}$  timing in this case.

**The Asynchronous  $\overline{\text{READY}}$**  is less restrictive, but requires additional waitstates caused by the internal synchronization. As the asynchronous  $\overline{\text{READY}}$  is sampled earlier (see [Figure 9-10](#)) programmed waitstates may be necessary to provide proper bus cycles (see also notes on “normally-ready” peripherals below).

A  $\overline{\text{READY}}$  signal (especially asynchronous  $\overline{\text{READY}}$ ) that has been activated by an external device may be deactivated in response to the trailing (rising) edge of the respective command ( $\overline{\text{RD}}$  or  $\overline{\text{WR}}$ ).

*Note: When the  $\overline{\text{READY}}$  function is enabled for a specific address window, each bus cycle within this window must be terminated with an active  $\overline{\text{READY}}$  signal. Otherwise the controller hangs until the next reset. A timeout function is only provided by the watchdog timer.*

**Combining the  $\overline{\text{READY}}$  function with predefined waitstates** is advantageous in two cases:

Memory components with a fixed access time and peripherals operating with  $\overline{\text{READY}}$  may be grouped into the same address window. The (external) waitstate control logic in this case would activate  $\overline{\text{READY}}$  either upon the memory's chip select or with the peripheral's  $\overline{\text{READY}}$  output. After the predefined number of waitstates the C161CS/JC/JI will check its  $\overline{\text{READY}}$  line to determine the end of the bus cycle. For a memory access it will be low already (see example a) in [Figure 9-10](#)), for a peripheral access it may be delayed (see example b) in [Figure 9-10](#)). As memories tend to be faster than peripherals, there should be no impact on system performance.

When using the  $\overline{\text{READY}}$  function with so-called “normally-ready” peripherals, it may lead to erroneous bus cycles, if the  $\overline{\text{READY}}$  line is sampled too early. These peripherals pull their  $\overline{\text{READY}}$  output low, while they are idle. When they are accessed, they deactivate  $\overline{\text{READY}}$  until the bus cycle is complete, then drive it low again. If, however, the peripheral deactivates  $\overline{\text{READY}}$  **after** the first sample point of the C161CS/JC/JI, the controller samples an active  $\overline{\text{READY}}$  and terminates the current bus cycle, which, of course, is too early. By inserting predefined waitstates the first  $\overline{\text{READY}}$  sample point can be shifted to a time, where the peripheral has safely controlled the  $\overline{\text{READY}}$  line (e.g. after 2 waitstates in [Figure 9-10](#)).



## 9.5 Controlling the External Bus Controller

A set of registers controls the functions of the EBC. General features like the usage of interface pins ( $\overline{WR}$ ,  $\overline{BHE}$ ), segmentation and internal ROM mapping are controlled via register SYSCON. The properties of a bus cycle like chip select mode, usage of  $\overline{READY}$ , length of ALE, external bus mode, read/write delay and waitstates are controlled via registers BUSCON4 ... BUSCON0. Four of these registers (BUSCON4 ... BUSCON1) have an address select register (ADDRSEL4 ... ADDRSEL1) associated with them, which allows to specify up to four address areas and the individual bus characteristics within these areas. All accesses that are not covered by these four areas are then controlled via BUSCON0. This allows to use memory components or peripherals with different interfaces within the same system, while optimizing accesses to each of them.

### SYSCON

System Control Register

SFR (FF12<sub>H</sub>/89<sub>H</sub>)

Reset Value: 0XX0<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STKSZ		ROM S1	SGT DIS	ROM EN	BYT DIS	CLK EN	WR CFG	CS CFG	-	OWD DIS	BD RST EN	XPEN	VISI-BLE	XPEN	SHARE
rw		rw	rw	rwh	rwh	rw	rwh	rw	-	rwh	rw	rw	rw	rw	rw

Bit	Function
XPEN-SHARE	<b>XBUS Peripheral Share Mode Control</b> 0: External accesses to XBUS peripherals are disabled 1: XBUS peripherals are accessible via the ext. bus during hold mode
VISIBLE	<b>Visible Mode Control</b> 0: Accesses to XBUS peripherals are done internally 1: XBUS peripheral accesses are made visible on the external pins
XPEN	<b>XBUS Peripheral Enable Bit</b> 0: Accesses to the on-chip X-Peripherals and their functions are disabled 1: The on-chip X-Peripherals are enabled and can be accessed
BDRSTEN	<b>Bidirectional Reset Enable Bit</b> 0: Pin $\overline{RSTIN}$ is an input only 1: Pin $\overline{RSTIN}$ is pulled low during the internal reset sequence after any reset
OWDDIS	<b>Oscillator Watchdog Disable Bit</b> (Configured via pin $\overline{RD}$ upon a reset) 0: The on-chip oscillator watchdog is enabled and active 1: The on-chip oscillator watchdog is disabled and the CPU clock is always fed from the oscillator input

**The External Bus Interface**

Bit	Function
<b>CSCFG</b>	<b>Chip Select Configuration Control</b> (Cleared after reset) 0: Latched $\overline{CS}$ mode. The $\overline{CS}$ signals are latched internally and driven to the (enabled) port pins synchronously. 1: Unlatched $\overline{CS}$ mode. The $\overline{CS}$ signals are directly derived from the address and driven to the (enabled) port pins.
<b>WRCFG</b>	<b>Write Configuration Control</b> (Configured via pin P0H.0 upon a reset) 0: Pins $\overline{WR}$ and $\overline{BHE}$ retain their normal function 1: Pin $\overline{WR}$ acts as $\overline{WRL}$ , pin $\overline{BHE}$ acts as $\overline{WRH}$
<b>CLKEN</b>	<b>System Clock Output Enable</b> (CLKOUT, cleared after reset) 0: CLKOUT disabled: pin may be used for FOUT or gen. purpose IO 1: CLKOUT enabled: pin outputs the system clock signal
<b>BYTDIS</b>	<b>Disable/Enable Control for Pin <math>\overline{BHE}</math></b> (Set according to data bus width) 0: Pin $\overline{BHE}$ enabled 1: Pin $\overline{BHE}$ disabled, pin may be used for general purpose IO
<b>ROMEN</b>	<b>Internal ROM Enable</b> (Set according to pin $\overline{EA}$ during reset) 0: Internal program memory disabled, accesses to the ROM area use the external bus 1: Internal program memory enabled
<b>SGTDIS</b>	<b>Segmentation Disable/Enable Control</b> (Cleared after reset) 0: Segmentation enabled (CSP is saved/restored during interrupt entry/exit) 1: Segmentation disabled (Only IP is saved/restored)
<b>ROMS1</b>	<b>Internal ROM Mapping</b> 0: Internal ROM area mapped to segment 0 (00'0000 <sub>H</sub> ... 00'7FFF <sub>H</sub> ) 1: Internal ROM area mapped to segment 1 (01'0000 <sub>H</sub> ... 01'7FFF <sub>H</sub> )
<b>STKSZ</b>	<b>System Stack Size</b> Selects the size of the system stack (in the internal RAM) from 32 to 1024 words

*Note: Register SYSCON cannot be changed after execution of the EINIT instruction.  
Bit SGTDIS controls the correct stack operation (push/pop of CSP or not) during traps and interrupts.*

The layout of the BUSCON registers and ADDRSEL registers is identical (respectively). Registers BUSCON4 ... BUSCON1, which control the selected address windows, are completely under software control, while register BUSCON0, which e.g. is also used for the very first code access after reset, is partly controlled by hardware, i.e. it is initialized via PORT0 during the reset sequence. This hardware control allows to define an appropriate external bus for systems, where no internal program memory is provided.

The External Bus Interface

**BUSCON0**

Bus Control Register 0

SFR (FF0C<sub>H</sub>/86<sub>H</sub>)

Reset Value: 0XX0<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSW EN0	CSR EN0	-	RDY EN0	BSW C0	BUS ACT 0	ALE CTL 0	EW EN0	BTYP		MTT C0	RWD C0	MCTC			
rw	rw	-	rw	rw	rwh	rwh	rw	rwh		rw	rw	rw			

**BUSCON1**

Bus Control Register 1

SFR (FF14<sub>H</sub>/8A<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSW EN1	CSR EN1	-	RDY EN1	BSW C1	BUS ACT 1	ALE CTL 1	EW EN1	BTYP		MTT C1	RWD C1	MCTC			
rw	rw	-	rw	rw	rw	rw	rw	rw		rw	rw	rw			

**BUSCON2**

Bus Control Register 2

SFR (FF16<sub>H</sub>/8B<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSW EN2	CSR EN2	-	RDY EN2	BSW C2	BUS ACT 2	ALE CTL 2	EW EN2	BTYP		MTT C2	RWD C2	MCTC			
rw	rw	-	rw	rw	rw	rw	rw	rw		rw	rw	rw			

**BUSCON3**

Bus Control Register 3

SFR (FF18<sub>H</sub>/8C<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSW EN3	CSR EN3	-	RDY EN3	BSW C3	BUS ACT 3	ALE CTL 3	EW EN3	BTYP		MTT C3	RWD C3	MCTC			
rw	rw	-	rw	rw	rw	rw	rw	rw		rw	rw	rw			

**BUSCON4**

Bus Control Register 4

SFR (FF1A<sub>H</sub>/8D<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSW EN4	CSR EN4	-	RDY EN4	BSW C4	BUS ACT 4	ALE CTL 4	EW EN4	BTYP		MTT C4	RWD C4	MCTC			
rw	rw	-	rw	rw	rw	rw	rw	rw		rw	rw	rw			

The External Bus Interface

Bit	Function
<b>MCTC</b>	<p><b>Memory Cycle Time Control</b> (Number of memory cycle time wait states)</p> <p>0000: 15 waitstates            ... (Number = 15 - &lt;MCTC&gt;)            1111: No waitstates</p> <p><i>Note: The definition of bitfield MCTCx changes if RDYENx = '1'            (see <a href="#">Chapter 9.4</a>)</i></p>
<b>RWDCx</b>	<p><b>Read/Write Delay Control for BUSCONx</b></p> <p>0: With rd/wr delay: activate command 1 TCL after falling edge of ALE            1: No rd/wr delay: activate command with falling edge of ALE</p>
<b>MTTCx</b>	<p><b>Memory Tristate Time Control</b></p> <p>0: 1 waitstate            1: No waitstate</p>
<b>BTYP</b>	<p><b>External Bus Configuration</b></p> <p>00: 8-bit Demultiplexed Bus            01: 8-bit Multiplexed Bus            10: 16-bit Demultiplexed Bus            11: 16-bit Multiplexed Bus</p> <p><i>Note: For BUSCON0 BTYP is defined via PORT0 during reset.</i></p>
<b>EWENx</b>	<p><b>Early Write Enable</b></p> <p>0: Normal <math>\overline{WR}</math> signal            1: Early write: <math>\overline{WR}</math> signal is deactivated and write data is tristated one TCL earlier</p>
<b>ALECTLx</b>	<p><b>ALE Lengthening Control</b></p> <p>0: Normal ALE signal            1: Lengthened ALE signal</p>
<b>BUSACTx</b>	<p><b>Bus Active Control</b></p> <p>0: External bus disabled            1: External bus enabled within respective address window (ADDRSEL)</p>
<b>BSWCx</b>	<p><b>BUSCON Switch Control</b></p> <p>0: Address windows are switched immediately            1: A tristate waitstate is inserted if the next bus cycle accesses a different address window than the one controlled by this BUSCON register.<sup>1)</sup></p>
<b>RDYENx</b>	<p><b>READY Input Enable</b></p> <p>0: External bus cycle is controlled by bit field MCTC only            1: External bus cycle is controlled by the <math>\overline{READY}</math> input signal</p>

**The External Bus Interface**

<b>Bit</b>	<b>Function</b>
<b>CSRENx</b>	<b>Read Chip Select Enable</b> 0: The $\overline{CS}$ signal is independent of the read command ( $\overline{RD}$ ) 1: The $\overline{CS}$ signal is generated for the duration of the read command
<b>CSWENx</b>	<b>Write Chip Select Enable</b> 0: The $\overline{CS}$ signal is independent of the write cmd. ( $\overline{WR}, \overline{WRL}, \overline{WRH}$ ) 1: The $\overline{CS}$ signal is generated for the duration of the write command

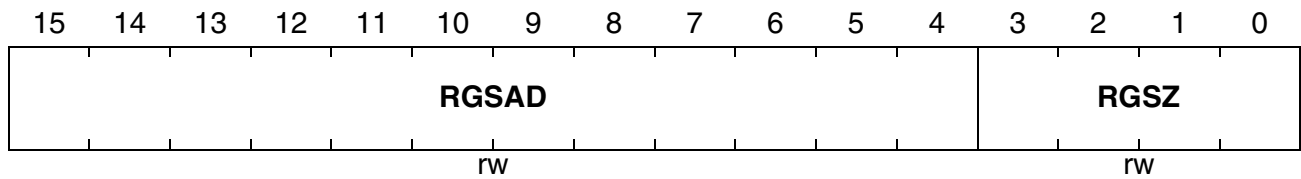
<sup>1)</sup> A BUSCON switch waitstate is enabled by bit BUSCONx.BSWCx of the address window that is left.

*Note: BUSCON0 is initialized with 00C0<sub>H</sub>, if pin  $\overline{EA}$  is high during reset. If pin  $\overline{EA}$  is low during reset, bits BUSACT0 and ALECTL0 are set ('1') and bit field BTYP is loaded with the bus configuration selected via PORT0.*

The External Bus Interface

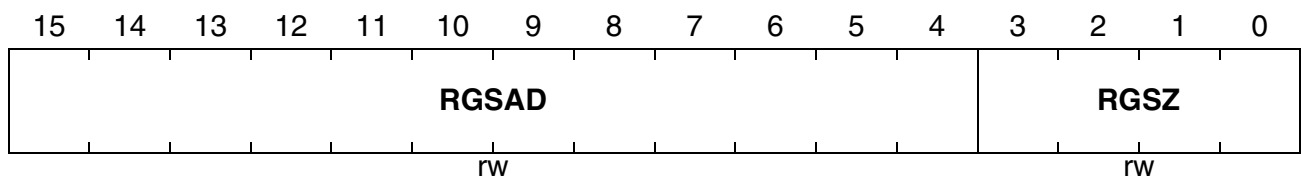
**ADDRSEL1**

**Address Select Register 1**      **SFR (FF18<sub>H</sub>/0C<sub>H</sub>)**      **Reset Value: 0000<sub>H</sub>**



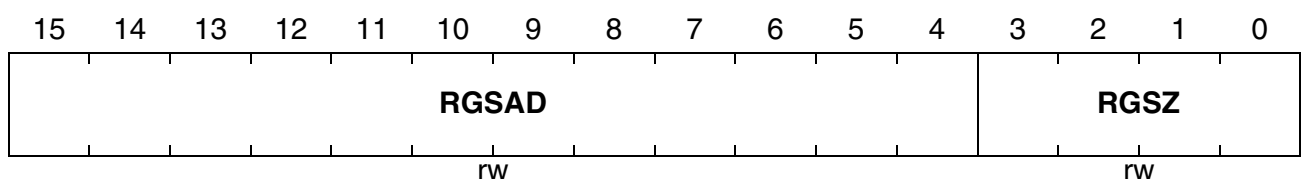
**ADDRSEL2**

**Address Select Register 2**      **SFR (FE1A<sub>H</sub>/0D<sub>H</sub>)**      **Reset Value: 0000<sub>H</sub>**



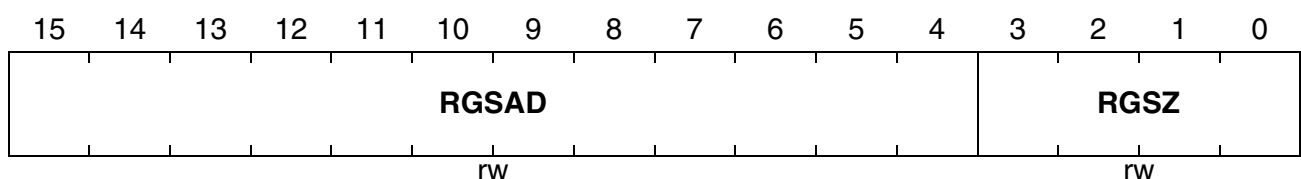
**ADDRSEL3**

**Address Select Register 3**      **SFR (FE1C<sub>H</sub>/0E<sub>H</sub>)**      **Reset Value: 0000<sub>H</sub>**



**ADDRSEL4**

**Address Select Register 4**      **SFR (FE1E<sub>H</sub>/0F<sub>H</sub>)**      **Reset Value: 0000<sub>H</sub>**



Bit	Function
<b>RGSZ</b>	<b>Range Size Selection</b> Defines the size of the address area controlled by the respective BUSCONx/ADDRSELx register pair. See <a href="#">Table 9-6</a> .
<b>RGSAD</b>	<b>Range Start Address</b> Defines the upper bits of the start address of the respective address area. See <a href="#">Table 9-6</a> .

**The External Bus Interface**

*Note: There is no register ADDRSEL0, as register BUSCON0 controls all external accesses outside the four address windows of BUSCON4 ... BUSCON1 within the complete address space.*

**Definition of Address Areas**

The four register pairs BUSCON4/ADDRSEL4 ... BUSCON1/ADDRSEL1 allow to define 4 separate address areas within the address space of the C161CS/JC/JI. Within each of these address areas external accesses can be controlled by one of the four different bus modes, independent of each other and of the bus mode specified in register BUSCON0. Each ADDRSELx register in a way cuts out an address window, within which the parameters in register BUSCONx are used to control external accesses. The range start address of such a window defines the upper address bits, which are not used within the address window of the specified size (see [Table 9-6](#)). For a given window size only those upper address bits of the start address are used (marked “R”), which are not implicitly used for addresses inside the window. The lower bits of the start address (marked “x”) are disregarded.

**Table 9-6 Address Window Definition**

Bit field RGSZ	Resulting Window Size	Relevant Bits (R) of Start Addr. (A12 ...)
0 0 0 0	4 KByte	R R R R R R R R R R R R R
0 0 0 1	8 KByte	R R R R R R R R R R R R R x
0 0 1 0	16 KByte	R R R R R R R R R R R R x x
0 0 1 1	32 KByte	R R R R R R R R R R x x x
0 1 0 0	64 KByte	R R R R R R R R x x x x
0 1 0 1	128 KByte	R R R R R R R x x x x x
0 1 1 0	256 KByte	R R R R R R x x x x x x
0 1 1 1	512 KByte	R R R R R x x x x x x
1 0 0 0	1 MByte	R R R R x x x x x x x
1 0 0 1	2 MByte	R R R x x x x x x x x
1 0 1 0	4 MByte	R R x x x x x x x x x
1 0 1 1	8 MByte	R x x x x x x x x x x
1 1 x x	Reserved.	

### Address Window Arbitration

The address windows that can be defined within the C161CS/JC/JI's address space may partly overlap each other. Thus e.g. small areas may be cut out of bigger windows in order to effectively utilize external resources, especially within segment 0.

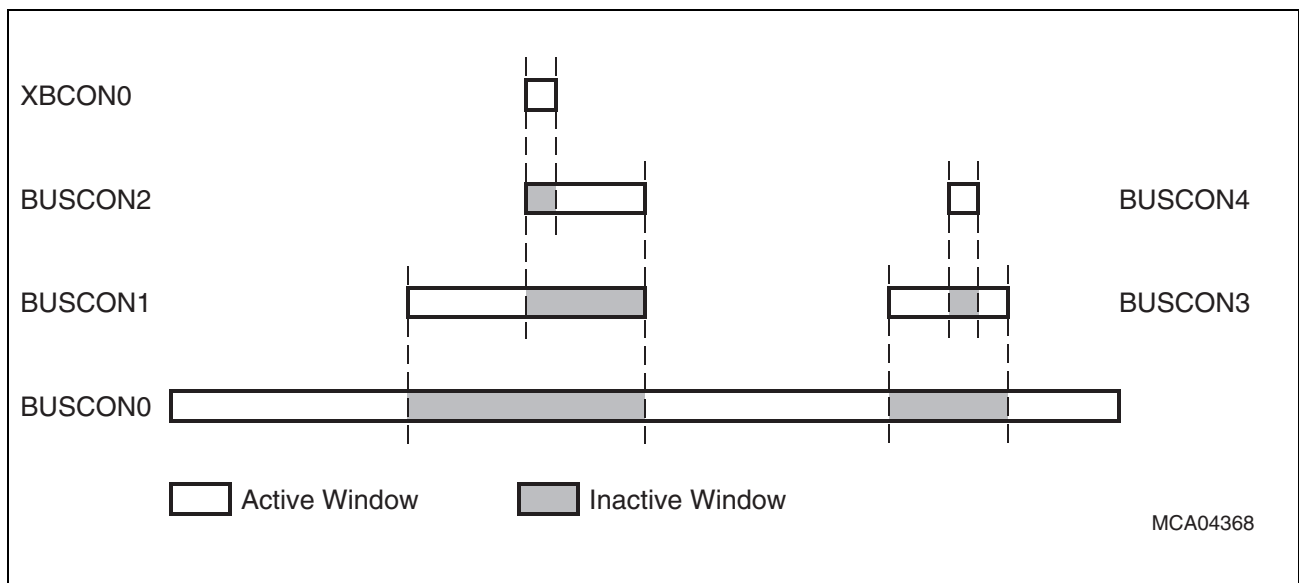
For each access the EBC compares the current address with all address select registers (programmable ADDRSELx and hardwired XADRSx). This comparison is done in four levels.

**Priority 1:** The hardwired XADRSx registers are evaluated first. A match with one of these registers directs the access to the respective X-Peripheral using the corresponding XBCONx register and ignoring all other ADDRSELx registers.

**Priority 2:** Registers ADDRSEL2 and ADDRSEL4 are evaluated before ADDRSEL1 and ADDRSEL3, respectively. A match with one of these registers directs the access to the respective external area using the corresponding BUSCONx register and ignoring registers ADDRSEL1/3 (see [Figure 9-11](#)).

**Priority 3:** A match with registers ADDRSEL1 or ADDRSEL3 directs the access to the respective external area using the corresponding BUSCONx register.

**Priority 4:** If there is no match with any XADRSx or ADDRSELx register the access to the external bus uses register BUSCON0.



**Figure 9-11 Address Window Arbitration**

*Note: Only the indicated overlaps are defined. All other overlaps lead to erroneous bus cycles. E.g. ADDRSEL4 may not overlap ADDRSEL2 or ADDRSEL1. The hardwired XADRSx registers are defined non-overlapping.*



The External Bus Interface

RP0H

Reset Value of P0H

SFR (F108<sub>H</sub>/84<sub>H</sub>)

Reset Value: - - XX<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								<b>CLKCFG</b>		<b>SALSEL</b>		<b>CSSEL</b>		<b>WRC</b>	
								rh		rh		rh		rh	

Bit	Function
<b>WRC</b>	<p><b>Write Configuration</b></p> <p>0: Pins <math>\overline{WR}</math> and <math>\overline{BHE}</math> operate as <math>\overline{WRL}</math> and <math>\overline{WRH}</math> signals</p> <p>1: Pins <math>\overline{WR}</math> and <math>\overline{BHE}</math> operate as <math>\overline{WR}</math> and <math>\overline{BHE}</math> signals</p>
<b>CSSEL</b>	<p><b>Chip Select Line Selection</b> (Number of active <math>\overline{CS}</math> outputs)</p> <p>00: 3 <math>\overline{CS}</math> lines: <math>\overline{CS2}</math> ... <math>\overline{CS0}</math></p> <p>01: 2 <math>\overline{CS}</math> lines: <math>\overline{CS1}</math> ... <math>\overline{CS0}</math></p> <p>10: No <math>\overline{CS}</math> lines at all</p> <p>11: 5 <math>\overline{CS}</math> lines: <math>\overline{CS4}</math> ... <math>\overline{CS0}</math> (Default without pulldowns)</p>
<b>SALSEL</b>	<p><b>Segment Address Line Selection</b> (nr. of active segment addr. outputs)</p> <p>00: 4-bit segment address: A19 ... A16</p> <p>01: No segment address lines at all</p> <p>10: 8-bit segment address: A23 ... A16</p> <p>11: 2-bit segment address: A17 ... A16 (Default without pulldowns)</p>
<b>CLKCFG</b>	<p><b>Clock Generation Mode Configuration</b></p> <p>These pins define the clock generation mode, i.e. the mechanism how the internal CPU clock is generated from the externally applied (XTAL) input clock.</p>

*Note: RP0H is initialized during the reset configuration and permits to check the current configuration.*

*This configuration (except for bit WRC) can be changed via register RSTCON (see [Section 22.5](#)).*

### Precautions and Hints

- The ext. bus interface is enabled as long as at least one of the BUSCON registers has its BUSACT bit set.
- PORT1 will output the intra-segment addr. as long as at least one of the BUSCON registers selects a demultiplexed external bus, even for multiplexed bus cycles.
- Not all addr. windows defined via registers ADDRSELx may overlap each other. The operation of the EBC will be unpredictable in such a case. See [“Address Window Arbitration” on Page 9-27](#).
- The addr. windows defined via registers ADDRSELx may overlap internal addr. areas. Internal accesses will be executed in this case.
- For any access to an internal addr. area the EBC will remain inactive (see [Chapter 9.6](#)).

## 9.6 EBC Idle State

When the external bus interface is enabled, but no external access is currently executed, the EBC is idle. As long as only internal resources (from an architecture point of view) like IRAM, GPRs or SFRs, etc. are used the external bus interface does not change (see [Table 9-7](#)).

Accesses to on-chip X-Peripherals are also controlled by the EBC. However, even though an X-Peripheral appears like an external peripheral to the controller, the respective accesses do not generate valid external bus cycles.

Due to timing constraints address and write data of an XBUS cycle are reflected on the external bus interface (see [Table 9-7](#)). The “address” mentioned above includes PORT1, Port 4,  $\overline{\text{BHE}}$  and ALE which also pulses for an XBUS cycle. The external  $\overline{\text{CS}}$  signals on Port 6 are driven inactive (high) because the EBC switches to an internal  $\overline{\text{XCS}}$  signal.

The **external control signals** ( $\overline{\text{RD}}$  and  $\overline{\text{WR}}$  or  $\overline{\text{WRL}}/\overline{\text{WRH}}$  if enabled) **remain inactive** (high).

**Table 9-7 Status of the External Bus Interface During EBC Idle State**

Pins	Internal Accesses only	XBUS Accesses
<b>PORT0</b>	Tristated (floating)	Tristated (floating) for read accesses XBUS write data for write accesses
<b>PORT1</b>	Last used external address (if used for the bus interface)	Last used XBUS address (if used for the bus interface) <sup>1)</sup>
<b>Port 4</b>	Last used external segment address (on selected pins)	Last used XBUS segment address (on selected pins)
<b>Port 6</b>	Active external $\overline{\text{CS}}$ signal corresponding to last used address	Inactive (high) for selected $\overline{\text{CS}}$ signals
<b><math>\overline{\text{BHE}}</math></b>	Level corresponding to last external access	Level corresponding to last XBUS access
<b>ALE</b>	Inactive (low)	Pulses as defined for X-Peripheral
<b><math>\overline{\text{RD}}</math></b>	Inactive (high)	<b>Inactive (high)<sup>1)</sup></b>
<b><math>\overline{\text{WR}}/\overline{\text{WRL}}</math></b>	Inactive (high)	<b>Inactive (high)<sup>1)</sup></b>
<b><math>\overline{\text{WRH}}</math></b>	Inactive (high)	<b>Inactive (high)<sup>1)</sup></b>

<sup>1)</sup> Used and driven in visible mode.

## 9.7 External Bus Arbitration

In embedded systems it may be efficient to share external resources like memory banks or peripheral devices among more than one microcontroller or processor. The C161CS/JC/JI supports this approach with the possibility to arbitrate the access to its external bus, i.e. to the external resources. Several bus masters can therefore use the same set of resources, resulting in compact, though powerful systems.

*Note: Sharing external resources is useful if these resources are used to a limited amount only. The performance of a bus master which relies on these external resources to a great extent (e.g. external code) will be reduced by the bandwidth used by the other masters.*

Bus arbitration uses three control signals ( $\overline{\text{HOLD}}$ ,  $\overline{\text{HLDA/BGR}}$ ,  $\overline{\text{BREQ}}$ ) and can be enabled and disabled via software (PSW.HLDEN), e.g. in order to protect time-critical code sections from being suspended by other bus masters. A bus arbiter logic can be designed to determine which of the bus masters controls the external system at a given time.

Using the specific master and slave modes for bus arbitration saves external glue logic (bus arbiter) when connecting two devices of the C166 Family.

*Note: Bus arbitration does not work if there is no clock signal for the EBC, i.e. during Idle mode and Powerdown mode.*

### Signals and Control

The upper three pins of Port 6 are used for the bus arbitration interface. [Table 9-8](#) summarizes the functions of these interface lines.

The external bus arbitration is enabled by setting bit HLDEN in register PSW to '1'. In this case the three bus arbitration pins  $\overline{\text{HOLD}}$ ,  $\overline{\text{HLDA}}$  and  $\overline{\text{BREQ}}$  are automatically controlled by the EBC independent of their IO configuration.

Bit HLDEN may be cleared during the execution of program sequences, where the external resources are required but cannot be shared with other bus masters, or during sequences which need to access on-chip XBUS resources but which shall not be interrupted by hold states. In this case the C161CS/JC/JI will not answer to  $\overline{\text{HOLD}}$  requests from other external masters. If HLDEN is cleared while the C161CS/JC/JI is in Hold State (code execution from internal RAM/ROM) this Hold State is left only after  $\overline{\text{HOLD}}$  has been deactivated again. I.e. in this case the current Hold State continues and only the next  $\overline{\text{HOLD}}$  request is not answered.

*Note: The pins  $\overline{\text{HOLD}}$ ,  $\overline{\text{HLDA}}$  and  $\overline{\text{BREQ}}$  keep their alternate function (bus arbitration) even after the arbitration mechanism has been switched off by clearing HLDEN. All three pins are used for bus arbitration after bit HLDEN was set once.*

**Table 9-8 Interface Pins for Bus Arbitration**

Pin	Function	Direction	Operational Description
P6.5	$\overline{\text{HOLD}}$	INput	The hold request signal requests the external bus system from the C161CS/JC/JI.
P6.6	$\overline{\text{HLDA}}$ (Master mode)	OUTput	The hold acknowledge signal acknowledges a hold request and indicates to the external partners that the C161CS/JC/JI has withdrawn from the bus and another external bus master may now use it.
	$\overline{\text{BGR}}^1)$ (Slave mode)	INput	The bus grant signal indicates to the C161CS/JC/JI (slave) that the master has <u>withdrawn</u> from the external bus in response to the slave's $\overline{\text{BREQ}}$ . The slave may now use the external bus system until the master requests it back.
P6.7	$\overline{\text{BREQ}}$	OUTput	The bus request signal indicates to the bus arbiter logic that the C161CS/JC/JI requires control over the external bus, which has been released and is currently controlled by another bus master.

<sup>1)</sup> In slave mode pin  $\overline{\text{HLDA}}$  inverts its direction to input. The changed functionality is indicated through the different name.

### Arbitration Sequences

An external master may request the C161CS/JC/JI's bus via the  $\overline{\text{HOLD}}$  input. After completing the currently running bus cycle the C161CS/JC/JI acknowledges this request via the  $\overline{\text{HLDA}}$  output and will then float its bus lines (internal pullups at  $\overline{\text{CSx}}$ ,  $\overline{\text{RD}}$ , and  $\overline{\text{WR}}$ , internal pulldown at ALE). The new master may now access the peripheral devices or memory banks via the same interface lines as the C161CS/JC/JI. During this time the C161CS/JC/JI can keep on executing, as long as it does not need access to the external bus. All actions that just require internal resources like instruction or data memory and on-chip generic peripherals, may be executed in parallel.

*Note: The XBUS is an internal representation of the external bus interface. Accesses to XBUS peripherals use the EBC and therefore also cannot be executed during the bus hold state.*

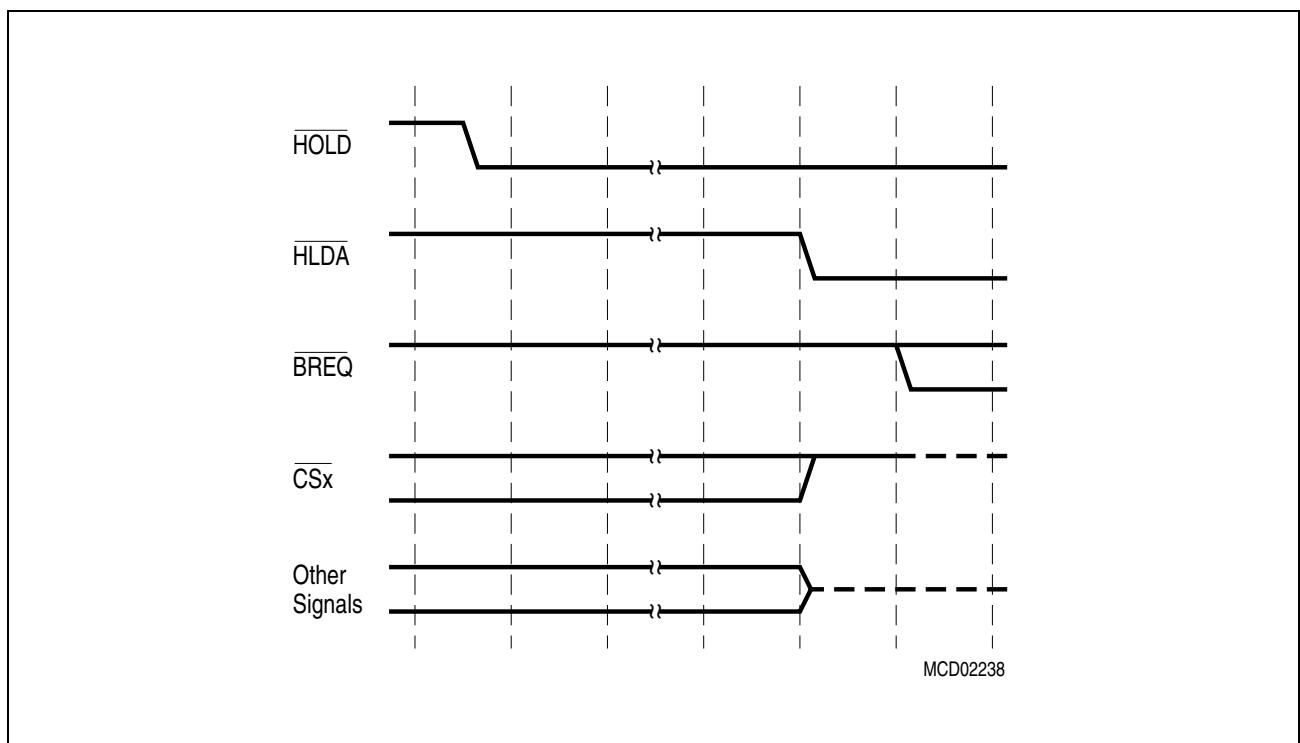
When the C161CS/JC/JI needs access to its external bus while it is occupied by another bus master, it demands it via the  $\overline{\text{BREQ}}$  output. The arbiter can then remove the current master from the bus. This is indicated to the C161CS/JC/JI by deactivating its  $\overline{\text{HOLD}}$  input. The C161CS/JC/JI responds by deactivating its  $\overline{\text{HLDA}}$  signal, and then taking control of the external bus itself. Of course also the request signal  $\overline{\text{BREQ}}$  is deactivated after getting control of the bus back.

### Entering the Hold State

Access to the C161CS/JC/JI's external bus is requested by driving its  $\overline{\text{HOLD}}$  input low. After synchronizing this signal the C161CS/JC/JI will complete a current external bus cycle or XBUS cycle (if any is active), release the external bus and grant access to it by driving the  $\overline{\text{HLDA}}$  output low. During hold state the C161CS/JC/JI treats the external bus interface as follows:

- Address and data bus(es) float to tri-state
- ALE is pulled low by an internal pulldown device
- Command lines are pulled high by internal pullup devices ( $\overline{\text{RD}}$ ,  $\overline{\text{WR}}$ )
- $\overline{\text{CSx}}$  outputs are driven high for 1 TCL and then pulled high (in push/pull mode), or float to tri-state (in open drain mode)

Should the C161CS/JC/JI require access to its external bus or XBUS during hold mode, it activates its bus request output  $\overline{\text{BREQ}}$  to notify the arbitration circuitry.  $\overline{\text{BREQ}}$  is activated only during hold mode. It will be inactive during normal operation.



**Figure 9-12 External Bus Arbitration, Releasing the Bus**

*Note: The C161CS/JC/JI will complete the currently running bus cycle before granting bus access as indicated by the broken lines. This may delay hold acknowledge compared to this figure.*

**Figure 9-12** shows the first possibility for  $\overline{\text{BREQ}}$  to get active.

During bus hold pin  $\overline{\text{BHE/WRH}}$  is floating. An external pullup should be used if this is required.

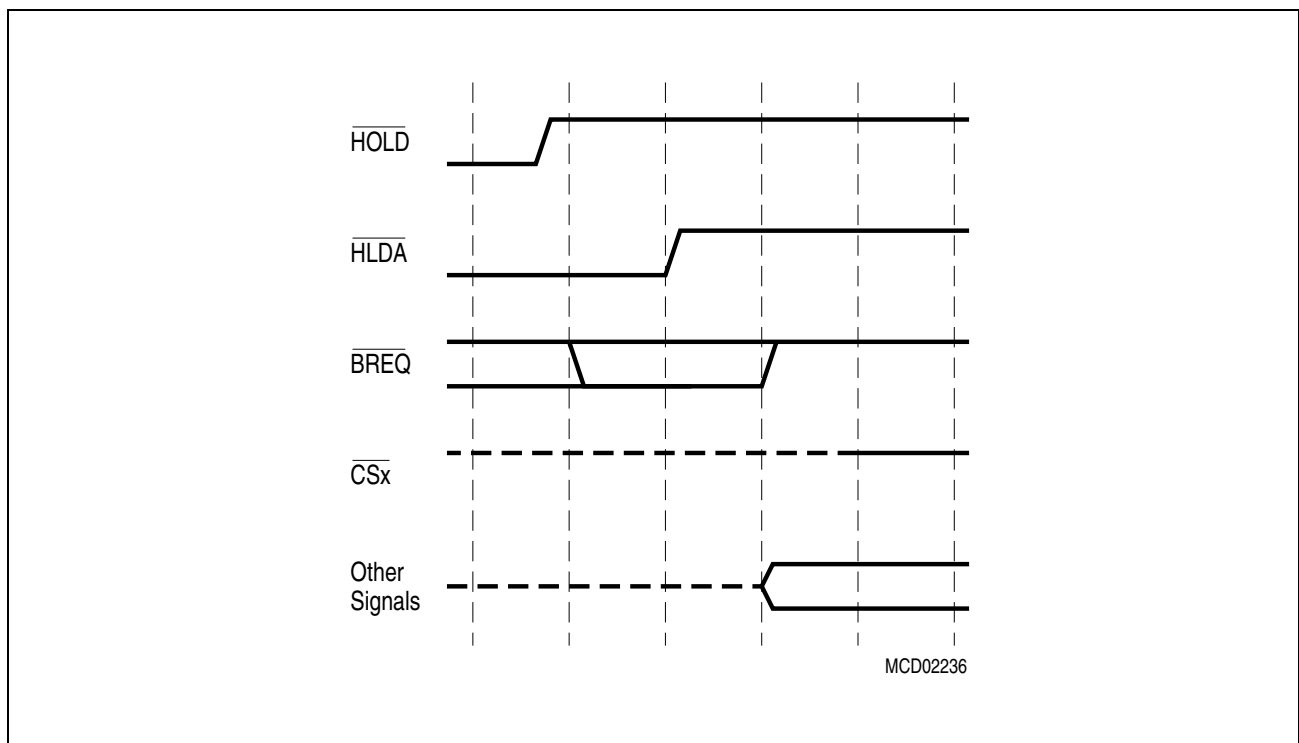
### Exiting the Hold State

The external bus master returns the access rights to the C161CS/JC/JI by driving the  $\overline{\text{HOLD}}$  input high. After synchronizing this signal the C161CS/JC/JI will drive the  $\overline{\text{HLDA}}$  output high, actively drive the control signals and resume executing external bus cycles if required.

Depending on the arbitration logic, the external bus can be returned to the C161CS/JC/JI under two circumstances:

- The external master does no more require access to the shared resources and gives up its own access rights, or
- The C161CS/JC/JI needs access to the shared resources and demands this by activating its  $\overline{\text{BREQ}}$  output. The arbitration logic may then activate the other master's  $\overline{\text{HOLD}}$  and so free the external bus for the C161CS/JC/JI, depending on the priority of the different masters.

*Note: The Hold State is not terminated by clearing bit  $\text{HLDEN}$ .*



**Figure 9-13 External Bus Arbitration, Regaining the Bus**

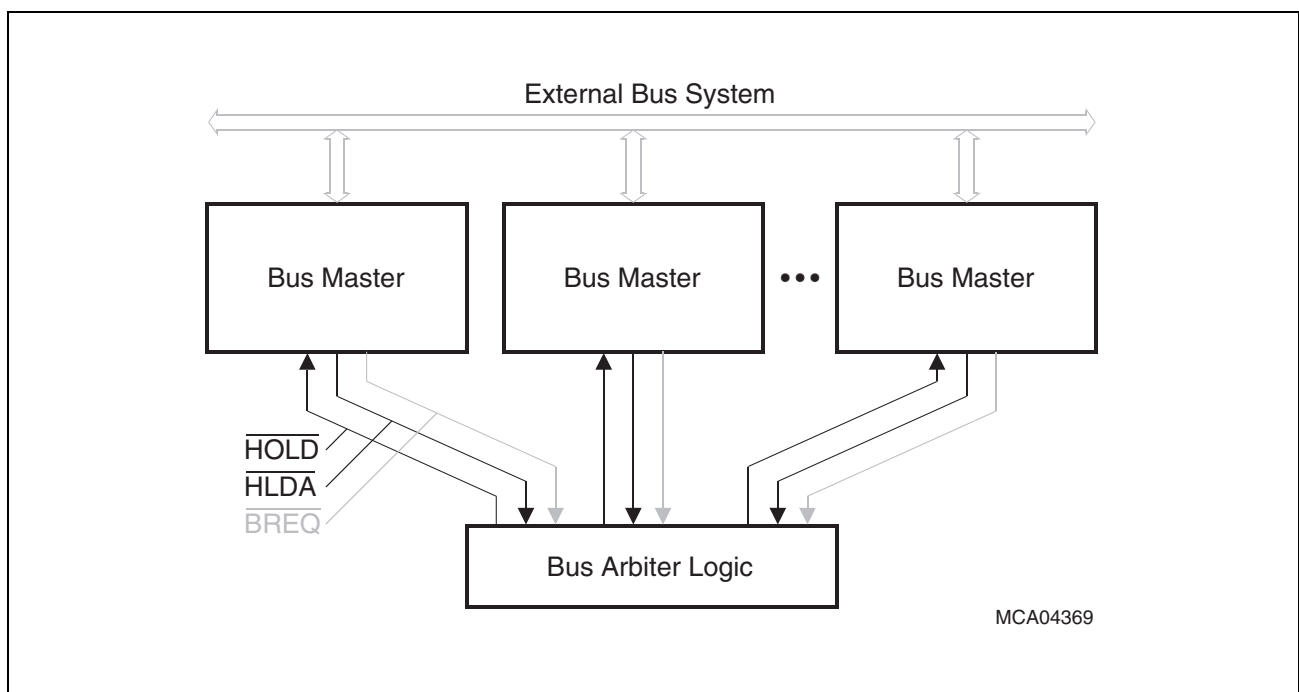
*Note: The falling  $\overline{\text{BREQ}}$  edge shows the last chance for  $\overline{\text{BREQ}}$  to trigger the indicated regain-sequence. Even if  $\overline{\text{BREQ}}$  is activated earlier the regain-sequence is initiated by  $\overline{\text{HOLD}}$  going high.  $\overline{\text{BREQ}}$  and  $\overline{\text{HOLD}}$  are connected via an external arbitration circuitry. Please note that  $\overline{\text{HOLD}}$  may also be deactivated without the C161CS/JC/JI requesting the bus.*

### Connecting Bus Masters

When multiple bus masters (C161CS/JC/JIs or other masters) shall share external resources a bus arbiter is required that determines the currently active bus master and also enables a C161CS/JC/JI which has surrendered its bus interface to regain control of it in case it must access the shared external resources.

The structure of this bus arbiter defines the degree of control the C161CS/JC/JI has over the external bus system. Whenever the C161CS/JC/JI has released the bus, there is no way of actively regaining control of it, besides the activation of the  $\overline{\text{BREQ}}$  signal. In any case the C161CS/JC/JI must wait for the deactivation of its  $\overline{\text{HOLD}}$  input.

*Note: The full arbitration logic is required if the “other” bus master does not automatically remove its hold request after having used the shared resources.*



**Figure 9-14 Principle Arbitration Logic**

### Compact Two-Master Systems (Master/Slave Mode)

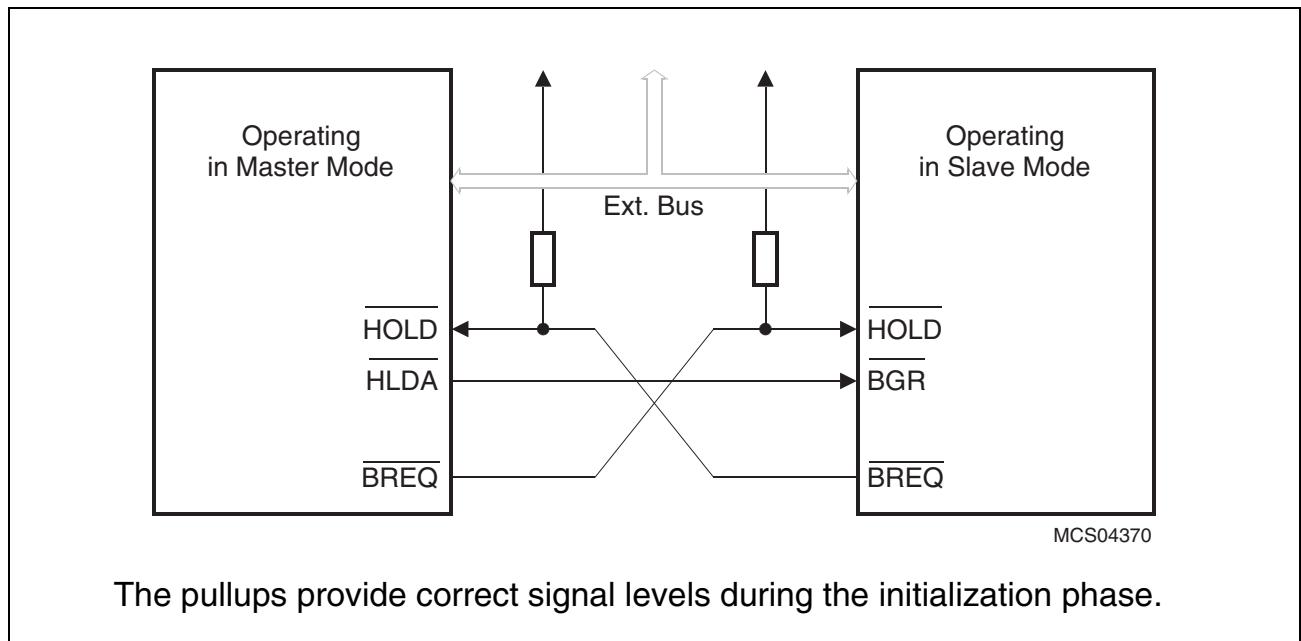
When two C161CS/JC/JIs (or other members of the C166 Family) are to be connected in this way the external bus arbitration logic (normally required to combine the respective output signals  $\overline{\text{HLDA}}$  and  $\overline{\text{BREQ}}$ ) can be left out.

In this case one of the controllers operates in Master Mode (the standard default operating mode, DP6.7 = '0'), while the other one must operate in Slave Mode (selected with DP6.7 = '1'). In this configuration the master-device normally controls the external bus, while the slave-device gets control of it on demand only. In most cases this requires that (at least) the slave-device operates out of internal resources most of the time, in order to get an acceptable overall system performance.



The External Bus Interface

**In Slave Mode** the C161CS/JC/JI inverts the direction of its  $\overline{\text{HLDA}}$  pin and uses it as the bus grant input  $\overline{\text{BGR}}$ , while the master's  $\overline{\text{HLDA}}$  pin remains an output. This permits the direct connection of these two signals without any additional glue logic for bus arbitration. The  $\overline{\text{BREQ}}$  outputs are mutually connected to the other partner's  $\overline{\text{HOLD}}$  input (see **Figure 9-15**).



**Figure 9-15 Sharing External Resources Using Slave Mode**

**Slave Mode is selected** by intentionally switching pin  $\overline{\text{BREQ}}$  to output (DP6.7 = '1'). Normally the port direction register bits for the arbitration interface pins retain their reset value which is '0'. Clearing bit DP6.7 (or preserving the reset value) selects Master Mode, where the device operates compatible with earlier versions without the slave mode feature.

*Note: If the C161CS/JC/JI operates in slave mode and executes a loop out of external memory which fits completely into the jump cache (e.g. JB bitaddr, \$) its  $\overline{\text{BREQ}}$  output may toggle (period = 2 CPU clock cycles).  $\overline{\text{BREQ}}$  is activated by the prefetcher that wants to read the next sequential instruction.  $\overline{\text{BREQ}}$  is deactivated, because the target of the taken jump is found in the jump cache. A loop of a minimum length of 3 words avoids this.*

## **9.8 The XBUS Interface**

The C161CS/JC/JI provides an on-chip interface (the XBUS interface), via which integrated customer/application specific peripherals can be connected to the standard controller core. The XBUS is an internal representation of the external bus interface, i.e. it is operated in the same way.

For each peripheral on the XBUS (X-Peripheral) there is a separate address window controlled by a register pair XBCONx/XADRSx (similar to registers BUSCONx and ADDRSELx). As an interface to a peripheral in many cases is represented by just a few registers, the XADRSx registers partly select smaller address windows than the standard ADDRSEL registers. As the XBCONx/XADRSx register pairs control integrated peripherals rather than externally connected ones, they are fixed by mask programming rather than being user programmable.

X-Peripheral accesses provide the same choices as external accesses, so these peripherals may be byte-wide or word-wide. Because the on-chip connection can be realized very efficiently and for performance reasons X-Peripherals are only implemented with a separate address bus, i.e. in demultiplexed bus mode. Interrupt nodes are provided for X-Peripherals to be integrated.

*Note: If you plan to develop a peripheral of your own to be integrated into a C161CS/JC/JI device to create a customer specific version, please ask for the specification of the XBUS interface and for further support.*

## 9.8.1 Accessing the On-chip XBUS Peripherals

### Enabling of XBUS Peripherals

After reset all on-chip XBUS peripherals are disabled. In order to be usable an XBUS peripheral must be enabled via the global enable bit XPEN in register SYSCON.

**Table 9-9** summarizes the XBUS peripherals and also the number of waitstates which are used when accessing the respective peripheral.

**Table 9-9 XBUS Peripherals in the C161CS/JC/JI**

Associated XBUS Peripheral	Waitstates
CAN1	2
CAN2	2
SDLM	2
ASC1	1
IIC	1
XRAM 8 KByte	0

### Visible Mode

The C161CS/JC/JI can mirror on-chip access cycles to its XBUS peripherals so these accesses can be observed or recorded by the external system. This function is enabled via bit VISIBLE in register SYSCON.

Accesses to XBUS peripherals also use the EBC. Due to timing constraints the address bus will change for all accesses using the EBC.

*Note: As XBUS peripherals use demultiplexed bus cycles, the respective address is driven on PORT1 in visible mode, even if the external system uses MUX buses only.*

**If visible mode is activated**, accesses to on-chip XBUS peripherals (including control signals  $\overline{RD}$ ,  $\overline{WR}$ , and  $\overline{BHE}$ ) are mirrored to the bus interface. Accesses to internal resources (program memory, IRAM, GPRs) do not use the EBC and cannot be mirrored to outside.

**If visible mode is deactivated**, however, no control signals ( $\overline{RD}$ ,  $\overline{WR}$ ) will be activated, i.e. there will be no valid external bus cycles.

*Note: Visible mode can only work if the external bus is enabled at all.*

### **9.8.2 External Accesses to XBUS Peripherals**

The on-chip XBUS peripherals of the C161CS/JC/JI can be accessed from outside via the external bus interface under certain circumstances. In emulation mode the XBUS peripherals are controlled by the bondout-chip. During normal operation this external access is accomplished selecting the XPER-Share mode.

#### **XPER-Share Mode**

The C161CS/JC/JI can share its on-chip XBUS peripherals with other (external) bus masters, i.e. it can allow them to access its X-Peripherals while it is in hold mode. This external access is enabled via bit XPERSHARE in register SYSCON and is only possible while the host controller is in hold mode.

During XPER-Share mode the C161CS/JC/JI's bus interface inverts its direction so the external master can drive address, control, and data signals to the respective peripheral. This can be used e.g. to install a mailbox memory in a multi-processor system.

*Note: When XPER-Share mode is disabled no accesses to on-chip XBUS peripherals can be executed from outside.*

## 10 The General Purpose Timer Units

The General Purpose Timer Units GPT1 and GPT2 represent very flexible multifunctional timer structures which may be used for timing, event counting, pulse width measurement, pulse generation, frequency multiplication, and other purposes. They incorporate five 16-bit timers that are grouped into the two timer blocks GPT1 and GPT2.

Block GPT1 contains 3 timers/counters with a maximum resolution of 16 TCL, while block GPT2 contains 2 timers/counters with a maximum resolution of 8 TCL and a 16-bit Capture/Reload register (CAPREL). Each timer in each block may operate independently in a number of different modes such as gated timer or counter mode, or may be concatenated with another timer of the same block. The auxiliary timers of GPT1 may optionally be configured as reload or capture registers for the core timer. In the GPT2 block, the additional CAPREL register supports capture and reload operation with extended functionality, and its core timer T6 may be concatenated with timers of the CAPCOM units (T0, T1, T7, and T8). Each block has alternate input/output functions and specific interrupts associated with it.

### 10.1 Timer Block GPT1

From a programmer's point of view, the GPT1 block is composed of a set of SFRs as summarized below. Those portions of port and direction registers which are used for alternate functions by the GPT1 block are shaded.

Ports & Direction Control Alternate Functions		Data Registers	Control Registers	Interrupt Control
ODP3	E	T2	T2CON	T2IC
DP3		T3	T3CON	T3IC
P3		T4	T4CON	T4IC
P5			P5DIDIS	
		T2IN/P3.7 T2EUD/P5.15 T3IN/P3.6 T3EUD/P3.4 T4IN/P3.5 T4EUD/P5.14 T3OUT/P3.3		
P5	Port 5 Data Register	P5DIDIS	Port 5 Digital Input Disable Register	
ODP3	Port 3 Open Drain Control Register	T2	GPT1 Timer 2 Register	
DP3	Port 3 Direction Control Register	T3	GPT1 Timer 3 Register	
P3	Port 3 Data Register	T4	GPT1 Timer 4 Register	
T2CON	GPT1 Timer 2 Control Register	T2IC	GPT1 Timer 2 Interrupt Control Register	
T3CON	GPT1 Timer 3 Control Register	T3IC	GPT1 Timer 3 Interrupt Control Register	
T4CON	GPT1 Timer 4 Control Register	T4IC	GPT1 Timer 4 Interrupt Control Register	

MCA04371

**Figure 10-1 SFRs and Port Pins Associated with Timer Block GPT1**

### The General Purpose Timer Units

All three timers of block GPT1 (T2, T3, T4) can run in 4 basic modes, which are timer, gated timer, counter and incremental interface mode, and all timers can either count up or down. Each timer has an alternate input function pin (TxIN) associated with it which serves as the gate control in gated timer mode, or as the count input in counter mode. The count direction (Up/Down) may be programmed via software or may be dynamically altered by a signal at an external control input pin. Each overflow/underflow of core timer T3 is latched in the toggle FlipFlop T3OTL and may be indicated on an alternate output function pin. The auxiliary timers T2 and T4 may additionally be concatenated with the core timer, or used as capture or reload registers for the core timer.

The current contents of each timer can be read or modified by the CPU by accessing the corresponding timer registers T2, T3, or T4, which are located in the non-bitaddressable SFR space. When any of the timer registers is written to by the CPU in the state immediately before a timer increment, decrement, reload, or capture is to be performed, the CPU write operation has priority in order to guarantee correct results.

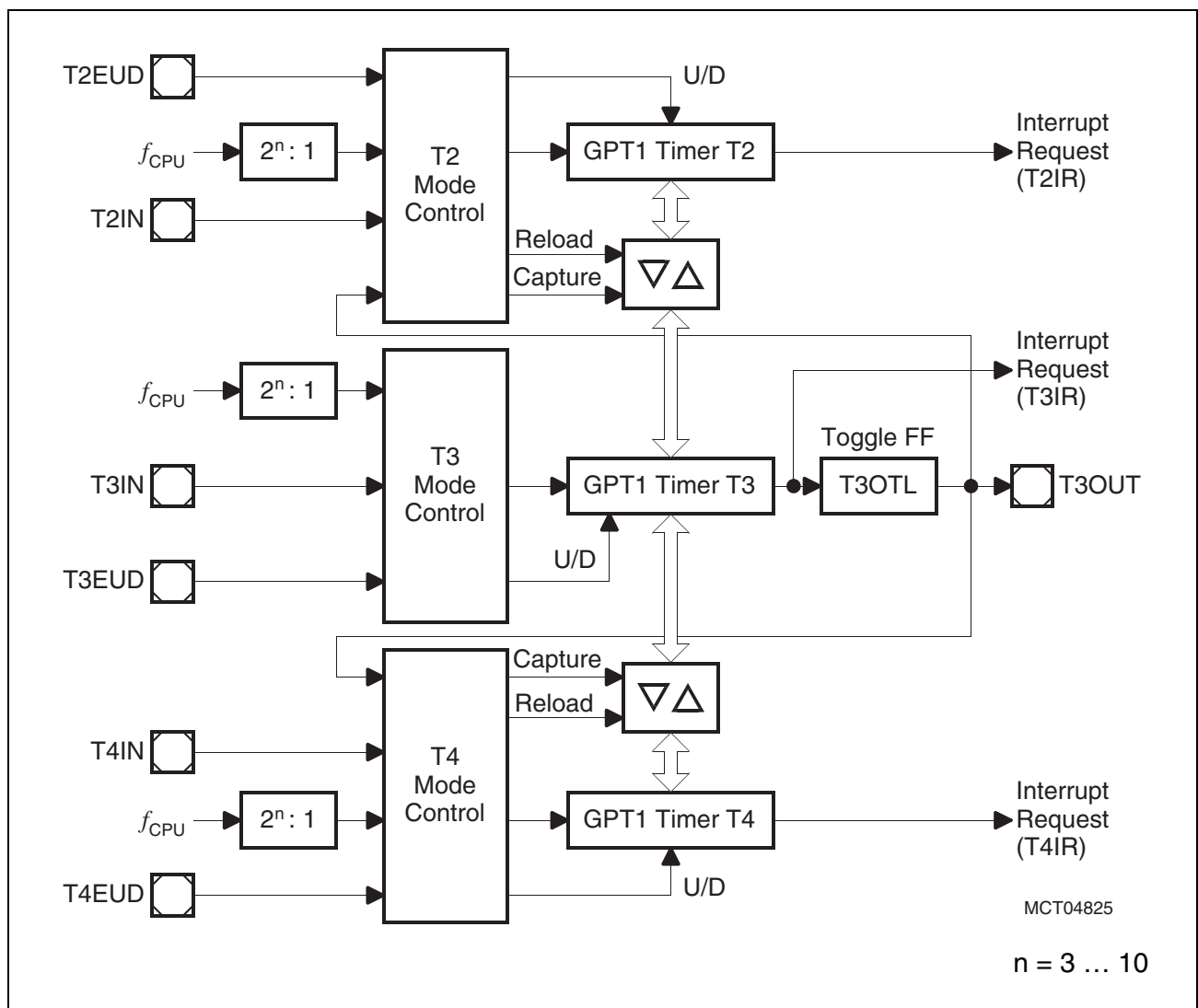


Figure 10-2 GPT1 Block Diagram

The General Purpose Timer Units

10.1.1 GPT1 Core Timer T3

The core timer T3 is configured and controlled via its bitaddressable control register T3CON.

**T3CON**

Timer 3 Control Register

SFR (FF42<sub>H</sub>/A1<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	T3 OTL	T3 OE	T3 UDE	T3 UD	T3R	T3M			T3I		
-	-	-	-	-	rwh	rw	rw	rw	rw	rw			rw		

Bit	Function
<b>T3I</b>	<b>Timer 3 Input Selection</b> Depends on the operating mode, see respective sections.
<b>T3M</b>	<b>Timer 3 Mode Control</b> (Basic Operating Mode) 000: Timer Mode 001: Counter Mode 010: Gated Timer with Gate active low 011: Gated Timer with Gate active high 100: <i>Reserved</i> . Do not use this combination. 101: <i>Reserved</i> . Do not use this combination. 110: Incremental Interface Mode 111: <i>Reserved</i> . Do not use this combination.
<b>T3R</b>	<b>Timer 3 Run Bit</b> 0: Timer/Counter 3 stops 1: Timer/Counter 3 runs
<b>T3UD</b>	<b>Timer 3 Up/Down Control<sup>1)</sup></b>
<b>T3UDE</b>	<b>Timer 3 External Up/Down Enable<sup>1)</sup></b>
<b>T3OE</b>	<b>Alternate Output Function Enable</b> 0: Alternate Output Function Disabled 1: Alternate Output Function Enabled
<b>T3OTL</b>	<b>Timer 3 Output Toggle Latch</b> Toggles on each overflow/underflow of T3. Can be set or reset by software.

<sup>1)</sup> For the effects of bits T3UD and T3UDE refer to the direction [Table 10-1](#).

**The General Purpose Timer Units**

**Timer 3 Run Bit**

The timer can be started or stopped by software through bit T3R (Timer T3 Run Bit). If T3R = '0', the timer stops. Setting T3R to '1' will start the timer. In gated timer mode, the timer will only run if T3R = '1' and the gate is active (high or low, as programmed).

**Count Direction Control**

The count direction of the core timer can be controlled either by software or by the external input pin T3EUD (Timer T3 External Up/Down Control Input), which is the alternate input function of port pin P3.4. These options are selected by bits T3UD and T3UDE in control register T3CON. When the up/down control is done by software (bit T3UDE = '0'), the count direction can be altered by setting or clearing bit T3UD. When T3UDE = '1', pin T3EUD is selected to be the controlling source of the count direction. However, bit T3UD can still be used to reverse the actual count direction, as shown in **Table 10-1**. If T3UD = '0' and pin T3EUD shows a low level, the timer is counting up. With a high level at T3EUD the timer is counting down. If T3UD = '1', a high level at pin T3EUD specifies counting up, and a low level specifies counting down. The count direction can be changed regardless of whether the timer is running or not.

When pin T3EUD/P3.4 is used as external count direction control input, it must be configured as input, i.e. its corresponding direction control bit DP3.4 must be set to '0'.

**Table 10-1 GPT1 Core Timer T3 Count Direction Control**

Pin TxEUD	Bit TxUDE	Bit TxUD	Count Direction
X	0	0	Count Up
X	0	1	Count Down
0	1	0	Count Up
1	1	0	Count Down
0	1	1	Count Down
1	1	1	Count Up

*Note: The direction control works the same for core timer T3 and for auxiliary timers T2 and T4. Therefore the pins and bits are named Tx ...*



### Timer 3 Output Toggle Latch

An overflow or underflow of timer T3 will clock the toggle bit T3OTL in control register T3CON. T3OTL can also be set or reset by software. Bit T3OE (Alternate Output Function Enable) in register T3CON enables the state of T3OTL to be an alternate function of the external output pin T3OUT. For that purpose, a '1' must be written into the respective port data latch and pin T3OUT must be configured as output by setting the corresponding direction control bit to '1'. If T3OE = '1', pin T3OUT then outputs the state of T3OTL. If T3OE = '0', pin T3OUT can be used as general purpose IO pin.

In addition, T3OTL can be used in conjunction with the timer over/underflows as an input for the counter function or as a trigger source for the reload function of the auxiliary timers T2 and T4. For this purpose, the state of T3OTL does not have to be available at pin T3OUT, because an internal connection is provided for this option.

### Timer 3 in Timer Mode

Timer mode for the core timer T3 is selected by setting bit field T3M in register T3CON to '000<sub>B</sub>'. In this mode, T3 is clocked with the internal system clock (CPU clock) divided by a programmable prescaler, which is selected by bit field T3I. The input frequency  $f_{T3}$  for timer T3 and its resolution  $r_{T3}$  are scaled linearly with lower clock frequencies  $f_{CPU}$ , as can be seen from the following formula:

$$f_{T3} = \frac{f_{CPU}}{8 \times 2^{\langle T3I \rangle}} \quad , \quad r_{T3} [\mu s] = \frac{8 \times 2^{\langle T3I \rangle}}{f_{CPU} [MHz]}$$

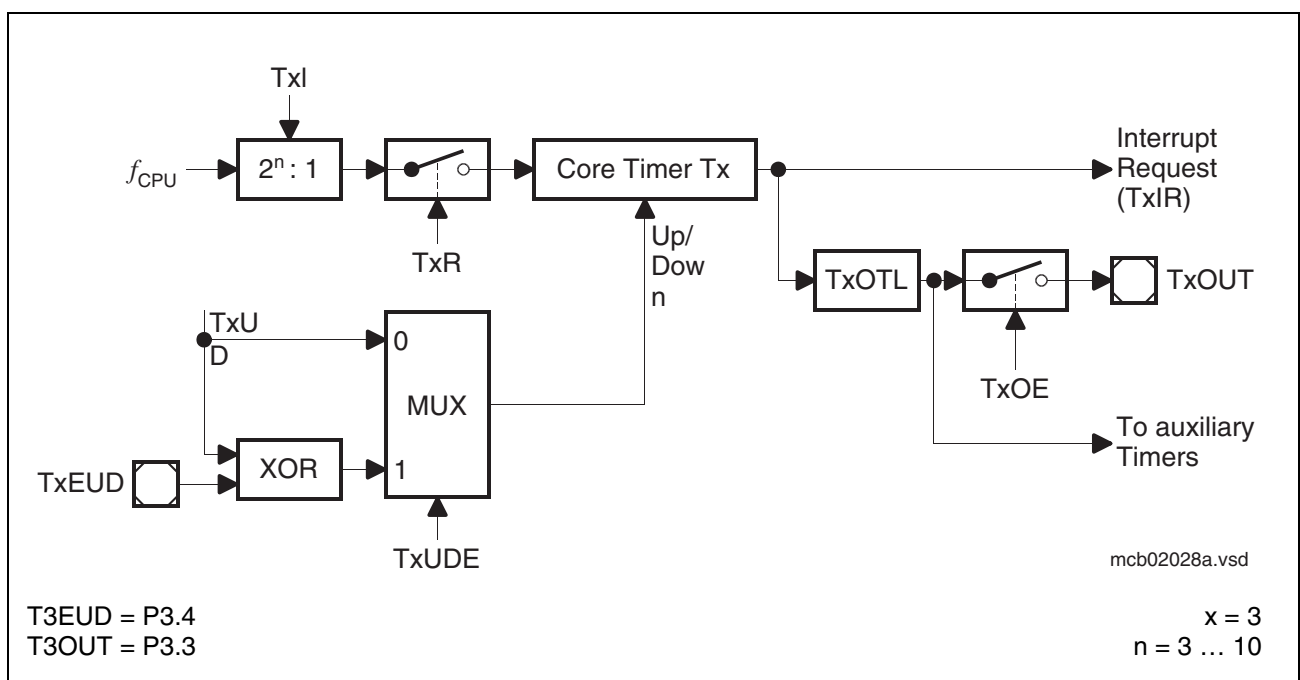


Figure 10-3 Block Diagram of Core Timer T3 in Timer Mode

**The General Purpose Timer Units**

The timer input frequencies, resolution and periods which result from the selected prescaler option are listed in **Table 10-2**. This table also applies to the Gated Timer Mode of T3 and to the auxiliary timers T2 and T4 in timer and gated timer mode. Note that some numbers may be rounded to 3 significant digits.

**Table 10-2 GPT1 Timer Input Frequencies, Resolution and Periods @ 20 MHz**

$f_{\text{CPU}} = 20 \text{ MHz}$	Timer Input Selection T2I/T3I/T4I							
	000 <sub>B</sub>	001 <sub>B</sub>	010 <sub>B</sub>	011 <sub>B</sub>	100 <sub>B</sub>	101 <sub>B</sub>	110 <sub>B</sub>	111 <sub>B</sub>
<b>Prescaler factor</b>	8	16	32	64	128	256	512	1024
<b>Input Frequency</b>	2.5 MHz	1.25 MHz	625 kHz	312.5 kHz	156.25 kHz	78.125 kHz	39.06 kHz	19.53 kHz
<b>Resolution</b>	400 ns	800 ns	1.6 $\mu\text{s}$	3.2 $\mu\text{s}$	6.4 $\mu\text{s}$	12.8 $\mu\text{s}$	25.6 $\mu\text{s}$	51.2 $\mu\text{s}$
<b>Period</b>	26.2 ms	52.5 ms	105 ms	210 ms	420 ms	840 ms	1.68 s	3.36 s

**Table 10-3 GPT1 Timer Input Frequencies, Resolution and Periods @ 25 MHz**

$f_{\text{CPU}} = 25 \text{ MHz}$	Timer Input Selection T2I/T3I/T4I							
	000 <sub>B</sub>	001 <sub>B</sub>	010 <sub>B</sub>	011 <sub>B</sub>	100 <sub>B</sub>	101 <sub>B</sub>	110 <sub>B</sub>	111 <sub>B</sub>
<b>Prescaler factor</b>	8	16	32	64	128	256	512	1024
<b>Input Frequency</b>	3.125 MHz	1.56 MHz	781.25 kHz	390.62 kHz	195.3 kHz	97.65 kHz	48.83 kHz	24.42 kHz
<b>Resolution</b>	320 ns	640 ns	1.28 $\mu\text{s}$	2.56 $\mu\text{s}$	5.12 $\mu\text{s}$	10.2 $\mu\text{s}$	20.5 $\mu\text{s}$	41.0 $\mu\text{s}$
<b>Period</b>	21.0 ms	41.9 ms	83.9 ms	168 ms	336 ms	671 ms	1.34 s	2.68 s

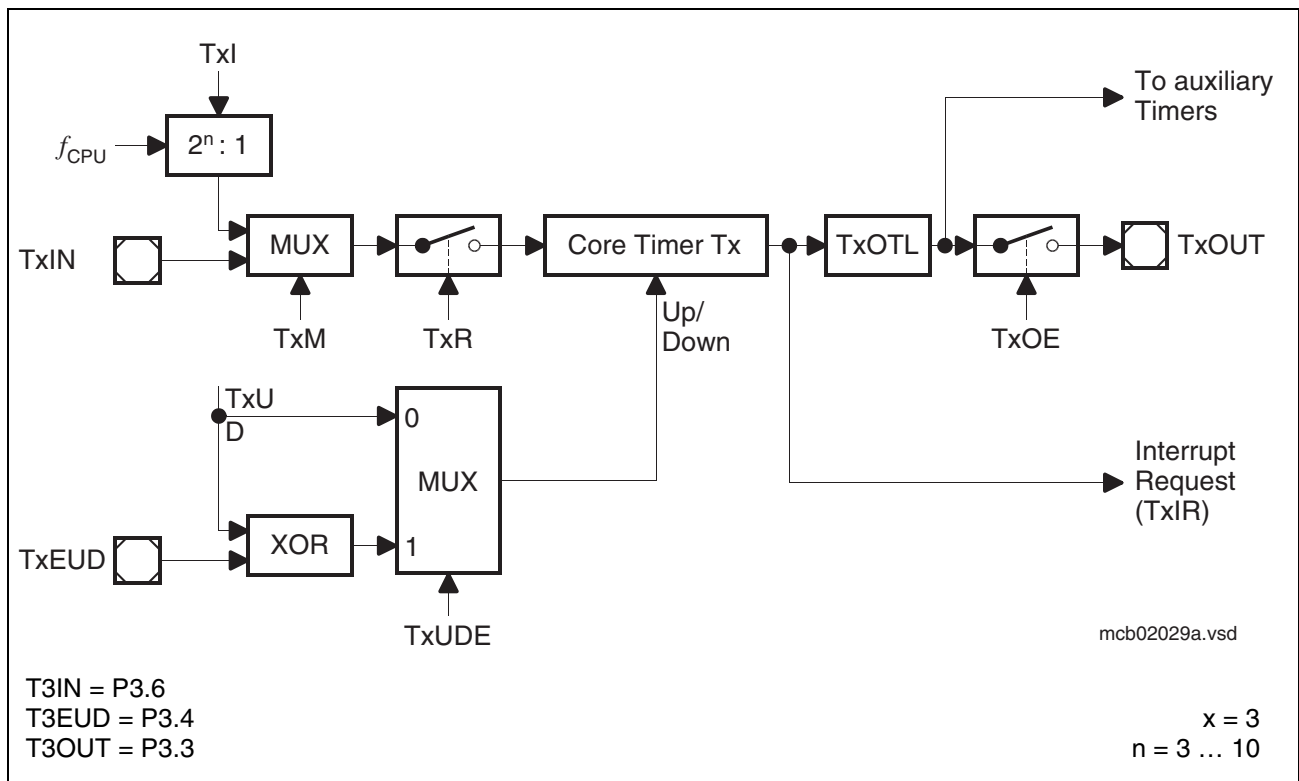
**Table 10-4 GPT1 Timer Input Frequencies, Resolution and Periods @ 33 MHz**

$f_{\text{CPU}} = 33 \text{ MHz}$	Timer Input Selection T2I/T3I/T4I							
	000 <sub>B</sub>	001 <sub>B</sub>	010 <sub>B</sub>	011 <sub>B</sub>	100 <sub>B</sub>	101 <sub>B</sub>	110 <sub>B</sub>	111 <sub>B</sub>
<b>Prescaler factor</b>	8	16	32	64	128	256	512	1024
<b>Input Frequency</b>	4.125 MHz	2.0625 MHz	1.031 MHz	515.62 kHz	257.81 kHz	128.91 kHz	64.45 kHz	32.23 kHz
<b>Resolution</b>	242 ns	485 ns	970 ns	1.94 $\mu\text{s}$	3.88 $\mu\text{s}$	7.76 $\mu\text{s}$	15.5 $\mu\text{s}$	31.0 $\mu\text{s}$
<b>Period</b>	15.9 ms	31.8 ms	63.6 ms	127 ms	254 ms	508 ms	1.02 s	2.03 s

### Timer 3 in Gated Timer Mode

Gated timer mode for the core timer T3 is selected by setting bit field T3M in register T3CON to '010<sub>B</sub>' or '011<sub>B</sub>'. Bit T3M.0 (T3CON.3) selects the active level of the gate input. In gated timer mode the same options for the input frequency as for the timer mode are available. However, the input clock to the timer in this mode is gated by the external input pin T3IN (Timer T3 External Input).

To enable this operation pin T3IN must be configured as input, i.e. the corresponding direction control bit must contain '0'.



**Figure 10-4 Block Diagram of Core Timer T3 in Gated Timer Mode**

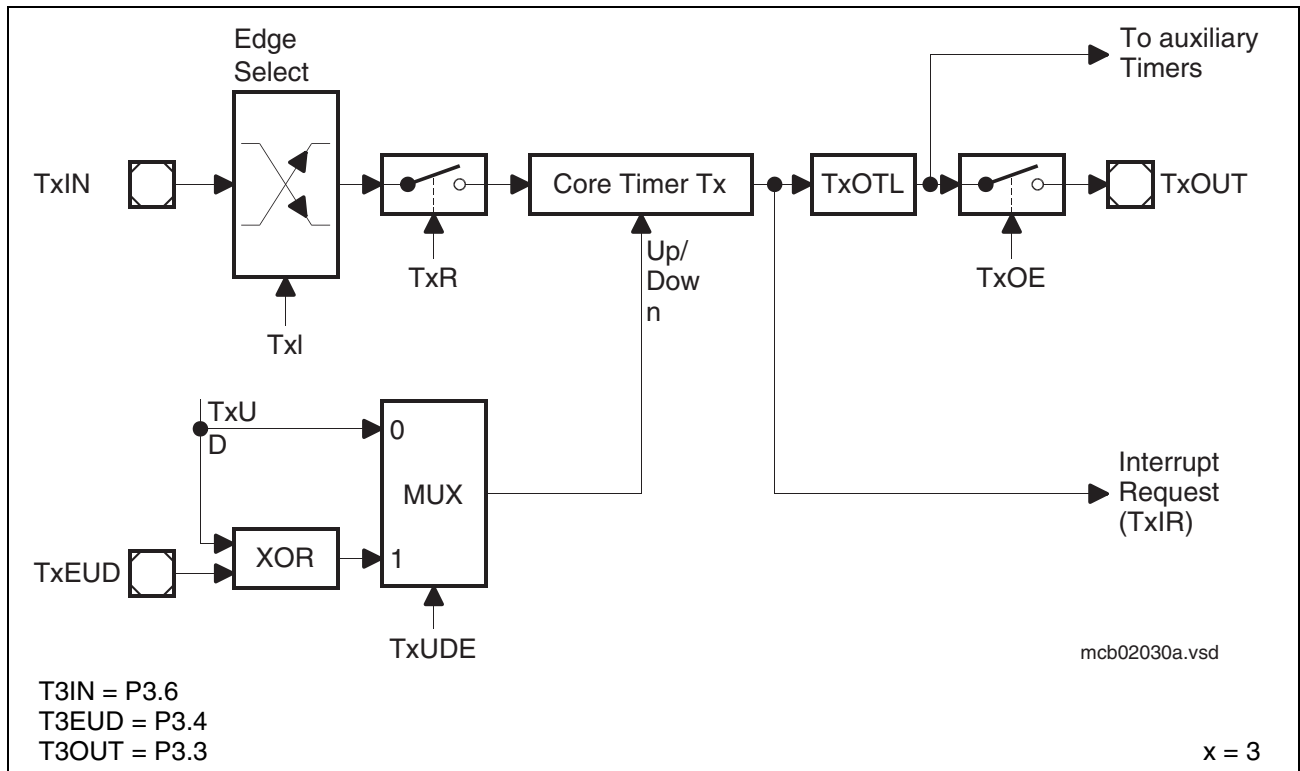
If T3M.0 = '0', the timer is enabled when T3IN shows a low level. A high level at this pin stops the timer. If T3M.0 = '1', pin T3IN must have a high level in order to enable the timer. In addition, the timer can be turned on or off by software using bit T3R. The timer will only run, if T3R = '1' and the gate is active. It will stop, if either T3R = '0' or the gate is inactive.

*Note: A transition of the gate signal at pin T3IN does not cause an interrupt request.*

The General Purpose Timer Units

**Timer 3 in Counter Mode**

Counter mode for the core timer T3 is selected by setting bit field T3M in register T3CON to '001<sub>B</sub>'. In counter mode timer T3 is clocked by a transition at the external input pin T3IN. The event causing an increment or decrement of the timer can be a positive, a negative, or both a positive and a negative transition at this pin. Bit field T3I in control register T3CON selects the triggering transition (see [Table 10-5](#)).



**Figure 10-5 Block Diagram of Core Timer T3 in Counter Mode**

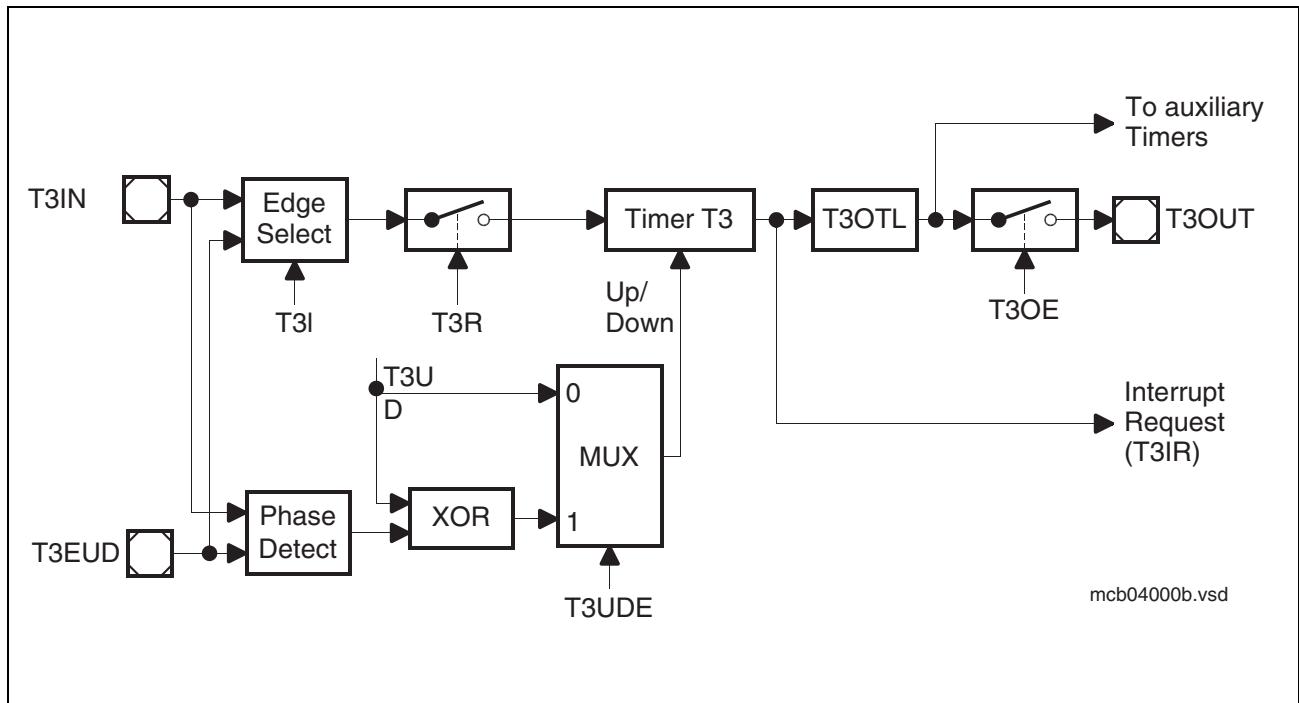
**Table 10-5 GPT1 Core Timer T3 (Counter Mode) Input Edge Selection**

T3I	Triggering Edge for Counter Increment/Decrement
0 0 0	None. Counter T3 is disabled
0 0 1	Positive transition (rising edge) on T3IN
0 1 0	Negative transition (falling edge) on T3IN
0 1 1	Any transition (rising or falling edge) on T3IN
1 X X	Reserved. Do not use this combination

For counter operation, pin T3IN must be configured as input, i.e. the respective direction control bit DPx.y must be '0'. The maximum input frequency which is allowed in counter mode is  $f_{CPU}/16$ . To ensure that a transition of the count input signal which is applied to T3IN is correctly recognized, its level should be held high or low for at least  $8f_{CPU}$  cycles before it changes.

### Timer 3 in Incremental Interface Mode

Incremental Interface mode for the core timer T3 is selected by setting bit field T3M in register T3CON to '110<sub>B</sub>'. In incremental interface mode the two inputs associated with timer T3 (T3IN, T3EUD) are used to interface to an incremental encoder. T3 is clocked by each transition on one or both of the external input pins which gives 2-fold or 4-fold resolution of the encoder input.



**Figure 10-6 Block Diagram of Core Timer T3 in Incremental Interface Mode**

Bitfield T3I in control register T3CON selects the triggering transitions (see [Table 10-6](#)). In this mode the sequence of the transitions of the two input signals is evaluated and generates count pulses as well as the direction signal. So T3 is modified automatically according to the speed and the direction of the incremental encoder and its contents therefore always represent the encoder's current position.

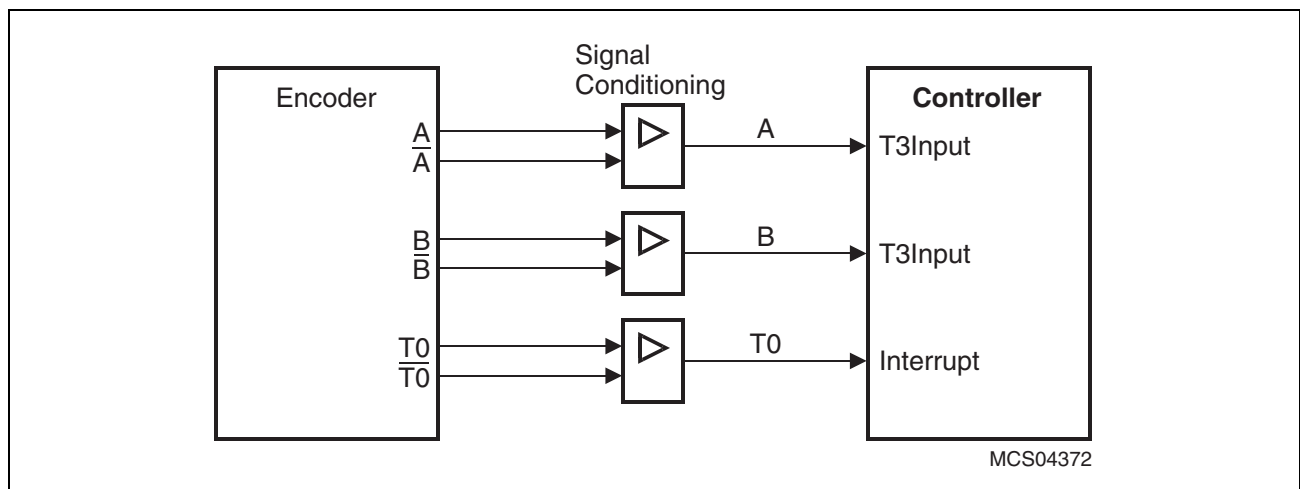
**Table 10-6 GPT1 Core Timer T3 (Incremental Interface Mode) Input Edge Selection**

T3I	Triggering Edge for Counter Increment/Decrement
0 0 0	None. Counter T3 stops.
0 0 1	Any transition (rising or falling edge) on T3IN.
0 1 0	Any transition (rising or falling edge) on T3EUD.
0 1 1	Any transition (rising or falling edge) on any T3 input (T3IN or T3EUD).
1 X X	Reserved. Do not use this combination.

**The General Purpose Timer Units**

The incremental encoder can be connected directly to the C161CS/JC/JI without external interface logic. In a standard system, however, comparators will be employed to convert the encoder's differential outputs (e.g. A,  $\bar{A}$ ) to digital signals (e.g. A). This greatly increases noise immunity.

*Note: The third encoder output Top0, which indicates the mechanical zero position, may be connected to an external interrupt input and trigger a reset of timer T3 (e.g. via PEC transfer from ZEROS).*



**Figure 10-7 Connection of the Encoder to the C161CS/JC/JI**

For incremental interface operation the following conditions must be met:

- Bitfield T3M must be '110<sub>B</sub>'.
- Both pins T3IN and T3EUD must be configured as input, i.e. the respective direction control bits must be '0'.
- Bit T3UDE must be '1' to enable automatic direction control.

The maximum counting frequency which is allowed in incremental interface mode is  $f_{CPU}/16$ . To ensure that a transition of any input signal is correctly recognized, its level should be held high or low for at least  $8f_{CPU}$  cycles before it changes. As in Incremental Interface Mode two input signals with a 90° phase shift are evaluated, their maximum input frequency can be  $f_{CPU}/32$ .

In Incremental Interface Mode the count direction is automatically derived from the sequence in which the input signals change, which corresponds to the rotation direction of the connected sensor. **Table 10-7** summarizes the possible combinations.

**Table 10-7 GPT1 Core Timer T3 (Incremental Interface Mode) Count Direction**

Level on respective other input	T3IN Input		T3EUD Input	
	Rising ↗	Falling ↘	Rising ↗	Falling ↘
<b>High</b>	Down	Up	Up	Down
<b>Low</b>	Up	Down	Down	Up

The General Purpose Timer Units

Figure 10-8 and Figure 10-9 give examples of T3's operation, visualizing count signal generation and direction control. It also shows how input jitter is compensated which might occur if the sensor rests near to one of its switching points.

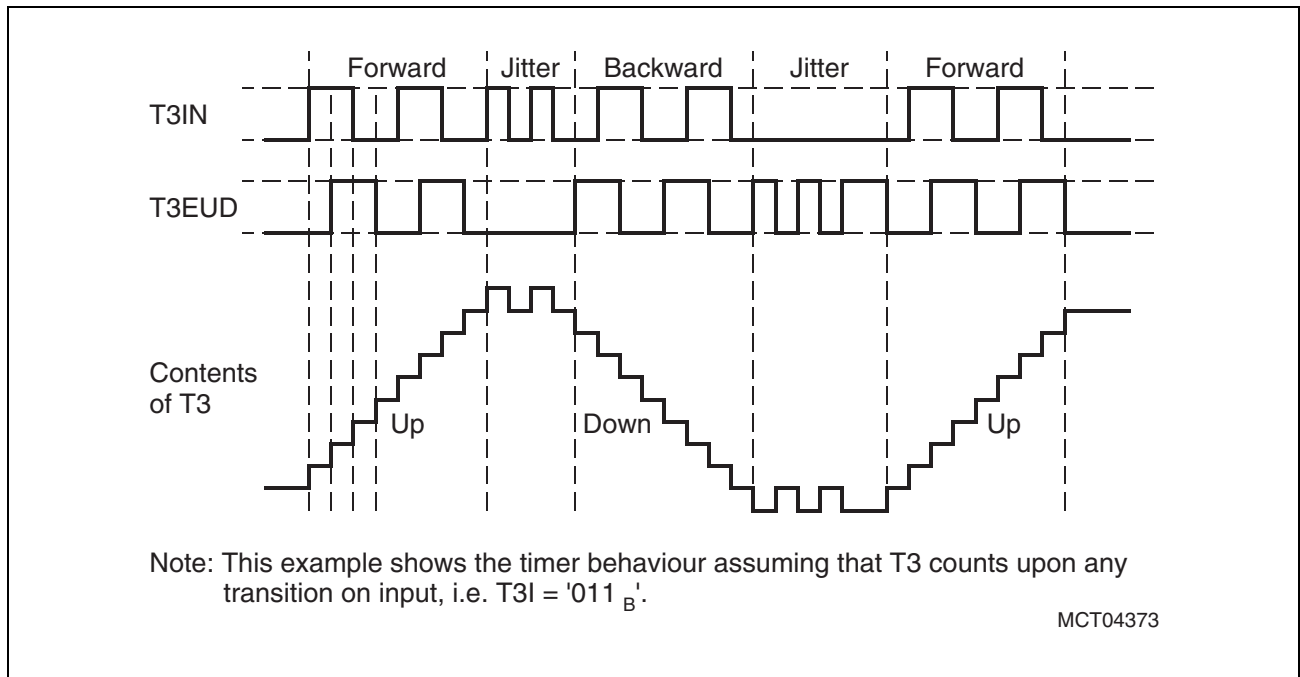


Figure 10-8 Evaluation of the Incremental Encoder Signals

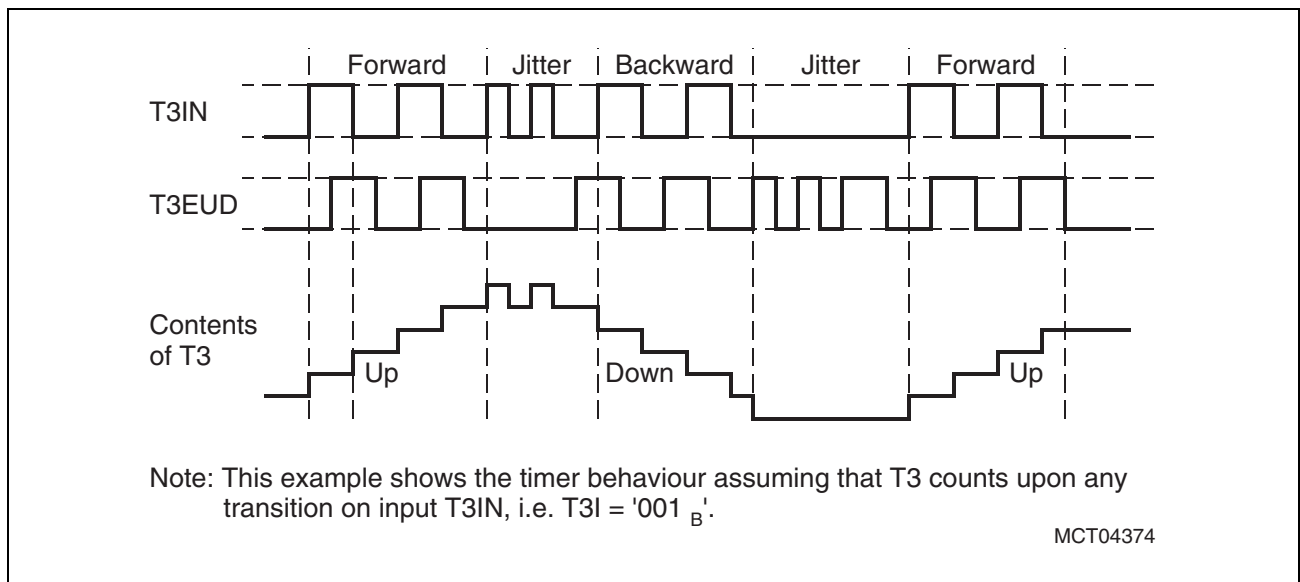


Figure 10-9 Evaluation of the Incremental Encoder Signals

Note: Timer T3 operating in incremental interface mode automatically provides information on the sensor's current position. Dynamic information (speed, acceleration, deceleration) may be obtained by measuring the incoming signal periods. This is facilitated by an additional special capture mode for timer T5.

**The General Purpose Timer Units**

**10.1.2 GPT1 Auxiliary Timers T2 and T4**

Both auxiliary timers T2 and T4 have exactly the same functionality. They can be configured for timer, gated timer, counter, or incremental interface mode with the same options for the timer frequencies and the count signal as the core timer T3. In addition to these 4 counting modes, the auxiliary timers can be concatenated with the core timer, or they may be used as reload or capture registers in conjunction with the core timer.

The individual configuration for timers T2 and T4 is determined by their bitaddressable control registers T2CON and T4CON, which are both organized identically. Note that functions which are present in all 3 timers of block GPT1 are controlled in the same bit positions and in the same manner in each of the specific control registers.

*Note: The auxiliary timers have no output toggle latch and no alternate output function.*

**T2CON**

**Timer 2 Control Register**                      **SFR (FF40<sub>H</sub>/A0<sub>H</sub>)**                      **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	T2 UDE	T2 UD	T2R	T2M		T2I			
-	-	-	-	-	-	-	rw	rw	rw	rw		rw			

**T4CON**

**Timer 4 Control Register**                      **SFR (FF44<sub>H</sub>/A2<sub>H</sub>)**                      **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	T4 UDE	T4 UD	T4R	T4M		T4I			
-	-	-	-	-	-	-	rw	rw	rw	rw		rw			



The General Purpose Timer Units

Bit	Function
<b>TxI</b>	<b>Timer x Input Selection</b> Depends on the Operating Mode, see respective sections.
<b>TxM</b>	<b>Timer x Mode Control</b> (Basic Operating Mode) 000: Timer Mode 001: Counter Mode 010: Gated Timer with Gate active low 011: Gated Timer with Gate active high 100: Reload Mode 101: Capture Mode 110: Incremental Interface Mode 111: <i>Reserved</i> . Do not use this combination.
<b>TxR</b>	<b>Timer x Run Bit</b> 0: Timer/Counter x stops 1: Timer/Counter x runs
<b>TxUD</b>	<b>Timer x Up/Down Control<sup>1)</sup></b>
<b>TxUDE</b>	<b>Timer x External Up/Down Enable<sup>1)</sup></b>

<sup>1)</sup> For the effects of bits TxUD and TxUDE refer to [Table 10-1](#) (see T3 section).

### Count Direction Control for Auxiliary Timers

The count direction of the auxiliary timers can be controlled in the same way as for the core timer T3. The description and the table apply accordingly.

### Timers T2 and T4 in Timer Mode or Gated Timer Mode

When the auxiliary timers T2 and T4 are programmed to timer mode or gated timer mode, their operation is the same as described for the core timer T3. The descriptions, figures and tables apply accordingly with one exception:

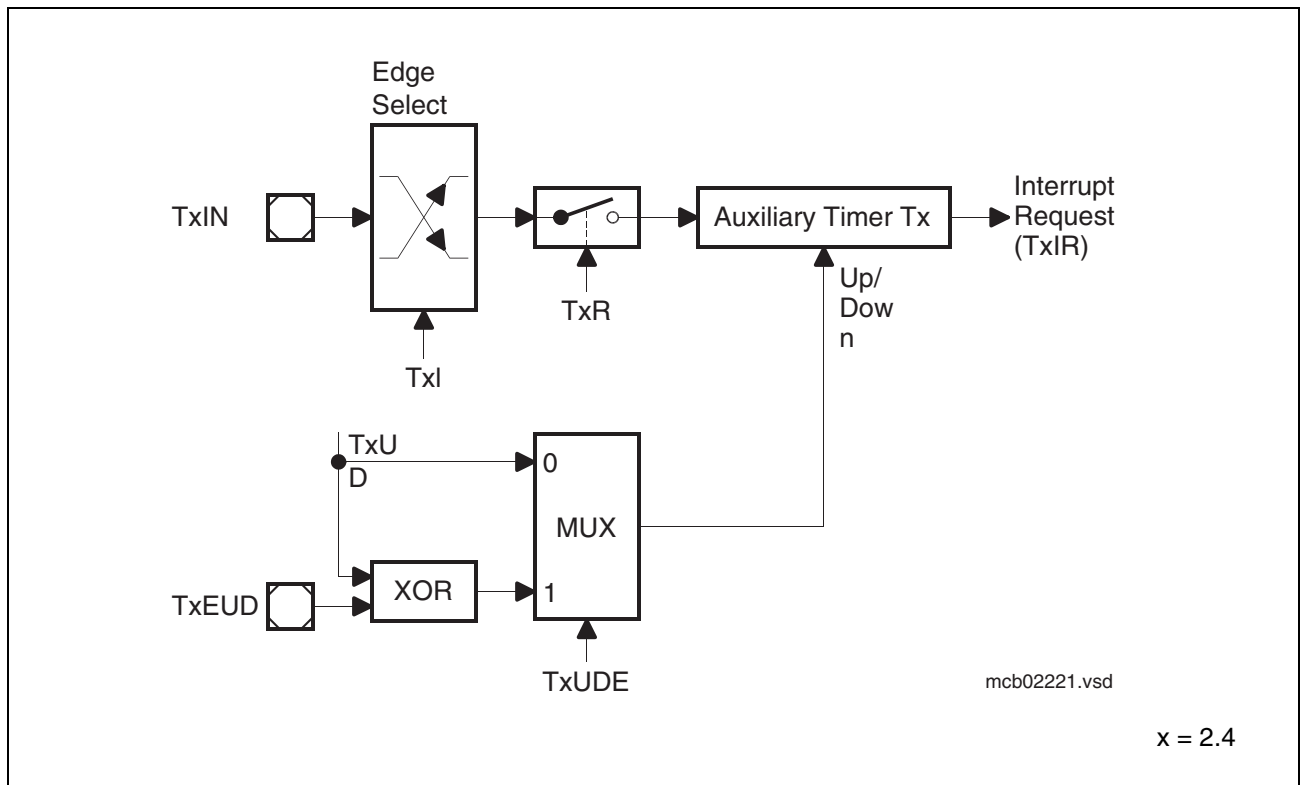
- There is no output toggle latch for T2 and T4.

### Timers T2 and T4 in Incremental Interface Mode

When the auxiliary timers T2 and T4 are programmed to incremental interface mode, their operation is the same as described for the core timer T3. The descriptions, figures and tables apply accordingly.

### Timers T2 and T4 in Counter Mode

Counter mode for the auxiliary timers T2 and T4 is selected by setting bit field TxM in the respective register TxCON to '001<sub>B</sub>'. In counter mode timers T2 and T4 can be clocked either by a transition at the respective external input pin TxIN, or by a transition of timer T3's output toggle latch T3OTL.



**Figure 10-10 Block Diagram of an Auxiliary Timer in Counter Mode**

The event causing an increment or decrement of a timer can be a positive, a negative, or both a positive and a negative transition at either the respective input pin, or at the toggle latch T3OTL.

Bit field TxI in the respective control register TxCON selects the triggering transition (see [Table 10-8](#)).

The General Purpose Timer Units

**Table 10-8 GPT1 Auxiliary Timer (Counter Mode) Input Edge Selection**

T2I/T4I	Triggering Edge for Counter Increment/Decrement
X 0 0	None. Counter Tx is disabled
0 0 1	Positive transition (rising edge) on TxIN
0 1 0	Negative transition (falling edge) on TxIN
0 1 1	Any transition (rising or falling edge) on TxIN
1 0 1	Positive transition (rising edge) of output toggle latch T3OTL
1 1 0	Negative transition (falling edge) of output toggle latch T3OTL
1 1 1	Any transition (rising or falling edge) of output toggle latch T3OTL

*Note: Only state transitions of T3OTL which are caused by the overflows/underflows of T3 will trigger the counter function of T2/T4. Modifications of T3OTL via software will NOT trigger the counter function of T2/T4.*

For counter operation, pin TxIN must be configured as input, i.e. the respective direction control bit must be '0'. The maximum input frequency which is allowed in counter mode is  $f_{CPU}/16$ . To ensure that a transition of the count input signal which is applied to TxIN is correctly recognized, its level should be held for at least  $8 f_{CPU}$  cycles before it changes.

The General Purpose Timer Units

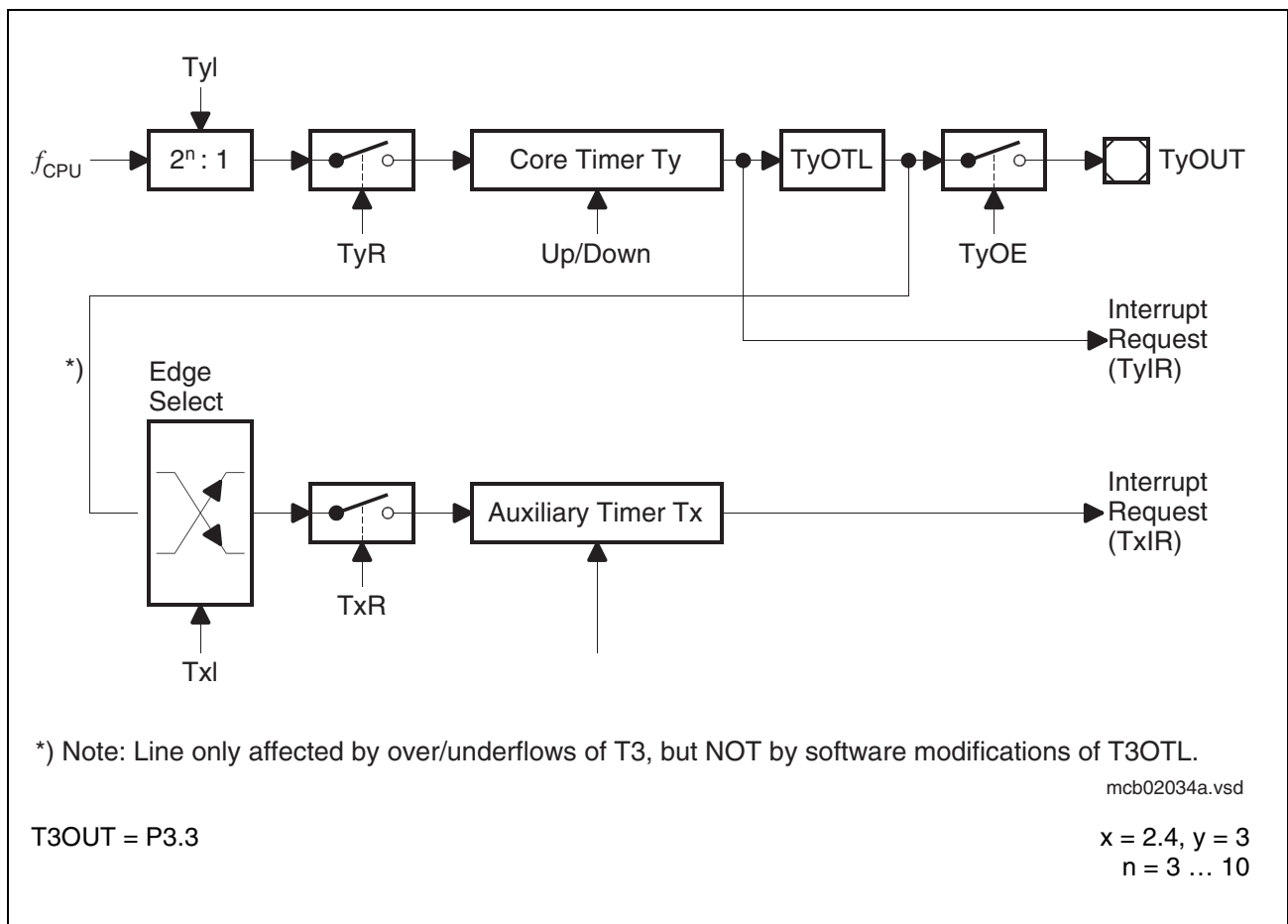
**Timer Concatenation**

Using the toggle bit T3OTL as a clock source for an auxiliary timer in counter mode concatenates the core timer T3 with the respective auxiliary timer. Depending on which transition of T3OTL is selected to clock the auxiliary timer, this concatenation forms a 32-bit or a 33-bit timer/counter.

- **32-bit Timer/Counter:** If both a positive and a negative transition of T3OTL is used to clock the auxiliary timer, this timer is clocked on every overflow/underflow of the core timer T3. Thus, the two timers form a 32-bit timer.
- **33-bit Timer/Counter:** If either a positive or a negative transition of T3OTL is selected to clock the auxiliary timer, this timer is clocked on every second overflow/underflow of the core timer T3. This configuration forms a 33-bit timer (16-bit core timer + T3OTL + 16-bit auxiliary timer).

The count directions of the two concatenated timers are not required to be the same. This offers a wide variety of different configurations.

T3 can operate in timer, gated timer or counter mode in this case.

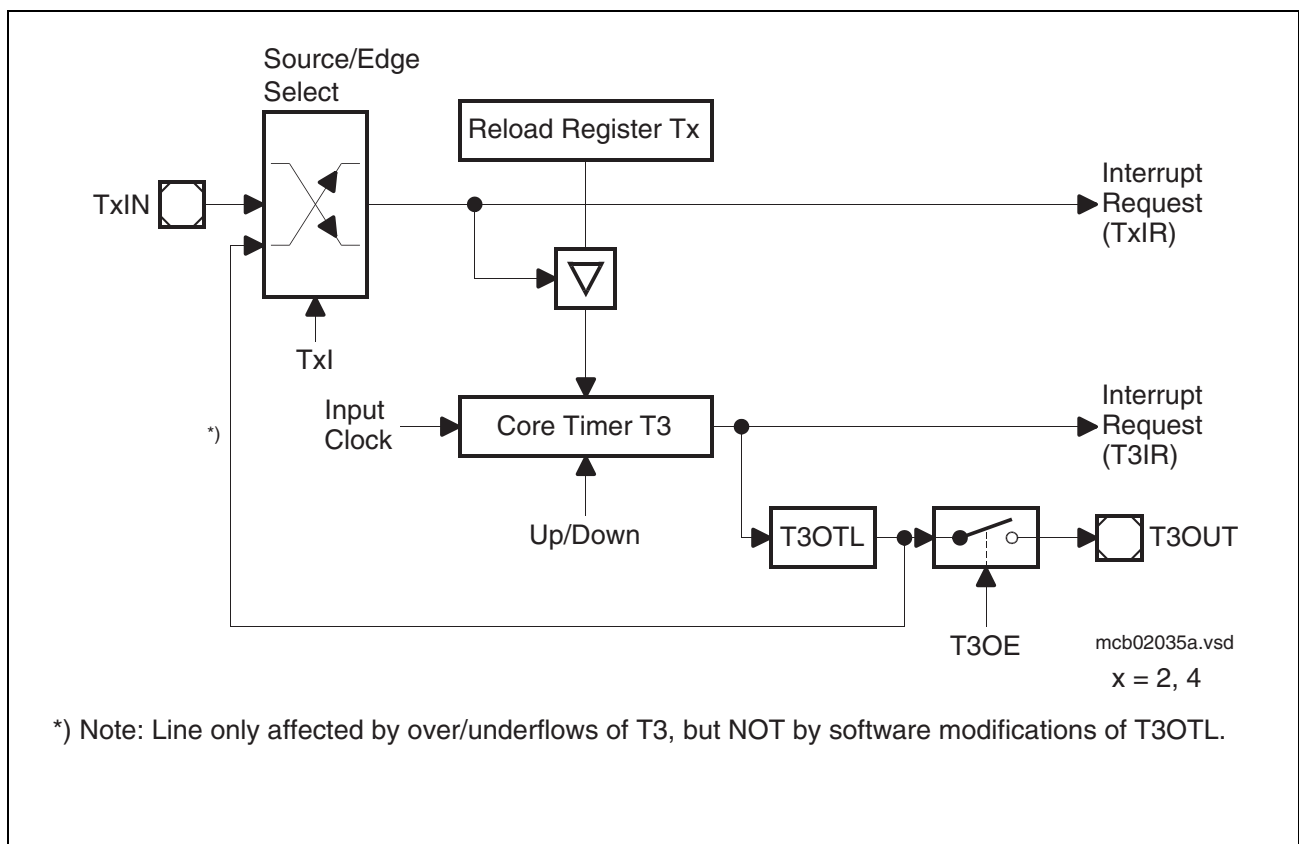


**Figure 10-11 Concatenation of Core Timer T3 and an Auxiliary Timer**

### Auxiliary Timer in Reload Mode

Reload mode for the auxiliary timers T2 and T4 is selected by setting bit field TxM in the respective register TxCON to '100<sub>B</sub>'. In reload mode the core timer T3 is reloaded with the contents of an auxiliary timer register, triggered by one of two different signals. The trigger signal is selected the same way as the clock source for counter mode (see [Table 10-8](#)), i.e. a transition of the auxiliary timer's input or the output toggle latch T3OTL may trigger the reload.

*Note: When programmed for reload mode, the respective auxiliary timer (T2 or T4) stops independent of its run flag T2R or T4R.*



**Figure 10-12 GPT1 Auxiliary Timer in Reload Mode**

Upon a trigger signal T3 is loaded with the contents of the respective timer register (T2 or T4) and the interrupt request flag (T2IR or T4IR) is set.

*Note: When a T3OTL transition is selected for the trigger signal, also the interrupt request flag T3IR will be set upon a trigger, indicating T3's overflow or underflow. Modifications of T3OTL via software will NOT trigger the counter function of T2/T4.*

## The General Purpose Timer Units

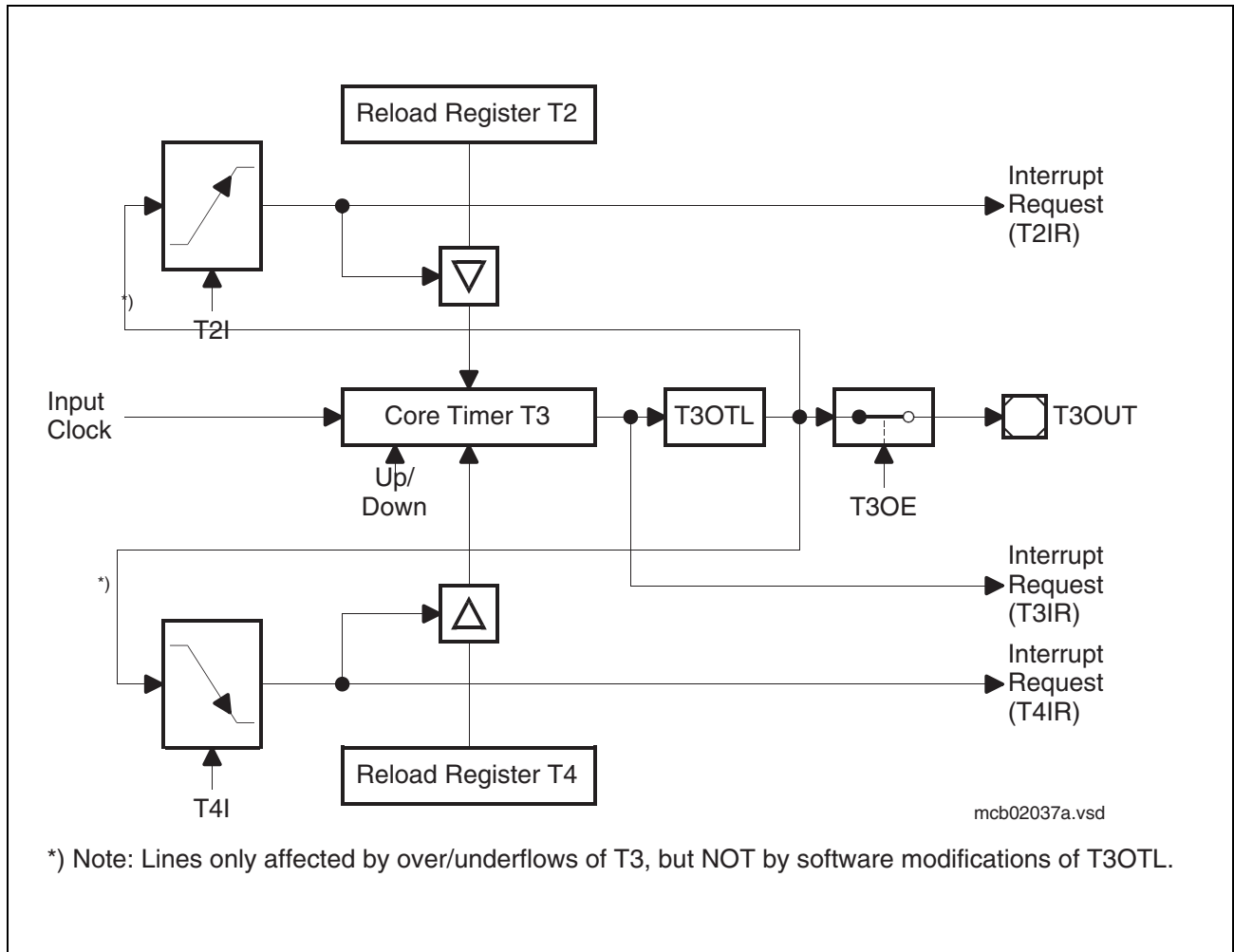
The reload mode triggered by T3OTL can be used in a number of different configurations. Depending on the selected active transition the following functions can be performed:

- If both a positive and a negative transition of T3OTL is selected to trigger a reload, the core timer will be reloaded with the contents of the auxiliary timer each time it overflows or underflows. This is the standard reload mode (reload on overflow/underflow).
- If either a positive or a negative transition of T3OTL is selected to trigger a reload, the core timer will be reloaded with the contents of the auxiliary timer on every second overflow or underflow.
- Using this “single-transition” mode for both auxiliary timers allows to perform very flexible pulse width modulation (PWM). One of the auxiliary timers is programmed to reload the core timer on a positive transition of T3OTL, the other is programmed for a reload on a negative transition of T3OTL. With this combination the core timer is alternately reloaded from the two auxiliary timers.

**Figure 10-13** shows an example for the generation of a PWM signal using the alternate reload mechanism. T2 defines the high time of the PWM signal (reloaded on positive transitions) and T4 defines the low time of the PWM signal (reloaded on negative transitions). The PWM signal can be output on T3OUT with T3OE = ‘1’, port latch = ‘1’ and direction bit = ‘1’. With this method the high and low time of the PWM signal can be varied in a wide range.

*Note: The output toggle latch T3OTL is accessible via software and may be changed, if required, to modify the PWM signal.*

*However, this will NOT trigger the reloading of T3.*



**Figure 10-13 GPT1 Timer Reload Configuration for PWM Generation**

*Note: Although it is possible, it should be avoided to select the same reload trigger event for both auxiliary timers. In this case both reload registers would try to load the core timer at the same time. If this combination is selected, T2 is disregarded and the contents of T4 is reloaded.*

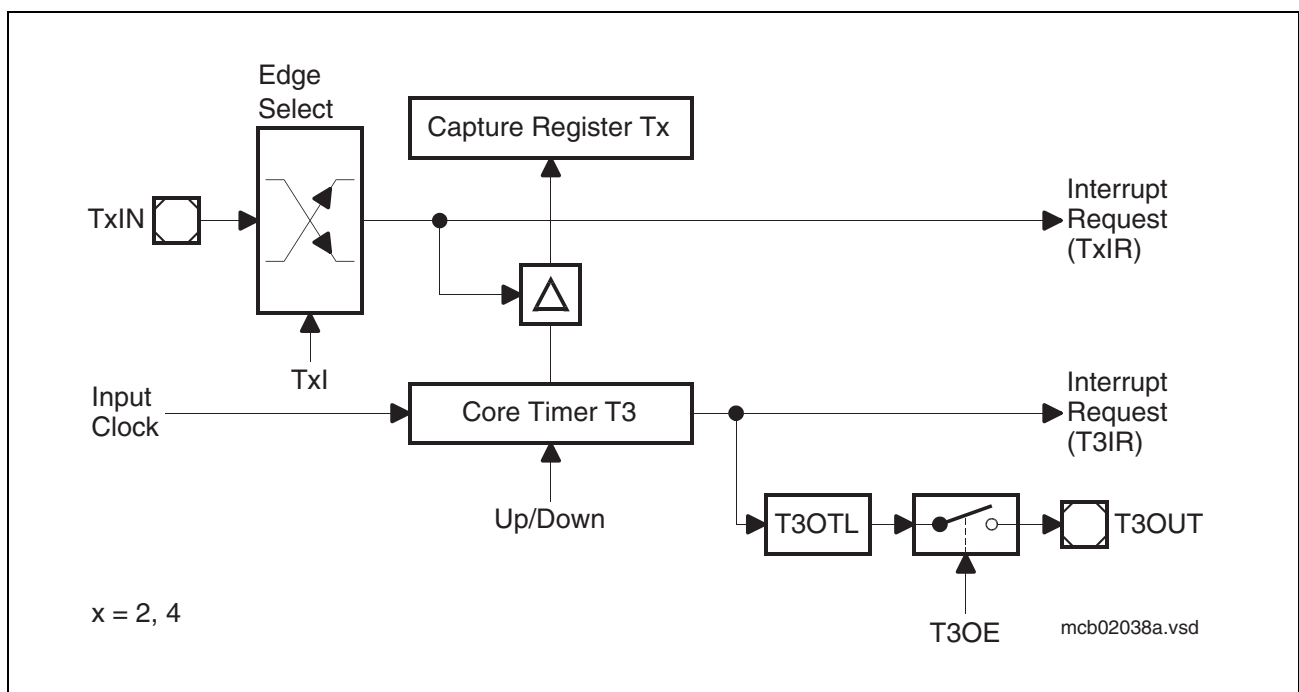
## The General Purpose Timer Units

### Auxiliary Timer in Capture Mode

Capture mode for the auxiliary timers T2 and T4 is selected by setting bit field TxM in the respective register TxCON to '101<sub>B</sub>'. In capture mode the contents of the core timer are latched into an auxiliary timer register in response to a signal transition at the respective auxiliary timer's external input pin TxIN. The capture trigger signal can be a positive, a negative, or both a positive and a negative transition.

The two least significant bits of bit field TxI are used to select the active transition (see [Table 10-8](#)), while the most significant bit TxI.2 is irrelevant for capture mode. It is recommended to keep this bit cleared (TxI.2 = '0').

*Note: When programmed for capture mode, the respective auxiliary timer (T2 or T4) stops independent of its run flag T2R or T4R.*



**Figure 10-14 GPT1 Auxiliary Timer in Capture Mode**

Upon a trigger (selected transition) at the corresponding input pin TxIN the contents of the core timer are loaded into the auxiliary timer register and the associated interrupt request flag TxIR will be set.

*Note: The direction control bits for T2IN and T4IN must be set to '0', and the level of the capture trigger signal should be held high or low for at least  $8f_{CPU}$  cycles before it changes to ensure correct edge detection.*



The General Purpose Timer Units

### 10.1.3 Interrupt Control for GPT1 Timers

When a timer overflows from FFFF<sub>H</sub> to 0000<sub>H</sub> (when counting up), or when it underflows from 0000<sub>H</sub> to FFFF<sub>H</sub> (when counting down), its interrupt request flag (T2IR, T3IR or T4IR) in register TxIC will be set. This will cause an interrupt to the respective timer interrupt vector (T2INT, T3INT or T4INT) or trigger a PEC service, if the respective interrupt enable bit (T2IE, T3IE or T4IE in register TxIC) is set. There is an interrupt control register for each of the three timers.

#### T2IC

Timer 2 Intr. Ctrl. Reg.                      SFR (FF60<sub>H</sub>/B0<sub>H</sub>)                      Reset Value: - - 00<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								T2IR	T2IE	ILVL			GLVL		
-								rwh	rw	rw			rw		

#### T3IC

Timer 3 Intr. Ctrl. Reg.                      SFR (FF62<sub>H</sub>/B1<sub>H</sub>)                      Reset Value: - - 00<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								T3IR	T3IE	ILVL			GLVL		
-								rwh	rw	rw			rw		

#### T4IC

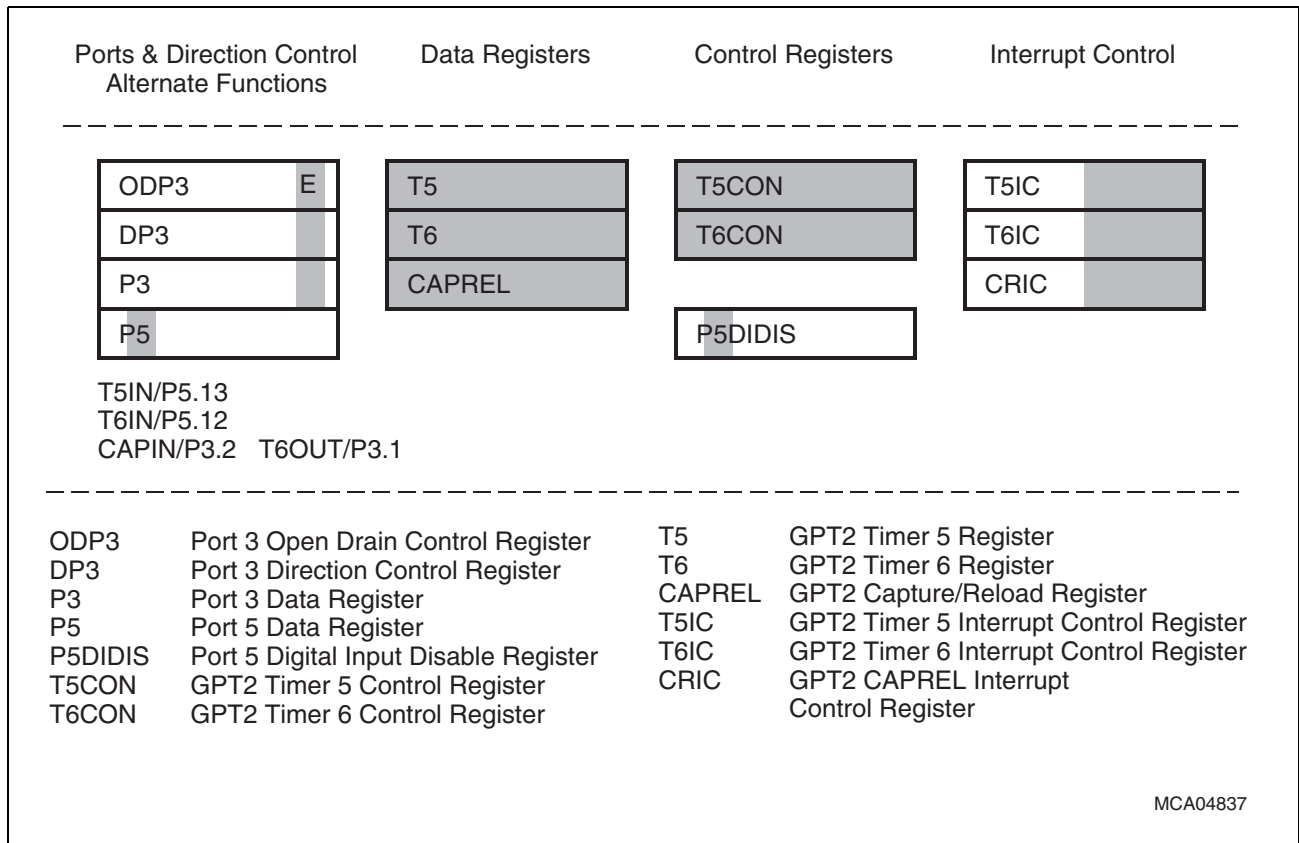
Timer 4 Intr. Ctrl. Reg.                      SFR (FF64<sub>H</sub>/B2<sub>H</sub>)                      Reset Value: - - 00<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								T4IR	T4IE	ILVL			GLVL		
-								rwh	rw	rw			rw		

*Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.*

## 10.2 Timer Block GPT2

From a programmer's point of view, the GPT2 block is represented by a set of SFRs as summarized below. Those portions of port and direction registers which are used for alternate functions by the GPT2 block are shaded.



**Figure 10-15 SFRs and Port Pins Associated with Timer Block GPT2**

Timer block GPT2 supports high precision event control with a maximum resolution of 8 TCL. It includes the two timers T5 and T6, and the 16-bit capture/reload register CAPREL. Timer T6 is referred to as the core timer, and T5 is referred to as the auxiliary timer of GPT2.

Each timer has an alternate input function pin associated with it which serves as the gate control in gated timer mode, or as the count input in counter mode. The count direction (Up/Down) may be programmed via software. An overflow/underflow of T6 is indicated by the output toggle bit T6OTL whose state may be output on an alternate function port pin (T6OUT). In addition, T6 may be reloaded with the contents of CAPREL.

The toggle bit also supports the concatenation of T6 with auxiliary timer T5, while concatenation of T6 with the timers of the CAPCOM units is provided through a direct connection. Triggered by an external signal, the contents of T5 can be captured into register CAPREL, and T5 may optionally be cleared. Both timer T6 and T5 can count up or down, and the current timer value can be read or modified by the CPU in the non-bitaddressable SFRs T5 and T6.

The General Purpose Timer Units

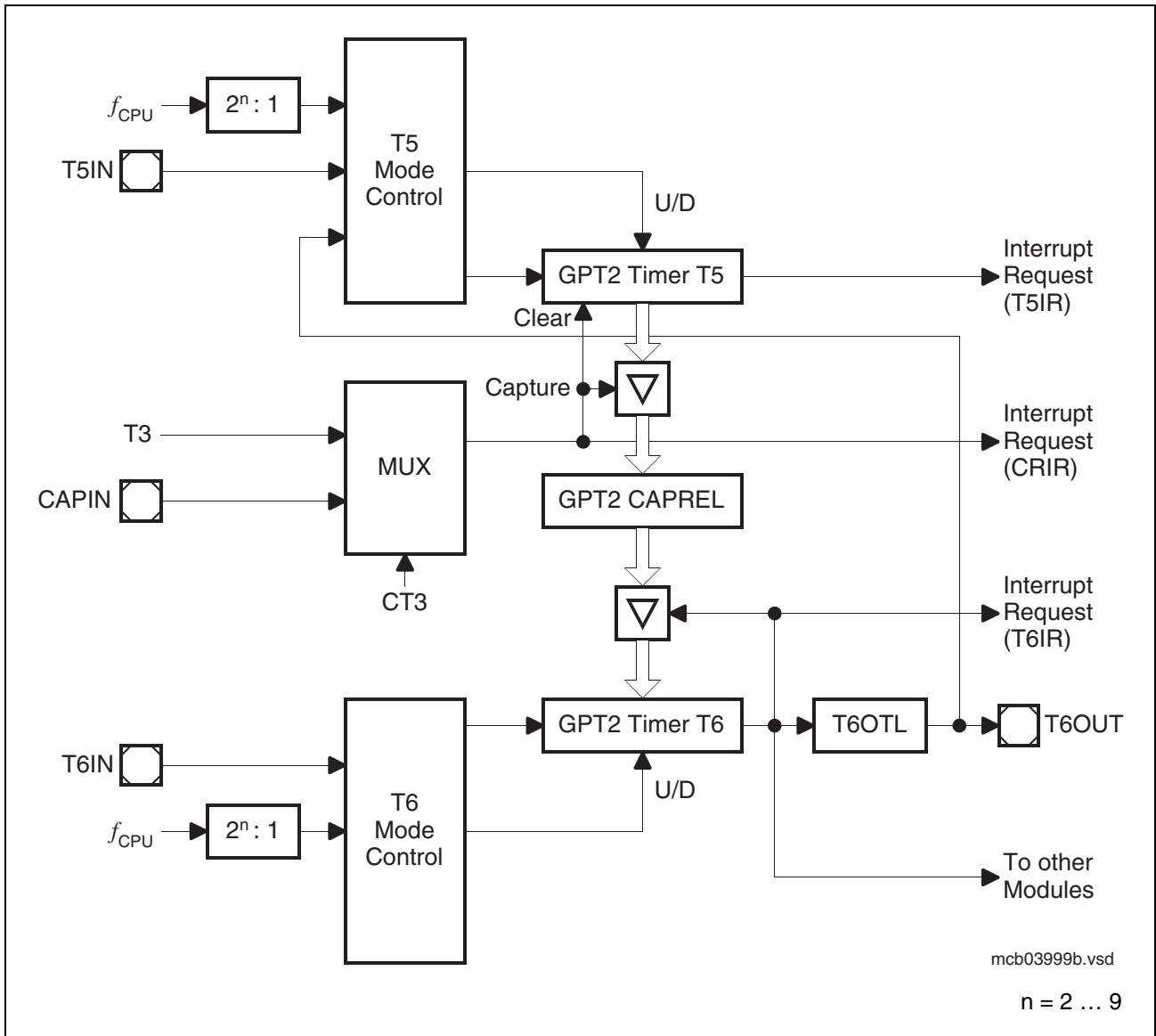


Figure 10-16 GPT2 Block Diagram

The General Purpose Timer Units

10.2.1 GPT2 Core Timer T6

The operation of the core timer T6 is controlled by its bitaddressable control register T6CON.

**T6CON**

Timer 6 Control Register

SFR (FF48<sub>H</sub>/A4<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T6 SR	-	-	-	-	T6 OTL	T6 OE	-	T6 UD	T6R	T6M			T6I		
rw	-	-	-	-	rwh	rw	-	rw	rw	rw			rw		

Bit	Function
<b>T6I</b>	<b>Timer 6 Input Selection</b> Depends on the Operating Mode, see respective sections.
<b>T6M</b>	<b>Timer 6 Mode Control</b> (Basic Operating Mode) 000: Timer Mode 001: Counter Mode 010: Gated Timer with Gate active low 011: Gated Timer with Gate active high 1XX: Reserved. Do not use this combination.
<b>T6R</b>	<b>Timer 6 Run Bit</b> 0: Timer/Counter 6 stops 1: Timer/Counter 6 runs
<b>T6UD</b>	<b>Timer 6 Up/Down Control</b> 0: Timer/Counter 6 counts up 1: Timer/Counter 6 counts down
<b>T6OE</b>	<b>Alternate Output Function Enable</b> 0: Alternate Output Function Disabled 1: Alternate Output Function Enabled
<b>T6OTL</b>	<b>Timer 6 Output Toggle Latch</b> Toggles on each overflow/underflow of T6. Can be set or reset by software.
<b>T6SR</b>	<b>Timer 6 Reload Mode Enable</b> 0: Reload from register CAPREL Disabled 1: Reload from register CAPREL Enabled

---

**The General Purpose Timer Units****Timer 6 Run Bit**

The timer can be started or stopped by software through bit T6R (Timer T6 Run Bit). If T6R = '0', the timer stops. Setting T6R to '1' will start the timer.

In gated timer mode, the timer will only run if T6R = '1' and the gate is active (high or low, as programmed).

**Timer 6 Output Toggle Latch**

An overflow or underflow of timer T6 will clock the toggle bit T6OTL in control register T6CON. T6OTL can also be set or reset by software. Bit T6OE (Alternate Output Function Enable) in register T6CON enables the state of T6OTL to be an alternate function of the external output pin T6OUT. For that purpose, a '1' must be written into the respective port data latch and pin T6OUT must be configured as output by setting the respective direction control bit to '1'. If T6OE = '1', pin T6OUT then outputs the state of T6OTL. If T6OE = '0' pin T6OUT can be used as general purpose IO pin.

In addition, T6OTL can be used in conjunction with the timer over/underflows as an input for the counter function of the auxiliary timer T5. For this purpose, the state of T6OTL does not have to be available at pin T6OUT, because an internal connection is provided for this option.

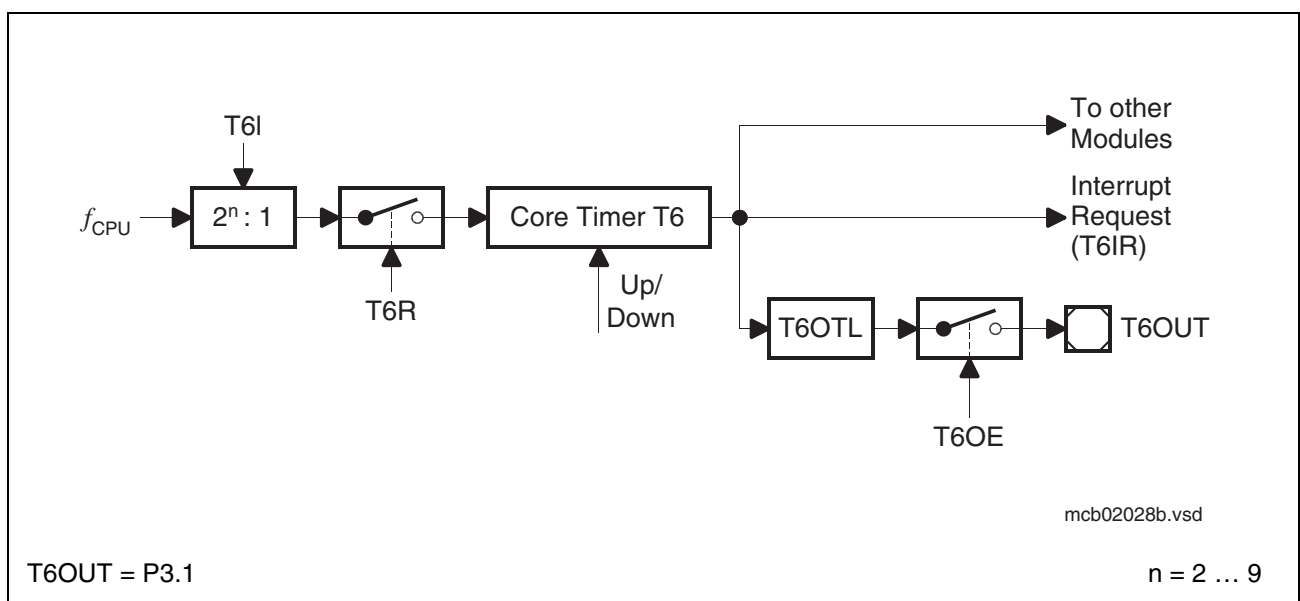
An overflow or underflow of timer T6 can also be used to clock the timers in the CAPCOM units. For this purpose, there is a direct internal connection between timer T6 and the CAPCOM timers.

The General Purpose Timer Units

**Timer 6 in Timer Mode**

Timer mode for the core timer T6 is selected by setting bitfield T6M in register T6CON to '000<sub>B</sub>'. In this mode, T6 is clocked with the internal system clock divided by a programmable prescaler, which is selected by bit field T6I. The input frequency  $f_{T6}$  for timer T6 and its resolution  $r_{T6}$  are scaled linearly with lower clock frequencies  $f_{CPU}$ , as can be seen from the following formula:

$$f_{T6} = \frac{f_{CPU}}{4 \times 2^{\langle T6I \rangle}} \quad , \quad r_{T6} [\mu s] = \frac{4 \times 2^{\langle T6I \rangle}}{f_{CPU} [MHz]}$$



**Figure 10-17 Block Diagram of Core Timer T6 in Timer Mode**

The timer input frequencies, resolution and periods which result from the selected prescaler option are listed in [Table 10-9](#). This table also applies to the gated timer mode of T6 and to the auxiliary timer T5 in timer and gated timer mode. Note that some numbers may be rounded to 3 significant digits.

**Table 10-9 GPT2 Timer Input Frequencies, Resolution and Periods @ 20 MHz**

$f_{CPU} = 20 \text{ MHz}$	Timer Input Selection T5I/T6I							
	000 <sub>B</sub>	001 <sub>B</sub>	010 <sub>B</sub>	011 <sub>B</sub>	100 <sub>B</sub>	101 <sub>B</sub>	110 <sub>B</sub>	111 <sub>B</sub>
<b>Prescaler Factor</b>	4	8	16	32	64	128	256	512
<b>Input Frequency</b>	5 MHz	2.5 MHz	1.25 MHz	625 kHz	312.5 kHz	156.25 kHz	78.125 kHz	39.06 kHz
<b>Resolution</b>	200 ns	400 ns	800 ns	1.6 $\mu$ s	3.2 $\mu$ s	6.4 $\mu$ s	12.8 $\mu$ s	25.6 $\mu$ s
<b>Period</b>	13 ms	26 ms	52.5 ms	105 ms	210 ms	420 ms	840 ms	1.68 s

**The General Purpose Timer Units**

**Table 10-10 GPT2 Timer Input Frequencies, Resolution and Periods @ 25 MHz**

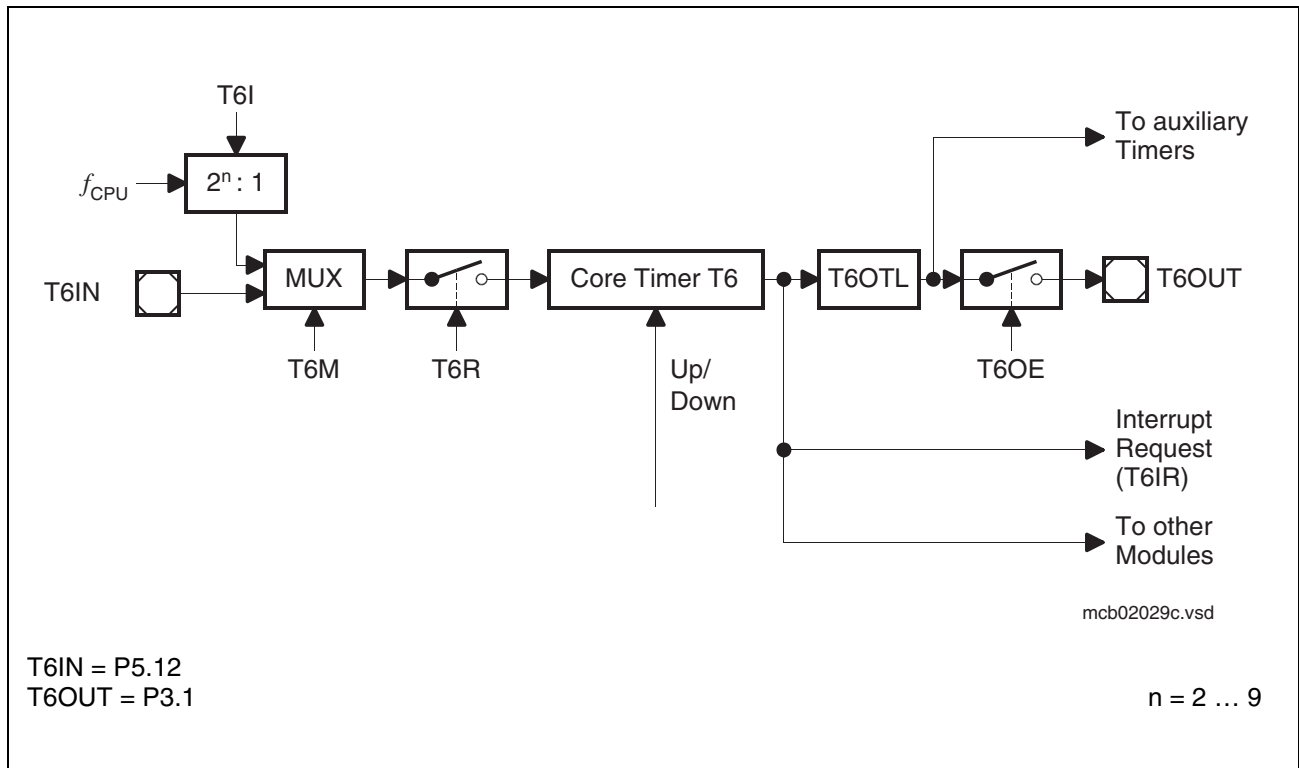
$f_{CPU} = 25 \text{ MHz}$	Timer Input Selection T5I/T6I							
	000 <sub>B</sub>	001 <sub>B</sub>	010 <sub>B</sub>	011 <sub>B</sub>	100 <sub>B</sub>	101 <sub>B</sub>	110 <sub>B</sub>	111 <sub>B</sub>
<b>Prescaler Factor</b>	4	8	16	32	64	128	256	512
<b>Input Frequency</b>	6.25 MHz	3.125 MHz	1.56 MHz	781.25 kHz	390.62 kHz	195.31 kHz	97.66 kHz	48.83 kHz
<b>Resolution</b>	160 ns	320 ns	640 ns	1.28 $\mu\text{s}$	2.56 $\mu\text{s}$	5.12 $\mu\text{s}$	10.2 $\mu\text{s}$	20.5 $\mu\text{s}$
<b>Period</b>	10.5 ms	21.0 ms	42.0 ms	83.9 ms	168 ms	336 ms	671 ms	1.34 s

**Table 10-11 GPT2 Timer Input Frequencies, Resolution and Periods @ 33 MHz**

$f_{CPU} = 33 \text{ MHz}$	Timer Input Selection T5I/T6I							
	000 <sub>B</sub>	001 <sub>B</sub>	010 <sub>B</sub>	011 <sub>B</sub>	100 <sub>B</sub>	101 <sub>B</sub>	110 <sub>B</sub>	111 <sub>B</sub>
<b>Prescaler Factor</b>	4	8	16	32	64	128	256	512
<b>Input Frequency</b>	2.06 MHz	4.125 MHz	2.0625 MHz	1.031 MHz	515.62 kHz	257.81 kHz	128.91 kHz	64.45 kHz
<b>Resolution</b>	121 ns	242 ns	485 ns	970 ns	1.94 $\mu\text{s}$	3.88 $\mu\text{s}$	7.76 $\mu\text{s}$	15.5 $\mu\text{s}$
<b>Period</b>	7.9 ms	15.9 ms	31.8 ms	63.6 ms	127 ms	254 ms	508 ms	1.02 s

### Timer 6 in Gated Timer Mode

Gated timer mode for the core timer T6 is selected by setting bit field T6M in register T6CON to '010<sub>B</sub>' or '011<sub>B</sub>'. Bit T6M.0 (T6CON.3) selects the active level of the gate input. In gated timer mode the same options for the input frequency as for the timer mode are available. However, the input clock to the timer in this mode is gated by the external input pin T6IN (Timer T6 External Input).



**Figure 10-18 Block Diagram of Core Timer T6 in Gated Timer Mode**

If T6M.0 = '0', the timer is enabled when T6IN shows a low level. A high level at this pin stops the timer. If T6M.0 = '1', pin T6IN must have a high level in order to enable the timer. In addition, the timer can be turned on or off by software using bit T6R. The timer will only run, if T6R = '1' and the gate is active. It will stop, if either T6R = '0' or the gate is inactive.

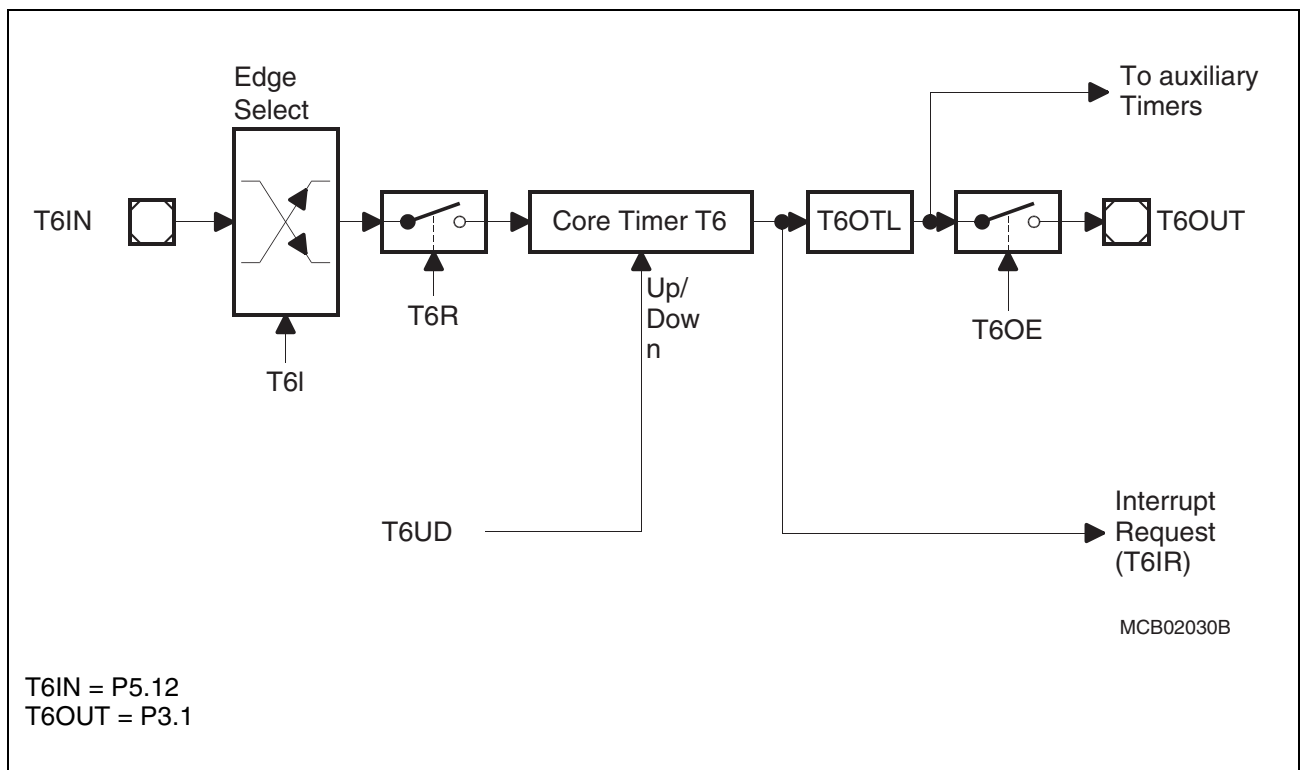
*Note: A transition of the gate signal at pin T6IN does not cause an interrupt request.*



The General Purpose Timer Units

**Timer 6 in Counter Mode**

Counter mode for the core timer T6 is selected by setting bit field T6M in register T6CON to '001<sub>B</sub>'. In counter mode timer T6 is clocked by a transition at the external input pin T6IN, which is an alternate function of P5.12. The event causing an increment or decrement of the timer can be a positive, a negative, or both a positive and a negative transition at this pin. Bit field T6I in control register T6CON selects the triggering transition (see Table 10-12).



**Figure 10-19 Block Diagram of Core Timer T6 in Counter Mode**

**Table 10-12 GPT2 Core Timer T6 (Counter Mode) Input Edge Selection**

T6I	Triggering Edge for Counter Increment/Decrement
0 0 0	None. Counter T6 is disabled
0 0 1	Positive transition (rising edge) on T6IN
0 1 0	Negative transition (falling edge) on T6IN
0 1 1	Any transition (rising or falling edge) on T6IN
1 X X	Reserved. Do not use this combination

The maximum input frequency which is allowed in counter mode is  $f_{CPU} / 8$ . To ensure that a transition of the count input signal which is applied to T6IN is correctly recognized, its level should be held high or low for at least  $4 f_{CPU}$  cycles before it changes.

The General Purpose Timer Units

### 10.2.2 GPT2 Auxiliary Timer T5

The auxiliary timer T5 can be configured for timer, gated timer or counter mode with the same options for the timer frequencies and the count signal as the core timer T6. In addition the auxiliary timer can be concatenated with the core timer (operation in counter mode). Its contents may be captured to register CAPREL upon a selectable trigger.

The individual configuration for timer T5 is determined by its bitaddressable control register T5CON. Note that functions which are present in both timers of block GPT2 are controlled in the same bit positions and in the same manner in each of the specific control registers.

*Note: The auxiliary timer has no output toggle latch and no alternate output function.*

#### T5CON

Timer 5 Control Register

SFR (FF46<sub>H</sub>/A3<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T5 SC	T5 CLR	CI	-	CT3	-	-	T5 UD	T5R	T5M			T5I			
rw	rw	rw	-	rw	-	-	rw	rw	rw			rw			

Bit	Function
T5I	<b>Timer 5 Input Selection</b> Depends on the Operating Mode, see respective sections.
T5M	<b>Timer 5 Mode Control</b> (Basic Operating Mode) 000: Timer Mode 001: Counter Mode 010: Gated Timer with Gate active low 011: Gated Timer with Gate active high 1XX: Reserved. Do not use this combination.
T5R	<b>Timer 5 Run Bit</b> 0: Timer / Counter 5 stops 1: Timer / Counter 5 runs
T5UD	<b>Timer 5 Up / Down Control</b> 0: Timer / Counter 5 counts up 1: Timer / Counter 5 counts down
CT3	<b>Timer 3 Capture Trigger Enable</b> 0: Capture trigger from pin CAPIN 1: Capture trigger from T3 input pins

The General Purpose Timer Units

Bit	Function
<b>CI</b>	<b>Register CAPREL Capture Trigger Selection</b> (depending on bit CT3) 00: Capture disabled 01: Positive transition (rising edge) on CAPIN or any transition on T3IN 10: Negative transition (falling edge) on CAPIN or any transition on T3EUD 11: Any transition (rising or falling edge) on CAPIN or any transition on T3IN or T3EUD
<b>T5CLR</b>	<b>Timer 5 Clear Bit</b> 0: Timer 5 not cleared on a capture 1: Timer 5 is cleared on a capture
<b>T5SC</b>	<b>Timer 5 Capture Mode Enable</b> 0: Capture into register CAPREL disabled 1: Capture into register CAPREL enabled

**Timer T5 in Timer Mode or Gated Timer Mode**

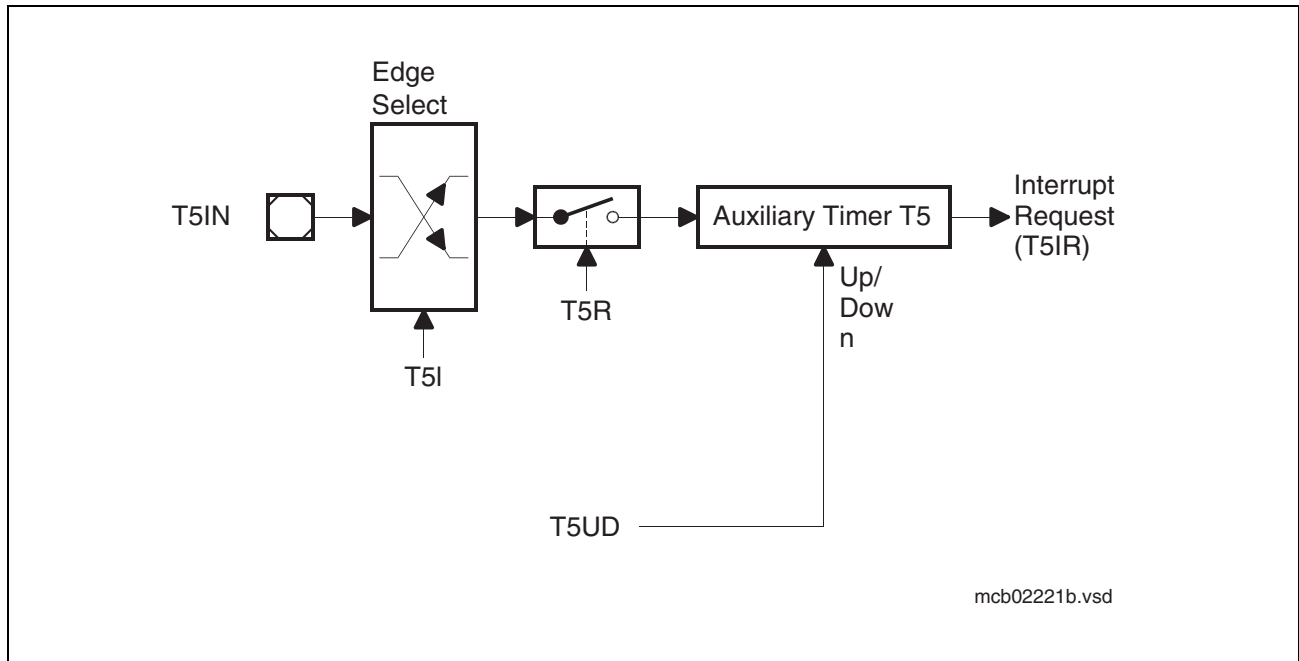
When the auxiliary timer T5 is programmed to timer mode or gated timer mode, its operation is the same as described for the core timer T6. The descriptions, figures and tables apply accordingly with one exception:

- There is no output toggle latch for T5.

**Timer T5 in Counter Mode**

Counter mode for the auxiliary timer T5 is selected by setting bit field T5M in register T5CON to '001<sub>B</sub>'. In counter mode timer T5 can be clocked either by a transition at the external input pin T5IN, or by a transition of timer T6's output toggle latch T6OTL (i.e. timer concatenation).

The General Purpose Timer Units



**Figure 10-20 Block Diagram of Auxiliary Timer T5 in Counter Mode**

The event causing an increment or decrement of the timer can be a positive, a negative, or both a positive and a negative transition at either the input pin, or at the toggle latch T6OTL.

Bitfield T5I in control register T5CON selects the triggering transition (see [Table 10-13](#)).

**Table 10-13 GPT2 Auxiliary Timer (Counter Mode) Input Edge Selection**

T5I	Triggering Edge for Counter Increment/Decrement
X 0 0	None. Counter T5 is disabled
0 0 1	Positive transition (rising edge) on T5IN
0 1 0	Negative transition (falling edge) on T5IN
0 1 1	Any transition (rising or falling edge) on T5IN
1 0 1	Positive transition (rising edge) of output toggle latch T6OTL
1 1 0	Negative transition (falling edge) of output toggle latch T6OTL
1 1 1	Any transition (rising or falling edge) of output toggle latch T6OTL

*Note: Only state transitions of T6OTL which are caused by the overflows/underflows of T6 will trigger the counter function of T5. Modifications of T6OTL via software will NOT trigger the counter function of T5.*

The maximum input frequency which is allowed in counter mode is  $f_{CPU} / 8$ . To ensure that a transition of the count input signal which is applied to T5IN is correctly recognized, its level should be held high or low for at least  $4 f_{CPU}$  cycles before it changes.

The General Purpose Timer Units

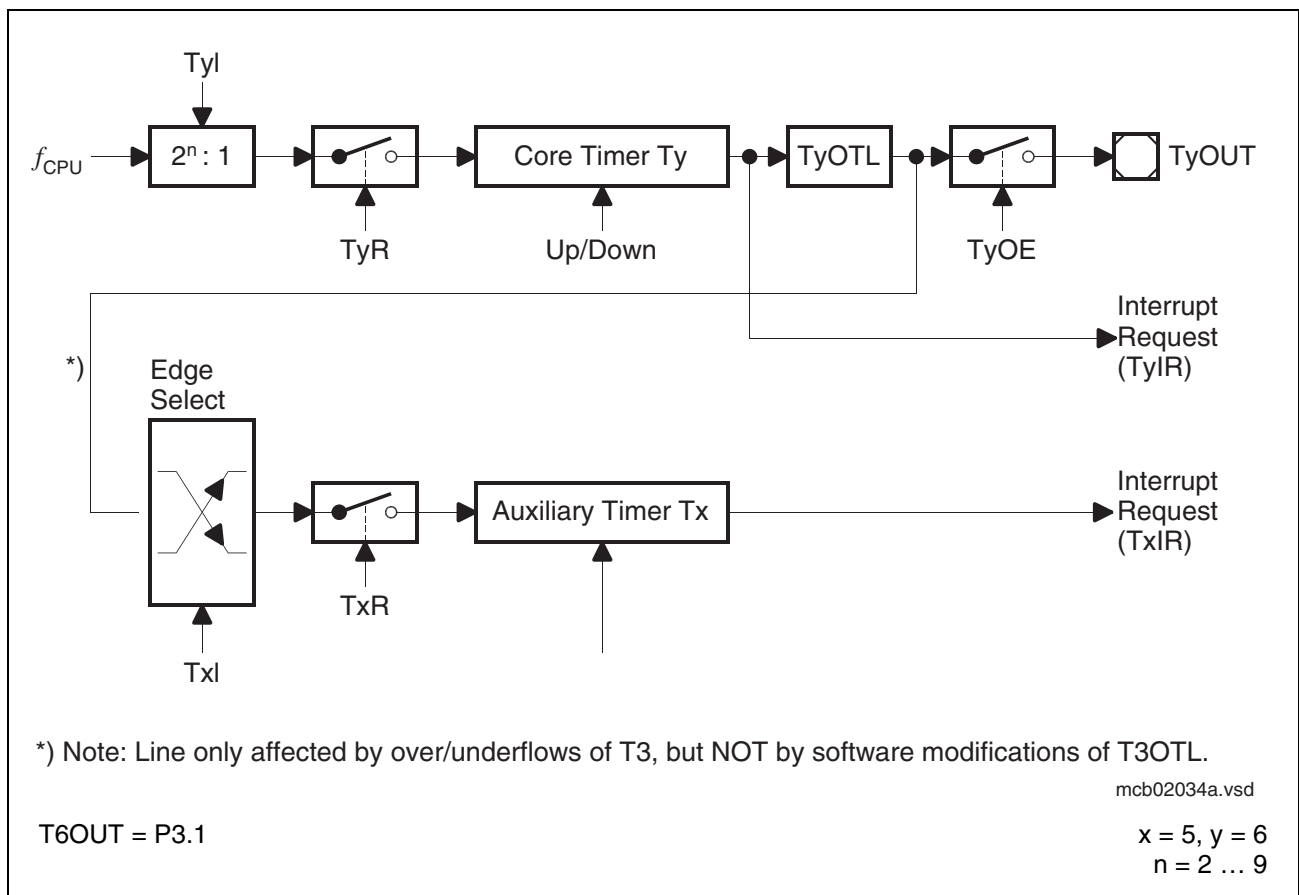
**Timer Concatenation**

Using the toggle bit T6OTL as a clock source for the auxiliary timer in counter mode concatenates the core timer T6 with the auxiliary timer. Depending on which transition of T6OTL is selected to clock the auxiliary timer, this concatenation forms a 32-bit or a 33-bit timer/counter.

- **32-bit Timer/Counter:** If both a positive and a negative transition of T6OTL is used to clock the auxiliary timer, this timer is clocked on every overflow/underflow of the core timer T6. Thus, the two timers form a 32-bit timer.
- **33-bit Timer/Counter:** If either a positive or a negative transition of T6OTL is selected to clock the auxiliary timer, this timer is clocked on every second overflow/underflow of the core timer T6. This configuration forms a 33-bit timer (16-bit core timer + T6OTL + 16-bit auxiliary timer).

The count directions of the two concatenated timers are not required to be the same. This offers a wide variety of different configurations.

T6 can operate in timer, gated timer or counter mode in this case.

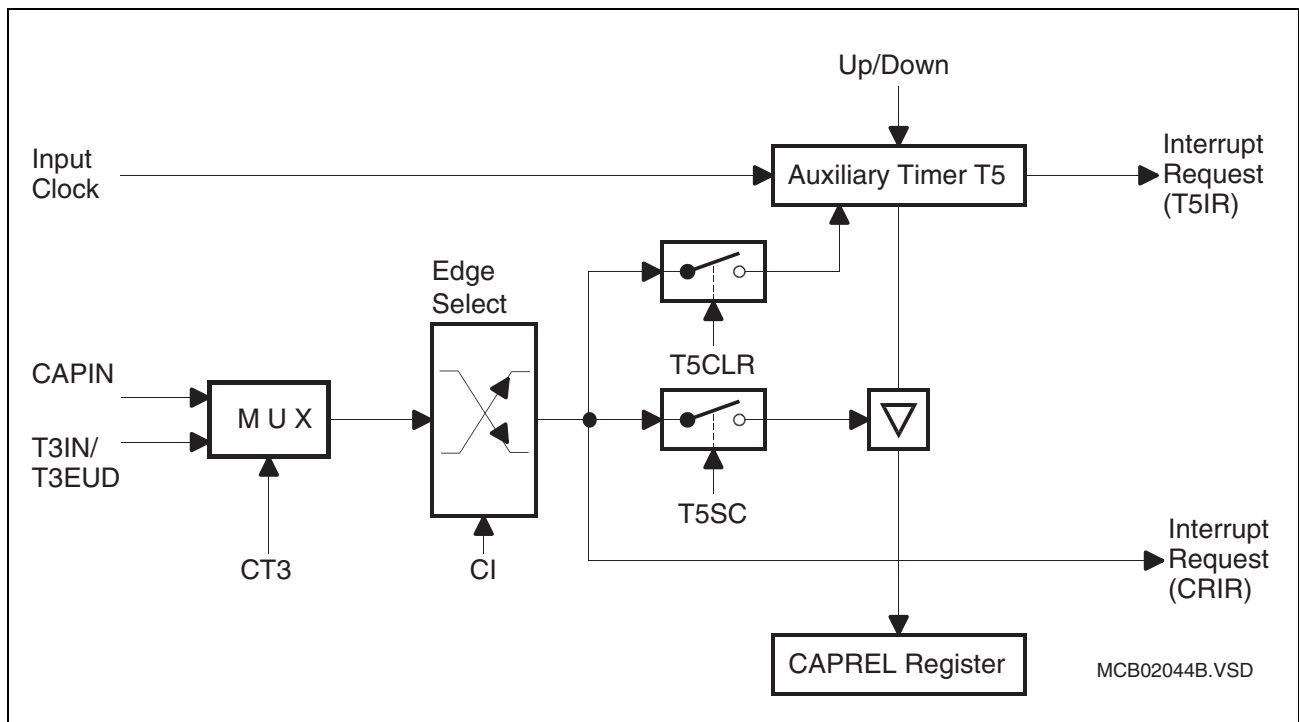


**Figure 10-21 Concatenation of Core Timer T6 and Auxiliary Timer T5**

### GPT2 Capture/Reload Register CAPREL in Capture Mode

This 16-bit register can be used as a capture register for the auxiliary timer T5. This mode is selected by setting bit T5SC = '1' in control register T5CON. Bit CT3 selects the external input pin CAPIN or the input pins of timer T3 as the source for a capture trigger. Either a positive, a negative, or both a positive and a negative transition at pin CAPIN can be selected to trigger the capture function, or transitions on input T3IN or input T3EUD or both inputs T3IN and T3EUD. The active edge is controlled by bit field CI in register T5CON. The same coding is used as in the two least significant bits of bit field T5I (see [Table 10-13](#)).

The maximum input frequency for the capture trigger signal at CAPIN is  $f_{CPU} / 8$ . To ensure that a transition of the capture trigger signal is correctly recognized, its level should be held for at least  $4 f_{CPU}$  cycles before it changes.



**Figure 10-22 GPT2 Register CAPREL in Capture Mode**

When the timer T3 capture trigger is enabled (CT3 = '1') register CAPREL captures the contents of T5 upon transitions of the selected input(s). These values can be used to measure T3's input signals. This is useful e.g. when T3 operates in incremental interface mode, in order to derive dynamic information (speed acceleration) from the input signals.

**The General Purpose Timer Units**

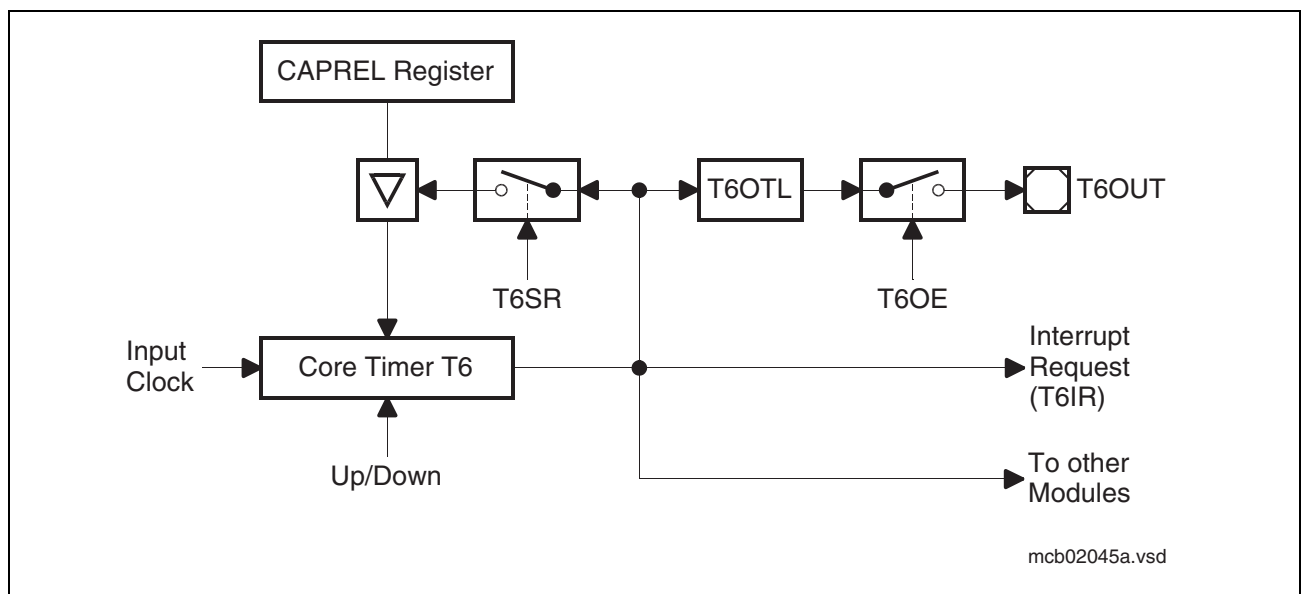
When a selected transition at the selected input pin(s) (CAPIN, T3IN, T3EUD) is detected, the contents of the auxiliary timer T5 are latched into register CAPREL, and interrupt request flag CRIR is set. With the same event, timer T5 can be cleared to 0000<sub>H</sub>. This option is controlled by bit T5CLR in register T5CON. If T5CLR = '0', the contents of timer T5 are not affected by a capture. If T5CLR = '1', timer T5 is cleared after the current timer value has been latched into register CAPREL.

*Note: Bit T5SC only controls whether a capture is performed or not. If T5SC = '0', the selected trigger event can still be used to clear timer T5 or to generate an interrupt request. This interrupt is controlled by the CAPREL interrupt control register CRIC.*

**GPT2 Capture/Reload Register CAPREL in Reload Mode**

This 16-bit register can be used as a reload register for the core timer T6. This mode is selected by setting bit T6SR = '1' in register T6CON. The event causing a reload in this mode is an overflow or underflow of the core timer T6.

When timer T6 overflows from FFFF<sub>H</sub> to 0000<sub>H</sub> (when counting up) or when it underflows from 0000<sub>H</sub> to FFFF<sub>H</sub> (when counting down), the value stored in register CAPREL is loaded into timer T6. This will not set the interrupt request flag CRIR associated with the CAPREL register. However, interrupt request flag T6IR will be set indicating the overflow/underflow of T6.

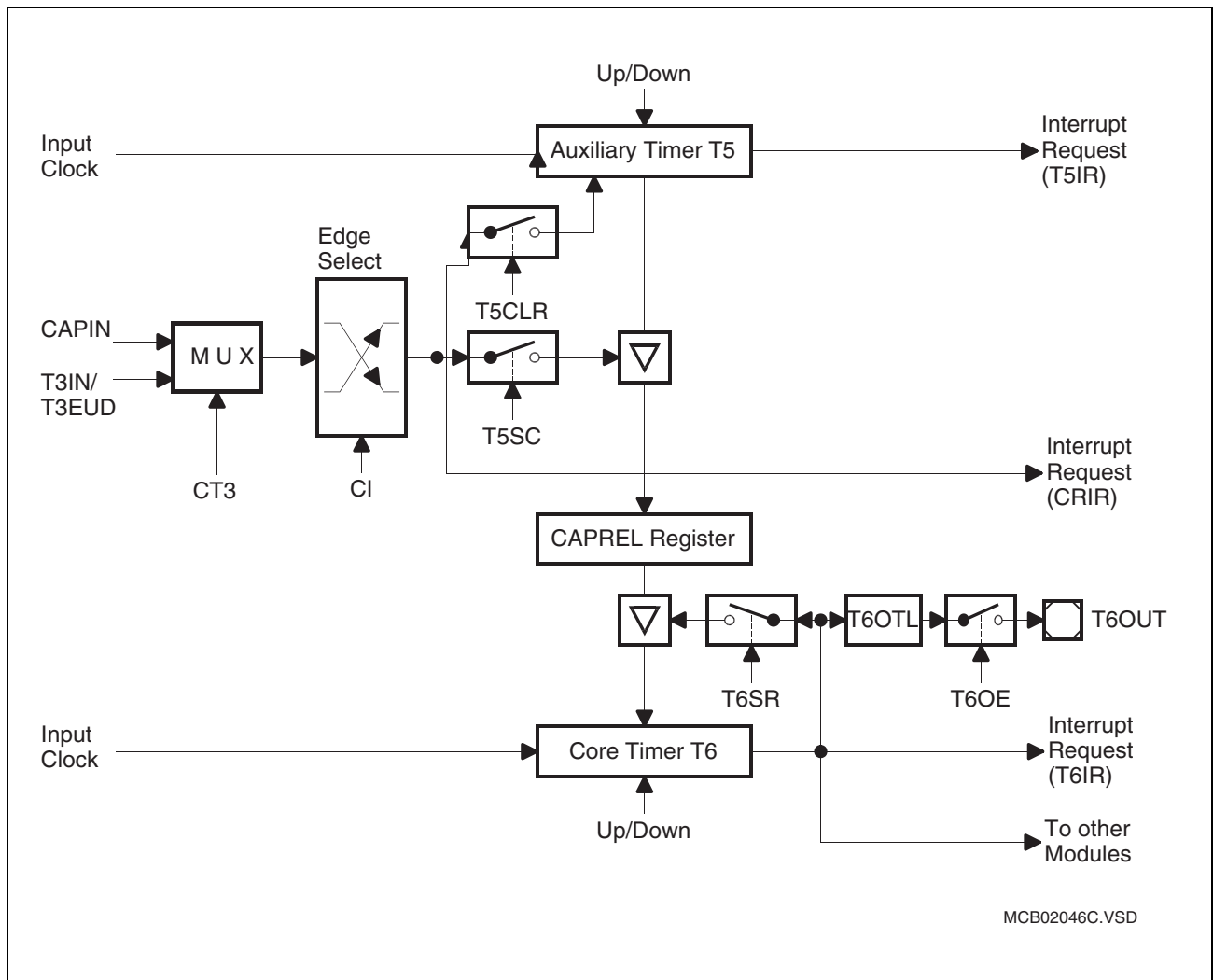


**Figure 10-23 GPT2 Register CAPREL in Reload Mode**

**The General Purpose Timer Units**

**GPT2 Capture/Reload Register CAPREL in Capture-And-Reload Mode**

Since the reload function and the capture function of register CAPREL can be enabled individually by bits T5SC and T6SR, the two functions can be enabled simultaneously by setting both bits. This feature can be used to generate an output frequency that is a multiple of the input frequency.



**Figure 10-24 GPT2 Register CAPREL in Capture-And-Reload Mode**

This combined mode can be used to detect consecutive external events which may occur aperiodically, but where a finer resolution, that means, more 'ticks' within the time between two external events is required.



---

**The General Purpose Timer Units**

For this purpose, the time between the external events is measured using timer T5 and the CAPREL register. Timer T5 runs in timer mode counting up with a frequency of e.g.  $f_{CPU}/32$ . The external events are applied to pin CAPIN. When an external event occurs, the timer T5 contents are latched into register CAPREL, and timer T5 is cleared ( $T5CLR = '1'$ ). Thus, register CAPREL always contains the correct time between two events, measured in timer T5 increments. Timer T6, which runs in timer mode counting down with a frequency of e.g.  $f_{CPU} / 4$ , uses the value in register CAPREL to perform a reload on underflow. This means, the value in register CAPREL represents the time between two underflows of timer T6, now measured in timer T6 increments. Since timer T6 runs 8 times faster than timer T5, it will underflow 8 times within the time between two external events. Thus, the underflow signal of timer T6 generates 8 'ticks'. Upon each underflow, the interrupt request flag T6IR will be set and bit T6OTL will be toggled. The state of T6OTL may be output on pin T6OUT. This signal has 8 times more transitions than the signal which is applied to pin CAPIN.

The underflow signal of timer T6 can furthermore be used to clock one or more of the timers of the CAPCOM units, which gives the user the possibility to set compare events based on a finer resolution than that of the external events.

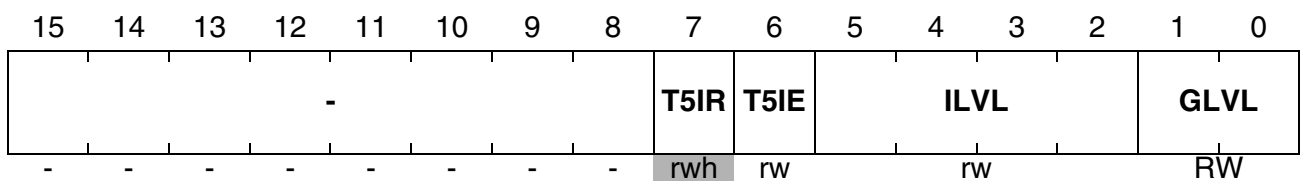
**The General Purpose Timer Units**

**10.2.3 Interrupt Control for GPT2 Timers and CAPREL**

When a timer overflows from FFFF<sub>H</sub> to 0000<sub>H</sub> (when counting up), or when it underflows from 0000<sub>H</sub> to FFFF<sub>H</sub> (when counting down), its interrupt request flag (T5IR or T6IR) in register TxIC will be set. Whenever a transition according to the selection in bit field CI is detected at pin CAPIN, interrupt request flag CRIR in register CRIC is set. Setting any request flag will cause an interrupt to the respective timer or CAPREL interrupt vector (T5INT, T6INT or CRINT) or trigger a PEC service, if the respective interrupt enable bit (T5IE or T6IE in register TxIC, CRIE in register CRIC) is set. There is an interrupt control register for each of the two timers and for the CAPREL register.

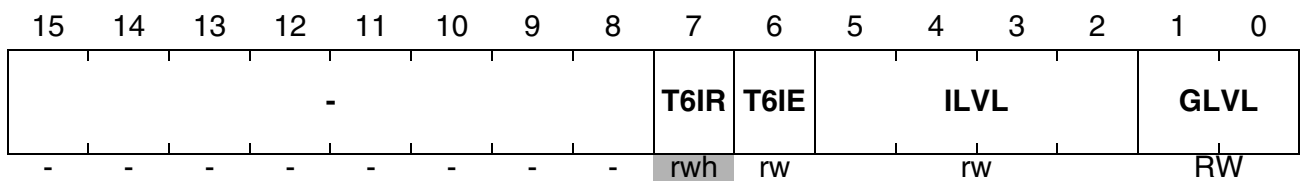
**T5IC**

**Timer 5 Intr. Ctrl. Reg.                      SFR (FF66<sub>H</sub>/B3<sub>H</sub>)                      Reset Value: - - 00<sub>H</sub>**



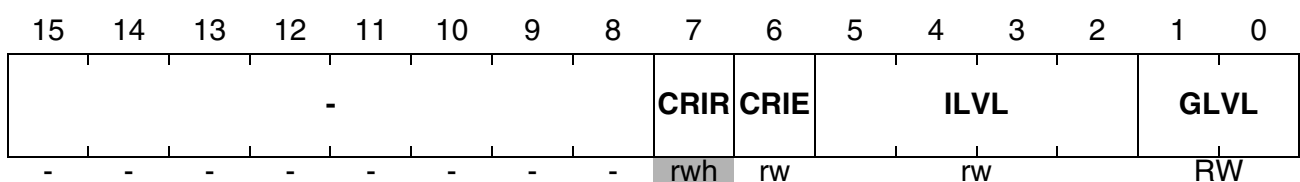
**T6IC**

**Timer 6 Intr. Ctrl. Reg.                      SFR (FF68<sub>H</sub>/B4<sub>H</sub>)                      Reset Value: - - 00<sub>H</sub>**



**CRIC**

**CAPREL Intr. Ctrl. Reg.                      SFR (FF6A<sub>H</sub>/B5<sub>H</sub>)                      Reset Value: - - 00<sub>H</sub>**



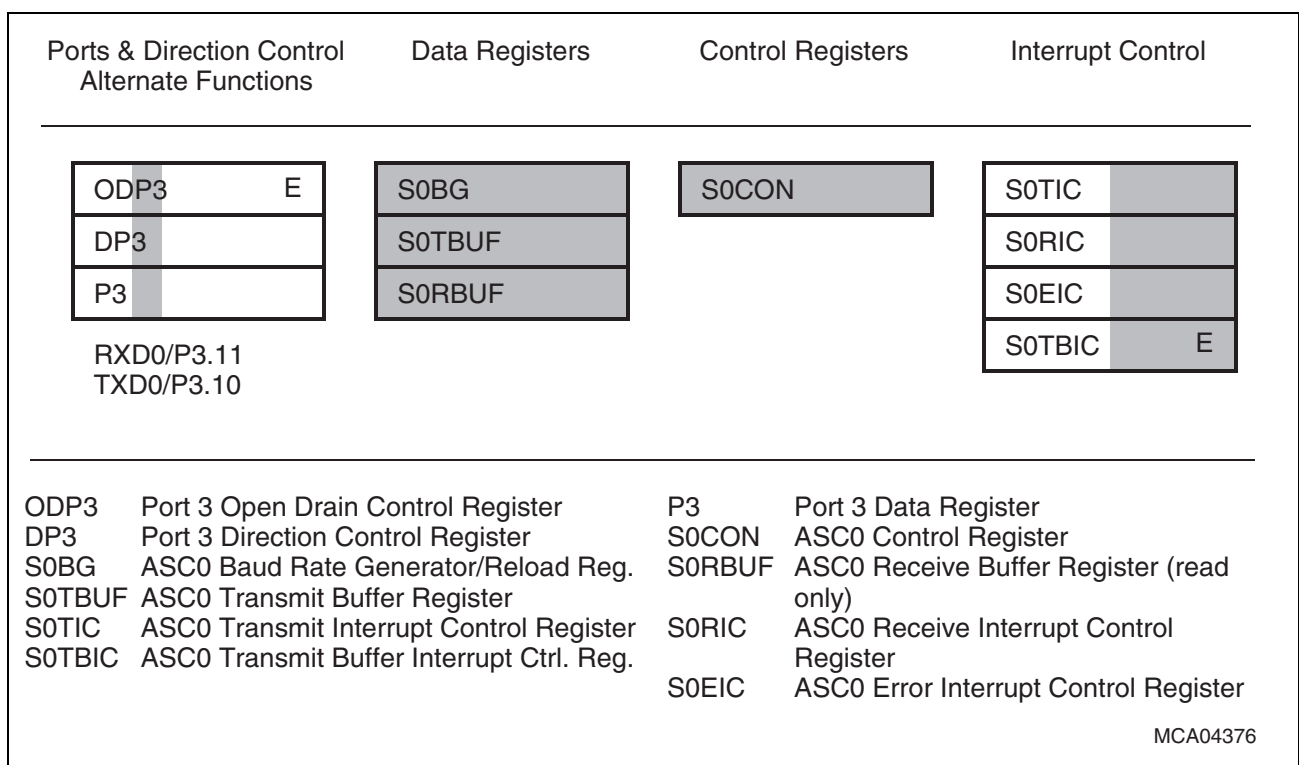
*Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.*

**The Asynchronous/Synchronous Serial Interface**

## 11 The Asynchronous/Synchronous Serial Interface

The Asynchronous/Synchronous Serial Interface ASC0 provides serial communication between the C161CS/JC/JI and other microcontrollers, microprocessors or external peripherals.

The ASC0 supports full-duplex asynchronous communication up to 781 kBaud/1.03 MBaud and half-duplex synchronous communication up to 3.1/4.1 MBaud (@ 25/33 MHz CPU clock). In synchronous mode, data are transmitted or received synchronous to a shift clock which is generated by the C161CS/JC/JI. In asynchronous mode, 8- or 9-bit data transfer, parity generation, and the number of stop bits can be selected. Parity, framing, and overrun error detection is provided to increase the reliability of data transfers. Transmission and reception of data is double-buffered. For multiprocessor communication, a mechanism to distinguish address from data bytes is included. Testing is supported by a loop-back option. A 13-bit baud rate generator provides the ASC0 with a separate serial clock signal.



**Figure 11-1 SFRs and Port Pins Associated with ASC0**

### The Asynchronous/Synchronous Serial Interface

The operating mode of the serial channel ASC0 is controlled by its bitaddressable control register S0CON. This register contains control bits for mode and error check selection, and status flags for error identification.

#### S0CON

#### ASC0 Control Register

SFR (FFB0<sub>H</sub>/D8<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S0R	S0 LB	S0 BRS	S0 ODD	-	S0 OE	S0 FE	S0 PE	S0 OEN	S0 FEN	S0 PEN	S0 REN	S0 STP	S0M		
rw	rw	rw	rw	-	rwh	rwh	rwh	rw	rw	rw	rwh	rw	rw		

Bit	Function
<b>S0M</b>	<b>ASC0 Mode Control</b> 000: 8-bit data synchronous operation 001: 8-bit data async. operation 010: Reserved. Do not use this combination! 011: 7-bit data + parity async. operation 100: 9-bit data async. operation 101: 8-bit data + wake up bit async. operation 110: Reserved. Do not use this combination! 111: 8-bit data + parity async. operation
<b>S0STP</b>	<b>Number of Stop Bits Selection</b> async. operation 0: One stop bit 1: Two stop bits
<b>S0REN</b>	<b>Receiver Enable Bit</b> 0: Receiver disabled 1: Receiver enabled (Reset by hardware after reception of byte in synchronous mode)
<b>S0PEN</b>	<b>Parity Check Enable Bit</b> async. operation 0: Ignore parity 1: Check parity
<b>S0FEN</b>	<b>Framing Check Enable Bit</b> async. operation 0: Ignore framing errors 1: Check framing errors
<b>S0OEN</b>	<b>Overrun Check Enable Bit</b> 0: Ignore overrun errors 1: Check overrun errors

**The Asynchronous/Synchronous Serial Interface**

<b>Bit</b>	<b>Function</b>
<b>S0PE</b>	<b>Parity Error Flag</b> Set by hardware on a parity error (S0PEN = '1'). Must be reset by software.
<b>S0FE</b>	<b>Framing Error Flag</b> Set by hardware on a framing error (S0FEN = '1'). Must be reset by software.
<b>S0OE</b>	<b>Overrun Error Flag</b> Set by hardware on an overrun error (S0OEN = '1'). Must be reset by software.
<b>S0ODD</b>	<b>Parity Selection Bit</b> 0: Even parity (parity bit set on odd number of '1's in data) 1: Odd parity (parity bit set on even number of '1's in data)
<b>S0BRS</b>	<b>Baudrate Selection Bit</b> 0: Divide clock by reload-value + constant (depending on mode) 1: Additionally reduce serial clock to 2/3rd
<b>S0LB</b>	<b>Loopback Mode Enable Bit</b> 0: Standard transmit/receive mode 1: Loopback mode enabled
<b>S0R</b>	<b>Baudrate Generator Run Bit</b> 0: Baudrate generator disabled (ASC0 inactive) 1: Baudrate generator enabled

A transmission is started by writing to the Transmit Buffer register S0TBUF (via an instruction or a PEC data transfer). Only the number of data bits which is determined by the selected operating mode will actually be transmitted, i.e. bits written to positions 9 through 15 of register S0TBUF are always insignificant. After a transmission has been completed, the transmit buffer register is cleared to 0000<sub>H</sub>.

Data transmission is double-buffered, so a new character may be written to the transmit buffer register, before the transmission of the previous character is complete. This allows the transmission of characters back-to-back without gaps.

Data reception is enabled by the Receiver Enable Bit S0REN. After reception of a character has been completed, the received data and, if provided by the selected operating mode, the received parity bit can be read from the (read-only) Receive Buffer register S0RBUF. Bits in the upper half of S0RBUF which are not valid in the selected operating mode will be read as zeros.

Data reception is double-buffered, so that reception of a second character may already begin before the previously received character has been read out of the receive buffer register. In all modes, receive buffer overrun error detection can be selected through bit

---

## The Asynchronous/Synchronous Serial Interface

S0OEN. When enabled, the overrun error status flag S0OE and the error interrupt request flag S0EIR will be set when the receive buffer register has not been read by the time reception of a second character is complete. The previously received character in the receive buffer is overwritten.

**The Loop-Back option** (selected by bit S0LB) allows the data currently being transmitted to be received simultaneously in the receive buffer. This may be used to test serial communication routines at an early stage without having to provide an external network. In loop-back mode the alternate input/output functions of the Port 3 pins are not necessary.

*Note: Serial data transmission or reception is only possible when the Baud Rate Generator Run Bit S0R is set to '1'. Otherwise the serial interface is idle.*

*Do not program the mode control field S0M in register S0CON to one of the reserved combinations to avoid unpredictable behavior of the serial interface.*

The Asynchronous/Synchronous Serial Interface

11.1 Asynchronous Operation

Asynchronous mode supports full-duplex communication, where both transmitter and receiver use the same data frame format and the same baud rate. Data is transmitted on pin TXD0 and received on pin RXD0. These signals are alternate port functions.

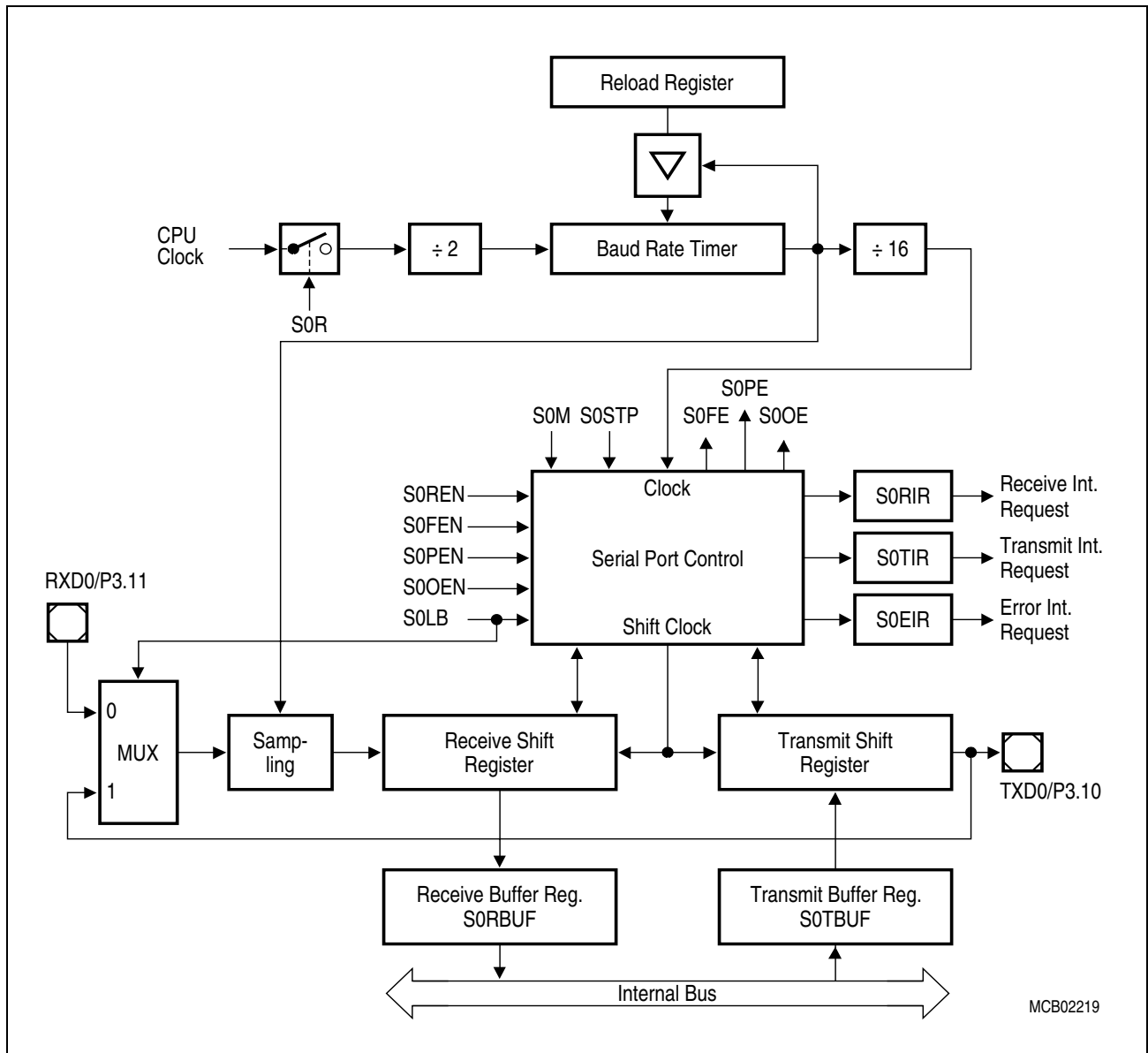


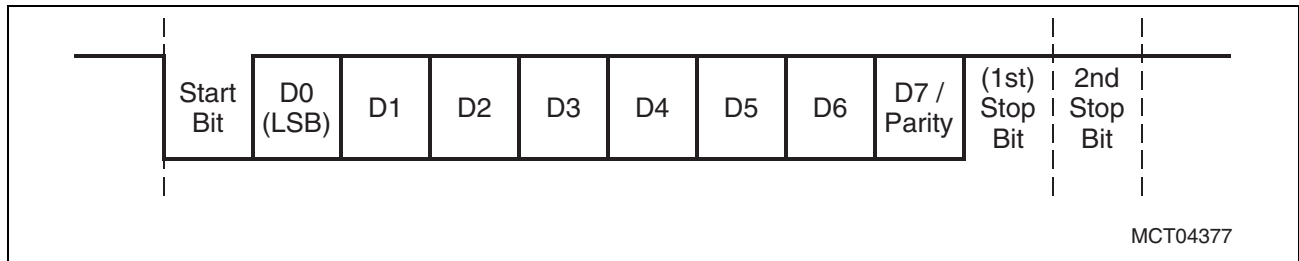
Figure 11-2 Asynchronous Mode of Serial Channel ASC0

Asynchronous Data Frames

**8-bit data frames** either consist of 8 data bits D7 ... D0 (SOM = '001<sub>B</sub>'), or of 7 data bits D6 ... D0 plus an automatically generated parity bit (SOM = '011<sub>B</sub>'). Parity may be odd or even, depending on bit S0ODD in register S0CON. An even parity bit will be set, if the modulo-2-sum of the 7 data bits is '1'. An odd parity bit will be cleared in this case. Parity checking is enabled via bit SOPEN (always OFF in 8-bit data mode). The parity error flag

### The Asynchronous/Synchronous Serial Interface

S0PE will be set along with the error interrupt request flag, if a wrong parity bit is received. The parity bit itself will be stored in bit S0RBUF.7.



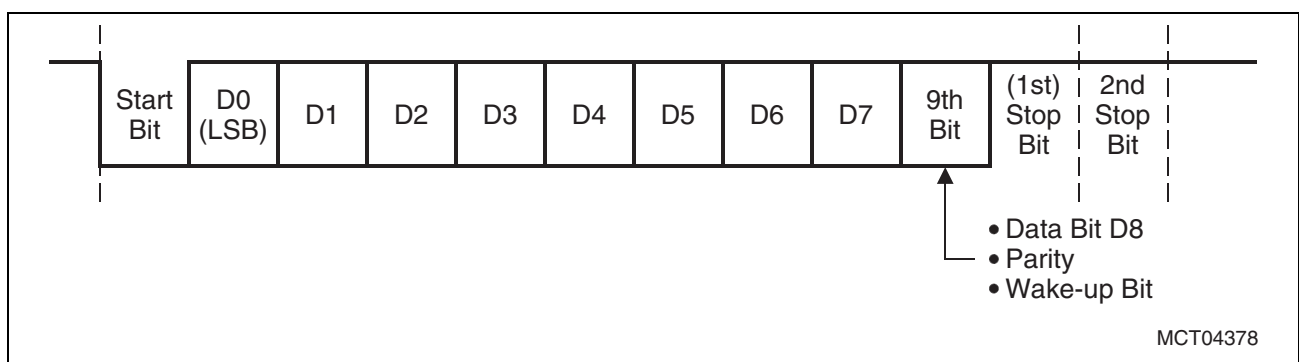
**Figure 11-3 Asynchronous 8-bit Data Frames**

**9-bit data frames** either consist of 9 data bits D8 ... D0 (S0M = '100<sub>B</sub>'), of 8 data bits D7 ... D0 plus an automatically generated parity bit (S0M = '111<sub>B</sub>') or of 8 data bits D7 ... D0 plus wake-up bit (S0M = '101<sub>B</sub>'). Parity may be odd or even, depending on bit S0ODD in register S0CON. An even parity bit will be set, if the modulo-2-sum of the 8 data bits is '1'. An odd parity bit will be cleared in this case. Parity checking is enabled via bit S0PEN (always OFF in 9-bit data and wake-up mode). The parity error flag S0PE will be set along with the error interrupt request flag, if a wrong parity bit is received. The parity bit itself will be stored in bit S0RBUF.8.

In wake-up mode received frames are only transferred to the receive buffer register, if the 9th bit (the wake-up bit) is '1'. If this bit is '0', no receive interrupt request will be activated and no data will be transferred.

This feature may be used to control communication in multi-processor system:

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the additional 9th bit is a '1' for an address byte and a '0' for a data byte, so no slave will be interrupted by a data 'byte'. An address 'byte' will interrupt all slaves (operating in 8-bit data + wake-up bit mode), so each slave can examine the 8 LSBs of the received character (the address). The addressed slave will switch to 9-bit data mode (e.g. by clearing bit S0M.0), which enables it to also receive the data bytes that will be coming (having the wake-up bit cleared). The slaves that were not being addressed remain in 8-bit data + wake-up bit mode, ignoring the following data bytes.



**Figure 11-4 Asynchronous 9-bit Data Frames**



## The Asynchronous/Synchronous Serial Interface

**Asynchronous transmission** begins at the next overflow of the divide-by-16 counter (see [Figure 11-4](#)), provided that S0R is set and data has been loaded into S0TBUF. The transmitted data frame consists of three basic elements:

- the start bit
- the data field (8 or 9 bits, LSB first, including a parity bit, if selected)
- the delimiter (1 or 2 stop bits)

Data transmission is double buffered. When the transmitter is idle, the transmit data loaded into S0TBUF is immediately moved to the transmit shift register thus freeing S0TBUF for the next data to be sent. This is indicated by the transmit buffer interrupt request flag S0TBIR being set. S0TBUF may now be loaded with the next data, while transmission of the previous one is still going on.

The transmit interrupt request flag S0TIR will be set before the last bit of a frame is transmitted, i.e. before the first or the second stop bit is shifted out of the transmit shift register.

The transmitter output pin TXD0 must be configured for alternate data output, i.e. the respective port output latch and the direction latch must be '1'.

**Asynchronous reception** is initiated by a falling edge (1-to-0 transition) on pin RXD0, provided that bits S0R and S0REN are set. The receive data input pin RXD0 is sampled at 16 times the rate of the selected baud rate. A majority decision of the 7th, 8th and 9th sample determines the effective bit value. This avoids erroneous results that may be caused by noise.

If the detected value is not a '0' when the start bit is sampled, the receive circuit is reset and waits for the next 1-to-0 transition at pin RXD0. If the start bit proves valid, the receive circuit continues sampling and shifts the incoming data frame into the receive shift register.

When the last stop bit has been received, the content of the receive shift register is transferred to the receive data buffer register S0RBUF. Simultaneously, the receive interrupt request flag S0RIR is set after the 9th sample in the last stop bit time slot (as programmed), regardless whether valid stop bits have been received or not. The receive circuit then waits for the next start bit (1-to-0 transition) at the receive data input pin.

The receiver input pin RXD0 must be configured for input, i.e. the respective direction latch must be '0'.

Asynchronous reception is stopped by clearing bit S0REN. A currently received frame is completed including the generation of the receive interrupt request and an error interrupt request, if appropriate. Start bits that follow this frame will not be recognized.

*Note: In wake-up mode received frames are only transferred to the receive buffer register, if the 9<sup>th</sup> bit (the wake-up bit) is '1'. If this bit is '0', no receive interrupt request will be activated and no data will be transferred.*

The Asynchronous/Synchronous Serial Interface

### 11.2 Synchronous Operation

Synchronous mode supports half-duplex communication, basically for simple IO expansion via shift registers. Data is transmitted and received via pin RXD0, while pin TXD0 outputs the shift clock. These signals are alternate port functions. Synchronous mode is selected with  $SOM = '000_B'$ .

8 data bits are transmitted or received synchronous to a shift clock generated by the internal baud rate generator. The shift clock is only active as long as data bits are transmitted or received.

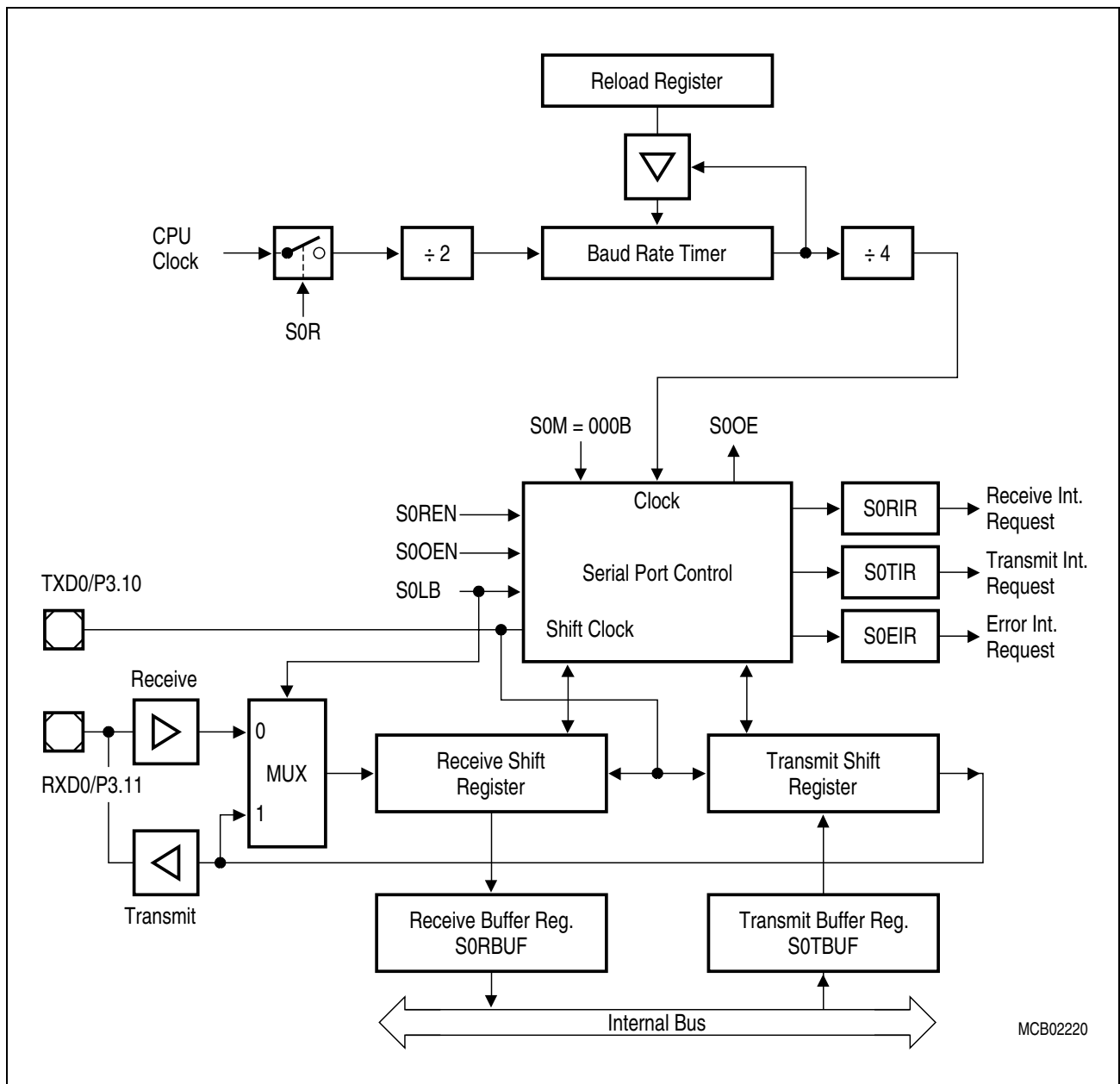


Figure 11-5 Synchronous Mode of Serial Channel ASC0

## The Asynchronous/Synchronous Serial Interface

**Synchronous transmission** begins within 4 state times after data has been loaded into S0TBUF, provided that S0R is set and S0REN = '0' (half-duplex, no reception). Data transmission is double buffered. When the transmitter is idle, the transmit data loaded into S0TBUF is immediately moved to the transmit shift register thus freeing S0TBUF for the next data to be sent. This is indicated by the transmit buffer interrupt request flag S0TBIR being set. S0TBUF may now be loaded with the next data, while transmission of the previous one is still going on. The data bits are transmitted synchronous with the shift clock. After the bit time for the 8th data bit, both pins TXD0 and RXD0 will go high, the transmit interrupt request flag S0TIR is set, and serial data transmission stops.

Pin TXD0 must be configured for alternate data output, i.e. the respective port output latch and the direction latch must be '1', in order to provide the shift clock. Pin RXD0 must also be configured for output (output/direction latch = '1') during transmission.

**Synchronous reception** is initiated by setting bit S0REN = '1'. If bit S0R = '1', the data applied at pin RXD0 are clocked into the receive shift register synchronous to the clock which is output at pin TXD0. After the 8th bit has been shifted in, the content of the receive shift register is transferred to the receive data buffer S0RBUF, the receive interrupt request flag S0RIR is set, the receiver enable bit S0REN is reset, and serial data reception stops.

Pin TXD0 must be configured for alternate data output, i.e. the respective port output latch and the direction latch must be '1', in order to provide the shift clock. Pin RXD0 must be configured as alternate data input, i.e. the respective direction latch must be '0'.

Synchronous reception is stopped by clearing bit S0REN. A currently received byte is completed including the generation of the receive interrupt request and an error interrupt request, if appropriate. Writing to the transmit buffer register while a reception is in progress has no effect on reception and will not start a transmission.

If a previously received byte has not been read out of the receive buffer register at the time the reception of the next byte is complete, both the error interrupt request flag S0EIR and the overrun error status flag S0OE will be set, provided the overrun check has been enabled by bit S0OEN.

---

**The Asynchronous/Synchronous Serial Interface****11.3 Hardware Error Detection Capabilities**

To improve the safety of serial data exchange, the serial channel ASC0 provides an error interrupt request flag, which indicates the presence of an error, and three (selectable) error status flags in register S0CON, which indicate which error has been detected during reception. Upon completion of a reception, the error interrupt request flag S0EIR will be set simultaneously with the receive interrupt request flag S0RIR, if one or more of the following conditions are met:

- If the framing error detection enable bit S0FEN is set and any of the expected stop bits is not high, the framing error flag S0FE is set, indicating that the error interrupt request is due to a framing error (Asynchronous mode only).
- If the parity error detection enable bit S0PEN is set in the modes where a parity bit is received, and the parity check on the received data bits proves false, the parity error flag S0PE is set, indicating that the error interrupt request is due to a parity error (Asynchronous mode only).
- If the overrun error detection enable bit S0OEN is set and the last character received was not read out of the receive buffer by software or PEC transfer at the time the reception of a new frame is complete, the overrun error flag S0OE is set indicating that the error interrupt request is due to an overrun error (Asynchronous and synchronous mode).

## The Asynchronous/Synchronous Serial Interface

### 11.4 ASC0 Baud Rate Generation

The serial channel ASC0 has its own dedicated 13-bit baud rate generator with 13-bit reload capability, allowing baud rate generation independent of the GPT timers.

The baud rate generator is clocked with the CPU clock divided by 2 ( $f_{CPU}/2$ ). The timer is counting downwards and can be started or stopped through the Baud Rate Generator Run Bit S0R in register S0CON. Each underflow of the timer provides one clock pulse to the serial channel. The timer is reloaded with the value stored in its 13-bit reload register each time it underflows. The resulting clock is again divided according to the operating mode and controlled by the Baudrate Selection Bit S0BRS. If S0BRS = '1', the clock signal is additionally divided to 2/3rd of its frequency (see formulas and table). So the baud rate of ASC0 is determined by the CPU clock, the reload value, the value of S0BRS and the operating mode (asynchronous or synchronous).

Register S0BG is the dual-function Baud Rate Generator/Reload register. Reading S0BG returns the content of the timer (bits 15 ... 13 return zero), while writing to S0BG always updates the reload register (bits 15 ... 13 are insignificant).

An auto-reload of the timer with the content of the reload register is performed each time S0BG is written to. However, if S0R = '0' at the time the write operation to S0BG is performed, the timer will not be reloaded until the first instruction cycle after S0R = '1'.

#### Asynchronous Mode Baud Rates

For asynchronous operation, the baud rate generator provides a clock with 16 times the rate of the established baud rate. Every received bit is sampled at the 7th, 8th and 9th cycle of this clock. The baud rate for asynchronous operation of serial channel ASC0 and the required reload value for a given baudrate can be determined by the following formulas:

$$B_{\text{Async}} = \frac{f_{\text{CPU}}}{16 \times (2 + \langle \text{S0BRS} \rangle) \times (\langle \text{S0BRL} \rangle + 1)}$$

$$\text{S0BRL} = \left( \frac{f_{\text{CPU}}}{16 \times (2 + \langle \text{S0BRS} \rangle) \times B_{\text{Async}}} \right) - 1$$

$\langle \text{S0BRL} \rangle$  represents the content of the reload register, taken as unsigned 13-bit integer,  $\langle \text{S0BRS} \rangle$  represents the value of bit S0BRS (i.e. '0' or '1'), taken as integer.

The tables below list various commonly used baud rates together with the required reload values and the deviation errors compared to the intended baudrate for a number of CPU frequencies.

*Note: The deviation errors given in the tables below are rounded. Using a baudrate crystal (e.g. 18.432 MHz) will provide correct baudrates without deviation errors.*

**The Asynchronous/Synchronous Serial Interface**
**Table 11-1 ASC0 Asynchronous Baudrate Generation for  $f_{CPU} = 16$  MHz**

Baud Rate	S0BRS = '0'		S0BRS = '1'	
	Deviation Error	Reload Value	Deviation Error	Reload Value
500 kBaud	±0.0%	0000 <sub>H</sub>	–	–
19.2 kBaud	+0.2%/ -3.5%	0019 <sub>H</sub> /001A <sub>H</sub>	+2.1%/ -3.5%	0010 <sub>H</sub> /0011 <sub>H</sub>
9600 Baud	+0.2%/ -1.7%	0033 <sub>H</sub> /0034 <sub>H</sub>	+2.1%/ -0.8%	0021 <sub>H</sub> /0022 <sub>H</sub>
4800 Baud	+0.2%/ -0.8%	0067 <sub>H</sub> /0068 <sub>H</sub>	+0.6%/ -0.8%	0044 <sub>H</sub> /0045 <sub>H</sub>
2400 Baud	+0.2%/ -0.3%	00CF <sub>H</sub> /00D0 <sub>H</sub>	+0.6%/ -0.1%	0089 <sub>H</sub> /008A <sub>H</sub>
1200 Baud	+0.4%/ -0.1%	019F <sub>H</sub> /01A0 <sub>H</sub>	+0.3%/ -0.1%	0114 <sub>H</sub> /0115 <sub>H</sub>
600 Baud	+0.0%/ -0.1%	0340 <sub>H</sub> /0341 <sub>H</sub>	+0.1%/ -0.1%	022A <sub>H</sub> /022B <sub>H</sub>
61 Baud	+0.1%	1FFF <sub>H</sub>	+0.0%/ -0.0%	115B <sub>H</sub> /115C <sub>H</sub>
40 Baud	–	–	+1.7%	1FFF <sub>H</sub>

**Table 11-2 ASC0 Asynchronous Baudrate Generation for  $f_{CPU} = 20$  MHz**

Baud Rate	S0BRS = '0'		S0BRS = '1'	
	Deviation Error	Reload Value	Deviation Error	Reload Value
625 kBaud	±0.0%	0000 <sub>H</sub>	–	–
19.2 kBaud	+1.7%/ -1.4%	001F <sub>H</sub> /0020 <sub>H</sub>	+3.3%/ -1.4%	0014 <sub>H</sub> /0015 <sub>H</sub>
9600 Baud	+0.2%/ -1.4%	0040 <sub>H</sub> /0041 <sub>H</sub>	+1.0%/ -1.4%	002A <sub>H</sub> /002B <sub>H</sub>
4800 Baud	+0.2%/ -0.6%	0081 <sub>H</sub> /0082 <sub>H</sub>	+1.0%/ -0.2%	0055 <sub>H</sub> /0056 <sub>H</sub>
2400 Baud	+0.2%/ -0.2%	0103 <sub>H</sub> /0104 <sub>H</sub>	+0.4%/ -0.2%	00AC <sub>H</sub> /00AD <sub>H</sub>
1200 Baud	+0.2%/ -0.4%	0207 <sub>H</sub> /0208 <sub>H</sub>	+0.1%/ -0.2%	015A <sub>H</sub> /015B <sub>H</sub>
600 Baud	+0.1%/ -0.0%	0410 <sub>H</sub> /0411 <sub>H</sub>	+0.1%/ -0.1%	02B5 <sub>H</sub> /02B6 <sub>H</sub>
75 Baud	+1.7%	1FFF <sub>H</sub>	+0.0%/ -0.0%	15B2 <sub>H</sub> /15B3 <sub>H</sub>
50 Baud	–	–	+1.7%	1FFF <sub>H</sub>

**The Asynchronous/Synchronous Serial Interface**
**Table 11-3 ASC0 Asynchronous Baudrate Generation for  $f_{CPU} = 25$  MHz**

Baud Rate	S0BRS = '0'		S0BRS = '1'	
	Deviation Error	Reload Value	Deviation Error	Reload Value
781 kBaud	+0.2%	0000 <sub>H</sub>	–	–
19.2 kBaud	+1.7%/ -0.8%	0027 <sub>H</sub> /0028 <sub>H</sub>	+0.5%/ -3.1%	001A <sub>H</sub> /001B <sub>H</sub>
9600 Baud	+0.5%/ -0.8%	0050 <sub>H</sub> /0051 <sub>H</sub>	+0.5%/ -1.4%	0035 <sub>H</sub> /0036 <sub>H</sub>
4800 Baud	+0.5%/ -0.2%	00A1 <sub>H</sub> /00A2 <sub>H</sub>	+0.5%/ -0.5%	006B <sub>H</sub> /006C <sub>H</sub>
2400 Baud	+0.2%/ -0.2%	0145 <sub>H</sub> /0146 <sub>H</sub>	+0.0%/ -0.5%	00D8 <sub>H</sub> /00D9 <sub>H</sub>
1200 Baud	+0.0%/ -0.2%	028A <sub>H</sub> /028B <sub>H</sub>	+0.0%/ -0.2%	01B1 <sub>H</sub> /01B2 <sub>H</sub>
600 Baud	+0.0%/ -0.1%	0515 <sub>H</sub> /0516 <sub>H</sub>	+0.0%/ -0.1%	0363 <sub>H</sub> /0364 <sub>H</sub>
95 Baud	+0.4%	1FFF <sub>H</sub>	+0.0%/ -0.0%	1569 <sub>H</sub> /156A <sub>H</sub>
63 Baud	–	–	+1.0%	1FFF <sub>H</sub>

**Table 11-4 ASC0 Asynchronous Baudrate Generation for  $f_{CPU} = 33$  MHz**

Baud Rate	S0BRS = '0'		S0BRS = '1'	
	Deviation Error	Reload Value	Deviation Error	Reload Value
1.031 MBaud	±0.0%	0000 <sub>H</sub>	–	–
19.2 kBaud	+1.3%/ -0.5%	0034 <sub>H</sub> /0035 <sub>H</sub>	+2.3%/ -0.5%	0022 <sub>H</sub> /0023 <sub>H</sub>
9600 Baud	+0.4%/ -0.5%	006A <sub>H</sub> /006B <sub>H</sub>	+0.9%/ -0.5%	0046 <sub>H</sub> /0047 <sub>H</sub>
4800 Baud	+0.4%/ -0.1%	00D5 <sub>H</sub> /00D6 <sub>H</sub>	+0.2%/ -0.5%	008E <sub>H</sub> /008F <sub>H</sub>
2400 Baud	+0.2%/ -0.1%	01AC <sub>H</sub> /01AD <sub>H</sub>	+0.2%/ -0.2%	011D <sub>H</sub> /011E <sub>H</sub>
1200 Baud	+0.0%/ -0.1%	035A <sub>H</sub> /035B <sub>H</sub>	+0.2%/ -0.0%	023B <sub>H</sub> /023C <sub>H</sub>
600 Baud	+0.0%/ -0.0%	06B5 <sub>H</sub> /06B6 <sub>H</sub>	+0.1%/ -0.0%	0478 <sub>H</sub> /0479 <sub>H</sub>
125 Baud	+7.1%	1FFF <sub>H</sub>	±0.0%	157C <sub>H</sub>
84 Baud	–	–	-0.9%	1FFF <sub>H</sub>

The Asynchronous/Synchronous Serial Interface

**Synchronous Mode Baud Rates**

For synchronous operation, the baud rate generator provides a clock with 4 times the rate of the established baud rate. The baud rate for synchronous operation of serial channel ASC0 can be determined by the following formula:

$$S0BRL = \left( \frac{f_{CPU}}{4 \times (2 + \langle S0BRS \rangle) \times B_{Sync}} \right) - 1$$

$$B_{Sync} = \frac{f_{CPU}}{4 \times (2 + \langle S0BRS \rangle) \times (\langle S0BRL \rangle + 1)}$$

$\langle S0BRL \rangle$  represents the content of the reload register, taken as unsigned 13-bit integers,  $\langle S0BRS \rangle$  represents the value of bit S0BRS (i.e. '0' or '1'), taken as integer.

**Table 11-5** gives the limit baudrates depending on the CPU clock frequency and bit S0BRS.

**Table 11-5 ASC0 Synchronous Baudrate Generation**

CPU clock $f_{CPU}$	S0BRS = '0'		S0BRS = '1'	
	Min. Baudrate	Max. Baudrate	Min. Baudrate	Max. Baudrate
16 MHz	244 Baud	2.000 MBaud	162 Baud	1.333 MBaud
20 MHz	305 Baud	2.500 MBaud	203 Baud	1.666 MBaud
25 MHz	381 Baud	3.125 MBaud	254 Baud	2.083 MBaud
33 MHz	504 Baud	4.125 MBaud	336 Baud	2.750 MBaud



The Asynchronous/Synchronous Serial Interface

### 11.5 ASC0 Interrupt Control

Four bit addressable interrupt control registers are provided for serial channel ASC0. Register S0TIC controls the transmit interrupt, S0TBIC controls the transmit buffer interrupt, S0RIC controls the receive interrupt and S0EIC controls the error interrupt of serial channel ASC0. Each interrupt source also has its own dedicated interrupt vector. S0TINT is the transmit interrupt vector, S0TBINT is the transmit buffer interrupt vector, S0RINT is the receive interrupt vector, and S0EINT is the error interrupt vector.

The cause of an error interrupt request (framing, parity, overrun error) can be identified by the error status flags in control register S0CON.

*Note: In contrast to the error interrupt request flag S0EIR, the error status flags S0FE/S0PE/S0OE are not reset automatically upon entry into the error interrupt service routine, but must be cleared by software.*

#### S0TIC

ASC0 Tx Intr. Ctrl. Reg.

SFR (FF6C<sub>H</sub>/B6<sub>H</sub>)

Reset Value: - - 00<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				-				S0TIR	S0TIE			ILVL			GLVL
-	-	-	-	-	-	-	-	rwh	rw			rw			rw

#### S0TBIC

ASC0 Tx Buf. Intr. Ctrl. Reg.

SFR (FF9C<sub>H</sub>/CE<sub>H</sub>)

Reset Value: - - 00<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				-				S0TBIR	S0TBIE			ILVL			GLVL
-	-	-	-	-	-	-	-	rwh	rw			rw			rw

#### S0RIC

ASC0 Rx Intr. Ctrl. Reg.

SFR (FF6E<sub>H</sub>/B7<sub>H</sub>)

Reset Value: - - 00<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				-				S0RIR	S0RIE			ILVL			GLVL
-	-	-	-	-	-	-	-	rwh	rw			rw			rw

The Asynchronous/Synchronous Serial Interface

S0EIC

ASC0 Error Intr. Ctrl. Reg.

SFR (FF70<sub>H</sub>/B8<sub>H</sub>)

Reset Value: - - 00<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
								S0 EIR		S0 EIE			ILVL			GLVL	
								rwh		rw			rw			rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

Using the ASC0 Interrupts

For normal operation (i.e. besides the error interrupt) the ASC0 provides three interrupt requests to control data exchange via this serial channel:

- S0TBIR is activated when data is moved from S0TBUF to the transmit shift register.
- S0TIR is activated before the last bit of an asynchronous frame is transmitted, or after the last bit of a synchronous frame has been transmitted.
- S0RIR is activated when the received frame is moved to S0RBUF.

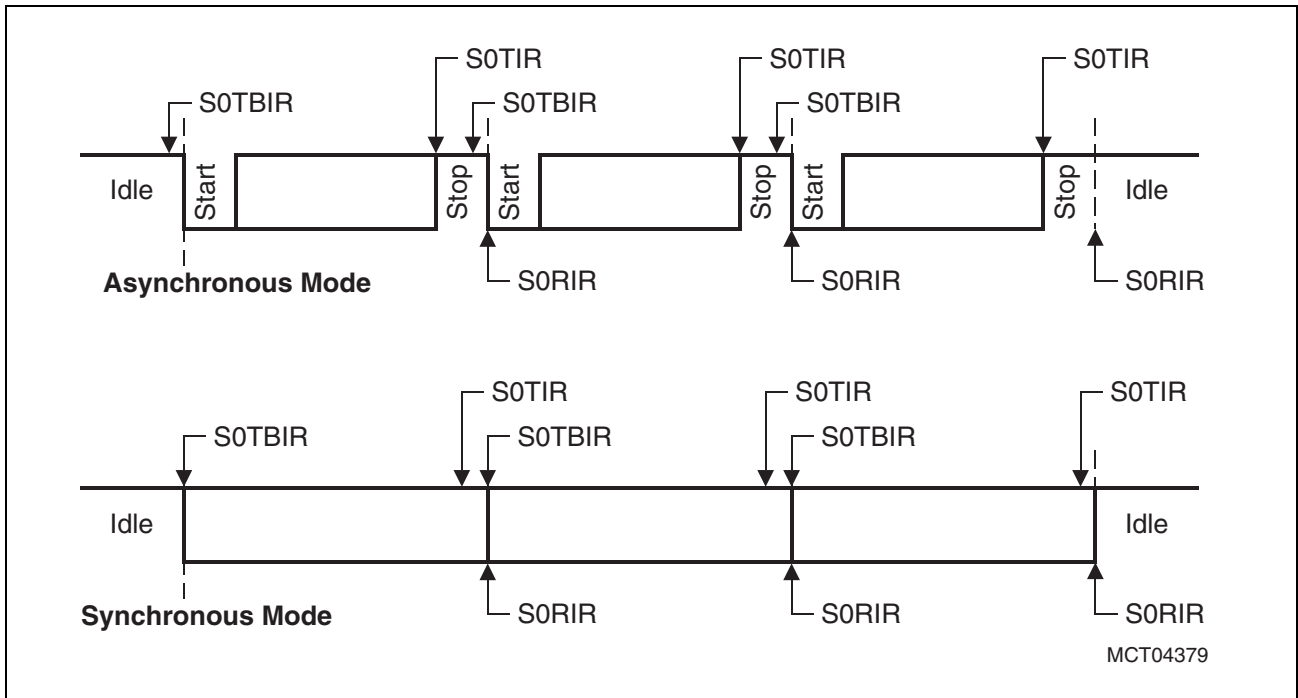
While the task of the receive interrupt handler is quite clear, the transmitter is serviced by two interrupt handlers. This provides advantages for the servicing software.

**For single transfers** is sufficient to use the transmitter interrupt (S0TIR), which indicates that the previously loaded data has been transmitted, except for the last bit of an asynchronous frame.

**For multiple back-to-back transfers** it is necessary to load the following piece of data at last until the time the last bit of the previous frame has been transmitted. In asynchronous mode this leaves just one bit-time for the handler to respond to the transmitter interrupt request, in synchronous mode it is impossible at all.

Using the transmit buffer interrupt (S0TBIR) to reload transmit data gives the time to transmit a complete frame for the service routine, as S0TBUF may be reloaded while the previous data is still being transmitted.

The Asynchronous/Synchronous Serial Interface



**Figure 11-6 ASC0 Interrupt Generation**

As shown in [Figure 11-6](#), S0TBIR is an early trigger for the reload routine, while S0TIR indicates the completed transmission. Software using handshake therefore should rely on S0TIR at the end of a data block to make sure that all data has really been transmitted.

**The Async./Synchronous Serial Interface ASC1**

## 12 The Async./Synchronous Serial Interface ASC1

The Asynchronous/Synchronous Serial Interface ASC1 provides serial communication between the C161CS/JC/JI and other microcontrollers, microprocessors or external peripherals.

The ASC0 supports full-duplex asynchronous communication up to 781 kBaud/ 1.03 MBaud and half-duplex synchronous communication up to 3.1/4.1 MBaud (@ 25/ 33 MHz CPU clock). In synchronous mode, data are transmitted or received synchronous to a shift clock which is generated by the C161CS/JC/JI. In asynchronous mode, 8- or 9-bit data transfer, parity generation, and the number of stop bits can be selected. Parity, framing, and overrun error detection is provided to increase the reliability of data transfers. Transmission and reception of data is double-buffered. For multiprocessor communication, a mechanism to distinguish address from data bytes is included. Testing is supported by a loop-back option. A 13-bit baud rate generator provides the ASC1 with a separate serial clock signal.

*Note: The ASC1 module is an XBUS peripheral and therefore requires bit XPEN in register SYSCON to be set in order to be operable.*

Ports & Direction Control Alternate Functions	Data Registers	Control Registers	Interrupt Control
No port registers involved. Port control is accomplished via register S1CON.	S1BG X	S1CON X	XP4IC (Tx) E
	S1TBUF X		XP5IC (Rx) E
	S1RBUF X		XP6IC (Err) E
S1BG	ASC1 Baud Rate Generator/ Reload Register	XP4IC	ASC1 Transmit Interrupt Control Register
S1TBUF	ASC1 Transmit Buffer Register	XP5IC	ASC1 Receive Interrupt Control Register
S1RBUF	ASC1 Receive Buffer Register (read-only)	XP6IC	ASC1 Error Interrupt Control Register
S1CON	ASC1 Control Register		

MCA04838

**Figure 12-1 SFRs and Port Pins associated with ASC1**

**The Async./Synchronous Serial Interface ASC1**

**ASC1 versus ASC0**

The general operation and the selectable modes and baudrates of the ASC1 module are exactly the same as for the ASC0 module. Also the control bits are on exactly the same locations as in ASC0 registers.

The descriptions for module ASC0 (see separate chapter) apply accordingly also for module ASC1, unless explicitly described differently in the following pages.

Other than ASC0 the ASC1 module is accessed via the on-chip XBUS, which makes it look like an external peripheral. The new interface is the reason for three changes for the ASC1 operation:

- XBUS supports no bit operations, control software must use standard operations for the ASC1 control/data registers (the interrupt control registers provide bit addressing)
- ASC1 is connected to X-Peripheral interrupt nodes
- Port control is accomplished via module-internal control bits

**The ASC1 Register Set**

Module ASC1 uses the same set of registers as module ASC0 to interface to the application. These registers, however, are accessed via the XBUS. The table below summarizes the ASC1 registers and their locations.

**Table 12-1 ASC1 Registers**

<b>Register</b>	<b>Description</b>	<b>Location</b>	<b>Notes</b>
<b>S1CON</b>	ASC1 control register	EDA6 <sub>H</sub>	Control port operation
<b>S1BG</b>	ASC1 baudrate generator register	EDA4 <sub>H</sub>	
<b>S1RBUF</b>	ASC1 receive buffer register	EDA2 <sub>H</sub>	
<b>S1TBUF</b>	ASC1 transmit buffer register	EDA0 <sub>H</sub>	

The operating mode of the serial channel ASC1 is controlled by its control register S1CON. This register contains control bits for mode and error check selection, and status flags for error identification.

Also port control is accomplished via control bits within register S1CON.

**The Async./Synchronous Serial Interface ASC1**
**S1CON**
**ASC1 Control Register**
**XReg (EDA6<sub>H</sub>)**
**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>S1R</b>	<b>S1LB</b>	<b>S1BRS</b>	<b>S1ODD</b>	<b>S1DIR</b>	<b>S1OE</b>	<b>S1FE</b>	<b>S1PE</b>	<b>S1OEN</b>	<b>S1FEN</b>	<b>S1PEN</b>	<b>S1REN</b>	<b>S1STP</b>	<b>S1M</b>		
rw	rw	rw	rw	rw	rwh	rwh	rwh	rw	rw	rw	rwh	rw	rw		

Bit	Function
<b>S1M</b>	<b>ASC1 Mode Control</b> 000: 8-bit data synchronous operation 001: 8-bit data async. operation 010: Reserved. Do not use this combination! 011: 7-bit data + parity async. operation 100: 9-bit data async. operation 101: 8-bit data + wake up bit async. operation 110: Reserved. Do not use this combination! 111: 8-bit data + parity async. operation
<b>S1STP</b>	<b>Number of Stop Bits Selection</b> async. operation 0: One stop bit 1: Two stop bits
<b>S1REN</b>	<b>Receiver Enable Bit</b> 0: Receiver disabled 1: Receiver enabled (Cleared by hardware after reception of byte in synchr. mode)
<b>S1PEN</b>	<b>Parity Check Enable Bit</b> async. operation 0: Ignore parity 1: Check parity
<b>S1FEN</b>	<b>Framing Check Enable Bit</b> async. operation 0: Ignore framing errors 1: Check framing errors
<b>S1OEN</b>	<b>Overrun Check Enable Bit</b> 0: Ignore overrun errors 1: Check overrun errors
<b>S1PE</b>	<b>Parity Error Flag</b> Set by hardware on a parity error (S1PEN = '1'). Must be cleared by software.
<b>S1FE</b>	<b>Framing Error Flag</b> Set by hardware on a framing error (S1FEN = '1'). Must be cleared by software.

**The Async./Synchronous Serial Interface ASC1**

<b>Bit</b>	<b>Function</b>
<b>S1OE</b>	<b>Overrun Error Flag</b> Set by hardware on an overrun error (S1OEN = '1'). Must be cleared by software.
<b>S1DIR</b>	<b>Direction Control for Pin RxD1</b> (valid if S1R = '1') 0: Pin RxD1 is input (asynchronous mode, synchronous reception) 1: Pin RxD1 is output (synchronous transmission)
<b>S1ODD</b>	<b>Parity Selection Bit</b> 0: Even parity (parity bit set on odd number of '1's in data) 1: Odd parity (parity bit set on even number of '1's in data)
<b>S1BRS</b>	<b>Baudrate Selection Bit</b> 0: Divide clock by reload-value + constant (depending on mode) 1: Additionally reduce serial clock to 2/3rd
<b>S1LB</b>	<b>LoopBack Mode Enable Bit</b> 0: Standard transmit/receive mode 1: Loopback mode enabled
<b>S1R</b>	<b>Baudrate Generator Run Bit</b> 0: Baudrate generator disabled (ASC1 inactive, pins RXD1/TXD1 not influenced) 1: Baudrate generator enabled (pin TXD1 switched to output, pin RXD1 controlled by bit S1DIR)

A transmission is started by writing to the Transmit Buffer register S1TBUF (via an instruction or a PEC data transfer), data reception is enabled by the Receiver Enable Bit S1REN.

General handling and error detection work in exactly the same way as in the ASC0 module.

**The Loop-Back option** (selected by bit S1LB) allows the data currently being transmitted to be received simultaneously in the receive buffer. This may be used to test serial communication routines at an early stage without having to provide an external network. In loop-back mode the RXD1 input is not used.

*Note: Serial data transmission or reception is only possible when the Baud Rate Generator Run Bit S1R is set to '1'. Otherwise the serial interface is idle.*

*Do not program the mode control field S1M in register S1CON to one of the reserved combinations to avoid unpredictable behavior of the serial interface*

---

## The Async./Synchronous Serial Interface ASC1

### 12.1 Asynchronous Operation

Asynchronous mode supports full-duplex communication, where both transmitter and receiver use the same data frame format and the same baud rate. Data is transmitted on pin TXD1 and received on pin RXD1. These signals are alternate functions of Port 3 pins.

Frame selection and operation is exactly the same as in the ASC0 module.

**Asynchronous transmission** requires pin TXD1 (transmit data) to be configured as output.

This is done automatically by enabling ASC1 (i.e. S1R = '1').

**Asynchronous reception** requires pin RXD1 (receive data) to be configured as input. This is done by clearing the direction control bit (i.e. S1DIR = '0').

### 12.2 Synchronous Operation

Synchronous mode supports half-duplex communication, basically for simple IO expansion via shift registers. Data is transmitted and received via pin RXD1, while pin TXD1 outputs the shift clock. These signals are alternate functions of Port 3 pins. Synchronous mode is selected with S1M = '000<sub>B</sub>'.

The operation is exactly the same as in the ASC0 module. The pin direction control, however, is handled in a different way (description below).

**Synchronous transmission** requires both pins TXD1 (shift clock) and RXD1 (transmit data) to be configured as output.

**Pin TXD1 is configured as output** simply by enabling ASC1 (i.e. S1R = '1').

**Pin RXD1 is configured as output** by setting the direction control bit (i.e. S1DIR = '1').

**Synchronous reception** requires pin TXD1 (shift clock) to be configured as output, and pin RXD1 (receive data) to be configured as input.

**Pin TXD1 is configured as output** simply by enabling ASC1 (i.e. S1R = '1').

**Pin RXD1 is configured as input** by clearing the direction control bit (i.e. S1DIR = '0')



## The Async./Synchronous Serial Interface ASC1

### 12.3 Hardware Error Detection Capabilities

To improve the safety of serial data exchange, the serial channel ASC1 provides an error interrupt, which indicates the presence of an error, and three (selectable) error status flags in register S1CON, which indicate which error has been detected during reception.

As in the ASC0 module three errors can be detected:

Framing error S1FE (enabled by S1FEN), parity error S1PE (enabled by S1PEN), and overrun error S1OE (enabled by S1OEN).

### 12.4 ASC1 Baud Rate Generation

The serial channel ASC1 has its own dedicated 13-bit baud rate generator with 13-bit reload capability, allowing baud rate generation independent of the GPT timers.

The serial channel ASC1 provides the same baudrate generation mechanism for asynchronous mode and for synchronous mode as module ASC0, controlled by the Baud Rate Generator/Reload register S1BG.

Formulas and tables to determine the serial baudrate and the corresponding control values can be found in the ASC0 description.

### 12.5 ASC1 Port Control

The port pins to which module ASC1 is connected (RXD1, TXD1) are controlled by the module itself as long as it is active.

**Pin TXD1** is switched to output and driven with ASC1's transmit signal as long as the module is active, i.e. bit S1R = '1'.

**Pin RXD1** is switched to output and driven with ASC1's transmit signal when ...

- the module is active (S1R = '1') *and*
- synchronous mode is selected (S1M = '000<sub>B</sub>') *and*
- the direction control bit S1DIR = '1'.

In all other cases these pins are not controlled by ASC1 but rather by the standard general purpose IO mechanism. Therefore it is important to provide suitable default modes for pins RXD1 and TXD1 for those cases when ASC1 is disabled. The default mode for TXD1 after reset would e.g. be input, so the transmit would begin floating as soon as ASC1 is disabled. This can be avoided by setting the respective port latch and switching the pin to output. TXD1 then would drive an active high level, which is the inactive state for the serial line.

The Async./Synchronous Serial Interface ASC1

## 12.6 ASC1 Interrupt Control

Three bit addressable interrupt control registers are provided for serial channel ASC1. Register XP4IC controls the transmit interrupt, XP5IC controls the receive interrupt, and XP6IC controls the error interrupt of serial channel ASC1. Each interrupt source also has its own dedicated interrupt vector. XP4INT is the transmit interrupt vector, XP5INT is the receive interrupt vector, and XP6INT is the error interrupt vector.

The cause of an error interrupt request (framing, parity, overrun error) can be identified by the error status flags in control register S1CON.

*Note: In contrast to the error interrupt request flag XP6IR, the error status flags S1FE/S1PE/S1OE are not reset automatically upon entry into the error interrupt service routine, but must be cleared by software.*

### XP4IC

ASC1 Tx Intr. Ctrl. Reg.                      ESFR (F182<sub>H</sub>/C1<sub>H</sub>)                      Reset Value: - - 00<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				-				XP4 IR	XP4 IE			ILVL			GLVL
-	-	-	-	-	-	-	-	rwh	rw			rw			rw

### XP5IC

ASC1 Rx Intr. Ctrl. Reg.                      ESFR (F18A<sub>H</sub>/C5<sub>H</sub>)                      Reset Value: - - 00<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				-				XP5 IR	XP5 IE			ILVL			GLVL
-	-	-	-	-	-	-	-	rwh	rw			rw			rw

### XP6IC

ASC1 Error Intr. Ctrl. Reg.                      ESFR (F192<sub>H</sub>/C9<sub>H</sub>)                      Reset Value: - - 00<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				-				XP6 IR	XP6 IE			ILVL			GLVL
-	-	-	-	-	-	-	-	rwh	rw			rw			rw

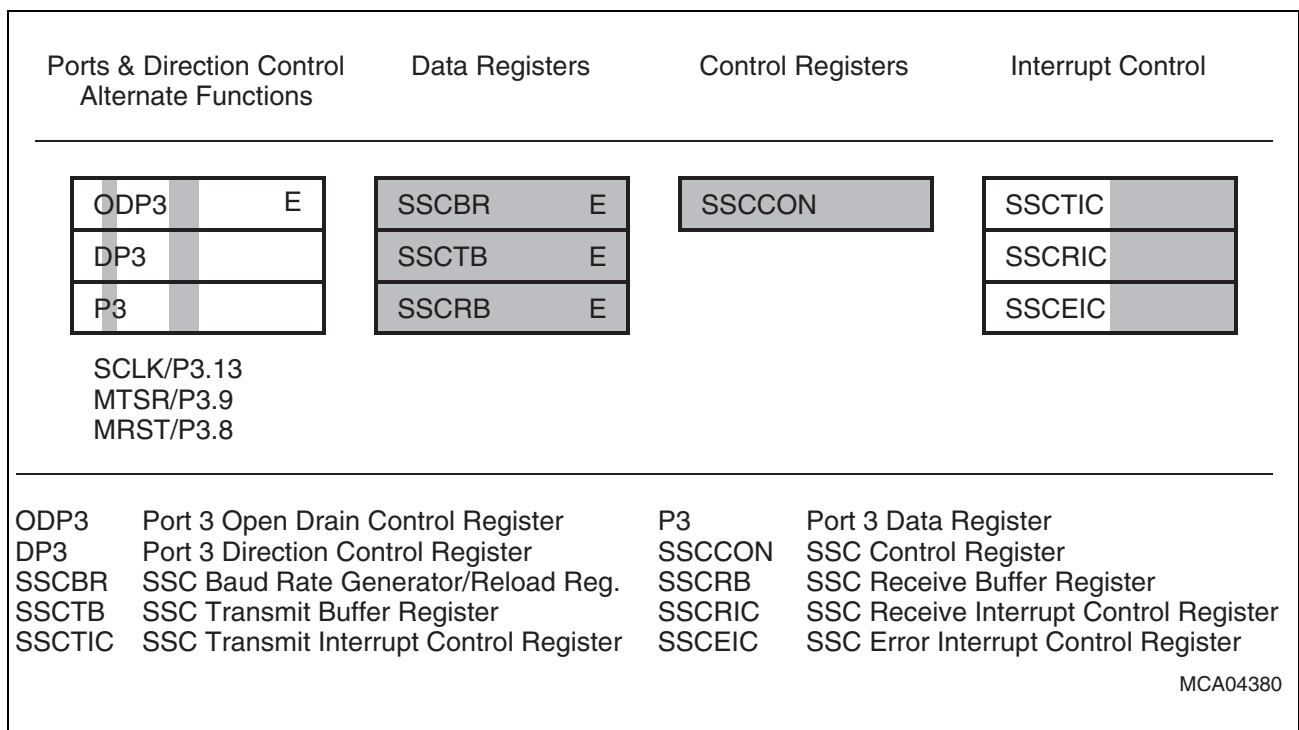
*Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.*

### 13 The High-Speed Synchronous Serial Interface

The high-speed Synchronous Serial Interface SSC provides flexible high-speed serial communication between the C161CS/JC/JI and other microcontrollers, microprocessors or external peripherals.

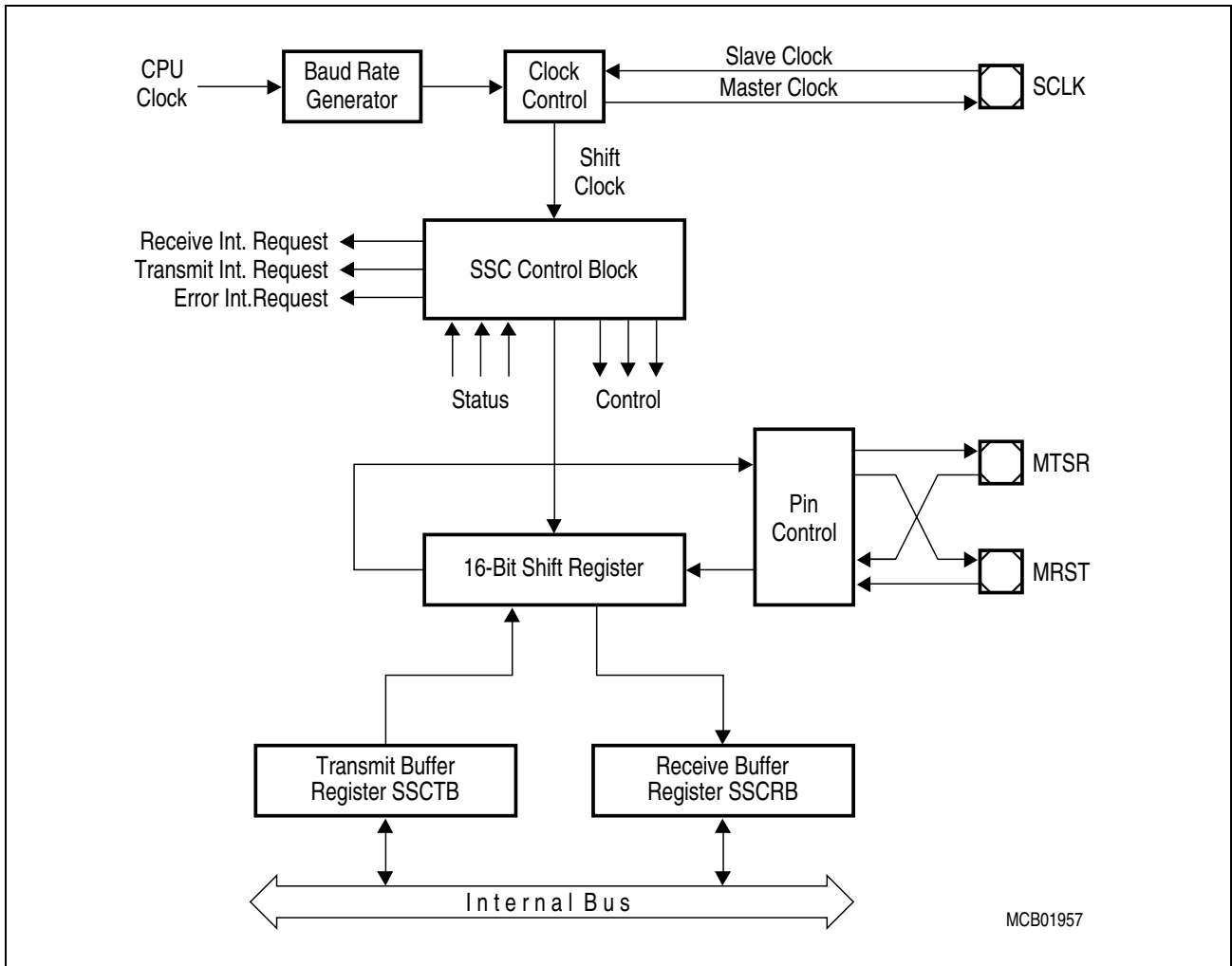
The SSC supports full-duplex and half-duplex synchronous communication up to 6.25/8.25 MBaud (@ 25/33 MHz CPU clock). The serial clock signal can be generated by the SSC itself (master mode) or be received from an external master (slave mode). Data width, shift direction, clock polarity and phase are programmable. This allows communication with SPI-compatible devices. Transmission and reception of data is double-buffered. A 16-bit baud rate generator provides the SSC with a separate serial clock signal.

The high-speed synchronous serial interface can be configured in a very flexible way, so it can be used with other synchronous serial interfaces (e.g. the ASC0 in synchronous mode), serve for master/slave or multimaster interconnections or operate compatible with the popular SPI interface. So it can be used to communicate with shift registers (IO expansion), peripherals (e.g. EEPROMs etc.) or other controllers (networking). The SSC supports half-duplex and full-duplex communication. Data is transmitted or received on pins MTSR/P3.9 (Master Transmit/Slave Receive) and MRST/P3.8 (Master Receive/Slave Transmit). The clock signal is output or input on pin SCLK/P3.13. These pins are alternate functions of Port 3 pins.



**Figure 13-1 SFRs and Port Pins Associated with the SSC**

**The High-Speed Synchronous Serial Interface**



**Figure 13-2 Synchronous Serial Channel SSC Block Diagram**

The operating mode of the serial channel SSC is controlled by its bit-addressable control register SSCCON. This register serves for two purposes:

- during programming (SSC disabled by SSCEN = '0') it provides access to a set of control bits,
- during operation (SSC enabled by SSCEN = '1') it provides access to a set of status flags.

Register SSCCON is shown below in each of the two modes.

**SSCCON**

**SSC Control Reg. (Pr.M.)**

**SFR (FFB2<sub>H</sub>/D9<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SSC EN = 0	SSC MS	-	SSC AR EN	SSC BEN	SSC PEN	SSC REN	SSC TEN	-	SSC PO	SSC PH	SSC HB	SSCBM			
rw	rw	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw			

The High-Speed Synchronous Serial Interface

Bit	Function (Programming Mode, SSCEN = '0')
<b>SSCBM</b>	<b>SSC Data Width Selection</b> 0: Reserved. Do not use this combination 1...15: Transfer Data Width is 2 ... 16 bit (<SSCBM> + 1)
<b>SSCHB</b>	<b>SSC Heading Control Bit</b> 0: Transmit/Receive LSB First 1: Transmit/Receive MSB First
<b>SSCPH</b>	<b>SSC Clock Phase Control Bit</b> 0: Shift transmit data on the leading clock edge, latch on trailing edge 1: Latch receive data on leading clock edge, shift on trailing edge
<b>SSCPO</b>	<b>SSC Clock Polarity Control Bit</b> 0: Idle clock line is low, leading clock edge is low-to-high transition 1: Idle clock line is high, leading clock edge is high-to-low transition
<b>SSCTEN</b>	<b>SSC Transmit Error Enable Bit</b> 0: Ignore transmit errors 1: Check transmit errors
<b>SSCREN</b>	<b>SSC Receive Error Enable Bit</b> 0: Ignore receive errors 1: Check receive errors
<b>SSCPEN</b>	<b>SSC Phase Error Enable Bit</b> 0: Ignore phase errors 1: Check phase errors
<b>SSCBEN</b>	<b>SSC Baudrate Error Enable Bit</b> 0: Ignore baudrate errors 1: Check baudrate errors
<b>SSCAREN</b>	<b>SSC Automatic Reset Enable Bit</b> 0: No additional action upon a baudrate error 1: The SSC is automatically reset upon a baudrate error
<b>SSCMS</b>	<b>SSC Master Select Bit</b> 0: Slave Mode. Operate on shift clock received via SCLK 1: Master Mode. Generate shift clock and output it via SCLK
<b>SSCEN</b>	<b>SSC Enable Bit = '0'</b> Transmission and reception disabled. Access to control bits

The High-Speed Synchronous Serial Interface

**SSCCON**

SSC Control Reg. (Op.M.)

SFR (FFB2<sub>H</sub>/D9<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SSC EN = 1	SSC MS	-	SSC BSY	SSC BE	SSC PE	SSC RE	SSC TE	-	-	-	-	SSCBC			
rw	rw	-	rw	rw	rw	rw	rw	-	-	-	-	r			

Bit	Function (Operating Mode, SSCEN = '1')
SSCBC	<b>SSC Bit Count Field</b> Shift counter is updated with every shifted bit. <b>Do not write to!!!</b>
SSCTE	<b>SSC Transmit Error Flag</b> 1: Transfer starts with the slave's transmit buffer not being updated.
SSCRE	<b>SSC Receive Error Flag</b> 1: Reception completed before the receive buffer was read.
SSCPE	<b>SSC Phase Error Flag</b> 1: Received data changes around sampling clock edge.
SSCBE	<b>SSC Baudrate Error Flag</b> 1: More than factor 2 or less than factor 0.5 between Slave's actual and expected baudrate.
SSCBSY	<b>SSC Busy Flag</b> Set while a transfer is in progress. <b>Do not write to!!!</b>
SSCMS	<b>SSC Master Select Bit</b> 0: Slave Mode. Operate on shift clock received via SCLK. 1: Master Mode. Generate shift clock and output it via SCLK.
SSCEN	<b>SSC Enable Bit = '1'</b> Transmission and reception enabled. Access to status flags and M/S control.

*Note: The target of an access to SSCCON (control bits or flags) is determined by the state of SSCEN prior to the access, i.e. writing C057<sub>H</sub> to SSCCON in programming mode (SSCEN = '0') will initialize the SSC (SSCEN was '0') and then turn it on (SSCEN = '1').*

*When writing to SSCCON, make sure that reserved locations receive zeros.*

## The High-Speed Synchronous Serial Interface

The shift register of the SSC is connected to both the transmit pin and the receive pin via the pin control logic (see block diagram). Transmission and reception of serial data is synchronized and takes place at the same time, i.e. the number of transmitted bits is also received.

The major steps of the state machine of the SSC are controlled by the shift clock signal (see [Figure 13-2](#)).

**In master mode** (SSCMS = '1') two clocks per bit-time are generated, each upon an underflow of the baudrate counter.

**In slave mode** (SSCMS = '0') one clock per bit-time is generated, when the latching edge of the external SCLK signal has been detected.

Transmit data is written into the transmit buffer SSCTB. When the contents of the buffer are moved to the shift register (immediately if no transfer is currently active) a transmit interrupt request (SSCTIR) is generated indicating that SSCTB may be reloaded again.

**The busy flag SSCBSY is set** when the transfer starts (with the next following shift clock in master mode, immediately in slave mode).

*Note: If no data is written to SSCTB prior to a slave transfer, this transfer starts after the first latching edge of the external SCLK signal is detected. No transmit interrupt is generated in this case.*

When the contents of the shift register are moved to the receive buffer SSCRIB after the programmed number of bits (2 ... 16) have been transferred, i.e. after the last latching edge of the current transfer, a receive interrupt request (SSCRIR) is generated.

**The busy flag SSCBSY is cleared** at the end of the current transfer (with the next following shift clock in master mode, immediately in slave mode).

When the transmit buffer is not empty at that time (in the case of continuous transfers) the busy flag is not cleared and the transfer goes on after moving data from the buffer to the shift register.

Software should not modify SSCBSY, as this flag is hardware controlled.

*Note: Only one SSC (etc.) can be master at a given time.*

The transfer of serial data bits can be programmed in many respects:

- the data width can be chosen from 2 bits to 16 bits
- transfer may start with the LSB or the MSB
- the shift clock may be idle low or idle high
- data bits may be shifted with the leading or trailing edge of the clock signal
- the baudrate may be set within a wide range (see baudrate generation)
- the shift clock can be generated (master) or received (slave)

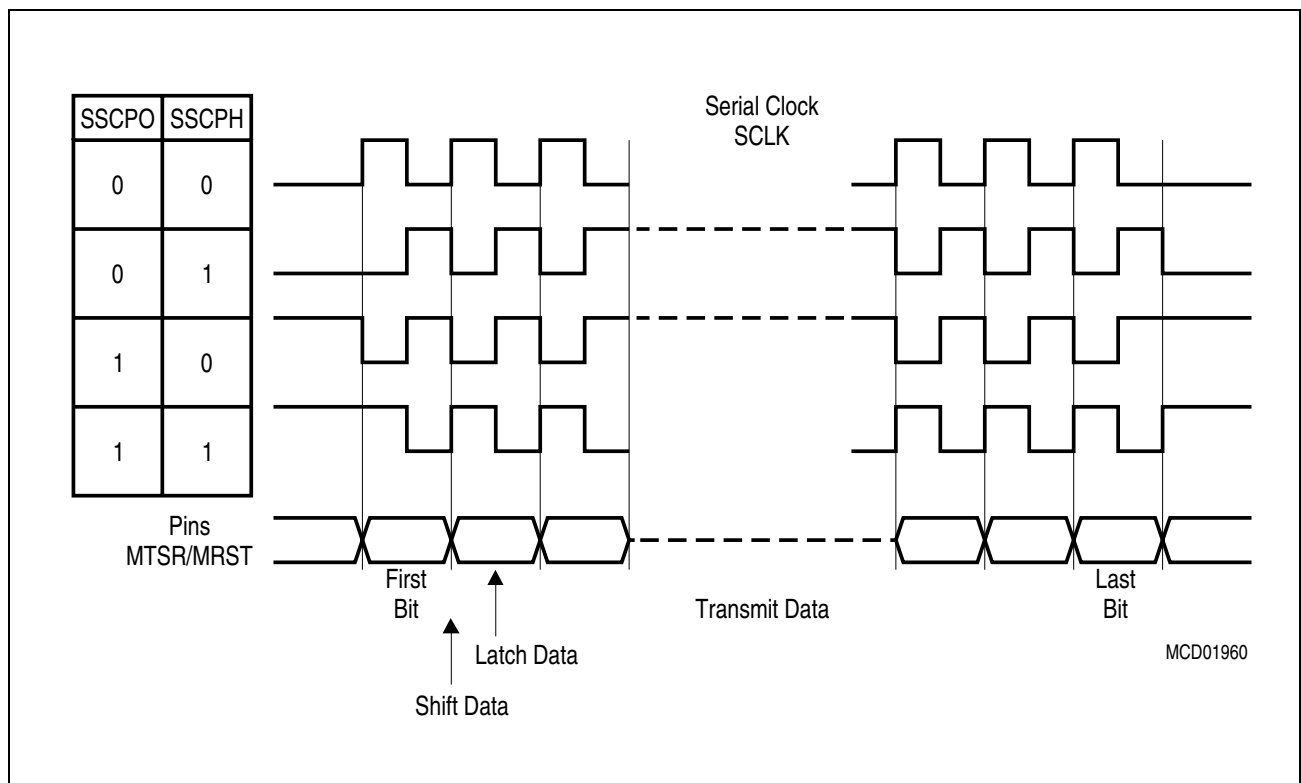
This allows the adaptation of the SSC to a wide range of applications, where serial data transfer is required.

## The High-Speed Synchronous Serial Interface

**The Data Width Selection** supports the transfer of frames of any length, from 2-bit “characters” up to 16-bit “characters”. Starting with the LSB (SSCHB = ‘0’) allows communication e.g. with ASC0 devices in synchronous mode (C166 Family) or 8051 like serial interfaces. Starting with the MSB (SSCHB = ‘1’) allows operation compatible with the SPI interface.

Regardless which data width is selected and whether the MSB or the LSB is transmitted first, the transfer data is always right aligned in registers SSCTB and SSCRb, with the LSB of the transfer data in bit 0 of these registers. The data bits are rearranged for transfer by the internal shift register logic. The unselected bits of SSCTB are ignored, the unselected bits of SSCRb will be not valid and should be ignored by the receiver service routine.

**The Clock Control** allows the adaptation of transmit and receive behavior of the SSC to a variety of serial interfaces. A specific clock edge (rising or falling) is used to shift out transmit data, while the other clock edge is used to latch in receive data. Bit SSCPH selects the leading edge or the trailing edge for each function. Bit SSCPO selects the level of the clock line in the idle state. So for an idle-high clock the leading edge is a falling one, a 1-to-0 transition. **Figure 13-3** is a summary.



**Figure 13-3 Serial Clock Phase and Polarity Options**



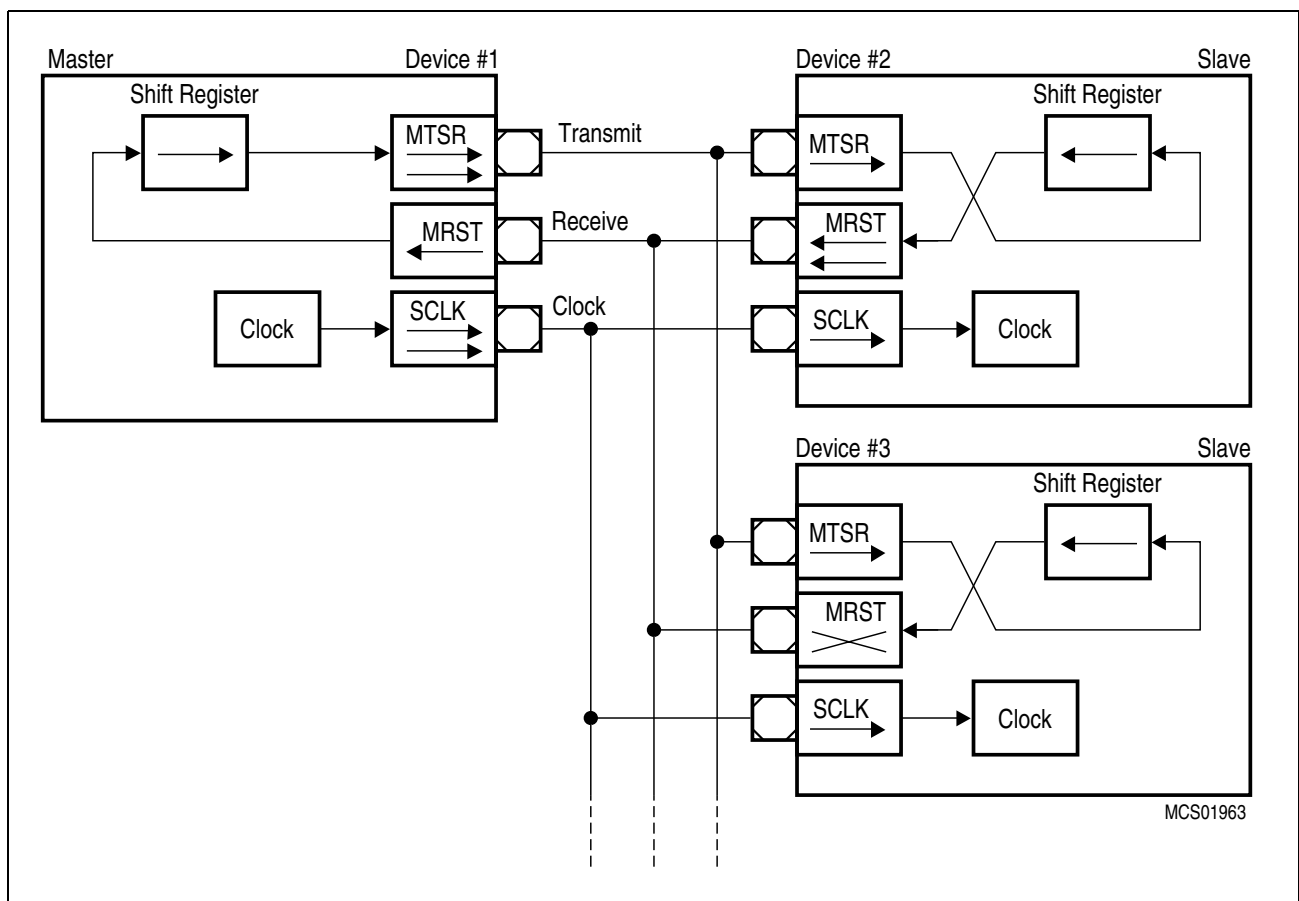
## The High-Speed Synchronous Serial Interface

### 13.1 Full-Duplex Operation

The different devices are connected through three lines. The definition of these lines is always determined by the master: The line connected to the master's data output pin MTSR is the transmit line, the receive line is connected to its data input line MRST, and the clock line is connected to pin SCLK. Only the device selected for master operation generates and outputs the serial clock on pin SCLK. All slaves receive this clock, so their pin SCLK must be switched to input mode (DP3.13 = '0'). The output of the master's shift register is connected to the external transmit line, which in turn is connected to the slaves' shift register input. The output of the slaves' shift register is connected to the external receive line in order to enable the master to receive the data shifted out of the slave. The external connections are hard-wired, the function and direction of these pins is determined by the master or slave operation of the individual device.

*Note: The shift direction shown in [Figure 13-4](#) applies for MSB-first operation as well as for LSB-first operation.*

When initializing the devices in this configuration, select one device for master operation (SSCMS = '1'), all others must be programmed for slave operation (SSCMS = '0'). Initialization includes the operating mode of the device's SSC and also the function of the respective port lines (see [Chapter 13.4](#)).



**Figure 13-4 SSC Full-Duplex Configuration**

## The High-Speed Synchronous Serial Interface

The data output pins MRST of all slave devices are connected together onto the one receive line in this configuration. During a transfer each slave shifts out data from its shift register. There are two ways to avoid collisions on the receive line due to different slave data:

**Only one slave drives the line**, i.e. enables the driver of its MRST pin. All the other slaves have to program their MRST pins to input. So only one slave can put its data onto the master's receive line. Only receiving of data from the master is possible. The master selects the slave device from which it expects data either by separate select lines, or by sending a special command to this slave. The selected slave then switches its MRST line to output, until it gets a deselection signal or command.

**The slaves use open drain output on MRST.** This forms a Wired-AND connection. The receive line needs an external pullup in this case. Corruption of the data on the receive line sent by the selected slave is avoided, when all slaves which are not selected for transmission to the master only send ones ('1'). Since this high level is not actively driven onto the line, but only held through the pullup device, the selected slave can pull this line actively to a low level when transmitting a zero bit. The master selects the slave device from which it expects data either by separate select lines, or by sending a special command to this slave.

After performing all necessary initializations of the SSC, the serial interfaces can be enabled. For a master device, the alternate clock line will now go to its programmed polarity. The alternate data line will go to either '0' or '1', until the first transfer will start.

When the serial interfaces are enabled, the master device can initiate the first data transfer by writing the transmit data into register SSCTB. This value is copied into the shift register (which is assumed to be empty at this time), and the selected first bit of the transmit data will be placed onto the MTSR line on the next clock from the baudrate generator (transmission only starts, if SSCEN = '1'). Depending on the selected clock phase, also a clock pulse will be generated on the SCLK line. With the opposite clock edge the master at the same time latches and shifts in the data detected at its input line MRST. This "exchanges" the transmit data with the receive data. Since the clock line is connected to all slaves, their shift registers will be shifted synchronously with the master's shift register, shifting out the data contained in the registers, and shifting in the data detected at the input line. After the preprogrammed number of clock pulses (via the data width selection) the data transmitted by the master is contained in all slaves' shift registers, while the master's shift register holds the data of the selected slave. In the master and all slaves the content of the shift register is copied into the receive buffer SSCRIB and the receive interrupt flag SSCRIR is set.

A slave device will immediately output the selected first bit (MSB or LSB of the transfer data) at pin MRST, when the content of the transmit buffer is copied into the slave's shift register. It will not wait for the next clock from the baudrate generator, as the master does. The reason for this is that, depending on the selected clock phase, the first clock

---

## The High-Speed Synchronous Serial Interface

edge generated by the master may be already used to clock in the first data bit. So the slave's first data bit must already be valid at this time.

*Note: On the SSC always a transmission **and** a reception takes place at the same time, regardless whether valid data has been transmitted or received. This is different e.g. from asynchronous reception on ASC0.*

**The initialization of the SCLK pin** on the master requires some attention in order to avoid undesired clock transitions, which may disturb the other receivers. The state of the internal alternate output lines is '1' as long as the SSC is disabled. This alternate output signal is ANDed with the respective port line output latch. Enabling the SSC with an idle-low clock (SSCPO = '0') will drive the alternate data output and (via the AND) the port pin SCLK immediately low. To avoid this, use the following sequence:

- select the clock idle level (SSCPO = 'x')
- load the port output latch with the desired clock idle level (P3.13 = 'x')
- switch the pin to output (DP3.13 = '1')
- enable the SSC (SSCEN = '1')
- if SSCPO = '0': enable alternate data output (P3.13 = '1')

The same mechanism as for selecting a slave for transmission (separate select lines or special commands) may also be used to move the role of the master to another device in the network. In this case the previous master and the future master (previous slave) will have to toggle their operating mode (SSCMS) and the direction of their port pins (see description above).

## The High-Speed Synchronous Serial Interface

### 13.2 Half-Duplex Operation

In a half duplex configuration only one data line is necessary for both receiving **and** transmitting of data. The data exchange line is connected to both pins MTSR and MRST of each device, the clock line is connected to the SCLK pin.

The master device controls the data transfer by generating the shift clock, while the slave devices receive it. Due to the fact that all transmit and receive pins are connected to the one data exchange line, serial data may be moved between arbitrary stations.

Similar to full duplex mode there are **two ways to avoid collisions** on the data exchange line:

- only the transmitting device may enable its transmit pin driver
- the non-transmitting devices use open drain output and only send ones

Since the data inputs and outputs are connected together, a transmitting device will clock in its own data at the input pin (MRST for a master device, MTSR for a slave). By these means any corruptions on the common data exchange line are detected, where the received data is not equal to the transmitted data.

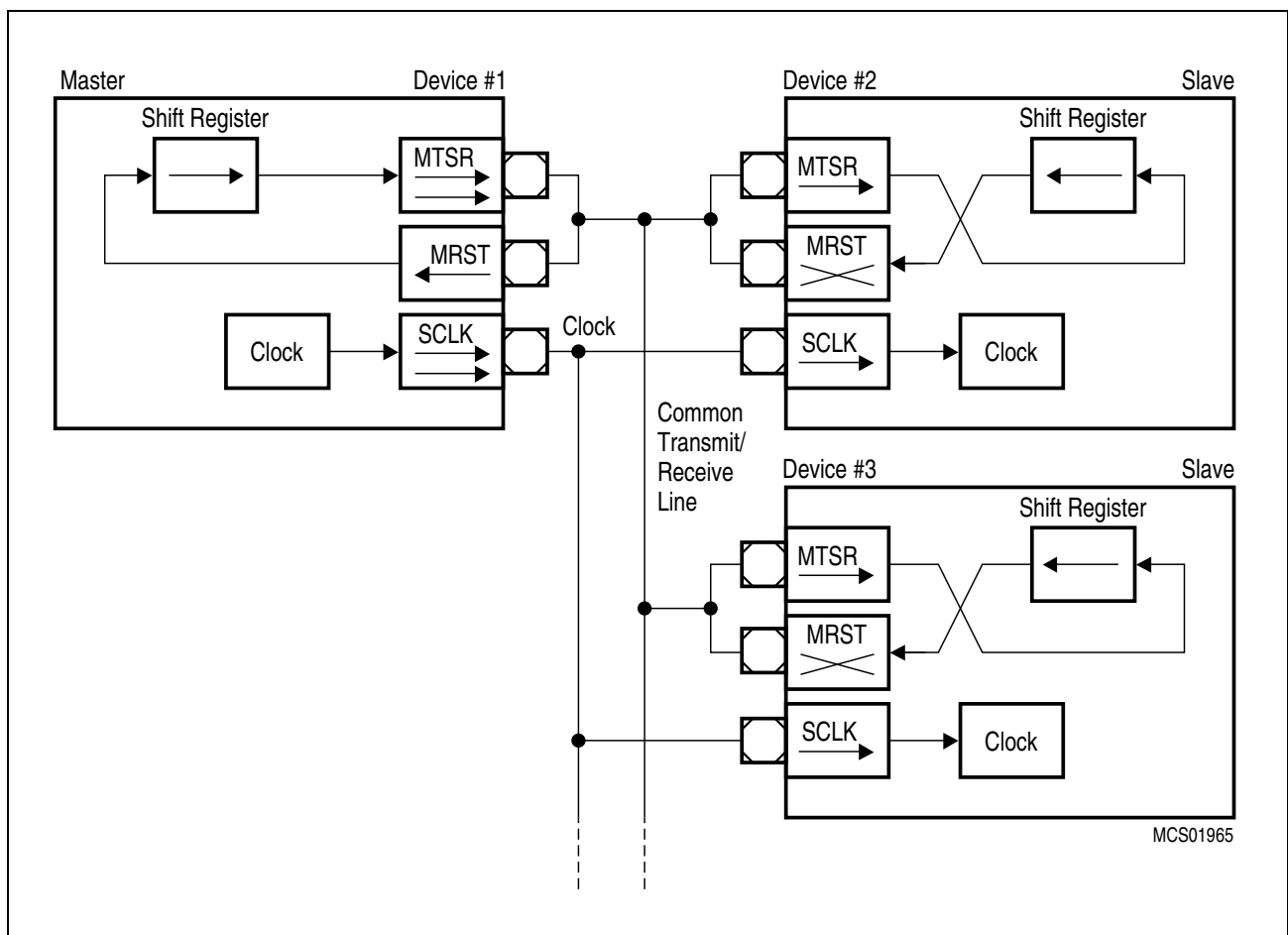


Figure 13-5 SSC Half-Duplex Configuration

---

## The High-Speed Synchronous Serial Interface

### 13.3 Continuous Transfers

When the transmit interrupt request flag is set, it indicates that the transmit buffer SSCTB is empty and ready to be loaded with the next transmit data. If SSCTB has been reloaded by the time the current transmission is finished, the data is immediately transferred to the shift register and the next transmission will start without any additional delay. On the data line there is no gap between the two successive frames. E.g. two byte transfers would look the same as one word transfer. This feature can be used to interface with devices which can operate with or require more than 16 data bits per transfer. It is just a matter of software, how long a total data frame length can be. This option can also be used e.g. to interface to byte-wide and word-wide devices on the same serial bus.

*Note: Of course, this can only happen in multiples of the selected basic data width, since it would require disabling/enabling of the SSC to reprogram the basic data width on-the-fly.*

The High-Speed Synchronous Serial Interface

13.4 Port Control

The SSC uses three pins of Port 3 to communicate with the external world. Pin P3.13/SCLK serves as the clock line, while pins P3.8/MRST (Master Receive/Slave Transmit) and P3.9/MTSR (Master Transmit/Slave Receive) serve as the serial data input/output lines. The operation of these pins depends on the selected operating mode (master or slave). In order to enable the alternate output functions of these pins instead of the general purpose IO operation, the respective port latches have to be set to '1', since the port latch outputs and the alternate output lines are ANDed. When an alternate data output line is not used (function disabled), it is held at a high level, allowing IO operations via the port latch. The direction of the port lines depends on the operating mode. The SSC will automatically use the correct alternate input or output line of the ports when switching modes. The direction of the pins, however, must be programmed by the user, as shown in the tables. Using the open drain output feature helps to avoid bus contention problems and reduces the need for hardwired hand-shaking or slave select lines. In this case it is not always necessary to switch the direction of a port pin. **Table 13-1** summarizes the required values for the different modes and pins.

Table 13-1 SSC Port Control

Pin	Master Mode			Slave Mode		
	Function	Port Latch	Direction	Function	Port Latch	Direction
SCLK	Serial Clock Output	P3.13 = '1'	DP3.13 = '1'	Serial Clock Input	P3.13 = 'x'	DP3.13 = '0'
MTSR	Serial Data Output	P3.9 = '1'	DP3.9 = '1'	Serial Data Input	P3.9 = 'x'	DP3.9 = '0'
MRST	Serial Data Input	P3.8 = 'x'	DP3.8 = '0'	Serial Data Output	P3.8 = '1'	DP3.8 = '1'

Note: In **Table 13-1**, an 'x' means that the actual value is irrelevant in the respective mode, however, it is recommended to set these bits to '1', so they are already in the correct state when switching between master and slave mode.

The High-Speed Synchronous Serial Interface

### 13.5 Baud Rate Generation

The serial channel SSC has its own dedicated 16-bit baud rate generator with 16-bit reload capability, permitting baud rate generation independent from the timers.

The baud rate generator is clocked with the CPU clock divided by 2 ( $f_{CPU} / 2$ ). The timer is counting downwards and can be started or stopped through the global enable bit SSCEN in register SSCON. Register SSCBR is the dual-function Baud Rate Generator/Reload register. Reading SSCBR, while the SSC is enabled, returns the content of the timer. Reading SSCBR, while the SSC is disabled, returns the programmed reload value. In this mode the desired reload value can be written to SSCBR.

*Note: Never write to SSCBR, while the SSC is enabled.*

The formulas below calculate either the resulting baud rate for a given reload value, or the required reload value for a given baudrate:

$$B_{SSC} = \frac{f_{CPU}}{2 \times (\langle SSCBR \rangle + 1)} \quad , \quad SSCBR = \left( \frac{f_{CPU}}{2 \times \text{Baudrate}_{SSC}} \right) - 1$$

$\langle SSCBR \rangle$  represents the content of the reload register, taken as an unsigned 16-bit integer.

**Table 13-2** lists some possible baud rates together with the required reload values and the resulting bit times, for different CPU clock frequencies.

**Table 13-2 SSC Bit-Time Calculation**

Bit-time for $f_{CPU} = \dots$				Reload Val. (SSCBR)
16 MHz	20 MHz	25 MHz	33 MHz	
Reserved. SSCBR must be > 0.				0000 <sub>H</sub>
250 ns	200 ns	160 ns	121 ns	0001 <sub>H</sub>
375 ns	300 ns	240 ns	182 ns	0002 <sub>H</sub>
500 ns	400 ns	320 ns	242 ns	0003 <sub>H</sub>
625 ns	500 ns	400 ns	303 ns	0004 <sub>H</sub>
1.00 $\mu$ s	800 ns	640 ns	485 ns	0007 <sub>H</sub>
1.25 $\mu$ s	1 $\mu$ s	800 ns	606 ns	0009 <sub>H</sub>
10 $\mu$ s	8 $\mu$ s	6.4 $\mu$ s	4.8 $\mu$ s	004F <sub>H</sub>
12.5 $\mu$ s	10 $\mu$ s	8 $\mu$ s	6.1 $\mu$ s	0063 <sub>H</sub>
15.6 $\mu$ s	12.5 $\mu$ s	10 $\mu$ s	7.6 $\mu$ s	007C <sub>H</sub>
20.6 $\mu$ s	16.5 $\mu$ s	13.2 $\mu$ s	10 $\mu$ s	00A4 <sub>H</sub>

The High-Speed Synchronous Serial Interface

**Table 13-2 SSC Bit-Time Calculation (cont'd)**

Bit-time for $f_{CPU} = \dots$				Reload Val. (SSCBR)
16 MHz	20 MHz	25 MHz	33 MHz	
1 ms	800 $\mu$ s	640 $\mu$ s	485 $\mu$ s	1F3F <sub>H</sub>
1.25 ms	1 ms	800 $\mu$ s	606 $\mu$ s	270F <sub>H</sub>
1.56 ms	1.25 ms	1 ms	758 $\mu$ s	30D3 <sub>H</sub>
8.2 ms	6.6 ms	5.2 ms	4.0 ms	FFFF <sub>H</sub>

**Table 13-3 SSC Baudrate Calculation**

Baud Rate for $f_{CPU} = \dots$				Reload Val. (SSCBR)
16 MHz	20 MHz	25 MHz	33 MHz	
Reserved. SSCBR must be > 0.				0000 <sub>H</sub>
<b>4.00 MBaud</b>	<b>5.00 MBaud</b>	<b>6.25 MBaud</b>	<b>8.25 MBaud</b>	0001 <sub>H</sub>
2.67 MBaud	3.33 MBaud	4.17 MBaud	5.50 MBaud	0002 <sub>H</sub>
2.00 MBaud	2.50 MBaud	3.13 MBaud	4.13 MBaud	0003 <sub>H</sub>
1.60 MBaud	2.00 MBaud	2.50 MBaud	3.30 MBaud	0004 <sub>H</sub>
<b>1.00 MBaud</b>	1.25 MBaud	1.56 MBaud	2.06 MBaud	0007 <sub>H</sub>
800 kBaud	<b>1.0 MBaud</b>	1.25 MBaud	1.65 MBaud	0009 <sub>H</sub>
<b>100 kBaud</b>	125 kBaud	156 kBaud	206 kBaud	004F <sub>H</sub>
80 kBaud	<b>100 kBaud</b>	125 kBaud	165 kBaud	0063 <sub>H</sub>
64 kBaud	80 kBaud	<b>100 kBaud</b>	132 kBaud	007C <sub>H</sub>
48.5 kBaud	60.6 kBaud	75.8 kBaud	<b>100 kBaud</b>	00A4 <sub>H</sub>
<b>1.0 kBaud</b>	1.25 kBaud	1.56 kBaud	2.06 kBaud	1F3F <sub>H</sub>
800 Baud	<b>1.0 kBaud</b>	1.25 kBaud	1.65 kBaud	270F <sub>H</sub>
640 Baud	800 Baud	<b>1.0 kBaud</b>	1.32 kBaud	30D3 <sub>H</sub>
122.1 Baud	152.6 Baud	190.7 Baud	251.7 Baud	FFFF <sub>H</sub>



---

## The High-Speed Synchronous Serial Interface

### 13.6 Error Detection Mechanisms

The SSC is able to detect four different error conditions. Receive Error and Phase Error are detected in all modes, while Transmit Error and Baudrate Error only apply to slave mode. When an error is detected, the respective error flag is set. When the corresponding Error Enable Bit is set, also an error interrupt request will be generated by setting SSCEIR (see [Figure 13-6](#)). The error interrupt handler may then check the error flags to determine the cause of the error interrupt. The error flags are not reset automatically (like SSCEIR), but rather must be cleared by software after servicing. This allows servicing of some error conditions via interrupt, while the others may be polled by software.

*Note: The error interrupt handler must clear the associated (enabled) errorflag(s) to prevent repeated interrupt requests.*

A **Receive Error** (Master or Slave mode) is detected, when a new data frame is completely received, but the previous data was not read out of the receive buffer register SSCRB. This condition sets the error flag SSCRE and, when enabled via SSCREN, the error interrupt request flag SSCEIR. The old data in the receive buffer SSCRB will be overwritten with the new value and is unretrievably lost.

A **Phase Error** (Master or Slave mode) is detected, when the incoming data at pin MRST (master mode) or MTSR (slave mode), sampled with the same frequency as the CPU clock, changes between one sample before and two samples after the latching edge of the clock signal (see "Clock Control"). This condition sets the error flag SSCPE and, when enabled via SSCPEN, the error interrupt request flag SSCEIR.

A **Baud Rate Error** (Slave mode) is detected, when the incoming clock signal deviates from the programmed baud rate by more than 100%, i.e. it either is more than double or less than half the expected baud rate. This condition sets the error flag SSCBE and, when enabled via SSCBEN, the error interrupt request flag SSCEIR. Using this error detection capability requires that the slave's baud rate generator is programmed to the same baud rate as the master device. This feature detects false additional, or missing pulses on the clock line (within a certain frame).

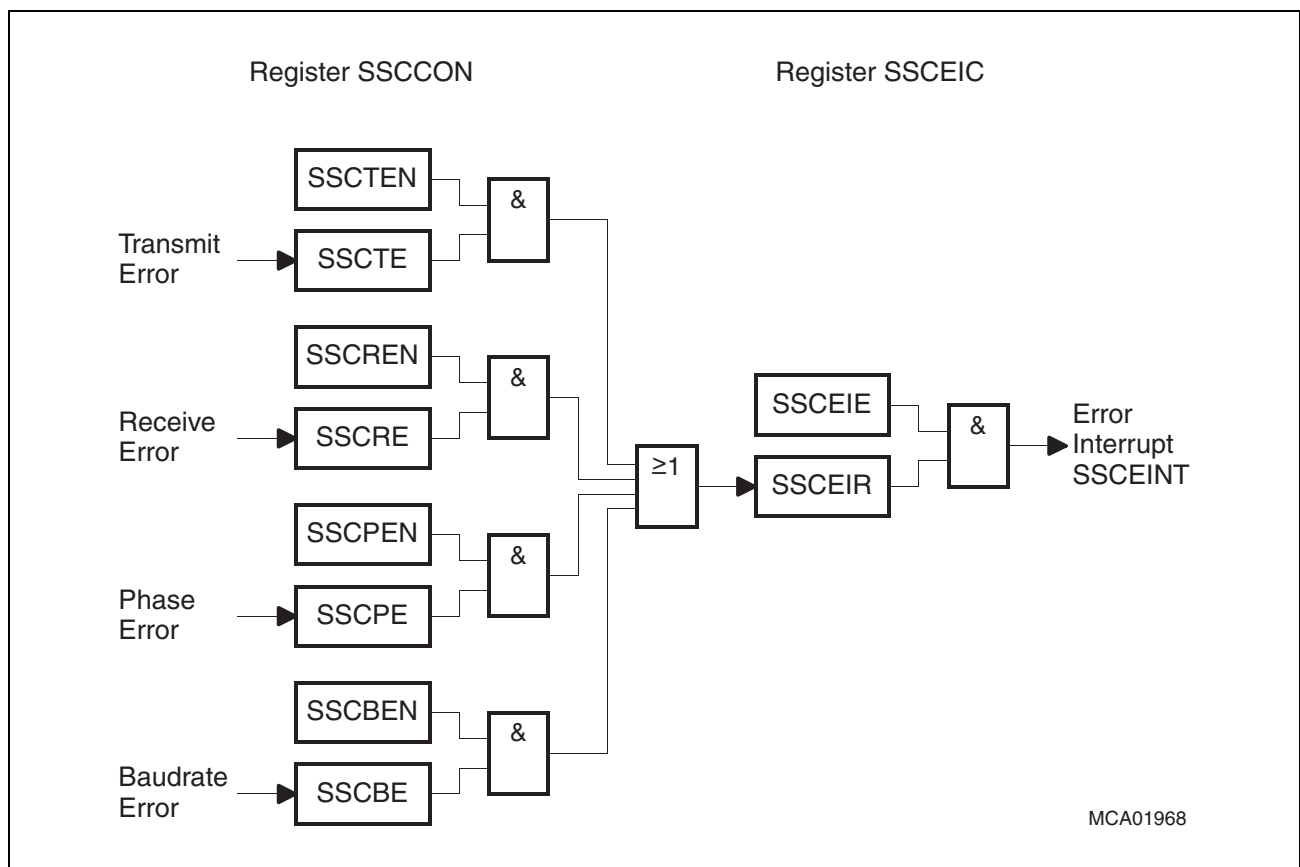
*Note: If this error condition occurs and bit SSCAREN = '1', an automatic reset of the SSC will be performed in case of this error. This is done to reinitialize the SSC, if too few or too many clock pulses have been detected.*

## The High-Speed Synchronous Serial Interface

A **Transmit Error** (Slave mode) is detected, when a transfer was initiated by the master (shift clock gets active), but the transmit buffer SSCTB of the slave was not updated since the last transfer. This condition sets the error flag SSCTE and, when enabled via SSCTEN, the error interrupt request flag SSCEIR. If a transfer starts while the transmit buffer is not updated, the slave will shift out the 'old' contents of the shift register, which normally is the data received during the last transfer.

This may lead to the corruption of the data on the transmit/receive line in half-duplex mode (open drain configuration), if this slave is not selected for transmission. This mode requires that slaves not selected for transmission only shift out ones, i.e. their transmit buffers must be loaded with 'FFFF<sub>H</sub>' prior to any transfer.

*Note: A slave with push/pull output drivers, which is not selected for transmission, will normally have its output drivers switched. However, in order to avoid possible conflicts or misinterpretations, it is recommended to always load the slave's transmit buffer prior to any transfer.*



**Figure 13-6 SSC Error Interrupt Control**

The High-Speed Synchronous Serial Interface

### 13.7 SSC Interrupt Control

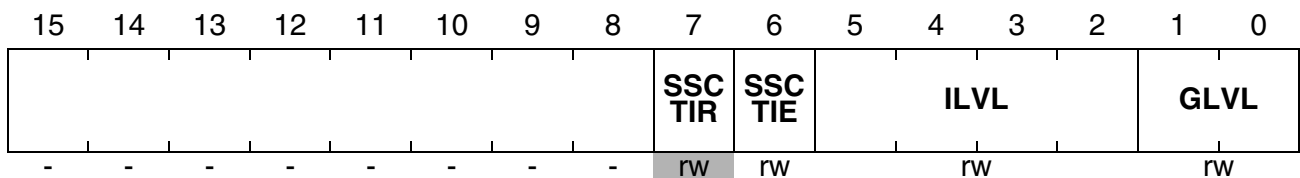
Three bit addressable interrupt control registers are provided for serial channel SSC. Register SSCTIC controls the transmit interrupt, SSCRIC controls the receive interrupt and SSCEIC controls the error interrupt of serial channel SSC. Each interrupt source also has its own dedicated interrupt vector. SCTINT is the transmit interrupt vector, SCRINT is the receive interrupt vector, and SCEINT is the error interrupt vector.

The cause of an error interrupt request (receive, phase, baudrate, transmit error) can be identified by the error status flags in control register SSCCON.

*Note: In contrary to the error interrupt request flag SSCEIR, the error status flags SSCxE are not reset automatically upon entry into the error interrupt service routine, but must be cleared by software.*

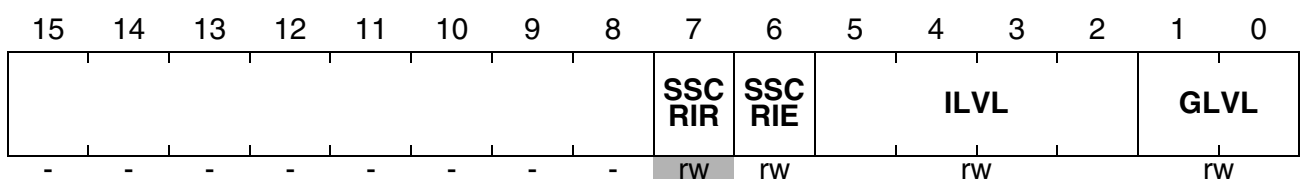
#### SSCTIC

SSC Transmit Intr. Ctrl. Reg.      SFR (FF72<sub>H</sub>/B9<sub>H</sub>)      Reset Value: - - 00<sub>H</sub>



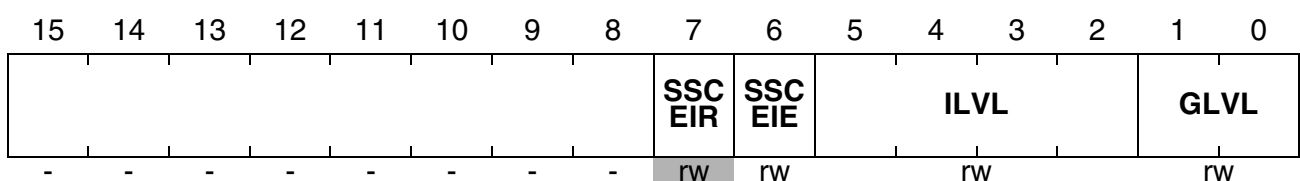
#### SSCRIC

SSC Receive Intr. Ctrl. Reg.      SFR (FF74<sub>H</sub>/BA<sub>H</sub>)      Reset Value: - - 00<sub>H</sub>



#### SSCEIC

SSC Error Intr. Ctrl. Reg.      SFR (FF76<sub>H</sub>/BB<sub>H</sub>)      Reset Value: - - 00<sub>H</sub>



*Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.*

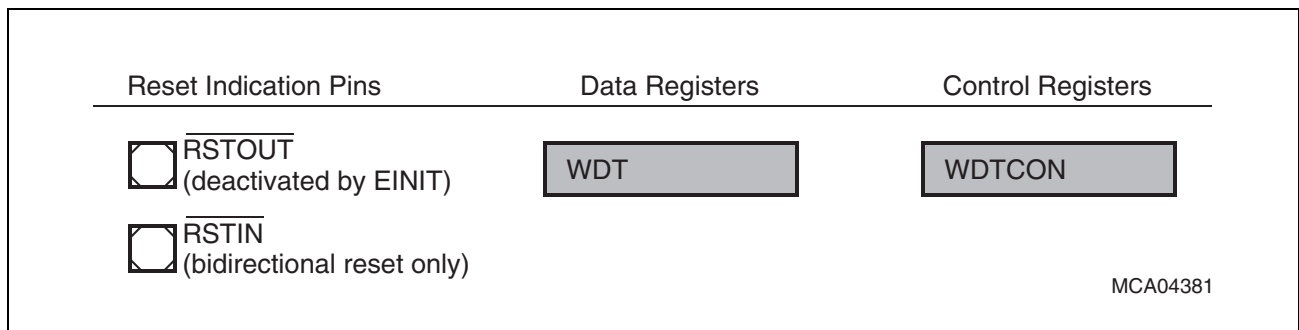
## 14 The Watchdog Timer (WDT)

To allow recovery from software or hardware failure, the C161CS/JC/JI provides a Watchdog Timer. If the software fails to service this timer before an overflow occurs, an internal reset sequence will be initiated. This internal reset will also pull the  $\overline{\text{RSTOUT}}$  pin low, which also resets the peripheral hardware which might be the cause for the malfunction. When the watchdog timer is enabled and the software has been designed to service it regularly before it overflows, the watchdog timer will supervise the program execution as it only will overflow if the program does not progress properly. The watchdog timer will also time out if a software error was due to hardware related failures. This prevents the controller from malfunctioning for longer than a user-specified time.

*Note: When the bidirectional reset is enabled also pin  $\overline{\text{RSTIN}}$  will be pulled low for the duration of the internal reset sequence upon a software reset or a watchdog timer reset.*

The watchdog timer provides two registers:

- a read-only timer register that contains the current count and
- a control register for initialization and reset source detection.



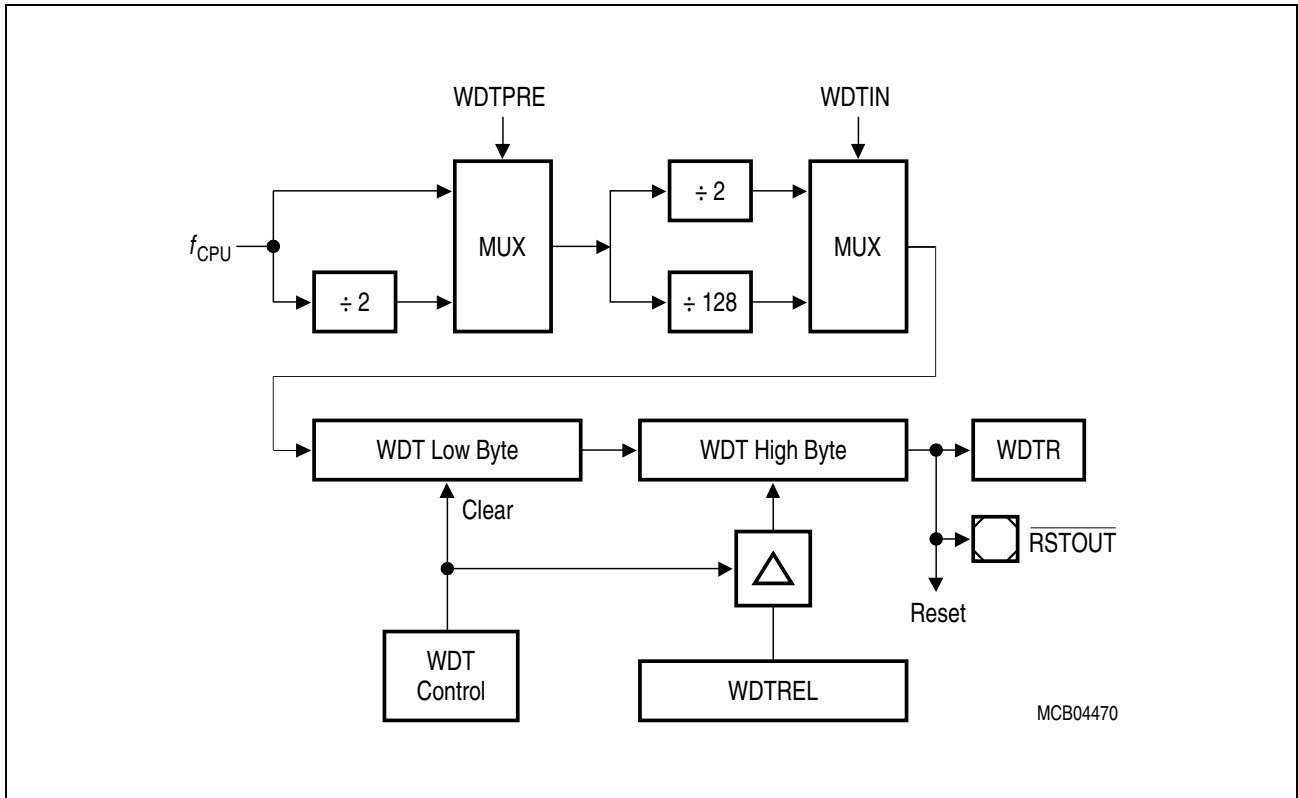
**Figure 14-1 SFRs and Port Pins Associated with the Watchdog Timer**

The watchdog timer is a 16-bit up counter which is clocked with the prescaled CPU clock ( $f_{\text{CPU}}$ ). The prescaler divides the CPU clock:

- by 2 (WDTIN = '0', WDTPRE = '0'), or
- by 4 (WDTIN = '0', WDTPRE = '1'), or
- by 128 (WDTIN = '1', WDTPRE = '0'), or
- by 256 (WDTIN = '1', WDTPRE = '1').

**The Watchdog Timer (WDT)**

The 16-bit watchdog timer is realized as two concatenated 8-bit timers (see [Figure 14-2](#)). The upper 8 bits of the watchdog timer can be preset to a user-programmable value via a watchdog service access in order to vary the watchdog expire time. The lower 8 bits are reset upon each service access.



**Figure 14-2 Watchdog Timer Block Diagram**

## The Watchdog Timer (WDT)

## 14.1 Operation of the Watchdog Timer

The current count value of the Watchdog Timer is contained in the Watchdog Timer Register WDT which is a non-bitaddressable read-only register. The operation of the Watchdog Timer is controlled by its bitaddressable Watchdog Timer Control Register WDTCON. This register specifies the reload value for the high byte of the timer, selects the input clock prescaling factor and also provides flags that indicate the source of a reset.

After any reset (except see note) the watchdog timer is enabled and starts counting up from 0000<sub>H</sub> with the default frequency  $f_{WDT} = f_{CPU} / 2$ . The default input frequency may be changed to another frequency ( $f_{WDT} = f_{CPU} / 4, 128, 256$ ) by programming the prescaler (bits WDTPRE and WDTIN).

The watchdog timer can be disabled by executing the instruction DISWDT (Disable Watchdog Timer). Instruction DISWDT is a protected 32-bit instruction which will ONLY be executed during the time between a reset and execution of either the EINIT (End of Initialization) or the SRVWDT (Service Watchdog Timer) instruction. Either one of these instructions disables the execution of DISWDT.

*Note: After a hardware reset that activates the Bootstrap Loader the watchdog timer will be disabled. The software reset that terminates the BSL mode will then enable the WDT.*

When the watchdog timer is not disabled via instruction DISWDT it will continue counting up, even during Idle Mode. If it is not serviced via the instruction SRVWDT by the time the count reaches FFFF<sub>H</sub> the watchdog timer will overflow and cause an internal reset. This reset will pull the external reset indication pin  $\overline{RSTOUT}$  low. The Watchdog Timer Reset Indication Flag (WDTR) in register WDTCON will be set in this case.

In bidirectional reset mode also pin  $\overline{RSTIN}$  will be pulled low for the duration of the internal reset sequence and a long hardware reset will be indicated instead.

A watchdog reset will also complete a running external bus cycle before starting the internal reset sequence if this bus cycle does not use  $\overline{READY}$  or samples  $\overline{READY}$  active (low) after the programmed waitstates. Otherwise the external bus cycle will be aborted.

To prevent the watchdog timer from overflowing it must be serviced periodically by the user software. The watchdog timer is serviced with the instruction SRVWDT which is a protected 32-bit instruction. Servicing the watchdog timer clears the low byte and reloads the high byte of the watchdog timer register WDT with the preset value from bitfield WDTREL which is the high byte of register WDTCON. Servicing the watchdog timer will also reset bit WDTR. After being serviced the watchdog timer continues counting up from the value  $(\langle WDTREL \rangle \times 2^8)$ .

Instruction SRVWDT has been encoded in such a way that the chance of unintentionally servicing the watchdog timer (e.g. by fetching and executing a bit pattern from a wrong location) is minimized. When instruction SRVWDT does not match the format for

The Watchdog Timer (WDT)

protected instructions the Protection Fault Trap will be entered, rather than the instruction be executed.

**WDTCON**

**WDT Control Register**

**SFR (FFAE<sub>H</sub>/D7<sub>H</sub>)**

**Reset Value: 00XX<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDTREL								WDT PRE	-	-	LHW R	SHW R	SW R	WDT R	WDT IN
								rw	-	-	rh	rh	rh	rh	rw

Bit	Function
WDTIN	<b>Watchdog Timer Input Frequency Select</b> Controls the input clock prescaler. See <a href="#">Table 14-1</a> .
WDTR	<b>Watchdog Timer Reset Indication Flag</b> Cleared by a hardware reset or by the SRVWDT instruction.
SWR	<b>Software Reset Indication Flag</b>
SHWR	<b>Short Hardware Reset Indication Flag</b>
LHWR	<b>Long Hardware Reset Indication Flag</b>
WDTPRE	<b>Watchdog Timer Input Prescaler Control</b> Controls the input clock prescaler. See <a href="#">Table 14-1</a> .
WDTREL	<b>Watchdog Timer Reload Value</b> (for the high byte of WDT)

*Note: The reset value depends on the reset source (see description below).  
The execution of EINIT clears the reset indication flags.*

The time period for an overflow of the watchdog timer is programmable in two ways:

- **The input frequency** to the watchdog timer can be selected via a prescaler controlled by bits WDTPRE and WDTIN in register WDTCON to be  $f_{CPU} / 2$ ,  $f_{CPU} / 4$ ,  $f_{CPU} / 128$ , or  $f_{CPU} / 256$ .
- **The reload value** WDTREL for the high byte of WDT can be programmed in register WDTCON.

The period  $P_{WDT}$  between servicing the watchdog timer and the next overflow can therefore be determined by the following formula:

$$P_{WDT} = \frac{2^{(1 + \langle WDTPRE \rangle + \langle WDTIN \rangle \times 6)} \times (2^{16} - \langle WDTREL \rangle \times 2^8)}{f_{CPU}}$$

**The Watchdog Timer (WDT)**

**Table 14-1** marks the possible ranges (depending on the prescaler bits WDTPRE and WDTIN) for the watchdog time which can be achieved using a certain CPU clock.

**Table 14-1 Watchdog Time Ranges**

CPU clock $f_{CPU}$	Prescaler		$f_{WDT}$	Reload value in WDTREL		
	WDT IN	WDT PRE		FF <sub>H</sub>	7F <sub>H</sub>	00 <sub>H</sub>
12 MHz	0	0	$f_{CPU} / 2$	42.67 $\mu$ s	5.50 ms	10.92 ms
	0	1	$f_{CPU} / 4$	85.33 $\mu$ s	11.01 ms	21.85 ms
	1	0	$f_{CPU} / 128$	2.73 ms	352.3 ms	699.1 ms
	1	1	$f_{CPU} / 256$	5.46 ms	704.5 ms	1398 ms
16 MHz	0	0	$f_{CPU} / 2$	32.00 $\mu$ s	4.13 ms	8.19 ms
	0	1	$f_{CPU} / 4$	64.00 $\mu$ s	8.26 ms	16.38 ms
	1	0	$f_{CPU} / 128$	2.05 ms	264.2 ms	524.3 ms
	1	1	$f_{CPU} / 256$	4.10 ms	528.4 ms	1049 ms
20 MHz	0	0	$f_{CPU} / 2$	25.60 $\mu$ s	3.30 ms	6.55 ms
	0	1	$f_{CPU} / 4$	51.20 $\mu$ s	6.61 ms	13.11 ms
	1	0	$f_{CPU} / 128$	1.64 ms	211.4 ms	419.4 ms
	1	1	$f_{CPU} / 256$	3.28 ms	422.7 ms	838.9 ms
25 MHz	0	0	$f_{CPU} / 2$	20.48 $\mu$ s	2.64 ms	5.24 ms
	0	1	$f_{CPU} / 4$	40.96 $\mu$ s	5.28 ms	10.49 ms
	1	0	$f_{CPU} / 128$	1.31 ms	169.1 ms	335.5 ms
	1	1	$f_{CPU} / 256$	2.62 ms	338.2 ms	671.1 ms
33 MHz	0	0	$f_{CPU} / 2$	15.52 $\mu$ s	2.00 ms	3.97 ms
	0	1	$f_{CPU} / 4$	31.03 $\mu$ s	4.00 ms	7.94 ms
	1	0	$f_{CPU} / 128$	0.99 ms	128.1 ms	254.2 ms
	1	1	$f_{CPU} / 256$	1.99 ms	256.2 ms	508.4 ms

*Note: For safety reasons, the user is advised to rewrite WDTCON each time before the watchdog timer is serviced.*



## 14.2 Reset Source Indication

The reset indication flags in register WDTCON provide information on the source for the last reset. As the C161CS/JC/JI starts executing from location 00'0000<sub>H</sub> after any possible reset event the initialization software may check these flags in order to determine if the recent reset event was triggered by an external hardware signal (via  $\overline{RSTIN}$ ), by software itself or by an overflow of the watchdog timer. The initialization (and also the further operation) of the microcontroller system can thus be adapted to the respective circumstances, e.g. a special routine may verify the software integrity after a watchdog timer reset.

The reset indication flags are not mutually exclusive, i.e. more than one flag may be set after reset depending on its source. [Table 14-2](#) summarizes the possible combinations:

**Table 14-2 Reset Indication Flag Combinations**

Event	Reset Indication Flags <sup>1)</sup>			
	LHWR	SHWR	SWR	WDTR
Long Hardware Reset	1	1	1	0
Short Hardware Reset	*	1	1	0
Software Reset	*	*	1	–
Watchdog Timer Reset	*	*	1	1
EINIT instruction	0	0	0	–
SRVWDT instruction	–	–	–	0

<sup>1)</sup> Description of table entries:

'1' = flag is set, '0' = flag is cleared, '–' = flag is not affected,

'\*' = flag is set in bidirectional reset mode, not affected otherwise.

**Long Hardware Reset** is indicated when the  $\overline{RSTIN}$  input is still sampled low (active) at the end of a hardware triggered internal reset sequence.

**Short Hardware Reset** is indicated when the  $\overline{RSTIN}$  input is sampled high (inactive) at the end of a hardware triggered internal reset sequence.

**Software Reset** is indicated after a reset triggered by the execution of instruction SRST.

**Watchdog Timer Reset** is indicated after a reset triggered by an overflow of the watchdog timer.

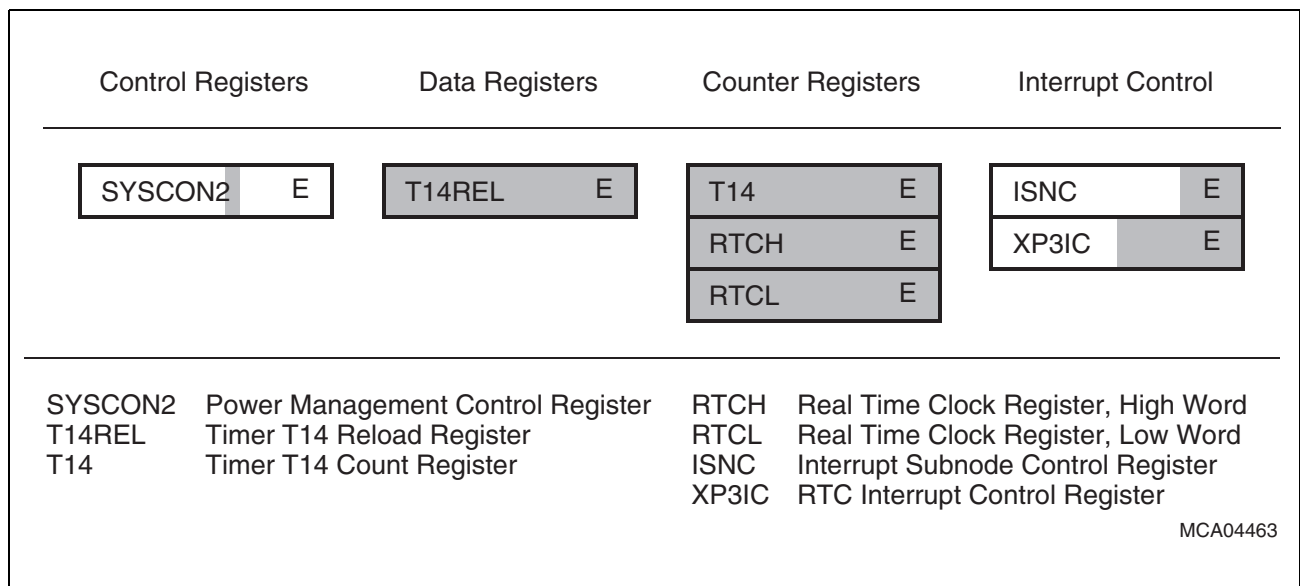
*Note: When bidirectional reset is enabled the  $\overline{RSTIN}$  pin is pulled low for the duration of the internal reset sequence upon any sort of reset.*

*Therefore always a long hardware reset (LHWR) will be recognized in any case.*

## 15 The Real Time Clock

The Real Time Clock (RTC) module of the C161CS/JC/JI basically is an independent timer chain which is clocked directly with the oscillator clock and serves for different purposes:

- System clock to determine the current time and date
- Cyclic time based interrupt
- 48-bit timer for long term measurements



**Figure 15-1 SFRs Associated with the RTC Module**

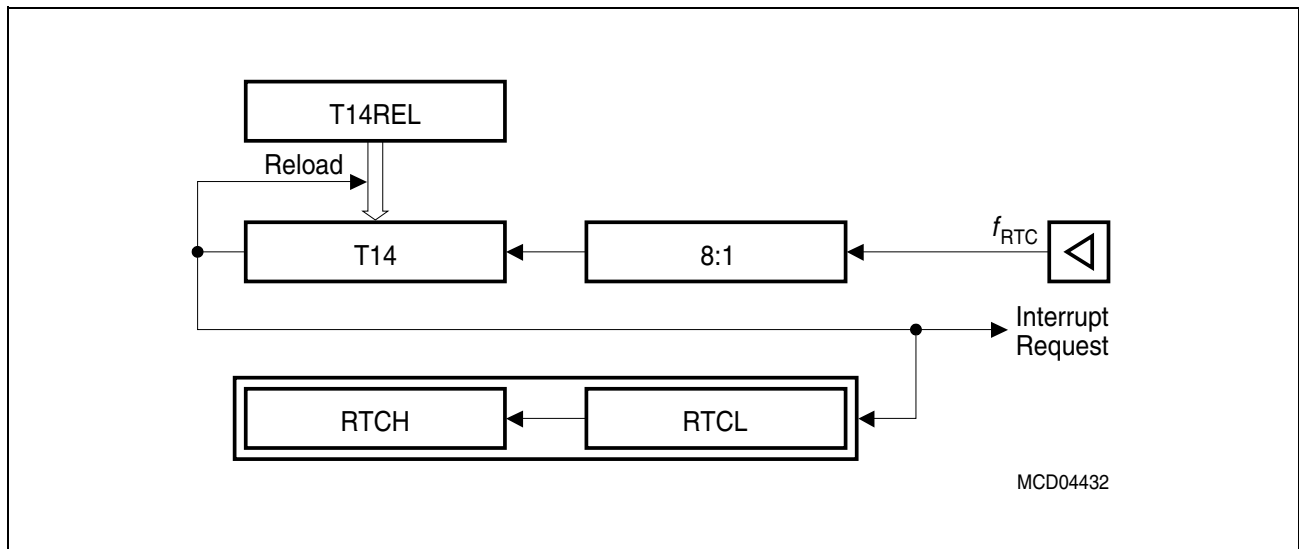
The RTC module consists of a chain of 3 divider blocks, a fixed 8:1 divider, the reloadable 16-bit timer T14 and the 32-bit RTC timer (accessible via registers RTCH and RTCL). Both timers count up.

The clock signal for the RTC module is directly derived from the on-chip oscillator frequency (not from the CPU clock) and fed through a separate clock driver. It is therefore independent from the selected clock generation mode of the C161CS/JC/JI and is controlled by the clock generation circuitry.

**Table 15-1 RTC Register Location within the ESFR space.**

Register Name	Long / Short Address	Reset Value	Notes
<b>T14</b>	F0D2 <sub>H</sub> / 69 <sub>H</sub>	UUUU <sub>H</sub>	Prescaler timer, generates input clock for RTC register and periodic interrupt
<b>T14REL</b>	F0D0 <sub>H</sub> / 68 <sub>H</sub>	UUUU <sub>H</sub>	Timer reload register
<b>RTCH</b>	F0D6 <sub>H</sub> / 6B <sub>H</sub>	UUUU <sub>H</sub>	High word of RTC register
<b>RTCL</b>	F0D4 <sub>H</sub> / 6A <sub>H</sub>	UUUU <sub>H</sub>	Low word of RTC register

*Note: The RTC registers are not affected by a reset. After a power on reset, however, they are undefined.*



**Figure 15-2 RTC Block Diagram**

### System Clock Operation

A real time system clock can be maintained that keeps on running also during idle mode and power down mode (optionally) and represents the current time and date. This is possible as the RTC module is not effected by a reset.

The maximum resolution (minimum stepwidth) for this clock information is determined by timer T14's input clock. The maximum usable timespan is achieved when T14REL is loaded with 0000<sub>H</sub> and so T14 divides by  $2^{16}$ .

### Cyclic Interrupt Generation

The RTC module can generate an interrupt request whenever timer T14 overflows and is reloaded. This interrupt request may e.g. be used to provide a system time tick independent of the CPU frequency without loading the general purpose timers, or to wake up regularly from idle mode. The interrupt cycle time can be adjusted via the timer T14 reload register T14REL. Please refer to "RTC Interrupt Generation" below for more details.

### 48-bit Timer Operation

The concatenation of the 16-bit reload timer T14 and the 32-bit RTC timer can be regarded as a 48-bit timer which is clocked with the RTC input frequency divided by the fixed prescaler. The reload register T14REL should be cleared to get a 48-bit binary timer. However, any other reload value may be used.

The maximum usable timespan is  $2^{48}$  ( $\approx 10^{14}$ ) T14 input clocks, which would equal more than 100 years at an oscillator frequency of 20 MHz.

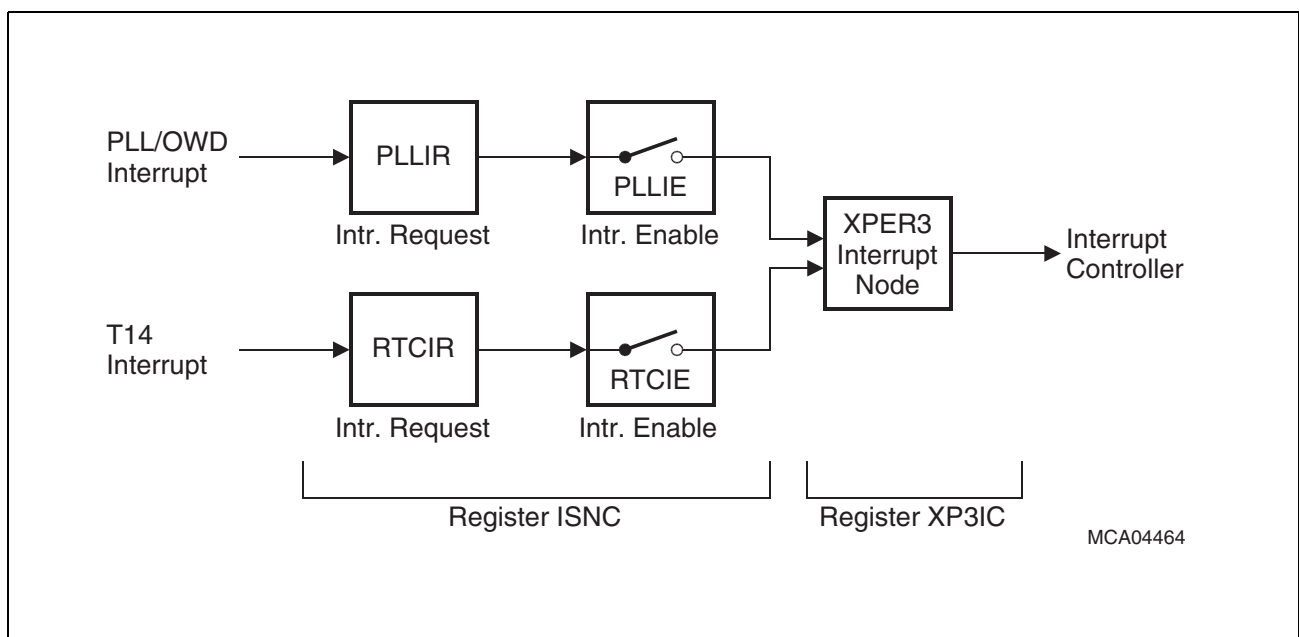
### RTC Register Access

The actual value of the RTC is represented by the 3 registers T14, RTCL and RTCH. As these registers are concatenated to build the RTC counter chain, internal overflows occur while the RTC is running. When reading or writing the RTC value make sure to account for such internal overflows in order to avoid reading/writing corrupted values. When reading/writing e.g. 0000<sub>H</sub> to RTCH and then accessing RTCL will produce a corrupted value as RTCL may overflow before it can be accessed. In this case, however, RTCH would be 0001<sub>H</sub>. The same precautions must be taken for T14 and T14REL.

### RTC Interrupt Generation

The RTC interrupt shares the XPER3 interrupt node with the PLL/OWD interrupt. This is controlled by the interrupt subnode control register ISNC. The interrupt handler can determine the source of an interrupt request via the separate interrupt request and enable flags (see [Figure 15-3](#)) provided in register ISNC.

*Note: If only one source is enabled no additional software check is required, of course.*



**Figure 15-3 RTC Interrupt Logic**

If T14 interrupts are to be used both stages, the interrupt node (XP3IE = '1') and the RTC subnode (RTCIE = '1') must be enabled.

Please note that the node request bit XP3IR is automatically cleared when the interrupt handler is vectored to, while the subnode request bit RTCIR must be cleared by software.

### Defining the RTC Time Base

The reload timer T14 determines the input frequency of the RTC timer, i.e. the RTC time base, as well as the T14 interrupt cycle time. **Table 15-2** lists the interrupt period range and the T14 reload values (for a time base of 1 s and 1 ms) for several oscillator frequencies:

**Table 15-2 RTC Interrupt Periods and Reload Values**

Oscillator Frequency	RTC Interrupt Period		Reload Value A		Reload Value B	
	Minimum	Maximum	T14REL	Base	T14REL	Base
32.768 kHz Aux.	244.14 $\mu$ s	16.0 s	F000 <sub>H</sub>	1.000 s	FFFC <sub>H</sub>	0.977 ms
32 kHz Aux.	250 $\mu$ s	16.38 s	F060 <sub>H</sub>	1.000 s	FFFC <sub>H</sub>	1.000 ms
32 kHz Main	8000 $\mu$ s	524.29 s	FF83 <sub>H</sub>	1.000 s	–	–
4 MHz Main	64.0 $\mu$ s	4.19 s	C2F7 <sub>H</sub>	1.000 s	FFF0 <sub>H</sub>	1.024 ms
5 MHz Main	51.2 $\mu$ s	3.35 s	B3B5 <sub>H</sub>	0.999 s	FFEC <sub>H</sub>	1.024 ms
8 MHz Main	32.0 $\mu$ s	2.10 s	85EE <sub>H</sub>	1.000 s	FFE1 <sub>H</sub>	0.992 ms
10 MHz Main	25.6 $\mu$ s	1.68 s	676A <sub>H</sub>	0.999 s	FFD9 <sub>H</sub>	0.998 ms
12 MHz Main	21.3 $\mu$ s	1.40 s	48E5 <sub>H</sub>	1.000 s	FFD2 <sub>H</sub>	1.003 ms
16 MHz Main	16.0 $\mu$ s	1.05 s	0BDC <sub>H</sub>	1.000 s	FFC2 <sub>H</sub>	0.992 ms

### Increased RTC Accuracy through Software Correction

The accuracy of the C161CS/JC/JI's RTC is determined by the oscillator frequency and by the respective prescaling factor (excluding or including T14). The accuracy limit generated by the prescaler is due to the quantization of a binary counter (where the average is zero), while the accuracy limit generated by the oscillator frequency is due to the difference between ideal and real frequency (and therefore accumulates over time). The total accuracy of the RTC can be further increased via software for specific applications that demand a high time accuracy.

The key to the improved accuracy is the knowledge of the exact oscillator frequency. The relation of this frequency to the expected ideal frequency is a measure for the RTC's deviation. The number N of cycles after which this deviation causes an error of  $\pm 1$  cycle can be easily computed. So the only action is to correct the count by  $\pm 1$  after each series of N cycles.

This correction may be applied to the RTC register as well as to T14.

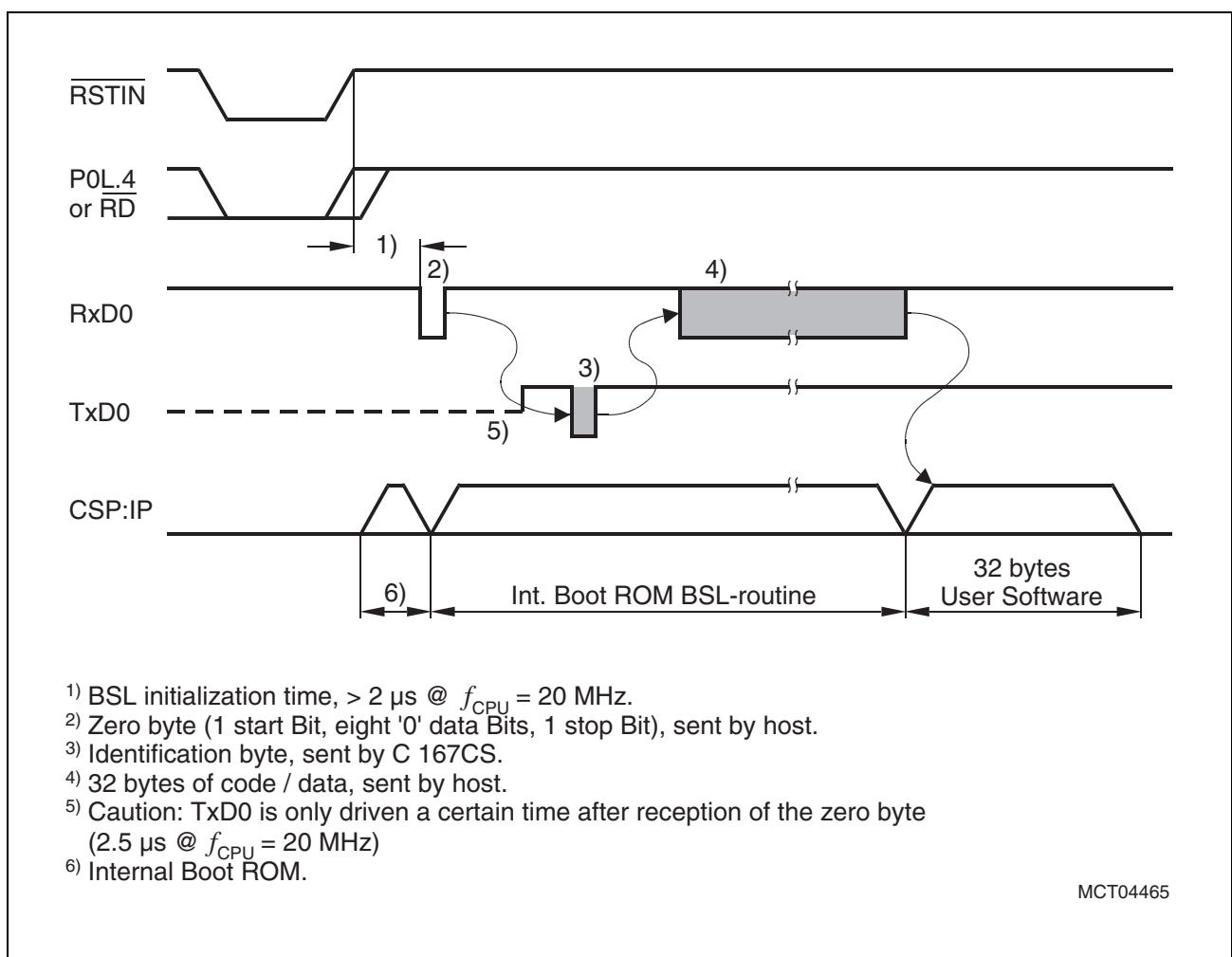
Also the correction may be done cyclic, e.g. within T14's interrupt service routine, or by evaluating a formula when the RTC registers are read (for this the respective "last" RTC value must be available somewhere).

*Note: For the majority of applications, however, the standard accuracy provided by the RTC's structure will be more than sufficient.*

## 16 The Bootstrap Loader

The built-in bootstrap loader of the C161CS/JC/JI provides a mechanism to load the startup program, which is executed after reset, via the serial interface. In this case no external memory or an internal ROM/OTP/Flash is required for the initialization code.

The bootstrap loader moves code/data into the internal RAM, but it is also possible to transfer data via the serial interface into an external RAM using a second level loader routine. ROM memory (internal or external) is not necessary. However, it may be used to provide lookup tables or may provide “core-code”, i.e. a set of general purpose subroutines, e.g. for IO operations, number crunching, system initialization, etc.



**Figure 16-1 Bootstrap Loader Sequence**

The Bootstrap Loader may be used to load the complete application software into ROMless systems, it may load temporary software into complete systems for testing or calibration, it may also be used to load a programming routine for Flash devices.

The BSL mechanism may be used for standard system startup as well as only for special occasions like system maintenance (firmware update) or end-of-line programming or testing.

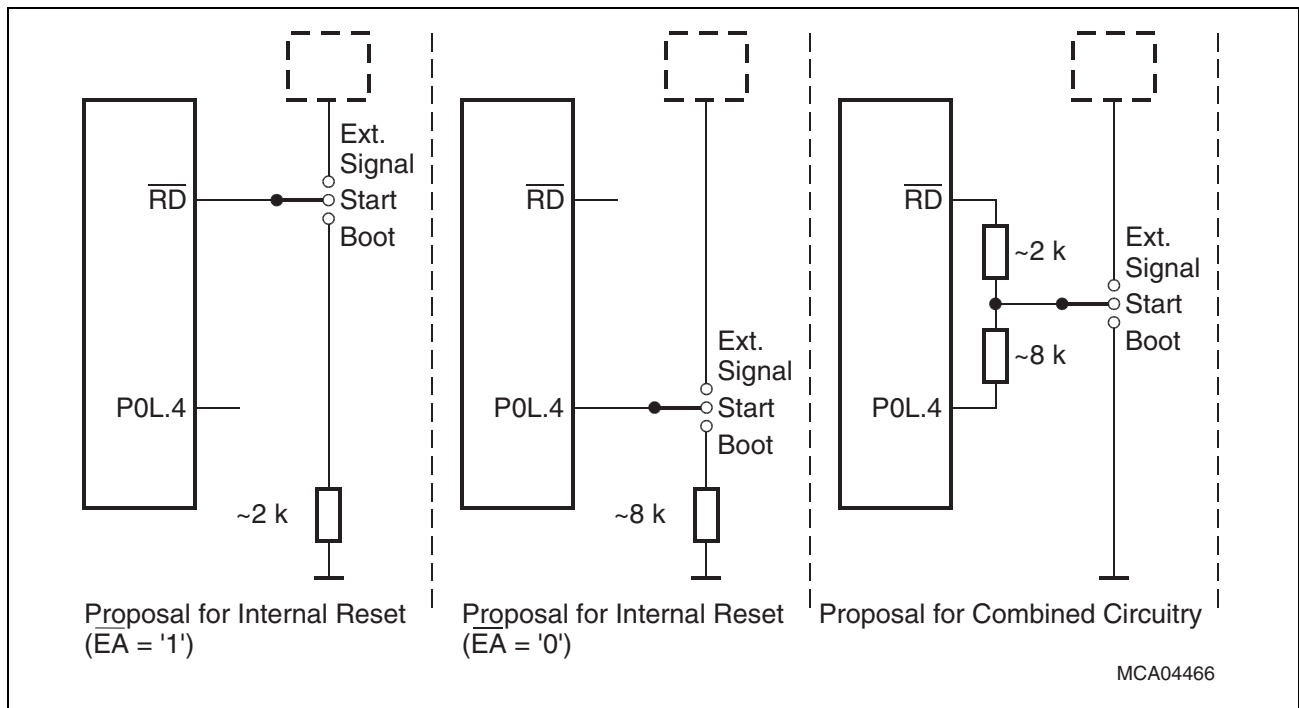
## 16.1 Entering the Bootstrap Loader

The C161CS/JC/JI enters BSL mode triggered by external configuration during a hardware reset:

- when pin P0L.4 is sampled low at the end of an external reset ( $\overline{EA} = '0'$ )
- when pin  $\overline{RD}$  is sampled low at the end of an internal reset ( $\overline{EA} = '1'$ ).

In this case the built-in bootstrap loader is activated independent of the selected bus mode. The bootstrap loader code is stored in a special Boot-ROM, no part of the standard mask ROM, OTP, or Flash memory area is required for this.

The hardware that activates the BSL during reset may be a simple pull-down resistor on P0L.4 or  $\overline{RD}$  for systems that use this feature upon every hardware reset. You may want to use a switchable solution (via jumper or an external signal) for systems that only temporarily use the bootstrap loader.



**Figure 16-2 Hardware Provisions to Activate the BSL**

The ASC0 receiver is only enabled after the identification byte has been transmitted. A half duplex connection to the host is therefore sufficient to feed the BSL.

*Note: The proper reset configuration for BSL mode requires more pins to be driven besides P0L.4 or  $\overline{RD}$ .*

*For an external reset ( $\overline{EA} = '0'$ ) bitfield SMOD must be configured as  $1011_B$  (see [Section 22.4.1](#)).*

*For an internal reset ( $\overline{EA} = '1'$ ) pin ALE must be driven to a defined level, e.g. ALE = '0' for the standard bootstrap loader (see [Section 22.4.2](#)).*

### Initial State in BSL Mode

After entering BSL mode and the respective initialization<sup>1)</sup> the C161CS/JC/JI scans the RxD0 line to receive a zero byte, i.e. one start bit, eight '0' data bits and one stop bit. From the duration of this zero byte it calculates the corresponding baudrate factor with respect to the current CPU clock, initializes the serial interface ASC0 accordingly and switches pin TxD0 to output. Using this baudrate, an identification byte is returned to the host that provides the loaded data.

This identification byte identifies the device to be booted. The following codes are defined:

55 <sub>H</sub> :	8xC166.
A5 <sub>H</sub> :	Previous versions of the C167 (obsolete).
B5 <sub>H</sub> :	Previous versions of the C165.
C5 <sub>H</sub> :	C167 derivatives.
D5 <sub>H</sub> :	All devices equipped with identification registers.

*Note: The identification byte D5<sub>H</sub> does not directly identify a specific derivative. This information can in this case be obtained from the identification registers.*

When the C161CS/JC/JI has entered BSL mode, the following configuration is automatically set (values that deviate from the normal reset values, are **marked**):

Watchdog Timer:	<b>Disabled</b>	Register STKUN:	FC00 <sub>H</sub>
Context Pointer CP:	<b>FA00<sub>H</sub></b>	Register STKOV:	<b>F600<sub>H</sub></b>
Stack Pointer SP:	<b>FA40<sub>H</sub></b>	Register BUSCON0:	acc. to startup config.
Register S0CON:	<b>8011<sub>H</sub></b>	P3.10/TXD0:	'1'
Register S0BG:	acc. to '00' byte	DP3.10:	'1'

Other than after a normal reset the watchdog timer is disabled, so the bootstrap loading sequence is not time limited. Pin TxD0 is configured as output, so the C161CS/JC/JI can return the identification byte.

*Note: Even if the internal ROM/OTP/Flash is enabled, no code can be executed out of it while the C161CS/JC/JI is in BSL mode.*

<sup>1)</sup> The external host should not send the zero byte before the end of the BSL initialization time (see [Figure 16-1](#)) to make sure that it is correctly received.

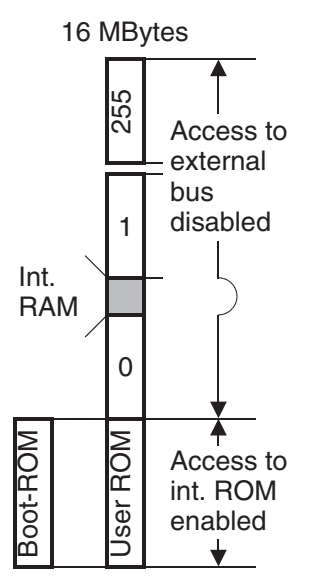
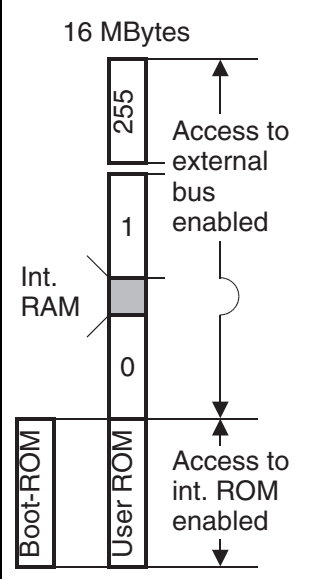
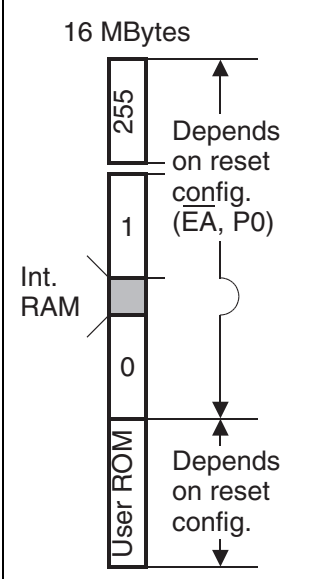


**Memory Configuration after Reset**

The configuration (i.e. the accessibility) of the C161CS/JC/JI's memory areas after reset in bootstrap loader mode differs from the standard case. Pin  $\overline{EA}$  does not select the code source in BSL mode, and accesses to the internal code memory are partly redirected, while the C161CS/JC/JI is in BSL mode (see [Table 16-1](#)). All code fetches are made from the special Boot-ROM, while data accesses read from the internal code memory. Data accesses will return undefined values on ROMless devices.

*Note: The code in the Boot-ROM is not an invariant feature of the C161CS/JC/JI. User software should not try to execute code from the internal ROM area while the BSL mode is still active, as these fetches will be redirected to the Boot-ROM.  
The Boot-ROM will also "move" to segment 1, when the internal ROM area is mapped to segment 1.*

**Table 16-1 BSL Memory Configurations**

	 <p>MCA04383</p>	 <p>MCA04384</p>	 <p>MCA04385</p>
BSL mode active	<b>Yes</b>	<b>Yes</b>	<b>No</b>
$\overline{EA}$ pin	high	low	acc. to application
Code fetch from internal ROM area	Boot-ROM access	Boot-ROM access	User ROM access
Data fetch from internal ROM area	User ROM access	User ROM access	User ROM access

## 16.2 Loading the Startup Code

After sending the identification byte the BSL enters a loop to receive 32 Bytes via ASC0. These bytes are stored sequentially into locations 00'FA40<sub>H</sub> through 00'FA5F<sub>H</sub> of the internal RAM. So up to 16 instructions may be placed into the RAM area. To execute the loaded code the BSL then jumps to location 00'FA40<sub>H</sub>, i.e. the first loaded instruction. The bootstrap loading sequence is now terminated, the C161CS/JC/JI remains in BSL mode, however. Most probably the initially loaded routine will load additional code or data, as an average application is likely to require substantially more than 16 instructions. This second receive loop may directly use the pre-initialized interface ASC0 to receive data and store it to arbitrary user-defined locations.

This second level of loaded code may be the final application code. It may also be another, more sophisticated, loader routine that adds a transmission protocol to enhance the integrity of the loaded code or data. It may also contain a code sequence to change the system configuration and enable the bus interface to store the received data into external memory.

This process may go through several iterations or may directly execute the final application. In all cases the C161CS/JC/JI will still run in BSL mode, i.e. with the watchdog timer disabled and limited access to the internal code memory. All code fetches from the internal ROM area (00'0000<sub>H</sub> ... 00'7FFF<sub>H</sub> or 01'0000<sub>H</sub> ... 01'7FFF<sub>H</sub>, if mapped to segment 1) are redirected to the special Boot-ROM. Data fetches access will access the internal code memory of the C161CS/JC/JI, if any is available, but will return undefined data on ROMless devices.

*Note: Data fetches from a protected ROM will not be executed.*

## 16.3 Exiting Bootstrap Loader Mode

In order to execute a program in normal mode (i.e. watchdog timer active, full access to user memory, etc.), the BSL mode must be terminated first. The C161CS/JC/JI exits BSL mode in two ways:

- upon a software reset, ignoring the external configuration (P0L.4 or  $\overline{RD}$ )
- upon a hardware reset, not configuring BSL mode.

After the (non-BSL) reset the C161CS/JC/JI will start executing out of user memory as externally configured via PORT0 or  $\overline{RD}/ALE$  (depending on  $\overline{EA}$ ).

## 16.4 Choosing the Baudrate for the BSL

The calculation of the serial baudrate for ASC0 from the length of the first zero byte that is received, allows the operation of the bootstrap loader of the C161CS/JC/JI with a wide range of baudrates. However, the upper and lower limits have to be kept, in order to ensure proper data transfer.

$$B_{C161CS/JC/JI} = \frac{f_{CPU}}{32 \times (S0BRL + 1)}$$

The C161CS/JC/JI uses timer T6 to measure the length of the initial zero byte. The quantization uncertainty of this measurement implies the first deviation from the real baudrate, the next deviation is implied by the computation of the S0BRL reload value from the timer contents. The formula below shows the association:

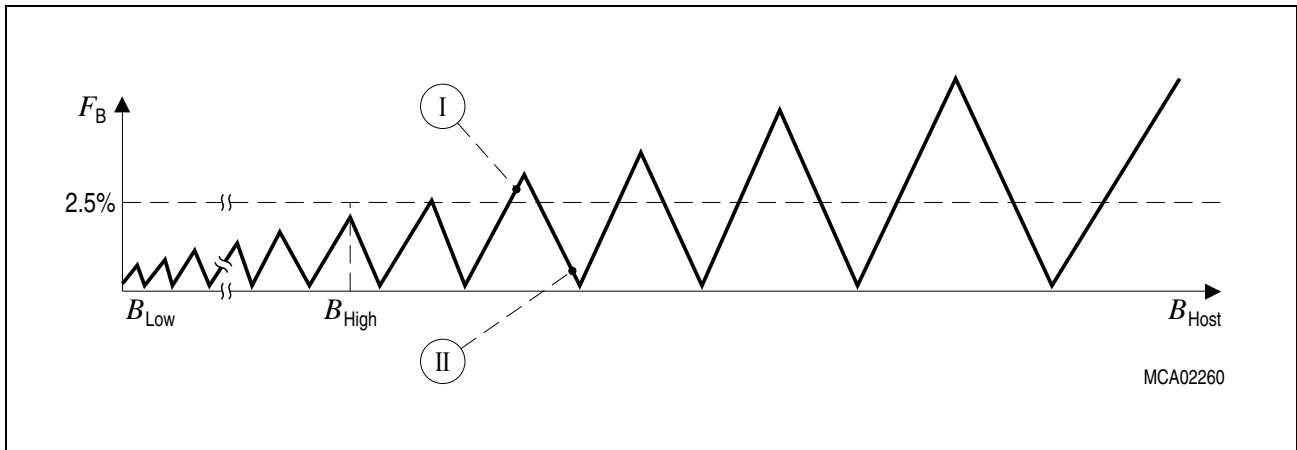
$$S0BRL = \frac{T6 - 36}{72} \quad , \quad T6 = \frac{9}{4} \times \frac{f_{CPU}}{B_{Host}}$$

For a correct data transfer from the host to the C161CS/JC/JI the maximum deviation between the internal initialized baudrate for ASC0 and the real baudrate of the host should be below 2.5%. The deviation ( $F_B$ , in percent) between host baudrate and C161CS/JC/JI baudrate can be calculated via the formula below:

$$F_B = \left| \frac{B_{Contr} - B_{Host}}{B_{Contr}} \right| \times 100\% \quad , \quad F_B \leq 2,5\%$$

*Note: Function ( $F_B$ ) does not consider the tolerances of oscillators and other devices supporting the serial communication.*

This baudrate deviation is a nonlinear function depending on the CPU clock and the baudrate of the host. The maxima of the function ( $F_B$ ) increase with the host baudrate due to the smaller baudrate prescaler factors and the implied higher quantization error (see [Figure 16-3](#)).



**Figure 16-3 Baudrate Deviation between Host and C161CS/JC/JI**

**The minimum baudrate** ( $B_{Low}$  in [Figure 16-3](#)) is determined by the maximum count capacity of timer T6, when measuring the zero byte, i.e. it depends on the CPU clock. The minimum baudrate is obtained by using the maximum T6 count  $2^{16}$  in the baudrate formula. Baudrates below  $B_{Low}$  would cause T6 to overflow. In this case ASC0 cannot be initialized properly and the communication with the external host is likely to fail.

**The maximum baudrate** ( $B_{High}$  in [Figure 16-3](#)) is the highest baudrate where the deviation still does not exceed the limit, i.e. all baudrates between  $B_{Low}$  and  $B_{High}$  are below the deviation limit.  $B_{High}$  marks the baudrate up to which communication with the external host will work properly without additional tests or investigations.

**Higher baudrates**, however, may be used as long as the actual deviation does not exceed the indicated limit. A certain baudrate (marked I) in [Figure 16-3](#) may e.g. violate the deviation limit, while an even higher baudrate (marked II) in [Figure 16-3](#) stays very well below it. Any baudrate can be used for the bootstrap loader provided that the following three prerequisites are fulfilled:

- the baudrate is within the specified operating range for the ASC0
- the external host is able to use this baudrate
- the computed deviation error is below the limit.

**Table 16-2 Bootstrap Loader Baudrate Ranges**

$f_{CPU}$ [MHz]	10	12	16	20	25	33
$B_{MAX}$	312,500	375,000	500,000	625,000	781,250	1,031,250
$B_{High}$	9,600	19,200	19,200	19,200	38,400	38,400
$B_{STDmin}$	600	600	600	1,200	1,200	1,200
$B_{Low}$	344	412	550	687	859	1,133

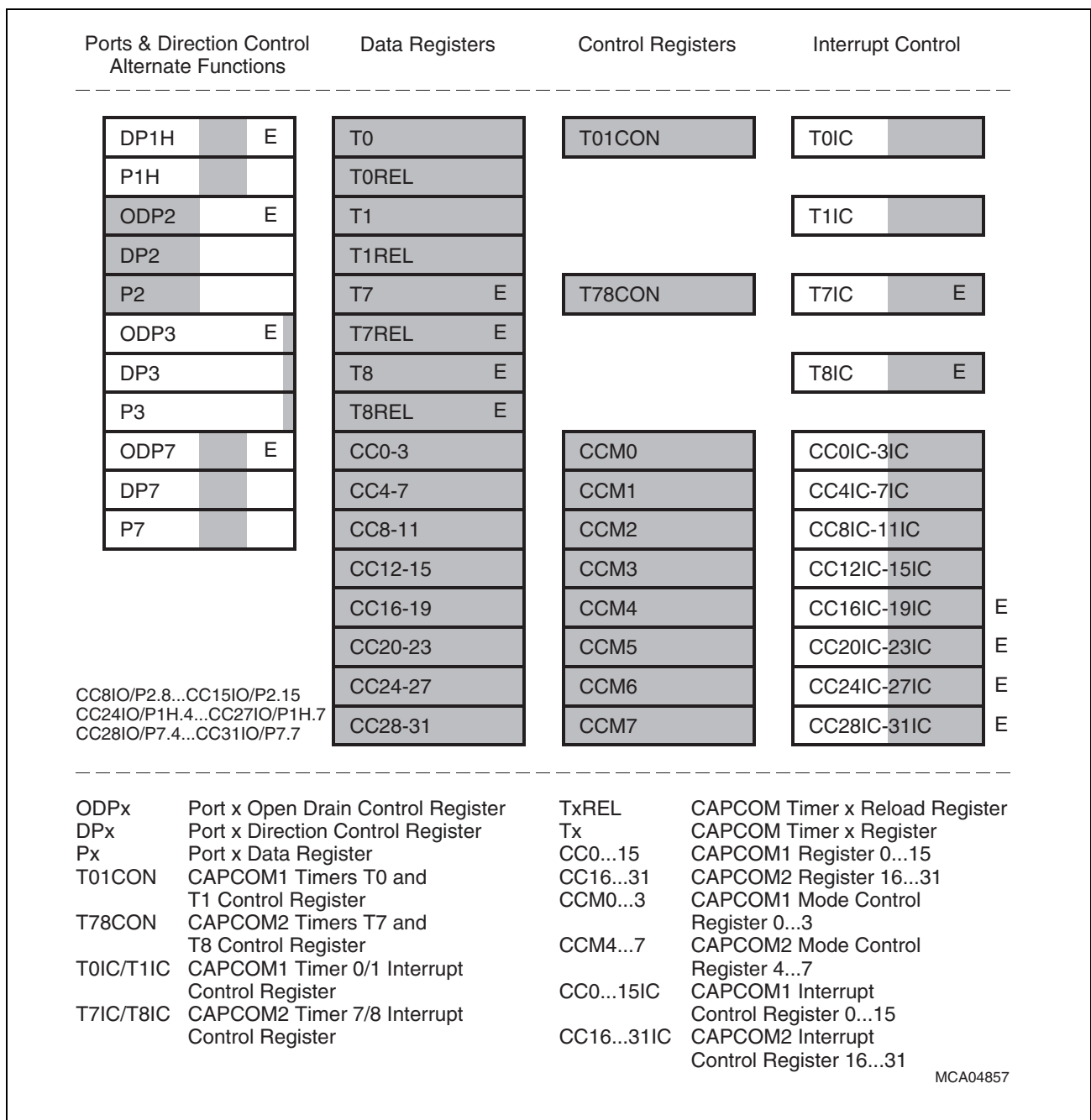
---

**The Bootstrap Loader**

*Note: When the bootstrap loader mode is entered via an internal reset ( $\overline{EA} = '1'$ ), the default configuration selects the prescaler for clock generation. In this case the bootstrap loader will begin to operate with  $f_{CPU} = f_{OSC} / 2$  which will limit the maximum baudrate for ASC0 at low input frequencies intended for PLL operation. Higher levels of the bootstrapping sequence can then switch the clock generation mode (via register RSTCON) e.g. to PLL in order to achieve higher baudrates for the download.*

## 17 The Capture/Compare Units

The C161CS/JC/JI provides two almost identical Capture/Compare (CAPCOM) units which only differ in the way they are connected to the C161CS/JC/JI's IO pins. They provide 32 channels (16 IO pins) which interact with 4 timers. The CAPCOM units can **capture** the contents of a timer on specific internal or external events, or they can **compare** a timer's content with given values and modify output signals in case of a match. With this mechanism they support generation and control of timing sequences on up to 16 channels per unit with a minimum of software intervention.



**Figure 17-1 SFRs and Port Pins Associated with the CAPCOM Units**

---

**The Capture/Compare Units**

From the programmer's point of view, the term 'CAPCOM unit' refers to a set of SFRs which are associated with this peripheral, including the port pins which may be used for alternate input/output functions including their direction control bits.

A CAPCOM unit is typically used to handle high speed IO tasks such as pulse and waveform generation, pulse width modulation, or recording of the time at which specific events occur. It also allows the implementation of up to 16 software timers. The maximum resolution of the CAPCOM units is 8 CPU clock cycles (=16 TCL).

Each CAPCOM unit consists of two 16-bit timers (T0/T1 in CAPCOM1, T7/T8 in CAPCOM2), each with its own reload register (TxREL), and a bank of sixteen dual purpose 16-bit capture/compare registers (CC0 through CC15 in CAPCOM1, CC16 through CC31 in CAPCOM2).

The input clock for the CAPCOM timers is programmable to several prescaled values of the CPU clock, or it can be derived from an overflow/underflow of timer T6 in block GPT2. T0 and T7 may also operate in counter mode (from an external input) where they can be clocked by external events.

Each capture/compare register may be programmed individually for capture or compare function, and each register may be allocated to either timer of the associated unit. Eight capture/compare registers of each module have one port pin associated with it, respectively, which serves as an input pin for the capture function or as an output pin for the compare function. The capture function causes the current timer contents to be latched into the respective capture/compare register triggered by an event (transition) on its associated port pin. The compare function may cause an output signal transition on that port pin whose associated capture/compare register matches the current timer contents. Specific interrupt requests are generated upon each capture/compare event or upon timer overflow.

**Figure 17-2** shows the basic structure of the two CAPCOM units.

The Capture/Compare Units

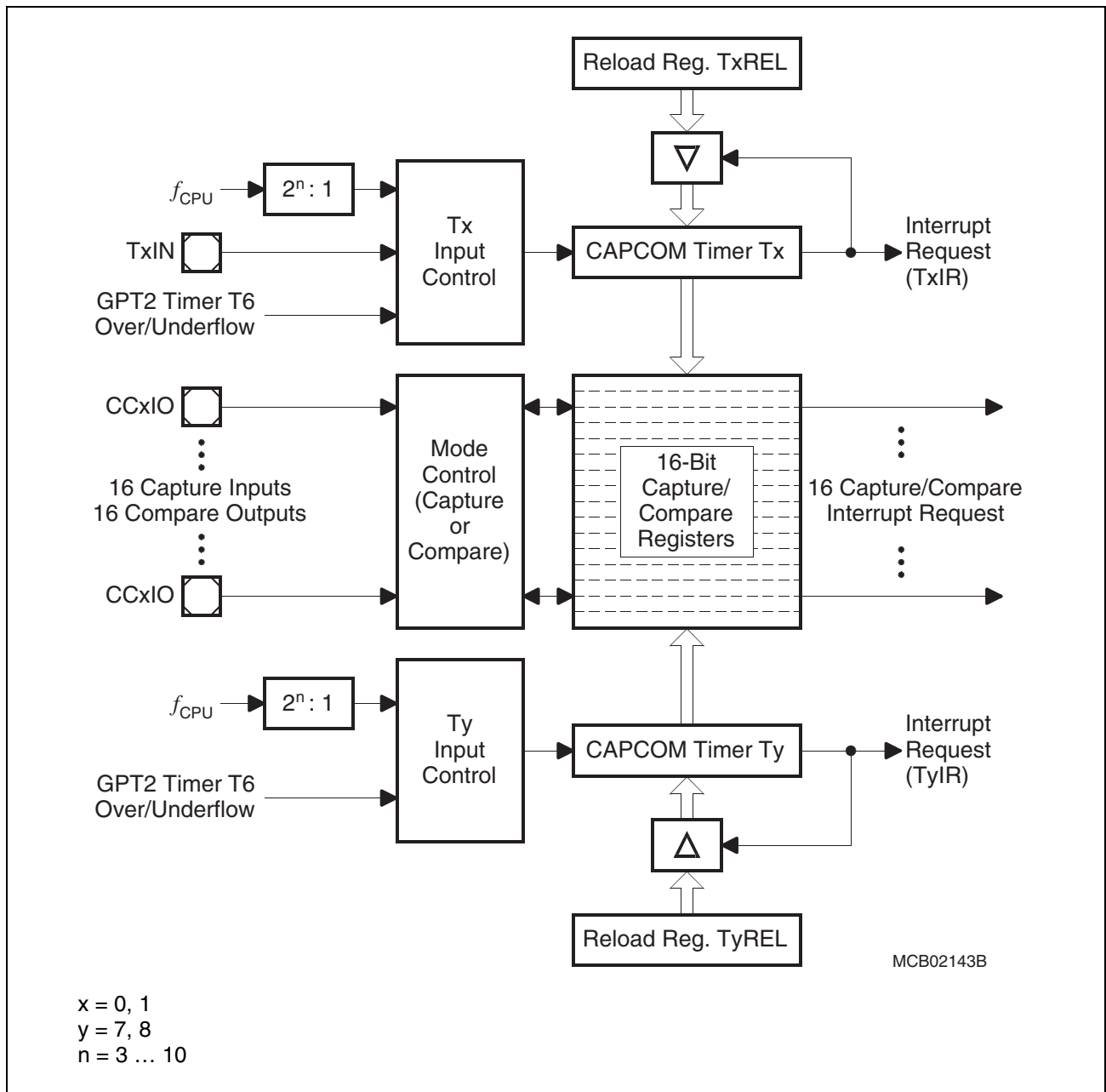


Figure 17-2 CAPCOM Unit Block Diagram

Table 17-1 CAPCOM Channel Port Connections

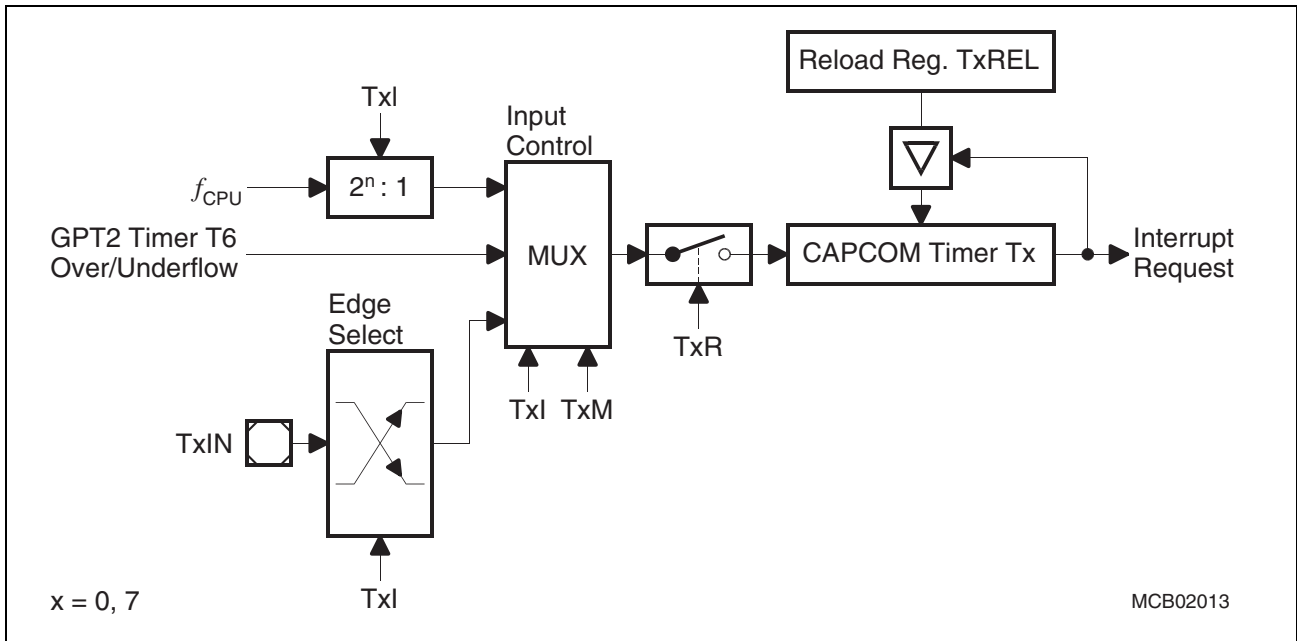
Unit	Channel	Port	Capture	Compare
CAPCOM1	CC8IO ... CC15IO	P2.8 ... P2.15	Input	Output
CAPCOM2	CC24IO ... CC27IO	P1H.4 ... P1H.7	Input	Output
	CC28IO ... CC31IO	P7.4 ... P7.7	Input	Output
	$\Sigma = 16$	$\Sigma = 16$	$\Sigma = 16$	$\Sigma = 16$



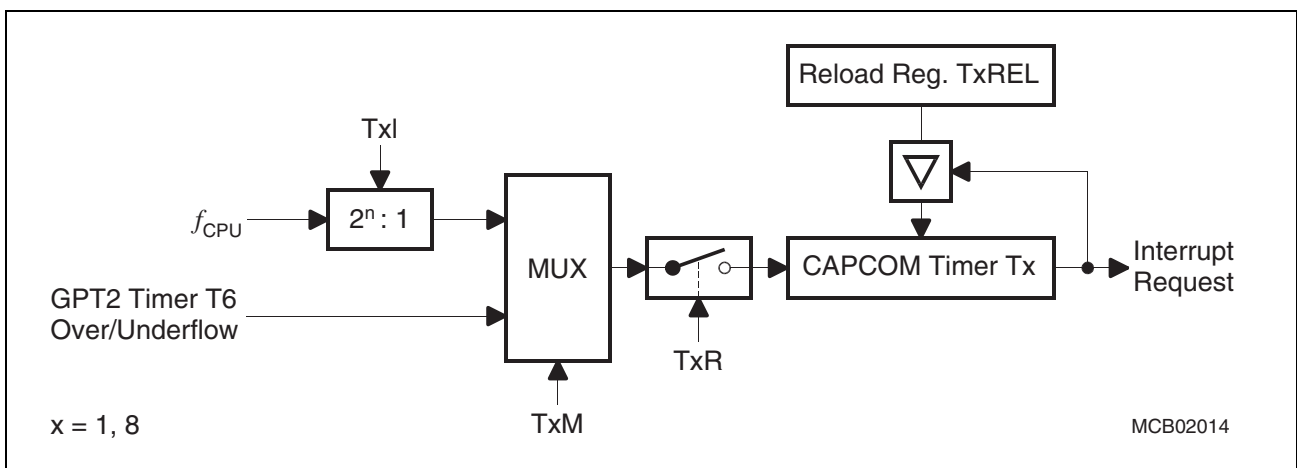
## 17.1 The CAPCOM Timers

The primary use of the timers T0/T1 and T7/T8 is to provide two independent time bases (16 TCL maximum resolution) for the capture/compare registers of each unit, but they may also be used independent of the capture/compare registers.

The basic structure of the four timers is identical, while the selection of input signals is different for timers T0/T7 and timers T1/T8 (see [Figure 17-3](#) and [Figure 17-4](#)).



**Figure 17-3 Block Diagram of CAPCOM Timers T0 and T7**



**Figure 17-4 Block Diagram of CAPCOM Timers T1 and T8**

*Note: When an external input signal is connected to the input lines of both T0 and T7, these timers count the input signal synchronously. Thus the two timers can be regarded as one timer whose contents can be compared with 32 capture registers.*

**The Capture/Compare Units**

The functions of the CAPCOM timers are controlled via the bitaddressable 16-bit control registers T01CON and T78CON. The high-byte of T01CON controls T1, the low-byte of T01CON controls T0, the high-byte of T78CON controls T8, the low-byte of T78CON controls T7. The control options are identical for all four timers (except for external input).

**T01CON**

**CAPCOM Timer 0/1 Ctrl. Reg. SFR (FF50<sub>H</sub>/A8<sub>H</sub>) Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	<b>T1R</b>	-	-	<b>T1M</b>		<b>T1I</b>		-	<b>T0R</b>	-	-	<b>T0M</b>		<b>T0I</b>	
-	rw	-	-	rw		rw		-	rw	-	-	rw		rw	

**T78CON**

**CAPCOM Timer 7/8 Ctrl. Reg. SFR (FF20<sub>H</sub>/90<sub>H</sub>) Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	<b>T8R</b>	-	-	<b>T8M</b>		<b>T8I</b>		-	<b>T7R</b>	-	-	<b>T7M</b>		<b>T7I</b>	
-	rw	-	-	rw		rw		-	rw	-	-	rw		rw	

Bit	Function
<b>Txl</b>	<p><b>Timer/Counter x Input Selection</b></p> <p>Timer Mode (TxM = '0'): Input Frequency = <math>f_{CPU} / 2^{(&lt;TxI&gt; + 3)}</math> See also <a href="#">Table 17-2</a> - <a href="#">Table 17-4</a> for examples.</p> <p>Counter Mode (TxM = '1'): 000 Overflow/Underflow of GPT2 Timer 6 001 Positive (rising) edge on pin T7IN<sup>1)</sup> 010 Negative (falling) edge on pin T7IN<sup>1)</sup> 011 Any edge (rising and falling) on pin T7IN<sup>1)</sup> 1XX Reserved</p>
<b>TxM</b>	<p><b>Timer/Counter x Mode Selection</b></p> <p>0: Timer Mode (Input derived from internal clock) 1: Counter Mode (Input from External Input or T6)</p>
<b>TxR</b>	<p><b>Timer/Counter x Run Control</b></p> <p>0: Timer/Counter x is disabled 1: Timer/Counter x is enabled</p>

<sup>1)</sup> This selection is available for timers T0 and T7. Timers T1 and T8 will stop at this selection!

## The Capture/Compare Units

The timer run flags T0R, T1R, T7R, and T8R allow for enabling and disabling the timers. The following description of the timer modes and operation always applies to the enabled state of the timers, i.e. the respective run flag is assumed to be set to '1'.

In all modes, the timers are always counting upward. The current timer values are accessible for the CPU in the timer registers Tx, which are non-bitaddressable SFRs. When the CPU writes to a register Tx in the state immediately before the respective timer increment or reload is to be performed, the CPU write operation has priority and the increment or reload is disabled to guarantee correct timer operation.

### Timer Mode

The bits TxM in SFRs T01CON and T78CON select between timer or counter mode for the respective timer. In timer mode (TxM = '0'), the input clock for a timer is derived from the internal CPU clock divided by a programmable prescaler. The different options for the prescaler are selected separately for each timer by the bit fields TxI.

The input frequencies  $f_{Tx}$  for Tx are determined as a function of the CPU clock as follows, where <TxI> represents the contents of the bit field TxI:

$$f_{Tx} = \frac{f_{CPU}}{2^{(<TxI> + 3)}}$$

When a timer overflows from FFFF<sub>H</sub> to 0000<sub>H</sub> it is reloaded with the value stored in its respective reload register TxREL. The reload value determines the period P<sub>Tx</sub> between two consecutive overflows of Tx as follows:

$$P_{Tx} = \frac{(2^{16} - <TxREL>) \times 2^{(<TxI> + 3)}}{f_{CPU}}$$

After a timer has been started by setting its run flag (TxR) to '1', the first increment will occur within the time interval which is defined by the selected timer resolution. All further increments occur exactly after the time defined by the timer resolution.

When both timers of a CAPCOM unit are to be incremented or reloaded at the same time T0 is always serviced one CPU clock before T1, T7 before T8, respectively.

The timer input frequencies, resolution and periods which result from the selected prescaler option in TxI when using a certain CPU clock are listed in [Table 17-2](#) - [Table 17-4](#). The numbers for the timer periods are based on a reload value of 0000<sub>H</sub>. Note that some numbers may be rounded to 3 significant digits.

**The Capture/Compare Units**
**Table 17-2 Timer Input Frequencies, Resolution and Period @ 20 MHz**

$f_{CPU} = 20 \text{ MHz}$	Timer Input Selection Tx1							
	000 <sub>B</sub>	001 <sub>B</sub>	010 <sub>B</sub>	011 <sub>B</sub>	100 <sub>B</sub>	101 <sub>B</sub>	110 <sub>B</sub>	111 <sub>B</sub>
<b>Prescaler (1:N)</b>	8	16	32	64	128	256	512	1024
<b>Input Frequency</b>	2.5 MHz	1.25 MHz	625 kHz	312.5 kHz	156.25 kHz	78.125 kHz	39.06 kHz	19.53 kHz
<b>Resolution</b>	400 ns	800 ns	1.6 $\mu\text{s}$	3.2 $\mu\text{s}$	6.4 $\mu\text{s}$	12.8 $\mu\text{s}$	25.6 $\mu\text{s}$	51.2 $\mu\text{s}$
<b>Period</b>	26 ms	52.5 ms	105 ms	210 ms	420 ms	840 ms	1.68 s	3.36 s

**Table 17-3 Timer Input Frequencies, Resolution and Period @ 25 MHz**

$f_{CPU} = 25 \text{ MHz}$	Timer Input Selection Tx1							
	000 <sub>B</sub>	001 <sub>B</sub>	010 <sub>B</sub>	011 <sub>B</sub>	100 <sub>B</sub>	101 <sub>B</sub>	110 <sub>B</sub>	111 <sub>B</sub>
<b>Prescaler (1:N)</b>	8	16	32	64	128	256	512	1024
<b>Input Frequency</b>	3.125 MHz	1.563 MHz	781.25 kHz	390.63 kHz	195.31 kHz	97.656 kHz	48.828 kHz	24.414 kHz
<b>Resolution</b>	320 ns	640 ns	1.28 $\mu\text{s}$	2.56 $\mu\text{s}$	5.12 $\mu\text{s}$	10.24 $\mu\text{s}$	20.48 $\mu\text{s}$	40.96 $\mu\text{s}$
<b>Period</b>	21 ms	42 ms	84 ms	168 ms	336 ms	672 ms	1.344 s	2.688 s

**Table 17-4 Timer Input Frequencies, Resolution and Period @ 33 MHz**

$f_{CPU} = 33 \text{ MHz}$	Timer Input Selection Tx1							
	000 <sub>B</sub>	001 <sub>B</sub>	010 <sub>B</sub>	011 <sub>B</sub>	100 <sub>B</sub>	101 <sub>B</sub>	110 <sub>B</sub>	111 <sub>B</sub>
<b>Prescaler (1:N)</b>	8	16	32	64	128	256	512	1024
<b>Input Frequency</b>	4.125 MHz	2.063 MHz	1.031 MHz	515.63 kHz	257.81 kHz	128.91 kHz	64.453 kHz	32.227 kHz
<b>Resolution</b>	242 ns	485 ns	970 ns	1.94 $\mu\text{s}$	3.88 $\mu\text{s}$	7.76 $\mu\text{s}$	15.52 $\mu\text{s}$	31.03 $\mu\text{s}$
<b>Period</b>	15.89 ms	31.78 ms	63.55 ms	127.10 ms	254.20 ms	508.40 ms	1.017 s	2.034 s

## Counter Mode

The bits TxM in SFRs T01CON and T78CON select between timer or counter mode for the respective timer. In Counter mode (TxM = '1') the input clock for a timer can be derived from the overflows/underflows of timer T6 in block GPT2. In addition, timers T0 and T7 can be clocked by external events. Either a positive, a negative, or both a positive and a negative transition at pin T0IN or T7IN (alternate port input function), respectively, can be selected to cause an increment of T0/T7.

When T1 or T8 is programmed to run in counter mode, bit field TxI is used to enable the overflows/underflows of timer T6 as the count source. This is the only option for these timers and it is selected by the combination TxI = 000<sub>B</sub>. When bit field TxI is programmed to any other valid combination, the respective timer will stop.

When T0 or T7 is programmed to run in counter mode, bit field TxI is used to select the count source and transition (if the source is the input pin) which should cause a count trigger (see description of TxyCON for the possible selections).

*Note: In order to use pin T0IN or T7IN as external count input pin, the respective port pin must be configured as input, i.e. the corresponding direction control bit must be cleared (DPx.y = '0').*

*If the respective port pin is configured as output, the associated timer may be clocked by modifying the port output latches Px.y via software, e.g. for testing purposes.*

The maximum external input frequency to T0 or T7 in counter mode is  $f_{\text{CPU}}/16$ . To ensure that a signal transition is properly recognized at the timer input, an external count input signal should be held for at least 8 CPU clock cycles before it changes its level again. The incremented count value appears in SFR T0/T7 within 8 CPU clock cycles after the signal transition at pin TxIN.

## Reload

A reload of a timer with the 16-bit value stored in its associated reload register in both modes is performed each time a timer would overflow from FFFF<sub>H</sub> to 0000<sub>H</sub>. In this case the timer does not wrap around to 0000<sub>H</sub>, but rather is reloaded with the contents of the respective reload register TxREL. The timer then resumes incrementing starting from the reloaded value.

The reload registers TxREL are not bitaddressable.

## 17.2 CAPCOM Unit Timer Interrupts

Upon a timer overflow the corresponding timer interrupt request flag TxIR for the respective timer will be set. This flag can be used to generate an interrupt or trigger a PEC service request, when enabled by the respective interrupt enable bit TxIE.

Each timer has its own bitaddressable interrupt control register (TxIC) and its own interrupt vector (TxINT). The organization of the interrupt control registers TxIC is identical with the other interrupt control registers.

### T0IC

**CAPCOM T0 Intr. Ctrl. Reg.**      **SFR (FF9C<sub>H</sub>/CE<sub>H</sub>)**      **Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								T0IR	T0IE	ILVL			GLVL		
								rwh	rw	rw			rw		

### T1IC

**CAPCOM T1 Intr. Ctrl. Reg.**      **SFR (FF9E<sub>H</sub>/CF<sub>H</sub>)**      **Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								T1IR	T1IE	ILVL			GLVL		
								rwh	rw	rw			rw		

### T7IC

**CAPCOM T7 Intr. Ctrl. Reg.**      **ESFR (F17A<sub>H</sub>/BE<sub>H</sub>)**      **Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								T7IR	T7IE	ILVL			GLVL		
								rwh	rw	rw			rw		

### T8IC

**CAPCOM T8 Intr. Ctrl. Reg.**      **ESFR (F17C<sub>H</sub>/BF<sub>H</sub>)**      **Reset Value: - - 00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								T8IR	T8IE	ILVL			GLVL		
								rwh	rw	rw			rw		

*Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.*

### 17.3 Capture/Compare Registers

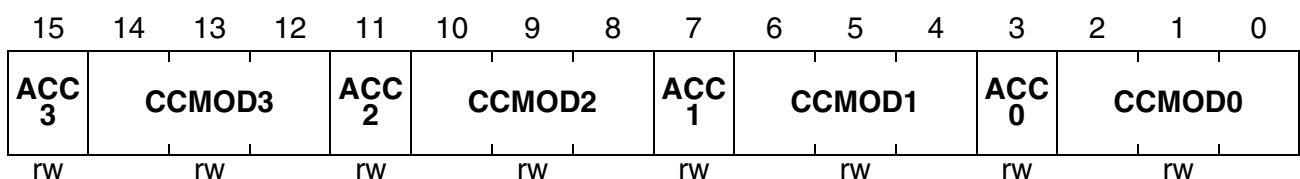
The 16-bit capture/compare registers CC0 through CC31 are used as data registers for capture or compare operations with respect to timers T0/T1 and T7/T8. The capture/compare registers are not bitaddressable.

The functions of the 32 capture/compare registers are controlled by 8 bitaddressable 16-bit mode control registers named CCM0 ... CCM7 which are organized identically (see description below). Each register contains bits for mode selection and timer allocation of four capture/compare registers.

#### Capture/Compare Mode Registers for the CAPCOM1 Unit (CC0 ... CC15)

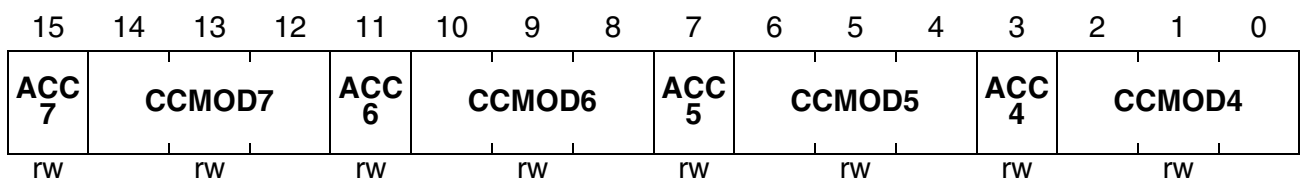
##### CCM0

**CAPCOM Mode Ctrl. Reg. 0      SFR (FF52<sub>H</sub>/A9<sub>H</sub>)      Reset Value: 0000<sub>H</sub>**



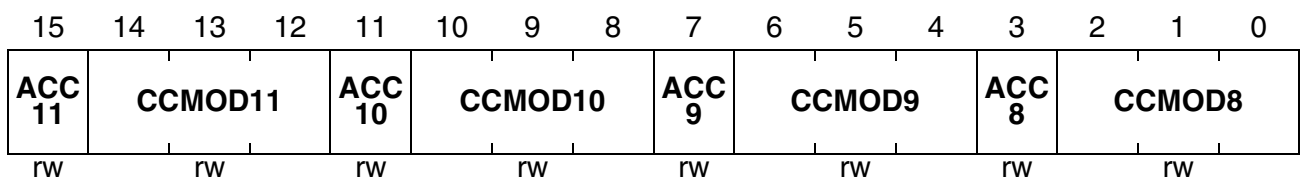
##### CCM1

**CAPCOM Mode Ctrl. Reg. 1      SFR (FF54<sub>H</sub>/AA<sub>H</sub>)      Reset Value: 0000<sub>H</sub>**



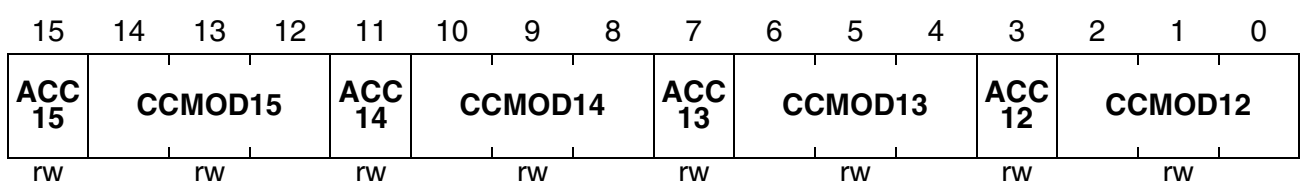
##### CCM2

**CAPCOM Mode Ctrl. Reg. 2      SFR (FF56<sub>H</sub>/AB<sub>H</sub>)      Reset Value: 0000<sub>H</sub>**



##### CCM3

**CAPCOM Mode Ctrl. Reg. 3      SFR (FF58<sub>H</sub>/AC<sub>H</sub>)      Reset Value: 0000<sub>H</sub>**



The Capture/Compare Units

Capture/Compare Mode Registers for the CAPCOM2 Unit (CC16 ... CC32)

**CCM4**

CAPCOM Mode Ctrl. Reg. 4      SFR (FF22<sub>H</sub>/91<sub>H</sub>)      Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC 19	CCMOD19		ACC 18	CCMOD18		ACC 17	CCMOD17		ACC 16	CCMOD16					
rw	rw		rw	rw		rw	rw		rw	rw					

**CCM5**

CAPCOM Mode Ctrl. Reg. 5      SFR (FF24<sub>H</sub>/92<sub>H</sub>)      Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC 23	CCMOD23		ACC 22	CCMOD22		ACC 21	CCMOD21		ACC 20	CCMOD20					
rw	rw		rw	rw		rw	rw		rw	rw					

**CCM6**

CAPCOM Mode Ctrl. Reg. 6      SFR (FF26<sub>H</sub>/93<sub>H</sub>)      Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC 27	CCMOD27		ACC 26	CCMOD26		ACC 25	CCMOD25		ACC 24	CCMOD24					
rw	rw		rw	rw		rw	rw		rw	rw					

**CCM7**

CAPCOM Mode Ctrl. Reg. 7      SFR (FF28<sub>H</sub>/94<sub>H</sub>)      Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC 31	CCMOD31		ACC 30	CCMOD30		ACC 29	CCMOD29		ACC 28	CCMOD28					
rw	rw		rw	rw		rw	rw		rw	rw					

Bit	Function
CCMODx	Mode Selection for Capture/Compare Register CCx The available capture/compare modes are listed in <a href="#">Table 17-5</a> .
ACCx	Allocation Bit for Capture/Compare Register CCx 0: CCx allocated to Timer T7 1: CCx allocated to Timer T8



**The Capture/Compare Units**

Each of the registers CCx may be individually programmed for capture mode or one of 4 different compare modes, and may be allocated individually to one of the two timers of the respective CAPCOM unit (T0 or T1, and T7 or T8, respectively). A special combination of compare modes additionally allows the implementation of a ‘double-register’ compare mode. When capture or compare operation is disabled for one of the CCx registers, it may be used for general purpose variable storage.

**Table 17-5 Selection of Capture Modes and Compare Modes**

<b>CCMODx</b>	<b>Selected Operating Mode</b>
<b>0 0 0</b>	<b>Disable Capture and Compare Modes</b> The respective CAPCOM register may be used for general variable storage.
<b>0 0 1</b>	<b>Capture on Positive Transition (Rising Edge) at Pin CCxIO</b>
<b>0 1 0</b>	<b>Capture on Negative Transition (Falling Edge) at Pin CCxIO</b>
<b>0 1 1</b>	<b>Capture on Positive and Negative Transition (Both Edges) at Pin CCxIO</b>
<b>1 0 0</b>	<b>Compare Mode 0: Interrupt Only</b> Several interrupts per timer period.
<b>1 0 1</b>	<b>Compare Mode 1: Toggle Output Pin on each Match</b> Several compare events per timer period.
<b>1 1 0</b>	<b>Compare Mode 2: Interrupt Only</b> Only one interrupt per timer period.
<b>1 1 1</b>	<b>Compare Mode 3: Set Output Pin on each Match</b> Reset output pin on each timer overflow; Only one interrupt per timer period.

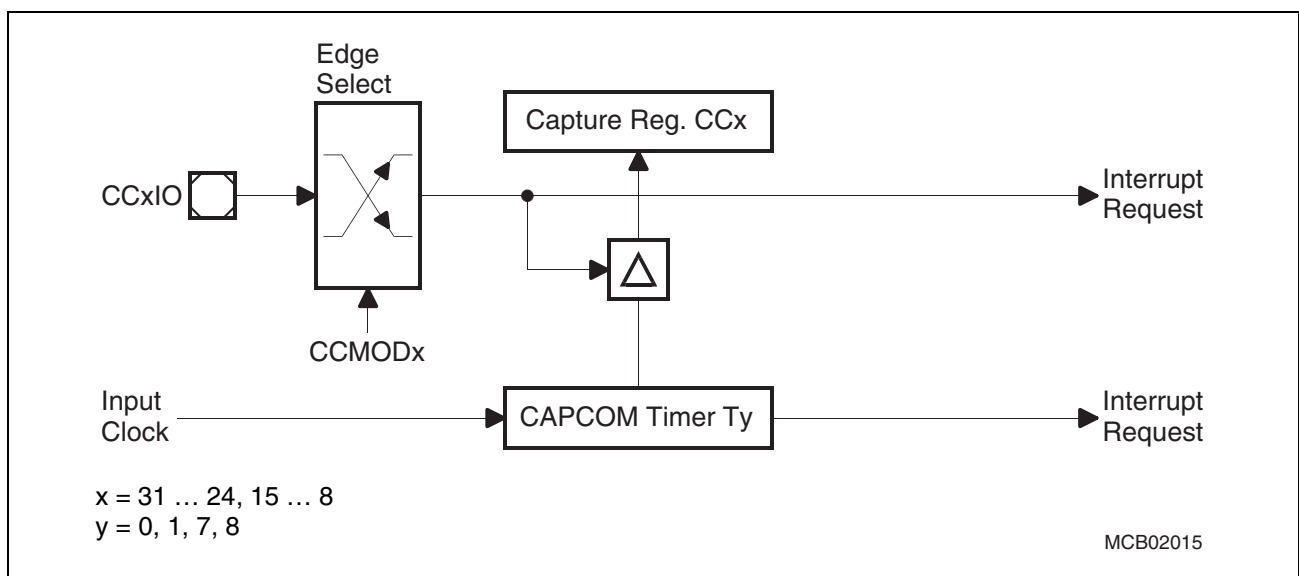
The detailed discussion of the capture and compare modes is valid for all the capture/compare channels, so registers, bits and pins are only referenced by the placeholder ‘x’.

*Note: A capture or compare event on channel 31 may be used to trigger a channel injection on the C161CS/JC/JI’s A/D converter if enabled.*

## 17.4 Capture Mode

In response to an external event the content of the associated timer (T0/T1 or T7/T8, depending on the used CAPCOM unit and the state of the allocation control bit ACCx) is latched into the respective capture register CCx. The external event causing a capture can be programmed to be either a positive, a negative, or both a positive or a negative transition at the respective external input pin CCxIO.

The triggering transition is selected by the mode bits CCMODx in the respective CAPCOM mode control register. In any case, the event causing a capture will also set the respective interrupt request flag CCxIR, which can cause an interrupt or a PEC service request, when enabled.



**Figure 17-5 Capture Mode Block Diagram**

In order to use the respective port pin as external capture input pin CCxIO for capture register CCx, this port pin must be configured as input, i.e. the corresponding direction control bit must be set to '0'. To ensure that a signal transition is properly recognized, an external capture input signal should be held for at least 8 CPU clock cycles before it changes its level.

During these 8 CPU clock cycles the capture input signals are scanned sequentially. When a timer is modified or incremented during this process, the new timer contents will already be captured for the remaining capture registers within the current scanning sequence.

*Note: When the timer modification can generate an overflow the capture interrupt routine should check if the timer overflow was serviced during these 8 CPU clock cycles.*

If pin CCxIO is configured as output, the capture function may be triggered by modifying the corresponding port output latch via software, e.g. for testing purposes.

## 17.5 Compare Modes

The compare modes allow triggering of events (interrupts and/or output signal transitions) with minimum software overhead. In all compare modes, the 16-bit value stored in compare register CCx (in the following also referred to as 'compare value') is continuously compared with the contents of the allocated timer (T0/T1 or T7/T8). If the current timer contents match the compare value, an appropriate output signal, which is based on the selected compare mode, can be generated at the corresponding output pin CCxIO and the associated interrupt request flag CCxIR is set, which can generate an interrupt request (if enabled).

As for capture mode, the compare registers are also processed sequentially during compare mode. When two or more compare registers are programmed to the same compare value, their corresponding interrupt request flags will be set to '1' and the selected output signals will be generated within 8 CPU clock cycles after the allocated timer is incremented to this compare value. Further compare events on the same compare value are disabled<sup>1)</sup> until the timer is incremented again or written to by software. After a reset, compare events for register CCx will only become enabled, if the allocated timer has been incremented or written to by software and one of the compare modes described in the following has been selected for this register.

The different compare modes which can be programmed for a given compare register CCx are selected by the mode control field CCMODx in the associated capture/compare mode control register. In the following, each of the compare modes is discussed in detail.

### Compare Mode 0

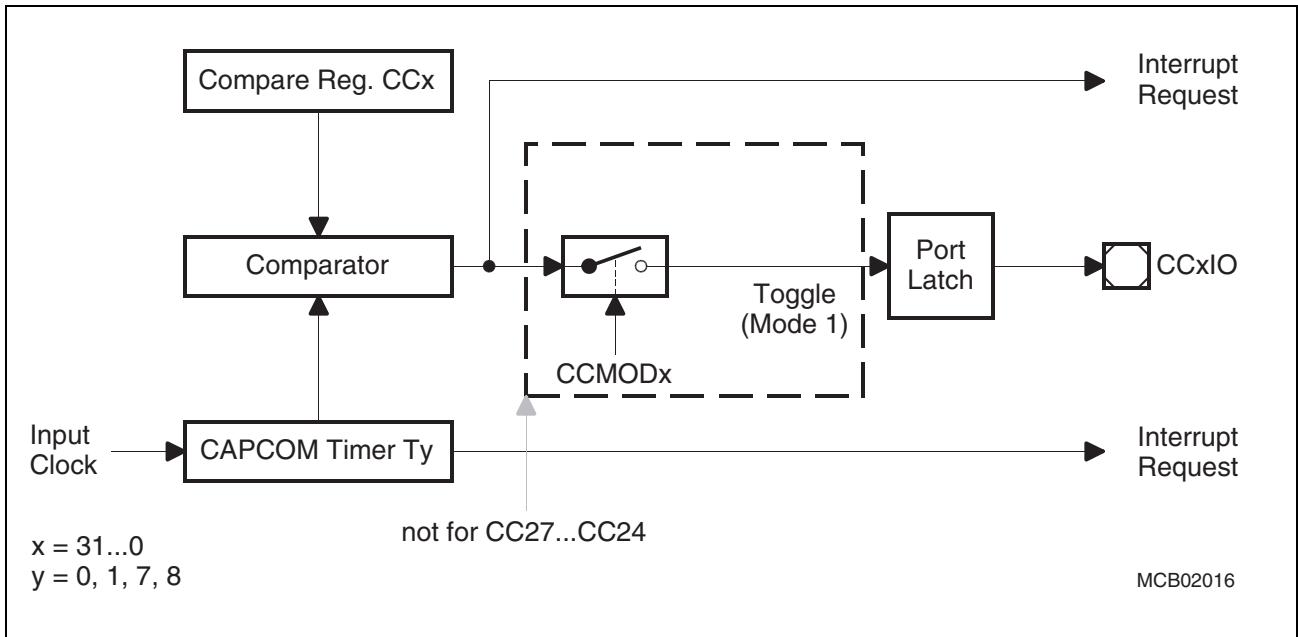
This is an interrupt-only mode which can be used for software timing purposes. Compare mode 0 is selected for a given compare register CCx by setting bit field CCMODx of the corresponding mode control register to '100<sub>B</sub>'.

In this mode, the interrupt request flag CCxIR is set each time a match is detected between the content of compare register CCx and the allocated timer. Several of these compare events are possible within a single timer period, when the compare value in register CCx is updated during the timer period. The corresponding port pin CCxIO is not affected by compare events in this mode and can be used as general purpose IO pin.

---

<sup>1)</sup> Compare events are detected sequentially, where a sequence (checking 8 times 2 channels each) takes 8 CPU clock cycles. Even if more sequences are executed before the timer increments (lower timer frequency) a given compare value only results in one single compare event.

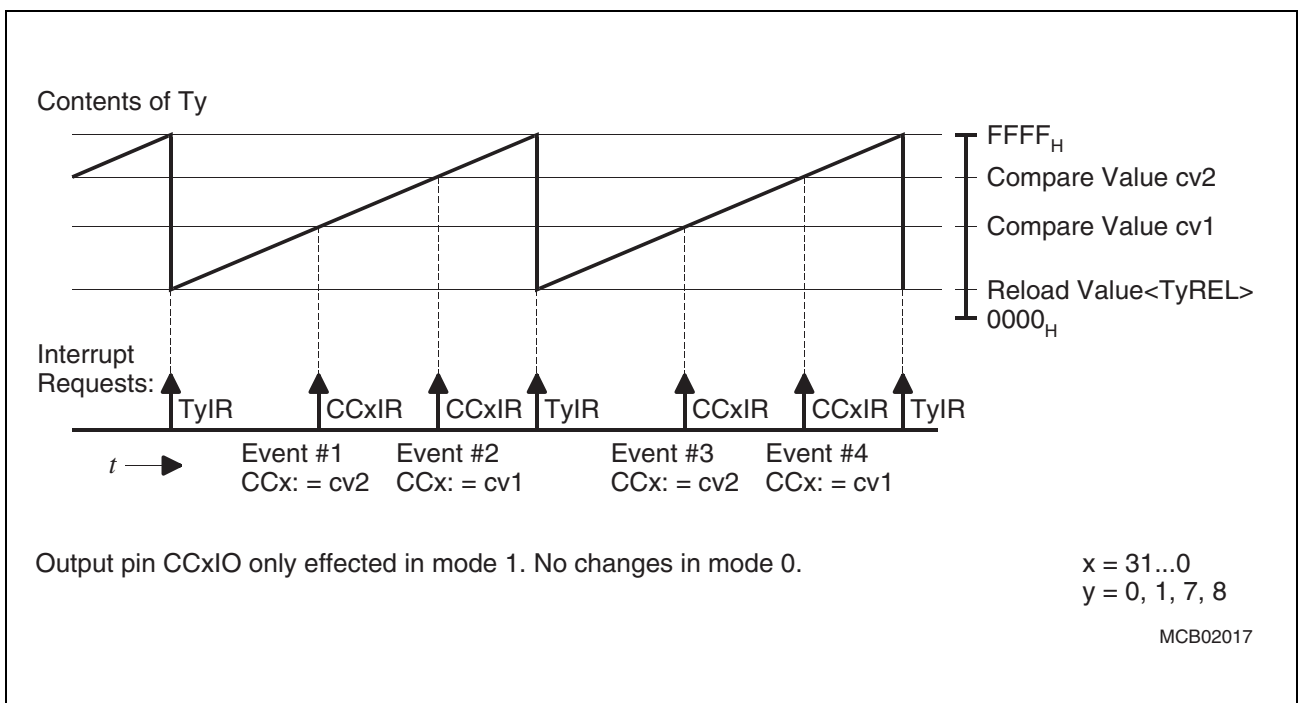
**The Capture/Compare Units**



**Figure 17-6 Compare Mode 0 and 1 Block Diagram**

*Note: The port latch and pin remain unaffected in compare mode 0.*

In the example below, the compare value in register CCx is modified from cv1 to cv2 after compare events #1 and #3, and from cv2 to cv1 after events #2 and #4, etc. This results in periodic interrupt requests from timer Ty, and in interrupt requests from register CCx which occur at the time specified by the user through cv1 and cv2.



**Figure 17-7 Timing Example for Compare Modes 0 and 1**

### Compare Mode 1

Compare mode 1 is selected for register CCx by setting bit field CCMODx of the corresponding mode control register to '101<sub>B</sub>'.

When a match between the content of the allocated timer and the compare value in register CCx is detected in this mode, interrupt request flag CCxIR is set to '1', and in addition the corresponding output pin CCxIO (alternate port output function) is toggled. For this purpose, the state of the respective port output latch (not the pin) is read, inverted, and then written back to the output latch.

Compare mode 1 allows several compare events within a single timer period. An overflow of the allocated timer has no effect on the output pin, nor does it disable or enable further compare events.

In order to use the respective port pin as compare signal output pin CCxIO for compare register CCx in compare mode 1, this port pin must be configured as output, i.e. the corresponding direction control bit must be set to '1'. With this configuration, the initial state of the output signal can be programmed or its state can be modified at any time by writing to the port output latch.

In compare mode 1 the port latch is toggled upon each compare event (see [Figure 17-7](#)).

*Note: If the port output latch is written to by software at the same time it would be altered by a compare event, the software write will have priority. In this case the hardware-triggered change will not become effective.*

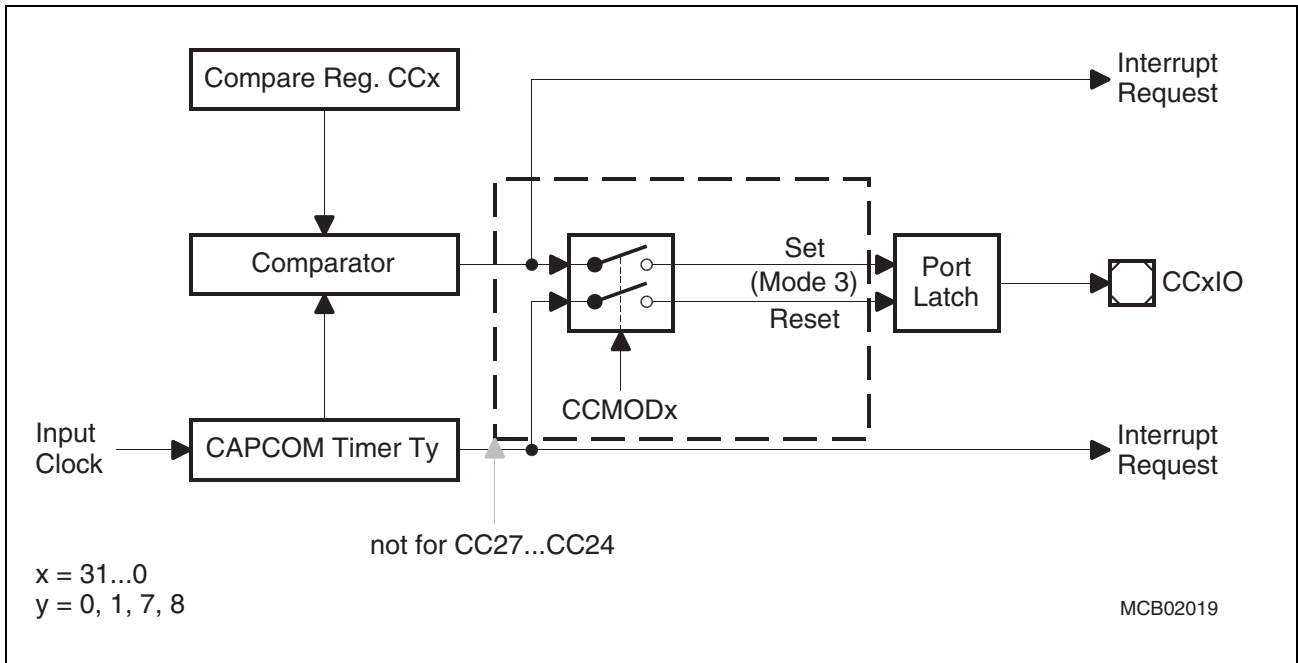
### Compare Mode 2

Compare mode 2 is an interrupt-only mode similar to compare mode 0, but only one interrupt request per timer period will be generated. Compare mode 2 is selected for register CCx by setting bit field CCMODx of the corresponding mode control register to '110<sub>B</sub>'.

When a match is detected in compare mode 2 for the first time within a timer period, the interrupt request flag CCxIR is set to '1'. The corresponding port pin is not affected and can be used for general purpose IO. However, after the first match has been detected in this mode, all further compare events within the same timer period are disabled for compare register CCx until the allocated timer overflows. This means, that after the first match, even when the compare register is reloaded with a value higher than the current timer value, no compare event will occur until the next timer period.

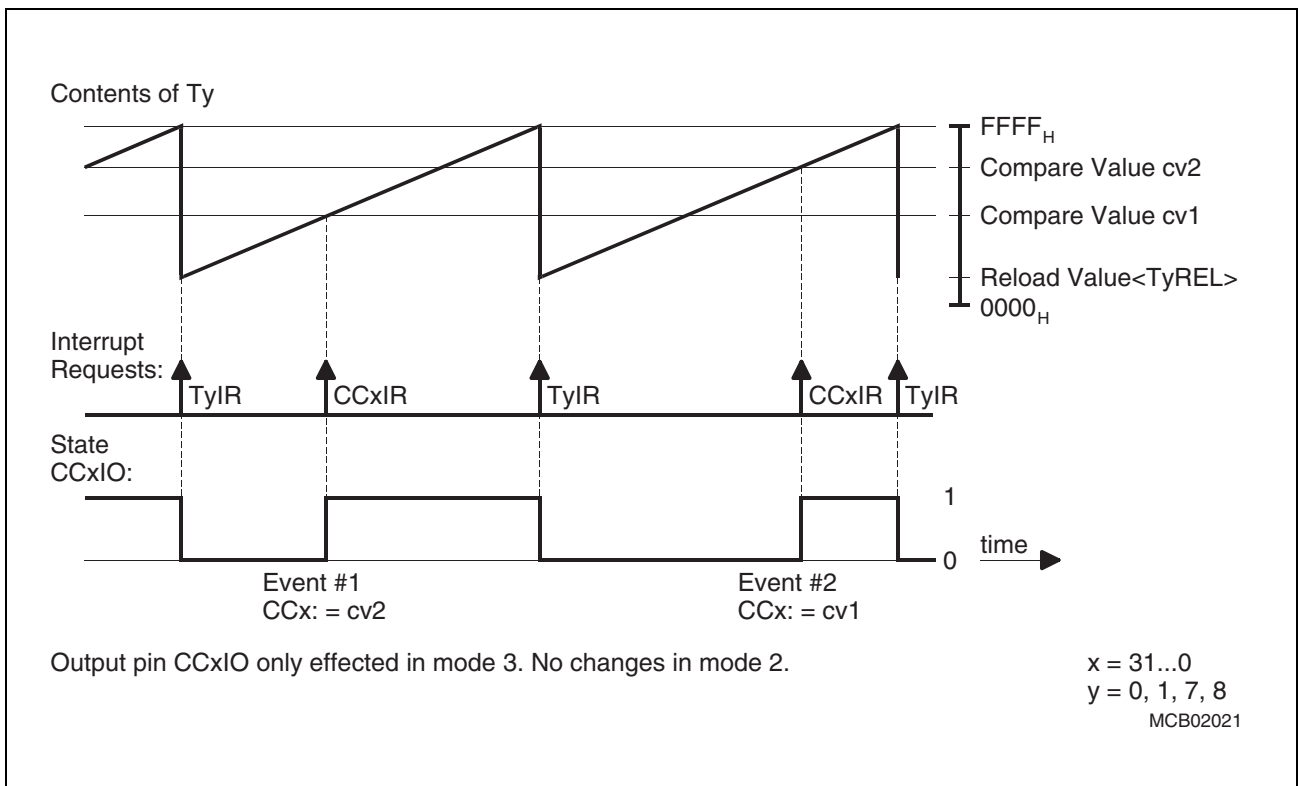
In the example below, the compare value in register CCx is modified from cv1 to cv2 after compare event #1. Compare event #2, however, will not occur until the next period of timer Ty.

The Capture/Compare Units



**Figure 17-8 Compare Mode 2 and 3 Block Diagram**

*Note: The port latch and pin remain unaffected in compare mode 2.*



**Figure 17-9 Timing Example for Compare Modes 2 and 3**

### **Compare Mode 3**

Compare mode 3 is selected for register CCx by setting bit field CCMODx of the corresponding mode control register to '111<sub>B</sub>'. In compare mode 3 only one compare event will be generated per timer period.

When the first match within the timer period is detected the interrupt request flag CCxIR is set to '1' and also the output pin CCxIO (alternate port function) will be set to '1'. The pin will be reset to '0', when the allocated timer overflows.

If a match was found for register CCx in this mode, all further compare events during the current timer period are disabled for CCx until the corresponding timer overflows. If, after a match was detected, the compare register is reloaded with a new value, this value will not become effective until the next timer period.

In order to use the respective port pin as compare signal output pin CCxIO for compare register CCx in compare mode 3 this port pin must be configured as output, i.e. the corresponding direction control bit must be set to '1'. With this configuration, the initial state of the output signal can be programmed or its state can be modified at any time by writing to the port output latch.

In compare mode 3 the port latch is set upon a compare event and cleared upon a timer overflow (see [Figure 17-9](#)).

However, when compare value and reload value for a channel are equal the respective interrupt requests will be generated, only the output signal is not changed (set and clear would coincide in this case).

*Note: If the port output latch is written to by software at the same time it would be altered by a compare event, the software write will have priority. In this case the hardware-triggered change will not become effective.*

## 17.6 Capture/Compare Interrupts

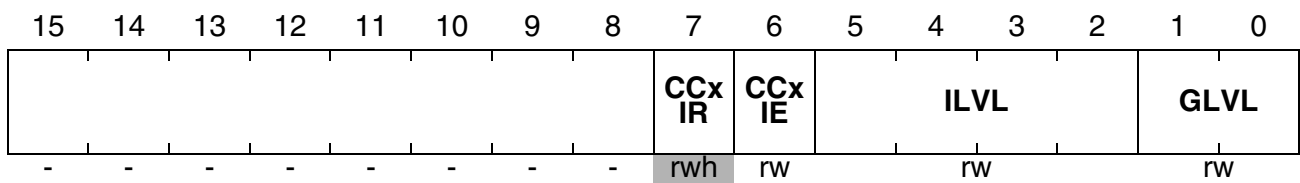
Upon a capture or compare event, the interrupt request flag CCxIR for the respective capture/compare register CCx is set to '1'. This flag can be used to generate an interrupt or trigger a PEC service request when enabled by the interrupt enable bit CCxIE.

Capture interrupts can be regarded as external interrupt requests with the additional feature of recording the time at which the triggering event occurred (see also [Section 5.8](#)).

Each of the 32 capture/compare registers has its own bitaddressable interrupt control register (CC0IC ... CC31IC) and its own interrupt vector (CC0INT ... CC31INT). These registers are organized the same way as all other interrupt control registers. The basic register layout is shown below, [Table 17-6](#) lists the associated addresses.

### CCxIC

**CAPCOM Intr. Ctrl. Reg. (E)SFR (See [Table 17-6](#)) Reset Value: - - 00<sub>H</sub>**



*Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.*



**The Capture/Compare Units**

**Table 17-6 CAPCOM Unit Interrupt Control Register Addresses**

CAPCOM1 Unit			CAPCOM2 Unit		
Register	Address	Reg. Space	Register	Address	Reg. Space
CC0IC	FF78 <sub>H</sub> /BC <sub>H</sub>	SFR	CC16IC	F160 <sub>H</sub> /B0 <sub>H</sub>	ESFR
CC1IC	FF7A <sub>H</sub> /BD <sub>H</sub>	SFR	CC17IC	F162 <sub>H</sub> /B1 <sub>H</sub>	ESFR
CC2IC	FF7C <sub>H</sub> /BE <sub>H</sub>	SFR	CC18IC	F164 <sub>H</sub> /B2 <sub>H</sub>	ESFR
CC3IC	FF7E <sub>H</sub> /BF <sub>H</sub>	SFR	CC19IC	F166 <sub>H</sub> /B3 <sub>H</sub>	ESFR
CC4IC	FF80 <sub>H</sub> /C0 <sub>H</sub>	SFR	CC20IC	F168 <sub>H</sub> /B4 <sub>H</sub>	ESFR
CC5IC	FF82 <sub>H</sub> /C1 <sub>H</sub>	SFR	CC21IC	F16A <sub>H</sub> /B5 <sub>H</sub>	ESFR
CC6IC	FF84 <sub>H</sub> /C2 <sub>H</sub>	SFR	CC22IC	F16C <sub>H</sub> /B6 <sub>H</sub>	ESFR
CC7IC	FF86 <sub>H</sub> /C3 <sub>H</sub>	SFR	CC23IC	F16E <sub>H</sub> /B7 <sub>H</sub>	ESFR
CC8IC	FF88 <sub>H</sub> /C4 <sub>H</sub>	SFR	CC24IC	F170 <sub>H</sub> /B8 <sub>H</sub>	ESFR
CC9IC	FF8A <sub>H</sub> /C5 <sub>H</sub>	SFR	CC25IC	F172 <sub>H</sub> /B9 <sub>H</sub>	ESFR
CC10IC	FF8C <sub>H</sub> /C6 <sub>H</sub>	SFR	CC26IC	F174 <sub>H</sub> /BA <sub>H</sub>	ESFR
CC11IC	FF8E <sub>H</sub> /C7 <sub>H</sub>	SFR	CC27IC	F176 <sub>H</sub> /BB <sub>H</sub>	ESFR
CC12IC	FF90 <sub>H</sub> /C8 <sub>H</sub>	SFR	CC28IC	F178 <sub>H</sub> /BC <sub>H</sub>	ESFR
CC13IC	FF92 <sub>H</sub> /C9 <sub>H</sub>	SFR	CC29IC	F184 <sub>H</sub> /C2 <sub>H</sub>	ESFR
CC14IC	FF94 <sub>H</sub> /CA <sub>H</sub>	SFR	CC30IC	F18C <sub>H</sub> /C6 <sub>H</sub>	ESFR
CC15IC	FF96 <sub>H</sub> /CB <sub>H</sub>	SFR	CC31IC	F194 <sub>H</sub> /CA <sub>H</sub>	ESFR

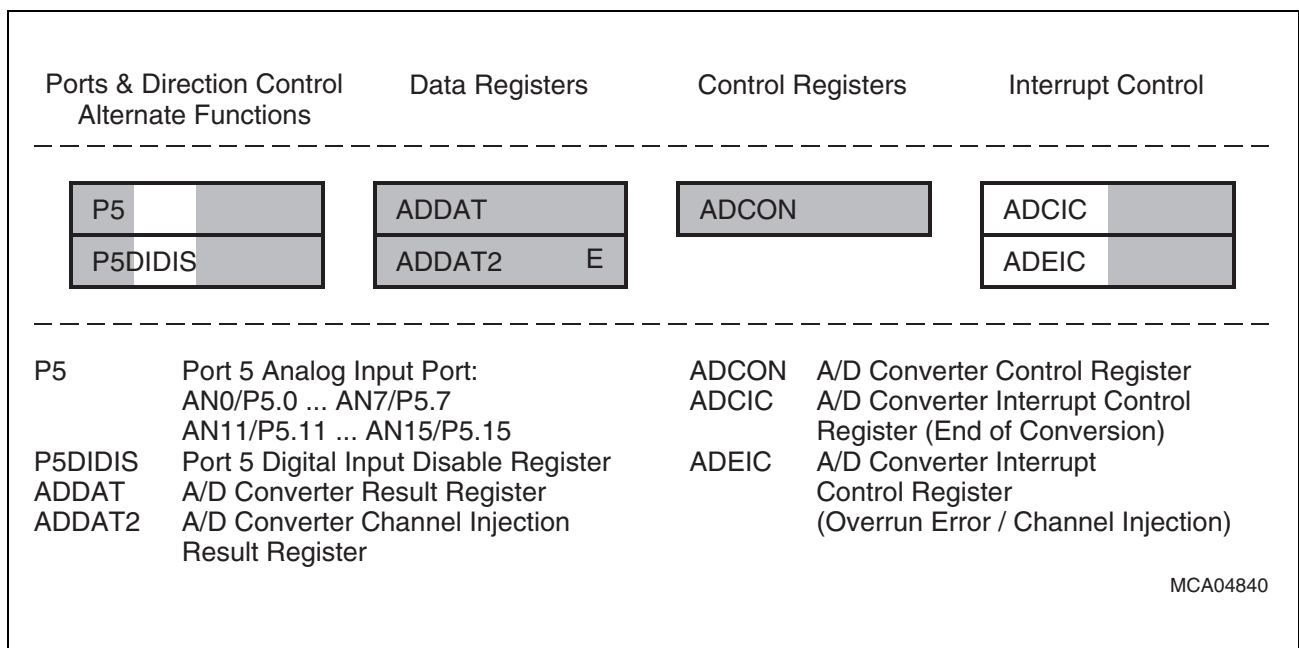
## 18 The Analog/Digital Converter

The C161CS/JC/JI provides an Analog/Digital Converter with 10-bit resolution and a sample & hold circuit on-chip. A multiplexer selects between up to 12 analog input channels (alternate functions of Port 5) either via software (fixed channel modes) or automatically (auto scan modes).

To fulfill most requirements of embedded control applications the ADC supports the following conversion modes:

- **Fixed Channel Single Conversion**  
produces just one result from the selected channel
- **Fixed Channel Continuous Conversion**  
repeatedly converts the selected channel
- **Auto Scan Single Conversion**  
produces one result from each of a selected group of channels
- **Auto Scan Continuous Conversion**  
repeatedly converts the selected group of channels
- **Wait for ADDAT Read Mode**  
start a conversion automatically when the previous result was read
- **Channel Injection Mode**  
insert the conversion of a specific channel into a group conversion (auto scan)

A set of SFRs and port pins provide access to control functions and results of the ADC.



**Figure 18-1 SFRs and Port Pins Associated with the A/D Converter**

The Analog/Digital Converter

The external analog reference voltages  $V_{AREF}$  and  $V_{AGND}$  are fixed. The separate supply for the ADC reduces the interference with other digital signals.

The sample time as well as the conversion time is programmable, so the ADC can be adjusted to the internal resistances of the analog sources and/or the analog reference voltage supply.

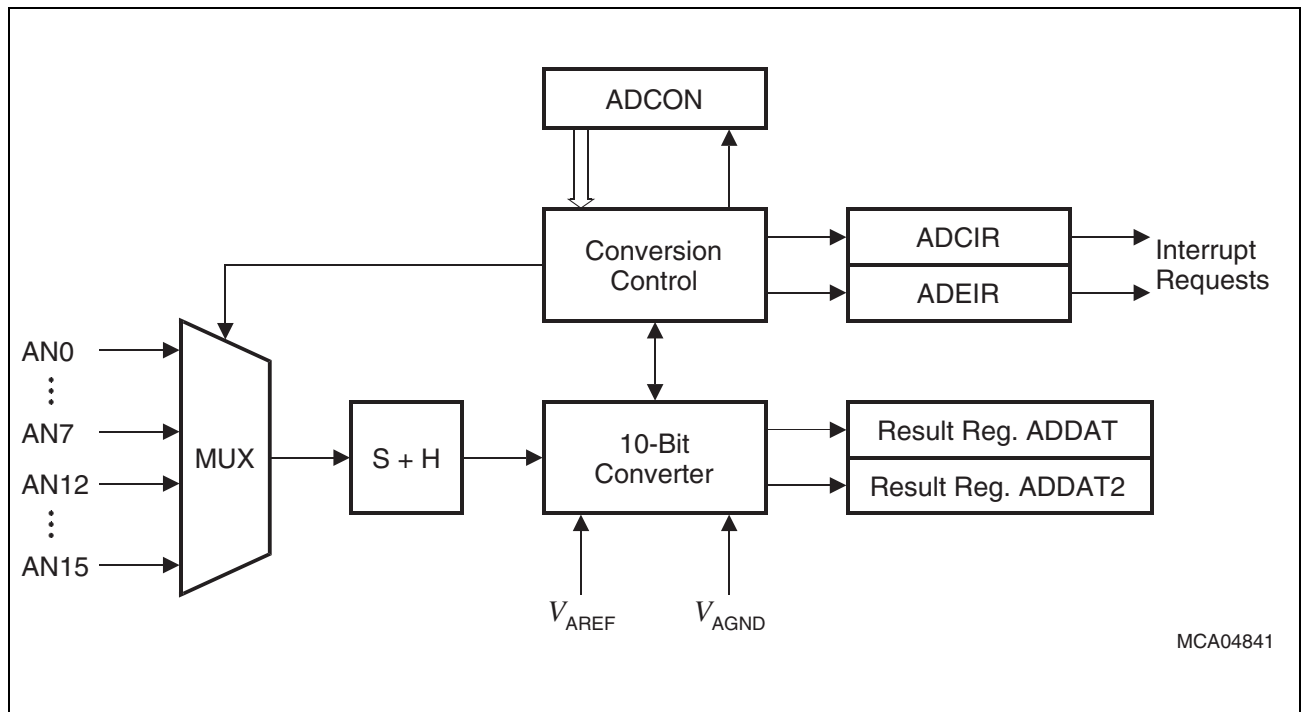


Figure 18-2 Analog/Digital Converter Block Diagram

## 18.1 Mode Selection and Operation

The analog input channels AN15 ... AN12, AN7 ... AN0 are alternate functions of Port 5 which is an input-only port. The Port 5 lines may either be used as analog or digital inputs. For pins that shall be used as analog inputs it is recommended to disable the digital input stage via register P5DIDIS. This avoids undesired cross currents and switching noise while the (analog) input signal level is between  $V_{IL}$  and  $V_{IH}$ .

The functions of the A/D converter are controlled by the bit-addressable A/D Converter Control Register ADCON. Its bitfields specify the analog channel to be acted upon, the conversion mode, and also reflect the status of the converter.

### ADCON

ADC Control Register

SFR (FFA0<sub>H</sub>/D0<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADCTC		ADSTC		AD CRQ	AD CIN	AD WR	AD BSY	AD ST	-	ADM		ADCH			
rw		rw		rwh	rw	rw	rwh	rwh	-	rw		rw			

Bit	Function
ADCH	<b>ADC Analog Channel Input Selection</b> Selects the (first) ADC channel which is to be converted.
ADM	<b>ADC Mode Selection</b> 00: Fixed Channel Single Conversion 01: Fixed Channel Continuous Conversion 10: Auto Scan Single Conversion 11: Auto Scan Continuous Conversion
ADST	<b>ADC Start Bit</b> 0: Stop a running conversion 1: Start conversion(s)
ADBSY	<b>ADC Busy Flag</b> 0: ADC is idle 1: A conversion is active
ADWR	<b>ADC Wait for Read Control</b>
ADCIN	<b>ADC Channel Injection Enable</b>
ADCRQ	<b>ADC Channel Injection Request Flag</b>

The Analog/Digital Converter

Bit	Function
<b>ADSTC</b>	<b>ADC Sample Time Control</b> (Defines the ADC sample time in a certain range) 00: $t_{BC} \times 8$ 01: $t_{BC} \times 16$ 10: $t_{BC} \times 32$ 11: $t_{BC} \times 64$
<b>ADCTC</b>	<b>ADC Conversion Time Control</b> (Defines the ADC basic conversion clock $f_{BC}$ ) 00: $f_{BC} = f_{CPU} / 4$ 01: $f_{BC} = f_{CPU} / 2$ 10: $f_{BC} = f_{CPU} / 16$ 11: $f_{BC} = f_{CPU} / 8$

Bitfield ADCH specifies the analog input channel which is to be converted (first channel of a conversion sequence in auto scan modes). Bitfield ADM selects the operating mode of the A/D converter. A conversion (or a sequence) is then started by setting bit ADST. Clearing ADST stops the A/D converter after a certain operation which depends on the selected operating mode.

The busy flag (read-only) ADBSY is set, as long as a conversion is in progress.

The result of a conversion is stored in the result register ADDAT, or in register ADDAT2 for an injected conversion.

*Note: Bitfield CHNR of register ADDAT is loaded by the ADC to indicate, which channel the result refers to.*

*Bitfield CHNR of register ADDAT2 is loaded by the CPU to select the analog channel, which is to be injected.*

**The Analog/Digital Converter**

**ADDAT**

**ADC Result Register**

**SFR (FEA0<sub>H</sub>/50<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>CHNR</b>				-	-	<b>ADRES</b>									
rwh				-	-	rwh									

**ADDAT2**

**ADC Chan. Inj. Result Reg.**

**ESFR (F0A0<sub>H</sub>/50<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>CHNR</b>				-	-	<b>ADRES</b>									
rw				-	-	rwh									

Bit	Function
<b>ADRES</b>	<b>A/D Conversion Result</b> The 10-bit digital result of the most recent conversion.
<b>CHNR</b>	<b>Channel Number</b> (identifies the converted analog channel)

**The Analog/Digital Converter**

A conversion is started by setting bit ADST = '1'. The busy flag ADBSY will be set and the converter then selects and samples the input channel, which is specified by the channel selection field ADCH in register ADCON. The sampled level will then be held internally during the conversion. When the conversion of this channel is complete, the 10-bit result together with the number of the converted channel is transferred into the result register ADDAT and the interrupt request flag ADCIR is set. The conversion result is placed into bitfield ADRES of register ADDAT.

If bit ADST is reset via software, while a conversion is in progress, the A/D converter will stop after the current conversion (fixed channel modes) or after the current conversion sequence (auto scan modes).

Setting bit ADST while a conversion is running, will abort this conversion and start a new conversion with the parameters specified in ADCON.

*Note: Abortion and restart (see above) are triggered by bit ADST changing from '0' to '1', i.e. ADST must be '0' before being set.*

While a conversion is in progress, the mode selection field ADM and the channel selection field ADCH may be changed. ADM will be evaluated after the current conversion. ADCH will be evaluated after the current conversion (fixed channel modes) or after the current conversion sequence (auto scan modes).

**Fixed Channel Conversion Modes**

These modes are selected by programming the mode selection bitfield ADM in register ADCON to '00<sub>B</sub>' (single conversion) or to '01<sub>B</sub>' (continuous conversion). After starting the converter through bit ADST the busy flag ADBSY will be set and the channel specified in bit field ADCH will be converted. After the conversion is complete, the interrupt request flag ADCIR will be set.

**In Single Conversion Mode** the converter will automatically stop and reset bits ADBSY and ADST.

**In Continuous Conversion Mode** the converter will automatically start a new conversion of the channel specified in ADCH. ADCIR will be set after each completed conversion.

When bit ADST is reset by software, while a conversion is in progress, the converter will complete the current conversion and then stop and reset bit ADBSY.

### Auto Scan Conversion Modes

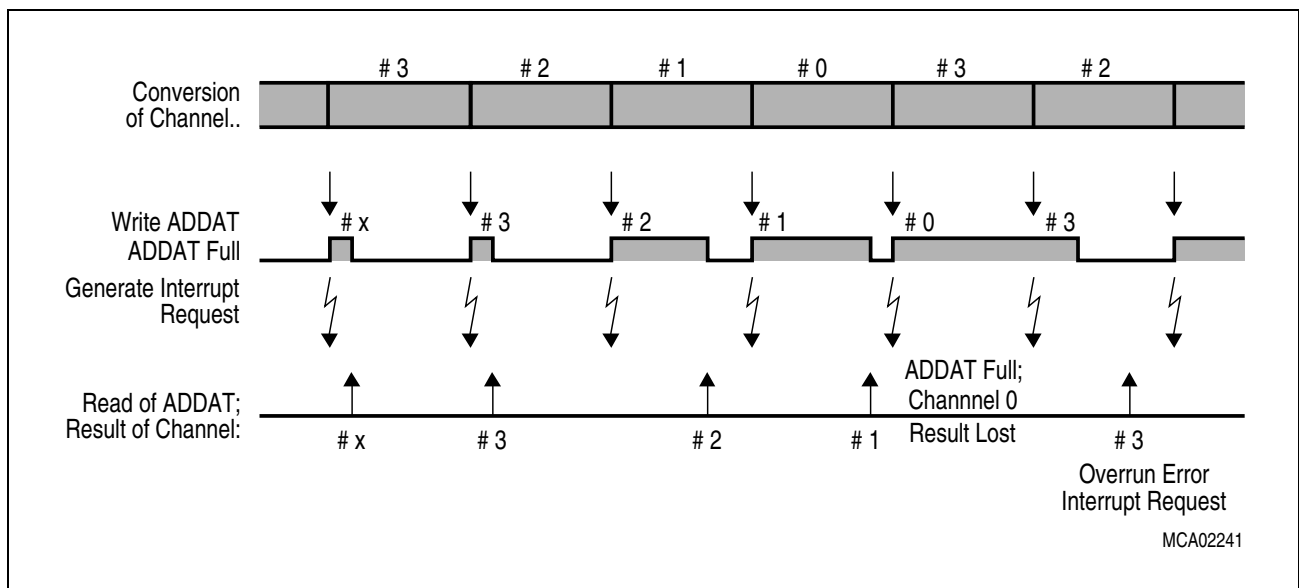
These modes are selected by programming the mode selection field ADM in register ADCON to '10<sub>B</sub>' (single conversion) or to '11<sub>B</sub>' (continuous conversion). Auto Scan modes automatically convert a sequence of analog channels, beginning with the channel specified in bit field ADCH and ending with channel 0, without requiring software to change the channel number.

After starting the converter through bit ADST, the busy flag ADBSY will be set and the channel specified in bit field ADCH will be converted. After the conversion is complete, the interrupt request flag ADCIR will be set and the converter will automatically start a new conversion of the next lower channel. ADCIR will be set after each completed conversion. After conversion of channel 0 the current sequence is complete.

**In Single Conversion Mode** the converter will automatically stop and reset bits ADBSY and ADST.

**In Continuous Conversion Mode** the converter will automatically start a new sequence beginning with the conversion of the channel specified in ADCH.

When bit ADST is reset by software, while a conversion is in progress, the converter will complete the current sequence (including conversion of channel 0) and then stop and reset bit ADBSY.



**Figure 18-3 Auto Scan Conversion Mode Example**

*Note: Auto Scan sequences that begin with channel numbers above 7 will generate (up to) 4 invalid results from channels 11 ... 8 which are not connected to input pins. Starting an Auto Scan sequence with  $ADCON.ADCH = E_H$  will generate the following 15 results: 14, 13, 12, x, x, x, x, 7, 6, 5, 4, 3, 2, 1, 0. Starting a sequence with  $ADCON.ADCH = B_H \dots 8_H$  generates 4 ... 1 invalid results at the beginning of the sequence and therefore makes no sense in an application.*



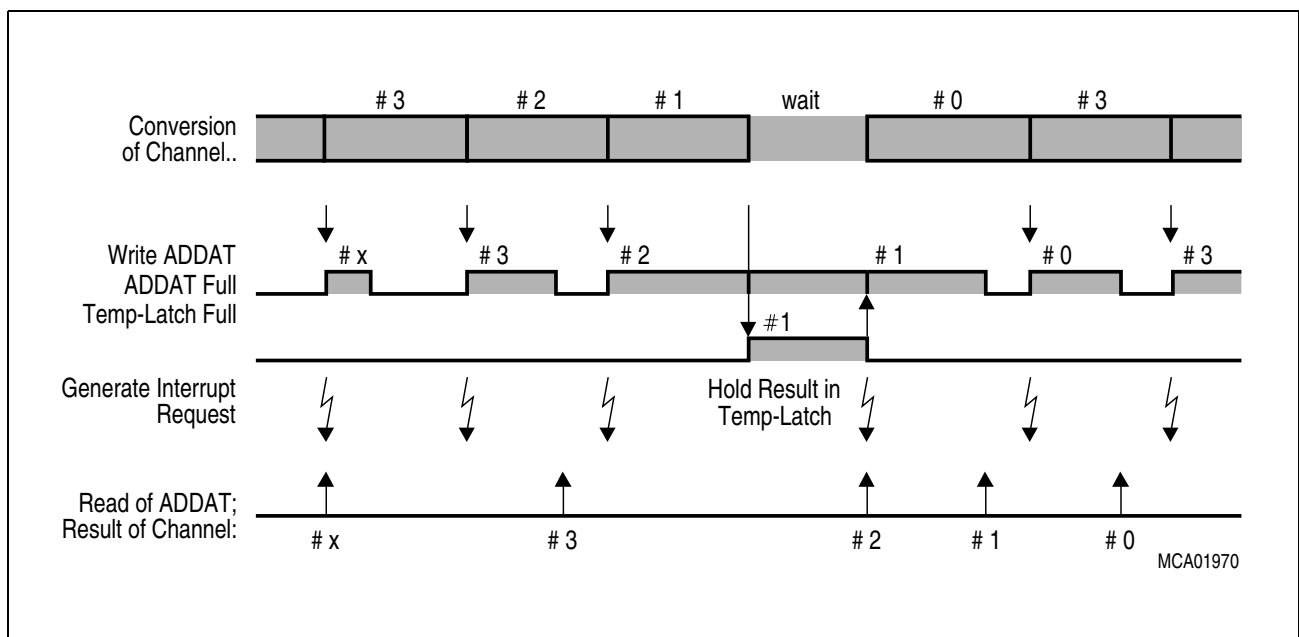
### Wait for ADDAT Read Mode

If in default mode of the ADC a previous conversion result has not been read out of register ADDAT by the time a new conversion is complete, the previous result in register ADDAT is lost because it is overwritten by the new value, and the A/D overrun error interrupt request flag ADEIR will be set.

In order to avoid error interrupts and the loss of conversion results especially when using continuous conversion modes, the ADC can be switched to “Wait for ADDAT Read Mode” by setting bit ADWR in register ADCON.

If the value in ADDAT has not been read by the time the current conversion is complete, the new result is stored in a temporary buffer and the next conversion is suspended (ADST and ADBSY will remain set in the meantime, but no end-of-conversion interrupt will be generated). After reading the previous value from ADDAT the temporary buffer is copied into ADDAT (generating an ADCIR interrupt) and the suspended conversion is started. This mechanism applies to both single and continuous conversion modes.

*Note: While in standard mode continuous conversions are executed at a fixed rate (determined by the conversion time), in “Wait for ADDAT Read Mode” there may be delays due to suspended conversions. However, this only affects the conversions, if the CPU (or PEC) cannot keep track with the conversion rate.*



**Figure 18-4 Wait for Read Mode Example**

### Channel Injection Mode

Channel Injection Mode allows the conversion of a specific analog channel (also while the ADC is running in a continuous or auto scan mode) without changing the current operating mode. After the conversion of this specific channel the ADC continues with the original operating mode.

Channel Injection mode is enabled by setting bit ADCIN in register ADCON and requires the Wait for ADDAT Read Mode (ADWR = '1'). The channel to be converted in this mode is specified in bitfield CHNR of register ADDAT2.

*Note: Bitfield CHNR in ADDAT2 is not modified by the A/D converter, but only the ADRES bit field. Since the channel number for an injected conversion is not buffered, bitfield CHNR of ADDAT2 must never be modified during the sample phase of an injected conversion, otherwise the input multiplexer will switch to the new channel. It is recommended to only change the channel number with no injected conversion running.*

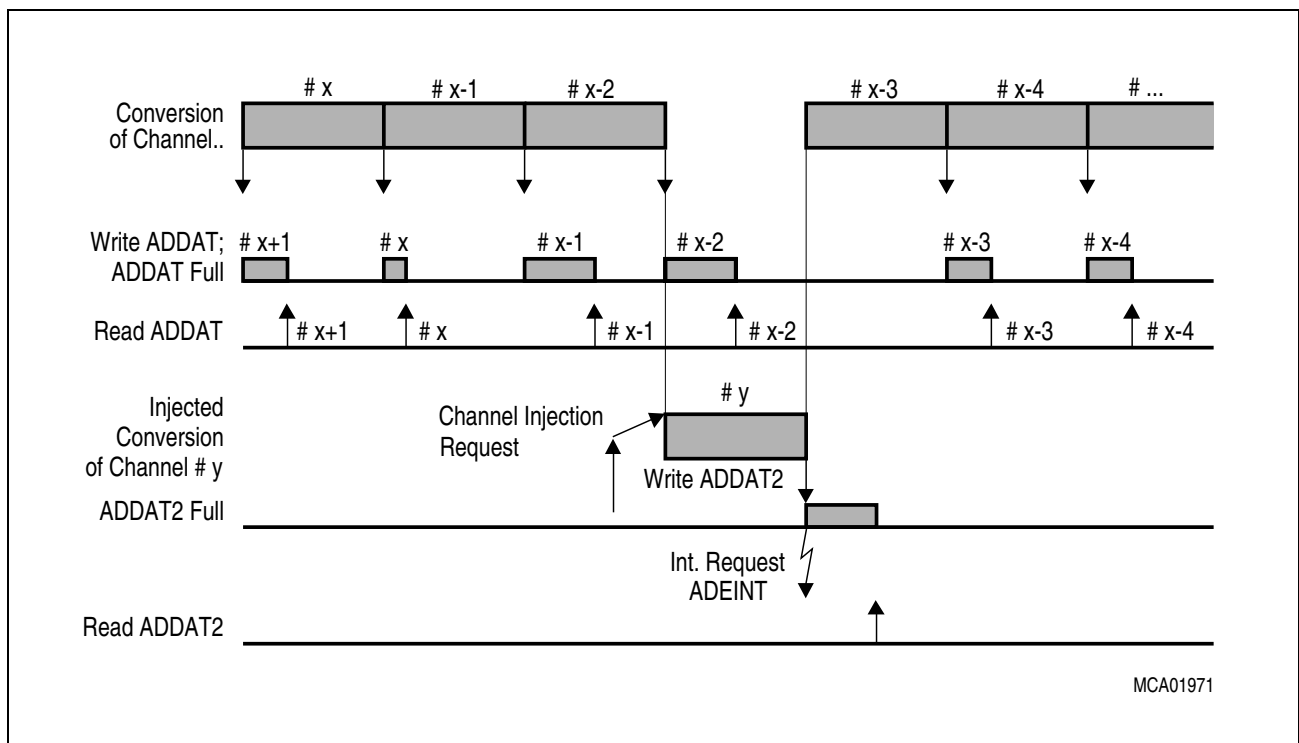


Figure 18-5 Channel Injection Example

**A channel injection can be triggered in two ways:**

- setting of the Channel Injection Request bit ADCRQ via software
- a compare or a capture event of Capture/Compare register CC31 of the CAPCOM2 unit, which also sets bit ADCRQ.

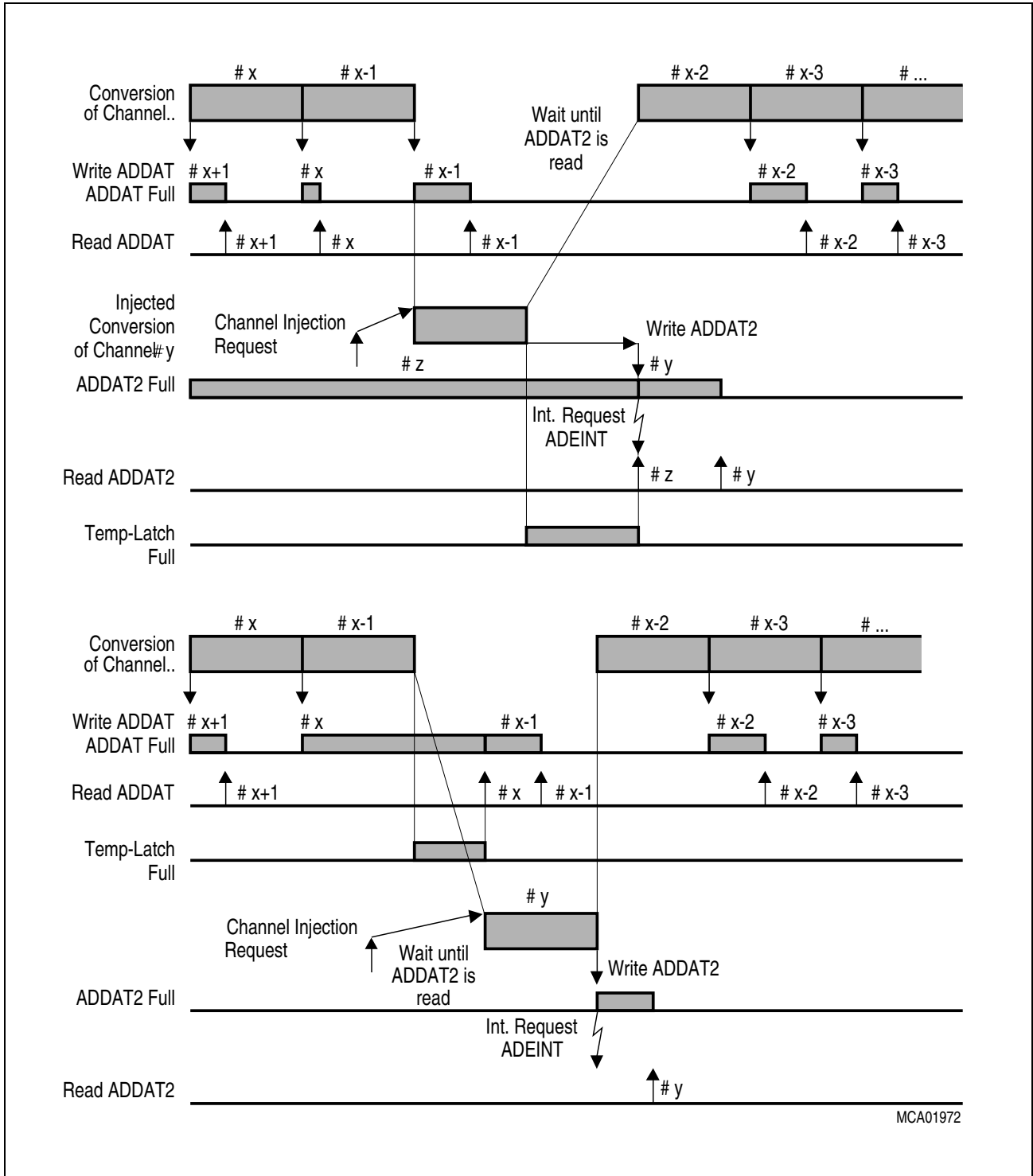
The second method triggers a channel injection at a specific time, on the occurrence of a predefined count value of the CAPCOM timers or on a capture event of register CC31. This can be either the positive, the negative, or both the positive and the negative edge of an external signal. In addition, this option allows recording the time of occurrence of this signal.

*Note: The channel injection request bit ADCRQ will be set on any interrupt request of CAPCOM2 channel CC31, regardless whether the channel injection mode is enabled or not. It is recommended to always clear bit ADCRQ before enabling the channel injection mode.*

After the completion of the current conversion (if any is in progress) the converter will start (inject) the conversion of the specified channel. When the conversion of this channel is complete, the result will be placed into the alternate result register ADDAT2, and a Channel Injection Complete Interrupt request will be generated, which uses the interrupt request flag ADEIR (for this reason the Wait for ADDAT Read Mode is required).

*Note: If the temporary data register used in Wait for ADDAT Read Mode is full, the respective next conversion (standard or injected) will be suspended. The temporary register can hold data for ADDAT (from a standard conversion) or for ADDAT2 (from an injected conversion).*

The Analog/Digital Converter



MCA01972

Figure 18-6 Channel Injection Example with Wait for Read

**Arbitration of Conversions**

Conversion requests that are activated while the ADC is idle immediately trigger the respective conversion. If a conversion is requested while another conversion is currently in progress the operation of the A/D converter depends on the kind of the involved conversions (standard or injected).

*Note: A conversion request is activated if the respective control bit (ADST or ADCRQ) is toggled from '0' to '1', i.e. the bit must have been zero before being set.*

**Table 18-1** summarizes the ADC operation in the possible situations.

**Table 18-1 Conversion Arbitration**

Conversion in Progress	New Requested Conversion	
	Standard	Injected
<b>Standard</b>	Abort running conversion, and start requested new conversion.	Complete running conversion, start requested conversion after that.
<b>Injected</b>	Complete running conversion, start requested conversion after that.	Complete running conversion, start requested conversion after that. Bit ADCRQ will be '0' for the second conversion, however.

## 18.2 Conversion Timing Control

When a conversion is started, first the capacitances of the converter are loaded via the respective analog input pin to the current analog input voltage. The time to load the capacitances is referred to as sample time. Next the sampled voltage is converted to a digital value in successive steps, which correspond to the resolution of the ADC. During these phases (except for the sample time) the internal capacitances are repeatedly charged and discharged via pins  $V_{AREF}$  and  $V_{AGND}$ .

The current that has to be drawn from the sources for sampling and changing charges depends on the time that each respective step takes, because the capacitors must reach their final voltage level within the given time, at least with a certain approximation. The maximum current, however, that a source can deliver, depends on its internal resistance.

The time that the two different actions during conversion take (sampling, and converting) can be programmed within a certain range in the C161CS/JC/JI relative to the CPU clock. The absolute time that is consumed by the different conversion steps therefore is independent from the general speed of the controller. This allows adjusting the A/D converter of the C161CS/JC/JI to the properties of the system:

**Fast Conversion** can be achieved by programming the respective times to their absolute possible minimum. This is preferable for scanning high frequency signals. The internal resistance of analog source and analog supply must be sufficiently low, however.

**High Internal Resistance** can be achieved by programming the respective times to a higher value, or the possible maximum. This is preferable when using analog sources and supply with a high internal resistance in order to keep the current as low as possible. The conversion rate in this case may be considerably lower, however.

The conversion time is programmed via the upper two bits of register ADCON. Bitfield ADCTC (conversion time control) selects the basic conversion clock ( $f_{BC}$ ), used for the operation of the A/D converter. The sample time is derived from this conversion clock.

**Table 18-2** lists the possible combinations. The timings refer to CPU clock cycles, where  $t_{CPU} = 1 / f_{CPU}$ .

The limit values for  $f_{BC}$  (see data sheet) must not be exceeded when selecting ADCTC and  $f_{CPU}$ .

**Table 18-2 ADC Conversion Timing Control**

ADCON.15 14 (ADCTC)	A/D Converter Basic Clock $f_{BC}$	ADCON.13 12 (ADSTC)	Sample Time $t_S$
00	$f_{CPU} / 4$	00	$t_{BC} \times 8$
01	$f_{CPU} / 2$	01	$t_{BC} \times 16$
10	$f_{CPU} / 16$	10	$t_{BC} \times 32$
11	$f_{CPU} / 8$	11	$t_{BC} \times 64$

---

**The Analog/Digital Converter**

The time for a complete conversion includes the sample time  $t_S$ , the conversion itself and the time required to transfer the digital value to the result register ( $2 t_{CPU}$ ) as shown in the example below.

*Note: The non-linear decoding of bit field ADCTC provides compatibility with 80C166 designs for the default value ('00' after reset).*

**Converter Timing Example**

Assumptions:  $f_{CPU} = 25 \text{ MHz}$  (i.e.  $t_{CPU} = 40 \text{ ns}$ ), ADCTC = '00', ADSTC = '00'.

Basic clock  $f_{BC} = f_{CPU} / 4 = 6.25 \text{ MHz}$ , i.e.  $t_{BC} = 160 \text{ ns}$ .

Sample time  $t_S = t_{BC} \times 8 = 1280 \text{ ns}$ .

Conversion time  $t_C = t_S + 40 t_{BC} + 2 t_{CPU} = (1280 + 6400 + 80) \text{ ns} = 7.76 \mu\text{s}$ .

*Note: For the exact specification please refer to the data sheet of the selected derivative.*

### 18.3 A/D Converter Interrupt Control

At the end of each conversion, interrupt request flag ADCIR in interrupt control register ADCIC is set. This end-of-conversion interrupt request may cause an interrupt to vector ADCINT, or it may trigger a PEC data transfer which reads the conversion result from register ADDAT e.g. to store it into a table in the internal RAM for later evaluation.

The interrupt request flag ADEIR in register ADEIC will be set either, if a conversion result overwrites a previous value in register ADDAT (error interrupt in standard mode), or if the result of an injected conversion has been stored into ADDAT2 (end-of-injected-conversion interrupt). This interrupt request may be used to cause an interrupt to vector ADEINT, or it may trigger a PEC data transfer.

#### ADCIC

ADC Conversion Intr.Ctrl.Reg. SFR (FF98<sub>H</sub>/CC<sub>H</sub>) Reset Value: - - 00<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				-				ADC IR	ADC IE			ILVL			GLVL
-	-	-	-	-	-	-	-	rwh	rw			rw			rw

#### ADEIC

ADC Error Intr.Ctrl.Reg. SFR (FF9A<sub>H</sub>/CD<sub>H</sub>) Reset Value: - - 00<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				-				ADE IR	ADE IE			ILVL			GLVL
-	-	-	-	-	-	-	-	rwh	rw			rw			rw

*Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.*



## 19 The On-Chip CAN Interface

*Note: The C161CS incorporates two CAN modules (CAN1 and CAN2), the C161JC incorporates one CAN module (CAN1). As the structure of the two modules is identical, this chapter mainly describes the functionality and the operation of module CAN1. The differences between modules CAN1 and CAN2 are described in a special section at the end of this chapter.*

The Controller Area Network (CAN) bus with its associated protocol allows communication between a number of stations which are connected to this bus with high efficiency.

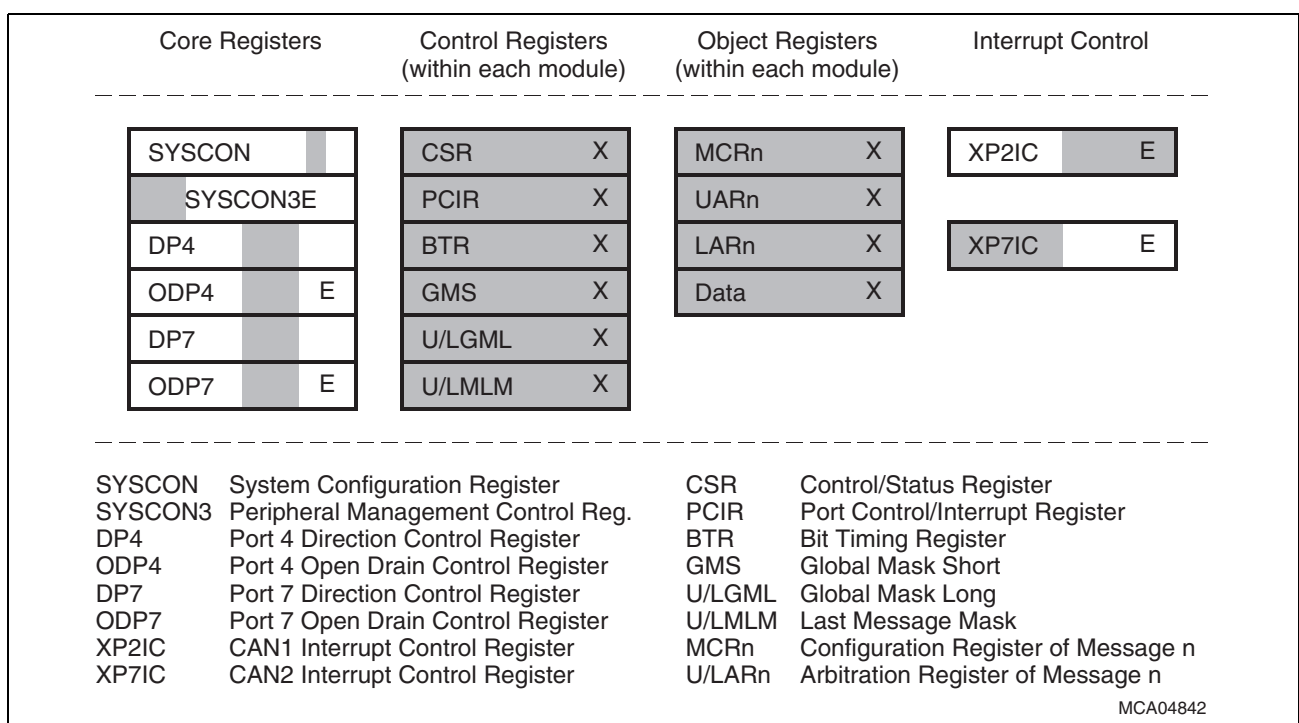
Efficiency in this context means:

- Transfer speed (Data rates of up to 1 Mbit/sec can be achieved)
- Data integrity (The CAN protocol provides several means for error checking)
- Host processor unloading (The controller here handles most of the tasks autonomously)
- Flexible and powerful message passing (The extended CAN protocol is supported)

The integrated CAN module handles the completely autonomous transmission and reception of CAN frames in accordance with the CAN specification V2.0 part B (active), i.e. the on-chip CAN module can receive and transmit:

- **Standard frames** with 11-bit identifiers, as well as
- **Extended frames** with 29-bit identifiers.

*Note: The CAN module is an XBUS peripheral and therefore requires bit XPEN in register SYSCON to be set in order to be operable.*



**Figure 19-1 Registers Associated with the CAN Module**

## The On-Chip CAN Interface

The bit timing is derived from the XCLK and is programmable up to a data rate of 1 MBaud. The minimum CPU clock frequency to achieve 1 MBaud is  $f_{\text{CPU}} \geq 8/16$  MHz, depending on the activation of the CAN module's clock prescaler.

The CAN module uses two pins of Port 4 or of Port 8 to interface to a bus transceiver.

It provides **Full CAN** functionality on up to 15 full-sized message objects (8 data bytes each). Message object 15 may be configured for **Basic CAN** functionality with a double-buffered receive object.

Both modes provide separate masks for acceptance filtering which allows the acceptance of a number of identifiers in Full CAN mode and also allows disregarding a number of identifiers in Basic CAN mode.

All message objects can be updated independent from the other objects during operation of the module and are equipped with buffers for the maximum message length of 8 Bytes.

### 19.1 Functional Blocks of the CAN Module

The CAN module combines several functional blocks (see [Figure 19-2](#)) that work in parallel and contribute to the controller's performance. These units and the functions they provide are described below.

Each of the message objects has a unique identifier and its own set of control and status bits. Each object can be configured with its direction as either transmit or receive, except the last message which is only a double receive buffer with a special mask register.

An object with its direction set as transmit can be configured to be automatically sent whenever a remote frame with a matching identifier (taking into account the respective global mask register) is received over the CAN bus. By requesting the transmission of a message with the direction set as receive, a remote frame can be sent to request that the appropriate object be sent by some other node. Each object has separate transmit and receive interrupts and status bits, giving the CPU full flexibility in detecting when a remote/data frame has been sent or received.

For general purpose two masks for acceptance filtering can be programmed, one for identifiers of 11 bits and one for identifiers of 29 bits. However the CPU must configure bit XTD (Normal or Extended Frame Identifier) for each valid message to determine whether a standard or extended frame will be accepted.

The last message object has its own programmable mask for acceptance filtering, allowing a large number of infrequent objects to be handled by the system.

The object layer architecture of the CAN controller is designed to be as regular and orthogonal as possible. This makes it easy to use.

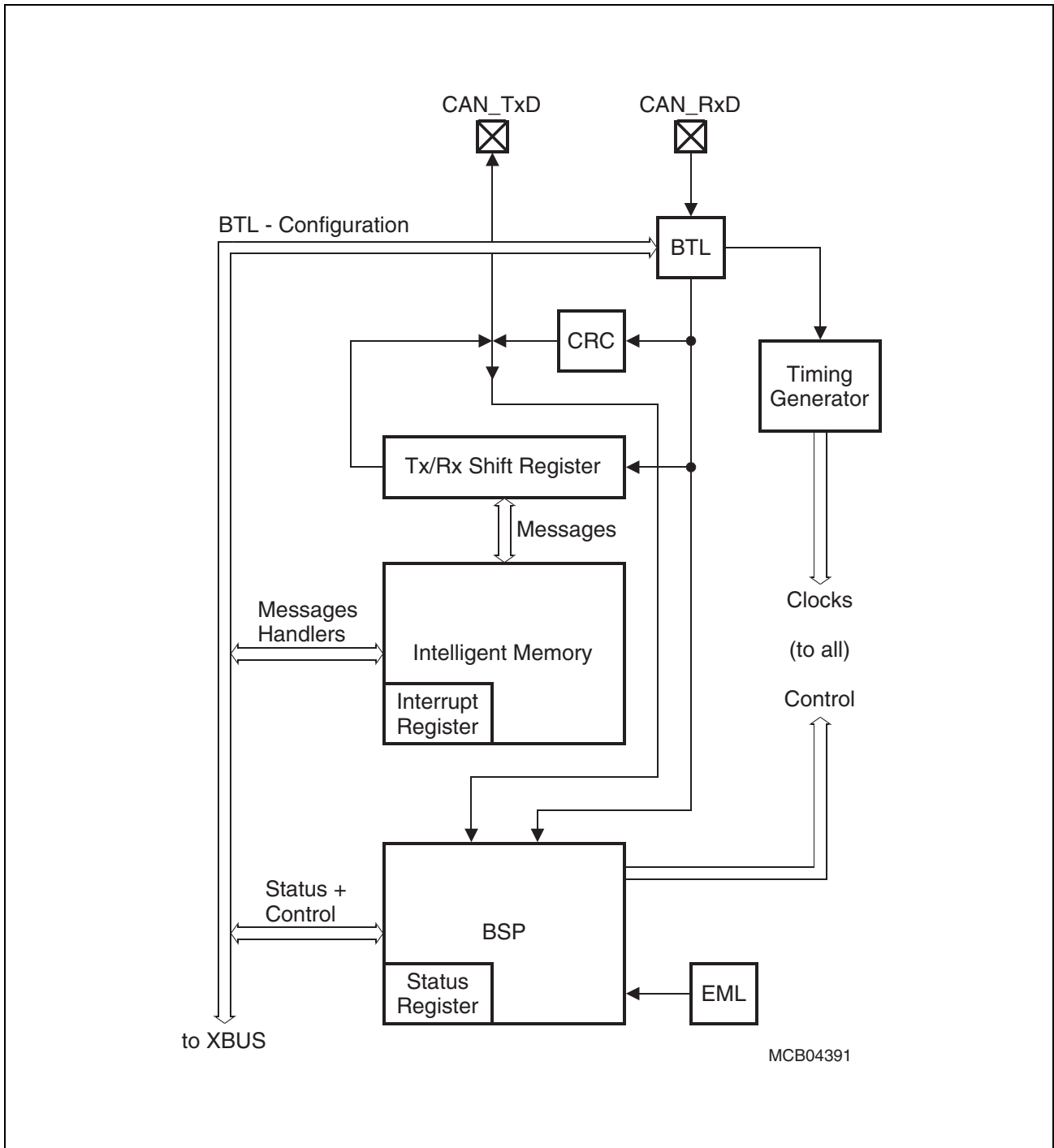


Figure 19-2 CAN Controller Block Diagram

### Tx/Rx Shift Register

The Transmit/Receive Shift Register holds the destuffed bit stream from the bus line to allow the parallel access to the whole data or remote frame for the acceptance match test and the parallel transfer of the frame to and from the Intelligent Memory.

### Bit Stream Processor

The Bit Stream Processor (BSP) is a sequencer controlling the sequential data stream between the Tx/Rx Shift Register, the CRC Register, and the bus line. The BSP also controls the EML and the parallel data stream between the Tx/Rx Shift Register and the Intelligent Memory such that the processes of reception, arbitration, transmission, and error signalling are performed according to the CAN protocol. Note that the automatic retransmission of messages which have been corrupted by noise or other external error conditions on the bus line is handled by the BSP.

### Cyclic Redundancy Check Register

This register generates the Cyclic Redundancy Check (CRC) code to be transmitted after the data bytes and checks the CRC code of incoming messages. This is done by dividing the data stream by the code generator polynomial.

### Error Management Logic

The Error Management Logic (EML) is responsible for the fault confinement of the CAN device. Its counters, the Receive Error Counter and the Transmit Error Counter, are incremented and decremented by commands from the Bit Stream Processor. According to the values of the error counters, the CAN controller is set into the states *error active*, *error passive* and *busoff*.

The CAN controller is *error active*, if both error counters are below the *error passive* limit of 128.

It is *error passive*, if at least one of the error counters equals or exceeds 128.

It goes *busoff*, if the Transmit Error Counter equals or exceeds the *busoff* limit of 256.

The device remains in this state, until the *busoff* recovery sequence is finished.

Additionally, there is the bit EWRN in the Status Register, which is set, if at least one of the error counters equals or exceeds the error warning limit of 96. EWRN is reset, if both error counters are less than the error warning limit.

### Bit Timing Logic

This block (BTL) monitors the busline input CAN\_RXD and handles the busline related bit timing according to the CAN protocol.

The BTL synchronizes on a *recessive* to *dominant* busline transition at *Start of Frame* (hard synchronization) and on any further *recessive* to *dominant* busline transition, if the CAN controller itself does not transmit a *dominant* bit (resynchronization).

The BTL also provides programmable time segments to compensate for the propagation delay time and for phase shifts and to define the position of the *Sample Point* in the bit time. The programming of the BTL depends on the baudrate and on external physical delay times.

### Intelligent Memory

The Intelligent Memory (CAM/RAM Array) provides storage for up to 15 message objects of maximum 8 data bytes length. Each of these objects has a unique identifier and its own set of control and status bits. After the initial configuration, the Intelligent Memory can handle the reception and transmission of data without further CPU actions.

### Organization of Registers and Message Objects

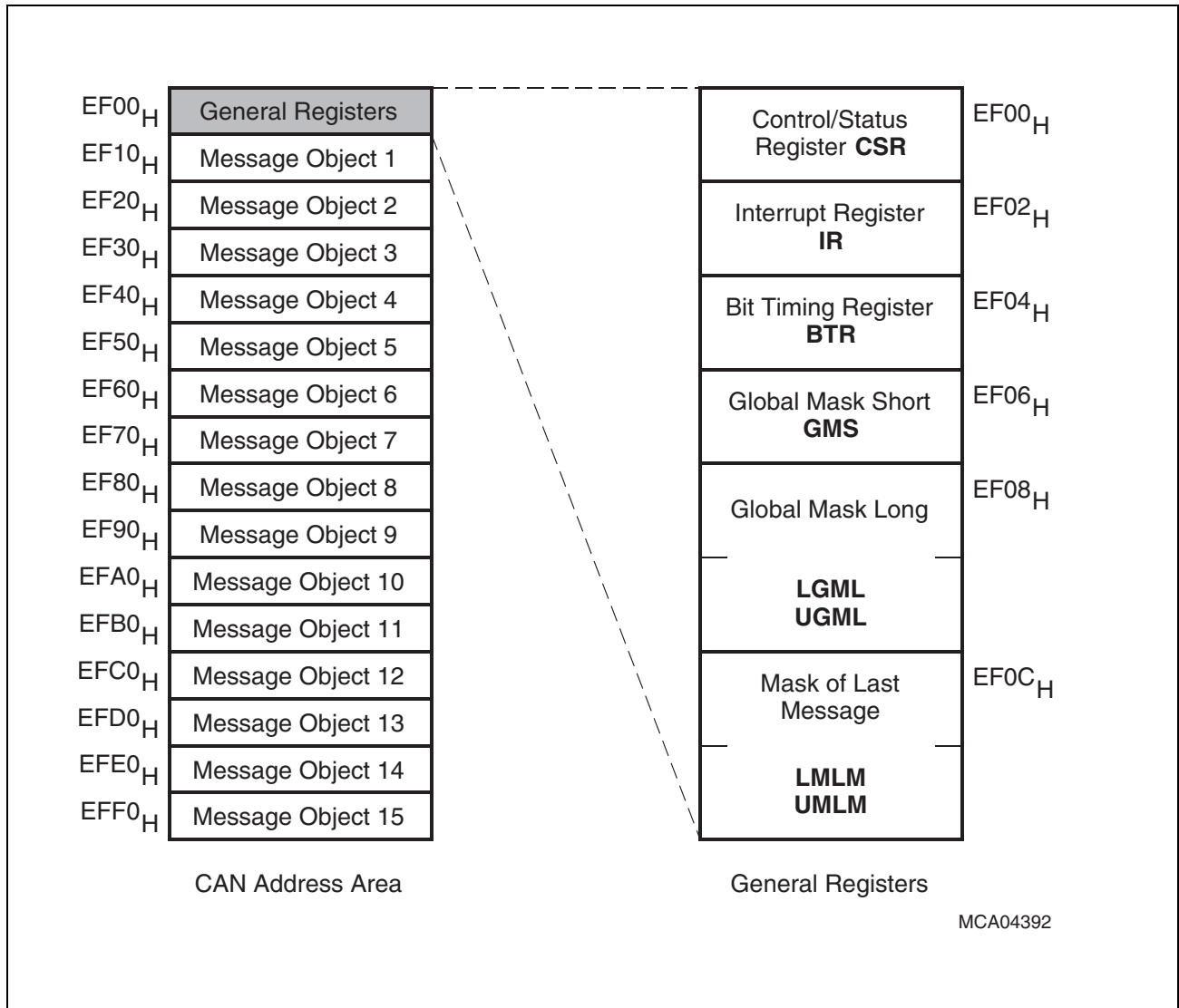
All registers and message objects of the CAN controller are located in the special CAN address area of 256 Bytes, which is mapped into segment 0 and uses addresses 00'EF00<sub>H</sub> through 00'EFFF<sub>H</sub>. All registers are organized as 16-bit registers, located on word addresses. However, all registers may be accessed byte-wise in order to select special actions without effecting other mechanisms.

**Register Naming** reflects the specific name of a register as well as a general module indicator. This results in unique register names.

**Example:** module indicator is **C1** (CAN module 1), specific name is Control/Status Register (**CSR**), unique register name is **C1CSR**.

*Note: The address map shown below lists the registers which are part of the CAN controller. There are also C161CS/JC/JI specific registers that are associated with the CAN module.*

**The On-Chip CAN Interface**



**Figure 19-3 CAN Module Address Map**

## 19.2 General Functional Description

The Control/Status Register (CSR) accepts general control settings for the module and provides general status information.

### CSR

Control/Status Register

XReg (EF00<sub>H</sub>)

Reset Value: XX01<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B OFF	E WRN	-	RX OK	TX OK		LEC		TM	CCE	0	CPS	EIE	SIE	IE	INIT
rh	rh	r	rwh	rwh		rwh		rw	rw	r	rw	rw	rw	rw	rwh

Bit	Function (Control Bits)
INIT	<p><b>Initialization</b> Starts the initialization of the CAN controller, when set. INIT is set – after a reset – when entering the <i>busoff</i> state – by the application software</p>
IE	<p><b>Interrupt Enable</b> Enables or disables interrupt generation from the CAN module via the signal XINTR. Does not affect status updates.</p>
SIE	<p><b>Status Change Interrupt Enable</b> Enables or disables interrupt generation when a message transfer (reception or transmission) is successfully completed or a CAN bus error is detected (and registered in the status partition).</p>
EIE	<p><b>Error Interrupt Enable</b> Enables or disables interrupt generation on a change of bit BOFF or EWARN in the status partition).</p>
CPS	<p><b>Clock Prescaler Control Bit</b> 0: <b>Standard mode:</b> the input clock is divided 2:1. The minimum input frequency to achieve a baudrate of 1 MBaud is <math>f_{CPU} = 16</math> MHz. 1: <b>Fast mode:</b> the input clock is used directly 1:1. The minimum input frequency to achieve a baudrate of 1 MBaud is <math>f_{CPU} = 8</math> MHz.</p>
CCE	<p><b>Configuration Change Enable</b> Allows or inhibits CPU access to the Bit Timing Register.</p>
TM	<p><b>Test Mode</b> (must be '0') Make sure that this bit is always cleared when writing to the Control Register, as this bit controls a special test mode, that is used for production testing. During normal operation, however, this test mode may lead to undesired behavior of the device.</p>

The On-Chip CAN Interface

Bit	Function (Control Bits)
LEC	<p><b>Last Error Code</b> This field holds a code which indicates the type of the last error occurred on the CAN bus. If a message has been transferred (reception or transmission) without error, this field will be cleared.</p> <p>0     <b>No Error</b></p> <p>1     <b>Stuff Error:</b> More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.</p> <p>2     <b>Form Error:</b> Wrong format in fixed format part of a received frame.</p> <p>3     <b>AckError:</b> The message this CAN controller transmitted was not acknowledged by another node.</p> <p>4     <b>Bit1Error:</b> During the transmission of a message (with the exception of the arbitration field), the device wanted to send a <i>recessive</i> level ("1"), but the monitored bus value was <i>dominant</i>.</p> <p>5     <b>Bit0Error:</b> During the transmission of a message (or acknowledge bit, active error flag, or overload flag), the device wanted to send a <i>dominant</i> level ("0"), but the monitored bus value was <i>recessive</i>. During <i>busoff</i> recovery this status is set each time a sequence of 11 <i>recessive</i> bits has been monitored. This enables the CPU to monitor the proceeding of the busoff recovery sequence (indicates that the bus is not stuck at <i>dominant</i> or continuously disturbed).</p> <p>6     <b>CRCErrror:</b> The received CRC check sum was incorrect.</p> <p>7     <b>Unused code:</b> may be written by the CPU to check for updates.</p>
TXOK	<p><b>Transmitted Message Successfully</b> Indicates that a message has been transmitted successfully (error free and acknowledged by at least one other node), since this bit was last reset by the CPU (the CAN controller does not reset this bit!).</p>
RXOK	<p><b>Received Message Successfully</b> This bit is set each time a message has been received successfully, since this bit was last reset by the CPU (the CAN controller does not reset this bit!). RXOK is also set when a message is received that is not accepted (i.e. stored).</p>
EWRN	<p><b>Error Warning Status</b> Indicates that at least one of the error counters in the EML has reached the error warning limit of 96.</p>
BOFF	<p><b>Busoff Status</b> Indicates when the CAN controller is in busoff state (see EML).</p>

*Note: Reading the upper half of the Control Register (status partition) will clear the Status Change Interrupt value in the Interrupt Register, if it is pending. Use byte accesses to the lower half to avoid this.*



### 19.2.1 CAN Interrupt Handling

The on-chip CAN module has one interrupt output, which is connected (through a synchronization stage) to a standard interrupt node in the C161CS/JC/JI in the same manner as all other interrupts of the standard on-chip peripherals. With this configuration, the user has all control options available for this interrupt, such as enabling/disabling, level and group priority, and interrupt or PEC service (see note below). The on-chip CAN module is connected to an XBUS interrupt control register.

As for all other interrupts, the node interrupt request flag is cleared automatically by hardware when this interrupt is serviced (either by standard interrupt or PEC service).

*Note: As a rule, CAN interrupt requests can be serviced by a PEC channel. However, because PEC channels only can execute single predefined data transfers (there are no conditional PEC transfers), PEC service can only be used, if the respective request is known to be generated by one specific source, and that no other interrupt request will be generated in between. In practice this seems to be a rare case.*

Since an interrupt request of the CAN module can be generated due to different conditions, the appropriate CAN interrupt status register must be read in the service routine to determine the cause of the interrupt request. The interrupt identifier INTID (a number) in the Port Control/Interrupt Register (PCIR) indicates the cause of an interrupt. When no interrupt is pending, the identifier will have the value 00<sub>H</sub>.

If the value in INTID is not 00<sub>H</sub>, then there is an interrupt pending. If bit IE in the control/status register is set also the interrupt signal to the CPU is activated. The interrupt signal (to the interrupt node) remains active until INTID gets 00<sub>H</sub> (i.e. all interrupt requests have been serviced) or until interrupt generation is disabled (CSR.IE = '0').

*Note: The interrupt node is activated only upon a 0 → 1 transition of the CAN interrupt signal. The CAN interrupt service routine should only be left after INTID has been verified to be 00<sub>H</sub>.*

The interrupt with the lowest number has the highest priority. If a higher priority interrupt (lower number) occurs before the current interrupt is processed, INTID is updated and the new interrupt overrides the last one.

INTID is also updated when the respective source request has been processed. This is indicated by clearing the INTPND flag in the respective object's message control register (MCRn) or by reading the status partition of register CSR (in case of a status change interrupt). The updating of INTID is done by the CAN state machine and takes up to 6 CAN clock cycles (1 CAN clock cycle = 1 or 2 CPU clock cycles, determined by the prescaler bit CPS), depending on current state of the state machine.

*Note: A worst case condition can occur when BRP = 00<sub>H</sub> **AND** the CAN controller is storing a just received message **AND** the CPU is executing consecutive accesses to the CAN module. In this rare case the maximum delay may be 26 CAN clock cycles.*

*The impact of this delay can be minimized by clearing bit INTPND at an early*

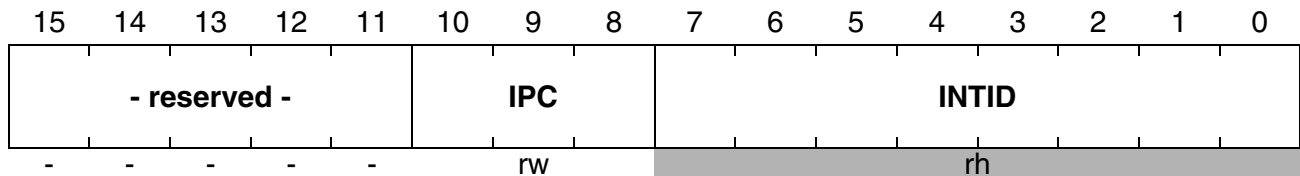
The On-Chip CAN Interface

stage of interrupt processing, and (if required) restricting CPU accesses to the CAN module until the anticipated updating is complete.

PCIR

Port Control / Interrupt Register XReg (EF02<sub>H</sub>)

Reset Value: XXXX<sub>H</sub>



Bit	Function
<b>INTID</b>	<b>Interrupt Identifier</b> This number indicates the cause of the interrupt (if pending).
	00 <sub>H</sub> <b>Interrupt Idle:</b> There is no interrupt request pending.
	01 <sub>H</sub> <b>Status Change Interrupt:</b> The CAN controller has updated (not necessarily changed) the status in the Control Register. This can refer to a change of the error status of the CAN controller (EIE is set and BOFF or EWRN change) or to a CAN transfer incident (SIE must be set), like reception or transmission of a message (RXOK or TXOK is set) or the occurrence of a CAN bus error (LEC is updated). The CPU may clear RXOK, TXOK, and LEC, however, writing to the status partition of the Control Register can never generate or reset an interrupt. To update the INTID value the status partition of the Control Register must be read.
	02 <sub>H</sub> <b>Message 15 Interrupt:</b> Bit INTPND in the Message Control Register of message object 15 (last message) has been set. The last message object has the highest interrupt priority of all message objects. <sup>1)</sup>
	(02 + N) <b>Message N Interrupt:</b> Bit INTPND in the Message Control Register of message object 'N' has been set (N = 1 ... 14). Note that a message interrupt code is only displayed, if there is no other interrupt request with a higher priority. <sup>1)</sup> Example: message 1: INTID = 03 <sub>H</sub> , message 14: INTID = 10 <sub>H</sub>
<b>IPC</b>	<b>Interface Port Control</b> (reset value = 111 <sub>B</sub> , i.e. no port connection) The encoding of bitfield IPC is described in <a href="#">Section 19.7</a> . <i>Note: Bitfield IPC can be written only while bit CCE is set.</i>

<sup>1)</sup> Bit INTPND of the corresponding message object has to be cleared to give messages with a lower priority the possibility to update INTID or to reset INTID to "00<sub>H</sub>" (idle state).

### 19.2.2 Configuration of the Bit Timing

According to the CAN protocol specification, a bit time is subdivided into four segments: Sync segment, propagation time segment, phase buffer segment 1 and phase buffer segment 2.

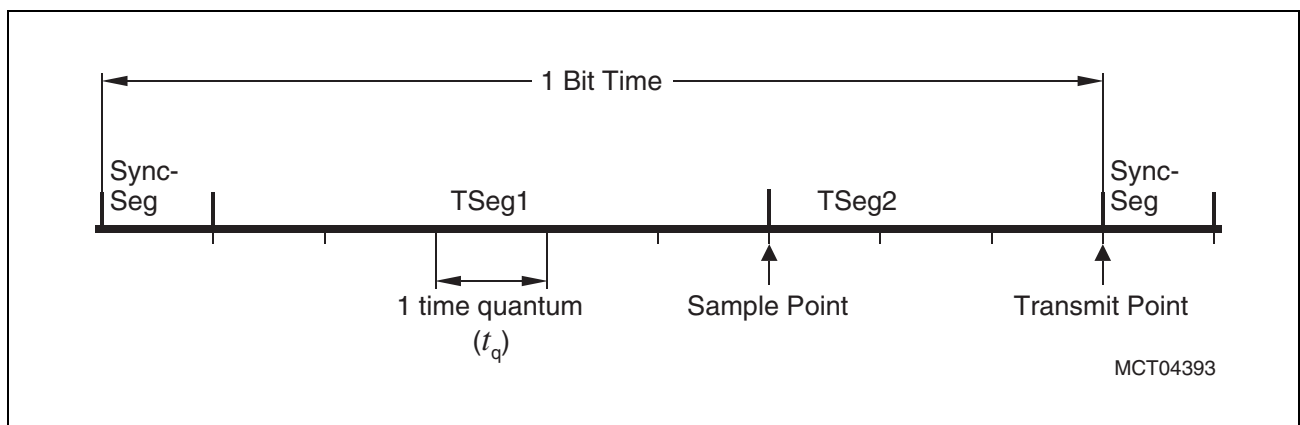
Each segment is a multiple of the time quantum  $t_q$ , with

$$t_q = (\text{BRP} + 1) \times 2^{(1 - \text{CPS})} \times t_{\text{XCLK}}$$

*Note: The CAN module is connected to the CPU clock signal, therefore  $t_{\text{XCLK}} = t_{\text{CPU}}$ .*

The Synchronization Segment (Sync Seg) is always 1  $t_q$  long. The Propagation Time Segment and the Phase Buffer Segment 1 (combined to TSeg1) define the time before the sample point, while Phase Buffer Segment 2 (TSeg2) defines the time after the sample point. The length of these segments is programmable (except Sync-Seg) via the Bit Timing Register (BTR).

*Note: For exact definition of these segments please refer to the CAN Protocol Specification.*



**Figure 19-4 Bit Timing Definition**

The bit time is determined by the XBUS clock period  $t_{\text{XCLK}}$ , the Baud Rate Prescaler, and the number of time quanta per bit:

$$\text{bit time} = t_{\text{Sync-Seg}} + t_{\text{TSeg1}} + t_{\text{TSeg2}} \quad [19.1]$$

$$t_{\text{Sync-Seg}} = 1 \times t_q$$

$$t_{\text{TSeg1}} = (\text{TSEG1} + 1) \times t_q$$

$$t_{\text{TSeg2}} = (\text{TSEG2} + 1) \times t_q$$

$$t_q = (\text{BRP} + 1) \times 2^{(1 - \text{CPS})} \times t_{\text{XCLK}} \quad [19.2]$$

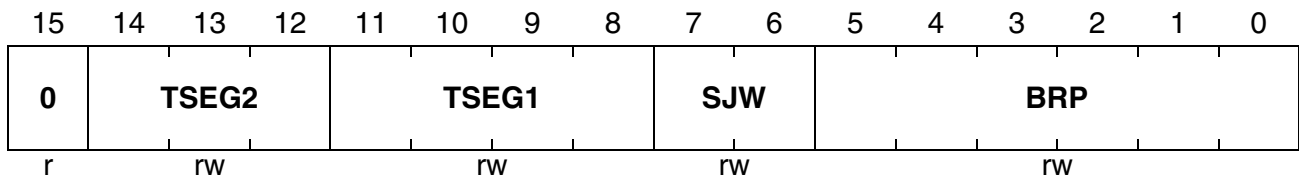
*Note: TSEG1, TSEG2, and BRP are the programmed numerical values from the respective fields of the Bit Timing Register.*

**BTR**

**Bit Timing Register**

**XReg (EF04<sub>H</sub>)**

**Reset Value: UUUU<sub>H</sub>**



Bit	Function
<b>BRP</b>	<b>Baud Rate Prescaler</b> For generating the bit time quanta the CPU frequency $f_{CPU}$ is divided by $2^{(1 - CPS)} \times (BRP + 1)$ . See also the prescaler control bit CPS in register CSR.
<b>SJW</b>	<b>(Re)Synchronization Jump Width</b> Adjust the bit time by maximum (SJW + 1) time quanta for resynchronization.
<b>TSEG1</b>	<b>Time Segment before sample point</b> There are (TSEG1 + 1) time quanta before the sample point. Valid values for TSEG1 are “2 ... 15”.
<b>TSEG2</b>	<b>Time Segment after sample point</b> There are (TSEG2 + 1) time quanta after the sample point. Valid values for TSEG2 are “1 ... 7”.

*Note: This register can only be written, if the config. change enable bit (CCE) is set.*

**Hard Synchronization and Resynchronization**

To compensate phase shifts between clock oscillators of different CAN controllers, any CAN controller has to synchronize on any edge from recessive to dominant bus level if the edge lies between a Sample Point and the next Synchronization Segment, and on any other edge if it itself does not send a dominant level. If the Hard Synchronization is enabled (at the Start of Frame), the bit time is restarted at the Synchronization Segment, otherwise the Resynchronization Jump Width (SJW) defines the maximum number of time quanta by which a bit time may be shortened or lengthened during one Resynchronization. The current bit time is adjusted by

$$t_{SJW} = (SJW + 1) \times t_q$$

*Note: SJW is the programmed numerical value from the respective field of the Bit Timing Register.*

### Calculation of the Bit Time

The programming of the bit time according to the CAN Specification depends on the desired baudrate, the XCLK frequency, and on the external physical delay times of the bus driver, of the bus line and of the input comparator. These delay times are summarized in the Propagation Time Segment  $t_{Prop}$ , where

$t_{Prop}$  is two times the maximum of the sum of physical bus delay, the input comparator delay, and the output driver delay rounded up to the nearest multiple of  $t_q$ .

To fulfill the requirements of the CAN specification, the following conditions must be met:

$$t_{TSeg2} \geq 2 \times t_q = \text{Information Processing Time}$$

$$t_{TSeg2} \geq t_{SJW}$$

$$t_{TSeg1} \geq 3 \times t_q$$

$$t_{TSeg1} \geq t_{SJW} + t_{Prop}$$

*Note: In order to achieve correct operation according to the CAN protocol the total bit time should be at least  $8 t_q$ , i.e.  $TSEG1 + TSEG2 \geq 5$ .*

*So, to operate with a baudrate of 1 MBit/sec, the XCLK frequency has to be at least 8/16 MHz (depending on the prescaler control bit CPS in register CSR).*

The maximum tolerance  $df$  for XCLK depends on the Phase Buffer Segment 1 (PB1), the Phase Buffer Segment 2 (PB2), and the Resynchronization Jump Width (SJW):

$$df \leq \frac{\min(\text{PB1}, \text{PB2})}{2 \times (13 \times \text{bit time} - \text{PB2})}$$

AND

$$df \leq \frac{t_{SJW}}{20 \times \text{bit time}}$$

The examples below show how the bit timing is to be calculated under specific circumstances.

### Bit Timing Example for High Baudrate

This example makes the following assumptions:

- XCLK frequency = 20 MHz
- BRP = 00, CPS = 0
- Baudrate = 1 Mbit/sec

$t_q$	100 ns	= $2 \times t_{XCLK}$
bus driver delay	50 ns	
receiver circuit delay	30 ns	
bus line (40 m) delay	220 ns	
$t_{Prop}$	600 ns	= $6 \times t_q$
$t_{SJW}$	100 ns	= $1 \times t_q$
$t_{TSeg1}$	700 ns	= $t_{Prop} + t_{SJW}$
$t_{TSeg2}$	200 ns	= <i>Information Processing Time</i>
$t_{Sync}$	100 ns	= $1 \times t_q$
$t_{Bit}$	1000 ns	= $t_{Sync} + t_{TSeg1} + t_{TSeg2}$
tolerance for $t_{XCLK}$	0.39%	= $\frac{\min(PB1, PB2)}{2 \times (13 \times \text{bit time} - PB2)}$
		= $\frac{0,1\mu s}{2 \times (13 \times 1\mu s - 0,2\mu s)}$

### Bit Timing Example for Low Baudrate

This example makes the following assumptions:

- XCLK frequency = 4 MHz
- BRP = 01, CPS = 0
- Baudrate = 100 kbit/s

$t_q$	1 $\mu s$	= $4 \times t_{XCLK}$
bus driver delay	200 ns	
receiver circuit delay	80 ns	
bus line (40 m) delay	220 ns	
$t_{Prop}$	1 $\mu s$	= $1 \times t_q$
$t_{SJW}$	4 $\mu s$	= $4 \times t_q$
$t_{TSeg1}$	5 $\mu s$	= $t_{Prop} + t_{SJW}$
$t_{TSeg2}$	4 $\mu s$	= <i>Information Processing Time</i> + $2 \times t_q$
$t_{Sync}$	1 $\mu s$	= $1 \times t_q$
$t_{Bit}$	10 $\mu s$	= $t_{Sync} + t_{TSeg1} + t_{TSeg2}$
tolerance for $f_{XCLK}$	1.58%	= $\frac{\min(PB1, PB2)}{2 \times (13 \times \text{bit time} - PB2)}$
		= $\frac{4\mu s}{2 \times (13 \times 10\mu s - 4\mu s)}$

### 19.2.3 Mask Registers

Messages can use standard or extended identifiers. Incoming frames are masked with their appropriate global masks. Bit IDE of the incoming message determines, if the standard 11-bit mask in Global Mask Short (GMS) is to be used, or the 29-bit extended mask in Global Mask Long (UGML&LGML). Bits holding a “0” mean “don’t care”, i.e. do not compare the message’s identifier in the respective bit position.

The last message object (15) has an additional individually programmable acceptance mask (Mask of Last Message, UMLM&LMLM) for the complete arbitration field. This allows classes of messages to be received in this object by masking some bits of the identifier.

*Note: The Mask of Last Message is ANDed with the Global Mask that corresponds to the incoming message.*

#### GMS

##### Global Mask Short

XReg (EF06<sub>H</sub>)

Reset Value: UFUU<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID20 ... 18			1	1	1	1	1	ID28 ... 21							
rw			r	r	r	r	r	rw							

Bit	Function
ID28 ... 18	<b>Identifier (11-bit)</b> Mask to filter incoming messages with standard identifier.

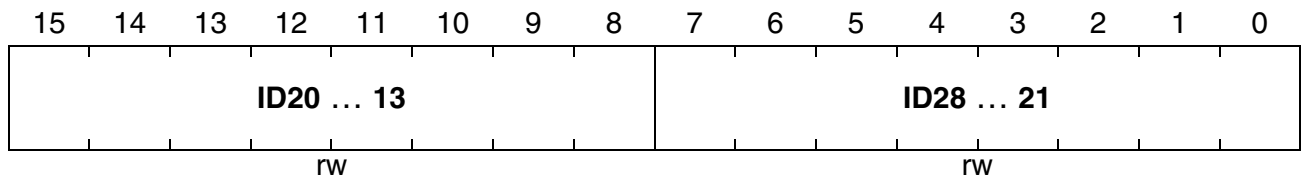
The On-Chip CAN Interface

**UGML**

Upper Global Mask Long

XReg (EF08<sub>H</sub>)

Reset Value: UUUU<sub>H</sub>

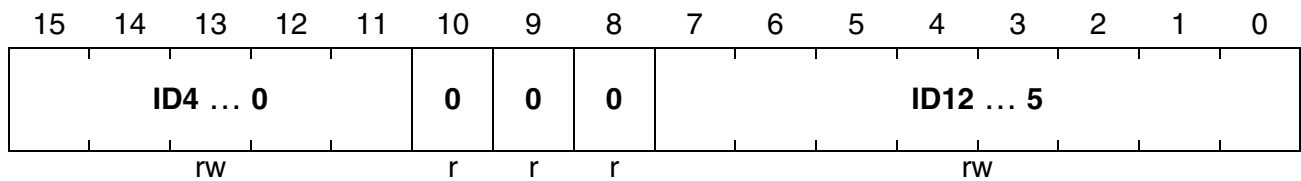


**LGML**

Lower Global Mask Long

XReg (EF0A<sub>H</sub>)

Reset Value: UUUU<sub>H</sub>



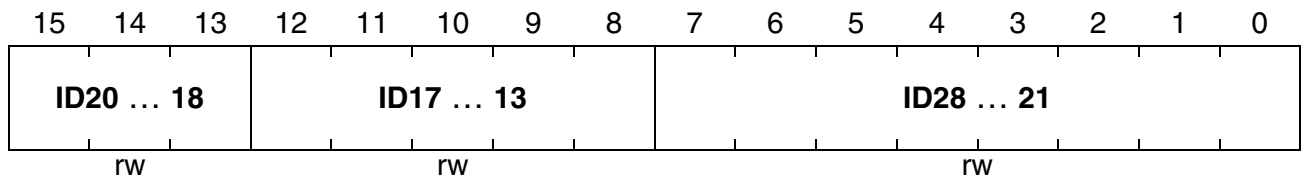
Bit	Function
ID28 ... 0	<b>Identifier (29-bit)</b> Mask to filter incoming messages with extended identifier.



The On-Chip CAN Interface

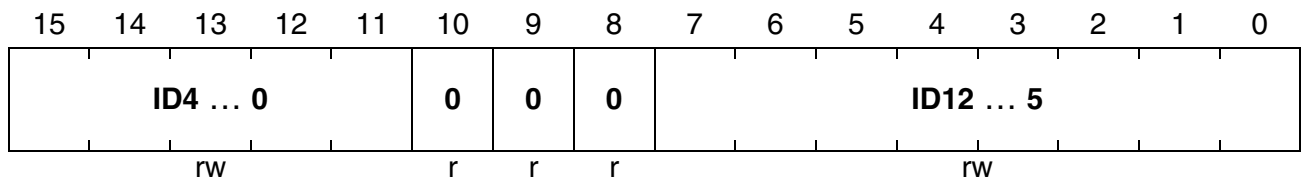
**UMLM**

**Upper Mask of Last Message**      XReg (EF0C<sub>H</sub>)      Reset Value: UUUU<sub>H</sub>



**LMLM**

**Lower Mask of Last Message**      XReg (EF0E<sub>H</sub>)      Reset Value: UUUU<sub>H</sub>

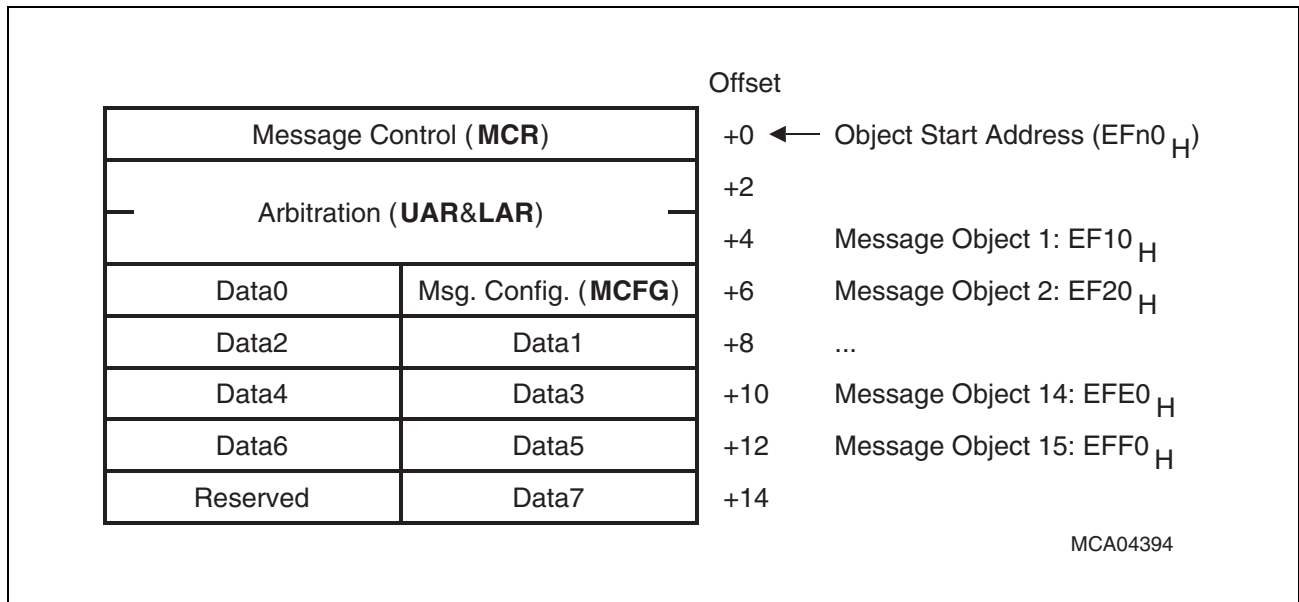


Bit	Function
<b>ID28 ... 0</b>	<p><b>Identifier (29-bit)</b> Mask to filter the last incoming message (Nr. 15) with standard or extended identifier (as configured).</p>

### 19.3 The Message Object

The message object is the primary means of communication between CPU and CAN controller. Each of the 15 message objects uses 15 consecutive bytes (see [Figure 19-5](#)) and starts at an address that is a multiple of 16.

*Note: All message objects must be initialized by the CPU, even those which are not going to be used, before clearing the INIT bit.*



**Figure 19-5 Message Object Address Map**

The general properties of a message object are defined via the Message Control Register (MCR). There is a dedicated register MCR<sub>n</sub> for each message object n.

Each element of the Message Control Register is made of two complementary bits. This special mechanism allows the selective setting or resetting of specific elements (leaving others unchanged) without requiring read-modify-write cycles. None of these elements will be affected by reset.

[Table 19-1](#) shows how to use and interpret these 2-bit fields.

**Table 19-1 MCR Bitfield Encoding**

Value	Function on Write	Meaning on Read
0 0	– reserved –	– reserved –
0 1	Reset element	Element is reset
1 0	Set element	Element is set
1 1	Leave element unchanged	– reserved –

The On-Chip CAN Interface

MCRn

Message Control Register

XReg (EFn0<sub>H</sub>)

Reset Value: UUUU<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RMT PND		TXRQ		MSGLST CPUUPD		NEWDAT		MSGVAL		TXIE		RXIE		INTPND	
rw		rw		rw		rw		rw		rw		rw		rw	

Bit	Function
INTPND	<b>Interrupt Pending</b> Indicates, if this message object has generated an interrupt request (see TXIE and RXIE), since this bit was last reset by the CPU, or not.
RXIE	<b>Receive Interrupt Enable</b> Defines, if bit INTPND is set after successful reception of a frame.
TXIE	<b>Transmit Interrupt Enable</b> Defines, if bit INTPND is set after successful transmission of a frame. <sup>1)</sup>
MSGVAL	<b>Message Valid</b> Indicates, if the corresponding message object is valid or not. The CAN controller only operates on valid objects. Message objects can be tagged invalid, while they are changed, or if they are not used at all.
NEWDAT	<b>New Data</b> Indicates, if new data has been written into the data portion of this message object by CPU (transmit-objects) or CAN controller (receive-objects) since this bit was last reset, or not. <sup>2)</sup>
MSGLST	<b>Message Lost</b> (This bit applies to <u>receive</u> -objects only!) Indicates that the CAN controller has stored a new message into this object, while NEWDAT was still set, i.e. the previously stored message is lost.
CPUUPD	<b>CPU Update</b> (This bit applies to <u>transmit</u> -objects only!) Indicates that the corresponding message object may not be transmitted now. The CPU sets this bit in order to inhibit the transmission of a message that is currently updated, or to control the automatic response to remote requests.
TXRQ	<b>Transmit Request</b> Indicates that the transmission of this message object is requested by the CPU or via a remote frame and is not yet done. TXRQ can be disabled by CPUUPD. <sup>1)3)</sup>

Bit	Function
<b>RMTPND</b>	<p><b>Remote Pending</b> (Used for transmit-objects)</p> <p>Indicates that the transmission of this message object has been requested by a remote node, but the data has not yet been transmitted. When RMTPND is set, the CAN controller also sets TXRQ. RMTPND and TXRQ are cleared, when the message object has been successfully transmitted.</p>

- 1) In message object 15 (last message) these bits are hardwired to “0” (inactive) in order to prevent transmission of message 15.
- 2) When the CAN controller writes new data into the message object, unused message bytes will be overwritten by non specified values. Usually the CPU will clear this bit before working on the data, and verify that the bit is still cleared once it has finished working to ensure that it has worked on a consistent set of data and not part of an old message and part of the new message.  
For transmit-objects the CPU will set this bit along with clearing bit CPUUPD. This will ensure that, if the message is actually being transmitted during the time the message was being updated by the CPU, the CAN controller will not reset bit TXRQ. In this way bit TXRQ is only reset once the actual data has been transferred.
- 3) When the CPU requests the transmission of a receive-object, a remote frame will be sent instead of a data frame to request a remote node to send the corresponding data frame. This bit will be cleared by the CAN controller along with bit RMTPND when the message has been successfully transmitted, if bit NEWDAT has not been set.  
If there are several valid message objects with pending transmission request, the message with the lowest message number is transmitted first. This arbitration is done when several objects are requested for transmission by the CPU, or when operation is resumed after an error frame or after arbitration has been lost.

## Arbitration Registers

The Arbitration Registers (UARn&LARn) are used for acceptance filtering of incoming messages and to define the identifier of outgoing messages. A received message with a matching identifier is accepted as a data frame (matching object has DIR = ‘0’) or as a remote frame (matching object has DIR = ‘1’). For matching, the corresponding Global Mask has to be considered (in case of message object 15 also the Mask of Last Message). Extended frames (using Global Mask Long) can be stored only in message objects with XTD = ‘1’, standard frames (using Global Mask Short) only in message objects with XTD = ‘0’.

Message objects should have unique identifiers, i.e. if some bits are masked out by the Global Mask Registers (i.e. “don’t care”), then the identifiers of the valid message objects should differ in the remaining bits which are used for acceptance filtering.

If a received message (data frame or remote frame) matches with more than one valid message object, it is associated with the object with the lowest message number. I.e. a received data frame is stored in the “lowest” object, or the “lowest” object is sent in response to a remote frame. The Global Mask is used for matching here.

The On-Chip CAN Interface

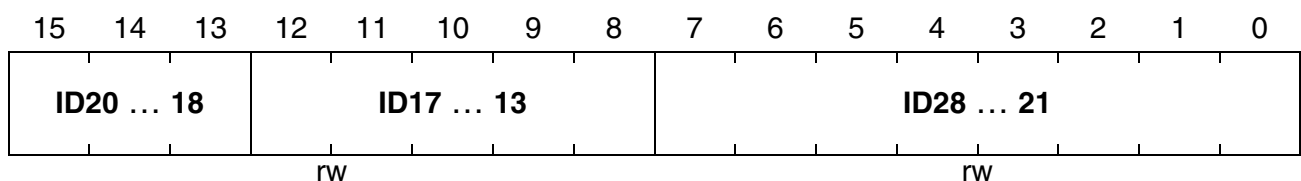
After a transmission (data frame or remote frame) the transmit request flag of the matching object with the lowest message number is cleared. The Global Mask is not used in this case.

**When the CAN controller accepts a data frame**, the complete message is stored into the corresponding message object, including the identifier (also masked bits, standard identifiers have bits ID17-0 filled with '0'), the data length code (DLC), and the data bytes (valid bytes indicated by DLC). This is implemented to keep the data bytes connected with the identifier, even if arbitration mask registers are used.

**When the CAN controller accepts a remote frame**, the corresponding transmit message object (1 ... 14) remains unchanged, except for bits TXRQ and RMTTPND, which are set, of course. In the last message object 15 (which cannot start a transmission) the identifier bits corresponding to the “don't care” bits of the Last Message Mask are copied from the received frame. Bits corresponding to the “don't care” bits of the corresponding global mask are not copied (i.e. bits masked out by the global **and** the last message mask cannot be retrieved from object 15).

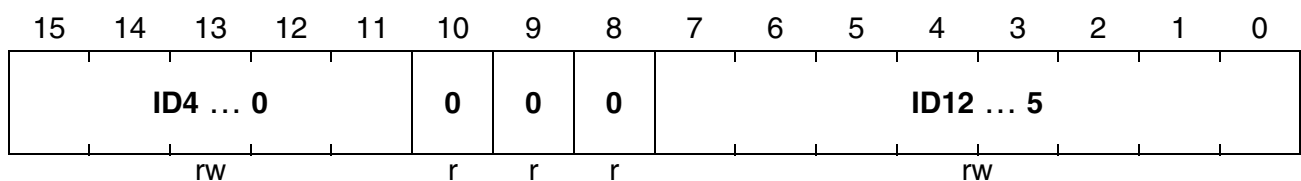
**UARn**

**Upper Arbitration Register**                      XReg (EFn2<sub>H</sub>)                      **Reset Value: UUUU<sub>H</sub>**



**LARn**

**Lower Arbitration Register**                      XReg (EFn4<sub>H</sub>)                      **Reset Value: UUUU<sub>H</sub>**



Bit	Function
ID28 ... 0	<b>Identifier (29-bit)</b> Identifier of a standard message (ID28 ... 18) or an extended message (ID28 ... 0). For standard identifiers bits ID17 ... 0 are “don't care”.

### Message Configuration

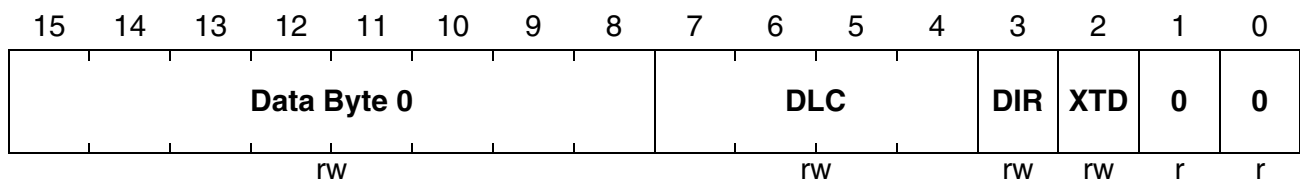
The Message Configuration Register (low byte of MCFGn) holds a description of the message within this object.

*Note: There is no “don’t care” option for bits XTD and DIR. So incoming frames can only match with corresponding message objects, either standard (XTD = 0) or extended (XTD = 1). Data frames only match with receive-objects, remote frames only match with transmit-objects.*

*When the CAN controller stores a data frame, it will write all the eight data bytes into a message object. If the data length code was less than 8, the remaining bytes of the message object will be overwritten by non specified values.*

### MCFGn

**Message Configuration Reg.      XReg (EFn6<sub>H</sub>)      Reset Value: - - UU<sub>H</sub>**



Bit	Function
<b>XTD</b>	<b>Extended Identifier</b> 0: <b>Standard</b> This message object uses a standard 11-bit identifier. 1: <b>Extended</b> This message object uses an extended 29-bit identifier.
<b>DIR</b>	<b>Message Direction</b> 0: <b>Receive Object.</b> On TXRQ, a remote frame with the identifier of this message object is transmitted. On reception of a data frame with matching identifier, that message is stored in this message object. 1: <b>Transmit Object.</b> On TXRQ, the respective message object is transmitted. On reception of a remote frame with matching identifier, the TXRQ and RMPND bits of this message object are set.
<b>DLC</b>	<b>Data Length Code</b> Defines the number of valid data bytes within the data area. Valid values for the data length are 0 ... 8.

*Note: The first data byte occupies the upper half of the message configuration register.*

**Data Area**

The data area occupies 8 successive byte positions after the Message Configuration Register, i.e. the data area of message object  $n$  covers locations  $00'EFn7_H$  through  $00'EFnE_H$ .

Location  $00'EFnF_H$  is reserved.

Message data for message object 15 (last message) will be written into a two-message-alternating buffer to avoid the loss of a message, if a second message has been received, before the CPU has read the first one.

**Handling of Message Objects**

The following diagrams summarize the actions that have to be taken in order to transmit and receive messages over the CAN bus. The actions taken by the CAN controller are described as well as the actions that have to be taken by the CPU (i.e. the servicing program).

The diagrams show:

- CAN controller handling of transmit objects
- CAN controller handling of receive objects
- CPU handling of transmit objects
- CPU handling of receive objects
- CPU handling of last message object
- Handling of the last message's alternating buffer

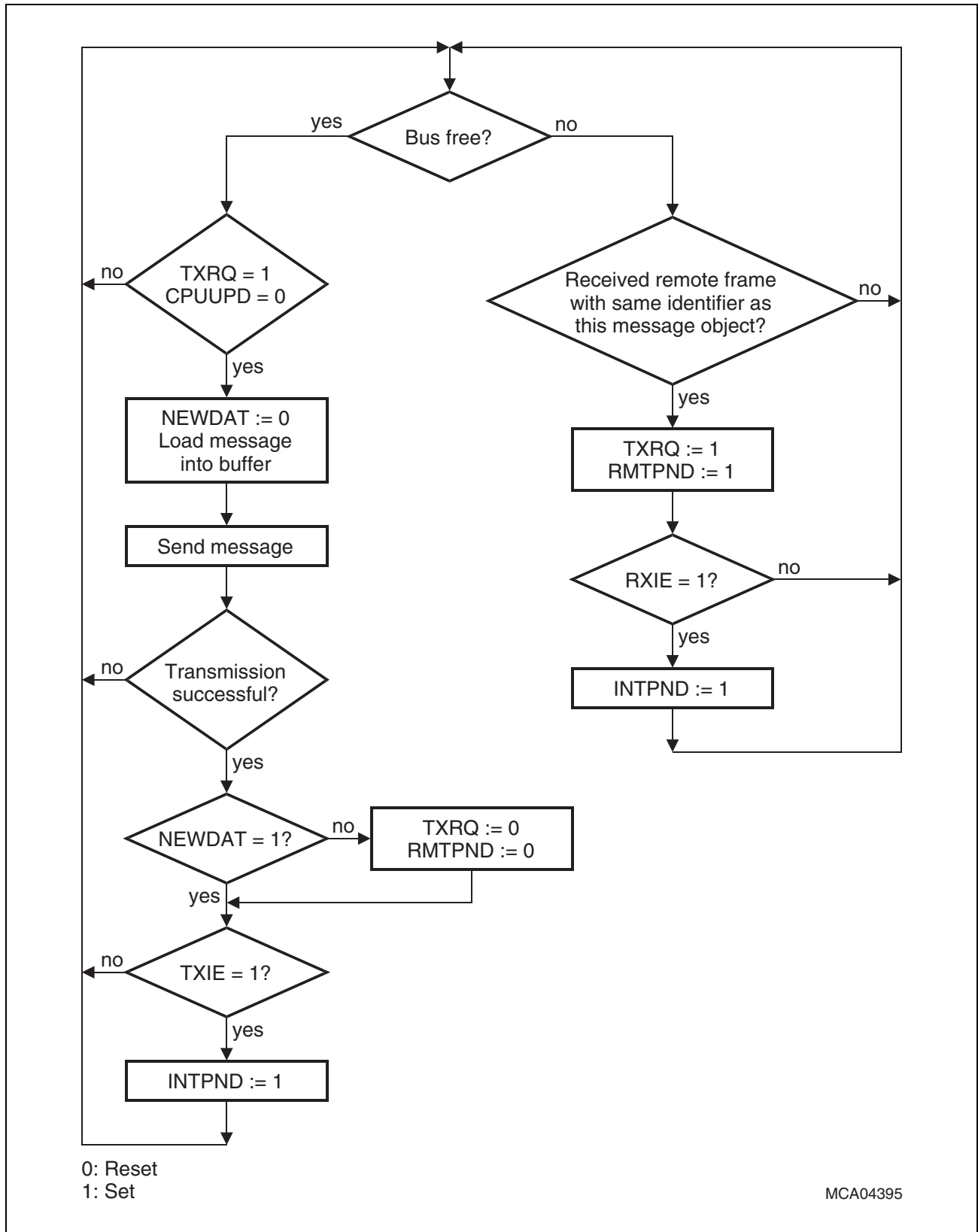


Figure 19-6 CAN Controller Handling of Transmit Objects (DIR = '1')



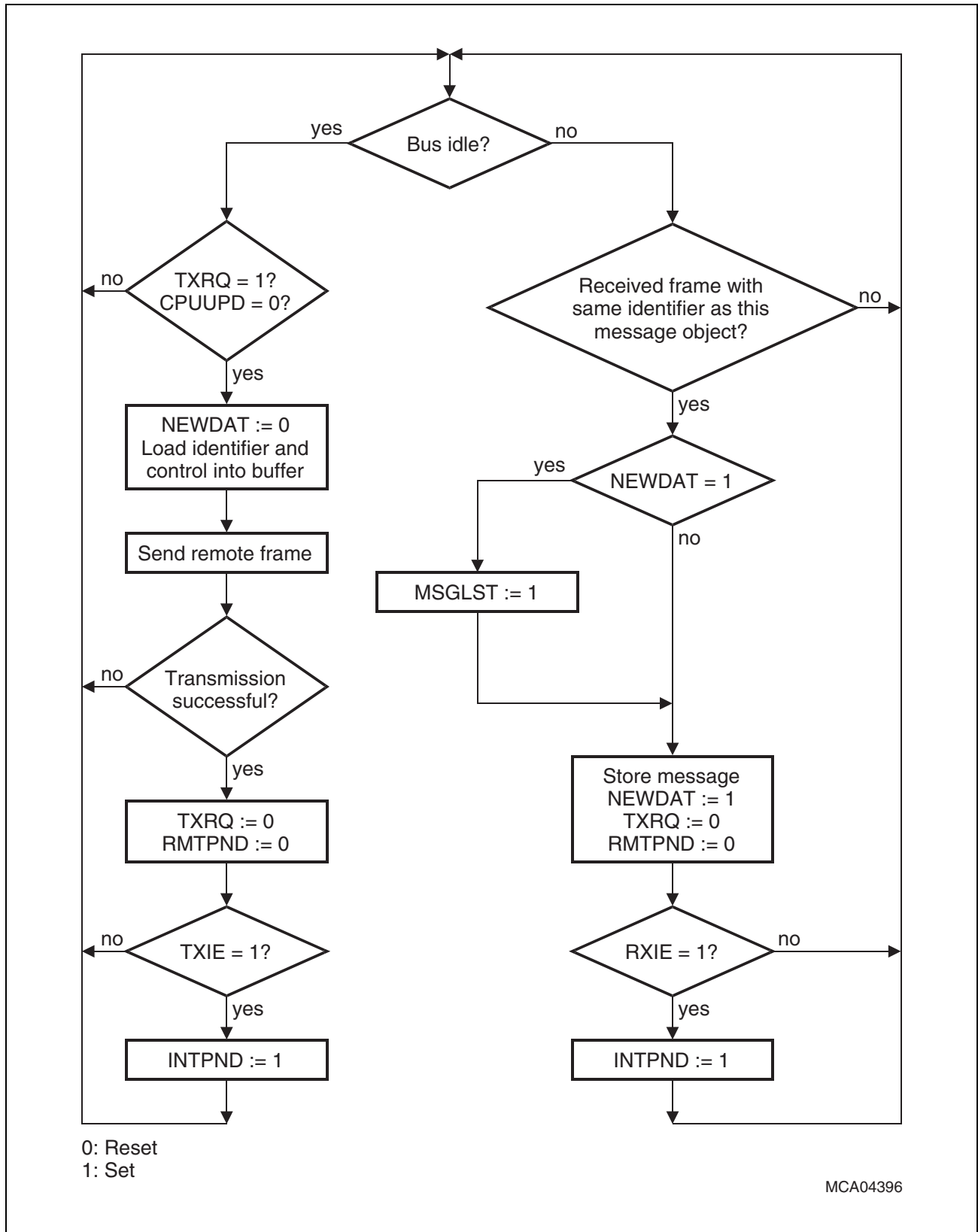


Figure 19-7 CAN Controller Handling of Receive Objects (DIR = '0')

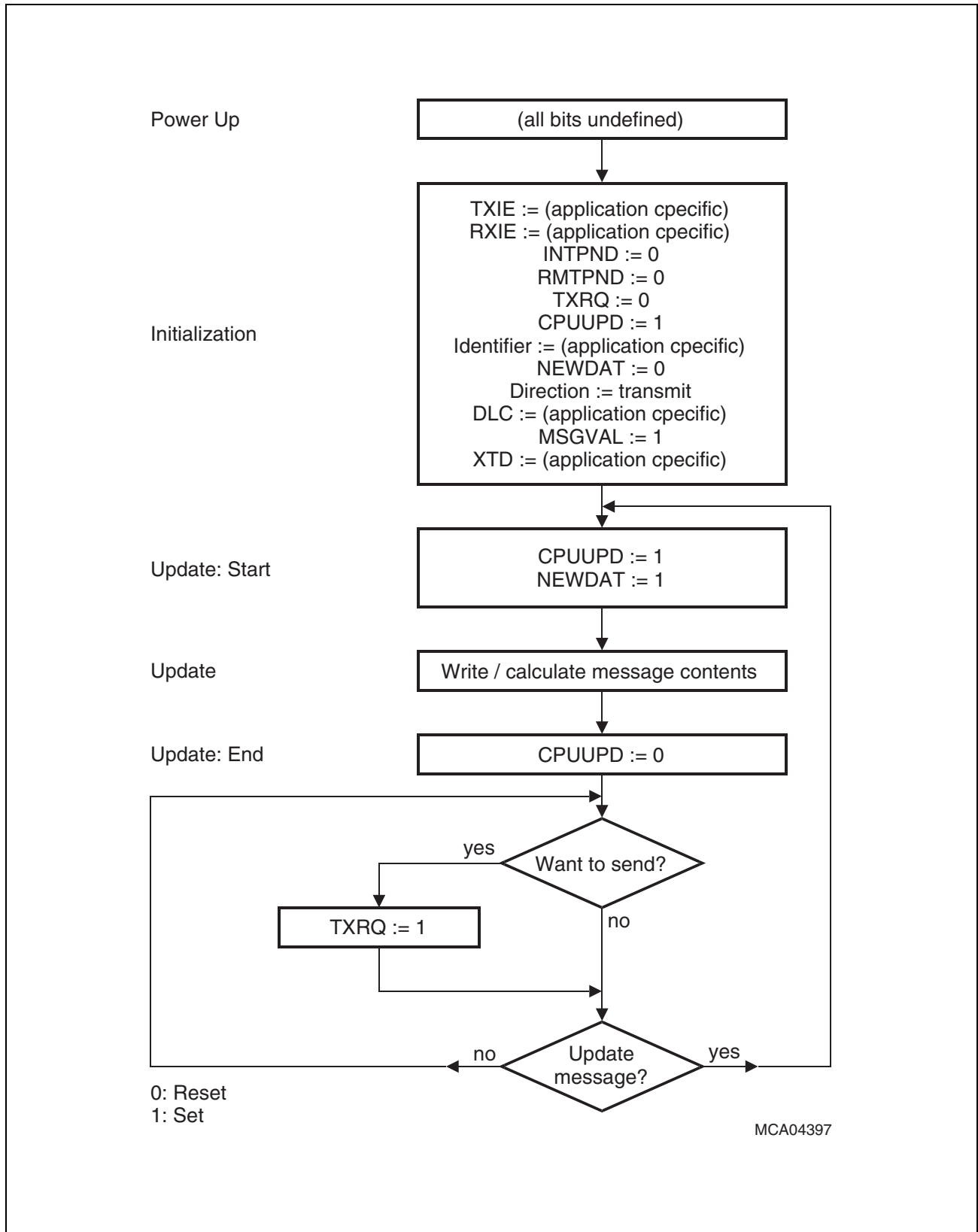


Figure 19-8 CPU Handling of Transmit Objects (DIR = '1')

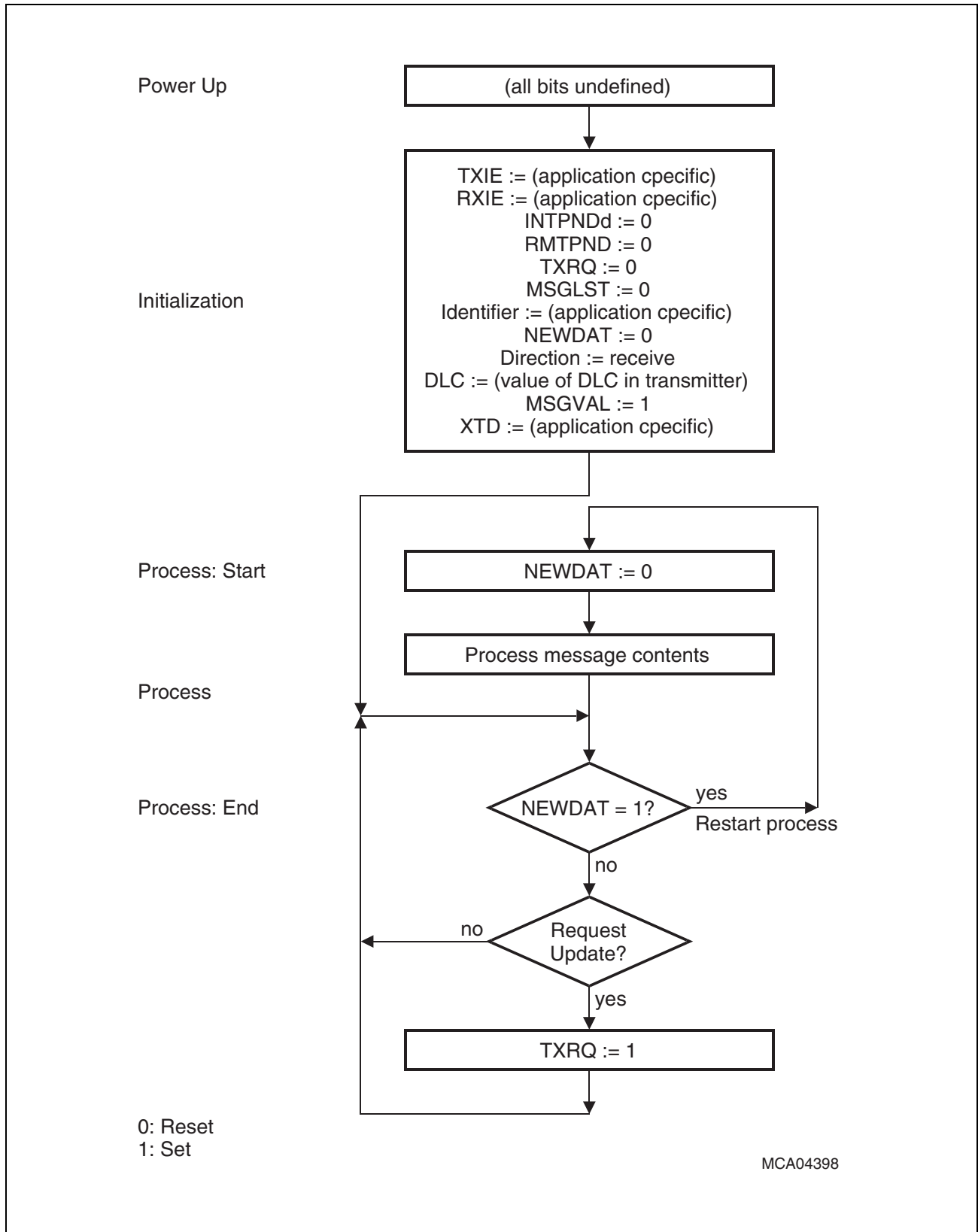


Figure 19-9 CPU Handling of Receive Objects (DIR = '0')

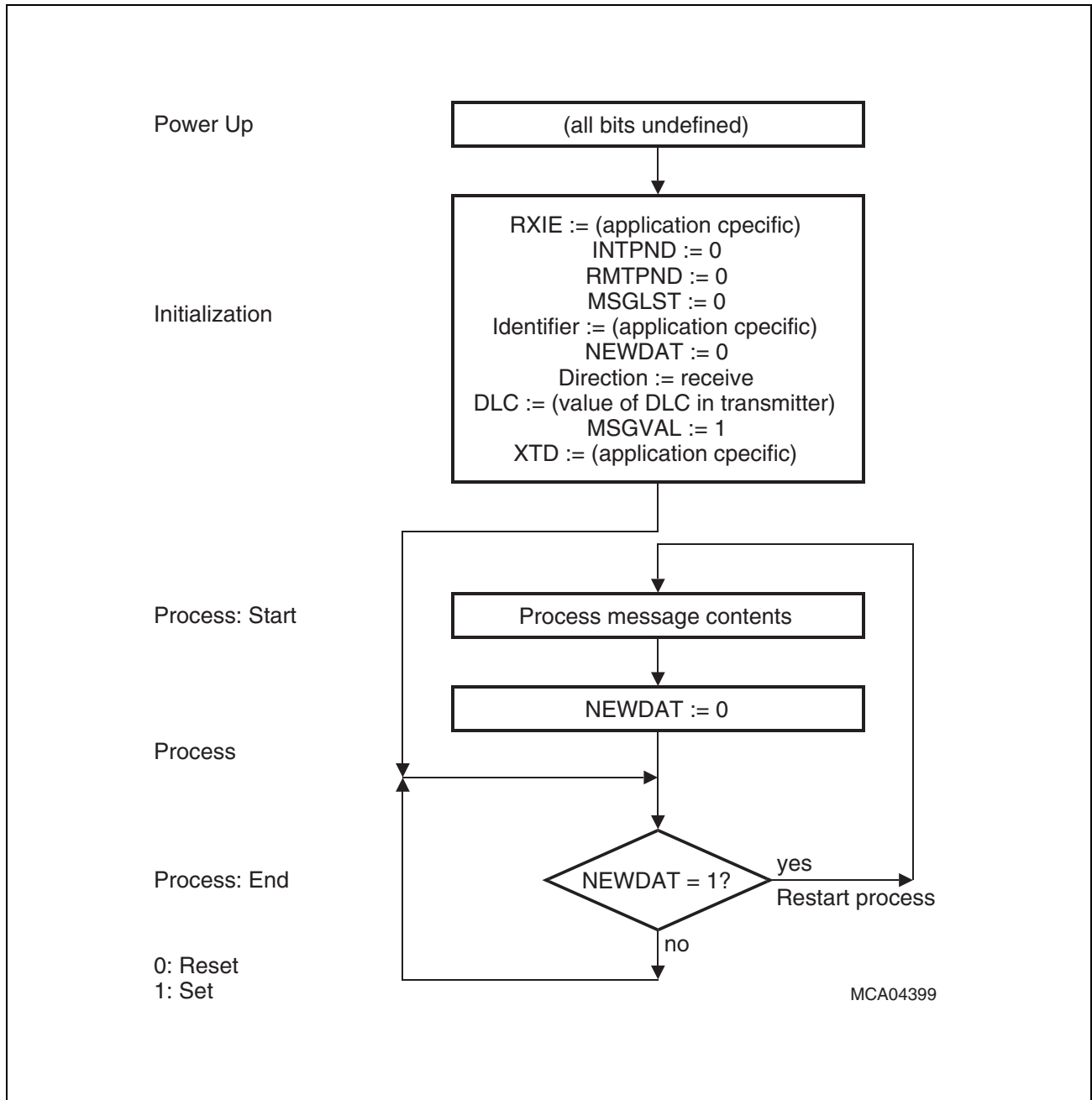


Figure 19-10 CPU Handling of the Last Message Object

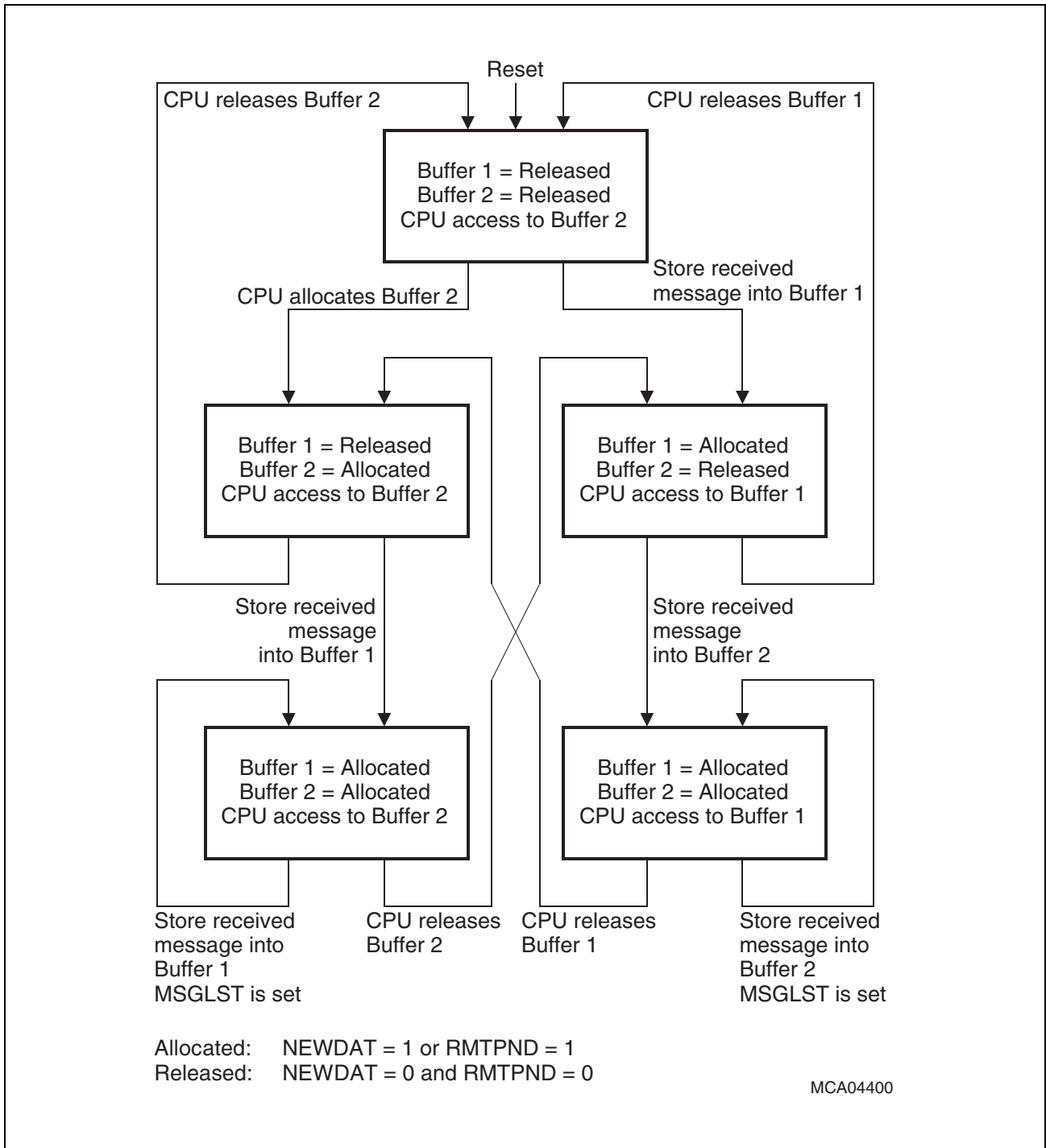


Figure 19-11 Handling of the Last Message Object's Alternating Buffer

## 19.4 Controlling the CAN Module

The CAN module is controlled by the C161CS/JC/JI via hardware signals (e.g. reset) and via register accesses executed by software.

### Accessing the On-Chip CAN Module

The CAN module is implemented as an X-Peripheral and is therefore accessed like an external memory or peripheral. That means that the registers of the CAN module can be read and written using 16-bit or 8-bit direct or indirect MEM addressing modes. Also bit handling is not supported via the XBUS. Since the XBUS, to which the CAN module is connected, also represents the external bus, CAN accesses follow the same rules and procedures as accesses to the external bus. CAN accesses cannot be executed in parallel to external instruction fetches or data read/writes, but are arbitrated and inserted into the external bus access stream.

Accesses to the CAN module use demultiplexed addresses, a 16-bit data bus (byte accesses possible), two waitstates and no tristate waitstate.

**The CAN address area** starts at 00'EF00<sub>H</sub> and covers 256 Bytes. This area is decoded internally, so none of the programmable address windows must be sacrificed in order to access the on-chip CAN module.

The advantage of locating the CAN address area in segment 0 is that the CAN module is accessible via data page 3, which is the 'system' data page, accessed usually through the 'system' data page pointer DPP3. In this way, the internal addresses, such like SFRs, internal RAM, and the CAN registers, are all located within the same data page and form a contiguous address space.

### Power Down Mode

If the C161CS/JC/JI enters Power Down mode, the XCLK signal will be turned off which will stop the operation of the CAN module. Any message transfer is interrupted. In order to ensure that the CAN controller is not stopped while sending a dominant level ('0') on the CAN bus, the CPU should set bit INIT in the Control Register prior to entering Power Down mode. The CPU can check if a transmission is in progress by reading bits TXRQ and NEWDAT in the message objects and bit TXOK in the Control Register. After returning from Power Down mode via hardware reset, the CAN module has to be reconfigured.

### Disabling the CAN Modules

When the CAN module is disabled by setting bit CANDISx in register SYSCON3 (peripheral management) no register accesses are possible. Also the module's logic blocks are stopped and no CAN bus transfers are possible. After re-enabling the CAN module (CANDISx = '0') it must be reconfigured (as after returning from Power Down mode).

**The On-Chip CAN Interface**

*Note: Incoming message frames can still be recognized (not received) in this case by monitoring the receive line CANx\_RXD. For this purpose the receive line CANx\_RXD can be connected to a fast external interrupt via register EXISEL.*

**CAN Module Reset**

The on-chip CAN module is connected to the XBUS Reset signal. This signal is activated, when the C161CS/JC/JI's reset input is activated, when a software reset is executed, and in case of a watchdog reset. Activating the CAN module's reset line triggers a hardware reset.

This hardware reset:

- disconnects the CAN\_TXD output from the port logic
- clears the error counters
- resets the busoff state
- switches the Control Register's low byte to 01<sub>H</sub>
- leaves the Control Register's high byte and the Interrupt Register undefined
- does not change the other registers including the message objects (notified as UUUU)

*Note: The first hardware reset after power-on leaves the **unchanged** registers in an **undefined** state, of course.*

*The value 01<sub>H</sub> in the Control Register's low byte prepares for the module initialization.*

**CAN Module Activation**

After a reset the CAN module is disabled. Before it can be used to receive or transmit messages the application software must activate the CAN module.

Three actions are required for this purpose:

- **General Module Enable** globally activates the CAN module. This is done by setting bit XPEN in register SYSCON after setting the corresponding selection bit in register XPERCON.
- **Pin Assignment** selects a pair of port pins that connect the CAN module to the external transceiver. This is done via bitfield IPC in register CSR.
- **Module Initialization** determines the functionality of the CAN module (baudrate, active objects, etc.). This is the major part of the activation and is described in the following.

## Module Initialization

The module initialization is enabled by setting bit INIT in the control register CSR. This can be done by the CPU via software, or automatically by the CAN controller on a hardware reset, or if the EML switches to busoff state.

While INIT is set:

- all message transfer from and to the CAN bus is stopped
- the CAN transmit line CAN\_TXD is "1" (recessive)
- the control bits NEWDAT and RMTPND of the last message object are reset
- the counters of the EML are left unchanged.

Setting bit CCE in addition, permits changing the configuration in the Bit Timing Register.

To initialize the CAN Controller, the following actions are required:

- configure the Bit Timing Register (CCE required)
- set the Global Mask Registers
- initialize each message object.

If a message object is not needed, it is sufficient to clear its message valid bit (MSGVAL), i.e. to define it as not valid. Otherwise, the whole message object has to be initialized.

After the initialization sequence has been completed, the CPU clears bit INIT.

Now the BSP synchronizes itself to the data transfer on the CAN bus by waiting for the occurrence of a sequence of 11 consecutive recessive bits (i.e. Bus Idle) before it can take part in bus activities and start message transfers.

The initialization of the message objects is independent of the state of bit INIT and can be done on the fly. The message objects should all be configured to particular identifiers or set to "not valid" before the BSP starts the message transfer, however.

To change the configuration of a message object during normal operation, the CPU first clears bit MSGVAL, which defines it as not valid. When the configuration is completed, MSGVAL is set again.



### Busoff Recovery Sequence

If the device goes *busoff*, it will set bit BOFF and also set bit INIT of its own accord, stopping all bus activities. To have the CAN module take part in the CAN bus activities again, the bus-off recovery sequence must be started by clearing the bit INIT (via software). Once INIT has been cleared, the module will then wait for 129 occurrences of *Bus idle* before resuming normal operation.

At the end of the *busoff* recovery sequence the Error Management Counters will be reset. This will automatically clear bits BOFF and EWRN.

During the waiting time after the resetting of INIT each time a sequence of 11 recessive bits has been monitored, a **Bit0Error** code is written to the Control Register, enabling the CPU to check up whether the CAN bus is stuck at dominant or continuously disturbed and to monitor the proceeding of the busoff recovery sequence.

*Note: An interrupt can be generated when entering the busoff state if bits IE and EIE are set. The corresponding interrupt code in bitfield INTID is 01<sub>H</sub>.*

*The busoff recovery sequence cannot be shortened by setting or resetting INIT.*

## 19.5 Configuration Examples for Message Objects

The two examples below represent standard applications for using CAN messages. Both examples assume that identifier and direction are already set up correctly.

The respective contents of the Message Control Register (MCR) are shown.

### Configuration Example of a Transmission Object

This object shall be configured for transmission. It shall be transmitted automatically in response to remote frames, but no receive interrupts shall be generated for this object.

**MCR** (Data bytes are not written completely → CPUUPD = '1')

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
01		01		10		01		10		01		01		01	
RMTPND		TXRQ		CPUUPD		NEWDAT		MSGVAL		TXIE		RXIE		INTPND	

**MCR** (Remote frame was received in the meantime → RMTPND = '1', TXRQ = '1')

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
10		10		10		01		10		01		01		01	
RMTPND		TXRQ		CPUUPD		NEWDAT		MSGVAL		TXIE		RXIE		INTPND	

After updating the message the CPU should clear CPUUPD and set NEWDAT. The previously received remote request will then be answered.

If the CPU wants to transmit the message actively it should also set TXRQ (which should otherwise be left alone).

**Configuration Example of a Reception Object**

This object shall be configured for reception. A receive interrupt shall be generated each time new data comes in. From time to time the CPU sends a remote request to trigger the sending of this data from a remote node.

**MCR** (Message object is idle, i.e. waiting for a frame to be received)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
01		01		01		01		10		01		10		01	
RMTPND		TXRQ		MSGLST		NEWDAT		MSGVAL		TXIE		RXIE		INTPND	

**MCR** (A data frame was received → NEWDAT = '1', INTPND = '1')

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
01		01		01		10		10		01		10		10	
RMTPND		TXRQ		MSGLST		NEWDAT		MSGVAL		TXIE		RXIE		INTPND	

To process the message the CPU should clear INTPND and NEWDAT, process the data, and check that NEWDAT is still clear after that. If not, the processing should be repeated.

To send a remote frame to request the data, simply bit TXRQ needs to be set. This bit will be cleared by the CAN controller, once the remote frame has been sent or if the data is received before the CAN controller could transmit the remote frame.

## 19.6 The Second CAN Module CAN2

Module CAN2 is basically identical with module CAN1. It provides the same set of message objects, operating modes, and registers. While the number, sequence, and function of the CAN2 registers are exactly the same as in module CAN1, module CAN2, of course, resides in a separate address window.

**Table 19-2 CAN Register Summary**

Register Locations in Module CAN2		Register Locations in Module CAN1	
C2CSR	EE00 <sub>H</sub>	C1CSR	EF00 <sub>H</sub>
C2PCIR	EE02 <sub>H</sub>	C1PCIR	EF02 <sub>H</sub>
C2BTR	EE04 <sub>H</sub>	C1BTR	EF04 <sub>H</sub>
C2GMS	EE06 <sub>H</sub>	C1GMS	EF06 <sub>H</sub>
C2UGML	EE08 <sub>H</sub>	C1UGML	EF08 <sub>H</sub>
C2LGML	EE0A <sub>H</sub>	C1LGML	EF0A <sub>H</sub>
C2UMLM	EE0C <sub>H</sub>	C1UMLM	EF0C <sub>H</sub>
C2LMLM	EE0E <sub>H</sub>	C1LMLM	EF0E <sub>H</sub>
C2MCR <sub>n</sub>	EEn0 <sub>H</sub>	C1MCR <sub>n</sub>	EFn0 <sub>H</sub>
C2UAR <sub>n</sub>	EEn2 <sub>H</sub>	C1UAR <sub>n</sub>	EFn2 <sub>H</sub>
C2LAR <sub>n</sub>	EEn4 <sub>H</sub>	C1LAR <sub>n</sub>	EFn4 <sub>H</sub>
C2MCFG <sub>n</sub>	EEn6 <sub>H</sub>	C1MCFG <sub>n</sub>	EFn6 <sub>H</sub>
Data area CAN2	EEn7 <sub>H</sub> ... EEnE <sub>H</sub>	Data area CAN1	EFn7 <sub>H</sub> ... EFnE <sub>H</sub>

The on-chip interrupt generation works in exactly the same way as in module CAN1. Module CAN2 is connected to a separate interrupt node. So each CAN module can be accessed and serviced independently.

**Table 19-3 CAN Interrupt Connection**

Module	Interrupt Node	Interrupt Flag	Interrupt Vector
CAN1	XP2IC	XP2IR	XP2INT
CAN2	XP7IC	XP7IR	XP7INT

It also uses a separate physical interface to connect to an external CAN bus (see [Section 19.7](#)).

## 19.7 The CAN Application Interface

The on-chip CAN modules of the C161CS/JC/JI are connected to the (external) physical layer (i.e. the CAN bus) via two signals each:

**Table 19-4 CAN Interface Signals**

CAN Signal	Port Pin	Function
CAN1_RXD	Controlled via C1PCIR.IPC	Receive data from the physical layer of the CAN bus 1.
CAN1_TXD		Transmit data to the physical layer of the CAN bus 1.
CAN2_RXD	Controlled via C2PCIR.IPC	Receive data from the physical layer of the CAN bus 2.
CAN2_TXD		Transmit data to the physical layer of the CAN bus 2.

*Note: The two interfaces may be combined on two port pins connecting to one single CAN bus.*

A logic low level ('0') is interpreted as the dominant CAN bus level, a logic high level ('1') is interpreted as the recessive CAN bus level.

### Connection to an External Transceiver

The two CAN modules of the C161CS/JC/JI can be connected to an external CAN bus via a CAN transceiver in several ways:

- **Separate Buses** permit communication on two independent CAN buses, e.g. with different baudrates. For this purpose the CAN interface lines must be assigned to separate pairs of port pins.
- **A Single Bus** can be connected, where both modules provide a total of 30 message objects. For this purpose the CAN interface lines must be assigned to the same pair of port pins.

*Note: Basically it is also possible to connect several CAN modules directly (on-board) without using CAN transceivers. The CAN modules may here reside on the same or on separate devices.*

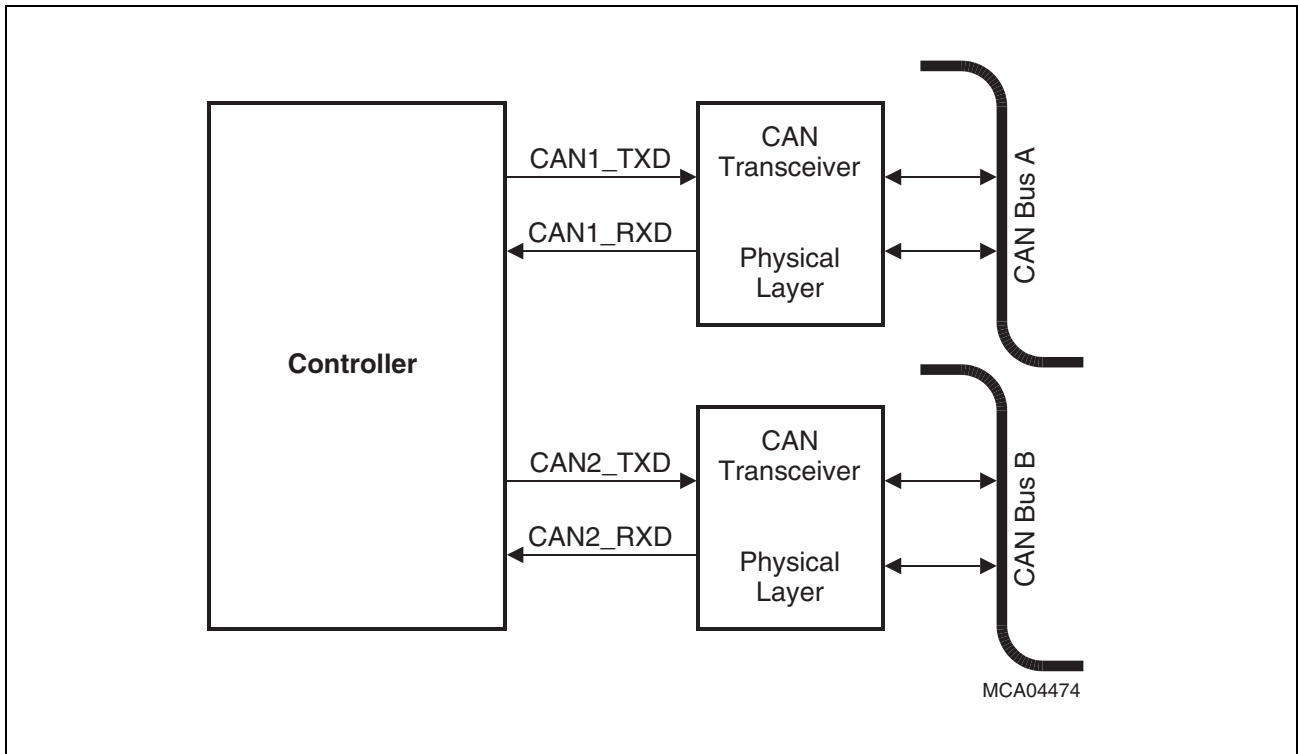


Figure 19-12 Connection to Separate CAN Buses

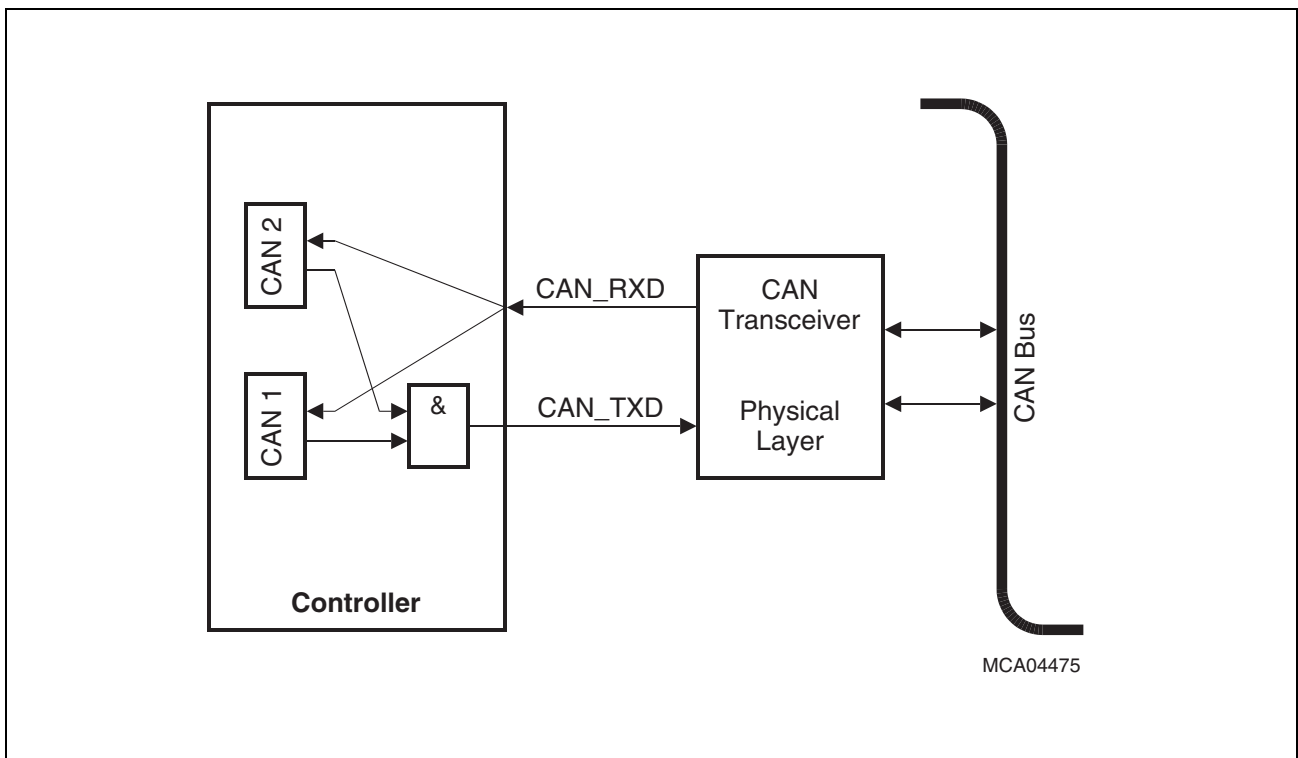


Figure 19-13 Connection to a Single CAN Bus

### Port Control

The receive data line and the transmit data line of the CAN module are alternate port functions. Make sure that the respective port pin for the receive line is switched to input in order to enable proper reception. The respective port driver for the transmit will automatically be switched ON.

This provides a standard pin configuration without additional software control and also works in emulation mode where the port direction registers cannot be controlled.

The receive and transmit line of the CAN module may be assigned to several port pins of the C161CS/JC/JI under software control. This assignment is selected via bitfield IPC (Interface Port Connection) in register PCIR.

**Table 19-5 Assignment of CAN Interface Lines to Port Pins**

IPC	CAN_RXD	CAN_TXD	Notes
000	P4.5 / P4.4	P4.6 / P4.7	Module specific assignments (CAN1 / CAN2). <sup>1)</sup>
001	P4.7	P4.6	Pins P4.5-0 available for segment address lines A21 ... A16 (4 MByte external address space).
010	P7.4	P7.5	Port 4 available for segment address lines A23 ... A16 (16 MByte external address space).
011	P7.6	P7.7	Port 4 available for segment address lines A23 ... A16 (16 MByte external address space).
100	–	–	<i>Reserved.</i> Do not use this combination.
101	–	–	<i>Reserved.</i> Do not use this combination.
110	–	–	<i>Reserved.</i> Do not use this combination.
111	Idle (recessive)	Disconnected	No port assigned. Default after Reset.

<sup>1)</sup> This assignment is compatible with previous derivatives where the assignment of CAN interface lines was fixed.

---

**The On-Chip CAN Interface**

The location of the CAN interface lines can now be selected via software according to the requirements of an application:

**Compatible Assignment** (IPC = 000<sub>B</sub>) makes the C161CS/JC/JI suitable for applications with a given hardware (board layout). The CAN interface lines are connected to the port pins to which they are hardwired in previous derivatives.

**Wide Address Assignment** (IPC = 001<sub>B</sub>) uses the two upper pins of Port 4, leaving room for six segment address lines (A21 ... A16). A contiguous external address space of 4 MByte is available in this case.

**Full Address Assignment** (IPC = 010<sub>B</sub> or 011<sub>B</sub>) removes the CAN interface lines completely from Port 4. The maximum external address space of 16 MByte is available in this case.

The CAN interface lines are mapped to Port 7. Two pairs of Port 7 pins can be selected.

**No Assignment** (IPC = 111<sub>B</sub>) disconnects the CAN interface lines from the port logic. This avoids undesired currents through the interface pin drivers while the C161CS/JC/JI is in a power saving state.

After reset the CAN interface lines are disconnected.

**Bus Sharing** internally combines the interface lines of both CAN modules (receive line is shared, transmit lines are ANDed). This provides up to 30 message objects (2 × 15) on a single physical CAN bus. Bus sharing is enabled by simply assigning both CAN interfaces to the same pair of port pins.

*Note: Assigning CAN interface signals to a port pin overrides the other alternate function of the respective pin (segment address on Port 4, CAPCOM lines on Port 7).*



## 20 The Serial Data Link Module (SDLM)

The Serial Data Link Module (SDLM) enables the C161CS/JC/JI to communicate with other stations via a J1850-based serial multiplexed bus, using an external J1850 bus transceiver chip. The module is conform to the SAE Class B J1850 specification and compatible to the class 2 protocol.

*Note: The SDLM is an XBUS peripheral and therefore requires bit XPEN in register SYSCON to be set in order to be operable.*

### J1850 Concept

The SAE Class-B specification establishes the requirements for a serial bus protocol used in automotive and industrial applications. Basically it describes the network's characteristics in three layers: the physical layer, the data link layer and the application layer.

The physical layer handles the frame transfer including bit/symbol encoding and timing. The data link layer defines the J1850 protocol in terms of frame elements, error detection, bus access, frame arbitration, and clock synchronization. Finally, the application layer needs to evaluate message screening/filtering by software and the handling of diagnostic parameters/codes.

J1850 is a multi-master based serial protocol. Each node has a local clock, which allows simultaneous access to the bus.

### General SDLM Features

- Compliant to SAE Class B J1850 specification
- GM class 2 protocol fully supported
- Variable Pulse Width (VPW) format with 10.4 kBaud
- High speed receive/transmit 4x mode with 41.6 kBaud
- Digital noise filter
- Power save mode and automatic wakeup upon bus activity
- Single-byte headers or consolidated headers supported
- CRC generation & check supported
- Receive and transmit block mode supported

### Data Link Operation Features

- 11 bytes transmit buffer
- Double-buffered 11 bytes receive buffer
- Support of In-frame response (IFR) types 1, 2, 3
- Automatic IFR transmission for IFR types 1, 2 for three byte consolidated headers
- Advanced interrupt handling for RX, TX and error conditions
- All interrupt sources can be separately enabled/disabled
- 8-Byte transmit FIFO and 16-Byte receive FIFO in block mode

*Note: The J1850 module does not support the Pulse Width Modulation (PWM) data format.*

**The Serial Data Link Module (SDLM)**

**20.1 Frame Format Basics**

In the “SAE Standard Class B Data Communication Network Interface protocol” the general J1850 frame format is defined as:

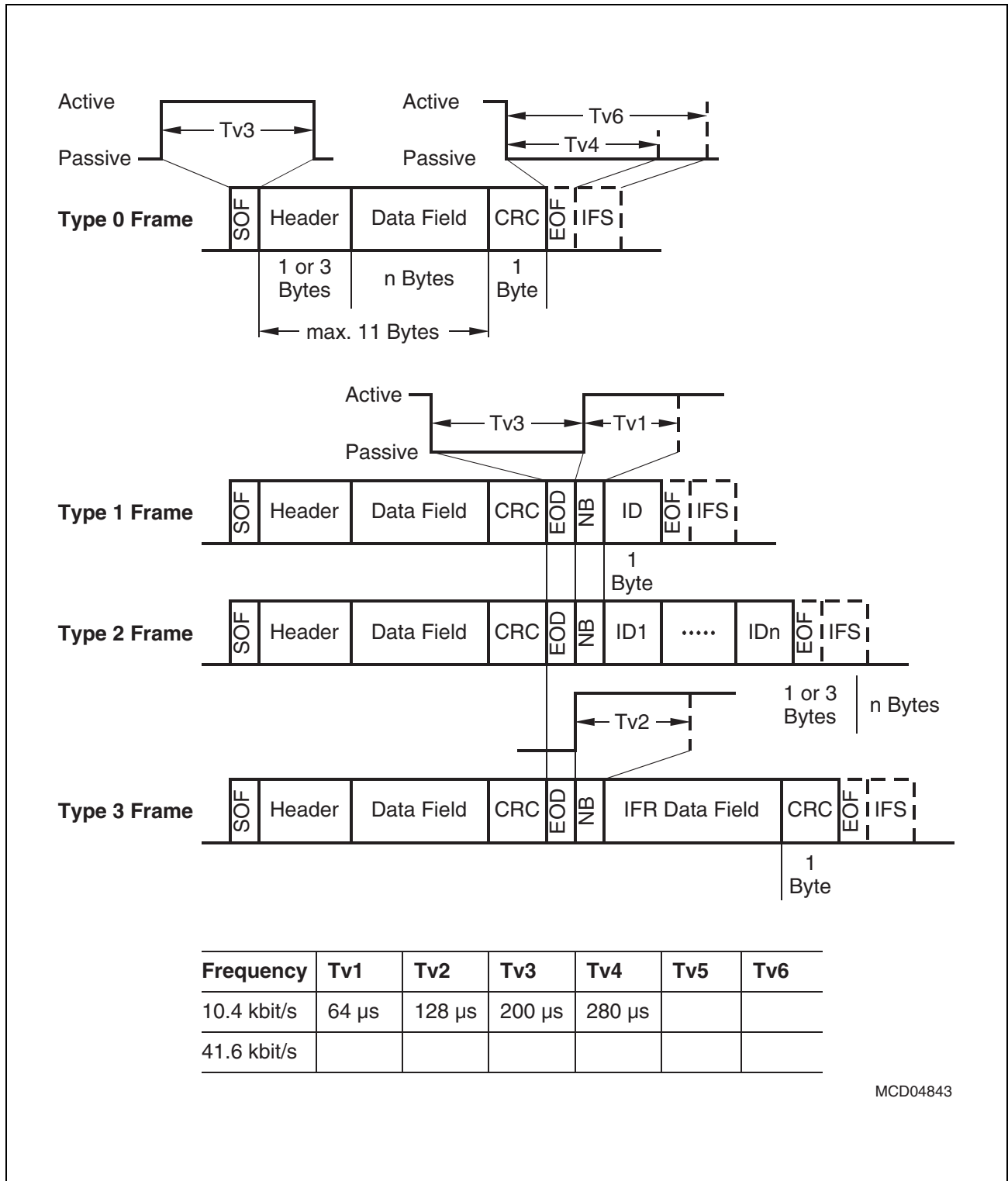
idle, SOF, DATA\_0, ..., Data\_N, CRC, EOD, NB, IFR\_1, ..., IFR\_N, EOF, IFS, idle

**Table 20-1 J1850 Frame Format Elements**

<b>Symbol</b>	<b>Name</b>	<b>Description</b>
<b>SOF</b>	Start of Frame	The SOF mark is used to uniquely identify the start of a frame. SOF is not used for CRC error calculation.
<b>DATA_0 - DATA_N</b>	Data bytes	Data bytes start with MSB first. The maximum frame length including header byte(s) and IFR byte(s) but excluding frame delimiters (SOF, EOD, EOF, and IFS) and CRC byte is 11 bytes.
<b>CRC</b>	CRC byte	The cyclic redundancy check byte is generated during transmission and is checked during reception.
<b>EOD</b>	End of Data	Indicates the end of a transmission by the originator of a frame. Directly after EOD an IFR can be started by the recipient(s) of the frame.
<b>NB</b>	Normalization Bit	Required for 10.4 kbps mode only; follows after an EOD and before an IFS symbol; defines the start of an in-frame response.
<b>IFR_1 - IFR_N</b>	In-Frame Response byte(s)	In Frame Response byte(s) (ID) can be sent by receiving devices after the sending device has sent an EOD.
<b>EOF</b>	End of Frame	This symbol defines the end of a frame.
<b>IFS</b>	Inter-Frame Separation	This symbol separates two consecutive frames.
<b>idle</b>	Idle state	The bus is idle if no transfer takes place (occurs before SOF or after IFS).

**The Serial Data Link Module (SDLM)**

The figures below illustrate the different J1850 frame formats as well as the used bits and symbols.



**Figure 20-1 J1850 Frame Formats**

The Serial Data Link Module (SDLM)

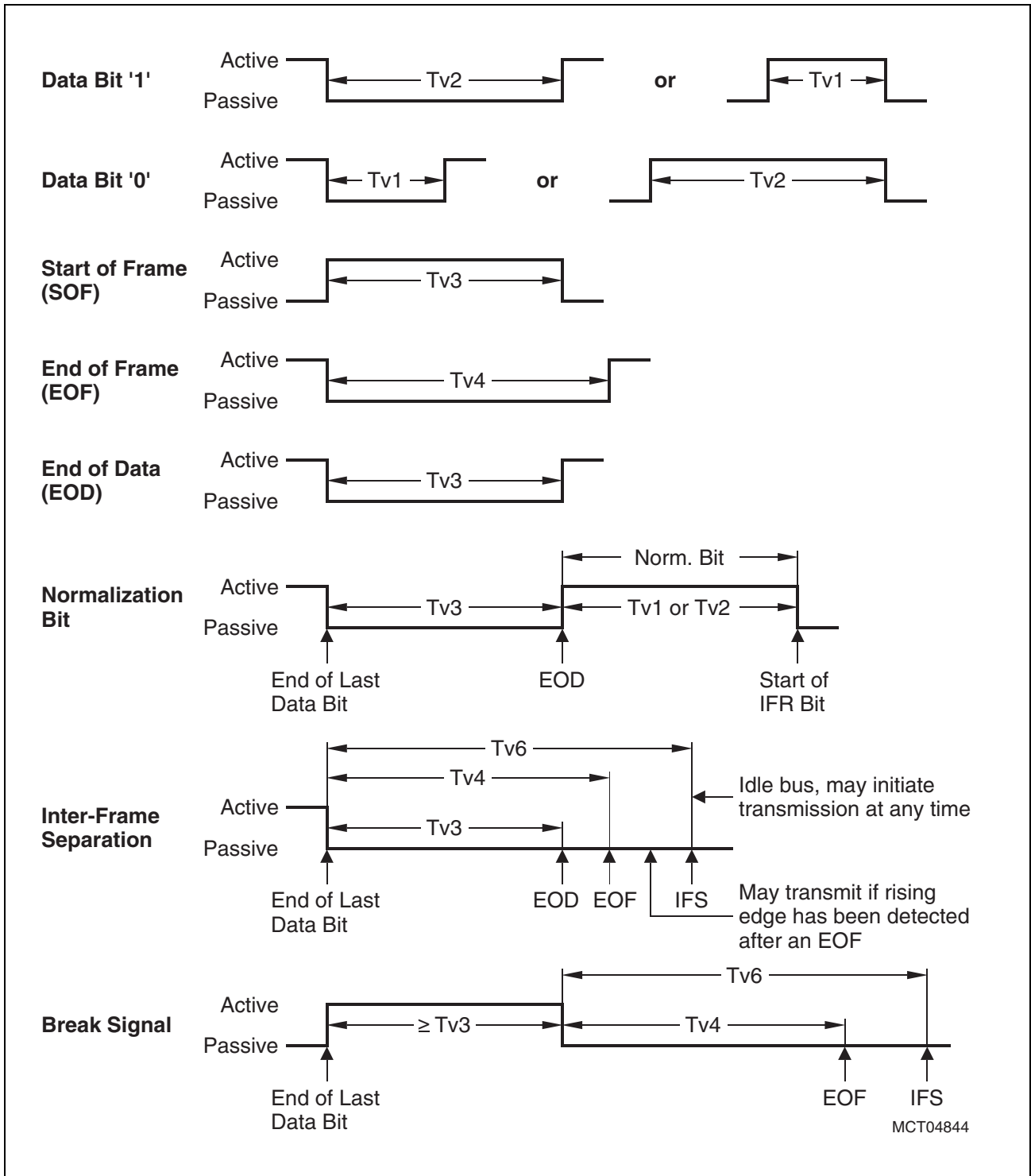


Figure 20-2 J1850 Variable Pulse Width (VPW) Format

### Frame Arbitration

The frame arbitration in J1850 compatible networks follows the concept of Carrier Sense Multiple Access (CSMA) with non-destructive message arbitration. When two nodes have access to the bus at the same time, the priority decision is made during transmission. The node which has won the arbitration will continue transmission and the other node will stop transmitting.

The SDLM always stores the current message on the bus into its bus-side receive buffer, even while transmitting (self-echo message).

In the example below two nodes are transmitting simultaneously. The transmitted data is different in the fourth bit location. Node 1 detects that its received bus signal is different to the transmitted signal and aborts its transmission.

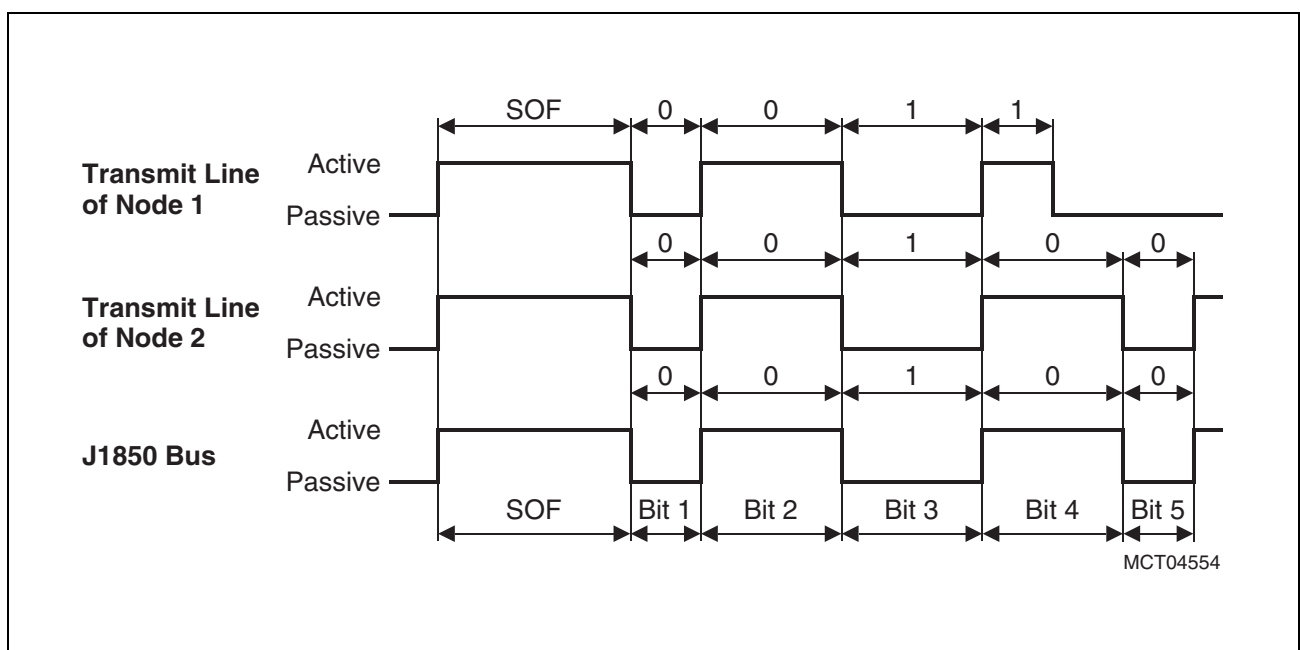


Figure 20-3 J1850 VPW Message Arbitration

The Serial Data Link Module (SDLM)

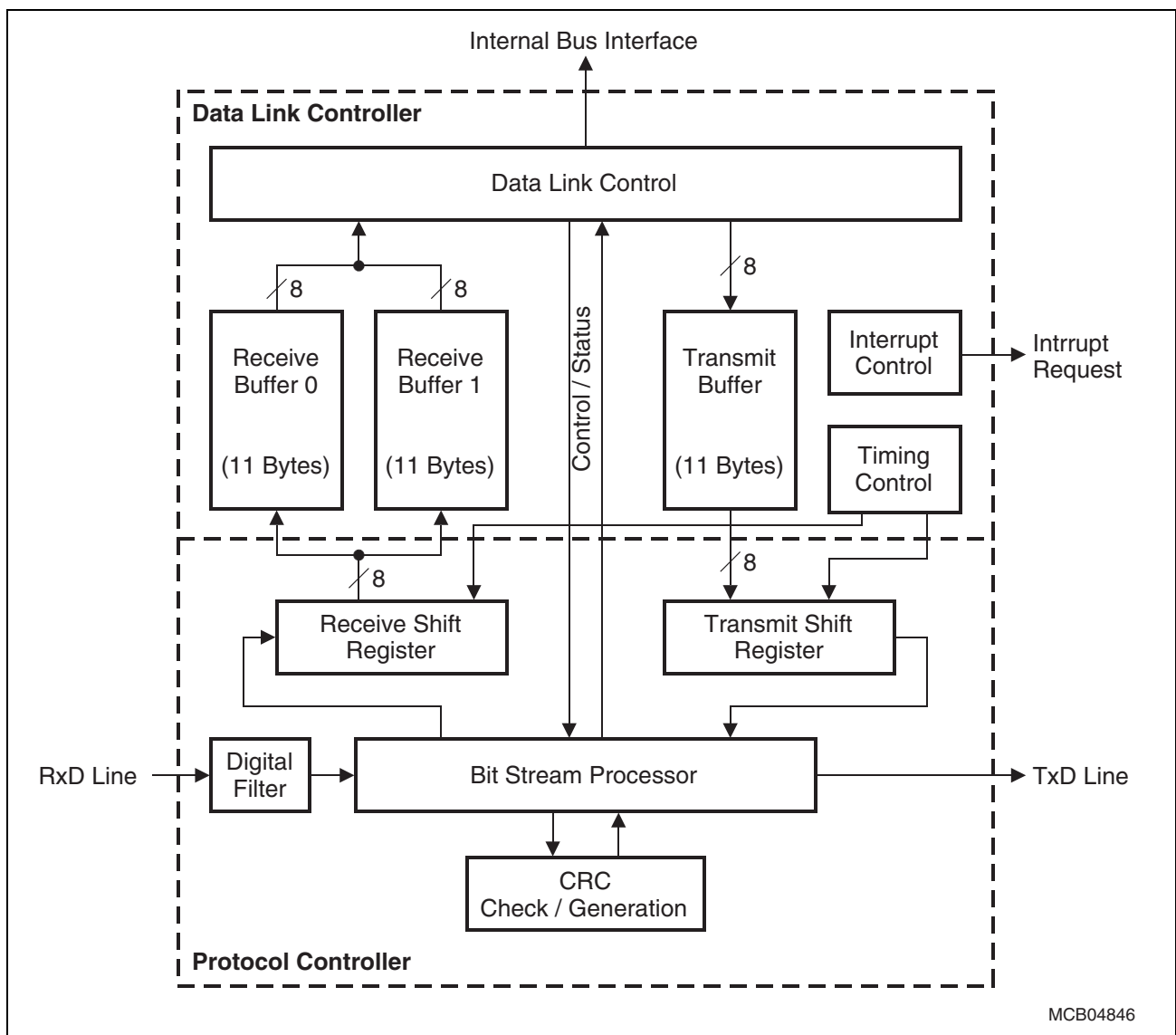
## 20.2 The Structure of the SDLM

The SDLM is built up by two basic blocks as shown in **Figure 20-4**:

- Protocol Controller
- Data Link Controller

**The Protocol Controller** basically contains the Bit Stream Processor and the two Shift Registers for the Transmit and the Receive path. The Bit Stream Processor encodes/decodes the Variable Pulse Width (VPW) data stream and translates incoming VPW symbols into data logic levels. The Protocol Controller further has 8-bit wide data interfaces to the Data Link Controller.

**The Data Link Controller** handles incoming and outgoing data using three 8-bit wide data buffers, the 11-byte transmit buffer and two 11-Byte receive buffers. The Data Link Controller also manages several control tasks (interrupt, timing, and buffer control).



**Figure 20-4 SDLM Block Diagram**

---

**The Serial Data Link Module (SDLM)****Configuration of the Data Link Controller**

The general configuration of the data link controller is accomplished by two registers, the Global Control Register and the Transceiver Delay Register. The bits within these two registers provide the following functions:

- SDLM enable/disable
- 4x Mode enable/disable
- Block Mode enable/disable
- Header type configuration (single or consolidated)
- Normalization bit polarity selection
- Receive buffer overwrite control
- Clock divider for J1850 bus rate depending on the external frequency
- Compensation of transceiver delay by SDLM
- CRCEN flag decides whether IFR type 3 should contain CRC checksum
- NB flag configures the polarity of NB symbol

**High Speed (4x) Mode**

- When high speed mode is used, all J1850 nodes should be configured to 4x mode when supported
- Those nodes which do not support 4x mode need to tolerate high speed operation (no error sign)
- Enable bit EN4X
- A Break symbol occurrence will generate an interrupt
- After Break occurrence CPU needs to reset RX/TX status flags

**Break Operation**

- Break allows bus communication to be terminated
- All nodes reset to an 'reset-to-receive' state (reset status bits by CPU)
- After Break symbol transition an IFS has to follow for resynchronization purpose
- When break is sent the current frame (if any) is ignored
- A break transmission can be generated by setting SBRK
- Break reception is indicated by bit BRK being set

*Note: After a hardware reset operation the SDLM module is disabled.*

## 20.3 Message Operating Mode

Two 11-Byte receive buffers and one 11-Byte transmit buffer are available for data transfers. This allows the transfer of a complete J1850 frame without buffer reloading. The data buffer access can be handled in two different modes:

**In Random Mode** all data bytes stored in the data buffers can be accessed directly via separate register addresses.

**In FIFO mode** the data stored in the data buffers are accessed sequentially as single bytes via one single register address.

In case of a loss of arbitration, an automatic retransmission is started, until the transmit request bit (TXRQ) is reset by the CPU. For correct transmission, the transmit buffer has to contain valid data (TXCPU >0) when TXRQ is set.

### 20.3.1 Receive Operation

The receive buffer structure contains two independent 11-Byte receive buffers. One of them can be directly accessed by the CPU (data and pointers). If this buffer is full (not yet completely read out), it can not be accessed by the J1850 module. Data reception over the bus is always done via the receive buffer on bus side. In order to release the receive buffer on CPU side, bit DONE has to be set. After complete reception of a frame, the buffer on J1850 side is declared full. If both buffers are full, the buffer on J1850 side can be overwritten by a new incoming frame or not, depending on the user-programmable overwrite enable bit (OVWR). In the case of the CPU buffer being empty and the J1850 buffer being full, both buffers are swapped. By this action, the full buffer can be accessed by the CPU and the empty one is available on J1850 side.

The total number of received bytes in the corresponding buffer is indicated by register RXPTR. Register RXCPU indicates how many bytes have already been read out. In FIFO mode (RXINCE = 1), CPU data read actions take place via register RXDATA, register RXCPU is automatically incremented by 1 after each read action. In Random Mode (RXINCE = 0), the buffer bytes can be directly accessed via their address. In this case, RXCPU is not incremented. In order to release a buffer for new message reception, the DONE bit has to be set. A receive interrupt is generated after complete reception of the whole frame (MSG REC = 1).

### Receive Control

Two bits control the receive operation of the SDLM:

**Bit RXINCE** (Receive Buffer Increment Enable) selects random mode ('0') or FIFO Mode ('1'). In random mode the CPU has access to each receive buffer byte via its own address. In FIFO mode, RXCPU pointer is incremented upon CPU read access until RXCPU == RXCNT (max. 11). This mode allows an easy CPU read transfer from the RXBuffer into a RAM location by reading one register only.



---

**The Serial Data Link Module (SDLM)**

**Bit DONE** (Receive Buffer on CPU Side Read Out Done) declares the receive buffer on CPU side to be empty, resets RXCPU and releases the buffer (reset of bit RBC). If there is a full receive buffer on the bus side, the buffers are swapped. This bit is reset after the buffer has been released.

**Receive Status Information**

Receive buffer(s) status information is provided in register BUFFSTAT:

**Bit RIP** (Reception in Progress) indicates whether the SDLM module is currently receiving a J1850 frame.

**Bit RBB** (Receive Buffer on Bus Side Full) indicates when the receive buffer on bus side is full (not set for a self-echo message).

**Bit RBC** (Receive Buffer on CPU Side Full) indicates when the receive buffer on CPU side is full (not set for a self-echo message).

**Bit MSGLST** (Message Lost) indicates when a received frame/byte has been discarded because the receive buffer is full (RBB = 1).

**Bit BREAK** (Break Received) is set when a break symbol has been received on the J1850 bus.

Receive operation status information is provided in register TRANSSTAT:

**Bit HEADER** (Header Received) indicates that a complete header has been received.

**Bit MSGREC** (Message Received) indicates when a complete frame have been received.

### 20.3.2 Transmit Operation

A data transmission is started by setting the transmission request bit TXRQ in register BUFFCON. Transmission is aborted by resetting bit TXRQ by software. FIFO mode and random mode are working in the same way as it is described for data reception. A transmit interrupt can be generated after a successful transmission of the complete frame (when bit MSGTRA is set).

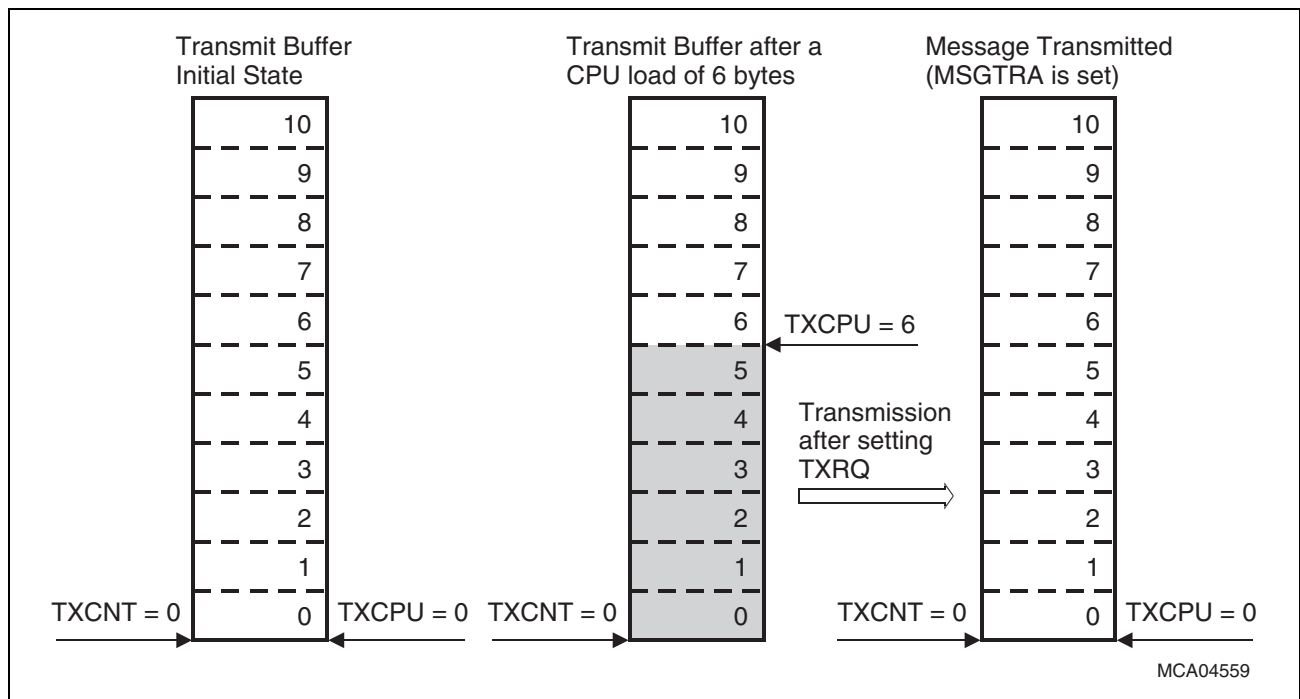


Figure 20-5 Transmit Buffer Operation

#### CPU Transmit Control

Three bits control the transmit operation of the SDLM:

**Bit TXINCE** (Transmit Buffer Increment Enable) enables FIFO Mode when set (random mode is always enabled).

**Bit TXRQ** (Transmit Request) initiates the transmission of a frame to the J1850 bus. In case of a lost arbitration, the module automatically retries transmission until the frame has been correctly sent out or the transmit request has been reset by software.

**Bit SBRK** initiates a break transmission when set by the CPU.

#### Transmit Status Information

Transmit status information is provided in register TRANSSTAT:

**Bit TIP** (Transmission in Progress) indicates that a message is currently transmitted.

**Bit MSGTRA** (Message Transmitted) indicates a successful frame transmission.

**Bit ARL** (Arbitration Lost) indicates that arbitration has been lost.

### **20.3.3 In-Frame Response (IFR) Operation**

The SDLM supports automatic IFR transmission for type 1, 2 IFR for three byte consolidated headers (no CPU load required). If the IFRs are handled via the transmit buffer, TXCPU indicates the number of bytes to be transmitted.

- If automatic IFR transmission function is not possible (single byte or one byte consolidated headers): If bit IFREN is not set, type 1 and 2 are handled via SDLTXD, too. If IFREN is set, the value stored in IFRVAL will be transmitted if bit TXIRF is set.
- In case of automatic IFR transmission, register IFRVAL delivers the source ID. The value has to be written by the CPU first.
- IFR type 3 transmission can only be handled over SDLTXD.
- Setting bit TXIFR initiates an IFR transmission (if automatic IFR not possible).
- Normalization symbol can be configured over NB configuration bit.
- If IFR with CRC (Type 3, CRCEN = 1) is used, the CRCERR indicates CRC error conditions.
- If register IFRVAL is used for transmission, no CRC will be sent out (not depending on CRCEN). If SDLTXD is used, CRC will be sent out if bit CRCEN is set.
- HEADER bit indicates complete reception of header byte(s) in the receive buffer on bus side.

Transmission of type 1 and type 2 IFRs for single byte headers and one byte consolidated headers is also accomplished by bit TXIFR, which has to be set by software. In case of automatic IFR (for type 1, 2 for three byte consolidated headers and IRFEN = 1) bit TXIFR is not needed.

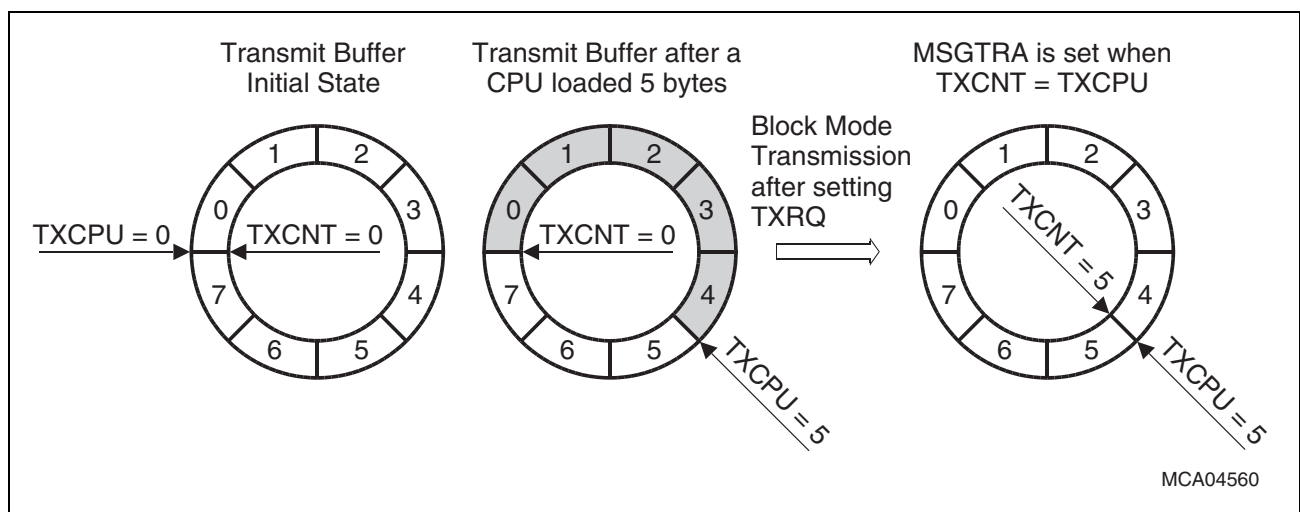
3-byte consolidated headers: If IFREN is set, automatic response to Type 1 and Type 2 IFRs via the IFRVAL register is enabled. Type 3 IFR is sent by writing the IFR to the transmit buffer and then setting the TXIFR bit of the SDLBUFCON register. If IFREN is not set, all IFRs are transmitted via SDLTXD upon setting of TXIFR.

Single byte headers and one byte consolidated headers: As there is no information in the header to indicate if an IFR is required or not, automatic transmission of Type 1 and 2 IFRs is not possible. If IFRs are used in the system, the header interrupt should be enabled in order to give time to decode the header through software to determine if an IFR is required. IFR transmission is always initiated by setting bit TXIFR. Type 3 IFR is always done via SDLTXD, whereas types 1, 2 are handled via SDLTXD (IFREN = 0) or register IFRVAL (IFREN = 1).

### 20.3.4 Block Mode

In block mode, the SDLM supports the transfer of J1850 frames with unlimited length (application specific). Block mode is selected by setting bit BMEN. In block mode, a 16-Byte receive buffer (no swapping) and an 8-Byte transmit buffer are available. Both buffers operate in FIFO mode, independent of bits RXINCE and TXINCE. Caused by the FIFO structure of the buffers in block mode, data bytes are written to the transmit buffer always via bitfield TXDATA0 (TXD00[7:0]) and data bytes are read from the receive buffer via bitfield RXDATA00 (TXD00[7:0]).

When a byte has been transmitted a transmit interrupt can be generated when  $TXCPU = TXCNT$ . After the reception of a byte a receive interrupt can be generated if  $RXCNT0 = RXCPU0$ .



**Figure 20-6 Transmit Operation in Block Mode**

*Note: Block transmission is terminated if  $TxCPU == TxCNT$  after  $TxCNT$  has been incremented due to the transmission of a byte and the last byte in the transmit register is sent out correctly. In order to avoid termination software must take care of the described condition.*

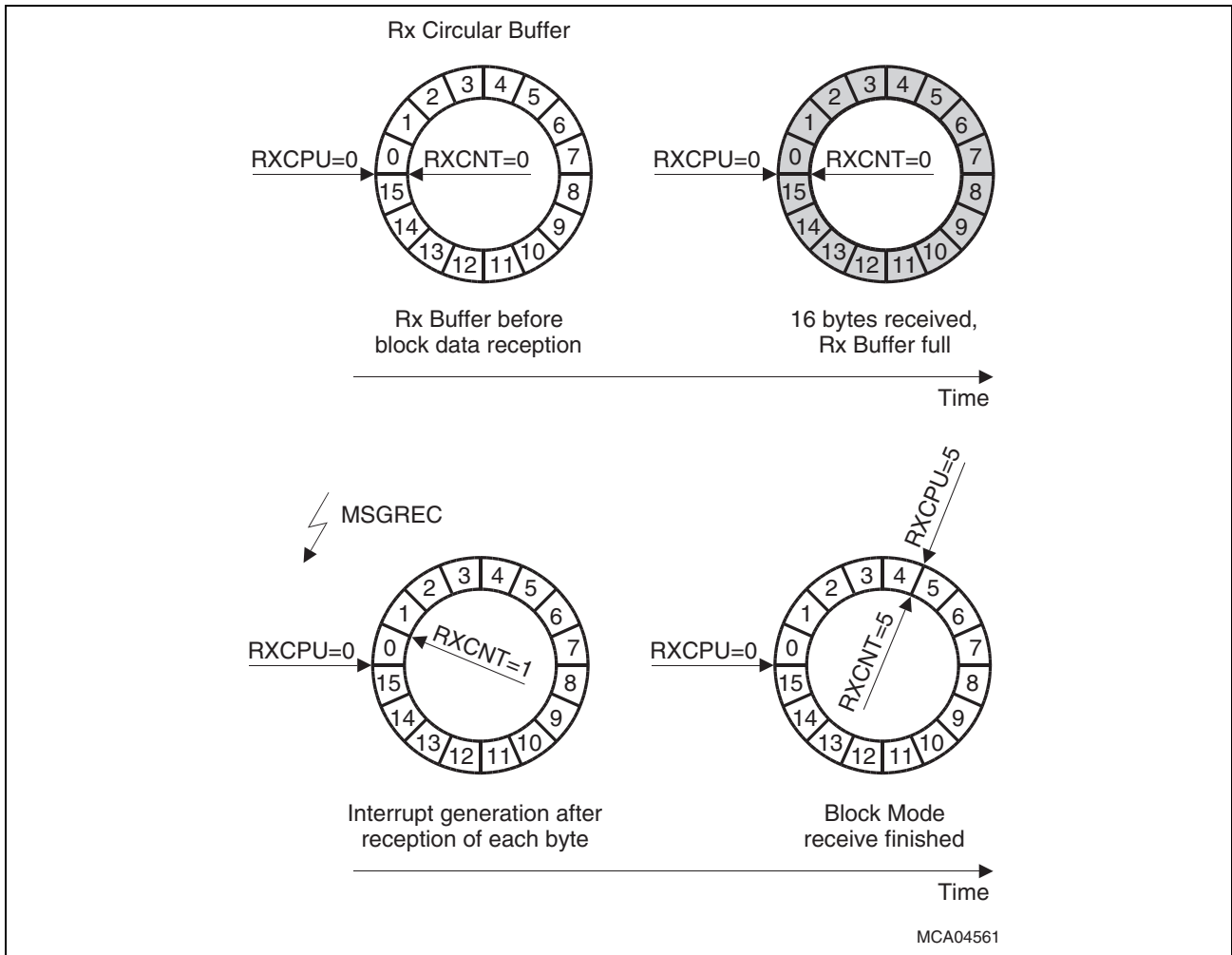
In order to monitor the status of the bus during transmission, the SDLM always reads on the bus, even while transmitting. As a result, the user can check whether the message to be sent is equal to the message on the bus (test for arbitration).

The receive buffer in block mode (FIFO mode) is accessed via register RXD00. The elements of the FIFO can always be accessed via their addresses, too. During operation in block mode the pointers are not reset.

The receive buffer in block mode is composed of data bytes RXDATA00 ... RXDATA07 (1<sup>st</sup> half) and RXDATA10 ... RXDATA17 (2<sup>nd</sup> half).

TxCNT is incremented each time a byte is loaded from the transmit buffer to the output register.

The Serial Data Link Module (SDLM)



**Figure 20-7 Receive Operation in Block Mode**

In block mode, the interrupt request flags MSGREC (reception) and MSGTRA (transmission) are automatically reset by hardware upon a read action from RXDATA, or a write action to TXDATA respectively. RBB(= RBC) is set upon a pointer match after reception of a byte (receive buffer full) and reset by a read action from this buffer. MSGLST is set if RBB has been set before and a new data byte is received. MSGLST is not automatically reset by hardware. All error flags remain pending (once set) and have to be cleared by software. In case of a detected error during transmission, the transmit request bit TxRQ is reset by hardware in order to abort the transmission (no automatic retry).

*Note: In FIFO mode (block mode or normal mode), the FIFOs are byte-oriented. In order to avoid mismatch in case of word accesses, the LSB of the pointer on CPU side selects which byte (lowbyte or highbyte of the word) is taken into account. If the pointer's LSB is 0, any access to the corresponding buffer is based on the lowbyte (read: highbyte = 0, lowbyte = FIFO value, write: only lowbyte written). If the pointer's LSB is 1, any access to the corresponding buffer is based on the highbyte (read: highbyte = FIFO value, lowbyte = 0, write: only highbyte written). In any case, the pointer is incremented by 1.*

The Serial Data Link Module (SDLM)

20.4 Flowcharts

The following flowcharts illustrate the operation of the SDLM. Each flowchart shows in a symbolic way how to handle the respective operation.

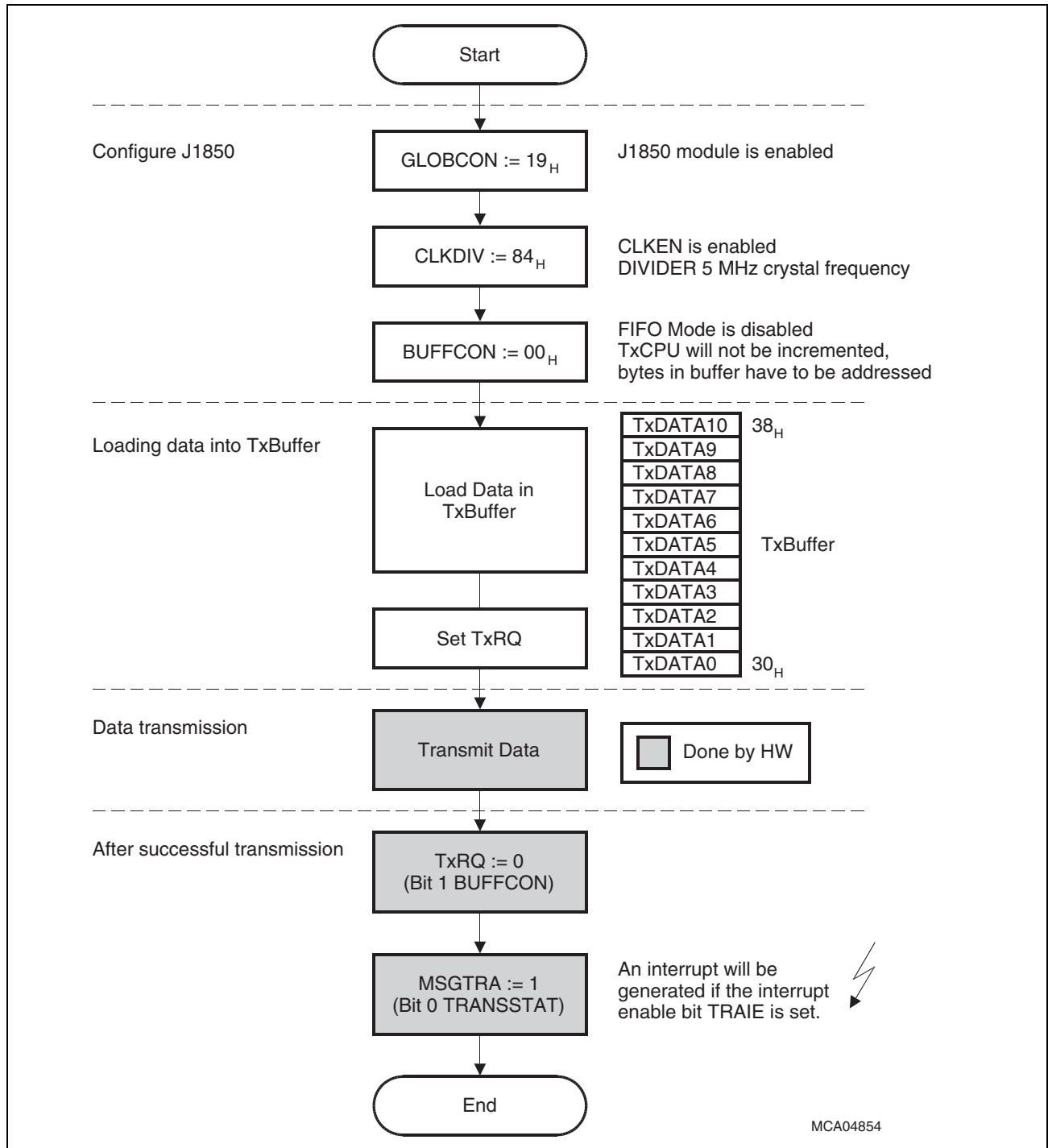
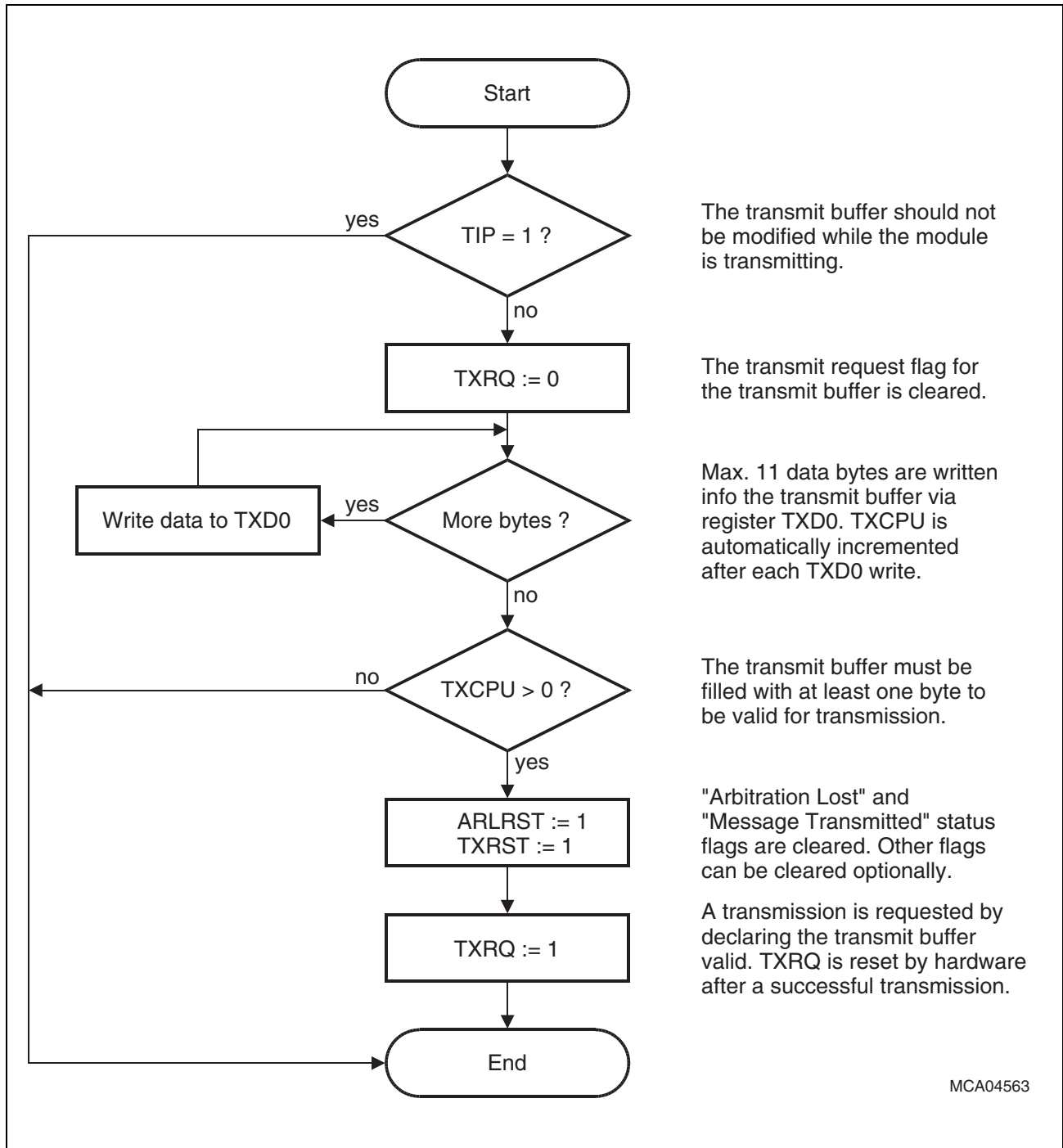


Figure 20-8 Initialization, Data Setup, and Transmission (Random Mode)

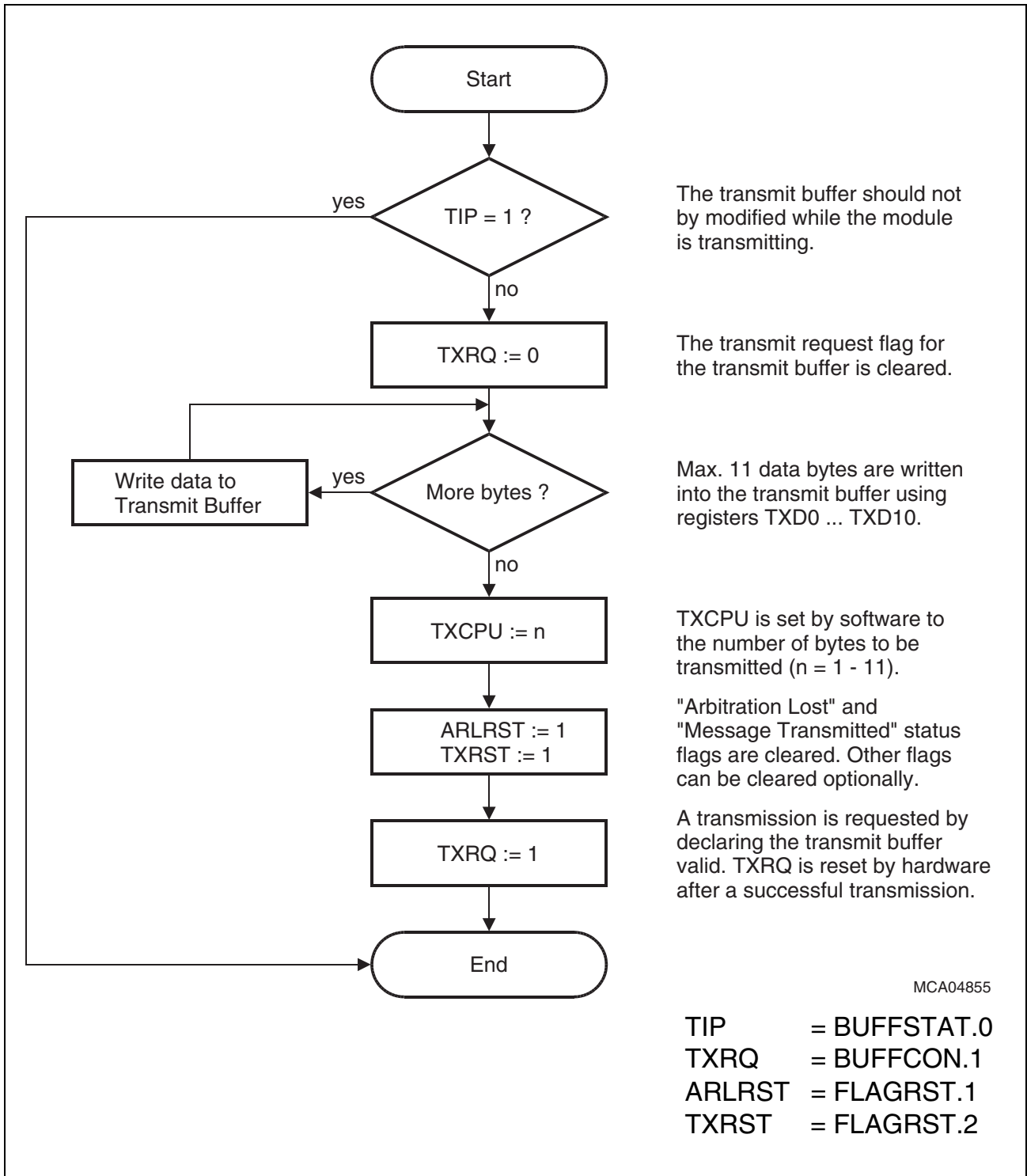
Note: In order to adapt the timing to the transceiver device, register TXDELAY has to be configured, too.



**Figure 20-9 Standard Message Transmission in FIFO Mode**

*Note: Bit TXINCE in register BUFFCON has to be set in order to provide FIFO functionality. The transmit buffer is filled by multiple write actions to TXDATA0. Register TXCPU is incremented after each write operation to TXDATA0. All other registers of the transmit buffer can always be directly accessed via their addresses without changing TXCPU.*

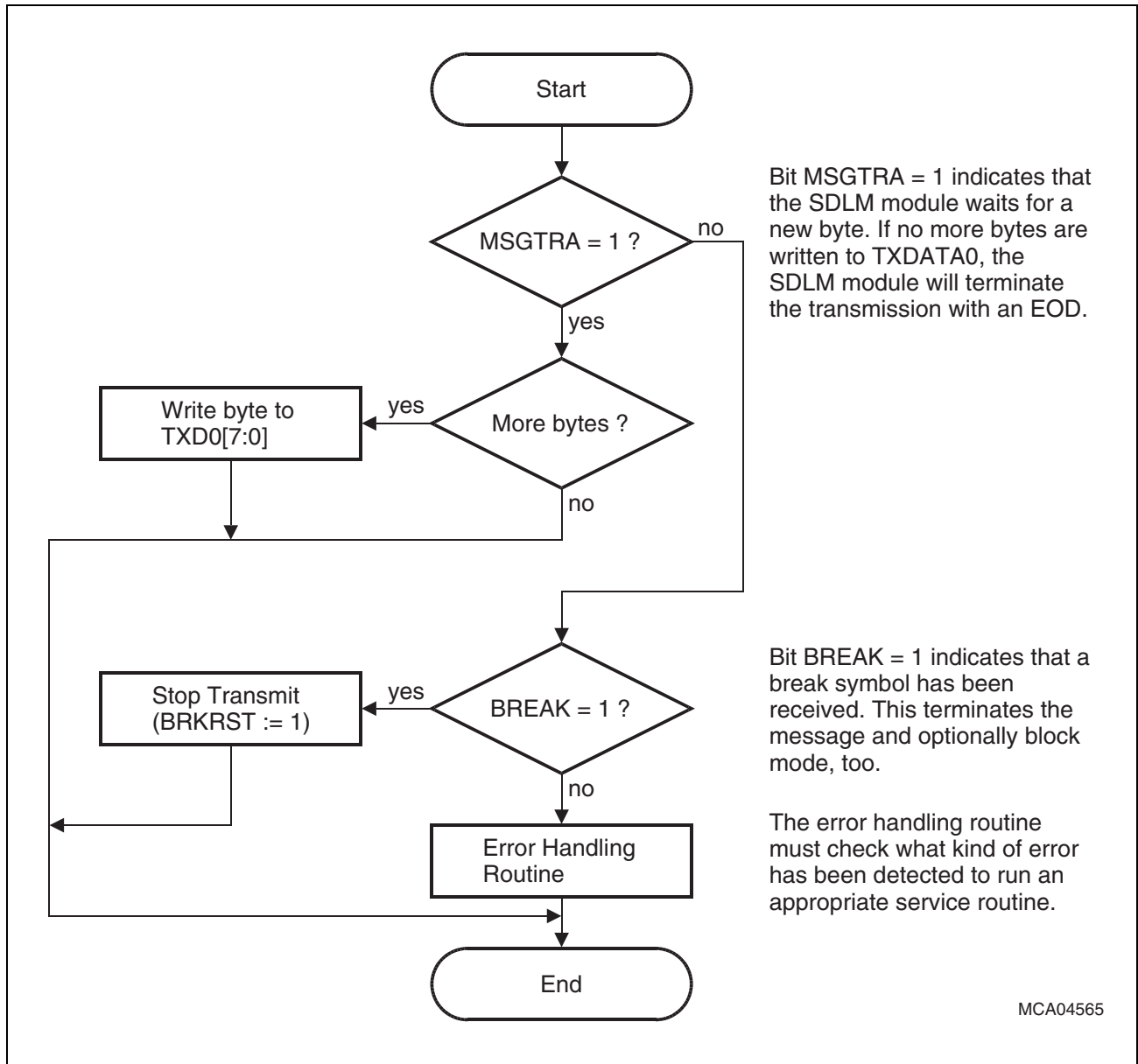
**The Serial Data Link Module (SDLM)**



**Figure 20-10 Standard Message Transmission in Random Mode**

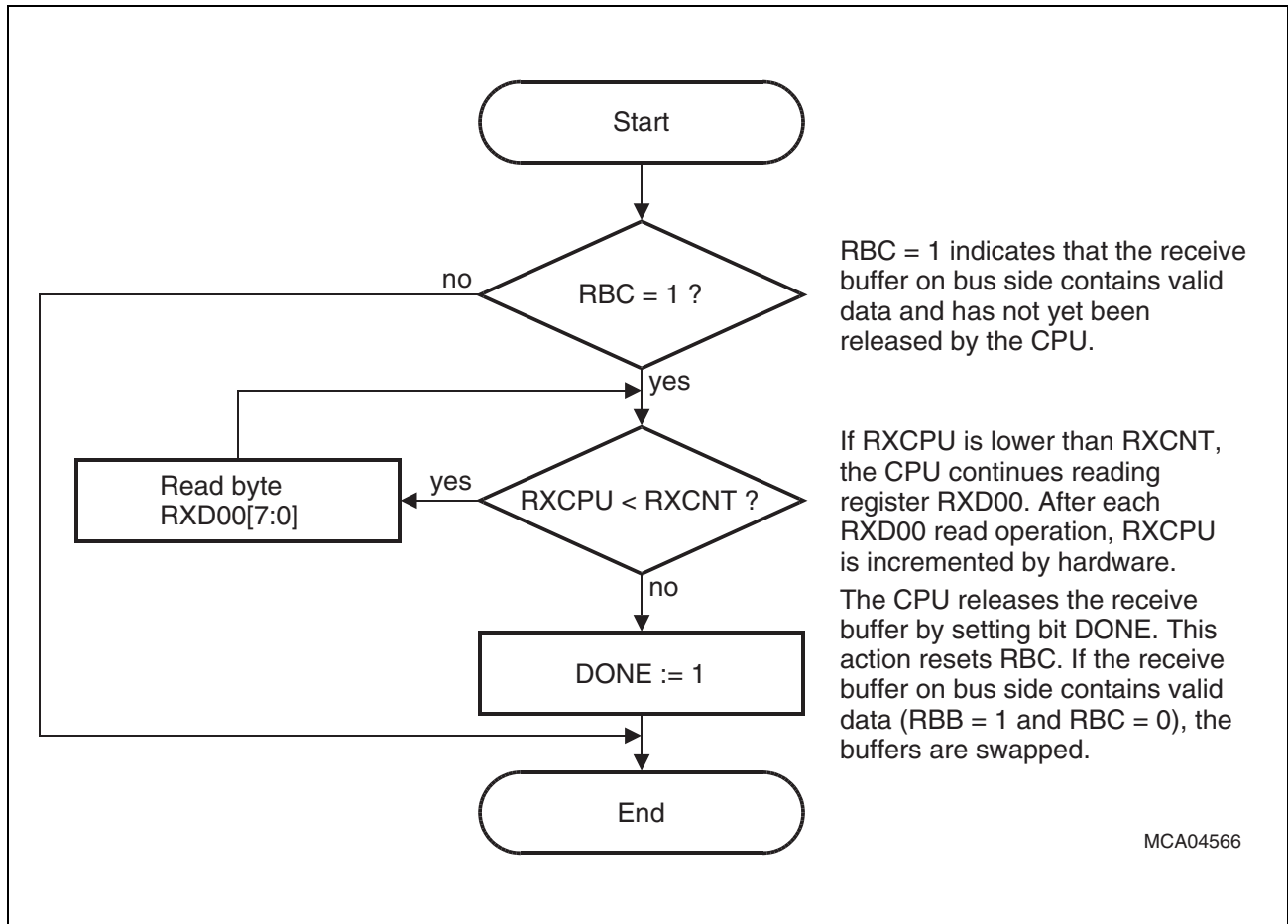


The Serial Data Link Module (SDLM)



**Figure 20-11 Transmission in Block Mode**

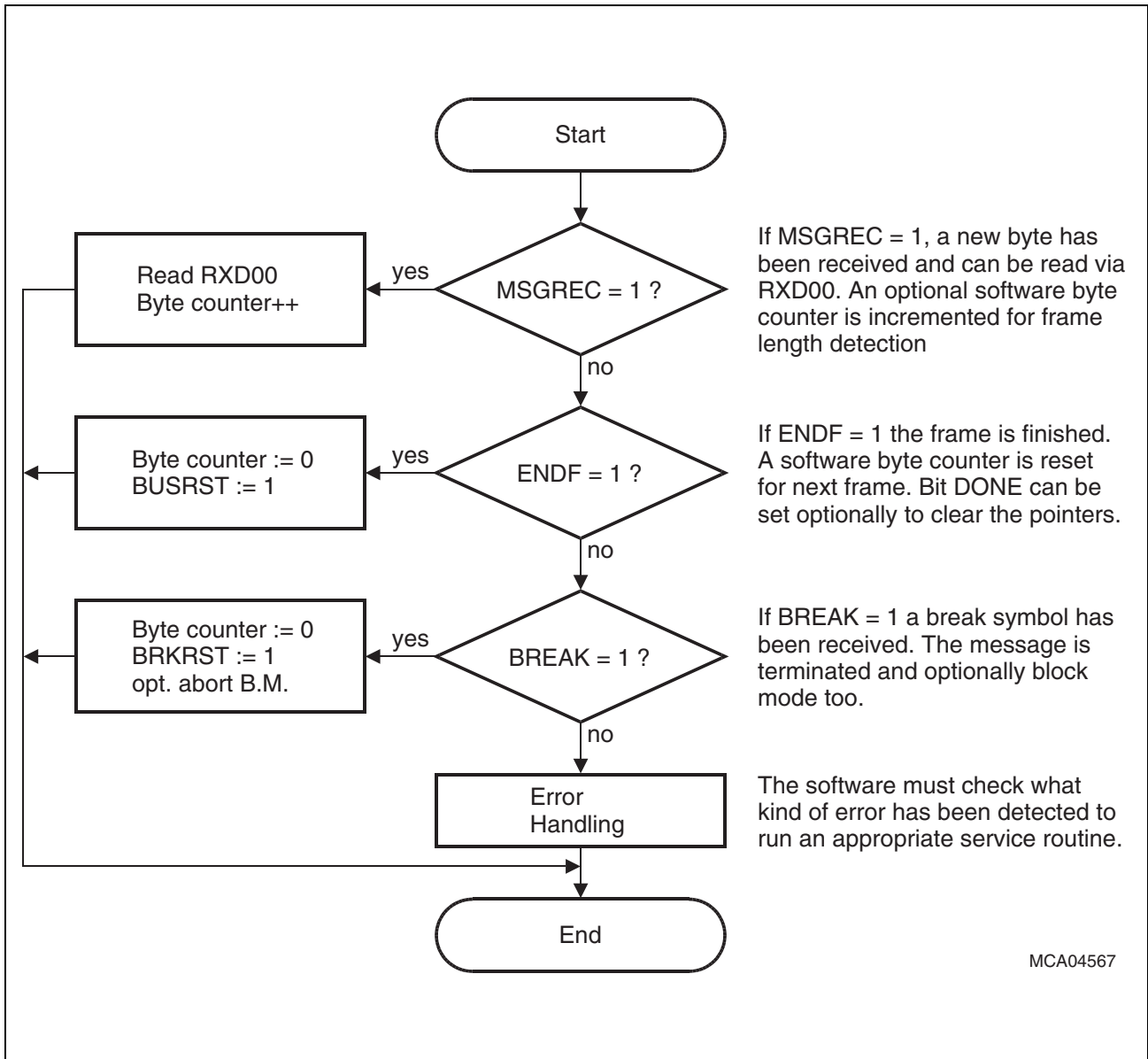
*Note: Block mode is selected by setting bit BMEN in register GLOBCON. In block mode, automatically always FIFO mode is selected (independent of bit TXINCE). The transmit buffer in Block Mode is 8 bytes long.*



**Figure 20-12 Read Operation in FIFO Mode**

*Note: If bit RXINCE in register BUFCON is set FIFO Mode is enabled. Register RXCPU is incremented after each read operation from register RXD00. The receive buffer is read out by the CPU using multiple read actions from RXD00. All other registers of the receive buffer can always be directly accessed via their addresses without changing RXCPU.*

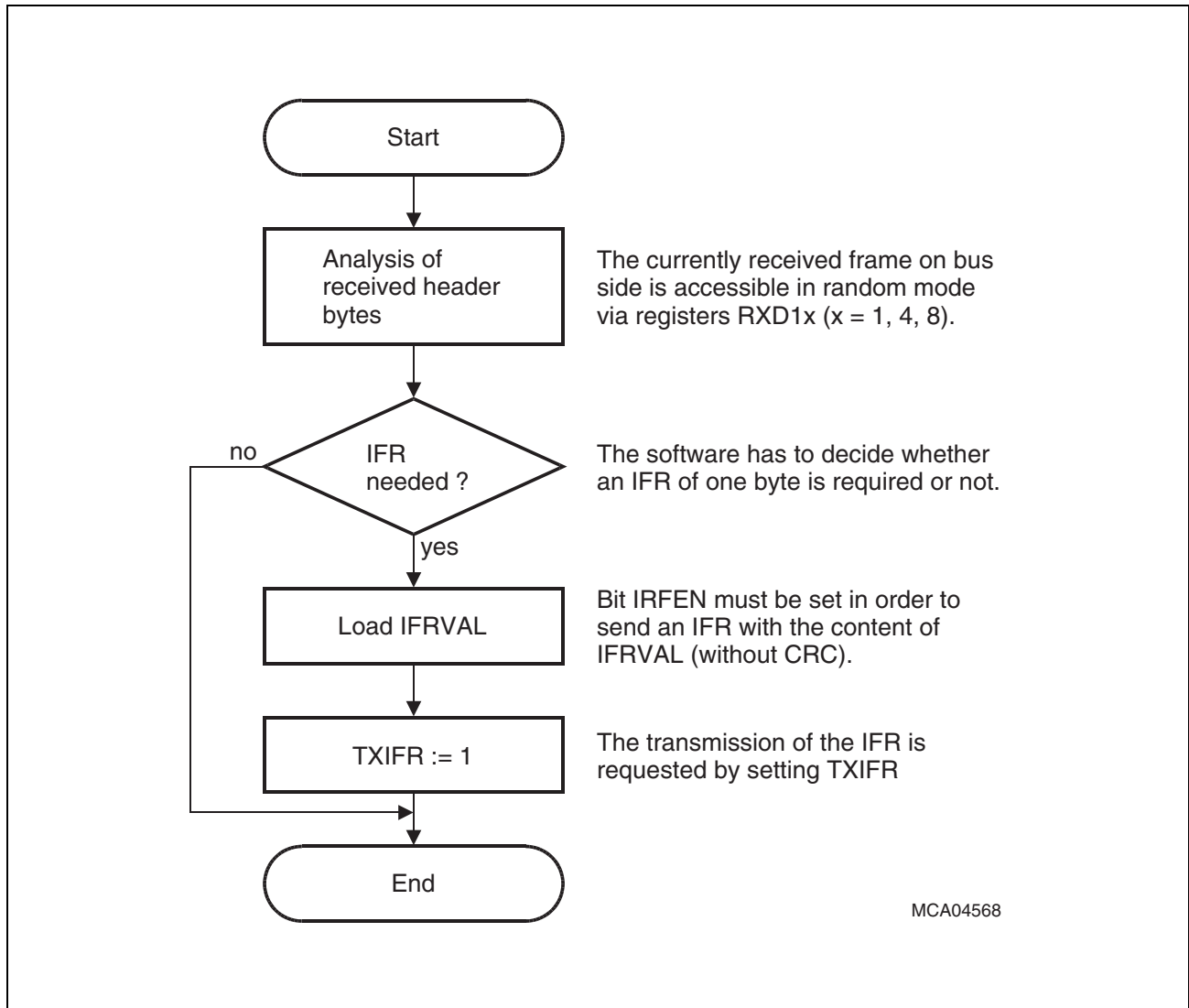
The Serial Data Link Module (SDLM)



**Figure 20-13 Reception in Block Mode**

*Note: Block Mode is selected by setting bit BMEN in register GLOBCON. In block mode automatically always FIFO mode is selected (independent of bit RXINCE). The receive buffer in block mode is 16 bytes long.*

The value of the RxCNT pointer is automatically copied to register SOFPTR (start of frame pointer) to allow the user to detect the begin of a new frame.



**Figure 20-14 IFR Handling via IFRVAL**

*Note: Bit HEADER is automatically set by hardware after reception of the complete header (1 or 3 bytes) in the receive buffer on bus side. This buffer can be accessed at consecutive addresses starting at 50<sub>H</sub>. In case of 3-Byte consolidated headers with the k bit set, an IFR (via IFRVAL) will be automatically generated if bit IFREN is set. Bit HEADER is reset when RXRST has been set by software or after the reception of the complete frame. In case of single byte headers or 1-Byte consolidated headers, the flowchart shows a possibility to send IFR. If an IFR is requested (automatically or by hardware), the IFR byte(s) are sent after the EOD symbol. In case of type 2 IFR, automatic retry after arbitration loss takes place depending on bit ARIFR.*

## 20.5 Interrupt Handling

The SDLM has one interrupt output, which is connected (through a synchronization stage) to a standard interrupt node in the C161CS/JC/JI in the same manner as all other interrupts of the standard on-chip peripherals. With this configuration, the user has all control options available for this interrupt, such as enabling/disabling, level and group priority, and interrupt or PEC service (see note below). The SDLM is connected to an XBUS interrupt control register.

As for all other interrupts, the node interrupt request flag is cleared automatically by hardware when this interrupt is serviced (either by standard interrupt or PEC service).

*Note: As a rule, SDLM interrupt requests can be serviced by a PEC channel. However, because PEC channels only can execute single predefined data transfers (there are no conditional PEC transfers), PEC service can only be used, if the respective request is known to be generated by one specific source, and that no other interrupt request will be generated in between. In practice this seems to be a rare case.*

Since an interrupt request of the SDLM can be generated due to different conditions, the appropriate interrupt status registers must be read in the service routine to determine the cause of the interrupt request.

The bit addressable interrupt control register XP7IC is assigned to the SDLM.

The SDLM can generate the interrupts on the following events:

- Data receive/transmit interrupt conditions
  - Message transmitted
  - Message received
  - Header received
  
- Protocol related interrupt conditions
  - End of frame detected
  - Break received
  - Arbitration lost
  - CRC error detected
  - Error detected

The Serial Data Link Module (SDLM)

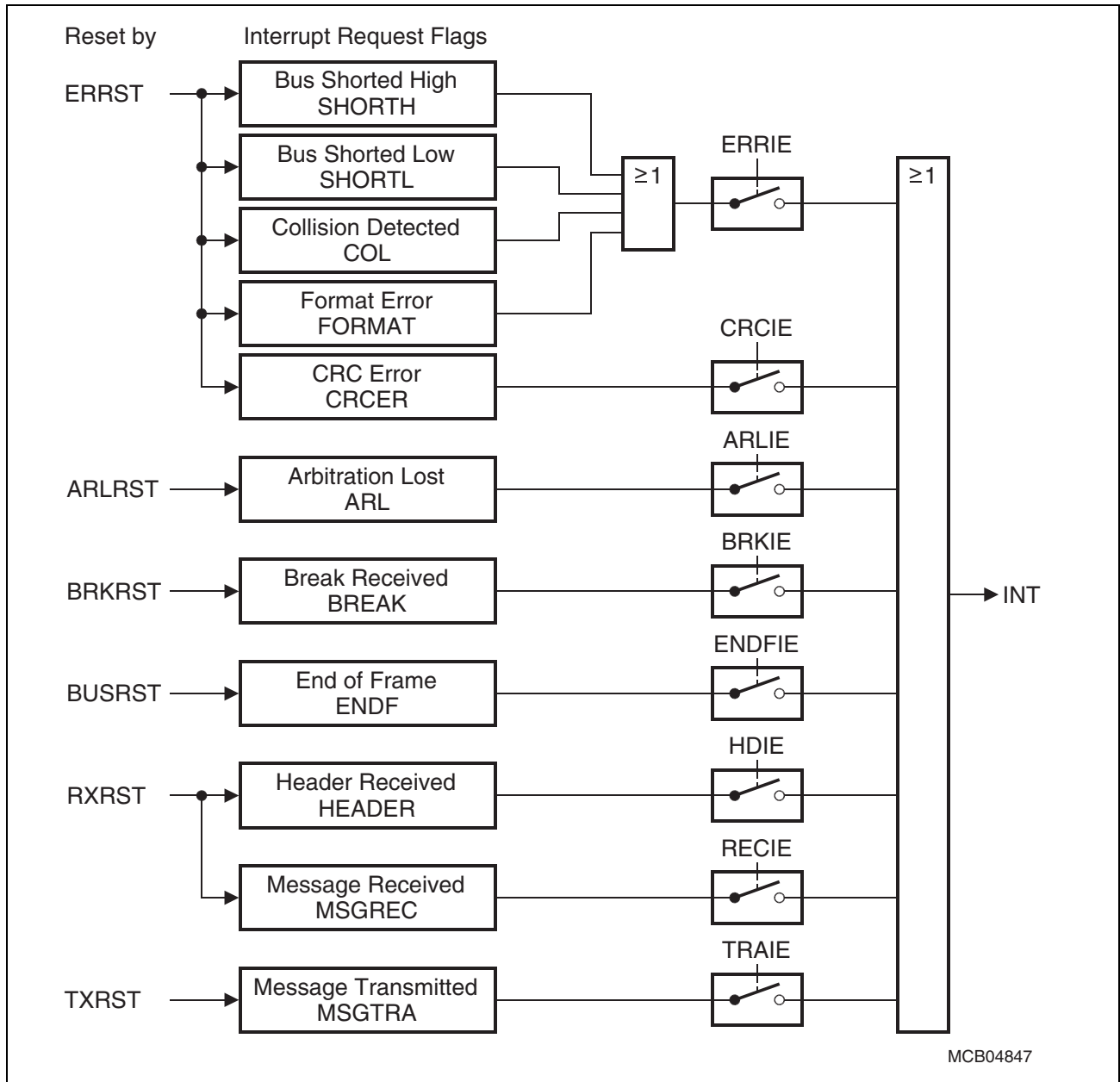


Figure 20-15 Interrupt Structure of SDLM

## 20.6 Port Control

The receive data line and the transmit data line of the SDLM are alternate port functions. The respective port driver for the receive line will automatically be switched OFF, the respective port driver for the transmit line will automatically be switched ON.

This provides a standard pin configuration without additional software control and also works in emulation mode where the port direction registers cannot be controlled.

The receive and transmit line can be assigned to several port pins of the C161CS/JC/JI under software control. This assignment is selected via bitfield IPC (Interface Port Connection) in register IPCR.

**Table 20-2 Assignment of SDLM Interface Lines to Port Pins**

IPC	SDL_RXD	SDL_TXD	Notes
000	P4.4	P4.7	Compatible with previous derivatives where the assignment of SDLM interface lines was fixed.
001	P4.6	P4.7	Pins P4.5-0 available for segment address lines A21 ... A16 (4 MByte external address space).
010	P7.5	P7.4	Port 4 available for segment address lines A23 ... A16 (16 MByte external address space).
011	P7.7	P7.6	Port 4 available for segment address lines A23 ... A16 (16 MByte external address space).
100	–	–	<i>Reserved.</i> Do not use this combination.
101	–	–	<i>Reserved.</i> Do not use this combination.
110	–	–	<i>Reserved.</i> Do not use this combination.
111	Idle (recessive)	Disconnected	No port assigned. Default after Reset.

---

## The Serial Data Link Module (SDLM)

The location of the SDLM interface lines can now be selected via software according to the requirements of an application:

**Compatible Assignment** (IPC = 000<sub>B</sub>) makes the C161CS/JC/JI suitable for applications with a given hardware (board layout). The SDLM interface lines are connected to the port pins to which they are hardwired in previous derivatives.

**Wide Address Assignment** (IPC = 001<sub>B</sub>) uses the two upper pins of Port 4, leaving room for six segment address lines (A21 ... A16). A contiguous external address space of 4 MByte is available in this case.

**Full Address Assignment** (IPC = 010<sub>B</sub> or 011<sub>B</sub>) removes the SDLM interface lines completely from Port 4. The maximum external address space of 16 MByte is available in this case.

The SDLM interface lines are mapped to Port 7. Two pairs of Port 7 pins can be selected.

**No Assignment** (IPC = 111<sub>B</sub>) disconnects the SDLM interface lines from the port logic. This avoids undesired currents through the interface pin drivers while the C161CS/JC/JI is in a power saving state.

After reset the SDLM interface lines are disconnected.

*Note: Assigning SDLM interface signals to a port pin overrides the other alternate function of the respective pin (segment address on Port 4, CAPCOM lines on Port 7).*



## 20.7 SDLM Register Description

This section summarizes and describes all registers which are provided in order to operate the SDLM.

The Interface Port Connect Register IPCR assigns the receive data line and the transmit data line to port pins of the C161CS/JC/JI.

### IPCR

Interface Port Connect Reg.                      XReg (EB04<sub>H</sub>)                      Reset Value: 0007<sub>H</sub>

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

Field	Bits	Type	Description
IPC	[2:0]	rw	<p><b>Interface Port Connection</b>  Assigns port pins to the SDLM's transmit and receive lines. The encoding of bitfield IPC is described in <a href="#">Section 20.6</a>.  Reset value = 111<sub>B</sub>, i.e. no port connection.</p>

The Serial Data Link Module (SDLM)

**Global Configuration Registers**

The Global Control Register contains bits to select different transfer modes and to determine the message handling.

**GLOBCON**

**Global Control Register**                      **XReg (EB10<sub>H</sub>)**                      **Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	AR IFR	OV WR	NB	HDT	BM EN	EN 4X	GM EN
-	-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
<b>GMEN</b>	0	rw	<b>Global Module Enable</b> 0: The complete SDLM module is disabled. 1: The SDLM module is enabled and data transmission via the serial bus is possible. Resetting GMEN by software (from '1' to '0') resets the module, except for the timings.
<b>EN4X</b>	1	rw	<b>High Speed Transfer Enable (4x)</b> 0: The data transfer rate is 10.4 kbit/s. 1: The data transfer rate is 41.6 kbit/s.
<b>BMEN</b>	2	rw	<b>Block Mode Enable</b> 0: The maximum frame length is 11 data bytes (normal mode). 1: Transfers of frames longer than 11 data bytes are enabled. The transmit and receive buffers are organized as circular buffers, which can be accessed in FIFO mode. Resetting BMEN resets the buffer pointers.
<b>HDT</b>	3	rw	<b>Header Type</b> 0: Single byte headers are supported. 1: Consolidated headers (1 byte and 3 bytes) are supported.
<b>NB</b>	4	rw	<b>Normalization Bit Polarity</b> 0: Normalization bit polarity is 0. 1: Normalization bit polarity is 1.

**The Serial Data Link Module (SDLM)**

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>OVWR</b>	5	rw	<p><b>Overwrite Enable</b></p> <p>0: Overwrite action of the receive buffer on SDLM side in case of an incoming frame and a full receive buffer on bus side (RBB = 1) disabled. The frame on the bus is not accepted and is lost.</p> <p>1: The full buffer on bus side is declared empty and then overwritten by the next incoming frame. The frame in the receive buffer on bus side is lost.</p>
<b>ARIFR</b>	6	rw	<p><b>Automatic Retry of IFR</b></p> <p>0: The module will not retry transmission of IFR byte(s) in case of an arbitration loss (for IFR types 1, 3). The collision detection mechanism is generally enabled during IFR.</p> <p>1: An automatic retry of IFR transmission in case of arbitration loss is enabled (for IFR type 2). The collision detection mechanism is disabled during EOD.</p>

**Note: In order to support transmission in 4x mode, the transceiver delay should not exceed 4  $\mu$ s.**

**The Serial Data Link Module (SDLM)**

The Clock Divider Register CLKDIV adapts the CPU clock fed to the SDLM to the internal module clock which controls the actual bit timing. The module clock timing can be adapted to regular crystals as well as to baudrate crystals.

**CLKDIV**

**Clock Divider Register**

**XReg (EB14<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	CLK EN	CLK SEL						
-	-	-	-	-	-	-	-	rw	rw						

Field	Bits	Type	Description
<b>CD</b>	[5:0]	rw	<b>Clock Divider</b> Defines the prescaler value (1 ... 64) which generates the internal module clock from the CPU clock signal. $f_{MOD} = f_{CPU} / (CD + 1)$ . 00 <sub>H</sub> : $f_{MOD} = f_{CPU} / 1$ ... 3F <sub>H</sub> : $f_{MOD} = f_{CPU} / 64$
<b>CLKSEL</b>	6	rw	<b>Clock Select</b> Selects the internal module clock which is used to generate the bit timing. 0: 1.00 MHz module clock (used for standard crystals) 1: 1.05 MHz module clock (used for baudrate crystals)
<b>CLKEN</b> <sup>1)</sup>	7	rw	<b>Clock Enable</b> 0: Module clock is gated off. 1: Module is clocked, protocol layer working.

<sup>1)</sup> Only registers GLOBCON, CLKDIV and IPCR can be accessed if CLKEN = 0.

**The Serial Data Link Module (SDLM)**

The Transceiver Delay Register TxDELAY allows for the compensation of the transceiver delay caused by the external bus transceiver and the bus lines. This ensures the correct arbitration of each bit transferred over the J1850 bus.

**TxDelay**

**Transceiver Delay Register      XReg (EB16<sub>H</sub>)      Reset Value: 0014<sub>H</sub>**

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

Field	Bits	Type	Description
<b>TD</b>	[5:0]	rw	<b>Transceiver Delay</b> Defines the transceiver delay, which is taken into account by the J1850 bitstream processor. TD defines the number of module clock cycles. The reset value equals 20 μs @ 1.00 MHz or 19 μs @ 1.05 MHz.
<b>RINV</b>	6	rw	<b>Invert Receive Input</b> 0: Receive input polarity is not inverted. 1: Receive input polarity is inverted.

The In-Frame Response Value Register IFR stores the byte which can be sent out as source ID in case of a one-byte IFR (automatic transmission or triggered by SW).

**IFR**

**In-Frame Response Value Reg.      XReg (EB18<sub>H</sub>)      Reset Value: 0000<sub>H</sub>**

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

Field	Bits	Type	Description
<b>IFRVAL</b>	[7:0]	rw	<b>In-Frame Response Value</b> Bitfield IFRVAL contains the value to be transmitted in case of requested in-frame response (type 1, 2 IFR, 1 byte response). In-Frame response via register IFR must be enabled by bit IFREN and initialized by the CPU.

The Serial Data Link Module (SDLM)

Control and Status Registers

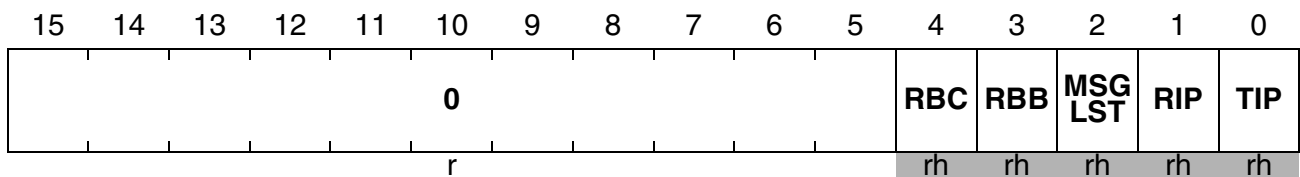
The Buffer Status Register BUFFSTAT contains the buffer-related status flags.

**BUFFSTAT**

Buffer Status Register

XReg (EB1C<sub>H</sub>)

Reset Value: 0000<sub>H</sub>



Field	Bits	Type	Description
<b>TIP</b>	0	rh	<p><b>Transmission In Progress</b></p> <p>0: The SDLM does not currently send a frame.</p> <p>1: The SDLM is transmitting data (transmit buffer or IFRVAL) without having lost arbitration.</p> <p>TIP is cleared by hardware when the arbitration is lost or EOD or ENDF are detected.</p>
<b>RIP</b>	1	rh	<p><b>Reception in Progress</b></p> <p>0: The SDLM does not currently receive a frame.</p> <p>1: The SDLM is receiving a frame from the bus.</p> <p>RIP is cleared by hardware when ENDF is detected.</p>
<b>MSGLST</b>	2	rh	<p><b>Message Lost</b></p> <p>0: All frames received correctly.</p> <p>1: A received frame/byte has been discarded because the receive buffer is full (RBB = 1).</p> <p>If overwrite is enabled, the receive buffer will be overwritten, otherwise the received frame (or bytes in block mode) are discarded.</p> <p>MSGLST is cleared when MSGREC is reset (in normal mode) or by software (in block mode).</p>
<b>RBB</b>	3	rh	<p><b>Receive Buffer on Bus Side Full</b></p> <p>0: The receive buffer on J1850 side has room.</p> <p>1: The receive buffer on J1850 side is full.</p> <p>RBB is cleared by hardware when:</p> <ul style="list-style-type: none"> <li>– the buffer is swapped to CPU side,</li> <li>– the CPU reads bytes from the buffer in block mode,</li> <li>– a new message is received with overwrite enabled.</li> </ul>

The Serial Data Link Module (SDLM)

Field	Bits	Type	Description
RBC	4	rh	<b>Receive Buffer on CPU Side Full</b> 0: The receive buffer on CPU side has room. 1: The receive buffer on CPU side is full. This buffer remains allocated by the CPU and bit RBC remains set until bit DONE has been set by software.

The Serial Data Link Module (SDLM)

The Transmission Status Register TRANSSTAT contains transmission-related status flags and monitors three functional bits of the header of the currently received frame.

**TRANSSTAT**

Transmission Status Register XReg (EB1E<sub>H</sub>) Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	Y	K	H	ARL	BR EAK	HEA DER	MSG REC	MSG TRA
-	-	-	-	-	-	-	-	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
<b>MSGTRA</b>	0	rh	<p><b>Message Transmitted</b></p> <p>0: Message transmission not complete. 1: Message transmitted (details below). <u>Normal mode:</u> TxBuffer or IFRVAL completely transmitted, i.e. arbitration won and EOD detected. Cleared via bit TxRST. <u>Block Mode:</u> Pointer match after transmission of a byte or ENDF detection. Cleared when the CPU writes to the transmit buffer or via bit TxRST.</p>
<b>MSGREC</b>	1	rh	<p><b>Message Received</b></p> <p>0: Message reception not complete. 1: Message received (details below). <u>Normal Mode:</u> New frame in the receive buffer on bus side (detection of ENDF). Cleared via bit RxRST or by hardware if the buffer is overwritten. <u>Block Mode:</u> Pointer mismatch after reception of a byte (FIFO not empty) or ENDF detection. Cleared when the CPU reads from the receive buffer or via bit RxRST. <i>Note: MSGREC will not be set upon the reception of a self-echo message.</i></p>
<b>HEADER</b>	2	rh	<p><b>Header Received</b></p> <p>0: Header not complete. 1: Complete header received. Cleared by hardware after reception of the complete frame.</p>



**The Serial Data Link Module (SDLM)**

<b>Field</b>	<b>Bits</b>	<b>Type</b>	<b>Description</b>
<b>BREAK</b>	3	rh	<b>Break Received</b> 0: No break symbol received. 1: A break symbol was received on the J1850 bus. BREAK must be cleared by software.
<b>ARL</b>	4	rh	<b>Arbitration Lost</b> 0: No arbitration conflict detected. 1: Arbitration for transmission has been lost. Cleared by software or by hardware if the arbitration has been won or the transmission has been aborted.
<b>H</b>	5	rh	<b>H Bit in Consolidated Headers</b> This bit monitors the status of bit 4 of the first byte in a frame on bus side.
<b>K</b>	6	rh	<b>K Bit in 3 Byte Consolidated Headers</b> This bit monitors the status of bit 3 of the first byte in a frame on bus side.
<b>Y</b>	7	rh	<b>Y Bit in 3 Byte Consolidated Headers</b> This bit monitors the status of bit 2 of the first byte in a frame on bus side.

**The Serial Data Link Module (SDLM)**

The Bus Status Register BUSSTAT contains the bus-related status bits.

**BUSSTAT**

**Bus Status Register**

**XReg (EB20<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	<b>IDLE</b>	<b>END F</b>	<b>EOD</b>	<b>SOF</b>
-	-	-	-	-	-	-	-	-	-	-	-	rh	rh	rh	rh

Field	Bits	Type	Description
<b>SOF</b>	0	rh	<b>Start Of Frame Detected</b> 0: No SOF detected since last reset via BUSRST. 1: SOF has been detected.
<b>EOD</b>	1	rh	<b>End Of Data Detected</b> 0: No EOD detected since last reset via BUSRST. 1: EOD has been detected.
<b>ENDF</b>	2	rh	<b>End Of Frame Detected</b> 0: No EOF detected since last reset via BUSRST. 1: EOF has been detected.
<b>IDLE</b>	3	rh	<b>Bus Idle</b> 0: A transfer takes place on the J1850 bus. 1: IFS has been detected.

The Serial Data Link Module (SDLM)

The Error Status Register ERRSTAT contains error bits. The bits in this register have to be reset by SW.

**ERRSTAT**

Error Status Register

XReg (EB22<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	CRC ER	COL	SHO RTH	SHO RTL	FOR MAT
-	-	-	-	-	-	-	-	-	-	-	rh	rh	rh	rh	rh

Field	Bits	Type	Description
<b>FORMAT</b>	0	rh	<b>Format Error</b> 0: No format error detected. 1: A frame/byte length error or symbol/bit timing error has occurred.
<b>SHORTL</b>	1	rh	<b>Bus Shorted Low</b> 0: No bus short detected. 1: The bus returns a recessive level while a dominant level is sent.
<b>SHORTH</b>	2	rh	<b>Bus Shorted High</b> 0: No bus short detected. 1: A '1' is detected on the bus for more than 1 s.
<b>COL</b>	3	rh	<b>Collision Detected (lost arbitration)</b> 0: No collision has been detected on the bus while transmitting. 1: At least one collision was detected on the bus. <i>Note: Bit COL must be cleared via software by setting bit ERRST.</i>
<b>CRCER</b>	4	rh	<b>CRC Error</b> 0: The CRC check was OK. 1: The calculated CRC differs from the received CRC.

**The Serial Data Link Module (SDLM)**

The Buffer Control Register BUFFCON contains the transfer-related control bits, including IFR control and FIFO control.

**BUFFCON**

**Buffer Control Register**

**XReg (EB24<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	RX INCE	TX INCE	IF REN	CR CEN	SB RK	DO NE	TX RQ	TX IFR
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
<b>TXIFR</b>	0	rwh	<b>Transmit In-Frame Response</b> Setting bit TXIFR declares the transmit buffer or the IFR register (if IFREN = 1, IFR type 1, 2, others than 3-Byte consolidated header) to be valid for IFR and initiates its transmission. TXIFR is automatically reset by hardware after successful transmission. Resetting TxIFR by software stops the transmission of the IFR.
<b>TXRQ</b>	1	rwh	<b>Transmit Request</b> Setting bit TXRQ declares the transmit buffer to be valid and starts its transmission. TXRQ is automatically reset by hardware after successful transmission. Resetting TxRQ by software stops the transmission in normal mode and in block mode. TXRQ cannot be used to start IFR transmission from the transmit buffer. If TXRQ is reset, TXCPU is cleared.
<b>DONE</b>	2	rwh	<b>Receive Buffer on CPU Side Read Out Done</b> Setting bit DONE declares the receive buffer on CPU side to be empty, resets RXCPU and releases the buffer (reset of RBC). If there is a full receive buffer on bus side, the buffers are swapped. This bit is reset by hardware after the buffer has been released.
<b>SBRK</b>	3	rwh	<b>Send Break</b> Setting bit SBRK initiates the transmission of a break symbol on the J1850 bus. Bit SBRK is reset by hardware after having sent the break symbol.

**The Serial Data Link Module (SDLM)**

Field	Bits	Type	Description
<b>CRCEEN</b>	4	rw	<p><b>CRC Enable</b></p> <p>0: No CRC generation for type IFR via TxBuffer.            1: CRC enabled for IFR via TxBuffer.</p> <p>If IFR is sent from IFRVAL, no CRC is generated (even if CRCEEN = 1). CRC generation is always enabled in normal mode and in block mode.</p>
<b>IFREN</b>	5	rw	<p><b>In-Frame Response Enable</b></p> <p>Setting bit IFREN enables the automatic IFR (type1, 2 for 3 byte consolidated header) of the SDLM module with the value stored in IFRVAL. If the IFR request can not be automatically detected (all headers, except see above), IFREN selects the data source for types 1, 2 IFR initiated by TXIFR.</p> <p>0: Transmit buffer contains IFR data byte(s), IFR types 1, 2, 3 supported, CRC depending on CRCEEN.            1: IFRVAL contains data byte for types 1, 2 IFR. Automatic IFR for types 1, 2 for 3-Byte consolidated headers supported. No CRC is used.</p>
<b>TXINCE</b>	6	rw	<p><b>Transmit Buffer Increment Enable<sup>1)</sup></b></p> <p>0: No FIFO mode for transmit buffer.            1: FIFO mode enabled for transmit buffer.</p> <p>TXCPU is incremented by one after each CPU write operation to register TXD0. In block mode, FIFO mode is automatically enabled (independent of TXINCE).</p>
<b>RXINCE</b>	7	rw	<p><b>Receive Buffer Increment Enable<sup>1)</sup></b></p> <p>0: No FIFO mode for receive buffer on CPU side.            1: FIFO mode enabled for receive buffer (CPU).</p> <p>RXCPU is incremented by one after each CPU read operation from register RXD00. In block mode, FIFO mode is automatically enabled (independent of RXINCE).</p>

<sup>1)</sup> Random access mode is always enabled.

The Serial Data Link Module (SDLM)

The Flag Reset Register FLAGRST contains the control bits to reset the error flags, the bus-related flags and the transfer-related status bits.

**FLAGRST**

Flag Reset Register

XReg (EB28<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	ER RST	BUS RST	RX RST	TX RST	ARL RST	BRK RST
-	-	-	-	-	-	-	-	-	-	wh	wh	wh	wh	wh	wh

Field <sup>1)</sup>	Bits	Type	Description
<b>BRKRST</b>	0	wh	<b>Reset Buffer Status</b> (clears bit BREAK) Automatically cleared by HW after clearing bit BREAK.
<b>ARLRST</b>	1	wh	<b>Reset Buffer Status</b> (clears bit ARL) Automatically cleared by HW after clearing bit ARL.
<b>TXRST</b>	2	wh	<b>Reset Buffer Status</b> (clears bit MSGTRA) Automatically cleared by HW after clearing bit MSGTRA.
<b>RXRST</b>	3	wh	<b>Reset Buffer Status</b> (clears bits MSGREC, MSGLST) Automatically cleared by HW after clearing bits MSGREC and MSGLST.
<b>BUSRST</b>	4	wh	<b>Reset Bus Status</b> (clears bits ENDF, EOD, SOF) Automatically cleared by HW after clearing bits ENDF, EOD and SOF.
<b>ERRST</b>	5	wh	<b>Reset Error</b> (clears bits SHORTH, SHORTL, COL, CRCER, FORMAT) Automatically cleared by HW after clearing bits SHORTH, SHORTL, COL, CRCER and FORMAT.

<sup>1)</sup> All bits return '0' when read.

The Serial Data Link Module (SDLM)

The Interrupt Control Register INTCON enables the different interrupt sources of the SDLM.

**INTCON**

Interrupt Control Register

XReg (EB2C<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	ERR IE	CRC IE	ARL IE	BRK IE	ENDF IE	HD IE	REC IE	TRA IE
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
<b>TRAIE</b>	0	rw	<b>Enable Transmit Interrupt</b> 0: No transmission interrupt. 1: An interrupt is generated if bit MSGTRA is set.
<b>RECIE</b>	1	rw	<b>Enable Receive Interrupt</b> 0: No receive interrupt. 1: An interrupt is generated if bit MSGREC is set.
<b>HDIE</b>	2	rw	<b>Enable Header Received Interrupt</b> 0: No header interrupt. 1: An interrupt is generated if bit HEADER is set.
<b>ENDFIE</b>	3	rw	<b>Enable End of Frame Detection</b> 0: No end-of-frame interrupt. 1: An interrupt is generated if bit ENDF is set.
<b>BRKIE</b>	4	rw	<b>Enable Break Received Interrupt</b> 0: No break interrupt. 1: An interrupt is generated if bit BREAK is set.
<b>ARLIE</b>	5	rw	<b>Enable Arbitration Lost Interrupt</b> 0: No arbitration-lost interrupt. 1: An interrupt is generated if bit ARL is set.
<b>CRCIE</b>	6	rw	<b>Enable CRC Error Interrupt</b> 0: No CRC error interrupt. 1: An interrupt is generated if bit CRCER is set.
<b>ERRIE</b>	7	rw	<b>Enable Error Interrupt</b> 0: No error interrupt. 1: An interrupt is generated if one of the bits SHORTH, SHORTL, COL or FORMAT is set.

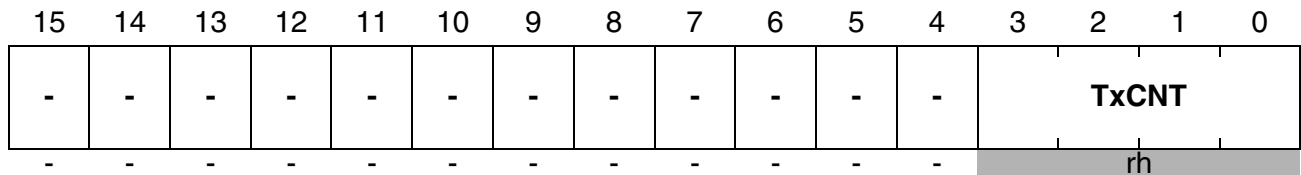
The Serial Data Link Module (SDLM)

**Data Handling Registers**

The Bus Transmit Byte Counter Register TXCNT indicates the number of bytes of the transmit buffer which have already been sent out on the bus.

**TXCNT**

**Bus Transmit Byte Count Reg. XReg (EB3C<sub>H</sub>)** **Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
<b>TxCNT</b>	[3:0]	rh	<p><b>Bus Transmit Byte Counter</b></p> <p>Contains the number of transmitted bytes.            TxCNT is cleared upon clearing bit TxRQ.            A transmit interrupt can be generated when TxRQ is reset by hardware (TxCPU==TxCNT and successful transmission, i.e. arbitration not lost resets bit TxRQ by hardware in Normal Mode).            TxCNT = pointer for SDLM access to transmit buffer.</p>

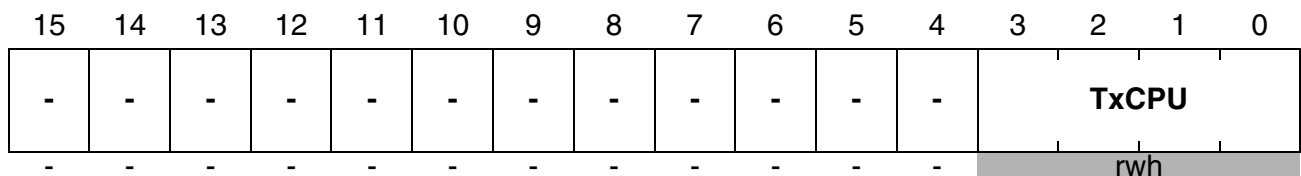


**The Serial Data Link Module (SDLM)**

The CPU Byte Counter Register TXCPU contains the pointer to the next empty byte in the transmit buffer (= number of bytes in the transmit buffer).

**TXCPU**

**CPU Transmit Byte Count Reg. XReg (EB3E<sub>H</sub>) Reset Value: 0000<sub>H</sub>**

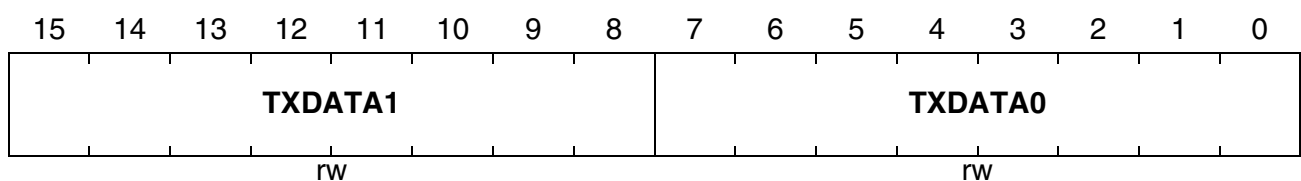


Field	Bits	Type	Description
<b>TxCPU</b>	[3:0]	rwh	<p><b>CPU Transmit Byte Counter</b></p> <p>Contains the number of bytes, which have been written to the transmit buffer by the CPU.</p> <p>In FIFO mode (TxINCE = '1' or BMEN = '1'), TxCPU is incremented by 1 after each CPU write action to TXD0.</p> <p>In random mode only (TxINCE = 0 and BMEN = 0) software must write TxCPU before setting the transmit request bit (TxRQ) in order to define the number of bytes to be sent.</p> <p>TxCPU is cleared upon clearing bit TxRQ in normal mode or upon clearing BMEN.</p> <p>TxCPU = pointer for CPU access to transmit buffer in FIFO mode</p>

The transmit data registers contain the data bytes in the transmit buffer. In random mode, all data bytes can be directly accessed via their addresses, whereas in FIFO mode, only TXD0 should be used.

**TXD0**

**Transmit Data Register 0 XReg (EB30<sub>H</sub>) Reset Value: 0000<sub>H</sub>**



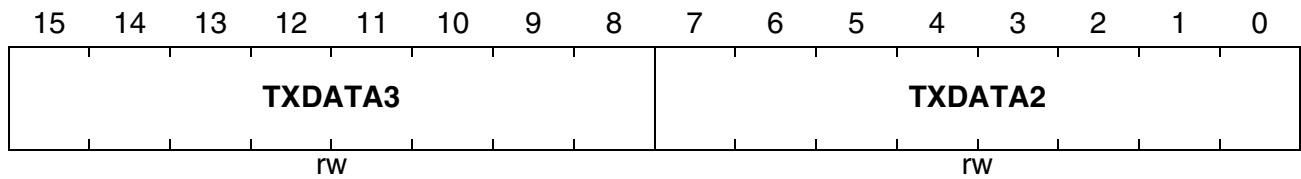
The Serial Data Link Module (SDLM)

**TXD2**

**Transmit Data Register 2**

**XReg (EB32<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

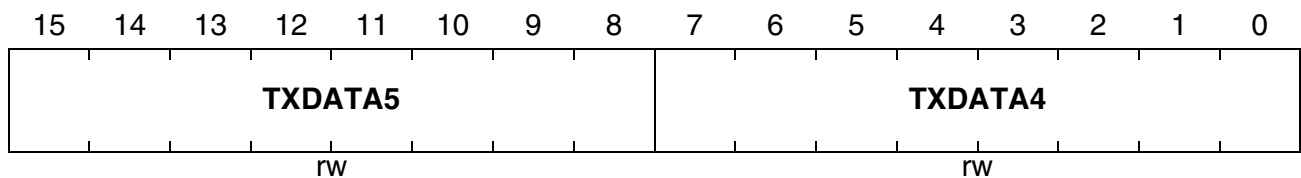


**TXD4**

**Transmit Data Register 4**

**XReg (EB34<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

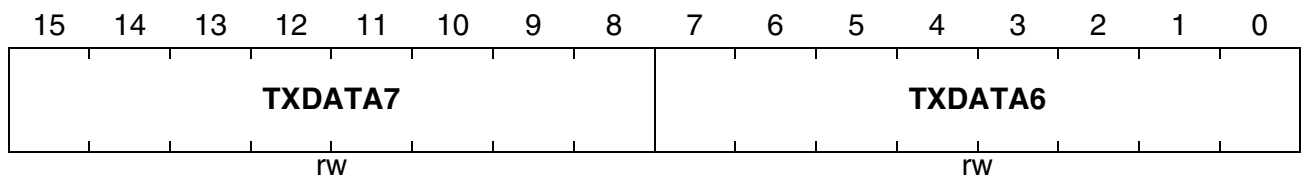


**TXD6**

**Transmit Data Register 6**

**XReg (EB36<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

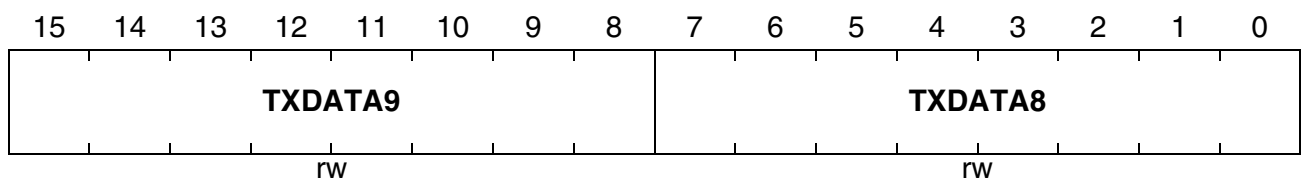


**TXD8**

**Transmit Data Register 8**

**XReg (EB38<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

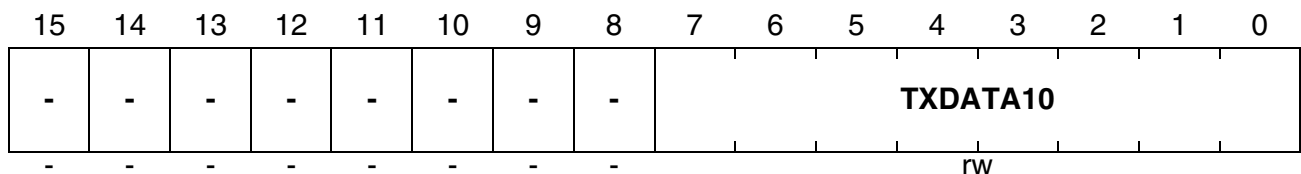


**TXD10**

**Transmit Data Register 10**

**XReg (EB3A<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**



Field	Bits	Type	Description
TXDATA <sub>n</sub>	[7:0], [15:8]	rw	Transmit Buffer Data Byte n (n = 0 ... 10)

**The Serial Data Link Module (SDLM)**

The Bus Receive Byte Counter Register RXCNT contains the number of bytes received in this buffer.

**RXCNT**

**Bus Rec. Byte Counter (CPU)      XReg (EB4C<sub>H</sub>)      Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	<b>RxCNT</b>			
-	-	-	-	-	-	-	-	-	-	-	-	rh			

Field	Bits	Type	Description
<b>RxCNT</b>	[3:0]	rh	<p><b>Receive Byte Count</b>            Contains the number of received bytes in the receive buffer on CPU side.            RxCNT is reset when the receive buffer on CPU side is released (see DONE).            RxCNT = pointer for SDLM access to receive buffer.</p>

The CPU Receive Byte Counter Register RXCPU contains the number of bytes already read out from this buffer.

**RXCPU**

**CPU Rec. Byte Counter (CPU)      XReg (EB4E<sub>H</sub>)      Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	<b>RxCPU</b>			
-	-	-	-	-	-	-	-	-	-	-	-	rwh			

Field	Bits	Type	Description
<b>RxCPU</b>	[3:0]	rwh	<p><b>CPU Receive Byte Count</b>            Contains the number of bytes read out by the CPU.            In FIFO mode (RxINCE = '1' or BMEN = '1'), RXCPU is incremented by 1 after each CPU read action to register RxD00.            In random mode (RxINCE = 0 and BMEN = 0), RxCPU is not used and is 0.            RxCPU is reset when the receive buffer on CPU side is released.            RxCPU = pointer for CPU access to receive buffer.</p>

**The Serial Data Link Module (SDLM)**

The Bus Receive Byte Counter Register RXCNTB contains the bitfield indicating the number of received bytes in the receive buffer on bus side.

**RXCNTB**

**Bus Rec. Byte Counter (bus)      XReg (EB5C<sub>H</sub>)      Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	<b>RxCNTB</b>			
-	-	-	-	-	-	-	-	-	-	-	-	rh			

Field	Bits	Type	Description
<b>RxCNTB</b>	[3:0]	rh	<b>Receive Byte Counter</b> Contains the number of received bytes in the receive buffer on bus side.

The Start-of-Frame Pointer Register SOFPTR contains the bitfield indicating the value of RXCNT after the last ENDF detection in block mode.

**SOFPTR**

**Start-of-Frame Pointer Register    XReg (EB60<sub>H</sub>)      Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	<b>SOFCNT</b>			
-	-	-	-	-	-	-	-	-	-	-	-	rh			

Field	Bits	Type	Description
<b>SOFCNT</b>	[3:0]	rh	<b>Start-of-Frame Counter for Block Mode</b> The value of bitfield RxCNT is automatically copied to this bitfield if an end-of-frame symbol is detected. This feature can be used in block mode to determine the position of the first new byte of a frame in the 16-Byte receive buffer.

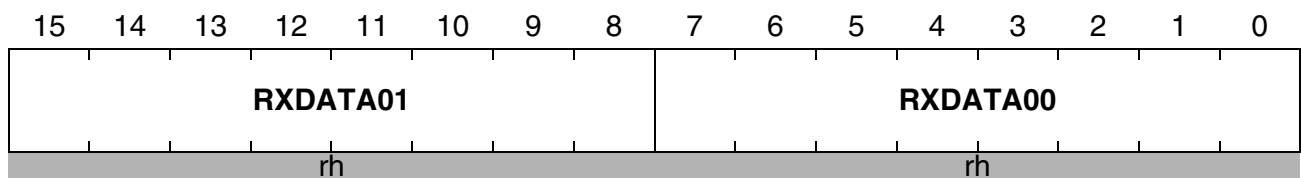
The Serial Data Link Module (SDLM)

The receive data registers contain the data bytes in the receive buffer. In random mode, all data bytes can be directly accessed via their addresses, whereas in FIFO mode, only RXD00 should be used.

Bitfields RXDATA0n (n = 0 ... 10) represent the receive buffer 0 on CPU side, bitfields RXDATA1n (n = 0 ... 10) represent the receive buffer 1 on bus side. In block mode, the 16-Byte receive buffer is built by bitfields RXDATA00-07 and bitfields RXDATA10-17.

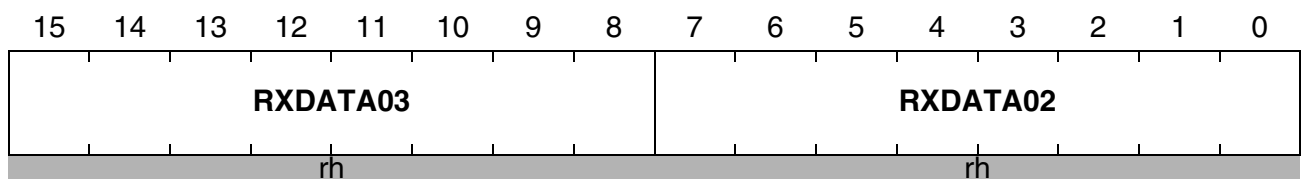
**RXD00**

**Receive Data Register 00 (CPU) XReg (EB40<sub>H</sub>) Reset Value: 0000<sub>H</sub>**



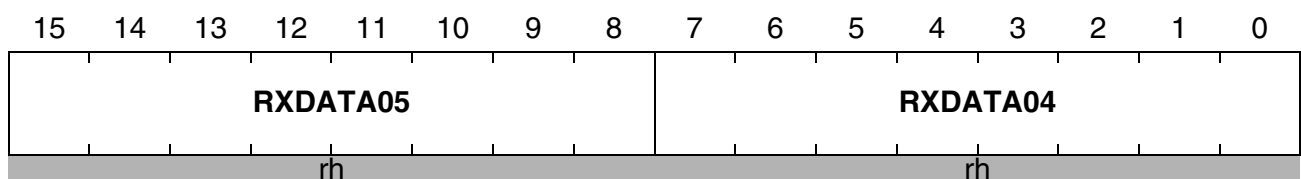
**RXD02**

**Receive Data Register 02 (CPU) XReg (EB42<sub>H</sub>) Reset Value: 0000<sub>H</sub>**



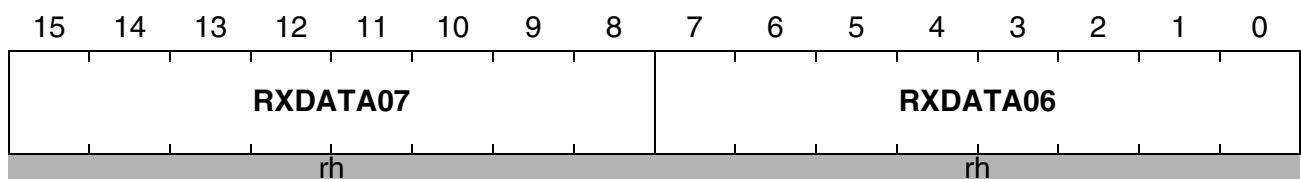
**RXD04**

**Receive Data Register 04 (CPU) XReg (EB44<sub>H</sub>) Reset Value: 0000<sub>H</sub>**



**RXD06**

**Receive Data Register 06 (CPU) XReg (EB46<sub>H</sub>) Reset Value: 0000<sub>H</sub>**

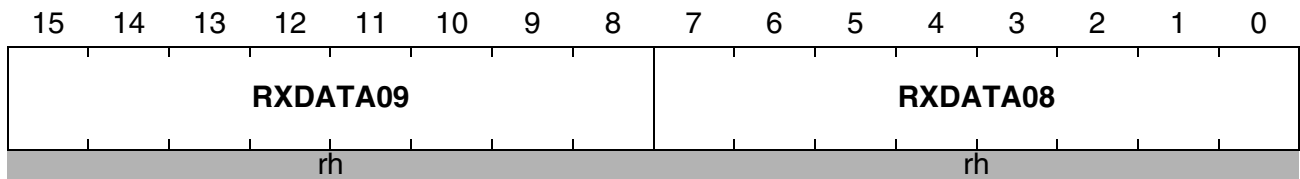


The Serial Data Link Module (SDLM)

**RXD08**

Receive Data Register 08 (CPU) XReg (EB48<sub>H</sub>)

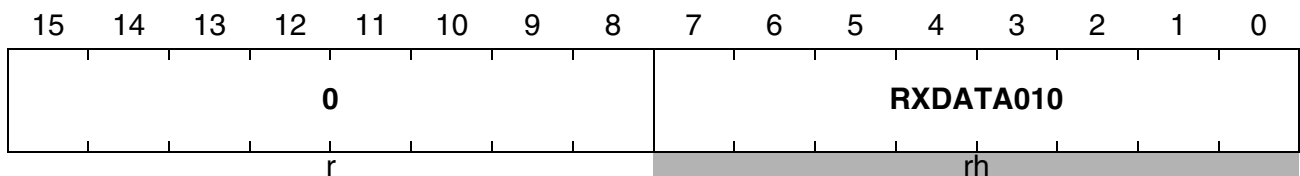
Reset Value: 0000<sub>H</sub>



**RXD010**

Receive Data Register 010 (CPU) XReg (EB4A<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

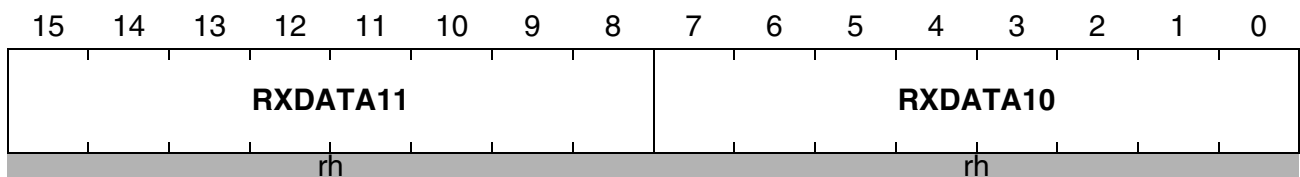


Field	Bits	Type	Description
RXDATA0n	[7:0], [15:8]	rh	Receive Buffer 0 Data Byte n

**RXD10**

Receive Data Register 10 (bus) XReg (EB50<sub>H</sub>)

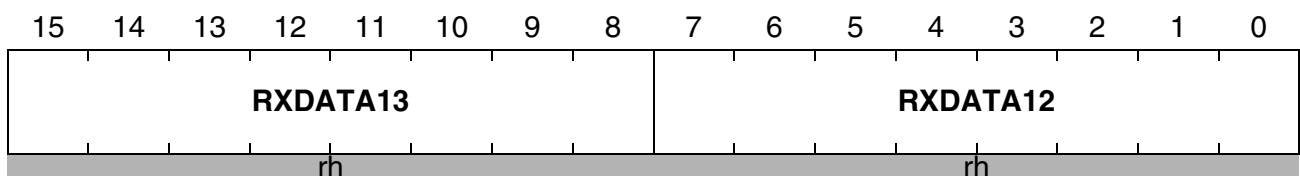
Reset Value: 0000<sub>H</sub>



**RXD12**

Receive Data Register 12 (bus) XReg (EB52<sub>H</sub>)

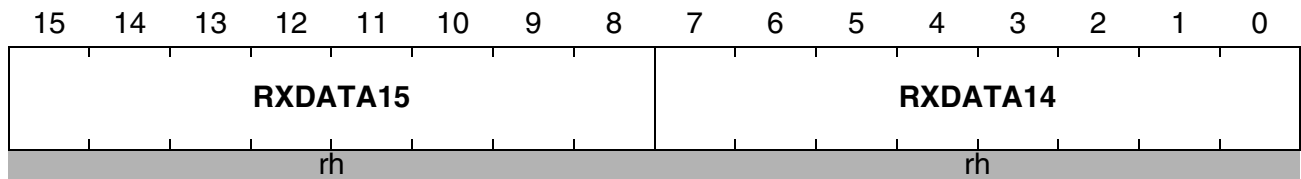
Reset Value: 0000<sub>H</sub>



The Serial Data Link Module (SDLM)

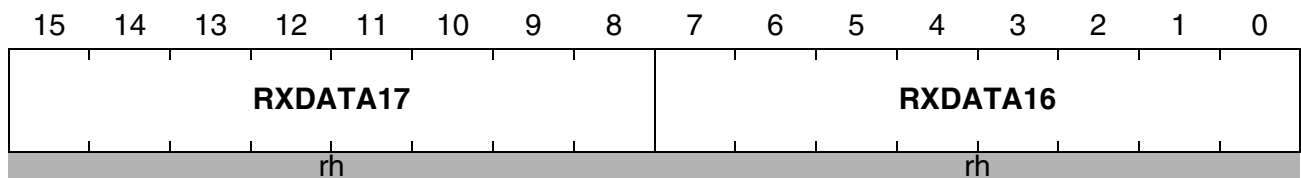
**RXD14**

Receive Data Register 14 (bus) XReg (EB54<sub>H</sub>) Reset Value: 0000<sub>H</sub>



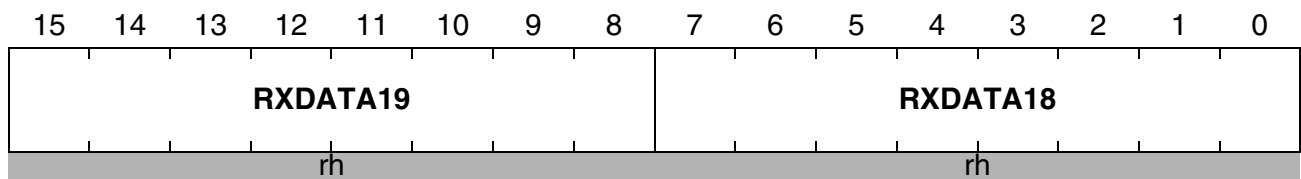
**RXD16**

Receive Data Register 16 (bus) XReg (EB56<sub>H</sub>) Reset Value: 0000<sub>H</sub>



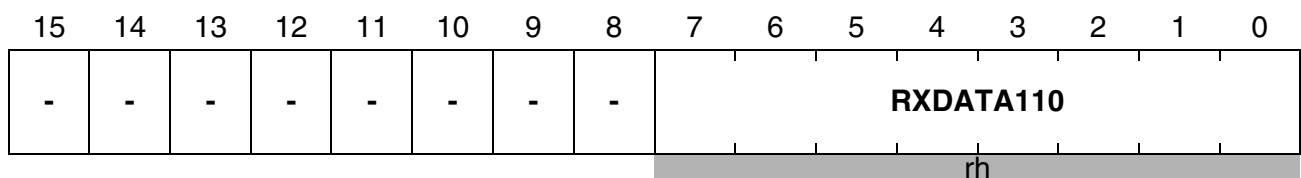
**RXD18**

Receive Data Register 18 (bus) XReg (EB58<sub>H</sub>) Reset Value: 0000<sub>H</sub>



**RXD110**

Receive Data Register 110 (bus) XReg (EB5A<sub>H</sub>) Reset Value: 0000<sub>H</sub>



Field	Bits	Type	Description
RXDATA1n	[7:0], [15:8]	rh	Receive Buffer 1 Data Byte n

## 20.8 Interrupt Node Control

The interrupt node control register XP7IC is not part of the SDLM (no XBUS register) but rather located within the C161CS/JC/JI's interrupt controller.

### XP7IC

Interrupt Node Control

ESFR (F19A<sub>H</sub>/CD<sub>H</sub>)

Reset Value: - - 00<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								XP7 IR	XP7 IE	ILVL			GLVL		
-	-	-	-	-	-	-	-	rwh	rw	rw			rw		

*Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.*



## 21 The IIC Bus Module

The on-chip IIC Bus module (Inter Integrated Circuit) connects the C161CS/JC/JI to other external controllers and/or peripherals via the two-line serial IIC interface. The IIC Bus module provides communication at data rates of up to 400 kbit/s in master and/or slave mode and features 7-bit addressing as well as 10-bit addressing.

*Note: The IIC Bus module is an XBUS peripheral and therefore requires bit XPEN in register SYSCON to be set in order to be operable.*

Core Registers	Control Registers	Data Registers	Interrupt Control
SYSCON	ICCFG X	ICRTB X	XP0IC E
SYSCON3E	ICCON X	ICADR X	XP1IC E
P9	ICST X		
DP9			

SYSCON	System Configuration Register	ICCFG	IIC Configuration Register
SYSCON3	Peripheral Management Control Register	ICCON	IIC Control Register
P9	Port 9 Data Register	ICST	IIC Status Register
DP9	Port 9 Direction Control Register	ICRTB	IIC Receive Transmit Buffer
XP0IC	IIC Data Interrupt Control Register	ICADR	IIC Address Register
XP1IC	IIC Protocol Interrupt Control Register		

MCA04848

**Figure 21-1 SFRs Associated with the IIC Bus Module**

The module can operate in three different modes:

- **Master mode**, where the C161CS/JC/JI controls the bus transactions and provides the clock signal.
- **Slave mode**, where an external master controls the bus transactions and provides the clock signal.
- **Multimaster mode**, where several masters can be connected to the bus, i.e. the C161CS/JC/JI can be master or slave.

The on-chip IIC bus module allows efficient communication over the common IIC bus. The module unloads the CPU of the C161CS/JC/JI of low level tasks like

- (De)Serialization of bus data
- Generation of start and stop conditions
- Monitoring the bus lines in slave mode
- Evaluation of the device address in slave mode
- Bus access arbitration in multimaster mode

## 21.1 IIC Bus Conditions

Data is transferred over the 2-line IIC bus (SDA, SCL) using a protocol that ensures reliable and efficient transfers. This protocol clearly distinguishes regular data transfers from defined control signals which control the data transfers.

The following bus conditions are defined:

- Bus Idle:** SDA and SCL remain high. The IIC bus is currently not used.
- Data Valid:** SDA stable during the high phase of SCL. SDA then represents the transferred bit. There is one clock pulse for each transferred bit of data. During data transfers SDA may only change while SCL is low (see below)!
- Start Transfer:** A falling edge on SDA ( $\searrow$ ) while SCL is high indicates a start condition. This start condition initiates a data transfer over the IIC bus.
- Stop Transfer:** A rising edge on SDA ( $\nearrow$ ) while SCL is high indicates a stop condition. This stop condition terminates a data transfer. Between a start condition and a stop condition an arbitrary number of bytes may be transferred.

**Figure 21-2** gives examples for these bus conditions.

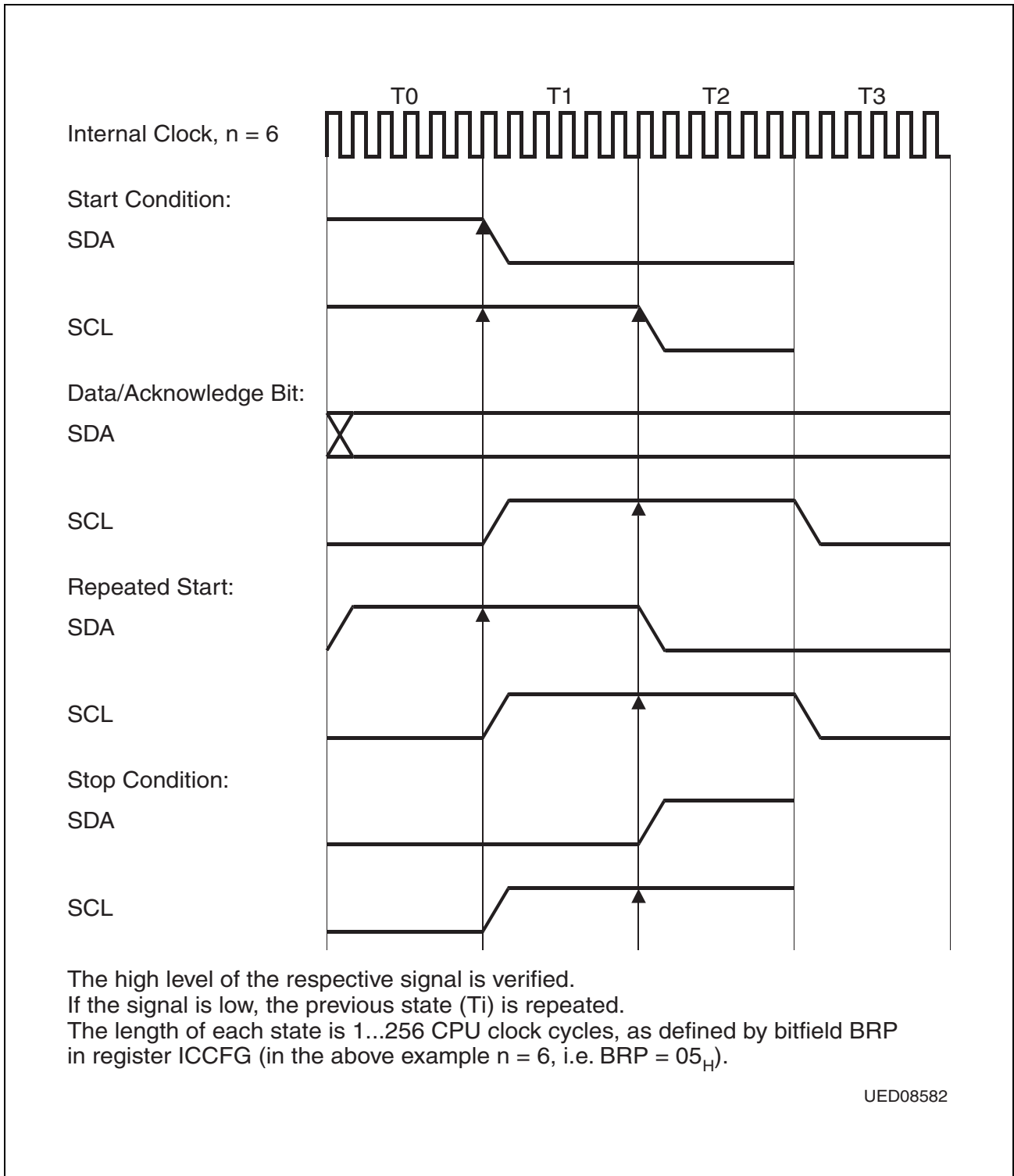
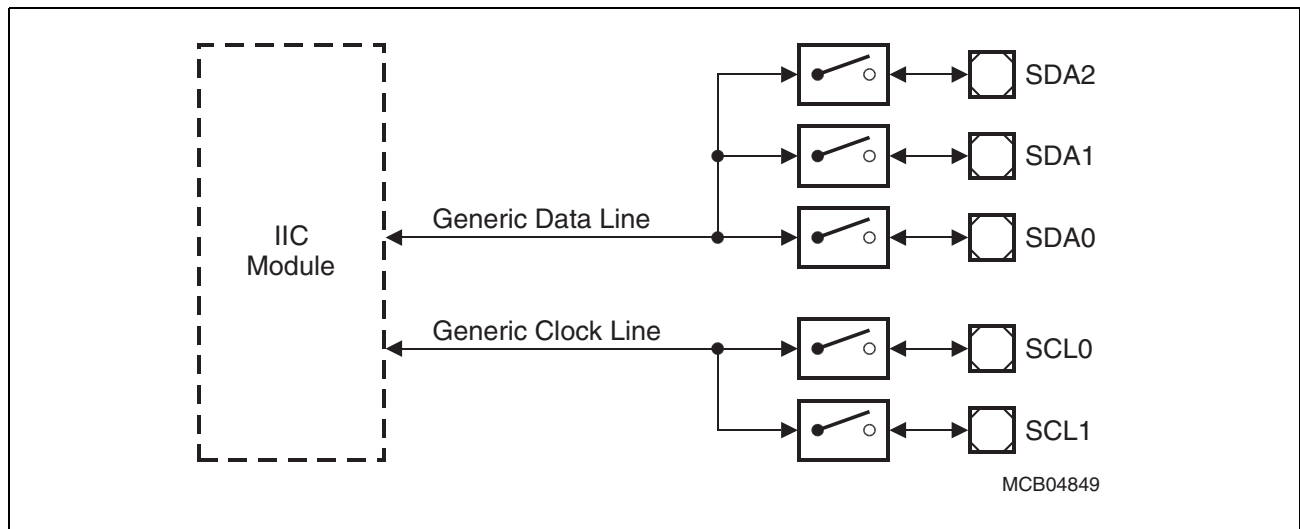


Figure 21-2 IIC Bus Conditions

## 21.2 The Physical IIC Bus Interface

Communication via the IIC Bus uses two bidirectional lines, the serial data line SDA and the serial clock line SCL. These two generic interface lines can each be connected to a number of IO port lines of the C161CS/JC/JI (see [Figure 21-3](#)). These connections can be established and released under software control.



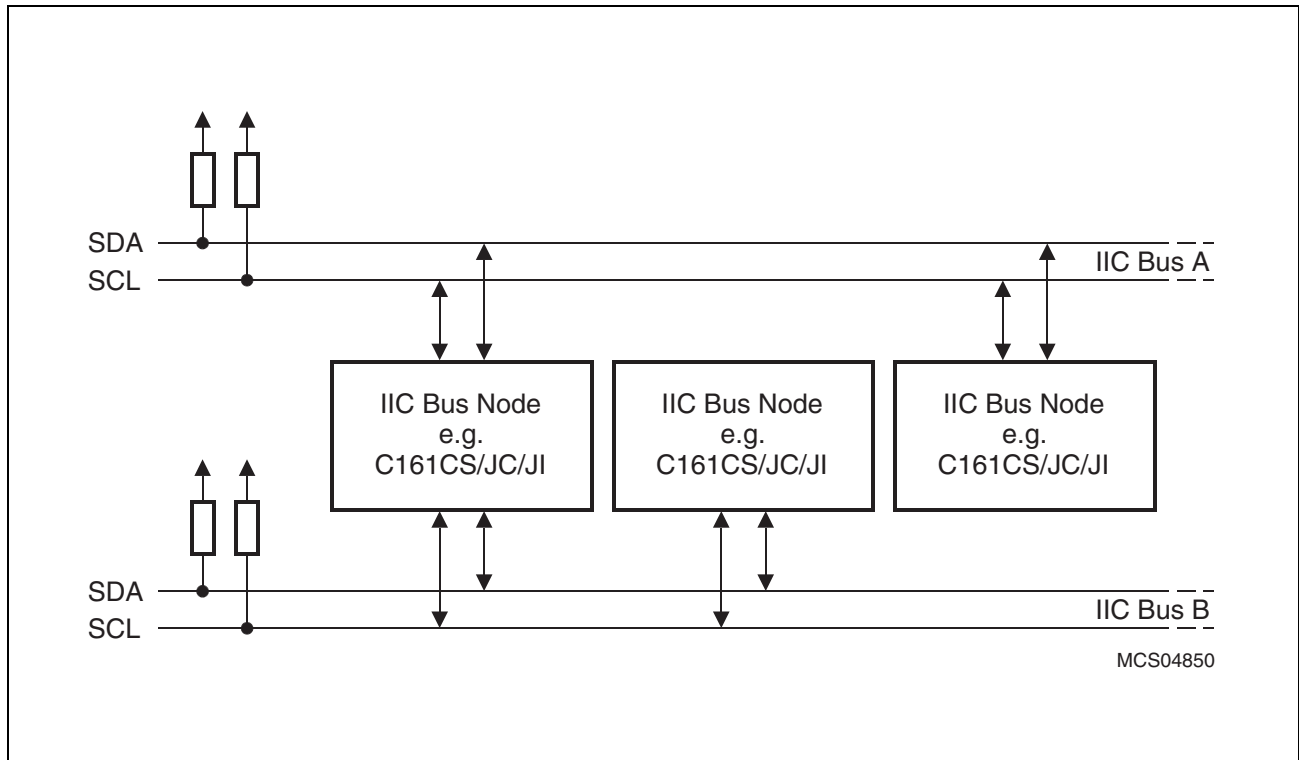
**Figure 21-3 IIC Bus Line Connections**

This mechanism allows a number of configurations of the physical IIC Bus interface:

**Channel switching:** The IIC module can be connected to a specific pair of pins (e.g. SDA0 and SCL0) which then forms a separate IIC channel to the external system. The channel can be dynamically switched by connecting the module to another pair of pins (e.g. SDA1 and SCL1). This establishes physically separate interface channels.

**Broadcasting:** Connecting the module to more than one pair of pins (e.g. SDA0/1 and SCL0/1) allows the transmission of messages over multiple physical channels at the same time. Please note that this configuration is critical when the C161CS/JC/JI is a slave or receives data.

*Note: Never change the physical bus interface configuration while a transfer is in progress.*



**Figure 21-4 Physical Bus Configuration Example**

### Output Pin Configuration

The pin drivers that are assigned to the IIC channel(s) provide open drain outputs (i.e. no upper transistor). This ensures that the IIC module does not put any load on the IIC bus lines while the C161CS/JC/JI is not powered. The IIC bus lines therefore require external pullup resistors (approx. 10 k $\Omega$  for operation at 100 kBaud, 2 k $\Omega$  for operation at 400 kBaud).

*Note: If the pins that are assigned to the IIC channel(s) are to be used as general purpose IO they must be used for open drain outputs or as inputs.*

All pins of the C161CS/JC/JI that are to be used for IIC bus communication must be switched to output and their alternate function must be enabled (by setting the respective port output latch to '1'), before any communication can be established.

If not driven by the IIC module (i.e. the corresponding enable bit in register ICCFG is '0') they then switch off their drivers (i.e. driving '1' to an open drain output). Due to the external pullup devices the respective bus levels will then be '1' which is idle.

The IIC module features digital input filters in order to improve the rejection of noise from the external bus lines.

## 21.3 Operating the IIC Bus

The on-chip IIC bus module of the C161CS/JC/JI can be operated in variety of operating modes.

**Master or Slave** operation can be selected, so the IIC module can control the external bus (master) or can be controlled via the bus (slave) by a remote master.

**7-bit or 10-bit** addressing can be selected, so the IIC module can communicate with standard 7-bit devices as well as with more sophisticated 10-bit devices.

**100 kBaud or 400 kBaud** transfer speed can be selected, so the IIC module can communicate with slow devices conforming to the standard IIC bus specification as well as with fast devices conforming to the extended specification.

**Physical channels** can be selected, so the IIC module can use electrically separated channels or increase the addressing range by using more data lines.

*Note: Baudrate and physical channels should never be changed (via ICCFG) during a transfer.*

### 21.3.1 Operation in Master Mode

If the on-chip IIC module shall control the IIC bus (i.e. be bus master) master mode must be selected via bitfield MOD in register ICCON. The physical channel is configured by a control word written to register ICCFG, defining the active interface pins and the used baudrate. More than one SDA and/or SCL line may be active at a time. The address of the remote slave that is to be accessed is written to ICRTB. The bus is claimed by setting bit BUM in register ICCON. This generates a start condition on the bus and automatically starts the transmission of the address in ICRTB. Bit TRX in register ICCON defines the transfer direction (TRX = '1', i.e. transmit, for the slave address). A repeated start condition is generated by setting bit RSC in register ICCON, which automatically starts the transmission of the address previously written to ICRTB. This may be used to change the transfer direction. RSC is cleared automatically after the repeated start condition has been generated.

The bus is released by clearing bit BUM in register ICCON. This generates a stop condition on the bus.

### 21.3.2 Operation in Multimaster Mode

If multimaster mode is selected via bitfield MOD in register ICCON the on-chip IIC module can operate concurrently as a bus master or as a slave. The descriptions of these modes apply accordingly.

Multimaster mode implies that several masters are connected to the same bus. As more than one master may try to claim the bus at a given time an arbitration is done on the SDA line. When a master device detects a mismatch between the data bit to be sent and the actual level on the SDA (bus) line it loses the arbitration and automatically switches to slave mode (leaving the other device as the remaining master). This loss of arbitration is indicated by bit AL in register ICST which must be checked by the driver software when operating in multimaster mode. Lost arbitration is also indicated when the software tries to claim the bus (by setting bit BUM) while the IIC module is operating in slave mode (indicated by bit BB = '1').

Bit AL must be cleared via software.

### 21.3.3 Operation in Slave Mode

If the on-chip IIC module shall be controlled via the IIC bus by a remote master (i.e. be a bus slave) slave mode must be selected via bitfield MOD in register ICCON. The physical channel is configured by a control word written to register ICCFG, defining the active interface pins and the used baudrate. It is recommended to have only one SDA and SCL line active at a time when operating in slave mode. The address by which the slave module can be selected is written to register ICADR.

The IIC module is selected by another master when it receives (after a start condition) either its own device address (stored in ICADR) or the general call address (00<sub>H</sub>). In this case an interrupt is generated and bit SLA in register ICST is set indicating the valid selection. The desired transfer mode is then selected via bit TRX (TRX = '0' for reception, TRX = '1' for transmission).

**For a transmission** the respective data byte is placed into the buffer ICRTB (which automatically sets bit TRX) and the acknowledge behavior is selected via bit ACKDIS.

**For a reception** the respective data byte is fetched from the buffer ICRTB after IRQD has been activated.

**In both cases** the data transfer itself is enabled by clearing bit IRQP which releases the SCL line.

When a stop condition is detected bit SLA is cleared.

The IIC bus configuration register ICCFG selects the bus baudrate as well as the activation of SDA and SCL lines. So an external IIC channel can be established (baudrate and physical lines) with one single register access.

**The IIC Bus Module**

Systems that utilize several IIC channels can prepare a set of control words which configure the respective channels. By writing one of these control words to ICCFG the respective channel is selected. Different channels may use different baudrates. Also different operating modes can be selected, e.g. enabling all physical interfaces for a broadcast transmission.

*Note: See also [Section 21.2](#).*

**ICCFG**

**IIC Configuration Reg.**

**XReg (ED00<sub>H</sub>)**

**Reset Value: XX00<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>BRP</b>						-	-	<b>SCL SEL 1</b>	<b>SCL SEL 0</b>	-	<b>SDA SEL 2</b>	<b>SDA SEL 1</b>	<b>SDA SEL 0</b>		
rW						-	-	rW	rW	-	rW	rW	rW		

Bit	Function
<b>SDASELx</b>	<b>SDA Pin Selection</b> These bits determine to which pins the IIC data line is connected. 0: SDA pin x is disconnected. 1: SDA pin x is connected with IIC data line.
<b>SCLSELx</b>	<b>SCL Pin Selection</b> These bits determine to which pins the IIC clock line is connected. 0: SCL pin x is disconnected. 1: SCL pin x is connected with IIC clock line.
<b>BRP</b>	<b>Baudrate Prescaler</b> Determines the baudrate for the active IIC channel(s). The resulting baudrate is $B_{IIC} = f_{CPU} / (4 \times (BRP + 1))$ . See <a href="#">Table 21-1</a> .

**Table 21-1 IIC Bus Baudrate Selection**

CPU Frequency $f_{CPU}$	Reload Value for BRP	
	100 kBaud	400 kBaud
20 MHz	31 <sub>H</sub>	0B <sub>H</sub> or 0C <sub>H</sub>
16 MHz	27 <sub>H</sub>	09 <sub>H</sub>
12 MHz	1D <sub>H</sub>	06 <sub>H</sub> or 07 <sub>H</sub>
10 MHz	18 <sub>H</sub>	05 <sub>H</sub>
1 MHz	01 <sub>H</sub> or 02 <sub>H</sub>	Not possible.



**ICCON**

**IIC Control Register**

**XReg (ED02<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	TRX	AIR DIS	ACK DIS	BUM	MOD	RSC	M10	
-	-	-	-	-	-	-	-	rwh	rw	rw	rwh	rw	rwh	rw	

Bit	Function
<b>M10</b>	<p><b>Address Mode</b></p> <p>0: 7-bit addressing using ICA.7-1. 1: 10-bit addressing using ICA.9-0.</p>
<b>RSC</b>	<p><b>Repeated Start Condition</b></p> <p>0: No operation. RSC is cleared automatically after the repeated start condition has been sent. 1: Generate a repeated start condition in (multi)master mode. RSC cannot be set in slave mode.</p>
<b>MOD</b>	<p><b>Basic Operating Mode</b></p> <p>00: IIC module is disabled and initialized. 01: Slave mode. 10: Master mode. 11: Multi-Master mode.</p>
<b>BUM</b>	<p><b>Busy Master</b></p> <p>0: Clearing bit BUM (↯) generates a stop condition. 1: Setting bit BUM generates a start condition in (multi)master mode. Setting BUM (↯) while BB = '1' generates an arbitration lost situation. In this case BUM is cleared and bit AL is set. BUM cannot be set in slave mode.</p>
<b>ACKDIS</b>	<p><b>Acknowledge Pulse Disable</b></p> <p>0: An acknowledge pulse is generated for each received frame. 1: No acknowledge pulse is generated.</p>
<b>AIRDIS</b>	<p><b>Auto Interrupt Reset Disable</b></p> <p>0: IRQD is cleared automatically upon a read/write access to ICRTB. (Advantageous if data are read/written via PEC transfers) 1: IRQD must explicitly be cleared via software. (Allows to trigger a stop condition after the last data transfer before the bus is released by clearing IRQD.)</p>
<b>TRX</b>	<p><b>Transmit Select</b></p> <p>0: Data is received from the IIC bus. 1: Data is transmitted to the IIC bus. TRX is set automatically when writing to the transmit buffer ICRTB.</p>

**The IIC Bus Module**

The IIC address register ICADR stores the device address (ICA) which identifies the IIC node when operating in slave mode. Bit M10 in register ICCON determines which part of ICADR is valid and used.

**ICADR**

IIC Address Register						XReg (ED06 <sub>H</sub> )				Reset Value: 0XXX <sub>H</sub>					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	ICA.9...8		ICA.7...1					ICA.0		
-	-	-	-	-	-	rw		rw					rw		

Bit	Function
ICA.7-1	Address in 7-bit mode (ICA.9, ICA.8, ICA.0 disregarded).
ICA.0-9	Address in 10-bit mode (all bits used).

The IIC Receive/Transmit Buffer (ICRTB) accepts bytes to be transmitted and provides received bytes.

**ICRTB**

IIC Receive/Transmit Buffer						XReg (ED08 <sub>H</sub> )				Reset Value: - - XX <sub>H</sub>					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
- reserved -						ICData									
-						rw									

Bit	Function
ICData	<p><b>Transmit and shift data</b></p> <p>This field accepts the byte to be transmitted or provides the received byte.</p> <p><i>Note: A data transfer event interrupt request (IRQD) is cleared automatically when reading from or writing to ICRTB, if bit AIRDIS = '0'.</i></p> <p><i>If AIRDIS = '1' the request flag IRQD must be cleared via software.</i></p>

*Note: It is recommended not to access the receive/transmit buffer while a data transfer is in progress.*

ICST

IIC Status Register

XReg (ED04<sub>H</sub>)

Reset Value: 000X<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	IRQP	IRQD	BB	LRB	SLA	AL	ADR
-	-	-	-	-	-	-	-	-	rwh	rwh	rh	rh	rh	rwh	rh

Bit	Function
<b>ADR</b>	<b>Address</b> Bit ADR is set after a start condition in slave mode until the address has been received (1 byte in 7-bit address mode, 2 bytes in 10-bit address mode).
<b>AL</b>	<b>Arbitration Lost</b> Bit AL is set when the IIC module has tried to become master on the bus but has lost the arbitration. Operation is continued until the 9 <sup>th</sup> clock pulse. Bit IRQP is set along with bit AL. Bit AL must be cleared via software.
<b>SLA</b>	<b>Slave</b> 0: The IIC bus is not busy, or the module is in master mode. 1: The IIC module has been selected as a slave (device addr. received).
<b>LRB</b>	<b>Last Received Bit</b> (undefined after reset) Bit LRB represents the last bit (i.e. the acknowledge bit) of the last transmitted or received frame.
<b>BB</b>	<b>Bus Busy</b> 0: The IIC bus is idle, i.e. a stop condition has occurred. 1: The IIC bus is active, i.e. a start condition has occurred. Bit BB is always '0' while the IIC module is disabled.
<b>IRQD</b>	<b>IIC Interrupt Request Bit for Data Transfer Events<sup>1)</sup></b> 0: No interrupt request pending. 1: A data transfer event interrupt request is pending. IRQD is set after the acknowledge bit of a byte has been received or transmitted, and is cleared automatically upon a read or write access to the buffer ICRTB if bit AIRDIS = '0'. IRQD must be cleared via software if bit AIRDIS = '1'.
<b>IRQP</b>	<b>IIC Interrupt Request Bit for Protocol Events<sup>1)</sup></b> 0: No interrupt request pending. 1: A protocol event interrupt request is pending. IRQP is set when bit SLA or bit AL is set (✓), and must be cleared via software.

- 1) While either IRQD or IRQP is set and the IIC module is in master mode or has been selected as a slave, the IIC clock line is held low which prevents further transfers on the IIC bus.  
The clock line (i.e. the IIC bus) is released when both IRQD **and** IRQP are cleared. Only in this case the next IIC bus action can take place.  
Note that IRQD is cleared automatically upon a read or write access to register ICRTB if bit AIRDIS is not set. Both interrupt request bits may be set or cleared via software, e.g. to control the IIC bus.

## 21.4 IIC Interrupt Control

The bit addressable interrupt control registers XP0IC and XP1IC are assigned to the IIC module. The occurrence of an interrupt request sets the respective interrupt request bit XP0IR/XP1IR. If this interrupt node is enabled (XPxEN = '1') a CPU interrupt is generated and arbitrated. These interrupt requests may be serviced via a standard service routine or with PEC transfers (see below). If polling of bits XP0IR and XP1IR is used please note that these request bits must be cleared via software.

**Data transfer event** interrupts are indicated by bit IRQD and allocated to vector **XP0INT**.

A data transfer event occurs after the acknowledge bit for a byte has been received or transmitted.

**Protocol transfer event** interrupts are indicated by bit IRQP and allocated to vector **XP1INT**. A protocol transfer event occurs when bit SLA is set, i.e. a slave address is received, or when bit AL is set, i.e. the bus arbitration has been lost.

As long as either interrupt request flag (IRQD or IRQP) of the IIC bus module is set the selected clock line(s) SCLx is/are held low. This disables any further transfer on the IIC bus and enables the driver software to react on the recent event. When both request bits are cleared the clock line(s) is/are released again and subsequent bus transfers can take place.

*Note: The interrupt node request bits XP0IR and XP1IR are cleared automatically when the CPU services the respective interrupt (not in case of polling!).*

*The IIC bus module interrupt request bit IRQP must be cleared via the driver software.*

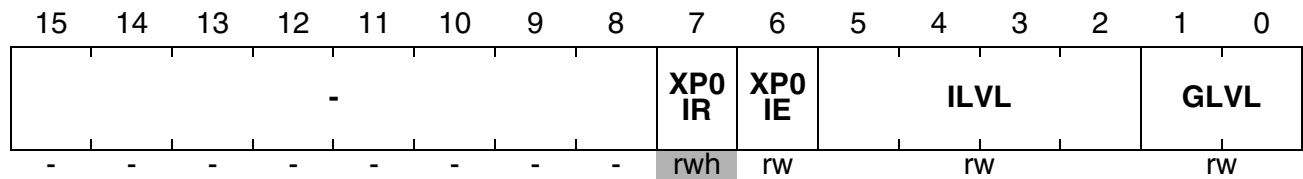
*The IIC bus module interrupt request bit IRQD is cleared automatically upon a read/write access to buffer ICRTB if bit AIRDIS = '0', otherwise it must be cleared via the driver software.*

**XP0IC**

**IIC Data Intr. Ctrl. Reg.**

**ESFR (F186<sub>H</sub>/C3<sub>H</sub>)**

**Reset Value: - - 00<sub>H</sub>**

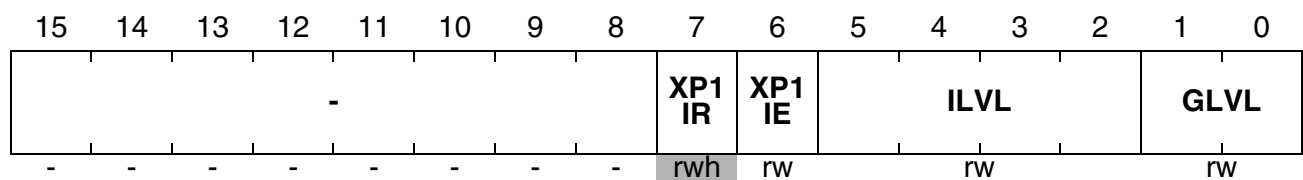


**XP1IC**

**IIC Protocol Intr. Ctrl. Reg.**

**ESFR (F18E<sub>H</sub>/C7<sub>H</sub>)**

**Reset Value: - - 00<sub>H</sub>**



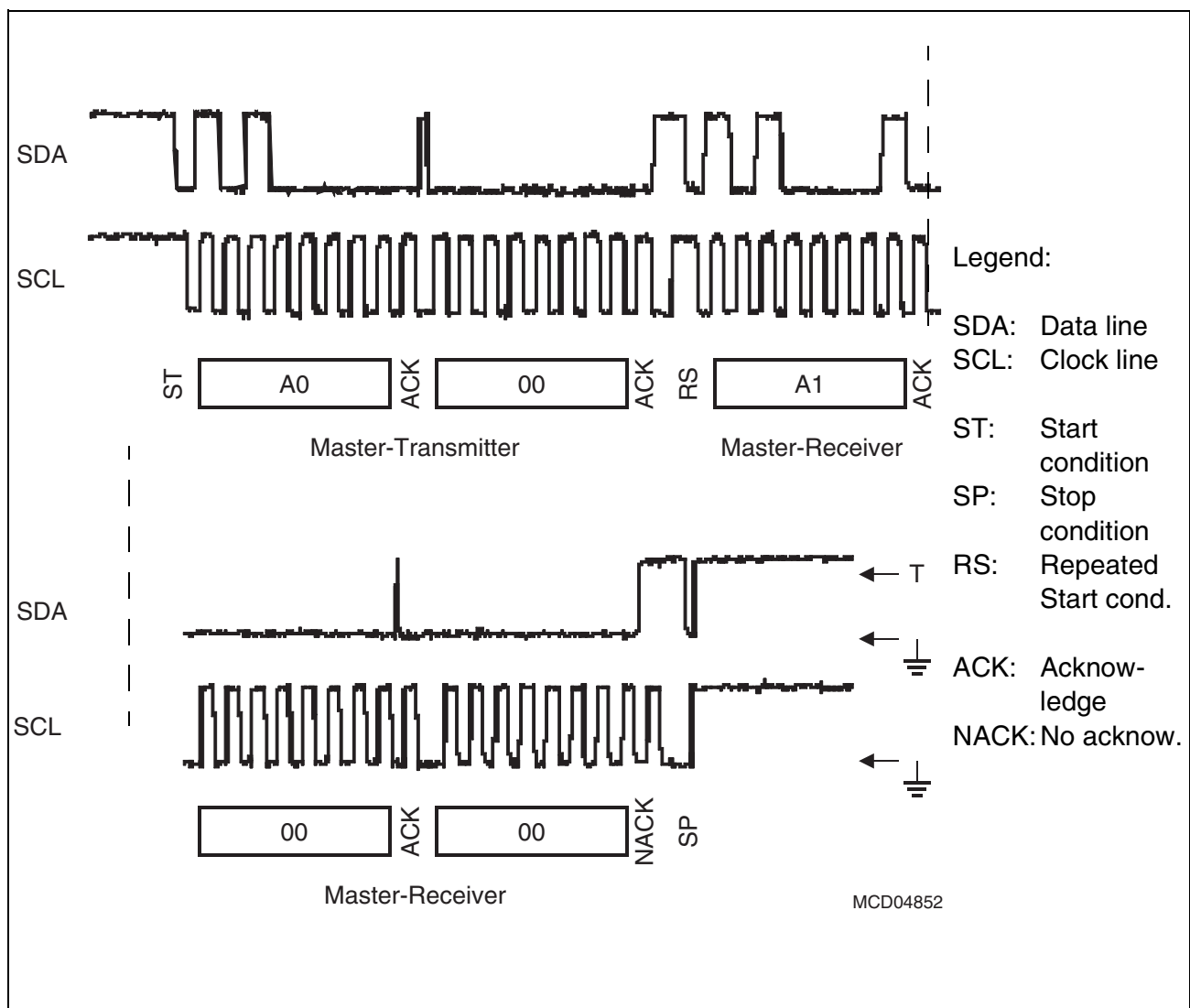
*Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.*

## 21.5 Programming Example

The sample program below illustrates an IIC communication between the C161CS/JC/JI and an NVRAM (such as SDA2526 or SLA24C04). It uses 7-bit addressing with a slave address of 50<sub>H</sub> which is concatenated with the Read/Write bit. This program does not use interrupts, but polls the corresponding IIC interrupt request flags.

The master (C161CS/JC/JI) starts in master transmitter mode and first sends the slave address (A0<sub>H</sub> = 50<sub>H</sub>//0<sub>B</sub>) followed by the subaddress (00<sub>H</sub>). The C161CS/JC/JI changes to master receiver mode, repeats the slave address (A1<sub>H</sub> = 50<sub>H</sub>//1<sub>B</sub>) and then receives two bytes. The first byte is acknowledged (ACK = '0') by the master, the second byte is not acknowledged (ACK = '1'). The transfer is finished with a STOP condition by the master.

**Figure 21-5** shows the waveforms for the described transfer. A programming example in “C” illustrates how the operation could be realized.



**Figure 21-5 IIC Bus Programming Example Waveforms**

```

/*-----*\
| Programming example to read 2 bytes from an NVRAM          |
| via the IIC bus.                                          |
\*-----*/

void main() {

// X-peripheral enable:
SYSCON |= 0x0004;          //set XPEN, before EINIT-instr.!!!

// IIC control register configuration:
ICCON = 0x0008;           //master mode
ICST  = 0x0000;           //reset status register
ICCFG = 0x2711;           //100kHz @ 16MHz, SDA0, SCL0
XP0IC = 0x0000;           //disable interrupt IRQD, use polling
XP1IC = 0x0000;           //disable interrupt IRQP, use polling

// Port configuration (provide external pullups on IIC-lines!):
// (The actual physical port depends on the respective device!
// The IIC interface e.g. uses P3 on the C161RI, P9 on the C161SI/
// CI)
_bfld_ (P*,0x0003,0x0003); //enable alternate function on P*.0-1
_bfld_ (DP*,0x0003,0x0003); //switch IIC pins to output

// slave address
ICRTB = 0x0000 | 0xA0;    //write transmit buffer
ICCON |= BUM;             //BUM=1: start cond.+send slave addr.
while((ICST&IRQD)==0x0000); //waiting for end of transmission
if (ICST & LRB)           //ACK?
{
    ICST &= ~AL;          //Clear bit AL
    ICST &= ~IRQP;        //Clear bit IRQP
}

// sub address
ICRTB = 0x0000|0x0000;    //send sub-address
while((ICST&IRQD)==0x0000); //waiting for end of transmission
if (ICST & LRB)           //ACK?
{
    ICST &= ~AL;          //Clear bit AL
    ICST &= ~IRQP;        //Clear bit IRQP
}

```

```
// switch to master-receiver,  
// send slave address with a repeated start:  
ICCON |= RSC;           //repeated start condition  
ICRTB = 0x0000|0xA1;   //write to transmit buffer  
while((ICST&IRQD)==0x0000); //waiting for end of transmission  
if (ICST & LRB)        //ACK?  
{  
    ICST &= ~AL;       //Clear bit AL  
    ICST &= ~IRQP;    //Clear bit IRQP  
}  
  
// drive clock (SCL) for first byte, give ack:  
ICCON &= ~ACKDIS;     //acknowledge from master  
ICCON &= ~TRX;       //TRX = 0 for master receiver  
dummy = ICRTB;       //start clock to receive the 1st byte  
while((ICST&IRQD)==0x0000); //waiting for end of 1st byte  
  
// read first byte, drive clock for second, give no ack:  
ICCON |= ACKDIS;     //no acknowledge from master  
array[0] = ICRTB;    //read ICRTB (1st byte),  
                    // start clock to receive the 2nd byte  
while((ICST&IRQD)==0x0000); //waiting for end of 2nd byte  
  
// read 2nd byte without automatic clear of IRQD, generate STOP:  
ICCON |= AIRDIS;     //AIRDIS=1: read ICRTB, send no clock  
array[1] = ICRTB;    //read ICRTB (2nd byte)  
ICCON &= ~BUM;       //BUM=0: initiate stop condition  
ICST  &= ~IRQD;     //Clear bit IRQD  
  
}
```



## 22 System Reset

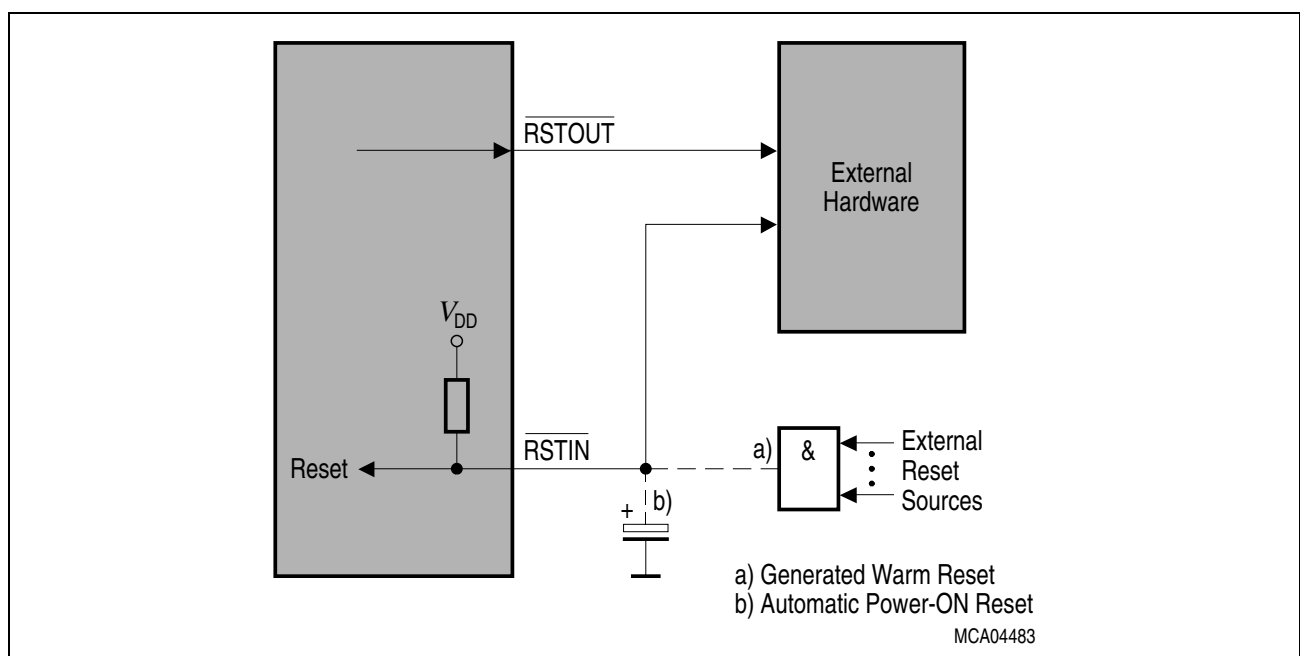
The internal system reset function provides initialization of the C161CS/JC/JI into a defined default state and is invoked either by asserting a hardware reset signal on pin  $\overline{RSTIN}$  (Hardware Reset Input), upon the execution of the SRST instruction (Software Reset) or by an overflow of the watchdog timer.

Whenever one of these conditions occurs, the microcontroller is reset into its predefined default state through an internal reset procedure. When a reset is initiated, pending internal hold states are cancelled and the current internal access cycle (if any) is completed. An external bus cycle is aborted, except for a watchdog reset (see description). After that the bus pin drivers and the IO pin drivers are switched off (tristate).

The internal reset procedure requires 516 CPU clock cycles in order to perform a complete reset sequence. This 516 cycle reset sequence is started upon a watchdog timer overflow, a SRST instruction or when the reset input signal  $\overline{RSTIN}$  is latched low (hardware reset). The internal reset condition is active at least for the duration of the reset sequence and then until the  $\overline{RSTIN}$  input is inactive and the PLL has locked (if the PLL is selected for the basic clock generation). When this internal reset condition is removed (reset sequence complete,  $\overline{RSTIN}$  inactive, PLL locked) the reset configuration is latched from PORT0,  $\overline{RD}$ , and ALE (depending on the start mode). After that pins ALE,  $\overline{RD}$ , and  $\overline{WR}$  are driven to their inactive levels.

*Note: Bit ADP which selects the Adapt mode is latched with the rising edge of  $\overline{RSTIN}$ .*

After the internal reset condition is removed, the microcontroller will either start program execution from external or internal memory, or enter boot mode.



**Figure 22-1 External Reset Circuitry**

## 22.1 Reset Sources

Several sources (external or internal) can generate a reset for the C161CS/JC/JI. Software can identify the respective reset source via the reset source indication flags in register WDTCON. Generally any reset causes the same actions on the C161CS/JC/JI's modules. The differences are described in the following sections.

### Hardware Reset

A hardware reset is triggered when the reset input signal  $\overline{\text{RSTIN}}$  is latched low. To ensure the recognition of the  $\overline{\text{RSTIN}}$  signal (latching), it must be held low for at least 100 ns plus 2 CPU clock cycles (input filter plus synchronization). Also shorter  $\overline{\text{RSTIN}}$  pulses may trigger a hardware reset, if they coincide with the latch's sample point. The actual minimum duration for a reset pulse depends on the current CPU clock generation mode. The worstcase is generating the CPU clock via the SlowDown Divider using the maximum factor while the configured basic mode uses the prescaler ( $f_{\text{CPU}} = f_{\text{OSC}} / 64$  in this case).

After the reset sequence has been completed, the  $\overline{\text{RSTIN}}$  input is sampled again. When the reset input signal is inactive at that time, the internal reset condition is terminated (indicated as short hardware reset, SHWR). When the reset input signal is still active at that time, the internal reset condition is prolonged until  $\overline{\text{RSTIN}}$  gets inactive (indicated as long hardware reset, LHWR).

During a hardware reset the inputs for the reset configuration (PORT0,  $\overline{\text{RD}}$ , ALE) need some time to settle on the required levels, especially if the hardware reset aborts a read operation from an external peripheral. During this settling time the configuration may intermittently be wrong. For the duration of one internal reset sequence after a reset has been recognized the configuration latches are not transparent, i.e. the (new) configuration becomes valid earliest after the completion of one reset sequence. This usually covers the required settling time.

When the basic clock is generated by the PLL the internal reset condition is automatically extended until the on-chip PLL has locked.

The input  $\overline{\text{RSTIN}}$  provides an internal pullup device equalling a resistor of 50 k $\Omega$  to 250 k $\Omega$  (the minimum reset time must be determined by the lowest value). Simply connecting an external capacitor is sufficient for an automatic power-on reset (see *b*) in [Figure 22-1](#)).  $\overline{\text{RSTIN}}$  may also be connected to the output of other logic gates (see *a*) in [Figure 22-1](#)). See also "[Bidirectional Reset](#)" on [Page 22-4](#) in this case.

*Note: A power-on reset requires an active time of two reset sequences (1036 CPU clock cycles) after a stable clock signal is available (about 10 ... 50 ms, depending on the oscillator frequency, to allow the on-chip oscillator to stabilize).*

## Software Reset

The reset sequence can be triggered at any time via the protected instruction SRST (Software Reset). This instruction can be executed deliberately within a program, e.g. to leave bootstrap loader mode, or upon a hardware trap that reveals a system failure.

*Note: A software reset only latches the configuration of the bus interface (SALSEL, CSSEL, WRC, BUSTYP) from PORT0 in case of an external reset.  
If bidirectional reset is enabled, a software reset is executed like a long hardware reset.*

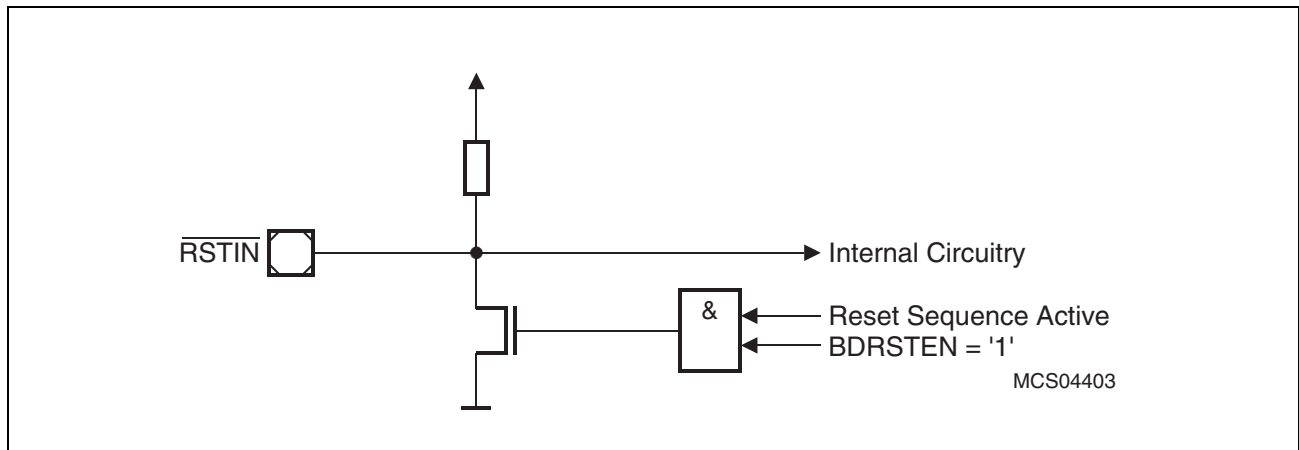
## Watchdog Timer Reset

When the watchdog timer is not disabled during the initialization or serviced regularly during program execution it will overflow and trigger the reset sequence. Other than hardware and software reset the watchdog reset completes a running external bus cycle if this bus cycle either does not use  $\overline{\text{READY}}$  at all, or if  $\overline{\text{READY}}$  is sampled active (low) after the programmed waitstates. When  $\overline{\text{READY}}$  is sampled inactive (high) after the programmed waitstates the running external bus cycle is aborted. Then the internal reset sequence is started.

*Note: A watchdog reset only latches the configuration of the bus interface (SALSEL, CSSEL, WRC, BUSTYP) from PORT0 in case of an external reset.  
If bidirectional reset is enabled a watchdog timer reset is executed like a long hardware reset.  
The watchdog reset cannot occur while the C161CS/JC/JI is in bootstrap loader mode!*

## Bidirectional Reset

In a special mode (bidirectional reset) the C161CS/JC/JI's line  $\overline{\text{RSTIN}}$  (normally an input) may be driven active by the chip logic e.g. in order to support external equipment which is required for startup (e.g. flash memory).



**Figure 22-2 Bidirectional Reset Operation**

Bidirectional reset reflects internal reset sources (software, watchdog) also to the  $\overline{\text{RSTIN}}$  pin and converts short hardware reset pulses to a minimum duration of the internal reset sequence. Bidirectional reset is enabled by setting bit BDRSTEN in register SYSCON and changes  $\overline{\text{RSTIN}}$  from a pure input to an open drain IO line. When an internal reset is triggered by the SRST instruction or by a watchdog timer overflow or a low level is applied to the  $\overline{\text{RSTIN}}$  line, an internal driver pulls it low for the duration of the internal reset sequence. After that it is released and is then controlled by the external circuitry alone.

The bidirectional reset function is useful in applications where external devices require a defined reset signal but cannot be connected to the C161CS/JC/JI's  $\overline{\text{RSTOUT}}$  signal, e.g. an external flash memory which must come out of reset and deliver code well before  $\overline{\text{RSTOUT}}$  can be deactivated via EINIT.

The following behavior differences must be observed when using the bidirectional reset feature in an application:

- Bit BDRSTEN in register SYSCON cannot be changed after EINIT.
- After a reset bit BDRSTEN is cleared.
- The reset indication flags always indicate a long hardware reset.
- The PORT0 configuration is treated like on a hardware reset. Especially the bootstrap loader may be activated when P0L.4 or  $\overline{\text{RD}}$  is low.
- Pin  $\overline{\text{RSTIN}}$  may only be connected to external reset devices with an open drain output driver.
- A short hardware reset is extended to the duration of the internal reset sequence.

## 22.2 Status After Reset

After a reset is completed most units of the C161CS/JC/JI enter a well-defined default status. This ensures repeatable start conditions and avoids spurious activities after reset.

### Watchdog Timer Operation after Reset

The watchdog timer starts running after the internal reset has completed. It will be clocked with the internal system clock divided by 2 ( $f_{CPU} / 2$ ), and its default reload value is 00<sub>H</sub>, so a watchdog timer overflow will occur 131,072 CPU clock cycles ( $2 \times 2^{16}$ ) after completion of the internal reset, unless it is disabled, serviced or reprogrammed meanwhile. When the system reset was caused by a watchdog timer overflow, the WDTR (Watchdog Timer Reset Indication) flag in register WDTCON will be set to '1'. This indicates the cause of the internal reset to the software initialization routine. WDTR is reset to '0' by an external hardware reset, by servicing the watchdog timer or after EINIT. After the internal reset has completed, the operation of the watchdog timer can be disabled by the DISWDT (Disable Watchdog Timer) instruction. This instruction has been implemented as a protected instruction. For further security, its execution is only enabled in the time period after a reset until either the SRVWDT (Service Watchdog Timer) or the EINIT instruction has been executed. Thereafter the DISWDT instruction will have no effect.

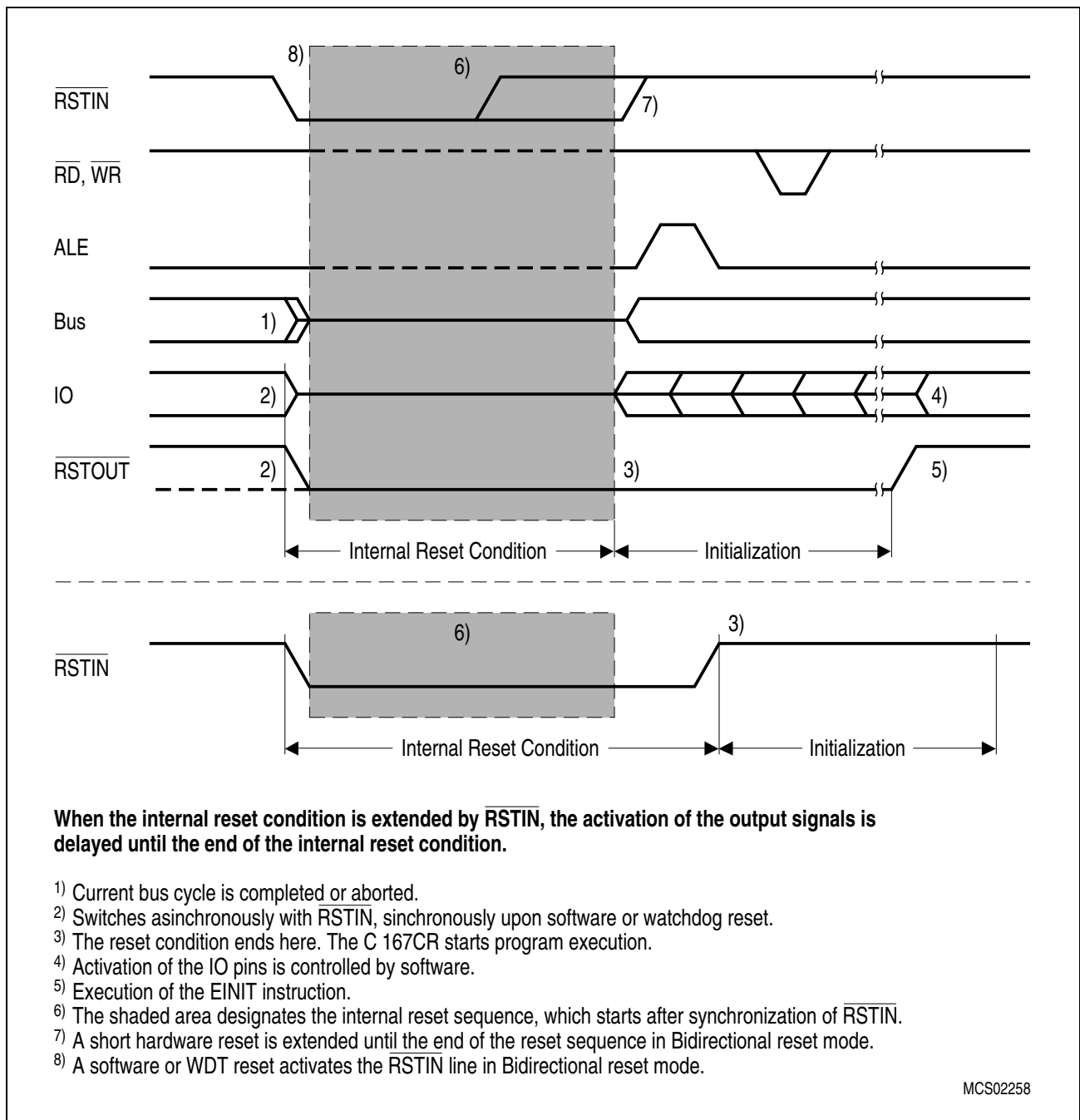
### Reset Values for the C161CS/JC/JI Registers

During the reset sequence the registers of the C161CS/JC/JI are preset with a default value. Most SFRs, including system registers and peripheral control and data registers, are cleared to zero, so all peripherals and the interrupt system are off or idle after reset. A few exceptions to this rule provide a first pre-initialization, which is either fixed or controlled by input pins.

DPP1:	0001 <sub>H</sub> (points to data page 1)
DPP2:	0002 <sub>H</sub> (points to data page 2)
DPP3:	0003 <sub>H</sub> (points to data page 3)
CP:	FC00 <sub>H</sub>
STKUN:	FC00 <sub>H</sub>
STKOV:	FA00 <sub>H</sub>
SP:	FC00 <sub>H</sub>
WDTCON:	00XX <sub>H</sub> (value depends on the reset source)
S0RBUF:	XX <sub>H</sub> (undefined)
SSCRB:	XXXX <sub>H</sub> (undefined)
SYSCON:	0XX0 <sub>H</sub> (set according to reset configuration)
BUSCON0:	0XX0 <sub>H</sub> (set according to reset configuration)
RP0H:	XX <sub>H</sub> (reset levels of P0H)
ONES:	FFFF <sub>H</sub> (fixed value)

### The C161CS/JC/JI's Pins after Reset

After the reset sequence the different groups of pins of the C161CS/JC/JI are activated in different ways depending on their function. Bus and control signals are activated immediately after the reset sequence according to the configuration latched from PORT0, so either external accesses can take place or the external control signals are inactive. The general purpose IO pins remain in input mode (high impedance) until reprogrammed via software (see [Figure 22-3](#)). The  $\overline{\text{RSTOUT}}$  pin remains active (low) until the end of the initialization routine (see description).



**Figure 22-3 Reset Input and Output Signals**

### Ports and External Bus Configuration during Reset

During the internal reset sequence all of the C161CS/JC/JI's port pins are configured as inputs by clearing the associated direction registers, and their pin drivers are switched to the high impedance state. This ensures that the C161CS/JC/JI and external devices will not try to drive the same pin to different levels. Pin ALE is held low through an internal pulldown, and pins  $\overline{RD}$ ,  $\overline{WR}$  and  $\overline{READY}$  are held high through internal pullups. Also the pins that can be configured for  $\overline{CS}$  output will be pulled high.

The registers SYSCON and BUSCON0 are initialized according to the configuration selected via PORT0.

When an external start is selected (pin  $\overline{EA} = '0'$ ):

- the Bus Type field (BTYP) in register BUSCON0 is initialized according to P0L.7 and P0L.6
- bit BUSACT0 in register BUSCON0 is set to '1'
- bit ALECTL0 in register BUSCON0 is set to '1'
- bit ROMEN in register SYSCON will be cleared to '0'
- bit BYTDIS in register SYSCON is set according to the data bus width (set if 8-bit)
- bit WRCFG in register SYSCON is set according to pin P0H.0 (WRC)

When an internal start is selected (pin  $\overline{EA} = '1'$ ):

- register BUSCON0 is initialized to 00C0<sub>H</sub>
- bit ROMEN in register SYSCON will be set to '1'
- bit BYTDIS in register SYSCON is set, i.e.  $\overline{BHE}/\overline{WRH}$  is disabled
- bit WRCFG in register SYSCON is set according to pin P0H.0 (WRC)

The other bits of register BUSCON0, and the other BUSCON registers are cleared. This default initialization selects the slowest possible external accesses using the configured bus type.

When the internal reset has completed, the configuration of PORT0, PORT1, Port 4, Port 6, and of the  $\overline{BHE}$  signal (High Byte Enable, alternate function of P3.12) depends on the bus type which was selected during reset. When any of the external bus modes was selected during reset, PORT0 will operate in the selected bus mode. Port 4 will output the selected number of segment address lines (all zero after reset). Port 6 will drive the selected number of  $\overline{CS}$  lines ( $\overline{CS0}$  will be '0', while the other active  $\overline{CS}$  lines will be '1'). When no memory accesses above 64 K are to be performed, segmentation may be disabled.

When the on-chip bootstrap loader was activated during reset, pin TxD0 (alternate port function) will be switched to output mode after the reception of the zero byte.

All other pins remain in the high-impedance state until they are changed by software or peripheral operation.



## Reset Output Pin

The  $\overline{\text{RSTOUT}}$  pin is dedicated to generate a reset signal for the system components besides the controller itself.  $\overline{\text{RSTOUT}}$  will be driven active (low) at the begin of any reset sequence (triggered by hardware, the SRST instruction or a watchdog timer overflow).  $\overline{\text{RSTOUT}}$  stays active (low) beyond the end of the internal reset sequence until the protected EINIT (End of Initialization) instruction is executed (see [Figure 22-3](#)). This allows the complete configuration of the controller including its on-chip peripheral units before releasing the reset signal for the external peripherals of the system.

*Note:  $\overline{\text{RSTOUT}}$  will float during emulation mode or adapt mode.*

## The Internal RAM after Reset

The contents of the internal RAM are not affected by a system reset. However, after a power-on reset, the contents of the internal RAM are undefined. This implies that the GPRs (R15 ... R0) and the PEC source and destination pointers (SRCP7 ... SRCP0, DSTP7 ... DSTP0) which are mapped into the internal RAM are also unchanged after a warm reset, software reset or watchdog reset, but are undefined after a power-on reset.

## The Extension RAM (XRAM) after Reset

The contents of the on-chip extension RAM are not affected by a system reset. However, after a power-on reset, the contents of the XRAM are undefined.

## Operation after Reset

After the internal reset condition is removed the C161CS/JC/JI fetches the first instruction from the program memory (location 00'0000<sub>H</sub> for a standard start). As a rule, this first location holds a branch instruction to the actual initialization routine that may be located anywhere in the address space.

*Note: When the Bootstrap Loader Mode was activated during a hardware reset the C161CS/JC/JI does not fetch instructions from the program memory.*

*The standard bootstrap loader expects data via serial interface ASC0.*



## 22.3 Application-Specific Initialization Routine

After a reset the modules of the C161CS/JC/JI must be initialized to enable their operation on a given application. This initialization depends on the task the C161CS/JC/JI is to fulfill in that application and on some system properties like operating frequency, connected external circuitry, etc.

The following initializations should typically be done, before the C161CS/JC/JI is prepared to run the actual application software:

### Memory Areas

**The external bus interface** can be reconfigured after an external reset because register BUSCON0 is initialized to the slowest possible bus cycle configuration. The programmable address windows can be enabled in order to adapt the bus cycle characteristics to different memory areas or peripherals. Also after a single-chip mode reset the external bus interface can be enabled and configured.

**The internal program memory** (if available) can be enabled and mapped after an external reset in order to use the on-chip resources. After a single-chip mode reset the internal program memory can be remapped or disabled at all in order to utilize external memory (partly or completely).

Programmable program memory can be programmed, e.g. with data received over a serial link.

*Note: Initial Flash or OTP programming will rather be done in bootstrap loader mode.*

### System Stack

The default setup for the system stack (size, stackpointer, upper and lower limit registers) can be adjusted to application-specific values. After reset, registers SP and STKUN contain the same reset value 00'FC00<sub>H</sub>, while register STKOV contains 00'FA00<sub>H</sub>. With the default reset initialization, 256 words of system stack are available, where the system stack selected by the SP grows downwards from 00'FBFE<sub>H</sub>.

*Note: The interrupt system, which is disabled upon completion of the internal reset, should remain disabled until the SP is initialized.*

*Traps (incl.  $\overline{NMI}$ ) may occur, even though the interrupt system is still disabled.*

### Register Bank

The location of a register bank is defined by the context pointer (CP) and can be adjusted to an application-specific bank before the general purpose registers (GPRs) are used. After reset, register CP contains the value 00'FC00<sub>H</sub>, i.e. the register bank selected by the CP grows upwards from 00'FC00<sub>H</sub>.

## **On-Chip RAM**

Based on the application, the user may wish to initialize portions of the internal writable memory (IRAM/XRAM) before normal program operation. Once the register bank has been selected by programming the CP register, the desired portions of the internal memory can easily be initialized via indirect addressing.

## **Interrupt System**

After reset the individual interrupt nodes and the global interrupt system are disabled. In order to enable interrupt requests the nodes must be assigned to their respective interrupt priority levels and be enabled. The vector locations must receive pointers to the respective exception handlers. The interrupt system must globally be enabled by setting bit IEN in register PSW. Care must be taken not to enable the interrupt system before the initialization is complete in order to avoid e.g. the corruption of internal memory locations by stack operations using an uninitialized stack pointer.

## **Watchdog Timer**

After reset the watchdog timer is active and is counting its default period. If the watchdog timer shall remain active the desired period should be programmed by selecting the appropriate prescaler value and reload value. Otherwise the watchdog timer must be disabled before EINIT.

## **Ports**

Generally all ports of the C161CS/JC/JI are switched to input after reset. Some pins may be automatically controlled, e.g. bus interface pins for an external start, TxD in Boot mode, etc. Pins that shall be used for general purpose IO must be initialized via software. The required mode (input/output, open drain/push pull, input threshold, etc.) depends on the intended function for a given pin.

## **Peripherals**

After reset the C161CS/JC/JI's on-chip peripheral modules enter a defined default state (see respective peripheral description) where it is disabled from operation. In order to use a certain peripheral it must be initialized according to its intended operation in the application.

This includes selecting the operating mode (e.g. counter/timer), operating parameters (e.g. baudrate), enabling interface pins (if required), assigning interrupt nodes to the respective priority levels, etc.

After these standard initialization also application-specific actions may be required like asserting certain levels to output pins, sending codes via interfaces, latching input levels, etc.

### Termination of Initialization

The software initialization routine should be terminated with the EINIT instruction. This instruction has been implemented as a protected instruction.

The execution of the EINIT instruction:

- disables the action of the DISWDT instruction,
- disables write accesses to reg. SYSCON (all configurations regarding reg. SYSCON (enable CLKOUT, stacksize, etc.) must be selected before the execution of EINIT),
- disables write accesses to registers SYSCON2 and SYSCON3 (further write accesses to SYSCON2 and SYSCON3 can be executed only using a special unlock mechanism),
- clears the reset source detection bits in register WDTCON,
- causes the  $\overline{\text{RSTOUT}}$  pin to go high (this signal can be used to indicate the end of the initialization routine and the proper operation of the microcontroller to external hardware).

## **22.4 System Startup Configuration**

Although most of the programmable features of the C161CS/JC/JI are selected by software either during the initialization phase or repeatedly during program execution, there are some features that must be selected earlier, because they are used for the first access of the program execution (e.g. internal or external start selected via  $\overline{EA}$ ).

These configurations are accomplished by latching the logic levels at a number of pins at the end of the internal reset sequence. During reset internal pullup/pulldown devices are active on those lines. They ensure inactive/default levels at pins which are not driven externally. External pulldown/pullup devices may override the default levels in order to select a specific configuration. Many configurations can therefore be coded with a minimum of external circuitry.

*Note: The load on those pins that shall be latched for configuration must be small enough for the internal pullup/pulldown device to sustain the default level, or external pullup/pulldown devices must ensure this level.*

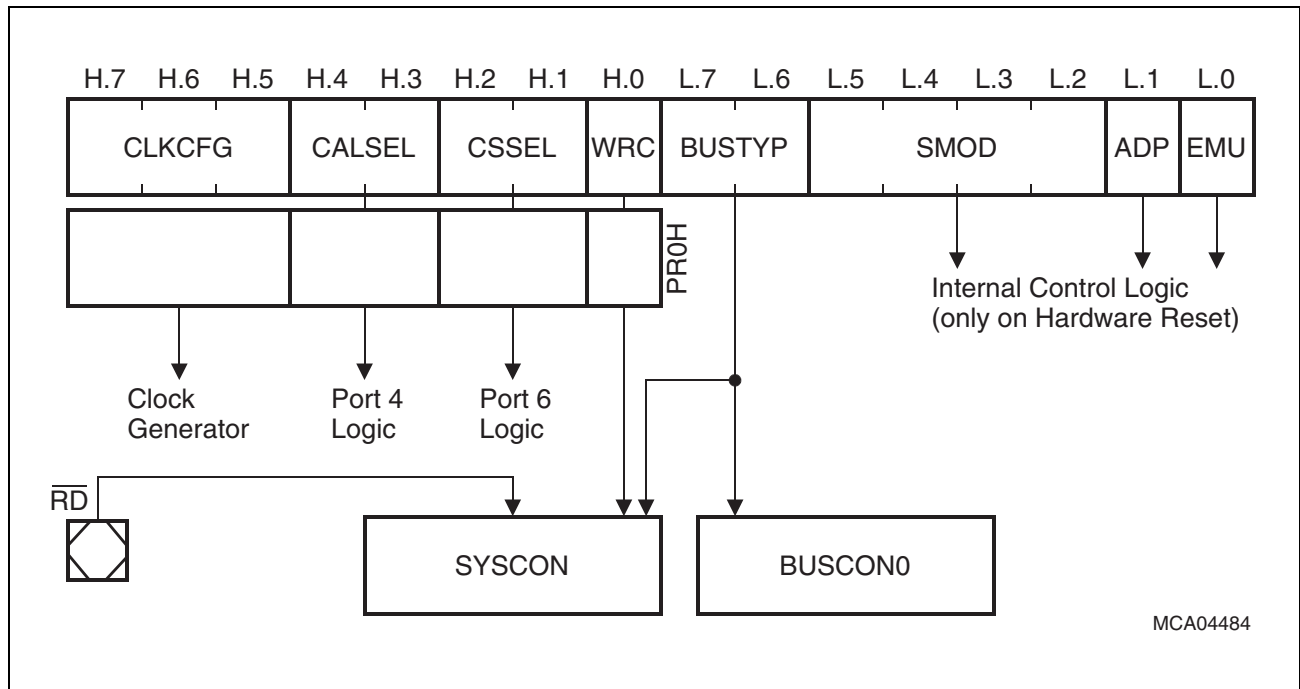
*Those pins whose default level shall be overridden must be pulled low/high externally.*

*Make sure that the valid target levels are reached until the end of the reset sequence.*

*There is a specific application note to illustrate this.*

### 22.4.1 System Startup Configuration upon an External Reset

For an external reset ( $\overline{EA} = '0'$ ) the startup configuration uses the pins of PORT0 and pin  $\overline{RD}$ . The value on the upper byte of PORT0 (P0H) is latched into register RP0H upon reset, the value on the lower byte (P0L) directly influences the BUSCON0 register (bus mode) or the internal control logic of the C161CS/JC/JI.



**Figure 22-4 PORT0 Configuration during Reset**

The pins that control the operation of the internal control logic, the clock configuration, and the reserved pins are evaluated only during a hardware triggered reset sequence. The pins that influence the configuration of the C161CS/JC/JI are evaluated during any reset sequence, i.e. also during software and watchdog timer triggered resets.

The configuration via P0H is latched in register RP0H for subsequent evaluation by software. Register RP0H is described in [Chapter 9](#).

The following describes the different selections that are offered for reset configuration. The default modes refer to pins at high level, i.e. without external pulldown devices connected.

Please also consider the note above.

## Emulation Mode

Pin P0L.0 (EMU) selects the Emulation Mode, when latched low at the end of reset. This mode is used for special emulation and testing purposes and is of minor use for standard C161CS/JC/JI applications, so P0L.0 should be held high.

Emulation mode provides access to integrated XBUS peripherals via the external bus interface pins (direction reversed) of the C161CS/JC/JI. The CPU and the generic peripherals are disabled, all modules connected via the XBUS are active.

**Table 22-1 Emulation Mode Summary**

Pin(s)	Function	Notes
Port 4, PORT1	Address input	The segment address lines configured at reset must be driven externally
PORT0	Data input/output	–
$\overline{RD}$ , $\overline{WR}$	Control signal input	–
ALE	Unused input	Hold LOW
CLKOUT	CPU clock output	Enabled automatically
$\overline{RSTOUT}$	Reset input	Drive externally for an XBUS peripheral reset
$\overline{RSTIN}$	Reset input	Standard reset for complete device
Port 6	Interrupt output	Sends XBUS peripheral interrupt request e.g. to the emulation system

**Default:** Emulation Mode is off.

*Note: In emulation mode pin P0.15 (P0H.7) is inverted, i.e. the configuration ‘111’ would select direct drive in emulation mode.*

*Emulation mode can only be activated upon an external reset ( $\overline{EA} = '0'$ ). Pin P0L.0 is not evaluated upon a single-chip reset ( $\overline{EA} = '1'$ ).*

**Adapt Mode**

Pin P0L.1 (ADP) selects the Adapt Mode, when latched low at the end of reset. In this mode the C161CS/JC/JI goes into a passive state, which is similar to its state during reset. The pins of the C161CS/JC/JI float to tristate or are deactivated via internal pullup/pulldown devices, as described for the reset state. In addition also the  $\overline{\text{RSTOUT}}$  pin floats to tristate rather than be driven low. The on-chip oscillator and the realtime clock are disabled.

This mode allows switching a C161CS/JC/JI that is mounted to a board virtually off, so an emulator may control the board's circuitry, even though the original C161CS/JC/JI remains in its place. The original C161CS/JC/JI also may resume to control the board after a reset sequence with P0L.1 high. Please note that adapt mode overrides any other configuration via PORT0.

**Default:** Adapt Mode is off.

*Note: When XTAL1 is fed by an external clock generator (while XTAL2 is left open), this clock signal may also be used to drive the emulator device.*

*However, if a crystal is used, the emulator device's oscillator can use this crystal only, if at least XTAL2 of the original device is disconnected from the circuitry (the output XTAL2 will be driven high in Adapt Mode).*

*Adapt mode can only be activated upon an external reset ( $\overline{\text{EA}} = '0'$ ). Pin P0L.1 is not evaluated upon a single-chip reset ( $\overline{\text{EA}} = '1'$ ).*

### Special Operation Modes

Pins P0L.5 to P0L.2 (SMOD) select special operation modes of the C161CS/JC/JI during reset (see [Table 22-2](#)). Make sure to only select valid configurations in order to ensure proper operation of the C161CS/JC/JI.

**Table 22-2 Definition of Special Modes for Reset Configuration**

P0.5-2 (P0L.5-2)	Special Mode	Notes
1 1 1 1	Normal Start	Default configuration. Begin of execution as defined via pin $\overline{EA}$ .
1 1 1 0	Reserved	Do not select this configuration!
1 1 0 1	Reserved	Do not select this configuration!
1 1 0 0	Reserved	Do not select this configuration!
1 0 1 1	Standard Bootstrap Loader	Load an initial boot routine of 32 Bytes via interface ASC0.
1 0 1 0	Reserved	Do not select this configuration!
1 0 0 1	Alternate Boot	Operation not yet defined. Do not use!
1 0 0 0	Reserved	Do not select this configuration!
0 1 1 1	No emulation mode: Alternate Start	Operation not yet defined. Do not use!
	Emulation mode: External Host Mode (EHM)	Programming mode for Flash memory via external host.
0 1 1 0	Reserved	Do not select this configuration!
0 1 0 1	Reserved	Do not select this configuration!
0 1 0 0	Reserved	Do not select this configuration!
0 0 X X	Reserved	Do not select this configuration!

**The On-Chip Bootstrap Loader** allows moving the start code into the internal RAM of the C161CS/JC/JI via the serial interface ASC0. The C161CS/JC/JI will remain in bootstrap loader mode until a hardware reset not selecting BSL mode or a software reset.

**Default:** The C161CS/JC/JI starts fetching code from location  $00'0000_H$ , the bootstrap loader is off.



## External Bus Type

Pins P0L.7 and P0L.6 ( $\overline{\text{BUSTYP}}$ ) select the external bus type during reset, if an external start is selected via pin  $\overline{\text{EA}}$ . This allows the configuration of the external bus interface of the C161CS/JC/JI even for the first code fetch after reset. The two bits are copied into bit field BTYP of register BUSCON0. P0L.7 controls the data bus width, while P0L.6 controls the address output (multiplexed or demultiplexed). This bit field may be changed via software after reset, if required.

**Table 22-3 Configuration of External Bus Type**

P0L.7-6 (BTYP) Encoding	External Data Bus Width	External Address Bus Mode
0 0	8-bit Data	Demultiplexed Addresses
0 1	8-bit Data	Multiplexed Addresses
1 0	16-bit Data	Demultiplexed Addresses
1 1	16-bit Data	Multiplexed Addresses

PORT0 and PORT1 are automatically switched to the selected bus mode. In multiplexed bus modes PORT0 drives both the 16-bit intra-segment address and the output data, while PORT1 remains in high impedance state as long as no demultiplexed bus is selected via one of the BUSCON registers. In demultiplexed bus modes PORT1 drives the 16-bit intra-segment address, while PORT0 or P0L (according to the selected data bus width) drives the output data.

For a 16-bit data bus  $\overline{\text{BHE}}$  is automatically enabled, for an 8-bit data bus  $\overline{\text{BHE}}$  is disabled via bit BYTDIS in register SYSCON.

**Default:** 16-bit data bus with multiplexed addresses.

*Note: If an internal start is selected via pin  $\overline{\text{EA}}$ , these two pins are disregarded and bit field BTYP of register BUSCON0 is cleared.*

## Write Configuration

Pin P0H.0 (WRC) selects the initial operation of the control pins  $\overline{\text{WR}}$  and  $\overline{\text{BHE}}$  during reset. When high, this pin selects the standard function, i.e.  $\overline{\text{WR}}$  control and  $\overline{\text{BHE}}$ . When low, it selects the alternate configuration, i.e.  $\overline{\text{WRH}}$  and  $\overline{\text{WRL}}$ . Thus even the first access after a reset can go to a memory controlled via  $\overline{\text{WRH}}$  and  $\overline{\text{WRL}}$ . This bit is latched in register RP0H and its inverted value is copied into bit WRCFG in register SYSCON.

**Default:** Standard function ( $\overline{\text{WR}}$  control and  $\overline{\text{BHE}}$ ).

## Chip Select Lines

Pins P0H.2 and P0H.1 (CSSEL) define the number of active chip select signals during reset. This allows the selection which pins of Port 6 drive external  $\overline{CS}$  signals and which are used for general purpose IO. The two bits are latched in register RP0H.

**Table 22-4 Configuration of Chip Select Lines**

P0H.2-1 (CSSEL)	Chip Select Lines	Note
1 1	Five: $\overline{CS4} \dots \overline{CS0}$	Default without pull-downs
1 0	None	–
0 1	Two: $\overline{CS1} \dots \overline{CS0}$	–
0 0	Three: $\overline{CS2} \dots \overline{CS0}$	–

**Default:** All 5 chip select lines active ( $\overline{CS4} \dots \overline{CS0}$ ).

*Note:* The selected number of  $\overline{CS}$  signals can be changed via software after reset (see [Section 22.4.2](#)).

## Segment Address Lines

Pins P0H.4 and P0H.3 (SALSEL) define the number of active segment address lines during reset. This allows the selection which pins of Port 4 drive address lines and which are used for general purpose IO. The two bits are latched in register RP0H. Depending on the system architecture the required address space is chosen and accessible right from the start, so the initialization routine can directly access all locations without prior programming. The required pins of Port 4 are automatically switched to address output mode.

**Table 22-5 Configuration of Segment Address Lines**

P0H.4-3 (SALSEL)	Segment Address Lines	Directly Accessible A. Space
1 1	Two: A17 ... A16	256 KByte (Default without pull-downs)
1 0	Eight: A23 ... A16	16 MByte (Maximum)
0 1	None	64 KByte (Minimum)
0 0	Four: A19 ... A16	1 MByte

Even if not all segment address lines are enabled on Port 4, the C161CS/JC/JI internally uses its complete 24-bit addressing mechanism. This allows the restriction of the width of the effective address bus, while still deriving  $\overline{CS}$  signals from the complete addresses.

**Default:** 2-bit segment address (A17 ... A16) allowing access to 256 KByte.

*Note:* The selected number of segment address lines can be changed via software after reset (see [Section 22.4.2](#)).

## Clock Generation Control

Pins P0H.7, P0H.6 and P0H.5 (CLKCFG) select the basic clock generation mode during reset. The oscillator clock either directly feeds the CPU and peripherals (direct drive), it is divided by 2 or it is fed to the on-chip PLL which then provides the CPU clock signal (selectable multiple of the oscillator frequency, i.e. the input frequency). These bits are latched in register RP0H.

**Table 22-6 C161CS/JC/JI Clock Generation Modes**

(P0H.7-5) (CLKCFG)	CPU Frequency $f_{\text{CPU}} = f_{\text{OSC}} \times F$	External Clock Input Range <sup>1)</sup>	Notes
1 1 1	$f_{\text{OSC}} \times 4$	2.5 to 8.25 MHz	Default configuration
1 1 0	$f_{\text{OSC}} \times 3$	3.33 to 11 MHz	–
1 0 1	$f_{\text{OSC}} \times 2$	5 to 16.5 MHz	–
1 0 0	$f_{\text{OSC}} \times 5$	2 to 6.6 MHz	–
0 1 1	$f_{\text{OSC}} \times 1$	1 to 33 MHz	Direct drive <sup>2)</sup>
0 1 0	$f_{\text{OSC}} \times 1.5$	6.66 to 22 MHz	–
0 0 1	$f_{\text{OSC}} / 2$	2 to 66 MHz	CPU clock via prescaler
0 0 0	$f_{\text{OSC}} \times 2.5$	4 to 13.2 MHz	–

<sup>1)</sup> The external clock input range refers to a CPU clock range of 10 ... 33 MHz.

<sup>2)</sup> The maximum frequency depends on the duty cycle of the external clock signal.

In emulation mode pin P0.15 (P0H.7) is inverted, i.e. the configuration '111' would select direct drive in emulation mode.

**Default:** On-chip PLL is active with a factor of 1:4.

Watch the different requirements for frequency and duty cycle of the oscillator input clock for the possible selections.

## Oscillator Watchdog Control

The on-chip oscillator watchdog (OWD) may be disabled via hardware by (externally) pulling the  $\overline{\text{RD}}$  line low upon a reset, similar to the standard reset configuration via PORT0. At the end of any reset bit OWDDIS in register SYSCON reflects the inverted level of pin  $\overline{\text{RD}}$  at that time. The software may again enable the oscillator watchdog by clearing bit OWDDIS before the execution of EINIT.

*Note: If direct drive or prescaler operation is selected as basic clock generation mode (see above) the PLL is switched off whenever bit OWDDIS is set (via software or via hardware configuration).*

## 22.4.2 System Startup Configuration upon a Single-Chip Mode Reset

For a single-chip mode reset (indicated by  $\overline{EA} = '1'$ ) the configuration via PORT0 is replaced by a fixed configuration value. In this case PORT0 needs no external circuitry (pullups/pulldowns) and also the internal configuration pullups are not activated. The necessary startup modes are configured via pins  $\overline{RD}$  and ALE.

This fixed default configuration is activated after each Long Hardware Reset and selects a safe worst-case configuration. The initialization software can then modify these parameters and select the intended configuration for a given application. [Table 22-7](#) lists the respective default configuration values which are selected, and the bitfields that permit software modification.

**Table 22-7 Default Configuration for Single-Chip Mode Reset**

Configuration Parameter	Default Values (RP0H = XX2DH)	External Config. <sup>1)</sup>	Software Access <sup>2)</sup>
<b>CLKCFG:</b> Generation mode of basic clock	'001' = Prescaler operation, i.e. $f_{CPU} = f_{OSC} / 2$	P0.15-13	RSTCON.15-13
<b>SALSEL:</b> Number of active segment address lines	'01' = No segment address lines	P0.12-11	RSTCON.12-11
<b>CSSEL:</b> Number of active $\overline{CS}$ lines	'10' = No chip select lines	P0.10-9	RSTCON.10-9
<b>WRC:</b> Write signal encoding	RP0H.0 = '1', SYSCON.WRCFG = '0', i.e. $\overline{WR}$ and $\overline{BHE}$	P0.8	SYSCON.WRCFG
<b>BTYP:</b> Default bustype (BUSCON0)	BUSCON0.BTYP = '11' i.e. 16-bit MUX bus	P0.7-6	BUSCON0.BTYP
<b>SMOD:</b> Special modes (start/boot modes)	Startup modes selected via pins $\overline{RD}$ and ALE	P0.5-2	–
<b>ADP:</b> Adapt mode	Not possible	P0.1	–
<b>EMU:</b> Emulation mode	Not possible	P0.0	–
<b>OWD</b> disable	SYSCON.OWDDIS = '0' i.e. OWD is active	$\overline{RD}$	SYSCON.OWDDIS

<sup>1)</sup> Refers to the configuration pins which are replaced by the default values.

<sup>2)</sup> Software can modify the default values via these bitfields.

*Note: Single-chip mode reset cannot be selected on ROMless devices. The attempt to read the first instruction after reset will fail in such a case.*

### Single-Chip Startup Modes

The startup mode (operation after reset) of the C161CS/JC/JI can be configured during reset. In single-chip mode this configuration is selected via pins  $\overline{RD}$  and ALE.

Pin  $\overline{RD}$  selects start or boot mode (instead of OWD control), pin ALE selects one of two alternatives in each case.

**Table 22-8 Startup Mode Configuration in Single-Chip Reset Mode**

$\overline{RD}$	ALE	Startup Mode	Notes
1	0	Standard Start	Execution starts at user memory location 00'0000 <sub>H</sub> .
1	1	Alternate Start	Operation not yet defined. Do not use!
0	0	Standard Bootstrap Loader	Load 32 bytes via ASC0.
0	1	Alternate Boot Mode	Operation not yet defined. Do not use!

## 22.5 System Configuration via Software

The system configuration which is selected via hardware after reset (latched pin levels or default value) can be changed via software by executing a specific code sequence. The respective control bits are located within registers SYSCON, BUSCONx, and RSTCON. Register SYSCON can only be modified before the execution of instruction EINIT, while registers BUSCONx and RSTCON (using the specific sequence) can be modified repeatedly at any time.

The clock generation mode (CLKCFG), the segment address width (SALSEL), and the number of chip select lines (CSSEL) are controlled by register RP0H. RP0H is initialized according to the selected reset mode (pins or default). The respective configuration bitfields can be copied from register RSTCON upon entering Slow Down Divider mode if enabled by bit SUE = '1'.

The following steps must be taken to change the current configuration (see also SW example):

- Write intended configuration value to RSTCON
- Enter SDD mode
- Return to basic clock mode

```
CHANGE_CLOCK_CONFIGURATION: ;"RSTCON" is a mem address, no SFR
MOV R15, #11100001xxxxxxxxxB ;Load a GPR with the target value
MOV RSTCON, R15 ;Enable update with PLL factor 4
EXTR #2 ;ESFR-access to SYSCON2
MOV SYSCON2, #0100H ;SDD mode, PLL on, factor 1
MOV SYSCON2, #0400H ;RSTCON.15-9 is copied to RP0H.15-9
;Switch to basic clock mode
;System will run on PLL (factor 4)..
;..after PLL has locked
```

*Note: This software example assumes execution before EINIT. Otherwise the unlock sequence has to be executed prior to each access to RSTCON/SYSCON2.*

Entering SDD mode temporarily ensures a correct clock signal synchronization in cases where the clock generation mode (e.g. PLL factor) is changed. If the target basic clock generation mode uses the PLL, the C161CS/JC/JI will run in direct-drive mode until the PLL has locked.

Software modification of system configuration values is protected by the following features:

- SYSCON is locked after EINIT
- RSTCON requires the unlock sequence after EINIT
- Copying RSTCON to RP0H must be explicitly enabled by setting bit SUE

**RSTCON**

Reset Control Register

mem (F1E0<sub>H</sub>/--)

Reset Value: 00XX<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CLKCFG		SALSEL		CSSEL		SUE	-	-	-	-	-	-	-	RSTLEN	
rw		rw		rw		rw	-	-	-	-	-	-	-	rw	

Bit	Function								
<b>RSTLEN</b>	<p><b>Reset Length Control</b> (duration of the next reset sequence to occur)<sup>1)</sup></p> <p>00: 1024 TCL: standard duration, corresponds to all other derivatives without control function</p> <p>01: 2048 TCL: extended duration, may be useful e.g. to provide additional settling for external configuration signals at high CPU clock frequencies</p> <p>10: Reserved</p> <p>11: Reserved</p>								
<b>SUE</b>	<p><b>Software Update Enable</b></p> <p>0: Configuration cannot be changed via software</p> <p>1: Software update of configuration is enabled</p>								
<b>CSSEL</b>	<p><b>Chip Select Line Selection</b> (Number of active <math>\overline{CS}</math> outputs)</p> <p>00: 3 <math>\overline{CS}</math> lines: <math>\overline{CS2}</math> ... <math>\overline{CS0}</math></p> <p>01: 2 <math>\overline{CS}</math> lines: <math>\overline{CS1}</math> ... <math>\overline{CS0}</math></p> <p>10: No <math>\overline{CS}</math> lines at all</p> <p>11: all <math>\overline{CS}</math> lines: <math>\overline{CSx}</math> ... <math>\overline{CS0}</math></p>								
<b>SALSEL</b>	<p><b>Segment Address Line Selection</b> (Number of active seg-addr. outputs)</p> <p>00: 4-bit segment address: A19 ... A16</p> <p>01: No segment address lines at all</p> <p>10: full segment address: Axx ... A16</p> <p>11: 2-bit segment address: A17 ... A16</p>								
<b>CLKCFG</b>	<p><b>Clock Generation Mode Configuration</b></p> <p>These pins define the clock generation mode, i.e. the mechanism how the internal CPU clock is generated from the externally applied (XTAL1) input clock.</p> <table border="0"> <tr> <td>000: PLL (<math>f \times 2.5</math>)</td> <td>100: PLL (<math>f \times 5</math>)</td> </tr> <tr> <td>001: Prescaler (<math>f / 2</math>)</td> <td>101: PLL (<math>f \times 2</math>)</td> </tr> <tr> <td>010: PLL (<math>f \times 1.5</math>)</td> <td>110: PLL (<math>f \times 3</math>)</td> </tr> <tr> <td>011: Direct Drive (<math>f = f</math>)</td> <td>111: PLL (<math>f \times 4</math>)</td> </tr> </table>	000: PLL ( $f \times 2.5$ )	100: PLL ( $f \times 5$ )	001: Prescaler ( $f / 2$ )	101: PLL ( $f \times 2$ )	010: PLL ( $f \times 1.5$ )	110: PLL ( $f \times 3$ )	011: Direct Drive ( $f = f$ )	111: PLL ( $f \times 4$ )
000: PLL ( $f \times 2.5$ )	100: PLL ( $f \times 5$ )								
001: Prescaler ( $f / 2$ )	101: PLL ( $f \times 2$ )								
010: PLL ( $f \times 1.5$ )	110: PLL ( $f \times 3$ )								
011: Direct Drive ( $f = f$ )	111: PLL ( $f \times 4$ )								

<sup>1)</sup> RSTLEN is always valid for the **next** reset sequence. An initial power up reset, however, is expected to last considerably longer than any configurable reset sequence.

*Note: RSTCON is write protected after the execution of EINIT unless it is released via the unlock sequence (see [Section 23.7](#)).*  
*RSTCON can only be accessed via its long (mem) address.*

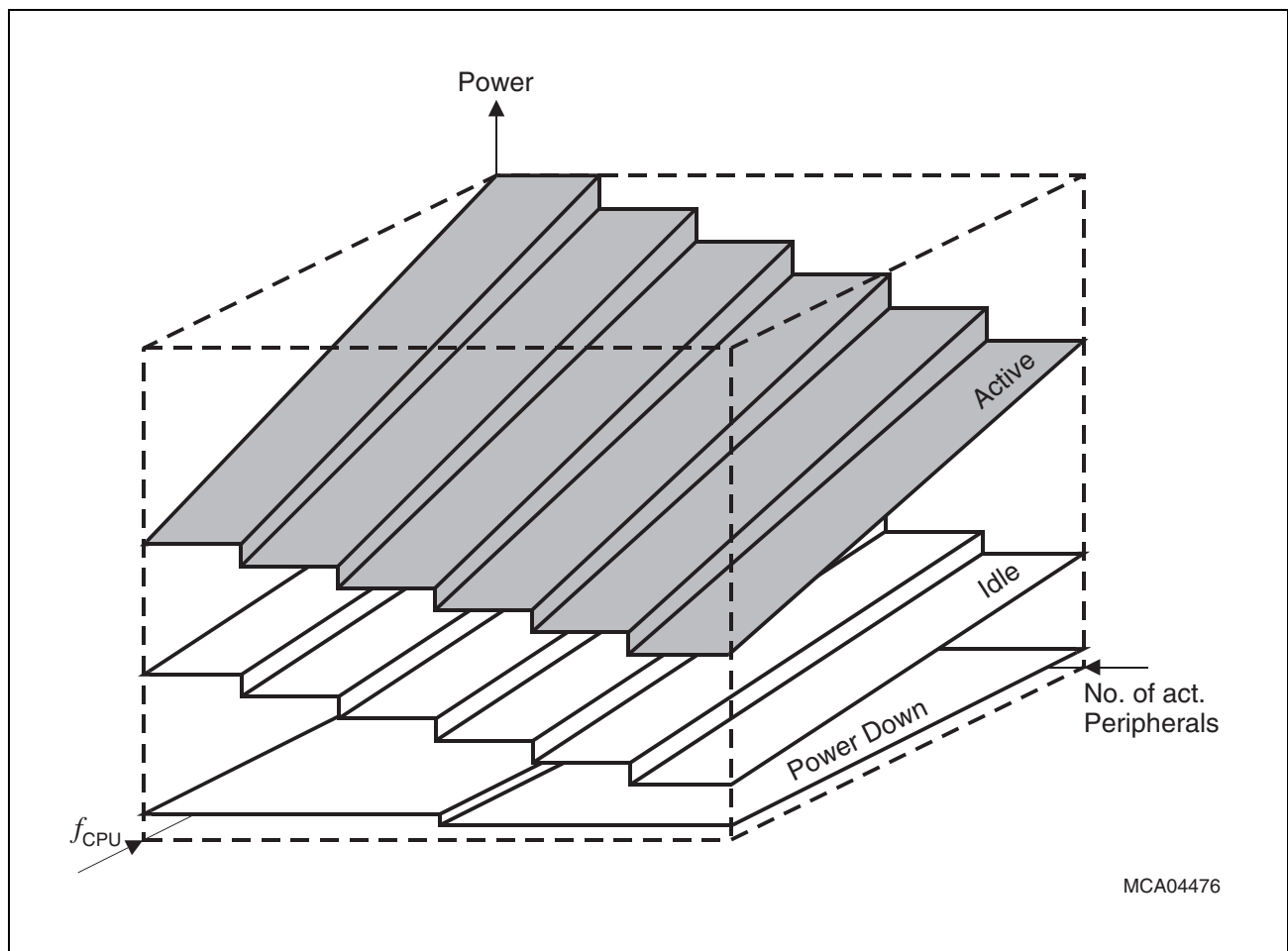


## 23 Power Management

For an increasing number of microcontroller based systems it is an important objective to reduce the power consumption of the system as much as possible. A contradictory objective is, however, to reach a certain level of system performance. Besides optimization of design and technology a microcontroller's power consumption can generally be reduced by lowering its operating frequency and/or by reducing the circuitry that is clocked. The architecture of the C161CS/JC/JI provides three major means of reducing its power consumption (see [Figure 23-1](#)) under software control:

- Reduction of the CPU frequency for Slow Down operation (Flexible Clock Generation Management)
- Selection of the active peripheral modules (Flexible Peripheral Management)
- Special operating modes to deactivate CPU, ports, and control logic (Idle, Sleep, Power Down)

This enables the application (i.e. the programmer) to choose the optimum constellation for each operating condition, so the power consumption can be adapted to conditions like maximum performance, partial performance, intermittent operation or standby.



**Figure 23-1 Power Reduction Possibilities**

**Power Management**

Intermittent operation (i.e. alternating phases of high performance and power saving) is supported by the cyclic interrupt generation mode of the on-chip RTC (real time clock).

These three means described above can be applied independent from each other and thus provide a maximum of flexibility for each application.

For the basic power reduction modes (Idle, Power Down) there are dedicated instructions, while special registers control Sleep mode (SYSCON1), clock generation (SYSCON2), and peripheral management (SYSCON3).

Three different general power reduction modes with different levels of power reduction have been implemented in the C161CS/JC/JI, which may be entered under software control:

In **Idle Mode** the CPU is stopped, while the (enabled) peripherals continue their operation. Idle mode can be terminated by any reset or interrupt request.

In **Sleep Mode** both the CPU and the peripherals are stopped. The real time clock and its selected oscillator may optionally be kept running. Sleep mode can be terminated by any reset or interrupt request (mainly hardware requests, stopped peripherals cannot generate interrupt requests).

In **Power Down Mode** both the CPU and the peripherals are stopped. The real time clock and its selected oscillator may optionally be kept running. Power Down mode can only be terminated by a hardware reset.

*Note: All external bus actions are completed before Idle or Power Down mode is entered. However, Idle or Power Down mode is **not** entered if  $\overline{READY}$  is enabled, but has not been activated (driven low) during the last bus access.*

In addition the power management selects the current CPU frequency and controls which peripherals are active.

During **Slow Down Operation** the basic clock generation path is bypassed and the CPU clock is generated via the programmable Slow Down Divider (SDD) from the selected oscillator clock signal.

**Peripheral Management** disables and enables the on-chip peripheral modules independently, reducing the amount of clocked circuitry including the respective clock drivers.

## 23.1 Idle Mode

The power consumption of the C161CS/JC/JI microcontroller can be decreased by entering Idle mode. In this mode all enabled peripherals, **including** the watchdog timer, continue to operate normally, only the CPU operation is halted and the on-chip memory modules are disabled.

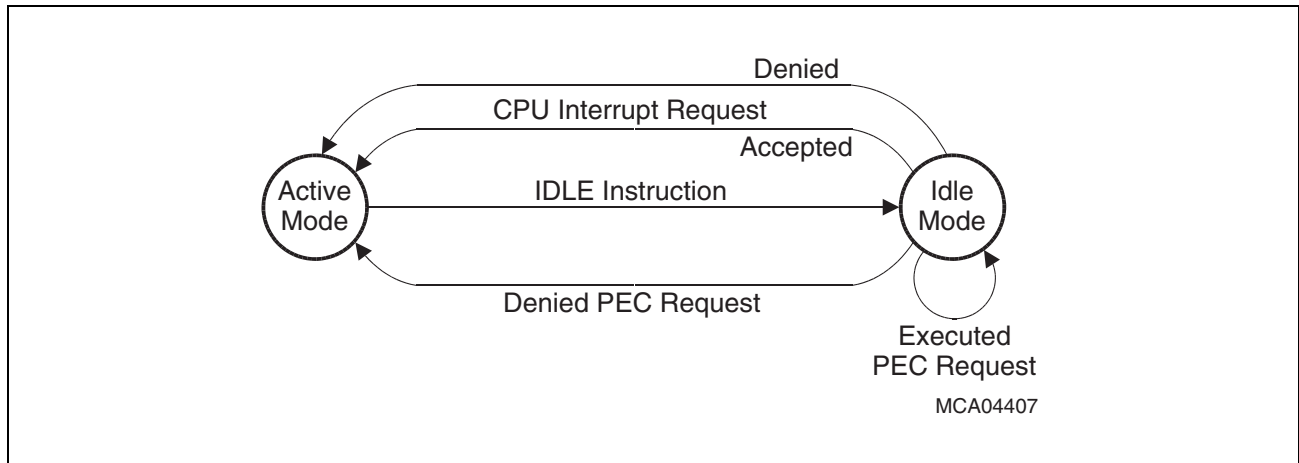
*Note: Peripherals that have been disabled via software also remain disabled after entering Idle mode, of course.*

Idle mode is entered after the IDLE instruction has been executed and the instruction before the IDLE instruction has been completed (bitfield SLEEPCON in register SYSCON1 must be '00<sub>B</sub>'). To prevent unintentional entry into Idle mode, the IDLE instruction has been implemented as a protected 32-bit instruction.

Idle mode is terminated by interrupt requests from any enabled interrupt source whose individual Interrupt Enable flag was set before the Idle mode was entered, regardless of bit IEN.

For a request selected for CPU interrupt service the associated interrupt service routine is entered if the priority level of the requesting source is higher than the current CPU priority and the interrupt system is globally enabled. After the RETI (Return from Interrupt) instruction of the interrupt service routine is executed the CPU continues executing the program with the instruction following the IDLE instruction. Otherwise, if the interrupt request cannot be serviced because of a too low priority or a globally disabled interrupt system the CPU immediately resumes normal program execution with the instruction following the IDLE instruction.

For a request which was programmed for PEC service a PEC data transfer is performed if the priority level of this request is higher than the current CPU priority and the interrupt system is globally enabled. After the PEC data transfer has been completed the CPU remains in Idle mode. Otherwise, if the PEC request cannot be serviced because of a too low priority or a globally disabled interrupt system the CPU does not remain in Idle mode but continues program execution with the instruction following the IDLE instruction.



**Figure 23-2 Transitions between Idle Mode and Active Mode**

Idle mode can also be terminated by a Non-Maskable Interrupt, i.e. a high to low transition on the  $\overline{\text{NMI}}$  pin. After Idle mode has been terminated by an interrupt or NMI request, the interrupt system performs a round of prioritization to determine the highest priority request. In the case of an NMI request, the NMI trap will always be entered.

Any interrupt request whose individual Interrupt Enable flag was set before Idle mode was entered will terminate Idle mode regardless of the current CPU priority. The CPU will **not** go back into Idle mode when a CPU interrupt request is detected, even when the interrupt was not serviced because of a higher CPU priority or a globally disabled interrupt system ( $\text{IEN} = '0'$ ). The CPU will **only** go back into Idle mode when the interrupt system is globally enabled ( $\text{IEN} = '1'$ ) **and** a PEC service on a priority level higher than the current CPU level is requested and executed.

*Note: An interrupt request which is individually enabled and assigned to priority level 0 will terminate Idle mode. The associated interrupt vector will not be accessed, however.*

The watchdog timer may be used to monitor the Idle mode: an internal reset will be generated if no interrupt or NMI request occurs before the watchdog timer overflows. To prevent the watchdog timer from overflowing during Idle mode it must be programmed to a reasonable time interval before Idle mode is entered.

## 23.2 Sleep Mode

To further reduce the power consumption the microcontroller can be switched to Sleep mode. Clocking of all internal blocks is stopped (RTC and selected oscillator optionally), the contents of the internal RAM, however, are preserved through the voltage supplied via the  $V_{DD}$  pins. The watchdog timer is stopped in Sleep mode.

Sleep mode is selected via bitfield SLEEPCON in register SYSCON1 and is entered after the IDLE instruction has been executed and the instruction before the IDLE instruction has been completed.

Sleep mode is terminated by interrupt requests from any enabled interrupt source whose individual Interrupt Enable flag was set before the Idle mode was entered, regardless of bit IEN. Mainly these are external interrupts and the RTC (if running).

*Note: The receive lines of serial interfaces may be internally routed to external interrupt inputs (see EXISEL). All peripherals except for the RTC are stopped and hence cannot generate an interrupt request.*

The realtime clock (RTC) can be kept running in Sleep mode in order to maintain a valid system time as long as the supply voltage is applied. This enables a system to determine the current time and the duration of the period while it was down (by comparing the current time with a timestamp stored when Sleep mode was entered). The supply current in this case remains well below 1 mA.

During Sleep mode the voltage at the  $V_{DD}$  pins can be lowered to 2.7 V while the RTC and its selected oscillator will still keep on running and the contents of the internal RAM will still be preserved.

When the RTC (and oscillator) is disabled the internal RAM is preserved down to a voltage of 2.5 V.

*Note: When the RTC remains active in Sleep mode also the oscillator which generates the RTC clock signal will keep on running, of course.*

*If the supply voltage is reduced the specified maximum CPU clock frequency for this case must be respected.*

*For wakeup (input edge recognition and CPU start) the power must be within the specified limits, however.*

The total power consumption in Sleep mode depends on the active circuitry (i.e. RTC on or off) and on the current that flows through the port drivers. Individual port drivers can be disabled simply by configuring them for input.

The bus interface pins can be separately disabled by releasing the external bus (disable all address windows by clearing the BUSACT bits) and switching the ports to input (if necessary). Of course the required software in this case must be executed from internal memory.

### SYSCON1

System Control Reg.1

ESFR (F1DC<sub>H</sub>/EE<sub>H</sub>)

Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	SLEEP CON
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	rw

Bit	Function
<b>SLEEPCON</b>	<b>SLEEP Mode Configuration</b> (mode entered upon the IDLE instruction) 00: Normal IDLE mode 01: SLEEP mode, with RTC running 10: Reserved. 11: SLEEP mode, with RTC and oscillator stopped

*Note: SYSCON1 is write protected after the execution of EINIT unless it is released via the unlock sequence (see [Table 23-6](#)).*

### 23.3 Power Down Mode

The microcontroller can be switched to Power Down mode which reduces the power consumption to a minimum. Clocking of all internal blocks is stopped (RTC and oscillator optionally), the contents of the internal RAM, however, are preserved through the voltage supplied via the  $V_{DD}$  pins. The watchdog timer is stopped in Power Down mode. This mode can only be terminated by an external hardware reset, i.e. by asserting a low level on the  $\overline{RSTIN}$  pin. This reset will initialize all SFRs and ports to their default state, but will not change the contents of the internal RAM.

There are two levels of protection against unintentionally entering Power Down mode. First, the PWRDN (Power Down) instruction which is used to enter this mode has been implemented as a protected 32-bit instruction. Second, this instruction is effective **only** if the  $\overline{NMI}$  (Non Maskable Interrupt) pin is externally pulled low while the PWRDN instruction is executed. The microcontroller will enter Power Down mode after the PWRDN instruction has completed.

This feature can be used in conjunction with an external power failure signal which pulls the  $\overline{NMI}$  pin low when a power failure is imminent. The microcontroller will enter the NMI trap routine which can save the internal state into RAM. After the internal state has been saved, the trap routine may then execute the PWRDN instruction. If the  $\overline{NMI}$  pin is still low at this time, Power Down mode will be entered, otherwise program execution continues.

The initialization routine (executed upon reset) can check the reset identification flags in register WDTCN to determine whether the controller was initially switched on, or whether it was properly restarted from Power Down mode.

The realtime clock (RTC) can be kept running in Power Down mode in order to maintain a valid system time as long as the supply voltage is applied. This enables a system to determine the current time and the duration of the period while it was down (by comparing the current time with a timestamp stored when Power Down mode was entered). The supply current in this case remains well below 1 mA.

During power down the voltage at the  $V_{DD}$  pins can be lowered to 2.7 V while the RTC and its selected oscillator will still keep on running and the contents of the internal RAM will still be preserved.

When the RTC (and oscillator) is disabled the internal RAM is preserved down to a voltage of 2.5 V.

*Note: When the RTC remains active in Power Down mode also the oscillator which generates the RTC clock signal will keep on running, of course.  
If the supply voltage is reduced the specified maximum CPU clock frequency for this case must be respected.*

The total power consumption in Power Down mode depends on the active circuitry (i.e. RTC on or off) and on the current that flows through the port drivers. To minimize the consumed current the RTC and/or all pin drivers can be disabled (pins switched to tristate) via a central control bitfield in register SYSCON2. If an application requires one or more port drivers to remain active even in Power Down mode also individual port drivers can be disabled simply by configuring them for input.

The bus interface pins can be separately disabled by releasing the external bus (disable all address windows by clearing the BUSACT bits) and switching the ports to input (if necessary). Of course the required software in this case must be executed from internal memory.



### **23.3.1 Status of Output Pins During Power Reduction Modes**

**During Idle mode** the CPU clocks are turned off, while all peripherals continue their operation in the normal way. Therefore all ports pins, which are configured as general purpose output pins, output the last data value which was written to their port output latches. If the alternate output function of a port pin is used by a peripheral, the state of the pin is determined by the operation of the peripheral.

Port pins which are used for bus control functions go into that state which represents the inactive state of the respective function (e.g.  $\overline{WR}$ ), or to a defined state which is based on the last bus access (e.g.  $\overline{BHE}$ ). Port pins which are used as external address/data bus hold the address/data which was output during the last external memory access before entry into Idle mode under the following conditions:

P0H outputs the high byte of the last address if a multiplexed bus mode with 8-bit data bus is used, otherwise P0H is floating. P0L is always floating in Idle mode.

PORT1 outputs the lower 16 bits of the last address if a demultiplexed bus mode is used, otherwise the output pins of PORT1 represent the port latch data.

Port 4 outputs the segment address for the last access on those pins that are selected as segment address lines, otherwise the output pins of Port 4 represent the port latch data.

**During Sleep mode** the oscillator (except for RTC operation) and the clocks to the CPU and to the peripherals are turned off. Like in Idle mode, all port pins which are configured as general purpose output pins output the last data value which was written to their port output latches.

When the alternate output function of a port pin is used by a peripheral the state of this pin is determined by the last action of the peripheral before the clocks were switched off.

**During Power Down mode** the oscillator (except for RTC operation) and the clocks to the CPU and to the peripherals are turned off. Like in Idle mode, all port pins which are configured as general purpose output pins output the last data value which was written to their port output latches.

When the alternate output function of a port pin is used by a peripheral the state of this pin is determined by the last action of the peripheral before the clocks were switched off.

*Note: All pin drivers can be switched off by selecting the general port disable function prior to entering Power Down mode.*

*When the supply voltage is lowered in Power Down mode the high voltage of output pins will decrease accordingly.*



**Table 23-1 State of C161CS/JC/JI Output Pins during Idle and Power Down Mode**

C161CS/JC/ JI Output Pin(s)	External Bus Enabled		No External Bus	
	Idle Mode	Sleep and Power Down	Idle Mode	Sleep and Power Down
CLKOUT	Active (toggling)	High	Active (toggling)	High
FOUT	Active (toggling)	Hold (high / low)	Active (toggling)	Hold (high / low)
ALE	Low		Low	
$\overline{RD}$ , $\overline{WR}$	High		High	
P0L	Floating		Port Latch Data	
P0H	A15 ... A8 <sup>1)</sup> / Float		Port Latch Data	
PORT1	Last Address <sup>2)</sup> / Port Latch Data		Port Latch Data	
Port 4	Port Latch Data / Last segment		Port Latch Data	
$\overline{BHE}$	Last value		Port Latch Data	
$\overline{CSx}$	Last value <sup>3)</sup>		Port Latch Data	
$\overline{RSTOUT}$	High if EINIT was executed before entering Idle or Power Down mode, Low otherwise.			
Other Port Output Pins	Port Latch Data / Alternate Function			

1) For multiplexed buses with 8-bit data bus.

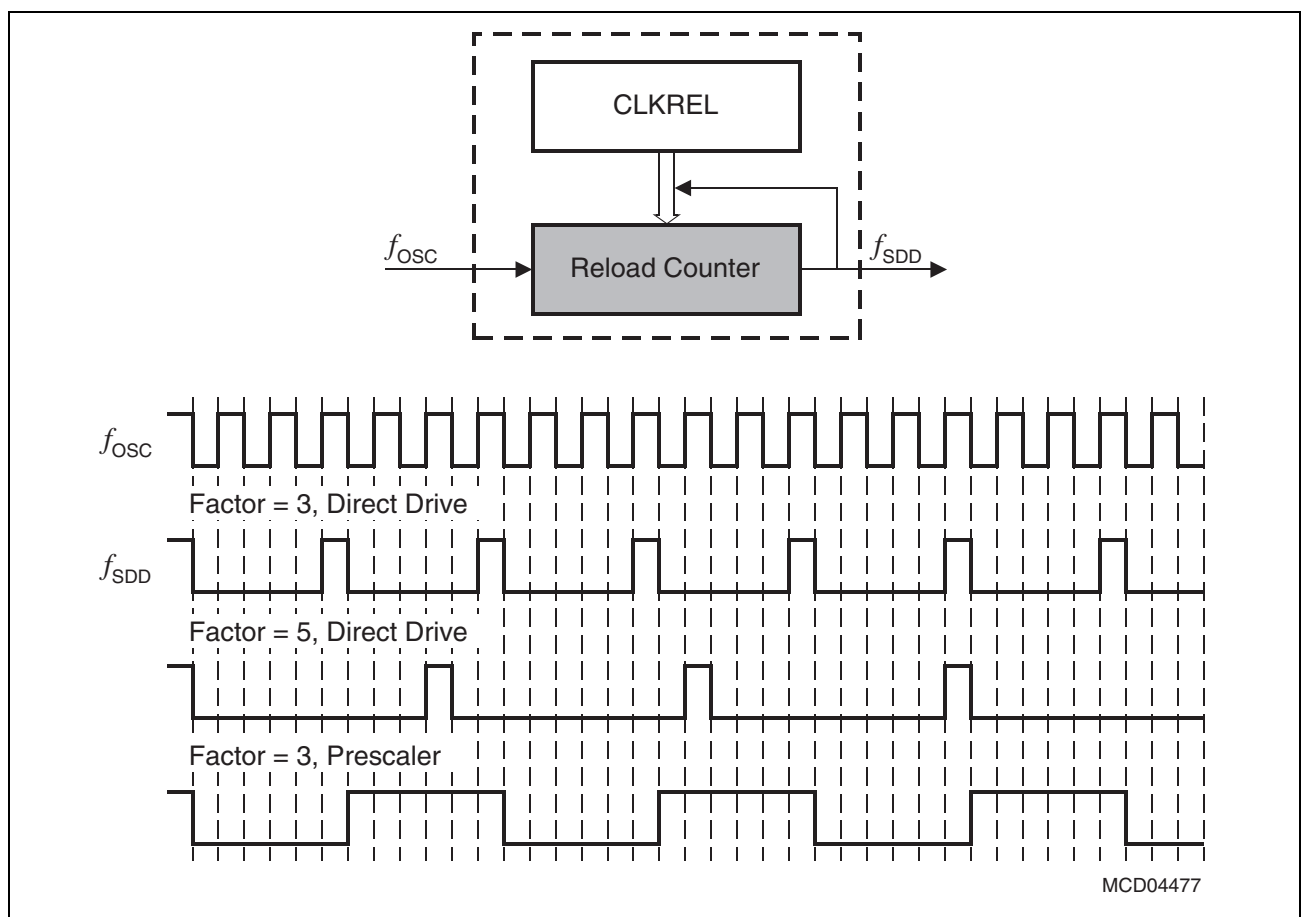
2) For demultiplexed buses.

3) The  $\overline{CS}$  signal that corresponds to the last address remains active (low), all other enabled  $\overline{CS}$  signals remain inactive (high). By accessing an on-chip X-Peripheral prior to entering a power save mode all external  $\overline{CS}$  signals can be deactivated.

## 23.4 Slow Down Operation

A separate clock path can be selected for Slow Down operation bypassing the basic clock path used for standard operation. The programmable Slow Down Divider (SDD) divides the oscillator frequency by a factor of 1 ... 32 which is specified via bitfield CLKREL in register SYSCON2 (factor = <CLKREL>+1). When bitfield CLKREL is written during SDD operation the reload counter will output one more clock pulse with the “old” frequency in order to resynchronize internally before generating the “new” frequency.

If direct drive mode is configured clock signal  $f_{DD}$  is directly fed to  $f_{CPU}$ , if prescaler mode is configured clock signal  $f_{DD}$  is additionally divided by 2:1 to generate  $f_{CPU}$  (see examples below).



**Figure 23-3 Slow Down Divider Operation**

Using e.g. a 5 MHz input clock the on-chip logic may be run at a frequency down to 156.25 kHz (or 78 kHz) without an external hardware change. An implemented PLL may be switched off in this case or kept running, depending on the requirements of the application (see [Table 23-2](#)).

*Note: During Slow Down operation the whole device (including bus interface and generation of signals CLKOUT or FOUT) is clocked with the SDD clock (see [Figure 23-3](#)).*

**Table 23-2 PLL Operation in Slow Down Mode**

	<b>Advantage</b>	<b>Disadvantage</b>	<b>Oscillator Watchdog</b>
<b>PLL running</b>	Fast switching back to basic clock source	PLL adds to power consumption	Active if not disabled via bit OWDDIS
<b>PLL off</b>	PLL causes no additional power consumption	PLL must lock before switching back to the basic clock source (if the PLL is the basic clock source)	Disabled

All these clock options are selected via bitfield CLKCON in register SYSCON2. A state machine controls the switching mechanism itself and ensures a continuous and glitch-free clock signal to the on-chip logic. This is especially important when switching back to PLL frequency when the PLL has temporarily been switched off. In this case the clock source can be switched back either automatically as soon as the PLL is locked again (indicated by bit CLKLOCK in register SYSCON2), or manually, i.e. under software control, after bit CLKLOCK has become '1'. The latter way is preferable if the application requires a defined point where the frequency changes.

*Note: When the PLL is the basic clock source and a reset occurs during SDD operation with the PLL off, the internal reset condition is extended so the PLL can lock before execution begins. The reset condition is terminated prematurely if no stable oscillator clock is detected. This ensures the operability of the device in case of a missing input clock signal.*

Switching to Slow Down operation affects frequency sensitive peripherals like serial interfaces, timers, PWM, etc. If these units are to be operated in Slow Down mode their precalers or reload values must be adapted. Please note that the reduced CPU frequency decreases e.g. timer resolution and increases the step width e.g. for baudrate generation. The oscillator frequency in such a case should be chosen to accommodate the required resolutions and/or baudrates.

**SYSCON2**

System Control Reg.2

ESFR (F1D0<sub>H</sub>/E8<sub>H</sub>)

Reset Value: 00X0<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>CLK LOCK</b>	<b>CLKREL</b>				<b>CLKCON</b>		<b>SCS</b>	<b>RCS</b>	<b>PDCON</b>		<b>SYSRLS</b>				
rh	rw				rw		rw	rw	rw		rwh				

Bit	Function
<b>SYSRLS</b>	<b>Register Release Function</b> (Unlock field) Must be written in a defined way in order to execute the unlock sequence. See separate description ( <a href="#">Table 23-6</a> ).
<b>PDCON</b>	<b>Power Down Control</b> (during power down mode) 00: RTC = On, Ports = On (default after reset). 01: RTC = On, Ports = Off. 10: RTC = Off, Ports = On. 11: RTC = Off, Ports = Off.
<b>RCS</b>	<b>RTC Clock Source</b> (not affected by a reset!) 0: Main oscillator. 1: Auxiliary oscillator (32 kHz).
<b>SCS</b>	<b>SDD Clock Source</b> (not affected by a reset!) 0: Main oscillator. 1: Auxiliary oscillator (32 kHz).
<b>CLKCON</b>	<b>Clock State Control</b> 00: Running on configured basic frequency. 01: Running on slow down frequency, PLL remains ON. 10: Running on slow down frequency, PLL switched OFF. 11: Reserved. Do not use this combination.
<b>CLKREL</b>	<b>Reload Counter Value for Slowdown Divider</b> (SDD factor = CLKREL + 1)
<b>CLKLOCK</b>	<b>Clock Signal Status Bit</b> 0: Main oscillator is unstable or PLL is unlocked. 1: Main oscillator is stable <b>and</b> PLL is locked.

*Note: SYSCON2 (except for bitfield SYSRLS, of course) is write protected after the execution of EINIT unless it is released via the unlock sequence (see [Table 23-6](#)).*

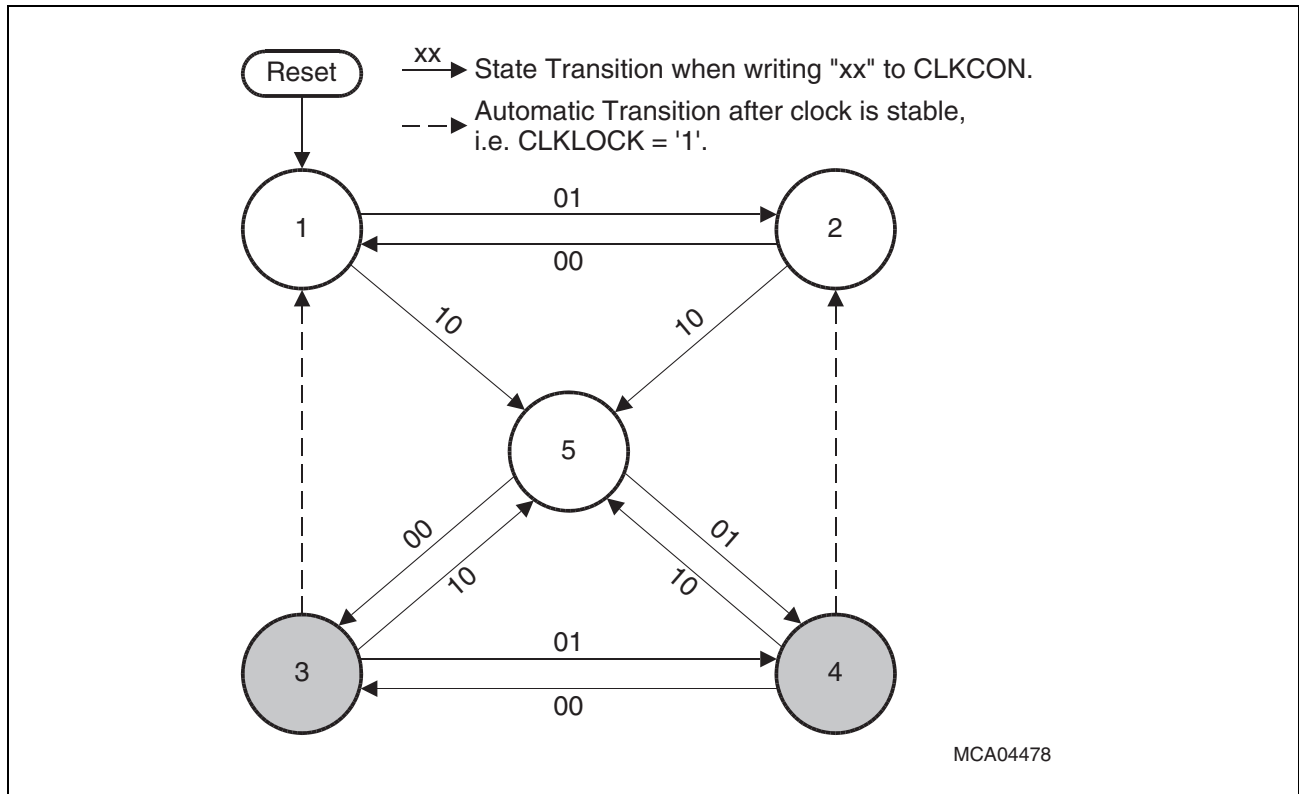


Figure 23-4 Clock Switching State Machine

Table 23-3 Clock Switching State Description

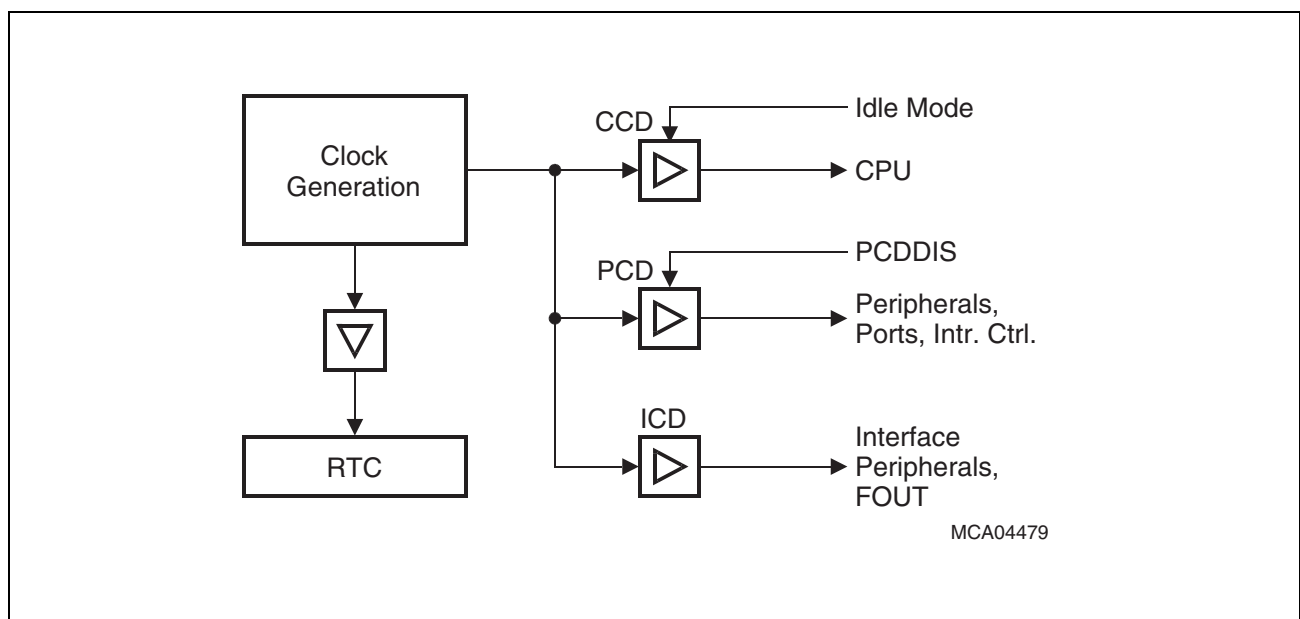
State Number	PLL Status	$f_{CPU}$ Source	CLK CON	Note
1	Locked <sup>1)</sup>	Basic	00	Standard operation on basic clock frequency.
2	Locked <sup>1)</sup>	SDD	01	SDD operation with PLL On <sup>1)</sup> . Fast (without delay) or manual switch back (from 5) to basic clock frequency.
3	Transient <sup>1)</sup>	SDD	(00)	Intermediate state leading to state 1.
4	Transient <sup>1)</sup>	SDD	(01)	Intermediate state leading to state 2.
5	Off	SDD	10	SDD operation with PLL Off. Reduced power consumption.

<sup>1)</sup> The indicated PLL status only applies if the PLL is selected as the basic clock source. If the basic clock source is direct drive or prescaler the PLL will not lock. If the oscillator watchdog is disabled (OWDDIS = '1') the PLL will be off.

### 23.5 Flexible Peripheral Management

The power consumed by the C161CS/JC/JI also depends on the amount of active logic. Peripheral management enables the system designer to deactivate those on-chip peripherals that are not required in a given system status (e.g. a certain interface mode or standby). All modules that remain active, however, will still deliver their usual performance. If all modules that are fed by the peripheral clock driver (PCD) are disabled and also the other functions fed by the PCD are not required, this clock driver itself may also be disabled to save additional power.

This flexibility is realized by distributing the CPU clock via several clock drivers which can be separately controlled, and may also be smaller.



**Figure 23-5 CPU Clock Distribution**

*Note: The Real Time Clock (RTC) is fed by a separate clock driver, so it can be kept running even in Power Down mode while still all the other circuitry is disconnected from the clock.*

The registers of the generic peripherals can be accessed even while the respective module is disabled, as long as PCD is running (the registers of peripherals which are connected to ICD can be accessed even in this case, of course). The registers of X-peripherals cannot be accessed while the respective module is disabled by any means.

While a peripheral is disabled its output pins remain in the state they had at the time of disabling.

Software controls this flexible peripheral management via register SYSCON3 where each control bit is associated with an on-chip peripheral module.

**SYSCON3**

**System Control Reg.3**

**ESFR (F1D4<sub>H</sub>/EA<sub>H</sub>)**

**Reset Value: 0000<sub>H</sub>**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>PCD DIS</b>	<b>CAN2 DIS</b>	<b>CAN1 DIS</b>	<b>SDLM DIS</b>	<b>IIC DIS</b>	<b>ASC 1 DIS</b>	-	-	<b>CC2 DIS</b>	<b>CC1 DIS</b>	-	-	<b>GPT DIS</b>	<b>SSC DIS</b>	<b>ASC 0 DIS</b>	<b>ADC DIS</b>
rw	rw	rw	rw	rw	rw	-	-	rw	rw	-	-	rw	rw	rw	rw

Bit	Function (associated peripheral module)
<b>ADCDIS</b>	Analog/Digital Converter
<b>ASC0DIS</b>	USART ASC0
<b>SSCDIS</b>	Synchronous Serial Channel SSC
<b>GPTDIS</b>	General Purpose Timer Blocks
<b>CC1DIS</b>	CAPCOM Unit 1
<b>CC2DIS</b>	CAPCOM Unit 2
<b>ASC1DIS</b>	USART ASC1
<b>IICDIS</b>	On-chip IIC Bus Module
<b>SDLMDIS</b>	On-chip SDLM (J1850 Module) exists only in the C161JC and C161JI
<b>CAN1DIS</b>	On-chip CAN Module 1 <sup>1)</sup>
<b>CAN2DIS</b>	On-chip CAN Module 2 <sup>1)</sup> exists only in the C161CS
<b>PCDDIS</b>	Peripheral Clock Driver (also X-Peripherals)

<sup>1)</sup> When bit CANxDIS is cleared the CAN module is re-activated by an internal reset signal and must then be re-configured in order to operate properly.

*Note: The allocation of peripheral disable bits within register SYSCON3 is device specific and may be different in other derivatives than the C161CS/JC/JI. SYSCON3 is write protected after the execution of EINIT unless it is released via the unlock sequence (see [Table 23-6](#)).*

When disabling the peripheral clock driver (PCD), the following details should be respected:

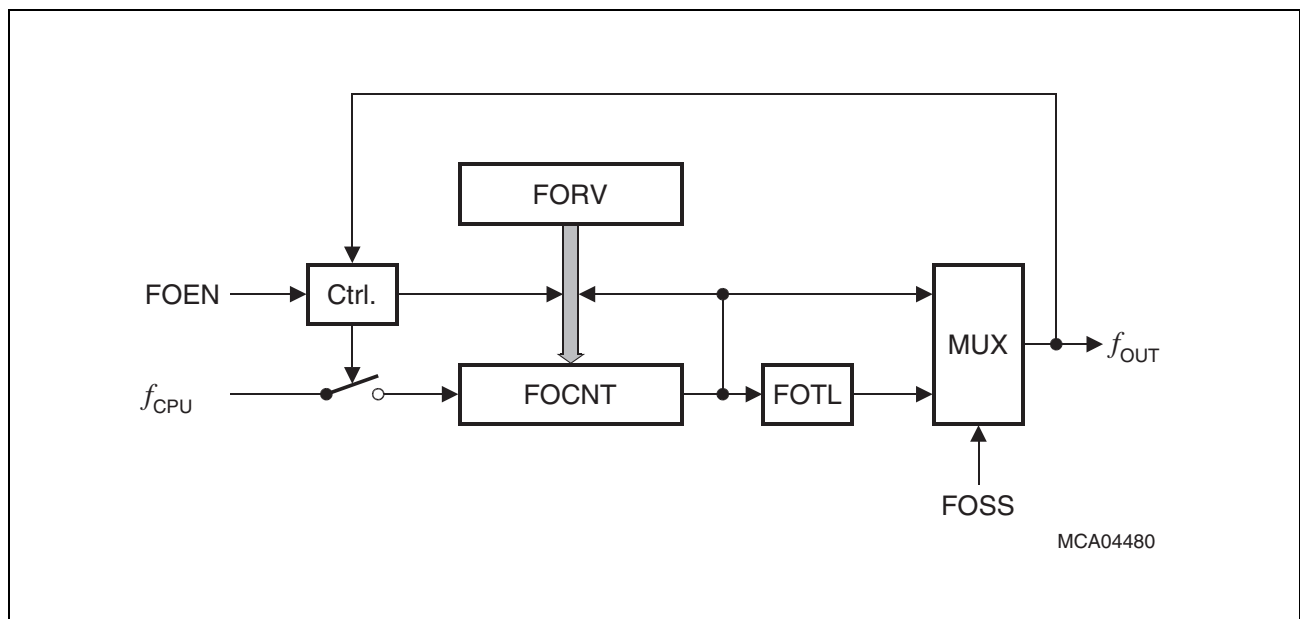
- The clock signal for all connected peripherals is stopped. Make sure that all peripherals enter a safe state before disabling PCD.
- The output signal CLKOUT will remain HIGH (FOUT will keep on toggling).
- Interrupt requests will still be recognized even while PCD is disabled.
- No new output values are gated from the port output latches to the output port pins and no new input values are latched from the input port pins.
- No register access is possible for generic peripherals  
(register access is possible for individually disabled generic peripherals,  
no register access at all is possible for disabled X-Peripherals)



## 23.6 Programmable Frequency Output Signal

The system clock output (CLKOUT) can be replaced by the programmable frequency output signal  $f_{OUT}$ . This signal can be controlled via software (contrary to CLKOUT), and so can be adapted to the requirements of the connected external circuitry. The programmability also extends the power management to a system level, as also circuitry (peripherals, etc.) outside the C161CS/JC/JI can be influenced, i.e. run at a scalable frequency or temporarily can be switched off completely.

This clock signal is generated via a reload counter, so the output frequency can be selected in small steps. An optional toggle latch provides a clock signal with a 50% duty cycle.



**Figure 23-6 Clock Output Signal Generation**

Signal  $f_{OUT}$  always provides complete output periods (see Signal Waveforms below):

- When  $f_{OUT}$  is started (FOEN-->'1') FOCNT is loaded from FORV
- When  $f_{OUT}$  is stopped (FOEN-->'0') FOCNT is stopped when  $f_{OUT}$  has reached (or is) '0'.

Signal  $f_{OUT}$  is independent from the peripheral clock driver PCD. While CLKOUT would stop when PCD is disabled,  $f_{OUT}$  will keep on toggling. Thus external circuitry may be controlled independent from on-chip peripherals.

*Note: Counter FOCNT is clocked with the CPU clock signal  $f_{CPU}$  (see [Figure 23-6](#)) and therefore will also be influenced by the SDD operation.*

Register FOCON provides control over the output signal generation (frequency, waveform, activation) as well as all status information (counter value, FOTL).

**FOCON**

Freque.Output Control Reg.

SFR (FFAA<sub>H</sub>/D5<sub>H</sub>)

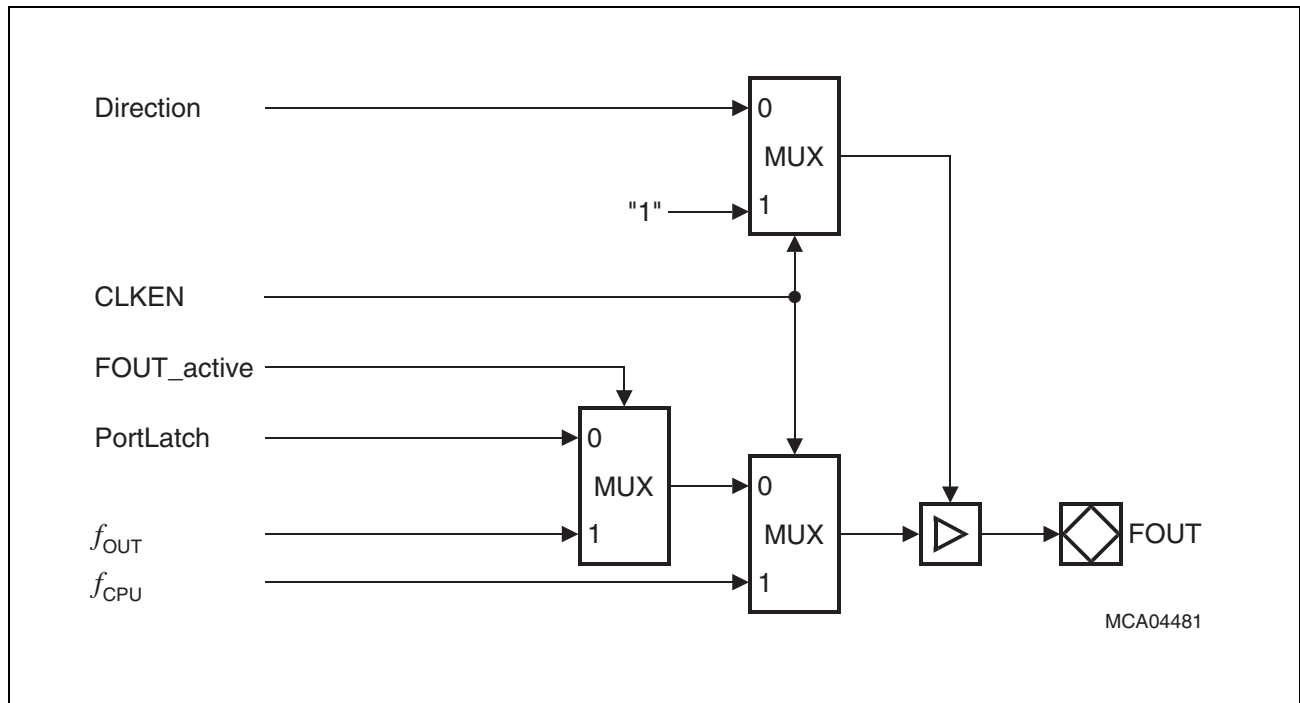
Reset Value: 0000<sub>H</sub>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
FO EN	FO SS	FORV						-	FO TL	FOCNT						
rw	rw	rw						-	rwh	rwh						

Bit	Function
FOCNT	Frequency Output Counter
FOTL	Frequency Output Toggle Latch Is toggled upon each underflow of FOCNT.
FORV	Frequency Output Reload Value Is copied to FOCNT upon each underflow of FOCNT.
FOSS	Frequency Output Signal Select 0: Output of the toggle latch: duty cycle = 50%. 1: Output of the reload counter: duty cycle depends on FORV.
FOEN	Frequency Output Enable 0: Frequency output generation stops when signal $f_{OUT}$ is/gets low. 1: FOCNT is running, $f_{OUT}$ is gated to pin. First reload after 0-1 transition.

*Note: It is not recommended to write to any part of bitfield FOCNT, especially not while the counter is running. Writing to FOCNT prior to starting the counter is obsolete because it will immediately be reloaded from FORV. Writing to FOCNT during operation may produce unintended counter values.*

Signal  $f_{OUT}$  in the C161CS/JC/JI is an alternate function of pin P3.15/CLKOUT/FOUT.



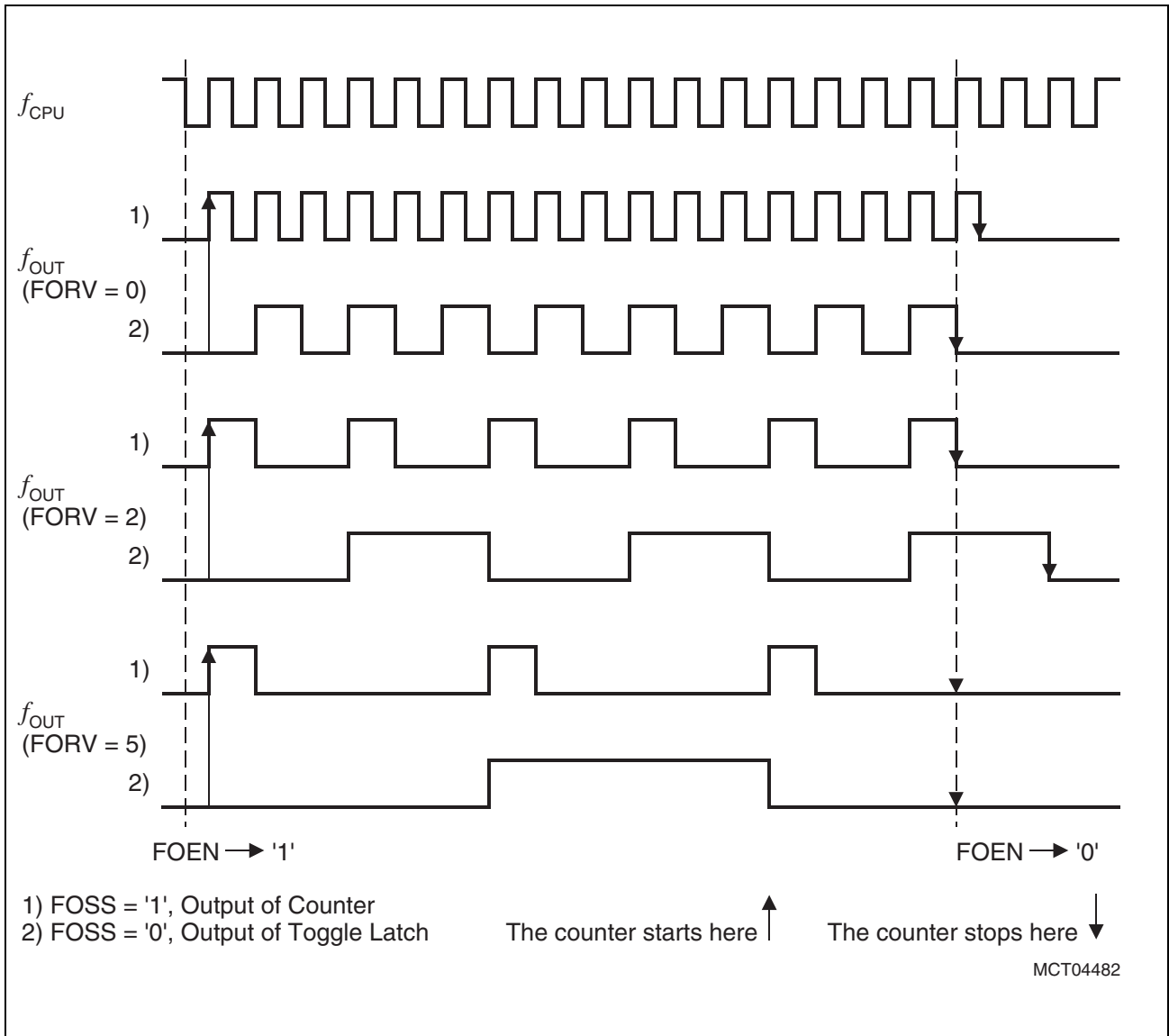
**Figure 23-7 Connection to Port Logic (Functional Approach)**

A priority ranking determines which function controls the shared pin:

**Table 23-4 Priority Ranking for Shared Output Pin**

Priority	Function	Control
1	CLKOUT	CLKEN = '1', FOEN = 'x'
2	FOUT	CLKEN = '0', FOEN = '1'
3	General purpose IO	CLKEN = '0', FOEN = '0'

*Note: For the generation of  $f_{OUT}$  pin FOUT must be switched to output, i.e. DP3.15 = '1'. While  $f_{OUT}$  is disabled the pin is controlled by the port latch (see [Figure 23-7](#)). The port latch P3.15 must be '0' in order to maintain the  $f_{OUT}$  inactive level on the pin.*



**Figure 23-8 Signal Waveforms**

*Note: The output signal (for  $FOSS = '1'$ ) is high for the duration of one  $f_{CPU}$  cycle for all reload values  $FORV > 0$ . For  $FORV = 0$  the output signal corresponds to  $f_{CPU}$ .*

**Output Frequency Calculation**

The output frequency can be calculated as  $f_{OUT} = f_{CPU} / ((FORV + 1) \times 2^{(1 - FOSS)})$ ,  
 so  $f_{OUTmin} = f_{CPU} / 128$  (FORV = 3F<sub>H</sub>, FOSS = '0'),  
 and  $f_{OUTmax} = f_{CPU} / 1$  (FORV = 00<sub>H</sub>, FOSS = '1').

**Table 23-5 Selectable Output Frequency Range for  $f_{OUT}$**

$f_{CPU}$	$f_{OUT}$ in [kHz] for FORV = xx, FOSS = 1/0					FORV for $f_{OUT} = 1$ MHz	
	00 <sub>H</sub>	01 <sub>H</sub>	02 <sub>H</sub>	3E <sub>H</sub>	3F <sub>H</sub>	FOSS = 0	FOSS = 1
<b>4 MHz</b>	4000	2000	1333.33	63.492	62.5	01 <sub>H</sub>	03 <sub>H</sub>
	2000	1000	666.67	31.746	31.25		
<b>10 MHz</b>	10000	5000	3333.33	158.73	156.25	04 <sub>H</sub>	09 <sub>H</sub>
	5000	2500	1666.67	79.365	78.125		
<b>12 MHz</b>	12000	6000	4000	190.476	187.5	05 <sub>H</sub>	0B <sub>H</sub>
	6000	3000	2000	95.238	93.75		
<b>16 MHz</b>	16000	8000	5333.33	253.968	250	07 <sub>H</sub>	0F <sub>H</sub>
	8000	4000	2666.67	126.984	125		
<b>20 MHz</b>	20000	10000	6666.67	317.46	312.5	09 <sub>H</sub>	13 <sub>H</sub>
	10000	5000	3333.33	158.73	156.25		
<b>25 MHz</b>	25000	12500	8333.33	396.825	390.625	0B <sub>H</sub> (1.04167) 0C <sub>H</sub> (0.96154)	18 <sub>H</sub>
	12500	6250	4166.67	198.413	195.313		
<b>33 MHz</b>	33000	16500	11000	523.810	515.625	0F <sub>H</sub> (1.03125) 10 <sub>H</sub> (0.97059)	20 <sub>H</sub>
	16500	8250	5500	261.905	257.816		

## 23.7 Security Mechanism

The power management control registers belong to a set of registers (see [Table 23-7](#)) which control functions and modes which are critical for the C161CS/JC/JI's operation. For this reason they are locked (except for bitfield SYSRSLs in register SYSCON2) after the execution of EINIT (like register SYSCON) so these vital system functions cannot be changed inadvertently e.g. by software errors. However, as these registers control important functions (e.g. the power management) they need to be accessed during operation to select the appropriate mode. The system control software gets this access via a special unlock sequence which allows **one single** write access **to one register** of this set when executed properly. This provides a maximum of security.

*Note: Of course all these registers may be read at any time without restrictions.*

The unlock sequence is executed by writing defined values to bitfield SYSRSLs using defined instructions (see [Table 23-6](#)). The instructions of the unlock sequence (including the intended write access) must be secured with an EXTR instruction (switch to ESFR space and lock interrupts).

*Note: The unlock sequence is aborted if the locked range (EXTR) does not cover the complete sequence.*

*The unlock sequence provides no write access to register SYSCON.*

**Table 23-6 Unlock Sequence for Secured Registers**

Step	SYSRSL	Instruction	Notes
–	0000 <sub>B</sub> <sup>1)</sup>	–	Status before release sequence
1	1001 <sub>B</sub>	BFLDL, OR, ORB <sup>2)</sup> , XOR, XORB <sup>2)</sup>	Read-Modify-Write access
2	0011 <sub>B</sub>	MOV, MOVB <sup>2)</sup> , MOVBS <sup>2)</sup> , MOVBZ <sup>2)</sup>	Write access
3	0111 <sub>B</sub>	BSET, BMOV <sup>2)</sup> , BMOVN <sup>2)</sup> , BOR <sup>2)</sup> , BXOR <sup>2)</sup>	Read-Modify-Write access, bit instruction
4	–	–	Single (read-modify-)write access to one of the secured registers
–	0000 <sub>B</sub> <sup>3)</sup>	–	Status after release sequence

<sup>1)</sup> SYSRSLs must be set to 0000<sub>B</sub> before the first step, if any OR command is used.

<sup>2)</sup> Usually byte accesses should not be used for special function registers.

<sup>3)</sup> SYSRSLs is cleared by hardware if unlock sequence and write access were successful. SYSRSLs shows the last value written otherwise.

The following registers are secured by the described unlock sequence:

**Table 23-7 Special Registers Secured by the Unlock Sequence**

Register Name	Description
SYSCON1	Controls sleep mode
SYSCON2	Controls clock generation (SDD) and the unlock sequence itself
SYSCON3	Controls the flexible peripheral management
RSTCON	Controls the configuration of the C161CS/JC/JI (basic clock generation mode, $\overline{CS}$ lines, segment address width) and the length of the reset sequence

### Code Examples

The code examples below show how the unlock sequence is used to access register SYSCON2 (marked **!\*!** in the comment column) in an application in order to change the basic clock generation mode.

#### Examples where the PLL keeps running:

```

;_____ ;_____
ENTER_SLOWDOWN: ;Currently running on basic clock frequ.
MOV SYSCON2, ZEROS ;Clear bits 3-0 (no EXTR required here)
EXTR #4H ;Switch to ESFR space and lock sequence
BFLDL SYSCON2,#0FH,#09H ;Unlock sequence, step 1 (1001B)
MOV SYSCON2,#0003H ;Unlock sequence, step 2 (0011B)
BSET SYSCON2.2 ;Unlock sequence, step 3 (0111B)
;Single access to one locked register
BFLDH SYSCON2,#03H,#01H ;CLKCON=01B --> SDD frequency, PLL on *!*

```

```

;_____ ;_____
EXIT_SLOWDOWN: ;Currently running on SDD frequency
MOV SYSCON2, ZEROS ;Clear bits 3-0 (no EXTR required here)
EXTR #4H ;Switch to ESFR space and lock sequence
BFLDL SYSCON2,#0FH,#09H ;Unlock sequence, step 1 (1001B)
MOV SYSCON2,#0003H ;Unlock sequence, step 2 (0011B)
BSET SYSCON2.2 ;Unlock sequence, step 3 (0111B)
;Single access to one locked register
BFLDH SYSCON2,#03H,#00H ;CLKCON=00B --> basic frequency *!*

```

**Examples where the PLL is disabled:**

```

; _____ ; _____
ENTER_SLOWDOWN:      ;Currently running on basic clock frequ.
EXTR  #1H             ;Next access to ESFR space
BCLR  ISNC.2          ;PLLIE='0', i.e. PLL interrupt disabled
MOV   SYSCON2, ZEROS ;Clear bits 3-0 (no EXTR required here)
EXTR  #4H             ;Switch to ESFR space and lock sequence
BFLDL SYSCON2,#0FH,#09H ;Unlock sequence, step 1 (1001B)
MOV   SYSCON2,#0003H ;Unlock sequence, step 2 (0011B)
BSET  SYSCON2.2       ;Unlock sequence, step 3 (0111B)
                               ;Single access to one locked register
BFLDH SYSCON2,#03H,#02H ;CLKCON=10B --> SDD frequency, PLL off*!*

; _____ ; _____
SDD_EXIT_AUTO:       ;Currently running on SDD frequency
MOV   SYSCON2, ZEROS ;Clear bits 3-0 (no EXTR required here)
EXTR  #4H             ;Switch to ESFR space and lock sequence
BFLDL SYSCON2,#0FH,#09H ;Unlock sequence, step 1 (1001B)
MOV   SYSCON2,#0003H ;Unlock sequence, step 2 (0011B)
BSET  SYSCON2.2       ;Unlock sequence, step 3 (0111B)
                               ;Single access to one locked register
BFLDH SYSCON2,#03H,#00H ;CLKCON=00B--> basic frequ./start PLL*!*
EXTR  #1H             ;Next access to ESFR space
BSET  ISNC.2          ;PLLIE='1', i.e. PLL interrupt enabled

```



```

; _____ ; _____
SDD_EXIT_MANUAL: ;Currently running on SDD frequency
MOV   SYSCON2, ZEROS ;Clear bits 3-0 (no EXTR required here)
EXTR  #4H ;Switch to ESFR space and lock sequence
BFLDL SYSCON2,#0FH,#09H ;Unlock sequence, step 1 (1001B)
MOV   SYSCON2,#0003H ;Unlock sequence, step 2 (0011B)
BSET  SYSCON2.2 ;Unlock sequence, step 3 (0111B)
;Single access to one locked register
BFLDH SYSCON2,#03H,#01H ;CLKCON=01B --> stay on SDD/start PLL *!*
;
USER_CODE: ;Space for any user code that...
;...must or can be executed before...
;...switching back to basic clock

CLOCK_OK:
EXTR  #1H ;Next access to ESFR space
JNB   SYSCON2.15,CLOCK_OK;Wait until clock OK (CLKLOCK='1')
;
MOV   SYSCON2, ZEROS ;Clear bits 3-0 (no EXTR required here)
EXTR  #4H ;Switch to ESFR space and lock sequence
BFLDL SYSCON2,#0FH,#09H ;Unlock sequence, step 1 (1001B)
MOV   SYSCON2,#0003H ;Unlock sequence, step 2 (0011B)
BSET  SYSCON2.2 ;Unlock sequence, step 3 (0111B)
;Single access to one locked register
BFLDH SYSCON2,#03H,#00H ;CLKCON=00B --> basic frequency *!*
EXTR  #1H ;Next access to ESFR space
BSET  ISNC.2 ;PLLIE='1', i.e. PLL interrupt enabled

```

## 24 System Programming

To aid in software development, a number of features has been incorporated into the instruction set of the C161CS/JC/JI, including constructs for modularity, loops, and context switching. In many cases commonly used instruction sequences have been simplified while providing greater flexibility. The following programming features help to fully utilize this instruction set.

### Instructions Provided as Subsets of Instructions

In many cases, instructions found in other microcontrollers are provided as subsets of more powerful instructions in the C161CS/JC/JI. This allows the same functionality to be provided while decreasing the hardware required and decreasing decode complexity. In order to aid assembly programming, these instructions, familiar from other microcontrollers, can be built in macros, thus providing the same names.

**Directly Substitutable Instructions** are instructions known from other microcontrollers that can be replaced by the following instructions of the C161CS/JC/JI:

**Table 24-1 Substitution of Instructions**

Substituted Instruction		C161CS/JC/JI Instruction		Function
CLR	Rn	AND	Rn, #0 <sub>H</sub>	Clear register
CPLB	Bit	BMOVN	Bit, Bit	Complement bit
DEC	Rn	SUB	Rn, #1 <sub>H</sub>	Decrement register
INC	Rn	ADD	Rn, #1 <sub>H</sub>	Increment register
SWAPB	Rn	ROR	Rn, #8 <sub>H</sub>	Swap bytes within word

**Modification of System Flags** is performed using bit set or bit clear instructions (BSET, BCLR). All bit and word instructions can access the PSW register, so no instructions like CLEAR CARRY or ENABLE INTERRUPTS are required.

**External Memory Data Access** does not require special instructions to load data pointers or explicitly load and store external data. The C161CS/JC/JI provides a Von Neumann memory architecture and its on-chip hardware automatically detects accesses to internal RAM, GPRs, and SFRs.

### Multiplication and Division

Multiplication and division of words and double words is provided through multiple cycle instructions implementing a Booth algorithm. Each instruction implicitly uses the 32-bit register MD (MDL = lower 16 bits, MDH = upper 16 bits). The MDRIU flag (Multiply or Divide Register In Use) in register MDC is set whenever either half of this register is written to or when a multiply/divide instruction is started. It is cleared whenever the MDL

register is read. Because an interrupt can be acknowledged before the contents of register MD are saved, this flag is required to alert interrupt routines, which require the use of the multiply/divide hardware, so they can preserve register MD. This register, however, only needs to be saved when an interrupt routine requires use of the MD register and a previous task has not saved the current result. This flag is easily tested by the Jump-on-bit instructions.

Multiplication or division is simply performed by specifying the correct (signed or unsigned) version of the multiply or divide instruction. The result is then stored in register MD. The overflow flag (V) is set if the result from a multiply or divide instruction is greater than 16 bits. This flag can be used to determine whether both word halves must be transferred from register MD. The high portion of register MD (MDH) must be moved into the register file or memory first, in order to ensure that the MDRIU flag reflects the correct state.

The following instruction sequence performs an unsigned 16 by 16-bit multiplication:

```

SAVE:
JNB      MDRIU, START;Test if MD was in use.
SCXT     MDC, #0010H ;Save and clear control register,
          ;leaving MDRIU set
          ;(only required for interrupted
          ;multiply/divide instructions)
BSET     SAVED          ;Indicate the save operation
PUSH     MDH            ;Save previous MD contents ...
PUSH     MDL            ;... on system stack
START:
MULU     R1, R2          ;Multiply 16·16 unsigned, Sets MDRIU
JMPR     cc_NV, COPYL;Test for only 16-bit result
MOV      R3, MDH        ;Move high portion of MD
COPYL:
MOV      R4, MDL        ;Move low portion of MD, Clears MDRIU
RESTORE:
JNB      SAVED, DONE   ;Test if MD registers were saved
POP      MDL            ;Restore registers
POP      MDH
POP      MDC
BCLR     SAVED          ;Multiplication is completed,
          ;program continues
DONE:    ...

```

The above save sequence and the restore sequence after COPYL are only required if the current routine could have interrupted a previous routine which contained a MUL or DIV instruction. Register MDC is also saved because it is possible that a previous routine's Multiply or Divide instruction was interrupted while in progress. In this case the information about how to restart the instruction is contained in this register. Register MDC must be cleared to be correctly initialized for a subsequent multiplication or division. The old MDC contents must be popped from the stack before the RETI instruction is executed.

For a division the user must first move the dividend into the MD register. If a 16/16-bit division is specified, only the low portion of register MD must be loaded. The result is also stored into register MD. The low portion (MDL) contains the integer result of the division, while the high portion (MDH) contains the remainder.

The following instruction sequence performs a 32 by 16-bit division:

```
MOV      MDH, R1      ;Move dividend to MD register. Sets MDRIU
MOV      MDL, R2      ;Move low portion to MD
DIV      R3           ;Divide 32/16 signed, R3 holds divisor
JMPCR   cc_V, ERROR ;Test for divide overflow
MOV      R3, MDH      ;Move remainder to R3
MOV      R4, MDL      ;Move integer result to R4. Clears MDRIU
```

Whenever a multiply or divide instruction is interrupted while in progress, the address of the interrupted instruction is pushed onto the stack and the MULIP flag in the PSW of the interrupting routine is set. When the interrupt routine is exited with the RETI instruction, this bit is implicitly tested before the old PSW is popped from the stack. If MULIP = '1' the multiply/divide instruction is re-read from the location popped from the stack (return address) and will be completed after the RETI instruction has been executed.

*Note: The MULIP flag is part of the **context of the interrupted task**. When the interrupting routine does not return to the interrupted task (e.g. scheduler switches to another task) the MULIP flag must be set or cleared according to the context of the task that is switched to.*

## BCD Calculations

No direct support for BCD calculations is provided in the C161CS/JC/JI. BCD calculations are performed by converting BCD data to binary data, performing the desired calculations using standard data types, and converting the result back to BCD data. Due to the enhanced performance of division instructions binary data is quickly converted to BCD data through division by  $10_D$ . Conversion from BCD data to binary data is enhanced by multiple bit shift instructions. This provides similar performance compared to instructions directly supporting BCD data types, while no additional hardware is required.

## 24.1 Stack Operations

The C161CS/JC/JI supports two types of stacks. The system stack is used implicitly by the controller and is located in the internal RAM. The user stack provides stack access to the user in either the internal or external memory. Both stack types grow from high memory addresses to low memory addresses.

### Internal System Stack

A system stack is provided to store return vectors, segment pointers, and processor status for procedures and interrupt routines. A system register, SP, points to the top of the stack. This pointer is decremented when data is pushed onto the stack, and incremented when data is popped.

The internal system stack can also be used to temporarily store data or pass it between subroutines or tasks. Instructions are provided to push or pop registers on/from the system stack. However, in most cases the register banking scheme provides the best performance for passing data between multiple tasks.

*Note: The system stack allows the storage of words only. Bytes must either be converted to words or the respective other byte must be disregarded.*

*Register SP can only be loaded with even byte addresses (The LSB of SP is always '0').*

Detection of stack overflow/underflow is supported by two registers, STKOV (Stack Overflow Pointer) and STKUN (Stack Underflow Pointer). Specific system traps (Stack Overflow trap, Stack Underflow trap) will be entered whenever the SP reaches either boundary specified in these registers.

The contents of the stack pointer are compared to the contents of the overflow register, whenever the SP is DECREMENTED either by a CALL, PUSH or SUB instruction. An overflow trap will be entered, when the SP value is less than the value in the stack overflow register.

The contents of the stack pointer are compared to the contents of the underflow register, whenever the SP is INCREMENTED either by a RET, POP or ADD instruction. An underflow trap will be entered, when the SP value is greater than the value in the stack underflow register.

*Note: When a value is MOVED into the stack pointer, NO check against the overflow/underflow registers is performed.*

In many cases the user will place a software reset instruction (SRST) into the stack underflow and overflow trap service routines. This is an easy approach, which does not require special programming. However, this approach assumes that the defined internal stack is sufficient for the current software and that exceeding its upper or lower boundary represents a fatal error.

It is also possible to use the stack underflow and stack overflow traps to cache portions of a larger external stack. Only the portion of the system stack currently being used is placed into the internal memory, thus allowing a greater portion of the internal RAM to be used for program, data or register banking. This approach assumes no error but requires a set of control routines (see below).

### Circular (Virtual) Stack

This basic technique allows pushing until the overflow boundary of the internal stack is reached. At this point a portion of the stacked data must be saved into external memory to create space for further stack pushes. This is called “stack flushing”. When executing a number of return or pop instructions, the upper boundary (since the stack empties upward to higher memory locations) is reached. The entries that have been previously saved in external memory must now be restored. This is called “stack filling”. Because procedure call instructions do not continue to nest infinitely and call and return instructions alternate, flushing and filling normally occurs very infrequently. If this is not true for a given program environment, this technique should not be used because of the overhead of flushing and filling.

**The basic mechanism** is the transformation of the addresses of a virtual stack area, controlled via registers SP, STKOV and STKUN, to a defined physical stack area within the internal RAM via hardware. This virtual stack area covers all possible locations that SP can point to, i.e. 00’F000<sub>H</sub> through 00’FFFE<sub>H</sub>. STKOV and STKUN accept the same 4 KByte address range.

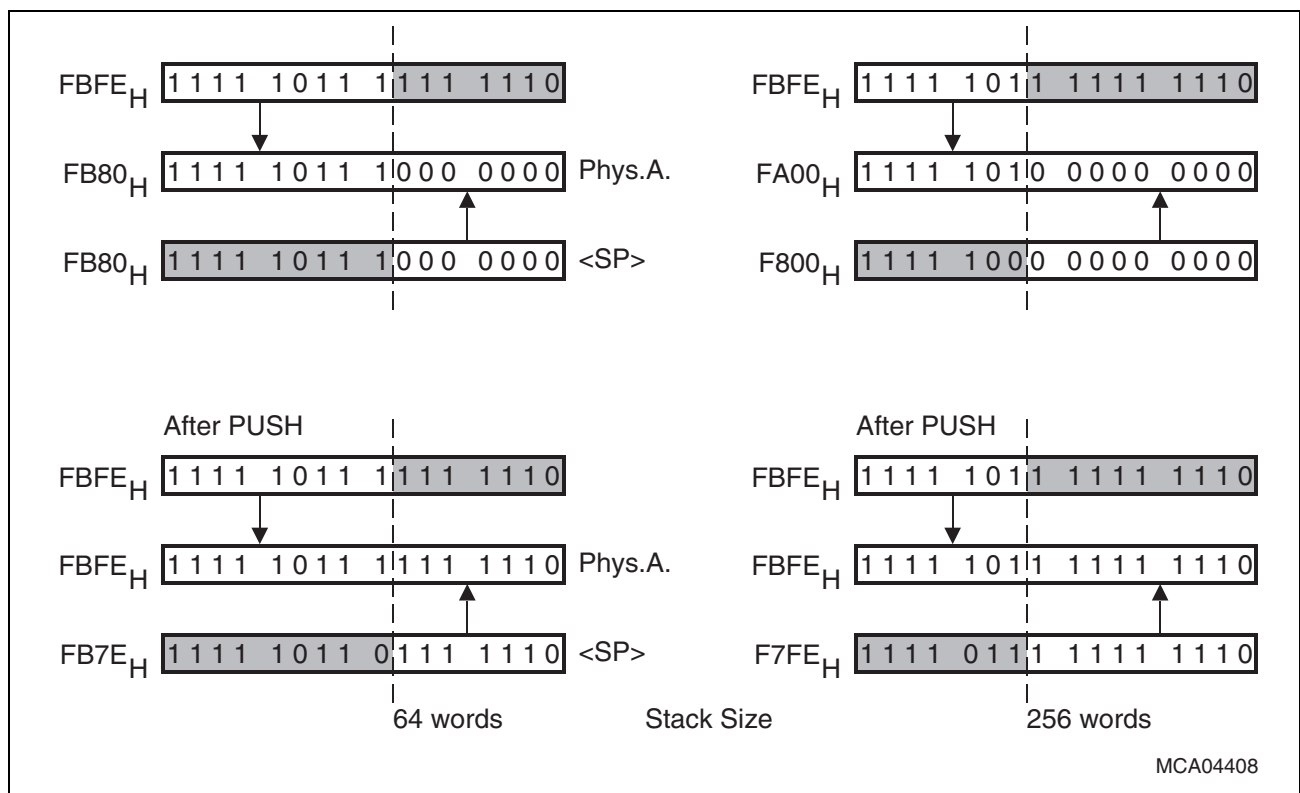
The size of the physical stack area within the internal RAM that effectively is used for standard stack operations is defined via bitfield STKSZ in register SYSCON (see below).

**Table 24-2 Circular Stack Address Transformation**

STKSZ	Stack Size (Words)	Internal RAM Addresses (Words) of Physical Stack	Significant Bits of Stack Ptr. SP
0 0 0 <sub>B</sub>	256	00’FBFE <sub>H</sub> ... 00’FA00 <sub>H</sub> (Default after Reset)	SP.8 ... SP.0
0 0 1 <sub>B</sub>	128	00’FBFE <sub>H</sub> ... 00’FB00 <sub>H</sub>	SP.7 ... SP.0
0 1 0 <sub>B</sub>	64	00’FBFE <sub>H</sub> ... 00’FB80 <sub>H</sub>	SP.6 ... SP.0
0 1 1 <sub>B</sub>	32	00’FBFE <sub>H</sub> ... 00’FBC0 <sub>H</sub>	SP.5 ... SP.0
1 0 0 <sub>B</sub>	512	00’FBFE <sub>H</sub> ... 00’F800 <sub>H</sub> (not for 1 KByte IRAM)	SP.9 ... SP.0
1 0 1 <sub>B</sub>	–	Reserved. Do not use this combination.	–
1 1 0 <sub>B</sub>	–	Reserved. Do not use this combination.	–
1 1 1 <sub>B</sub>	512 / 1024 / 1536	00’FDFF <sub>H</sub> ... 00’FX00 <sub>H</sub> (Note: No circular stack) 00’FX00 <sub>H</sub> represents the lower IRAM limit, i.e. 1 KB: 00’FA00 <sub>H</sub> , 2 KB: 00’F600 <sub>H</sub> , 3 KB: 00’F200 <sub>H</sub>	SP.11 ... SP.0

The virtual stack addresses are transformed to physical stack addresses by concatenating the significant bits of the stack pointer register SP (see [Table 24-2](#)) with the complementary most significant bits of the upper limit of the physical stack area ( $00'FBFE_H$ ). This transformation is done via hardware (see [Figure 24-1](#)).

The reset values ( $STKOV = FA00_H$ ,  $STKUN = FC00_H$ ,  $SP = FC00_H$ ,  $STKSZ = 000_B$ ) map the virtual stack area directly to the physical stack area and allow using the internal system stack without any changes, provided that the 256 word area is not exceeded.



**Figure 24-1 Physical Stack Address Generation**

The following example demonstrates the circular stack mechanism which is also an effect of this virtual stack mapping: First, register R1 is pushed onto the lowest physical stack location according to the selected maximum stack size. With the following instruction, register R2 will be pushed onto the highest physical stack location although the SP is decremented by 2 as for the previous push operation.

```
MOV      SP, #0F802H ;Set SP before last entry ...
                               ;... of physical stack of 256 words
...      ;(SP) = F802H: Physical stack addr.= FA02H
PUSH    R1      ;(SP) = F800H: Physical stack addr.= FA00H
PUSH    R2      ;(SP) = F7FEH: Physical stack addr.= FBFEH
```



The effect of the address transformation is that the physical stack addresses wrap around from the end of the defined area to its beginning. When flushing and filling the internal stack, this circular stack mechanism only requires to move that portion of stack data which is really to be re-used (i.e. the upper part of the defined stack area) instead of the whole stack area. Stack data that remain in the lower part of the internal stack need not be moved by the distance of the space being flushed or filled, as the stack pointer automatically wraps around to the beginning of the freed part of the stack area.

*Note: This circular stack technique is applicable for stack sizes of 32 to 512 words (STKSZ = '000<sub>B</sub>' to '100<sub>B</sub>'), it does not work with option STKSZ = '111<sub>B</sub>', which uses the complete internal RAM for system stack.*

*In the latter case the address transformation mechanism is deactivated.*

When a boundary is reached, the stack underflow or overflow trap is entered, where the user moves a predetermined portion of the internal stack to or from the external stack. The amount of data transferred is determined by the average stack space required by routines and the frequency of calls, traps, interrupts and returns. In most cases this will be approximately one quarter to one tenth the size of the internal stack. Once the transfer is complete, the boundary pointers are updated to reflect the newly allocated space on the internal stack. Thus, the user is free to write code without concern for the internal stack limits. Only the execution time required by the trap routines affects user programs.

The following procedure initializes the controller for usage of the circular stack mechanism:

- Specify the size of the physical system stack area within the internal RAM (bitfield STKSZ in register SYSCON).
- Define two pointers, which specify the upper and lower boundary of the external stack. These values are then tested in the stack underflow and overflow trap routines when moving data.
- Set the stack overflow pointer (STKOV) to the limit of the defined internal stack area plus six words (for the reserved space to store two interrupt entries).

The internal stack will now fill until the overflow pointer is reached. After entry into the overflow trap procedure, the top of the stack will be copied to the external memory. The internal pointers will then be modified to reflect the newly allocated space. After exiting from the trap procedure, the internal stack will wrap around to the top of the internal stack, and continue to grow until the new value of the stack overflow pointer is reached.

When the underflow pointer is reached while the stack is emptied the bottom of stack is reloaded from the external memory and the internal pointers are adjusted accordingly.



## Linear Stack

The C161CS/JC/JI also offers a linear stack option ( $STKSZ = '111_B'$ ), where the system stack may use the complete internal RAM area. This provides a large system stack without requiring procedures to handle data transfers for a circular stack. However, this method also leaves less RAM space for variables or code. The RAM area that may effectively be consumed by the system stack is defined via the  $STKUN$  and  $STKOV$  pointers. The underflow and overflow traps in this case serve for fatal error detection only.

For the linear stack option all modifiable bits of register  $SP$  are used to access the physical stack. Although the stack pointer may cover addresses from  $00'F000_H$  up to  $00'FFFE_H$  the (physical) system stack must be located within the internal RAM and therefore may only use the address range  $00'F200_H/00'F600_H/00'FA00_H$  to  $00'FDFF_H$ . It is the user's responsibility to restrict the system stack to the internal RAM range.

*Note: Avoid stack accesses below the IRAM area (ESFR space and reserved area) and within address range  $00'FE00_H$  and  $00'FFFE_H$  (SFR space). Otherwise unpredictable results will occur.*

## User Stacks

User stacks provide the ability to create task specific data stacks and to off-load data from the system stack. The user may push both bytes and words onto a user stack, but is responsible for using the appropriate instructions when popping data from the specific user stack. No hardware detection of overflow or underflow of a user stack is provided. The following addressing modes allow implementation of user stacks:

**$[-Rw]$ ,  $Rb$  or  $[-Rw]$ ,  $Rw$ :** Pre-decrement Indirect Addressing.

Used to push one byte or word onto a user stack. This mode is only available for MOV instructions and can specify any GPR as the user stack pointer.

**$Rb$ ,  $[Rw_i+]$  or  $Rw$ ,  $[Rw_i+]$ :** Post-increment Index Register Indirect Addressing.

Used to pop one byte or word from a user stack. This mode is available to most instructions, but only GPRs R0-R3 can be specified as the user stack pointer.

**$Rb$ ,  $[Rw+]$  or  $Rw$ ,  $[Rw+]$ :** Post-increment Indirect Addressing.

Used to pop one byte or word from a user stack. This mode is only available for MOV instructions and can specify any GPR as the user stack pointer.

## 24.2 Register Banking

Register banking provides the user with an extremely fast method to switch user context. A single machine cycle instruction saves the old bank and enters a new register bank. Each register bank may assign up to 16 registers. Each register bank should be allocated during coding based on the needs of each task. Once the internal memory has been partitioned into a register bank space, internal stack space and a global internal memory area, each bank pointer is then assigned. Thus, upon entry into a new task, the appropriate bank pointer is used as the operand for the SCXT (switch context) instruction. Upon exit from a task a simple POP instruction to the context pointer (CP) restores the previous task's register bank.

## 24.3 Procedure Call Entry and Exit

To support modular programming a procedure mechanism is provided to allow coding of frequently used portions of code into subroutines. The CALL and RET instructions store and restore the value of the instruction pointer (IP) on the system stack before and after a subroutine is executed.

Procedures may be called conditionally with instructions CALLA or CALLI, or be called unconditionally using instructions CALLR or CALLS.

*Note: Any data pushed onto the system stack during execution of the subroutine must be popped before the RET instruction is executed.*

### Passing Parameters on the System Stack

Parameters may be passed via the system stack through PUSH instructions before the subroutine is called, and POP instructions during execution of the subroutine. Base plus offset indirect addressing also permits access to parameters without popping these parameters from the stack during execution of the subroutine. Indirect addressing provides a mechanism of accessing data referenced by data pointers, which are passed to the subroutine.

In addition, two instructions have been implemented to allow one parameter to be passed on the system stack without additional software overhead.

The PCALL (push and call) instruction first pushes the 'reg' operand and the IP contents onto the system stack and then passes control to the subroutine specified by the 'caddr' operand.

When exiting from the subroutine, the RETP (return and pop) instruction first pops the IP and then the 'reg' operand from the system stack and returns to the calling program.

## Cross Segment Subroutine Calls

Calls to subroutines in different segments require the use of the CALLS (call inter-segment subroutine) instruction. This instruction preserves both the CSP (code segment pointer) and IP on the system stack.

Upon return from the subroutine, a RETS (return from inter-segment subroutine) instruction must be used to restore both the CSP and IP. This ensures that the next instruction after the CALLS instruction is fetched from the correct segment.

*Note: It is possible to use CALLS within the same segment, but still two words of the stack are used to store both the IP and CSP.*

## Providing Local Registers for Subroutines

For subroutines which require local storage, the following methods are provided:

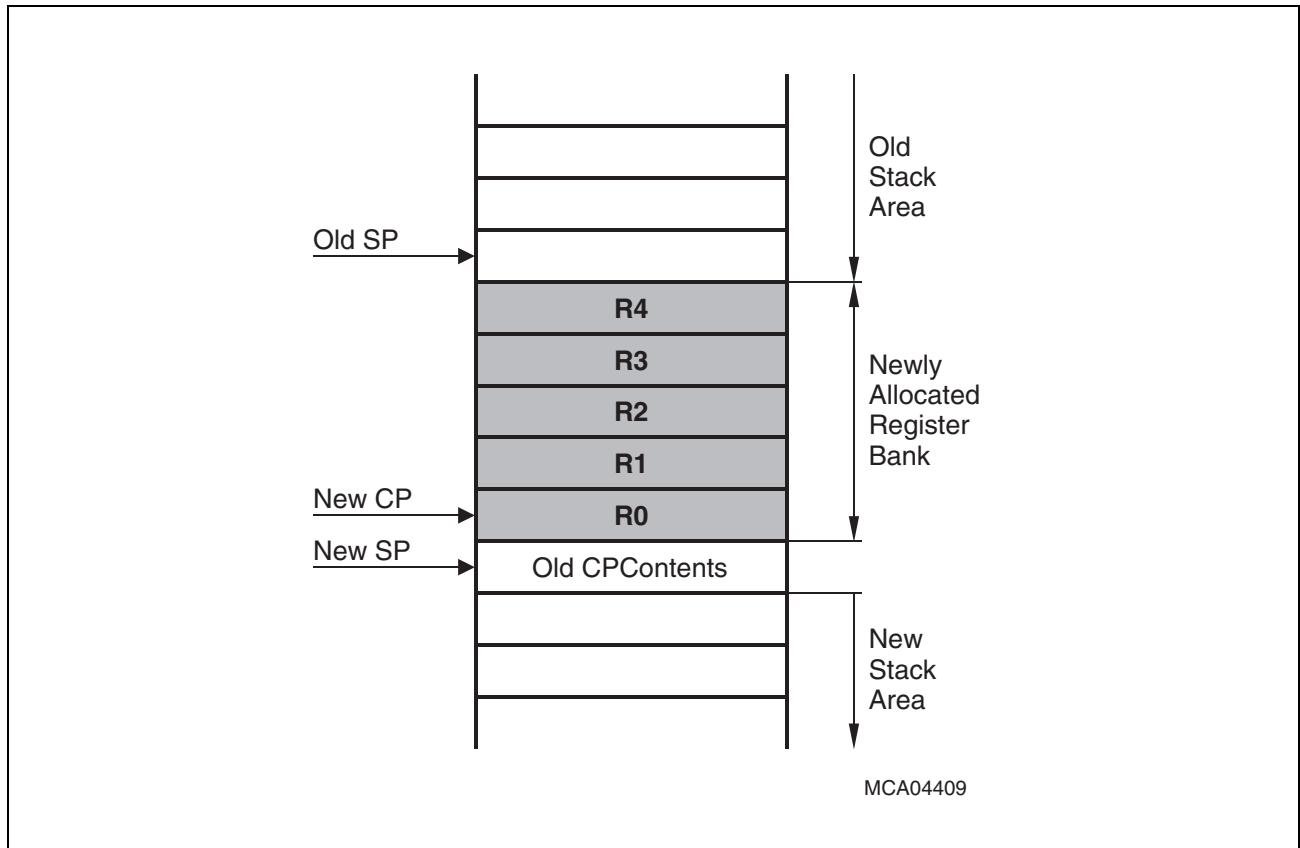
**Alternate Bank of Registers:** Upon entry into a subroutine, it is possible to specify a new set of local registers by executing the SCXT (switch context) instruction. This mechanism does not provide a method to recursively call a subroutine.

**Saving and Restoring of Registers:** To provide local registers, the contents of the registers which are required for use by the subroutine can be pushed onto the stack and the previous values be popped before returning to the calling routine. This is the most common technique used today and it does provide a mechanism to support recursive procedures. This method, however, requires two machine cycles per register stored on the system stack (one cycle to PUSH the register, and one to POP the register).

**Use of the System Stack for Local Registers:** It is possible to use the SP and CP to set up local subroutine register frames. This enables subroutines to dynamically allocate local variables as needed within two machine cycles. A local frame is allocated by simply subtracting the number of required local registers from the SP, and then moving the value of the new SP to the CP.

This operation is supported through the SCXT (switch context) instruction with the addressing mode 'reg, mem'. Using this instruction saves the old contents of the CP on the system stack and moves the value of the SP into CP (see example below). Each local register is then accessed as if it was a normal register. Upon exit from the subroutine, first the old CP must be restored by popping it from the stack and then the number of used local registers must be added to the SP to restore the allocated local space back to the system stack.

*Note: The system stack is growing downwards, while the register bank is growing upwards.*



**Figure 24-2 Local Registers**

The software to provide the local register bank for the example above is very compact:

After entering the subroutine:

```
SUB     SP, #10D      ;Free 5 words in the current system stack
SCXT   CP, SP        ;Set the new register bank pointer
```

Before exiting the subroutine:

```
POP    CP            ;Restore the old register bank
ADD    SP, #10D     ;Release the 5 words ...
                    ;...of the current system stack
```

## 24.4 Table Searching

A number of features have been included to decrease the execution time required to search tables. First, branch delays are eliminated by the branch target cache after the first iteration of the loop. Second, in non-sequentially searched tables, the enhanced performance of the ALU allows more complicated hash algorithms to be processed to obtain better table distribution. For sequentially searched tables, the auto-increment indirect addressing mode and the E (end of table) flag stored in the PSW decrease the number of overhead instructions executed in the loop.

The two examples below illustrate searching ordered tables and non-ordered tables, respectively:

```
MOV      R0, #BASE      ;Move table base into R0
LOOP:
CMP      R1, [R0+]      ;Compare target to table entry
JMPR    cc_SGT, LOOP;Test whether target has not been found
```

*Note: The last entry in the table must be greater than the largest possible target.*

```
MOV      R0, #BASE      ;Move table base into R0
LOOP:
CMP      R1, [R0+]      ;Compare target to table entry
JMPR    cc_NET, LOOP;Test whether target is not found AND ...
                        ;...the end of table has not been reached.
```

*Note: The last entry in the table must be equal to the lowest signed integer (8000<sub>H</sub>).*

## 24.5 Floating Point Support

All floating point operations are performed using software. Standard multiple precision instructions are used to perform calculations on data types that exceed the size of the ALU. Multiple bit rotate and logic instructions allow easy masking and extracting of portions of floating point numbers.

To decrease the time required to perform floating point operations, two hardware features have been implemented in the CPU core. First, the PRIOR instruction aids in normalizing floating point numbers by indicating the position of the first set bit in a GPR. This result can be used to rotate the floating point result accordingly. The second feature aids in properly rounding the result of normalized floating point numbers through the overflow (V) flag in the PSW. This flag is set when a one is shifted out of the carry bit during shift right operations. The overflow flag and the carry flag are then used to round the floating point result based on the desired rounding algorithm.

## **24.6 Peripheral Control and Interface**

All communication between peripherals and the CPU is performed either by PEC transfers to and from internal memory, or by explicitly addressing the SFRs associated with the specific peripherals. After resetting the C161CS/JC/JI all peripherals (except the watchdog timer) are disabled and initialized to default values. A desired configuration of a specific peripheral is programmed using MOV instructions of either constants or memory values to specific SFRs. Specific control flags may also be altered via bit instructions.

Once in operation, the peripheral operates autonomously until an end condition is reached at which time it requests a PEC transfer or requests CPU servicing through an interrupt routine. Information may also be polled from peripherals through read accesses to SFRs or bit operations including branch tests on specific control bits in SFRs. To ensure proper allocation of peripherals among multiple tasks, a portion of the internal memory has been made bit addressable to allow user semaphores. Instructions have also been provided to lock out tasks via software by setting or clearing user specific bits and conditionally branching based on these specific bits.

It is recommended that bit fields in control SFRs are updated using the BFLDH and BFLDL instructions or a MOV instruction to avoid undesired intermediate modes of operation which can occur, when BCLR/BSET or AND/OR instruction sequences are used.

## **24.7 Trap/Interrupt Entry and Exit**

Interrupt routines are entered when a requesting interrupt has a priority higher than the current CPU priority level. Traps are entered regardless of the current CPU priority. When either a trap or interrupt routine is entered, the state of the machine is preserved on the system stack and a branch to the appropriate trap/interrupt vector is made.

All trap and interrupt routines require the use of the RETI (return from interrupt) instruction to exit from the called routine. This instruction restores the system state from the system stack and then branches back to the location where the trap or interrupt occurred.

## 24.8 Unseparable Instruction Sequences

The instructions of the C161CS/JC/JI are very efficient (most instructions execute in one machine cycle) and even the multiplication and division are interruptible in order to minimize the response latency to interrupt requests (internal and external). In many microcontroller applications this is vital.

Some special occasions, however, require certain code sequences (e.g. semaphore handling) to be uninterruptedly to function properly. This can be provided by inhibiting interrupts during the respective code sequence by disabling and enabling them before and after the sequence. The necessary overhead may be reduced by means of the ATOMIC instruction which allows locking 1 ... 4 instructions to an unseparable code sequence, during which the interrupt system (standard interrupts and PEC requests) **and Class A Traps** ( $\overline{\text{NMI}}$ , stack overflow/underflow) are disabled. A **Class B Trap** (illegal opcode, illegal bus access, etc.), however, will interrupt the atomic sequence, since it indicates a severe hardware problem.

The interrupt inhibit caused by an ATOMIC instruction gets active immediately, i.e. no other instruction will enter the pipeline except the one that follows the ATOMIC instruction, and no interrupt request will be serviced in between. All instructions requiring multiple cycles or hold states are regarded as one instruction in this sense (e.g. MUL is one instruction). Any instruction type can be used within an unseparable code sequence.

```

ATOMIC   #3           ;The next 3 instr. are locked (No NOP requ.)
MOV      R0, #1234H   ;Instr. 1 (no other instr. enters pipeline!)
MOV      R1, #5678H   ;Instr. 2
MUL      R0, R1       ;Instr. 3: MUL regarded as one instruction
MOV      R2, MDL      ;This instruction is out of the scope ...
                        ;...of the ATOMIC instruction sequence

```

*Note: As long as any Class B trap is pending (any of the class B trap flags in register TFR is set) the ATOMIC instruction will not work. Clear the respective B trap flag at the beginning of a B trap routine if ATOMIC shall be used within the routine.*

## 24.9 Overriding the DPP Addressing Mechanism

The standard mechanism to access data locations uses one of the four data page pointers (DPPx), which selects a 16-KByte data page, and a 14-bit offset within this data page. The four DPPs allow immediate access to up to 64 KByte of data. In applications with big data arrays, especially in HLL applications using large memory models, this may require frequent reloading of the DPPs, even for single accesses.



**The EXTP (extend page) instruction** allows switching to an arbitrary data page for 1 ... 4 instructions without having to change the current DPPs.

```
EXTP    R15, #1      ;The override page number is stored in R15
MOV     R0, [R14]   ;The (14-bit) page offset is stored in R14
MOV     R1, [R13]   ;This instruction uses the std. DPP scheme!
```

**The EXTS (extend segment) instruction** allows switching to a 64 KByte segment oriented data access scheme for 1 ... 4 instructions without having to change the current DPPs. In this case all 16 bits of the operand address are used as segment offset, with the segment taken from the EXTS instruction. This greatly simplifies address calculation with continuous data like huge arrays in “C”.

```
EXTS    #15, #1     ;The override seg. is 15 (0F'0000H..0F'FFFFH)
MOV     R0, [R14]   ;The (16-bit) segment offset is stored in R14
MOV     R1, [R13]   ;This instruction uses the std. DPP scheme!
```

*Note: Instructions EXTP and EXTS inhibit interrupts the same way as ATOMIC.*

*As long as any Class B trap is pending (any of the class B trap flags in register TFR is set) the EXTend instructions will not work. Clear the respective B trap flag at the beginning of a B trap routine if EXT\* shall be used within the routine.*

### Short Addressing in the Extended SFR (ESFR) Space

The short addressing modes of the C161CS/JC/JI (REG or BITOFF) implicitly access the SFR space. The additional ESFR space would have to be accessed via long addressing modes (MEM or [Rw]). The EXTR (extend register) instruction redirects accesses in short addressing modes to the ESFR space for 1 ... 4 instructions, so the additional registers can be accessed this way, too.

The EXTPR and EXTISR instructions combine the DPP override mechanism with the redirection to the ESFR space using a single instruction.

*Note: Instructions EXTR, EXTPR and EXTISR inhibit interrupts the same way as ATOMIC.*

*The switching to the ESFR area and data page overriding is checked by the development tools or handled automatically.*

### Nested Locked Sequences

Each of the described extension instruction and the ATOMIC instruction starts an internal “extension counter” counting the effected instructions. When another extension or ATOMIC instruction is contained in the current locked sequence this counter is restarted with the value of the new instruction. This allows the construction of locked sequences longer than 4 instructions.

*Note: Interrupt latencies may be increased when using locked code sequences.*

*PEC requests are not serviced during idle mode, if the IDLE instruction is part of a locked sequence.*



## 24.10 Handling the Internal Code Memory

The Mask-ROM/OTP/Flash versions of the C161CS/JC/JI provide on-chip code memory that may store code as well as data. The lower 32 KByte of this code memory are referred to as the “internal ROM area”. Access to this internal ROM area is controlled during the reset configuration and via software. The ROM area may be mapped to segment 0, to segment 1 or the code memory may be disabled at all.

*Note: The internal ROM area always occupies an address area of 32 KByte, even if the implemented mask ROM/OTP/Flash memory is smaller than that (e.g. 8 KByte). Of course the total implemented memory may exceed 32 KBytes.*

### Code Memory Configuration During Reset

The control input pin  $\overline{EA}$  (External Access) enables the user to define the address area from which the first instructions after reset are fetched. When  $\overline{EA}$  is low ('0') during reset, the internal code memory is disabled and the first instructions are fetched from external memory. When  $\overline{EA}$  is high ('1') during reset, the internal code memory is globally enabled and the first instructions are fetched from the internal memory.

*Note: Be sure not to select internal memory access after reset on ROMless devices.*

### Mapping the Internal ROM Area

After reset the internal ROM area is mapped into segment 0, the “system segment” (00'0000<sub>H</sub> ... 00'7FFF<sub>H</sub>) as a default. This is necessary to allow the first instructions to be fetched from locations 00'0000<sub>H</sub> ff. The ROM area may be mapped to segment 1 (01'0000<sub>H</sub> ... 01'7FFF<sub>H</sub>) by setting bit ROMS1 in register SYSCON. The internal ROM area may now be accessed through the lower half of segment 1, while accesses to segment 0 will now be made to external memory. This adds flexibility to the system software. The interrupt/trap vector table, which uses locations 00'0000<sub>H</sub> through 00'01FF<sub>H</sub>, is now part of the external memory and may therefore be modified, i.e. the system software may now change interrupt/trap handlers according to the current condition of the system. The internal code memory can still be used for fixed software routines like IO drivers, math libraries, application specific invariant routines, tables, etc. This combines the advantage of an integrated non-volatile memory with the advantage of a flexible, adaptable software system.

### **Enabling and Disabling the Internal Code Memory After Reset**

If the internal code memory does not contain an appropriate startup code, the system may be booted from external memory, while the internal memory is enabled afterwards to provide access to library routines, tables, etc.

If the internal code memory only contains the startup code and/or test software, the system may be booted from internal memory, which may then be disabled, after the software has switched to executing from (e.g.) external memory, in order to free the address space occupied by the internal code memory, which is now unnecessary.

## 24.11 Pits, Traps and Mines

Although handling the internal code memory provides powerful means to enhance the overall performance and flexibility of a system, extreme care must be taken in order to avoid a system crash. Instruction memory is the most crucial resource for the C161CS/JC/JI and it must be made sure that it never runs out of it. The following precautions help to take advantage of the methods mentioned above without jeopardizing system security.

**Internal code memory access after reset:** When the first instructions are to be fetched from internal memory ( $\overline{EA} = '1'$ ), the device must contain code memory, and this must contain a valid reset vector and valid code at its destination.

**Mapping the internal ROM area to segment 1:** Due to instruction pipelining, any new ROM mapping will at the earliest become valid for the second instruction after the instruction which has changed the ROM mapping. To enable accesses to the ROM area after mapping a branch to the newly selected ROM area (JMPS) and reloading of all data page pointers is required.

This also applies to re-mapping the internal ROM area to segment 0.

**Enabling the internal code memory after reset:** When enabling the internal code memory after having booted the system from external memory, note that the C161CS/JC/JI will then access the internal memory using the current segment offset, rather than accessing external memory.

**Disabling the internal code memory after reset:** When disabling the internal code memory after having booted the system from there, note that the C161CS/JC/JI will not access external memory before a jump to segment 0 (in this case) is executed.

### General Rules

When mapping the code memory no instruction or data accesses should be made to the internal memory, otherwise unpredictable results may occur.

To avoid these problems, the instructions that configure the internal code memory should be executed from external memory or from the on-chip RAM.

Whenever the internal code memory is disabled, enabled or remapped the DPPs must be explicitly (re)loaded to enable correct data accesses to the internal and/or external memory.

## 25 The Register Set

This section summarizes all registers, which are implemented in the C161CS/JC/JI and explains the description format which is used in the chapters describing the function and layout of the SFRs.

For easy reference the registers are ordered according to two different keys (except for GPRs):

- Ordered by address, to check which register a given address references,
- Ordered by register name, to find the location of a specific register.

### 25.1 Register Description Format

In the respective chapters the function and the layout of the SFRs is described in a specific format which provides a number of details about the described special function register. The example below shows how to interpret these details.

#### REG\_NAME

Name of Register		E/SFR (A16 <sub>H</sub> /A8 <sub>H</sub> )						Reset Value: **** <sub>H</sub>							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<empty for byte registers>							std	hw	read/ write bit	read bit	write bit	bitfield			
-	-	-	-	-	-	-	-	-	rw	rwh	rw	r	w	rw	

Bit	Function
bit(field)name	Explanation of bit(field)name <i>Description of the functions controlled by the different possible values of this bit(field).</i>

#### Elements

REG_NAME	Short name of this register
A16/A8	Long 16-bit address/Short 8-bit address
SFR/ESFR/XReg	Register space (SFR, ESFR or External/XBUS Register)
(**) **	Register contents after reset <b>0/1</b> : defined value, <b>'X'</b> : undefined, <b>'U'</b> : unchanged (undefined ( <b>'X'</b> ) after power up)
r/w	Access modes: can be <u>r</u> ead and/or <u>w</u> rite
xh	Bits that are set/cleared by hardware are marked with a shaded access box and an <b>'h'</b> in it.

## 25.2 CPU General Purpose Registers (GPRs)

The GPRs form the register bank that the CPU works with. This register bank may be located anywhere within the internal RAM via the Context Pointer (CP). Due to the addressing mechanism, GPR banks can only reside within the internal RAM. All GPRs are bit-addressable.

**Table 25-1 General Purpose Word Registers**

<b>Name</b>	<b>Physical Address</b>	<b>8-bit Address</b>	<b>Description</b>	<b>Reset Value</b>
R0	(CP) + 0	F0 <sub>H</sub>	CPU General Purpose (Word) Reg. R0	UUUU <sub>H</sub>
R1	(CP) + 2	F1 <sub>H</sub>	CPU General Purpose (Word) Reg. R1	UUUU <sub>H</sub>
R2	(CP) + 4	F2 <sub>H</sub>	CPU General Purpose (Word) Reg. R2	UUUU <sub>H</sub>
R3	(CP) + 6	F3 <sub>H</sub>	CPU General Purpose (Word) Reg. R3	UUUU <sub>H</sub>
R4	(CP) + 8	F4 <sub>H</sub>	CPU General Purpose (Word) Reg. R4	UUUU <sub>H</sub>
R5	(CP) + 10	F5 <sub>H</sub>	CPU General Purpose (Word) Reg. R5	UUUU <sub>H</sub>
R6	(CP) + 12	F6 <sub>H</sub>	CPU General Purpose (Word) Reg. R6	UUUU <sub>H</sub>
R7	(CP) + 14	F7 <sub>H</sub>	CPU General Purpose (Word) Reg. R7	UUUU <sub>H</sub>
R8	(CP) + 16	F8 <sub>H</sub>	CPU General Purpose (Word) Reg. R8	UUUU <sub>H</sub>
R9	(CP) + 18	F9 <sub>H</sub>	CPU General Purpose (Word) Reg. R9	UUUU <sub>H</sub>
R10	(CP) + 20	FA <sub>H</sub>	CPU General Purpose (Word) Reg. R10	UUUU <sub>H</sub>
R11	(CP) + 22	FB <sub>H</sub>	CPU General Purpose (Word) Reg. R11	UUUU <sub>H</sub>
R12	(CP) + 24	FC <sub>H</sub>	CPU General Purpose (Word) Reg. R12	UUUU <sub>H</sub>
R13	(CP) + 26	FD <sub>H</sub>	CPU General Purpose (Word) Reg. R13	UUUU <sub>H</sub>
R14	(CP) + 28	FE <sub>H</sub>	CPU General Purpose (Word) Reg. R14	UUUU <sub>H</sub>
R15	(CP) + 30	FF <sub>H</sub>	CPU General Purpose (Word) Reg. R15	UUUU <sub>H</sub>

The first 8 GPRs (R7 ... R0) may also be accessed byte-wise. Other than with SFRs, writing to a GPR byte does not affect the other byte of the respective GPR.

The respective halves of the byte-accessible registers receive special names:

**Table 25-2 General Purpose Byte Registers**

<b>Name</b>	<b>Physical Address</b>	<b>8-bit Address</b>	<b>Description</b>	<b>Reset Value</b>
RL0	(CP) + 0	F0 <sub>H</sub>	CPU General Purpose (Byte) Reg. RL0	UU <sub>H</sub>
RH0	(CP) + 1	F1 <sub>H</sub>	CPU General Purpose (Byte) Reg. RH0	UU <sub>H</sub>
RL1	(CP) + 2	F2 <sub>H</sub>	CPU General Purpose (Byte) Reg. RL1	UU <sub>H</sub>
RH1	(CP) + 3	F3 <sub>H</sub>	CPU General Purpose (Byte) Reg. RH1	UU <sub>H</sub>
RL2	(CP) + 4	F4 <sub>H</sub>	CPU General Purpose (Byte) Reg. RL2	UU <sub>H</sub>
RH2	(CP) + 5	F5 <sub>H</sub>	CPU General Purpose (Byte) Reg. RH2	UU <sub>H</sub>
RL3	(CP) + 6	F6 <sub>H</sub>	CPU General Purpose (Byte) Reg. RL3	UU <sub>H</sub>
RH3	(CP) + 7	F7 <sub>H</sub>	CPU General Purpose (Byte) Reg. RH3	UU <sub>H</sub>
RL4	(CP) + 8	F8 <sub>H</sub>	CPU General Purpose (Byte) Reg. RL4	UU <sub>H</sub>
RH4	(CP) + 9	F9 <sub>H</sub>	CPU General Purpose (Byte) Reg. RH4	UU <sub>H</sub>
RL5	(CP) + 10	FA <sub>H</sub>	CPU General Purpose (Byte) Reg. RL5	UU <sub>H</sub>
RH5	(CP) + 11	FB <sub>H</sub>	CPU General Purpose (Byte) Reg. RH5	UU <sub>H</sub>
RL6	(CP) + 12	FC <sub>H</sub>	CPU General Purpose (Byte) Reg. RL6	UU <sub>H</sub>
RH6	(CP) + 13	FD <sub>H</sub>	CPU General Purpose (Byte) Reg. RH6	UU <sub>H</sub>
RL7	(CP) + 14	FE <sub>H</sub>	CPU General Purpose (Byte) Reg. RL7	UU <sub>H</sub>
RH7	(CP) + 15	FF <sub>H</sub>	CPU General Purpose (Byte) Reg. RH7	UU <sub>H</sub>

### 25.3 Special Function Registers Ordered by Name

**Table 25-3** lists all SFRs which are implemented in the C161CS/JC/JI in alphabetical order.

**Bit-addressable** SFRs are marked with the letter “**b**” in column “Name”.

SFRs within the **extended SFR-space** (ESFRs) are marked with the letter “**E**” in column “Physical Address”. Registers within on-chip X-Peripherals are marked with the letter “**X**” in column “Physical Address”.

**Table 25-3 C161CS/JC/JI Registers, Ordered by Name**

Name	Physical Address	8-bit Addr.	Description	Reset Value
<b>ADCIC</b> <b>b</b>	FF98 <sub>H</sub>	CC <sub>H</sub>	A/D Converter End of Conversion Interrupt Control Register	0000 <sub>H</sub>
<b>ADCON</b> <b>b</b>	FFA0 <sub>H</sub>	D0 <sub>H</sub>	A/D Converter Control Register	0000 <sub>H</sub>
<b>ADDAT</b>	FEA0 <sub>H</sub>	50 <sub>H</sub>	A/D Converter Result Register	0000 <sub>H</sub>
<b>ADDAT2</b>	F0A0 <sub>H</sub> <b>E</b>	50 <sub>H</sub>	A/D Converter 2 Result Register	0000 <sub>H</sub>
<b>ADDRSEL1</b>	FE18 <sub>H</sub>	0C <sub>H</sub>	Address Select Register 1	0000 <sub>H</sub>
<b>ADDRSEL2</b>	FE1A <sub>H</sub>	0D <sub>H</sub>	Address Select Register 2	0000 <sub>H</sub>
<b>ADDRSEL3</b>	FE1C <sub>H</sub>	0E <sub>H</sub>	Address Select Register 3	0000 <sub>H</sub>
<b>ADDRSEL4</b>	FE1E <sub>H</sub>	0F <sub>H</sub>	Address Select Register 4	0000 <sub>H</sub>
<b>ADEIC</b> <b>b</b>	FF9A <sub>H</sub>	CD <sub>H</sub>	A/D Converter Overrun Error Interrupt Control Register	0000 <sub>H</sub>
<b>BUFFCON</b>	EB24 <sub>H</sub> <b>X</b>	–	SDLM Buffer Control Register	0000 <sub>H</sub>
<b>BUFFSTAT</b>	EB1C <sub>H</sub> <b>X</b>	–	SDLM Buffer Status Register	0000 <sub>H</sub>
<b>BUSCON0</b> <b>b</b>	FF0C <sub>H</sub>	86 <sub>H</sub>	Bus Configuration Register 0	0000 <sub>H</sub>
<b>BUSCON1</b> <b>b</b>	FF14 <sub>H</sub>	8A <sub>H</sub>	Bus Configuration Register 1	0000 <sub>H</sub>
<b>BUSCON2</b> <b>b</b>	FF16 <sub>H</sub>	8B <sub>H</sub>	Bus Configuration Register 2	0000 <sub>H</sub>
<b>BUSCON3</b> <b>b</b>	FF18 <sub>H</sub>	8C <sub>H</sub>	Bus Configuration Register 3	0000 <sub>H</sub>
<b>BUSCON4</b> <b>b</b>	FF1A <sub>H</sub>	8D <sub>H</sub>	Bus Configuration Register 4	0000 <sub>H</sub>
<b>BUSSTAT</b>	EB20 <sub>H</sub> <b>X</b>	–	SDLM Bus Status Register	0000 <sub>H</sub>
<b>C1BTR</b>	EF04 <sub>H</sub> <b>X</b>	–	CAN1 Bit Timing Register	UUUU <sub>H</sub>
<b>C1CSR</b>	EF00 <sub>H</sub> <b>X</b>	–	CAN1 Control/Status Register	XX01 <sub>H</sub>
<b>C1GMS</b>	EF06 <sub>H</sub> <b>X</b>	–	CAN1 Global Mask Short	UFUU <sub>H</sub>
<b>C1LAR<sub>n</sub></b>	EFn4 <sub>H</sub> <b>X</b>	–	CAN1 Lower Arbitration Register (msg. n)	UUUU <sub>H</sub>
<b>C1LGML</b>	EF0A <sub>H</sub> <b>X</b>	–	CAN1 Lower Global Mask Long	UUUU <sub>H</sub>

**Table 25-3 C161CS/JC/JI Registers, Ordered by Name (cont'd)**

Name	Physical Address	8-bit Addr.	Description	Reset Value
<b>C1LMLM</b>	EF0E <sub>H</sub> X	–	CAN1 Lower Mask of Last Message	UUUU <sub>H</sub>
<b>C1MCFGn</b>	EFn6 <sub>H</sub> X	–	CAN1 Message Configuration Register (msg. n)	UU <sub>H</sub>
<b>C1MCRn</b>	EFn0 <sub>H</sub> X	–	CAN1 Message Ctrl. Reg. (msg. n)	UUUU <sub>H</sub>
<b>C1PCIR</b>	EF02 <sub>H</sub> X	–	CAN1 Port Control and Interrupt Register	XXXX <sub>H</sub>
<b>C1UARn</b>	EFn2 <sub>H</sub> X	–	CAN1 Upper Arbitration Reg. (msg. n)	UUUU <sub>H</sub>
<b>C1UGML</b>	EF08 <sub>H</sub> X	–	CAN1 Upper Global Mask Long	UUUU <sub>H</sub>
<b>C1UMLM</b>	EF0C <sub>H</sub> X	–	CAN1 Upper Mask of Last Message	UUUU <sub>H</sub>
<b>C2BTR</b>	EE04 <sub>H</sub> X	–	CAN2 Bit Timing Register	UUUU <sub>H</sub>
<b>C2CSR</b>	EE00 <sub>H</sub> X	–	CAN2 Control/Status Register	XX01 <sub>H</sub>
<b>C2GMS</b>	EE06 <sub>H</sub> X	–	CAN2 Global Mask Short	UFUU <sub>H</sub>
<b>C2LARn</b>	EEn4 <sub>H</sub> X	–	CAN2 Lower Arbitration Register (msg. n)	UUUU <sub>H</sub>
<b>C2LGML</b>	EE0A <sub>H</sub> X	–	CAN2 Lower Global Mask Long	UUUU <sub>H</sub>
<b>C2LMLM</b>	EE0E <sub>H</sub> X	–	CAN2 Lower Mask of Last Message	UUUU <sub>H</sub>
<b>C2MCFGn</b>	EEn6 <sub>H</sub> X	–	CAN2 Message Configuration Register (msg. n)	UU <sub>H</sub>
<b>C2MCRn</b>	EEn0 <sub>H</sub> X	–	CAN2 Message Ctrl. Reg. (msg. n)	UUUU <sub>H</sub>
<b>C2PCIR</b>	EE02 <sub>H</sub> X	–	CAN2 Port Control and Interrupt Register	XXXX <sub>H</sub>
<b>C2UARn</b>	EEn2 <sub>H</sub> X	–	CAN2 Upper Arbitration Reg. (msg. n)	UUUU <sub>H</sub>
<b>C2UGML</b>	EE08 <sub>H</sub> X	–	CAN2 Upper Global Mask Long	UUUU <sub>H</sub>
<b>C2UMLM</b>	EE0C <sub>H</sub> X	–	CAN2 Upper Mask of Last Message	UUUU <sub>H</sub>
<b>CAPREL</b>	FE4A <sub>H</sub>	25 <sub>H</sub>	GPT2 Capture/Reload Register	0000 <sub>H</sub>
<b>CC0</b>	FE80 <sub>H</sub>	40 <sub>H</sub>	CAPCOM Register 0	0000 <sub>H</sub>
<b>CC0IC</b>	<b>b</b> FF78 <sub>H</sub>	BC <sub>H</sub>	CAPCOM Register 0 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC1</b>	FE82 <sub>H</sub>	41 <sub>H</sub>	CAPCOM Register 1	0000 <sub>H</sub>
<b>CC10</b>	FE94 <sub>H</sub>	4A <sub>H</sub>	CAPCOM Register 10	0000 <sub>H</sub>
<b>CC10IC</b>	<b>b</b> FF8C <sub>H</sub>	C6 <sub>H</sub>	CAPCOM Register 10 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC11</b>	FE96 <sub>H</sub>	4B <sub>H</sub>	CAPCOM Register 11	0000 <sub>H</sub>
<b>CC11IC</b>	<b>b</b> FF8E <sub>H</sub>	C7 <sub>H</sub>	CAPCOM Register 11 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC12</b>	FE98 <sub>H</sub>	4C <sub>H</sub>	CAPCOM Register 12	0000 <sub>H</sub>



**Table 25-3 C161CS/JC/JI Registers, Ordered by Name (cont'd)**

<b>Name</b>	<b>Physical Address</b>	<b>8-bit Addr.</b>	<b>Description</b>	<b>Reset Value</b>
<b>CC12IC b</b>	FF90 <sub>H</sub>	C8 <sub>H</sub>	CAPCOM Register 12 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC13</b>	FE9A <sub>H</sub>	4D <sub>H</sub>	CAPCOM Register 13	0000 <sub>H</sub>
<b>CC13IC b</b>	FF92 <sub>H</sub>	C9 <sub>H</sub>	CAPCOM Register 13 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC14</b>	FE9C <sub>H</sub>	4E <sub>H</sub>	CAPCOM Register 14	0000 <sub>H</sub>
<b>CC14IC b</b>	FF94 <sub>H</sub>	CA <sub>H</sub>	CAPCOM Register 14 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC15</b>	FE9E <sub>H</sub>	4F <sub>H</sub>	CAPCOM Register 15	0000 <sub>H</sub>
<b>CC15IC b</b>	FF96 <sub>H</sub>	CB <sub>H</sub>	CAPCOM Register 15 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC16</b>	FE60 <sub>H</sub>	30 <sub>H</sub>	CAPCOM Register 16	0000 <sub>H</sub>
<b>CC16IC b</b>	F160 <sub>H</sub> <b>E</b>	B0 <sub>H</sub>	CAPCOM Register 16 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC17</b>	FE62 <sub>H</sub>	31 <sub>H</sub>	CAPCOM Register 17	0000 <sub>H</sub>
<b>CC17IC b</b>	F162 <sub>H</sub> <b>E</b>	B1 <sub>H</sub>	CAPCOM Register 17 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC18</b>	FE64 <sub>H</sub>	32 <sub>H</sub>	CAPCOM Register 18	0000 <sub>H</sub>
<b>CC18IC b</b>	F164 <sub>H</sub> <b>E</b>	B2 <sub>H</sub>	CAPCOM Register 18 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC19</b>	FE66 <sub>H</sub>	33 <sub>H</sub>	CAPCOM Register 19	0000 <sub>H</sub>
<b>CC19IC b</b>	F166 <sub>H</sub> <b>E</b>	B3 <sub>H</sub>	CAPCOM Register 19 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC1IC b</b>	FF7A <sub>H</sub>	BD <sub>H</sub>	CAPCOM Register 1 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC2</b>	FE84 <sub>H</sub>	42 <sub>H</sub>	CAPCOM Register 2	0000 <sub>H</sub>
<b>CC20</b>	FE68 <sub>H</sub>	34 <sub>H</sub>	CAPCOM Register 20	0000 <sub>H</sub>
<b>CC20IC b</b>	F168 <sub>H</sub> <b>E</b>	B4 <sub>H</sub>	CAPCOM Register 20 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC21</b>	FE6A <sub>H</sub>	35 <sub>H</sub>	CAPCOM Register 21	0000 <sub>H</sub>
<b>CC21IC b</b>	F16A <sub>H</sub> <b>E</b>	B5 <sub>H</sub>	CAPCOM Register 21 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC22</b>	FE6C <sub>H</sub>	36 <sub>H</sub>	CAPCOM Register 22	0000 <sub>H</sub>
<b>CC22IC b</b>	F16C <sub>H</sub> <b>E</b>	B6 <sub>H</sub>	CAPCOM Register 22 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC23</b>	FE6E <sub>H</sub>	37 <sub>H</sub>	CAPCOM Register 23	0000 <sub>H</sub>
<b>CC23IC b</b>	F16E <sub>H</sub> <b>E</b>	B7 <sub>H</sub>	CAPCOM Register 23 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC24</b>	FE70 <sub>H</sub>	38 <sub>H</sub>	CAPCOM Register 24	0000 <sub>H</sub>
<b>CC24IC b</b>	F170 <sub>H</sub> <b>E</b>	B8 <sub>H</sub>	CAPCOM Register 24 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC25</b>	FE72 <sub>H</sub>	39 <sub>H</sub>	CAPCOM Register 25	0000 <sub>H</sub>
<b>CC25IC b</b>	F172 <sub>H</sub> <b>E</b>	B9 <sub>H</sub>	CAPCOM Register 25 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC26</b>	FE74 <sub>H</sub>	3A <sub>H</sub>	CAPCOM Register 26	0000 <sub>H</sub>

**Table 25-3 C161CS/JC/JI Registers, Ordered by Name (cont'd)**

Name	Physical Address	8-bit Addr.	Description	Reset Value
<b>CC26IC</b>	<b>b</b> F174 <sub>H</sub> <b>E</b>	BA <sub>H</sub>	CAPCOM Register 26 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC27</b>	FE76 <sub>H</sub>	3B <sub>H</sub>	CAPCOM Register 27	0000 <sub>H</sub>
<b>CC27IC</b>	<b>b</b> F176 <sub>H</sub> <b>E</b>	BB <sub>H</sub>	CAPCOM Register 27 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC28</b>	FE78 <sub>H</sub>	3C <sub>H</sub>	CAPCOM Register 28	0000 <sub>H</sub>
<b>CC28IC</b>	<b>b</b> F178 <sub>H</sub> <b>E</b>	BC <sub>H</sub>	CAPCOM Register 28 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC29</b>	FE7A <sub>H</sub>	3D <sub>H</sub>	CAPCOM Register 29	0000 <sub>H</sub>
<b>CC29IC</b>	<b>b</b> F184 <sub>H</sub> <b>E</b>	C2 <sub>H</sub>	CAPCOM Register 29 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC2IC</b>	<b>b</b> FF7C <sub>H</sub>	BE <sub>H</sub>	CAPCOM Register 2 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC3</b>	FE86 <sub>H</sub>	43 <sub>H</sub>	CAPCOM Register 3	0000 <sub>H</sub>
<b>CC30</b>	FE7C <sub>H</sub>	3E <sub>H</sub>	CAPCOM Register 30	0000 <sub>H</sub>
<b>CC30IC</b>	<b>b</b> F18C <sub>H</sub> <b>E</b>	C6 <sub>H</sub>	CAPCOM Register 30 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC31</b>	FE7E <sub>H</sub>	3F <sub>H</sub>	CAPCOM Register 31	0000 <sub>H</sub>
<b>CC31IC</b>	<b>b</b> F194 <sub>H</sub> <b>E</b>	CA <sub>H</sub>	CAPCOM Register 31 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC3IC</b>	<b>b</b> FF7E <sub>H</sub>	BF <sub>H</sub>	CAPCOM Register 3 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC4</b>	FE88 <sub>H</sub>	44 <sub>H</sub>	CAPCOM Register 4	0000 <sub>H</sub>
<b>CC4IC</b>	<b>b</b> FF80 <sub>H</sub>	C0 <sub>H</sub>	CAPCOM Register 4 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC5</b>	FE8A <sub>H</sub>	45 <sub>H</sub>	CAPCOM Register 5	0000 <sub>H</sub>
<b>CC5IC</b>	<b>b</b> FF82 <sub>H</sub>	C1 <sub>H</sub>	CAPCOM Register 5 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC6</b>	FE8C <sub>H</sub>	46 <sub>H</sub>	CAPCOM Register 6	0000 <sub>H</sub>
<b>CC6IC</b>	<b>b</b> FF84 <sub>H</sub>	C2 <sub>H</sub>	CAPCOM Register 6 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC7</b>	FE8E <sub>H</sub>	47 <sub>H</sub>	CAPCOM Register 7	0000 <sub>H</sub>
<b>CC7IC</b>	<b>b</b> FF86 <sub>H</sub>	C3 <sub>H</sub>	CAPCOM Register 7 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC8</b>	FE90 <sub>H</sub>	48 <sub>H</sub>	CAPCOM Register 8	0000 <sub>H</sub>
<b>CC8IC</b>	<b>b</b> FF88 <sub>H</sub>	C4 <sub>H</sub>	CAPCOM Register 8 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC9</b>	FE92 <sub>H</sub>	49 <sub>H</sub>	CAPCOM Register 9	0000 <sub>H</sub>
<b>CC9IC</b>	<b>b</b> FF8A <sub>H</sub>	C5 <sub>H</sub>	CAPCOM Register 9 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CCM0</b>	<b>b</b> FF52 <sub>H</sub>	A9 <sub>H</sub>	CAPCOM Mode Control Register 0	0000 <sub>H</sub>
<b>CCM1</b>	<b>b</b> FF54 <sub>H</sub>	AA <sub>H</sub>	CAPCOM Mode Control Register 1	0000 <sub>H</sub>
<b>CCM2</b>	<b>b</b> FF56 <sub>H</sub>	AB <sub>H</sub>	CAPCOM Mode Control Register 2	0000 <sub>H</sub>
<b>CCM3</b>	<b>b</b> FF58 <sub>H</sub>	AC <sub>H</sub>	CAPCOM Mode Control Register 3	0000 <sub>H</sub>

**Table 25-3 C161CS/JC/JI Registers, Ordered by Name (cont'd)**

Name	Physical Address	8-bit Addr.	Description	Reset Value
<b>CCM4</b>	<b>b</b> FF22 <sub>H</sub>	91 <sub>H</sub>	CAPCOM Mode Control Register 4	0000 <sub>H</sub>
<b>CCM5</b>	<b>b</b> FF24 <sub>H</sub>	92 <sub>H</sub>	CAPCOM Mode Control Register 5	0000 <sub>H</sub>
<b>CCM6</b>	<b>b</b> FF26 <sub>H</sub>	93 <sub>H</sub>	CAPCOM Mode Control Register 6	0000 <sub>H</sub>
<b>CCM7</b>	<b>b</b> FF28 <sub>H</sub>	94 <sub>H</sub>	CAPCOM Mode Control Register 7	0000 <sub>H</sub>
<b>CLKDIV</b>	EB14 <sub>H</sub> <b>X</b>	–	SDLM Clock Divider Register	0000 <sub>H</sub>
<b>CP</b>	FE10 <sub>H</sub>	08 <sub>H</sub>	CPU Context Pointer Register	FC00 <sub>H</sub>
<b>CRIC</b>	<b>b</b> FF6A <sub>H</sub>	B5 <sub>H</sub>	GPT2 CAPREL Interrupt Control Register	0000 <sub>H</sub>
<b>CSP</b>	FE08 <sub>H</sub>	04 <sub>H</sub>	CPU Code Segment Pointer Register (8 bits, not directly writeable)	0000 <sub>H</sub>
<b>DP0H</b>	<b>b</b> F102 <sub>H</sub> <b>E</b>	81 <sub>H</sub>	P0H Direction Control Register	00 <sub>H</sub>
<b>DP0L</b>	<b>b</b> F100 <sub>H</sub> <b>E</b>	80 <sub>H</sub>	P0L Direction Control Register	00 <sub>H</sub>
<b>DP1H</b>	<b>b</b> F106 <sub>H</sub> <b>E</b>	83 <sub>H</sub>	P1H Direction Control Register	00 <sub>H</sub>
<b>DP1L</b>	<b>b</b> F104 <sub>H</sub> <b>E</b>	82 <sub>H</sub>	P1L Direction Control Register	00 <sub>H</sub>
<b>DP2</b>	<b>b</b> FFC2 <sub>H</sub>	E1 <sub>H</sub>	Port 2 Direction Control Register	0000 <sub>H</sub>
<b>DP3</b>	<b>b</b> FFC6 <sub>H</sub>	E3 <sub>H</sub>	Port 3 Direction Control Register	0000 <sub>H</sub>
<b>DP4</b>	<b>b</b> FFCA <sub>H</sub>	E5 <sub>H</sub>	Port 4 Direction Control Register	00 <sub>H</sub>
<b>DP6</b>	<b>b</b> FFCE <sub>H</sub>	E7 <sub>H</sub>	Port 6 Direction Control Register	00 <sub>H</sub>
<b>DP7</b>	<b>b</b> FFD2 <sub>H</sub>	E9 <sub>H</sub>	Port 7 Direction Control Register	00 <sub>H</sub>
<b>DP9</b>	<b>b</b> FFDA <sub>H</sub>	ED <sub>H</sub>	Port 9 Direction Control Register	00 <sub>H</sub>
<b>DPP0</b>	FE00 <sub>H</sub>	00 <sub>H</sub>	CPU Data Page Pointer 0 Register (10 bits)	0000 <sub>H</sub>
<b>DPP1</b>	FE02 <sub>H</sub>	01 <sub>H</sub>	CPU Data Page Pointer 1 Register (10 bits)	0001 <sub>H</sub>
<b>DPP2</b>	FE04 <sub>H</sub>	02 <sub>H</sub>	CPU Data Page Pointer 2 Register (10 bits)	0002 <sub>H</sub>
<b>DPP3</b>	FE06 <sub>H</sub>	03 <sub>H</sub>	CPU Data Page Pointer 3 Register (10 bits)	0003 <sub>H</sub>
<b>ERRSTAT</b>	EB22 <sub>H</sub> <b>X</b>	–	SDLM Error Status Register	0000 <sub>H</sub>
<b>EXICON</b>	<b>b</b> F1C0 <sub>H</sub> <b>E</b>	E0 <sub>H</sub>	External Interrupt Control Register	0000 <sub>H</sub>
<b>EXISEL</b>	<b>b</b> F1DA <sub>H</sub> <b>E</b>	ED <sub>H</sub>	External Interrupt Source Select Register	0000 <sub>H</sub>
<b>FLAGRST</b>	EB28 <sub>H</sub> <b>X</b>	–	SDLM Flag Reset Register	0000 <sub>H</sub>
<b>FOCON</b>	<b>b</b> FFAA <sub>H</sub>	D5 <sub>H</sub>	Frequency Output Control Register	0000 <sub>H</sub>
<b>GLOBCON</b>	EB10 <sub>H</sub> <b>X</b>	–	SDLM Global Control Register	0000 <sub>H</sub>
<b>ICADR</b>	ED06 <sub>H</sub> <b>X</b>	–	IIC Address Register	0XXX <sub>H</sub>

**Table 25-3 C161CS/JC/JI Registers, Ordered by Name (cont'd)**

Name	Physical Address	8-bit Addr.	Description	Reset Value
ICCFG	ED00 <sub>H</sub> X	–	IIC Configuration Register	XX00 <sub>H</sub>
ICCON	ED02 <sub>H</sub> X	–	IIC Control Register	0000 <sub>H</sub>
ICRTB	ED08 <sub>H</sub> X	–	IIC Receive/Transmit Buffer	XX <sub>H</sub>
ICST	ED04 <sub>H</sub> X	–	IIC Status Register	0000 <sub>H</sub>
IDCHIP	F07C <sub>H</sub> E	3E <sub>H</sub>	Identifier	1XXX <sub>H</sub>
IDMANUF	F07E <sub>H</sub> E	3F <sub>H</sub>	Identifier	1820 <sub>H</sub>
IDMEM	F07A <sub>H</sub> E	3D <sub>H</sub>	Identifier	X040 <sub>H</sub>
IDPROG	F078 <sub>H</sub> E	3C <sub>H</sub>	Identifier	XXXX <sub>H</sub>
IFR	EB18 <sub>H</sub> X	–	SDLM In-Frame Response Value Register	0000 <sub>H</sub>
INTCON	EB2C <sub>H</sub> X	–	SDLM Interrupt Control Register	0000 <sub>H</sub>
IPCR	EB04 <sub>H</sub> X	–	SDLM Interface Port Connect Register	0007 <sub>H</sub>
ISNC	b F1DE <sub>H</sub> E	EF <sub>H</sub>	Interrupt Subnode Control Register	0000 <sub>H</sub>
MDC	b FF0E <sub>H</sub>	87 <sub>H</sub>	CPU Multiply Divide Control Register	0000 <sub>H</sub>
MDH	FE0C <sub>H</sub>	06 <sub>H</sub>	CPU Multiply Divide Register – High Word	0000 <sub>H</sub>
MDL	FE0E <sub>H</sub>	07 <sub>H</sub>	CPU Multiply Divide Register – Low Word	0000 <sub>H</sub>
ODP2	b F1C2 <sub>H</sub> E	E1 <sub>H</sub>	Port 2 Open Drain Control Register	0000 <sub>H</sub>
ODP3	b F1C6 <sub>H</sub> E	E3 <sub>H</sub>	Port 3 Open Drain Control Register	0000 <sub>H</sub>
ODP4	b F1CA <sub>H</sub> E	E5 <sub>H</sub>	Port 4 Open Drain Control Register	00 <sub>H</sub>
ODP6	b F1CE <sub>H</sub> E	E7 <sub>H</sub>	Port 6 Open Drain Control Register	00 <sub>H</sub>
ODP7	b F1D2 <sub>H</sub> E	E9 <sub>H</sub>	Port 7 Open Drain Control Register	00 <sub>H</sub>
ONES	b FF1E <sub>H</sub>	8F <sub>H</sub>	Constant Value 1's Register (read only)	FFFF <sub>H</sub>
P0H	b FF02 <sub>H</sub>	81 <sub>H</sub>	Port 0 High Register (Upper half of PORT0)	00 <sub>H</sub>
P0L	b FF00 <sub>H</sub>	80 <sub>H</sub>	Port 0 Low Register (Lower half of PORT0)	00 <sub>H</sub>
P1DIDIS	FEA4 <sub>H</sub>	52 <sub>H</sub>	PORT1 Digital Input Disable Register	0000 <sub>H</sub>
P1H	b FF06 <sub>H</sub>	83 <sub>H</sub>	Port 1 High Register (Upper half of PORT1)	00 <sub>H</sub>
P1L	b FF04 <sub>H</sub>	82 <sub>H</sub>	Port 1 Low Register (Lower half of PORT1)	00 <sub>H</sub>
P2	b FFC0 <sub>H</sub>	E0 <sub>H</sub>	Port 2 Register	0000 <sub>H</sub>
P3	b FFC4 <sub>H</sub>	E2 <sub>H</sub>	Port 3 Register	0000 <sub>H</sub>
P4	b FFC8 <sub>H</sub>	E4 <sub>H</sub>	Port 4 Register (7 bits)	00 <sub>H</sub>
P5	b FFA2 <sub>H</sub>	D1 <sub>H</sub>	Port 5 Register (read only)	XXXX <sub>H</sub>

**Table 25-3 C161CS/JC/JI Registers, Ordered by Name (cont'd)**

Name		Physical Address	8-bit Addr.	Description	Reset Value
<b>P5DIDIS</b>	<b>b</b>	FFA4 <sub>H</sub>	D2 <sub>H</sub>	Port 5 Digital Input Disable Register	0000 <sub>H</sub>
<b>P6</b>	<b>b</b>	FFCC <sub>H</sub>	E6 <sub>H</sub>	Port 6 Register (8 bits)	00 <sub>H</sub>
<b>P7</b>	<b>b</b>	FFD0 <sub>H</sub>	E8 <sub>H</sub>	Port 7 Register (8 bits)	00 <sub>H</sub>
<b>P9</b>	<b>b</b>	FFD8 <sub>H</sub>	EC <sub>H</sub>	Port 9 Register (8 bits)	00 <sub>H</sub>
<b>PECC0</b>		FEC0 <sub>H</sub>	60 <sub>H</sub>	PEC Channel 0 Control Register	0000 <sub>H</sub>
<b>PECC1</b>		FEC2 <sub>H</sub>	61 <sub>H</sub>	PEC Channel 1 Control Register	0000 <sub>H</sub>
<b>PECC2</b>		FEC4 <sub>H</sub>	62 <sub>H</sub>	PEC Channel 2 Control Register	0000 <sub>H</sub>
<b>PECC3</b>		FEC6 <sub>H</sub>	63 <sub>H</sub>	PEC Channel 3 Control Register	0000 <sub>H</sub>
<b>PECC4</b>		FEC8 <sub>H</sub>	64 <sub>H</sub>	PEC Channel 4 Control Register	0000 <sub>H</sub>
<b>PECC5</b>		FECA <sub>H</sub>	65 <sub>H</sub>	PEC Channel 5 Control Register	0000 <sub>H</sub>
<b>PECC6</b>		FECC <sub>H</sub>	66 <sub>H</sub>	PEC Channel 6 Control Register	0000 <sub>H</sub>
<b>PECC7</b>		FECE <sub>H</sub>	67 <sub>H</sub>	PEC Channel 7 Control Register	0000 <sub>H</sub>
<b>PICON</b>		F1C4 <sub>H</sub> <b>E</b>	E2 <sub>H</sub>	Port Input Threshold Control Register	0000 <sub>H</sub>
<b>POCON0H</b>		F082 <sub>H</sub> <b>E</b>	41 <sub>H</sub>	Port P0H Output Control Register	0000 <sub>H</sub>
<b>POCON0L</b>		F080 <sub>H</sub> <b>E</b>	40 <sub>H</sub>	Port P0L Output Control Register	0000 <sub>H</sub>
<b>POCON1H</b>		F086 <sub>H</sub> <b>E</b>	43 <sub>H</sub>	Port P1H Output Control Register	0000 <sub>H</sub>
<b>POCON1L</b>		F084 <sub>H</sub> <b>E</b>	42 <sub>H</sub>	Port P1L Output Control Register	0000 <sub>H</sub>
<b>POCON2</b>		F088 <sub>H</sub> <b>E</b>	44 <sub>H</sub>	Port P2 Output Control Register	0000 <sub>H</sub>
<b>POCON20</b>		F0AA <sub>H</sub> <b>E</b>	55 <sub>H</sub>	Dedicated Pin Output Control Register	0000 <sub>H</sub>
<b>POCON3</b>		F08A <sub>H</sub> <b>E</b>	45 <sub>H</sub>	Port P3 Output Control Register	0000 <sub>H</sub>
<b>POCON4</b>		F08C <sub>H</sub> <b>E</b>	46 <sub>H</sub>	Port P4 Output Control Register	0000 <sub>H</sub>
<b>POCON6</b>		F08E <sub>H</sub> <b>E</b>	47 <sub>H</sub>	Port P6 Output Control Register	0000 <sub>H</sub>
<b>POCON7</b>		F090 <sub>H</sub> <b>E</b>	48 <sub>H</sub>	Port P7 Output Control Register	0000 <sub>H</sub>
<b>PSW</b>	<b>b</b>	FF10 <sub>H</sub>	88 <sub>H</sub>	CPU Program Status Word	0000 <sub>H</sub>
<b>RP0H</b>	<b>b</b>	F108 <sub>H</sub> <b>E</b>	84 <sub>H</sub>	System Startup Configuration Register (read only)	XX <sub>H</sub>
<b>RSTCON</b>	<b>b</b>	F1E0 <sub>H</sub> <b>m</b>	–	Reset Control Register	00XX <sub>H</sub>
<b>RTCH</b>		F0D6 <sub>H</sub> <b>E</b>	6B <sub>H</sub>	RTC High Register	XXXX <sub>H</sub>
<b>RTCL</b>		F0D4 <sub>H</sub> <b>E</b>	6A <sub>H</sub>	RTC Low Register	XXXX <sub>H</sub>
<b>RXCNT</b>		EB4C <sub>H</sub> <b>X</b>	–	SDLM Bus Receive Byte Counter (CPU)	0000 <sub>H</sub>

**Table 25-3 C161CS/JC/JI Registers, Ordered by Name (cont'd)**

Name	Physical Address	8-bit Addr.	Description	Reset Value
<b>RXCNTB</b>	EB4A <sub>H</sub> X	–	SDLM Bus Receive Byte Counter (Bus)	0000 <sub>H</sub>
<b>RXCPU</b>	EB4E <sub>H</sub> X	–	SDLM CPU Receive Byte Counter (CPU)	0000 <sub>H</sub>
<b>RXD00..10</b>	EB4n <sub>H</sub> X	–	SDLM Receive Data Registers (n = 0..A <sub>H</sub> )	0000 <sub>H</sub>
<b>RXD10..10</b>	EB5n <sub>H</sub> X	–	SDLM Receive Data Registers (n = 0..A <sub>H</sub> )	0000 <sub>H</sub>
<b>S0BG</b>	FEB4 <sub>H</sub>	5A <sub>H</sub>	Serial Channel 0 Baud Rate Generator Reload Register	0000 <sub>H</sub>
<b>S0CON</b>	<b>b</b> FFB0 <sub>H</sub>	D8 <sub>H</sub>	Serial Channel 0 Control Register	0000 <sub>H</sub>
<b>S0EIC</b>	<b>b</b> FF70 <sub>H</sub>	B8 <sub>H</sub>	Serial Channel 0 Error Interrupt Control Register	0000 <sub>H</sub>
<b>S0RBUF</b>	FEB2 <sub>H</sub>	59 <sub>H</sub>	Serial Channel 0 Receive Buffer Register (read only)	XXXX <sub>H</sub>
<b>S0RIC</b>	<b>b</b> FF6E <sub>H</sub>	B7 <sub>H</sub>	Serial Channel 0 Receive Interrupt Control Register	0000 <sub>H</sub>
<b>S0TBIC</b>	<b>b</b> F19C <sub>H</sub> <b>E</b>	CE <sub>H</sub>	Serial Channel 0 Transmit Buffer Interrupt Control Register	0000 <sub>H</sub>
<b>S0TBUF</b>	FEB0 <sub>H</sub>	58 <sub>H</sub>	Serial Channel 0 Transmit Buffer Register	0000 <sub>H</sub>
<b>S0TIC</b>	<b>b</b> FF6C <sub>H</sub>	B6 <sub>H</sub>	Serial Channel 0 Transmit Interrupt Control Register	0000 <sub>H</sub>
<b>S1BG</b>	EDA4 <sub>H</sub> X	–	Serial Channel 1 Baud Rate Generator Reload Register	0000 <sub>H</sub>
<b>S1CON</b>	EDA6 <sub>H</sub> X	–	Serial Channel 1 Control Register	0000 <sub>H</sub>
<b>S1RBUF</b>	EDA2 <sub>H</sub> X	–	Serial Channel 1 Receive Buffer Register (read only)	XXXX <sub>H</sub>
<b>S1TBUF</b>	EDA0 <sub>H</sub> X	–	Serial Channel 1 Transmit Buffer Register	0000 <sub>H</sub>
<b>S0FPTR</b>	EB60 <sub>H</sub> X	–	SDLM Start-of-frame Pointer Register	0000 <sub>H</sub>
<b>SP</b>	FE12 <sub>H</sub>	09 <sub>H</sub>	CPU System Stack Pointer Register	FC00 <sub>H</sub>
<b>SSCBR</b>	F0B4 <sub>H</sub> <b>E</b>	5A <sub>H</sub>	SSC Baudrate Register	0000 <sub>H</sub>
<b>SSCCON</b>	<b>b</b> FF2 <sub>H</sub>	D9 <sub>H</sub>	SSC Control Register	0000 <sub>H</sub>
<b>SSCEIC</b>	<b>b</b> FF76 <sub>H</sub>	BB <sub>H</sub>	SSC Error Interrupt Control Register	0000 <sub>H</sub>
<b>SSCRB</b>	F0B2 <sub>H</sub> <b>E</b>	59 <sub>H</sub>	SSC Receive Buffer	XXXX <sub>H</sub>
<b>SSCRIC</b>	<b>b</b> FF74 <sub>H</sub>	BA <sub>H</sub>	SSC Receive Interrupt Control Register	0000 <sub>H</sub>
<b>SSCTB</b>	F0B0 <sub>H</sub> <b>E</b>	58 <sub>H</sub>	SSC Transmit Buffer	0000 <sub>H</sub>



**Table 25-3 C161CS/JC/JI Registers, Ordered by Name (cont'd)**

Name	Physical Address	8-bit Addr.	Description	Reset Value
<b>SSCTIC</b> b	FF72 <sub>H</sub>	B9 <sub>H</sub>	SSC Transmit Interrupt Control Register	0000 <sub>H</sub>
<b>STKOV</b>	FE14 <sub>H</sub>	0A <sub>H</sub>	CPU Stack Overflow Pointer Register	FA00 <sub>H</sub>
<b>STKUN</b>	FE16 <sub>H</sub>	0B <sub>H</sub>	CPU Stack Underflow Pointer Register	FC00 <sub>H</sub>
<b>SYSCON</b> b	FF12 <sub>H</sub>	89 <sub>H</sub>	CPU System Configuration Register	<sup>1)</sup> 0xx0 <sub>H</sub>
<b>SYSCON1</b> b	F1DC <sub>H</sub> E	EE <sub>H</sub>	CPU System Configuration Register 1	0000 <sub>H</sub>
<b>SYSCON2</b> b	F1D0 <sub>H</sub> E	E8 <sub>H</sub>	CPU System Configuration Register 2	0000 <sub>H</sub>
<b>SYSCON3</b> b	F1D4 <sub>H</sub> E	EA <sub>H</sub>	CPU System Configuration Register 3	0000 <sub>H</sub>
<b>T0</b>	FE50 <sub>H</sub>	28 <sub>H</sub>	CAPCOM Timer 0 Register	0000 <sub>H</sub>
<b>T01CON</b> b	FF50 <sub>H</sub>	A8 <sub>H</sub>	CAPCOM Timer 0 and Timer 1 Ctrl. Reg.	0000 <sub>H</sub>
<b>T0IC</b> b	FF9C <sub>H</sub>	CE <sub>H</sub>	CAPCOM Timer 0 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>T0REL</b>	FE54 <sub>H</sub>	2A <sub>H</sub>	CAPCOM Timer 0 Reload Register	0000 <sub>H</sub>
<b>T1</b>	FE52 <sub>H</sub>	29 <sub>H</sub>	CAPCOM Timer 1 Register	0000 <sub>H</sub>
<b>T14</b>	F0D2 <sub>H</sub> E	69 <sub>H</sub>	RTC Timer 14 Register	XXXX <sub>H</sub>
<b>T14REL</b>	F0D0 <sub>H</sub> E	68 <sub>H</sub>	RTC Timer 14 Reload Register	XXXX <sub>H</sub>
<b>T1IC</b> b	FF9E <sub>H</sub>	CF <sub>H</sub>	CAPCOM Timer 1 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>T1REL</b>	FE56 <sub>H</sub>	2B <sub>H</sub>	CAPCOM Timer 1 Reload Register	0000 <sub>H</sub>
<b>T2</b>	FE40 <sub>H</sub>	20 <sub>H</sub>	GPT1 Timer 2 Register	0000 <sub>H</sub>
<b>T2CON</b> b	FF40 <sub>H</sub>	A0 <sub>H</sub>	GPT1 Timer 2 Control Register	0000 <sub>H</sub>
<b>T2IC</b> b	FF60 <sub>H</sub>	B0 <sub>H</sub>	GPT1 Timer 2 Interrupt Control Register	0000 <sub>H</sub>
<b>T3</b>	FE42 <sub>H</sub>	21 <sub>H</sub>	GPT1 Timer 3 Register	0000 <sub>H</sub>
<b>T3CON</b> b	FF42 <sub>H</sub>	A1 <sub>H</sub>	GPT1 Timer 3 Control Register	0000 <sub>H</sub>
<b>T3IC</b> b	FF62 <sub>H</sub>	B1 <sub>H</sub>	GPT1 Timer 3 Interrupt Control Register	0000 <sub>H</sub>
<b>T4</b>	FE44 <sub>H</sub>	22 <sub>H</sub>	GPT1 Timer 4 Register	0000 <sub>H</sub>
<b>T4CON</b> b	FF44 <sub>H</sub>	A2 <sub>H</sub>	GPT1 Timer 4 Control Register	0000 <sub>H</sub>
<b>T4IC</b> b	FF64 <sub>H</sub>	B2 <sub>H</sub>	GPT1 Timer 4 Interrupt Control Register	0000 <sub>H</sub>
<b>T5</b>	FE46 <sub>H</sub>	23 <sub>H</sub>	GPT2 Timer 5 Register	0000 <sub>H</sub>
<b>T5CON</b> b	FF46 <sub>H</sub>	A3 <sub>H</sub>	GPT2 Timer 5 Control Register	0000 <sub>H</sub>
<b>T5IC</b> b	FF66 <sub>H</sub>	B3 <sub>H</sub>	GPT2 Timer 5 Interrupt Control Register	0000 <sub>H</sub>
<b>T6</b>	FE48 <sub>H</sub>	24 <sub>H</sub>	GPT2 Timer 6 Register	0000 <sub>H</sub>
<b>T6CON</b> b	FF48 <sub>H</sub>	A4 <sub>H</sub>	GPT2 Timer 6 Control Register	0000 <sub>H</sub>

**Table 25-3 C161CS/JC/JI Registers, Ordered by Name (cont'd)**

Name	Physical Address	8-bit Addr.	Description	Reset Value
<b>T6IC</b>	<b>b</b> FF68 <sub>H</sub>	B4 <sub>H</sub>	GPT2 Timer 6 Interrupt Control Register	0000 <sub>H</sub>
<b>T7</b>	F050 <sub>H</sub>	<b>E</b> 28 <sub>H</sub>	CAPCOM Timer 7 Register	0000 <sub>H</sub>
<b>T78CON</b>	<b>b</b> FF20 <sub>H</sub>	90 <sub>H</sub>	CAPCOM Timer 7 and 8 Control Register	0000 <sub>H</sub>
<b>T7IC</b>	<b>b</b> F17A <sub>H</sub>	<b>E</b> BD <sub>H</sub>	CAPCOM Timer 7 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>T7REL</b>	F054 <sub>H</sub>	<b>E</b> 2A <sub>H</sub>	CAPCOM Timer 7 Reload Register	0000 <sub>H</sub>
<b>T8</b>	F052 <sub>H</sub>	<b>E</b> 29 <sub>H</sub>	CAPCOM Timer 8 Register	0000 <sub>H</sub>
<b>T8IC</b>	<b>b</b> F17C <sub>H</sub>	<b>E</b> BE <sub>H</sub>	CAPCOM Timer 8 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>T8REL</b>	F056 <sub>H</sub>	<b>E</b> 2B <sub>H</sub>	CAPCOM Timer 8 Reload Register	0000 <sub>H</sub>
<b>TFR</b>	<b>b</b> FFAC <sub>H</sub>	D6 <sub>H</sub>	Trap Flag Register	0000 <sub>H</sub>
<b>TRANS STAT</b>	EB1E <sub>H</sub>	<b>X</b> –	SDLM Transmission Status Register	0000 <sub>H</sub>
<b>TXCNT</b>	EB3C <sub>H</sub>	<b>X</b> –	SDLM Bus Transmit Byte Counter	0000 <sub>H</sub>
<b>TXCPU</b>	EB3E <sub>H</sub>	<b>X</b> –	SDLM CPU Transmit Byte Counter	0000 <sub>H</sub>
<b>TXD0..10</b>	EB3 <sub>n</sub> <sub>H</sub>	<b>X</b> –	SDLM Transmit Data Registers (n = 0..A <sub>H</sub> )	0000 <sub>H</sub>
<b>TxDDELAY</b>	EB16 <sub>H</sub>	<b>X</b> –	SDLM Trabsceiver Delay Register	0014 <sub>H</sub>
<b>WDT</b>	FEAE <sub>H</sub>	57 <sub>H</sub>	Watchdog Timer Register (read only)	0000 <sub>H</sub>
<b>WDTCON</b>	<b>b</b> FFAE <sub>H</sub>	D7 <sub>H</sub>	Watchdog Timer Control Register	<sup>2)</sup> 00xx <sub>H</sub>
<b>XP0IC</b>	<b>b</b> F186 <sub>H</sub>	<b>E</b> C3 <sub>H</sub>	IIC Data Interrupt Control Register	0000 <sub>H</sub>
<b>XP1IC</b>	<b>b</b> F18E <sub>H</sub>	<b>E</b> C7 <sub>H</sub>	IIC Protocol Interrupt Control Register	0000 <sub>H</sub>
<b>XP2IC</b>	<b>b</b> F196 <sub>H</sub>	<b>E</b> CB <sub>H</sub>	CAN1 Interrupt Control Register	0000 <sub>H</sub>
<b>XP3IC</b>	<b>b</b> F19E <sub>H</sub>	<b>E</b> CF <sub>H</sub>	RTC/PLL/OWD Interrupt Control Register	0000 <sub>H</sub>
<b>XP4IC</b>	<b>b</b> F182 <sub>H</sub>	<b>E</b> C1 <sub>H</sub>	ASC1 Transmit Interrupt Control Register	0000 <sub>H</sub>
<b>XP5IC</b>	<b>b</b> F18A <sub>H</sub>	<b>E</b> C5 <sub>H</sub>	ASC1 Receive Interrupt Control Register	0000 <sub>H</sub>
<b>XP6IC</b>	<b>b</b> F192 <sub>H</sub>	<b>E</b> C9 <sub>H</sub>	ASC1 Error Interrupt Control Register	0000 <sub>H</sub>
<b>XP7IC</b>	<b>b</b> F19A <sub>H</sub>	<b>E</b> CD <sub>H</sub>	CAN2/SDLM Interrupt Control Register	0000 <sub>H</sub>
<b>ZEROS</b>	<b>b</b> FF1C <sub>H</sub>	8E <sub>H</sub>	Constant Value 0's Register (read only)	0000 <sub>H</sub>

<sup>1)</sup> The system configuration is selected during reset.

<sup>2)</sup> The reset value depends on the indicated reset source.



## 25.4 Special Function Registers Ordered by Address

**Table 25-4** lists all SFRs which are implemented in the C161CS/JC/JI ordered by their physical address. **Bit-addressable** SFRs are marked with the letter “**b**” in column “Name”.

SFRs within the **extended SFR-space** (ESFRs) are marked with the letter “**E**” in column “Physical Address”. Registers within on-chip X-Peripherals are marked with the letter “**X**” in column “Physical Address”.

**Table 25-4 C161CS/JC/JI Registers, Ordered by Address**

Name	Physical Address	8-bit Addr.	Description	Reset Value
IPCR	EB04 <sub>H</sub> X	–	SDLM Interface Port Connect Register	0007 <sub>H</sub>
GLOBCON	EB10 <sub>H</sub> X	–	SDLM Global Control Register	0000 <sub>H</sub>
CLKDIV	EB14 <sub>H</sub> X	–	SDLM Clock Divider Register	0000 <sub>H</sub>
TxDelay	EB16 <sub>H</sub> X	–	SDLM Trabsceiver Delay Register	0014 <sub>H</sub>
IFR	EB18 <sub>H</sub> X	–	SDLM In-Frame Response Value Register	0000 <sub>H</sub>
BUFFSTAT	EB1C <sub>H</sub> X	–	SDLM Buffer Status Register	0000 <sub>H</sub>
TRANS STAT	EB1E <sub>H</sub> X	–	SDLM Transmission Status Register	0000 <sub>H</sub>
BUSSTAT	EB20 <sub>H</sub> X	–	SDLM Bus Status Register	0000 <sub>H</sub>
ERRSTAT	EB22 <sub>H</sub> X	–	SDLM Error Status Register	0000 <sub>H</sub>
BUFFCON	EB24 <sub>H</sub> X	–	SDLM Buffer Control Register	0000 <sub>H</sub>
FLAGRST	EB28 <sub>H</sub> X	–	SDLM Flag Reset Register	0000 <sub>H</sub>
INTCON	EB2C <sub>H</sub> X	–	SDLM Interrupt Control Register	0000 <sub>H</sub>
TXCNT	EB3C <sub>H</sub> X	–	SDLM Bus Transmit Byte Counter	0000 <sub>H</sub>
TXCPU	EB3E <sub>H</sub> X	–	SDLM CPU Transmit Byte Counter	0000 <sub>H</sub>
TXD0..10	EB3 <sub>n</sub> <sub>H</sub> X	–	SDLM Transmit Data Registers (n = 0..A <sub>H</sub> )	0000 <sub>H</sub>
RXCNTB	EB4A <sub>H</sub> X	–	SDLM Bus Receive Byte Counter (Bus)	0000 <sub>H</sub>
RXCNT	EB4C <sub>H</sub> X	–	SDLM Bus Receive Byte Counter (CPU)	0000 <sub>H</sub>
RXCPU	EB4E <sub>H</sub> X	–	SDLM CPU Receive Byte Counter (CPU)	0000 <sub>H</sub>
RXD00..10	EB4 <sub>n</sub> <sub>H</sub> X	–	SDLM Receive Data Registers (n = 0..A <sub>H</sub> )	0000 <sub>H</sub>
RXD10..10	EB5 <sub>n</sub> <sub>H</sub> X	–	SDLM Receive Data Registers (n = 0..A <sub>H</sub> )	0000 <sub>H</sub>
SOFPTR	EB60 <sub>H</sub> X	–	SDLM Start-of-frame Pointer Register	0000 <sub>H</sub>
ICCFG	ED00 <sub>H</sub> X	–	IIC Configuration Register	XX00 <sub>H</sub>
ICCON	ED02 <sub>H</sub> X	–	IIC Control Register	0000 <sub>H</sub>

**Table 25-4 C161CS/JC/JI Registers, Ordered by Address (cont'd)**

Name	Physical Address	8-bit Addr.	Description	Reset Value
<b>ICST</b>	ED04 <sub>H</sub> X	–	IIC Status Register	0000 <sub>H</sub>
<b>ICADR</b>	ED06 <sub>H</sub> X	–	IIC Address Register	0XXX <sub>H</sub>
<b>ICRTB</b>	ED08 <sub>H</sub> X	–	IIC Receive/Transmit Buffer	XX <sub>H</sub>
<b>S1TBUF</b>	EDA0 <sub>H</sub> X	–	Serial Channel 1 Transmit Buffer Register	0000 <sub>H</sub>
<b>S1RBUF</b>	EDA2 <sub>H</sub> X	–	Serial Channel 1 Receive Buffer Register (read only)	XXXX <sub>H</sub>
<b>S1BG</b>	EDA4 <sub>H</sub> X	–	Serial Channel 1 Baud Rate Generator Reload Register	0000 <sub>H</sub>
<b>S1CON</b>	EDA6 <sub>H</sub> X	–	Serial Channel 1 Control Register	0000 <sub>H</sub>
<b>C2CSR</b>	EE00 <sub>H</sub> X	–	CAN2 Control/Status Register	XX01 <sub>H</sub>
<b>C2PCIR</b>	EE02 <sub>H</sub> X	–	CAN2Port Control and Interrupt Register	XXXX <sub>H</sub>
<b>C2BTR</b>	EE04 <sub>H</sub> X	–	CAN2 Bit Timing Register	UUUU <sub>H</sub>
<b>C2GMS</b>	EE06 <sub>H</sub> X	–	CAN2 Global Mask Short	UFUU <sub>H</sub>
<b>C2UGML</b>	EE08 <sub>H</sub> X	–	CAN2 Upper Global Mask Long	UUUU <sub>H</sub>
<b>C2LGML</b>	EE0A <sub>H</sub> X	–	CAN2 Lower Global Mask Long	UUUU <sub>H</sub>
<b>C2UMLM</b>	EE0C <sub>H</sub> X	–	CAN2 Upper Mask of Last Message	UUUU <sub>H</sub>
<b>C2LMLM</b>	EE0E <sub>H</sub> X	–	CAN2 Lower Mask of Last Message	UUUU <sub>H</sub>
<b>C2MCR<sub>n</sub></b>	EEn0 <sub>H</sub> X	–	CAN2 Message Ctrl. Reg. (msg. n)	UUUU <sub>H</sub>
<b>C2UAR<sub>n</sub></b>	EEn2 <sub>H</sub> X	–	CAN2 Upper Arbitration Reg. (msg. n)	UUUU <sub>H</sub>
<b>C2LAR<sub>n</sub></b>	EEn4 <sub>H</sub> X	–	CAN2 Lower Arbitration Register (msg. n)	UUUU <sub>H</sub>
<b>C2MCFG<sub>n</sub></b>	EEn6 <sub>H</sub> X	–	CAN2 Message Configuration Register (msg. n)	UU <sub>H</sub>
<b>C1CSR</b>	EF00 <sub>H</sub> X	–	CAN1 Control/Status Register	XX01 <sub>H</sub>
<b>C1PCIR</b>	EF02 <sub>H</sub> X	–	CAN1 Port Control and Interrupt Register	XXXX <sub>H</sub>
<b>C1BTR</b>	EF04 <sub>H</sub> X	–	CAN1 Bit Timing Register	UUUU <sub>H</sub>
<b>C1GMS</b>	EF06 <sub>H</sub> X	–	CAN1 Global Mask Short	UFUU <sub>H</sub>
<b>C1UGML</b>	EF08 <sub>H</sub> X	–	CAN1 Upper Global Mask Long	UUUU <sub>H</sub>
<b>C1LGML</b>	EF0A <sub>H</sub> X	–	CAN1 Lower Global Mask Long	UUUU <sub>H</sub>
<b>C1UMLM</b>	EF0C <sub>H</sub> X	–	CAN1 Upper Mask of Last Message	UUUU <sub>H</sub>
<b>C1LMLM</b>	EF0E <sub>H</sub> X	–	CAN1 Lower Mask of Last Message	UUUU <sub>H</sub>

**Table 25-4 C161CS/JC/JI Registers, Ordered by Address (cont'd)**

Name	Physical Address	8-bit Addr.	Description	Reset Value
<b>C1MCRn</b>	EFn0 <sub>H</sub> X	–	CAN1 Message Ctrl. Reg. (msg. n)	UUUU <sub>H</sub>
<b>C1UARn</b>	EFn2 <sub>H</sub> X	–	CAN1 Upper Arbitration Reg. (msg. n)	UUUU <sub>H</sub>
<b>C1LARn</b>	EFn4 <sub>H</sub> X	–	CAN1 Lower Arbitration Register (msg. n)	UUUU <sub>H</sub>
<b>C1MCFGn</b>	EFn6 <sub>H</sub> X	–	CAN1 Message Configuration Register (msg. n)	UU <sub>H</sub>
<b>T7</b>	F050 <sub>H</sub> E	28 <sub>H</sub>	CAPCOM Timer 7 Register	0000 <sub>H</sub>
<b>T8</b>	F052 <sub>H</sub> E	29 <sub>H</sub>	CAPCOM Timer 8 Register	0000 <sub>H</sub>
<b>T7REL</b>	F054 <sub>H</sub> E	2A <sub>H</sub>	CAPCOM Timer 7 Reload Register	0000 <sub>H</sub>
<b>T8REL</b>	F056 <sub>H</sub> E	2B <sub>H</sub>	CAPCOM Timer 8 Reload Register	0000 <sub>H</sub>
<b>IDPROG</b>	F078 <sub>H</sub> E	3C <sub>H</sub>	Identifier	XXXX <sub>H</sub>
<b>IDMEM</b>	F07A <sub>H</sub> E	3D <sub>H</sub>	Identifier	X040 <sub>H</sub>
<b>IDCHIP</b>	F07C <sub>H</sub> E	3E <sub>H</sub>	Identifier	1XXX <sub>H</sub>
<b>IDMANUF</b>	F07E <sub>H</sub> E	3F <sub>H</sub>	Identifier	1820 <sub>H</sub>
<b>POCON0L</b>	F080 <sub>H</sub> E	40 <sub>H</sub>	Port P0L Output Control Register	0000 <sub>H</sub>
<b>POCON0H</b>	F082 <sub>H</sub> E	41 <sub>H</sub>	Port P0H Output Control Register	0000 <sub>H</sub>
<b>POCON1L</b>	F084 <sub>H</sub> E	42 <sub>H</sub>	Port P1L Output Control Register	0000 <sub>H</sub>
<b>POCON1H</b>	F086 <sub>H</sub> E	43 <sub>H</sub>	Port P1H Output Control Register	0000 <sub>H</sub>
<b>POCON2</b>	F088 <sub>H</sub> E	44 <sub>H</sub>	Port P2 Output Control Register	0000 <sub>H</sub>
<b>POCON3</b>	F08A <sub>H</sub> E	45 <sub>H</sub>	Port P3 Output Control Register	0000 <sub>H</sub>
<b>POCON4</b>	F08C <sub>H</sub> E	46 <sub>H</sub>	Port P4 Output Control Register	0000 <sub>H</sub>
<b>POCON6</b>	F08E <sub>H</sub> E	47 <sub>H</sub>	Port P6 Output Control Register	0000 <sub>H</sub>
<b>POCON7</b>	F090 <sub>H</sub> E	48 <sub>H</sub>	Port P7 Output Control Register	0000 <sub>H</sub>
<b>ADDAT2</b>	F0A0 <sub>H</sub> E	50 <sub>H</sub>	A/D Converter 2 Result Register	0000 <sub>H</sub>
<b>POCON20</b>	F0AA <sub>H</sub> E	55 <sub>H</sub>	Dedicated Pin Output Control Register	0000 <sub>H</sub>
<b>SSCTB</b>	F0B0 <sub>H</sub> E	58 <sub>H</sub>	SSC Transmit Buffer	0000 <sub>H</sub>
<b>SSCRB</b>	F0B2 <sub>H</sub> E	59 <sub>H</sub>	SSC Receive Buffer	XXXX <sub>H</sub>
<b>SSCBR</b>	F0B4 <sub>H</sub> E	5A <sub>H</sub>	SSC Baudrate Register	0000 <sub>H</sub>
<b>T14REL</b>	F0D0 <sub>H</sub> E	68 <sub>H</sub>	RTC Timer 14 Reload Register	XXXX <sub>H</sub>
<b>T14</b>	F0D2 <sub>H</sub> E	69 <sub>H</sub>	RTC Timer 14 Register	XXXX <sub>H</sub>
<b>RTCL</b>	F0D4 <sub>H</sub> E	6A <sub>H</sub>	RTC Low Register	XXXX <sub>H</sub>

**Table 25-4 C161CS/JC/JI Registers, Ordered by Address (cont'd)**

Name	Physical Address	8-bit Addr.	Description	Reset Value
<b>RTCH</b>	F0D6 <sub>H</sub> E	6B <sub>H</sub>	RTC High Register	XXXX <sub>H</sub>
<b>DP0L</b>	<b>b</b> F100 <sub>H</sub> E	80 <sub>H</sub>	P0L Direction Control Register	00 <sub>H</sub>
<b>DP0H</b>	<b>b</b> F102 <sub>H</sub> E	81 <sub>H</sub>	P0H Direction Control Register	00 <sub>H</sub>
<b>DP1L</b>	<b>b</b> F104 <sub>H</sub> E	82 <sub>H</sub>	P1L Direction Control Register	00 <sub>H</sub>
<b>DP1H</b>	<b>b</b> F106 <sub>H</sub> E	83 <sub>H</sub>	P1H Direction Control Register	00 <sub>H</sub>
<b>RP0H</b>	<b>b</b> F108 <sub>H</sub> E	84 <sub>H</sub>	System Startup Configuration Register (read only)	XX <sub>H</sub>
<b>CC16IC</b>	<b>b</b> F160 <sub>H</sub> E	B0 <sub>H</sub>	CAPCOM Register 16 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC17IC</b>	<b>b</b> F162 <sub>H</sub> E	B1 <sub>H</sub>	CAPCOM Register 17 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC18IC</b>	<b>b</b> F164 <sub>H</sub> E	B2 <sub>H</sub>	CAPCOM Register 18 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC19IC</b>	<b>b</b> F166 <sub>H</sub> E	B3 <sub>H</sub>	CAPCOM Register 19 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC20IC</b>	<b>b</b> F168 <sub>H</sub> E	B4 <sub>H</sub>	CAPCOM Register 20 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC21IC</b>	<b>b</b> F16A <sub>H</sub> E	B5 <sub>H</sub>	CAPCOM Register 21 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC22IC</b>	<b>b</b> F16C <sub>H</sub> E	B6 <sub>H</sub>	CAPCOM Register 22 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC23IC</b>	<b>b</b> F16E <sub>H</sub> E	B7 <sub>H</sub>	CAPCOM Register 23 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC24IC</b>	<b>b</b> F170 <sub>H</sub> E	B8 <sub>H</sub>	CAPCOM Register 24 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC25IC</b>	<b>b</b> F172 <sub>H</sub> E	B9 <sub>H</sub>	CAPCOM Register 25 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC26IC</b>	<b>b</b> F174 <sub>H</sub> E	BA <sub>H</sub>	CAPCOM Register 26 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC27IC</b>	<b>b</b> F176 <sub>H</sub> E	BB <sub>H</sub>	CAPCOM Register 27 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC28IC</b>	<b>b</b> F178 <sub>H</sub> E	BC <sub>H</sub>	CAPCOM Register 28 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>T7IC</b>	<b>b</b> F17A <sub>H</sub> E	BD <sub>H</sub>	CAPCOM Timer 7 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>T8IC</b>	<b>b</b> F17C <sub>H</sub> E	BE <sub>H</sub>	CAPCOM Timer 8 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>XP4IC</b>	<b>b</b> F182 <sub>H</sub> E	C1 <sub>H</sub>	ASC1 Transmit Interrupt Control Register	0000 <sub>H</sub>
<b>CC29IC</b>	<b>b</b> F184 <sub>H</sub> E	C2 <sub>H</sub>	CAPCOM Register 29 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>XP0IC</b>	<b>b</b> F186 <sub>H</sub> E	C3 <sub>H</sub>	IIC Data Interrupt Control Register	0000 <sub>H</sub>
<b>XP5IC</b>	<b>b</b> F18A <sub>H</sub> E	C5 <sub>H</sub>	ASC1 Receive Interrupt Control Register	0000 <sub>H</sub>
<b>CC30IC</b>	<b>b</b> F18C <sub>H</sub> E	C6 <sub>H</sub>	CAPCOM Register 30 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>XP1IC</b>	<b>b</b> F18E <sub>H</sub> E	C7 <sub>H</sub>	IIC Protocol Interrupt Control Register	0000 <sub>H</sub>
<b>XP6IC</b>	<b>b</b> F192 <sub>H</sub> E	C9 <sub>H</sub>	ASC1 Error Interrupt Control Register	0000 <sub>H</sub>
<b>CC31IC</b>	<b>b</b> F194 <sub>H</sub> E	CA <sub>H</sub>	CAPCOM Register 31 Interrupt Ctrl. Reg.	0000 <sub>H</sub>

**Table 25-4 C161CS/JC/JI Registers, Ordered by Address (cont'd)**

Name	Physical Address	8-bit Addr.	Description	Reset Value
<b>XP2IC</b>	<b>b</b> F196 <sub>H</sub> <b>E</b>	CB <sub>H</sub>	CAN1 Interrupt Control Register	0000 <sub>H</sub>
<b>XP7IC</b>	<b>b</b> F19A <sub>H</sub> <b>E</b>	CD <sub>H</sub>	CAN2/SDLM Interrupt Control Register	0000 <sub>H</sub>
<b>S0TBIC</b>	<b>b</b> F19C <sub>H</sub> <b>E</b>	CE <sub>H</sub>	Serial Channel 0 Transmit Buffer Interrupt Control Register	0000 <sub>H</sub>
<b>XP3IC</b>	<b>b</b> F19E <sub>H</sub> <b>E</b>	CF <sub>H</sub>	RTC/PLL/OWD Interrupt Control Register	0000 <sub>H</sub>
<b>EXICON</b>	<b>b</b> F1C0 <sub>H</sub> <b>E</b>	E0 <sub>H</sub>	External Interrupt Control Register	0000 <sub>H</sub>
<b>ODP2</b>	<b>b</b> F1C2 <sub>H</sub> <b>E</b>	E1 <sub>H</sub>	Port 2 Open Drain Control Register	0000 <sub>H</sub>
<b>PICON</b>	F1C4 <sub>H</sub> <b>E</b>	E2 <sub>H</sub>	Port Input Threshold Control Register	0000 <sub>H</sub>
<b>ODP3</b>	<b>b</b> F1C6 <sub>H</sub> <b>E</b>	E3 <sub>H</sub>	Port 3 Open Drain Control Register	0000 <sub>H</sub>
<b>ODP4</b>	<b>b</b> F1CA <sub>H</sub> <b>E</b>	E5 <sub>H</sub>	Port 4 Open Drain Control Register	00 <sub>H</sub>
<b>ODP6</b>	<b>b</b> F1CE <sub>H</sub> <b>E</b>	E7 <sub>H</sub>	Port 6 Open Drain Control Register	00 <sub>H</sub>
<b>SYSICON2</b>	<b>b</b> F1D0 <sub>H</sub> <b>E</b>	E8 <sub>H</sub>	CPU System Configuration Register 2	0000 <sub>H</sub>
<b>ODP7</b>	<b>b</b> F1D2 <sub>H</sub> <b>E</b>	E9 <sub>H</sub>	Port 7 Open Drain Control Register	00 <sub>H</sub>
<b>SYSICON3</b>	<b>b</b> F1D4 <sub>H</sub> <b>E</b>	EA <sub>H</sub>	CPU System Configuration Register 3	0000 <sub>H</sub>
<b>EXISEL</b>	<b>b</b> F1DA <sub>H</sub> <b>E</b>	ED <sub>H</sub>	External Interrupt Source Select Register	0000 <sub>H</sub>
<b>SYSICON1</b>	<b>b</b> F1DC <sub>H</sub> <b>E</b>	EE <sub>H</sub>	CPU System Configuration Register 1	0000 <sub>H</sub>
<b>ISNC</b>	<b>b</b> F1DE <sub>H</sub> <b>E</b>	EF <sub>H</sub>	Interrupt Subnode Control Register	0000 <sub>H</sub>
<b>RSTCON</b>	<b>b</b> F1E0 <sub>H</sub> <b>m</b>	–	Reset Control Register	00XX <sub>H</sub>
<b>DPP0</b>	FE00 <sub>H</sub>	00 <sub>H</sub>	CPU Data Page Pointer 0 Register (10 bits)	0000 <sub>H</sub>
<b>DPP1</b>	FE02 <sub>H</sub>	01 <sub>H</sub>	CPU Data Page Pointer 1 Register (10 bits)	0001 <sub>H</sub>
<b>DPP2</b>	FE04 <sub>H</sub>	02 <sub>H</sub>	CPU Data Page Pointer 2 Register (10 bits)	0002 <sub>H</sub>
<b>DPP3</b>	FE06 <sub>H</sub>	03 <sub>H</sub>	CPU Data Page Pointer 3 Register (10 bits)	0003 <sub>H</sub>
<b>CSP</b>	FE08 <sub>H</sub>	04 <sub>H</sub>	CPU Code Segment Pointer Register (8 bits, not directly writeable)	0000 <sub>H</sub>
<b>MDH</b>	FE0C <sub>H</sub>	06 <sub>H</sub>	CPU Multiply Divide Register – High Word	0000 <sub>H</sub>
<b>MDL</b>	FE0E <sub>H</sub>	07 <sub>H</sub>	CPU Multiply Divide Register – Low Word	0000 <sub>H</sub>
<b>CP</b>	FE10 <sub>H</sub>	08 <sub>H</sub>	CPU Context Pointer Register	FC00 <sub>H</sub>
<b>SP</b>	FE12 <sub>H</sub>	09 <sub>H</sub>	CPU System Stack Pointer Register	FC00 <sub>H</sub>
<b>STKOV</b>	FE14 <sub>H</sub>	0A <sub>H</sub>	CPU Stack Overflow Pointer Register	FA00 <sub>H</sub>
<b>STKUN</b>	FE16 <sub>H</sub>	0B <sub>H</sub>	CPU Stack Underflow Pointer Register	FC00 <sub>H</sub>

**Table 25-4 C161CS/JC/JI Registers, Ordered by Address (cont'd)**

<b>Name</b>	<b>Physical Address</b>	<b>8-bit Addr.</b>	<b>Description</b>	<b>Reset Value</b>
<b>ADDRSEL1</b>	FE18 <sub>H</sub>	0C <sub>H</sub>	Address Select Register 1	0000 <sub>H</sub>
<b>ADDRSEL2</b>	FE1A <sub>H</sub>	0D <sub>H</sub>	Address Select Register 2	0000 <sub>H</sub>
<b>ADDRSEL3</b>	FE1C <sub>H</sub>	0E <sub>H</sub>	Address Select Register 3	0000 <sub>H</sub>
<b>ADDRSEL4</b>	FE1E <sub>H</sub>	0F <sub>H</sub>	Address Select Register 4	0000 <sub>H</sub>
<b>T2</b>	FE40 <sub>H</sub>	20 <sub>H</sub>	GPT1 Timer 2 Register	0000 <sub>H</sub>
<b>T3</b>	FE42 <sub>H</sub>	21 <sub>H</sub>	GPT1 Timer 3 Register	0000 <sub>H</sub>
<b>T4</b>	FE44 <sub>H</sub>	22 <sub>H</sub>	GPT1 Timer 4 Register	0000 <sub>H</sub>
<b>T5</b>	FE46 <sub>H</sub>	23 <sub>H</sub>	GPT2 Timer 5 Register	0000 <sub>H</sub>
<b>T6</b>	FE48 <sub>H</sub>	24 <sub>H</sub>	GPT2 Timer 6 Register	0000 <sub>H</sub>
<b>CAPREL</b>	FE4A <sub>H</sub>	25 <sub>H</sub>	GPT2 Capture/Reload Register	0000 <sub>H</sub>
<b>T0</b>	FE50 <sub>H</sub>	28 <sub>H</sub>	CAPCOM Timer 0 Register	0000 <sub>H</sub>
<b>T1</b>	FE52 <sub>H</sub>	29 <sub>H</sub>	CAPCOM Timer 1 Register	0000 <sub>H</sub>
<b>T0REL</b>	FE54 <sub>H</sub>	2A <sub>H</sub>	CAPCOM Timer 0 Reload Register	0000 <sub>H</sub>
<b>T1REL</b>	FE56 <sub>H</sub>	2B <sub>H</sub>	CAPCOM Timer 1 Reload Register	0000 <sub>H</sub>
<b>CC16</b>	FE60 <sub>H</sub>	30 <sub>H</sub>	CAPCOM Register 16	0000 <sub>H</sub>
<b>CC17</b>	FE62 <sub>H</sub>	31 <sub>H</sub>	CAPCOM Register 17	0000 <sub>H</sub>
<b>CC18</b>	FE64 <sub>H</sub>	32 <sub>H</sub>	CAPCOM Register 18	0000 <sub>H</sub>
<b>CC19</b>	FE66 <sub>H</sub>	33 <sub>H</sub>	CAPCOM Register 19	0000 <sub>H</sub>
<b>CC20</b>	FE68 <sub>H</sub>	34 <sub>H</sub>	CAPCOM Register 20	0000 <sub>H</sub>
<b>CC21</b>	FE6A <sub>H</sub>	35 <sub>H</sub>	CAPCOM Register 21	0000 <sub>H</sub>
<b>CC22</b>	FE6C <sub>H</sub>	36 <sub>H</sub>	CAPCOM Register 22	0000 <sub>H</sub>
<b>CC23</b>	FE6E <sub>H</sub>	37 <sub>H</sub>	CAPCOM Register 23	0000 <sub>H</sub>
<b>CC24</b>	FE70 <sub>H</sub>	38 <sub>H</sub>	CAPCOM Register 24	0000 <sub>H</sub>
<b>CC25</b>	FE72 <sub>H</sub>	39 <sub>H</sub>	CAPCOM Register 25	0000 <sub>H</sub>
<b>CC26</b>	FE74 <sub>H</sub>	3A <sub>H</sub>	CAPCOM Register 26	0000 <sub>H</sub>
<b>CC27</b>	FE76 <sub>H</sub>	3B <sub>H</sub>	CAPCOM Register 27	0000 <sub>H</sub>
<b>CC28</b>	FE78 <sub>H</sub>	3C <sub>H</sub>	CAPCOM Register 28	0000 <sub>H</sub>
<b>CC29</b>	FE7A <sub>H</sub>	3D <sub>H</sub>	CAPCOM Register 29	0000 <sub>H</sub>
<b>CC30</b>	FE7C <sub>H</sub>	3E <sub>H</sub>	CAPCOM Register 30	0000 <sub>H</sub>
<b>CC31</b>	FE7E <sub>H</sub>	3F <sub>H</sub>	CAPCOM Register 31	0000 <sub>H</sub>



**Table 25-4 C161CS/JC/JI Registers, Ordered by Address (cont'd)**

<b>Name</b>	<b>Physical Address</b>	<b>8-bit Addr.</b>	<b>Description</b>	<b>Reset Value</b>
<b>CC0</b>	FE80 <sub>H</sub>	40 <sub>H</sub>	CAPCOM Register 0	0000 <sub>H</sub>
<b>CC1</b>	FE82 <sub>H</sub>	41 <sub>H</sub>	CAPCOM Register 1	0000 <sub>H</sub>
<b>CC2</b>	FE84 <sub>H</sub>	42 <sub>H</sub>	CAPCOM Register 2	0000 <sub>H</sub>
<b>CC3</b>	FE86 <sub>H</sub>	43 <sub>H</sub>	CAPCOM Register 3	0000 <sub>H</sub>
<b>CC4</b>	FE88 <sub>H</sub>	44 <sub>H</sub>	CAPCOM Register 4	0000 <sub>H</sub>
<b>CC5</b>	FE8A <sub>H</sub>	45 <sub>H</sub>	CAPCOM Register 5	0000 <sub>H</sub>
<b>CC6</b>	FE8C <sub>H</sub>	46 <sub>H</sub>	CAPCOM Register 6	0000 <sub>H</sub>
<b>CC7</b>	FE8E <sub>H</sub>	47 <sub>H</sub>	CAPCOM Register 7	0000 <sub>H</sub>
<b>CC8</b>	FE90 <sub>H</sub>	48 <sub>H</sub>	CAPCOM Register 8	0000 <sub>H</sub>
<b>CC9</b>	FE92 <sub>H</sub>	49 <sub>H</sub>	CAPCOM Register 9	0000 <sub>H</sub>
<b>CC10</b>	FE94 <sub>H</sub>	4A <sub>H</sub>	CAPCOM Register 10	0000 <sub>H</sub>
<b>CC11</b>	FE96 <sub>H</sub>	4B <sub>H</sub>	CAPCOM Register 11	0000 <sub>H</sub>
<b>CC12</b>	FE98 <sub>H</sub>	4C <sub>H</sub>	CAPCOM Register 12	0000 <sub>H</sub>
<b>CC13</b>	FE9A <sub>H</sub>	4D <sub>H</sub>	CAPCOM Register 13	0000 <sub>H</sub>
<b>CC14</b>	FE9C <sub>H</sub>	4E <sub>H</sub>	CAPCOM Register 14	0000 <sub>H</sub>
<b>CC15</b>	FE9E <sub>H</sub>	4F <sub>H</sub>	CAPCOM Register 15	0000 <sub>H</sub>
<b>ADDAT</b>	FEA0 <sub>H</sub>	50 <sub>H</sub>	A/D Converter Result Register	0000 <sub>H</sub>
<b>P1DIDIS</b>	FEA4 <sub>H</sub>	52 <sub>H</sub>	PORT1 Digital Input Disable Register	0000 <sub>H</sub>
<b>WDT</b>	FEAE <sub>H</sub>	57 <sub>H</sub>	Watchdog Timer Register (read only)	0000 <sub>H</sub>
<b>S0TBUF</b>	FEB0 <sub>H</sub>	58 <sub>H</sub>	Serial Channel 0 Transmit Buffer Register	0000 <sub>H</sub>
<b>S0RBUF</b>	FEB2 <sub>H</sub>	59 <sub>H</sub>	Serial Channel 0 Receive Buffer Register (read only)	XXXX <sub>H</sub>
<b>S0BG</b>	FEB4 <sub>H</sub>	5A <sub>H</sub>	Serial Channel 0 Baud Rate Generator Reload Register	0000 <sub>H</sub>
<b>PECC0</b>	FEC0 <sub>H</sub>	60 <sub>H</sub>	PEC Channel 0 Control Register	0000 <sub>H</sub>
<b>PECC1</b>	FEC2 <sub>H</sub>	61 <sub>H</sub>	PEC Channel 1 Control Register	0000 <sub>H</sub>
<b>PECC2</b>	FEC4 <sub>H</sub>	62 <sub>H</sub>	PEC Channel 2 Control Register	0000 <sub>H</sub>
<b>PECC3</b>	FEC6 <sub>H</sub>	63 <sub>H</sub>	PEC Channel 3 Control Register	0000 <sub>H</sub>
<b>PECC4</b>	FEC8 <sub>H</sub>	64 <sub>H</sub>	PEC Channel 4 Control Register	0000 <sub>H</sub>
<b>PECC5</b>	FECA <sub>H</sub>	65 <sub>H</sub>	PEC Channel 5 Control Register	0000 <sub>H</sub>

**Table 25-4 C161CS/JC/JI Registers, Ordered by Address (cont'd)**

Name	Physical Address	8-bit Addr.	Description	Reset Value
<b>PECC6</b>	FECC <sub>H</sub>	66 <sub>H</sub>	PEC Channel 6 Control Register	0000 <sub>H</sub>
<b>PECC7</b>	FECE <sub>H</sub>	67 <sub>H</sub>	PEC Channel 7 Control Register	0000 <sub>H</sub>
<b>P0L</b> <b>b</b>	FF00 <sub>H</sub>	80 <sub>H</sub>	Port 0 Low Register (Lower half of PORT0)	00 <sub>H</sub>
<b>P0H</b> <b>b</b>	FF02 <sub>H</sub>	81 <sub>H</sub>	Port 0 High Register (Upper half of PORT0)	00 <sub>H</sub>
<b>P1L</b> <b>b</b>	FF04 <sub>H</sub>	82 <sub>H</sub>	Port 1 Low Register (Lower half of PORT1)	00 <sub>H</sub>
<b>P1H</b> <b>b</b>	FF06 <sub>H</sub>	83 <sub>H</sub>	Port 1 High Register (Upper half of PORT1)	00 <sub>H</sub>
<b>BUSCON0</b> <b>b</b>	FF0C <sub>H</sub>	86 <sub>H</sub>	Bus Configuration Register 0	0000 <sub>H</sub>
<b>MDC</b> <b>b</b>	FF0E <sub>H</sub>	87 <sub>H</sub>	CPU Multiply Divide Control Register	0000 <sub>H</sub>
<b>PSW</b> <b>b</b>	FF10 <sub>H</sub>	88 <sub>H</sub>	CPU Program Status Word	0000 <sub>H</sub>
<b>SYSCON</b> <b>b</b>	FF12 <sub>H</sub>	89 <sub>H</sub>	CPU System Configuration Register	<sup>1)</sup> 0xx0 <sub>H</sub>
<b>BUSCON1</b> <b>b</b>	FF14 <sub>H</sub>	8A <sub>H</sub>	Bus Configuration Register 1	0000 <sub>H</sub>
<b>BUSCON2</b> <b>b</b>	FF16 <sub>H</sub>	8B <sub>H</sub>	Bus Configuration Register 2	0000 <sub>H</sub>
<b>BUSCON3</b> <b>b</b>	FF18 <sub>H</sub>	8C <sub>H</sub>	Bus Configuration Register 3	0000 <sub>H</sub>
<b>BUSCON4</b> <b>b</b>	FF1A <sub>H</sub>	8D <sub>H</sub>	Bus Configuration Register 4	0000 <sub>H</sub>
<b>ZEROS</b> <b>b</b>	FF1C <sub>H</sub>	8E <sub>H</sub>	Constant Value 0's Register (read only)	0000 <sub>H</sub>
<b>ONES</b> <b>b</b>	FF1E <sub>H</sub>	8F <sub>H</sub>	Constant Value 1's Register (read only)	FFFF <sub>H</sub>
<b>T78CON</b> <b>b</b>	FF20 <sub>H</sub>	90 <sub>H</sub>	CAPCOM Timer 7 and 8 Control Register	0000 <sub>H</sub>
<b>CCM4</b> <b>b</b>	FF22 <sub>H</sub>	91 <sub>H</sub>	CAPCOM Mode Control Register 4	0000 <sub>H</sub>
<b>CCM5</b> <b>b</b>	FF24 <sub>H</sub>	92 <sub>H</sub>	CAPCOM Mode Control Register 5	0000 <sub>H</sub>
<b>CCM6</b> <b>b</b>	FF26 <sub>H</sub>	93 <sub>H</sub>	CAPCOM Mode Control Register 6	0000 <sub>H</sub>
<b>CCM7</b> <b>b</b>	FF28 <sub>H</sub>	94 <sub>H</sub>	CAPCOM Mode Control Register 7	0000 <sub>H</sub>
<b>T2CON</b> <b>b</b>	FF40 <sub>H</sub>	A0 <sub>H</sub>	GPT1 Timer 2 Control Register	0000 <sub>H</sub>
<b>T3CON</b> <b>b</b>	FF42 <sub>H</sub>	A1 <sub>H</sub>	GPT1 Timer 3 Control Register	0000 <sub>H</sub>
<b>T4CON</b> <b>b</b>	FF44 <sub>H</sub>	A2 <sub>H</sub>	GPT1 Timer 4 Control Register	0000 <sub>H</sub>
<b>T5CON</b> <b>b</b>	FF46 <sub>H</sub>	A3 <sub>H</sub>	GPT2 Timer 5 Control Register	0000 <sub>H</sub>
<b>T6CON</b> <b>b</b>	FF48 <sub>H</sub>	A4 <sub>H</sub>	GPT2 Timer 6 Control Register	0000 <sub>H</sub>
<b>T01CON</b> <b>b</b>	FF50 <sub>H</sub>	A8 <sub>H</sub>	CAPCOM Timer 0 and Timer 1 Ctrl. Reg.	0000 <sub>H</sub>
<b>CCM0</b> <b>b</b>	FF52 <sub>H</sub>	A9 <sub>H</sub>	CAPCOM Mode Control Register 0	0000 <sub>H</sub>
<b>CCM1</b> <b>b</b>	FF54 <sub>H</sub>	AA <sub>H</sub>	CAPCOM Mode Control Register 1	0000 <sub>H</sub>
<b>CCM2</b> <b>b</b>	FF56 <sub>H</sub>	AB <sub>H</sub>	CAPCOM Mode Control Register 2	0000 <sub>H</sub>



**Table 25-4 C161CS/JC/JI Registers, Ordered by Address (cont'd)**

Name	Physical Address	8-bit Addr.	Description	Reset Value
<b>CCM3</b>	<b>b</b> FF58 <sub>H</sub>	AC <sub>H</sub>	CAPCOM Mode Control Register 3	0000 <sub>H</sub>
<b>T2IC</b>	<b>b</b> FF60 <sub>H</sub>	B0 <sub>H</sub>	GPT1 Timer 2 Interrupt Control Register	0000 <sub>H</sub>
<b>T3IC</b>	<b>b</b> FF62 <sub>H</sub>	B1 <sub>H</sub>	GPT1 Timer 3 Interrupt Control Register	0000 <sub>H</sub>
<b>T4IC</b>	<b>b</b> FF64 <sub>H</sub>	B2 <sub>H</sub>	GPT1 Timer 4 Interrupt Control Register	0000 <sub>H</sub>
<b>T5IC</b>	<b>b</b> FF66 <sub>H</sub>	B3 <sub>H</sub>	GPT2 Timer 5 Interrupt Control Register	0000 <sub>H</sub>
<b>T6IC</b>	<b>b</b> FF68 <sub>H</sub>	B4 <sub>H</sub>	GPT2 Timer 6 Interrupt Control Register	0000 <sub>H</sub>
<b>CRIC</b>	<b>b</b> FF6A <sub>H</sub>	B5 <sub>H</sub>	GPT2 CAPREL Interrupt Control Register	0000 <sub>H</sub>
<b>S0TIC</b>	<b>b</b> FF6C <sub>H</sub>	B6 <sub>H</sub>	Serial Channel 0 Transmit Interrupt Control Register	0000 <sub>H</sub>
<b>S0RIC</b>	<b>b</b> FF6E <sub>H</sub>	B7 <sub>H</sub>	Serial Channel 0 Receive Interrupt Control Register	0000 <sub>H</sub>
<b>S0EIC</b>	<b>b</b> FF70 <sub>H</sub>	B8 <sub>H</sub>	Serial Channel 0 Error Interrupt Control Register	0000 <sub>H</sub>
<b>SSCTIC</b>	<b>b</b> FF72 <sub>H</sub>	B9 <sub>H</sub>	SSC Transmit Interrupt Control Register	0000 <sub>H</sub>
<b>SSCRIC</b>	<b>b</b> FF74 <sub>H</sub>	BA <sub>H</sub>	SSC Receive Interrupt Control Register	0000 <sub>H</sub>
<b>SSCEIC</b>	<b>b</b> FF76 <sub>H</sub>	BB <sub>H</sub>	SSC Error Interrupt Control Register	0000 <sub>H</sub>
<b>CC0IC</b>	<b>b</b> FF78 <sub>H</sub>	BC <sub>H</sub>	CAPCOM Register 0 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC1IC</b>	<b>b</b> FF7A <sub>H</sub>	BD <sub>H</sub>	CAPCOM Register 1 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC2IC</b>	<b>b</b> FF7C <sub>H</sub>	BE <sub>H</sub>	CAPCOM Register 2 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC3IC</b>	<b>b</b> FF7E <sub>H</sub>	BF <sub>H</sub>	CAPCOM Register 3 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC4IC</b>	<b>b</b> FF80 <sub>H</sub>	C0 <sub>H</sub>	CAPCOM Register 4 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC5IC</b>	<b>b</b> FF82 <sub>H</sub>	C1 <sub>H</sub>	CAPCOM Register 5 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC6IC</b>	<b>b</b> FF84 <sub>H</sub>	C2 <sub>H</sub>	CAPCOM Register 6 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC7IC</b>	<b>b</b> FF86 <sub>H</sub>	C3 <sub>H</sub>	CAPCOM Register 7 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC8IC</b>	<b>b</b> FF88 <sub>H</sub>	C4 <sub>H</sub>	CAPCOM Register 8 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC9IC</b>	<b>b</b> FF8A <sub>H</sub>	C5 <sub>H</sub>	CAPCOM Register 9 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC10IC</b>	<b>b</b> FF8C <sub>H</sub>	C6 <sub>H</sub>	CAPCOM Register 10 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC11IC</b>	<b>b</b> FF8E <sub>H</sub>	C7 <sub>H</sub>	CAPCOM Register 11 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC12IC</b>	<b>b</b> FF90 <sub>H</sub>	C8 <sub>H</sub>	CAPCOM Register 12 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC13IC</b>	<b>b</b> FF92 <sub>H</sub>	C9 <sub>H</sub>	CAPCOM Register 13 Interrupt Ctrl. Reg.	0000 <sub>H</sub>

**Table 25-4 C161CS/JC/JI Registers, Ordered by Address (cont'd)**

Name	Physical Address	8-bit Addr.	Description	Reset Value
<b>CC14IC</b>	<b>b</b> FF94 <sub>H</sub>	CA <sub>H</sub>	CAPCOM Register 14 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>CC15IC</b>	<b>b</b> FF96 <sub>H</sub>	CB <sub>H</sub>	CAPCOM Register 15 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>ADCIC</b>	<b>b</b> FF98 <sub>H</sub>	CC <sub>H</sub>	A/D Converter End of Conversion Interrupt Control Register	0000 <sub>H</sub>
<b>ADEIC</b>	<b>b</b> FF9A <sub>H</sub>	CD <sub>H</sub>	A/D Converter Overrun Error Interrupt Control Register	0000 <sub>H</sub>
<b>T0IC</b>	<b>b</b> FF9C <sub>H</sub>	CE <sub>H</sub>	CAPCOM Timer 0 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>T1IC</b>	<b>b</b> FF9E <sub>H</sub>	CF <sub>H</sub>	CAPCOM Timer 1 Interrupt Ctrl. Reg.	0000 <sub>H</sub>
<b>ADCON</b>	<b>b</b> FFA0 <sub>H</sub>	D0 <sub>H</sub>	A/D Converter Control Register	0000 <sub>H</sub>
<b>P5</b>	<b>b</b> FFA2 <sub>H</sub>	D1 <sub>H</sub>	Port 5 Register (read only)	XXXX <sub>H</sub>
<b>P5DIDIS</b>	<b>b</b> FFA4 <sub>H</sub>	D2 <sub>H</sub>	Port 5 Digital Input Disable Register	0000 <sub>H</sub>
<b>FOCON</b>	<b>b</b> FFAA <sub>H</sub>	D5 <sub>H</sub>	Frequency Output Control Register	0000 <sub>H</sub>
<b>TFR</b>	<b>b</b> FFAC <sub>H</sub>	D6 <sub>H</sub>	Trap Flag Register	0000 <sub>H</sub>
<b>WDTCON</b>	<b>b</b> FFAE <sub>H</sub>	D7 <sub>H</sub>	Watchdog Timer Control Register	<sup>2)</sup> 00xx <sub>H</sub>
<b>S0CON</b>	<b>b</b> FFB0 <sub>H</sub>	D8 <sub>H</sub>	Serial Channel 0 Control Register	0000 <sub>H</sub>
<b>SSCCON</b>	<b>b</b> FFB2 <sub>H</sub>	D9 <sub>H</sub>	SSC Control Register	0000 <sub>H</sub>
<b>P2</b>	<b>b</b> FFC0 <sub>H</sub>	E0 <sub>H</sub>	Port 2 Register	0000 <sub>H</sub>
<b>DP2</b>	<b>b</b> FFC2 <sub>H</sub>	E1 <sub>H</sub>	Port 2 Direction Control Register	0000 <sub>H</sub>
<b>P3</b>	<b>b</b> FFC4 <sub>H</sub>	E2 <sub>H</sub>	Port 3 Register	0000 <sub>H</sub>
<b>DP3</b>	<b>b</b> FFC6 <sub>H</sub>	E3 <sub>H</sub>	Port 3 Direction Control Register	0000 <sub>H</sub>
<b>P4</b>	<b>b</b> FFC8 <sub>H</sub>	E4 <sub>H</sub>	Port 4 Register (7 bits)	00 <sub>H</sub>
<b>DP4</b>	<b>b</b> FFCA <sub>H</sub>	E5 <sub>H</sub>	Port 4 Direction Control Register	00 <sub>H</sub>
<b>P6</b>	<b>b</b> FFCC <sub>H</sub>	E6 <sub>H</sub>	Port 6 Register (8 bits)	00 <sub>H</sub>
<b>DP6</b>	<b>b</b> FFCE <sub>H</sub>	E7 <sub>H</sub>	Port 6 Direction Control Register	00 <sub>H</sub>
<b>P7</b>	<b>b</b> FFD0 <sub>H</sub>	E8 <sub>H</sub>	Port 7 Register (8 bits)	00 <sub>H</sub>
<b>DP7</b>	<b>b</b> FFD2 <sub>H</sub>	E9 <sub>H</sub>	Port 7 Direction Control Register	00 <sub>H</sub>
<b>P9</b>	<b>b</b> FFD8 <sub>H</sub>	EC <sub>H</sub>	Port 9 Register (8 bits)	00 <sub>H</sub>
<b>DP9</b>	<b>b</b> FFDA <sub>H</sub>	ED <sub>H</sub>	Port 9 Direction Control Register	00 <sub>H</sub>

<sup>1)</sup> The system configuration is selected during reset.

<sup>2)</sup> The reset value depends on the indicated reset source.

## 25.5 Special Notes

### PEC Pointer Registers

The source and destination pointers for the peripheral event controller are mapped to a special area within the internal RAM. Pointers that are not occupied by the PEC may therefore be used like normal RAM. During Power Down mode or any warm reset the PEC pointers are preserved.

The PEC and its registers are described in [Chapter 5](#).

### GPR Access in the ESFR Area

The locations 00'F000<sub>H</sub> ... 00'F01E<sub>H</sub> within the ESFR area are reserved and allow to access the current register bank via short register addressing modes. The GPRs are mirrored to the ESFR area which allows access to the current register bank even after switching register spaces (see example below).

```
MOV    R5, DP3           ;GPR access via SFR area
EXTR   #1
MOV    R5, ODP3          ;GPR access via ESFR area
```

### Writing Bytes to SFRs

All special function registers may be accessed wordwise or byte-wise (some of them even bitwise). Reading bytes from word SFRs is a non-critical operation. However, when writing bytes to word SFRs the complementary byte of the respective SFR is cleared with the write operation.

## 26 Instruction Set Summary

This chapter briefly summarizes the C161CS/JC/JI's instructions ordered by instruction classes. This provides a basic understanding of the C161CS/JC/JI's instruction set, the power and versatility of the instructions and their general usage.

**A detailed description** of each single instruction, including its operand data type, condition flag settings, addressing modes, length (number of bytes) and object code format is provided in the “**Instruction Set Manual**” for the C166 Family. This manual also provides tables ordering the instructions according to various criteria, to allow quick references.

### Summary of Instruction Classes

Grouping the various instruction into classes aids in identifying similar instructions (e.g. SHR, ROR) and variations of certain instructions (e.g. ADD, ADDB). This provides an easy access to the possibilities and the power of the instructions of the C161CS/JC/JI.

*Note: The used mnemonics refer to the detailed description.*

### Arithmetic Instructions

• Addition of two words or bytes:	ADD	ADDB
• Addition with Carry of two words or bytes:	ADDC	ADDCB
• Subtraction of two words or bytes:	SUB	SUBB
• Subtraction with Carry of two words or bytes:	SUBC	SUBCB
• 16 × 16 bit signed or unsigned multiplication:	MUL	MULU
• 16 / 16 bit signed or unsigned division:	DIV	DIVU
• 32 / 16 bit signed or unsigned division:	DIVL	DIVLU
• 1's complement of a word or byte:	CPL	CPLB
• 2's complement (negation) of a word or byte:	NEG	NEGB

### Logical Instructions

• Bitwise ANDing of two words or bytes:	AND	ANDB
• Bitwise ORing of two words or bytes:	OR	ORB
• Bitwise XORing of two words or bytes:	XOR	XORB

### Compare and Loop Control Instructions

• Comparison of two words or bytes:	CMP	CMPB
• Comparison of two words with post-increment by either 1 or 2:	CMPI1	CMPI2
• Comparison of two words with post-decrement by either 1 or 2:	CMPD1	CMPD2

### Boolean Bit Manipulation Instructions

- Manipulation of a maskable bit field in either the high or the low byte of a word: BFLDH BFLDL
- Setting a single bit (to '1'): BSET
- Clearing a single bit (to '0'): BCLR
- Movement of a single bit: BMOV
- Movement of a negated bit: BMOVN
- ANDing of two bits: BAND
- ORing of two bits: BOR
- XORing of two bits: BXOR
- Comparison of two bits: BCMP

### Shift and Rotate Instructions

- Shifting right of a word: SHR
- Shifting left of a word: SHL
- Rotating right of a word: ROR
- Rotating left of a word: ROL
- Arithmetic shifting right of a word (sign bit shifting): ASHR

### Prioritize Instruction

- Determination of the number of shift cycles required to normalize a word operand (floating point support): PRIOR

### Data Movement Instructions

- Standard data movement of a word or byte: MOV MOVB
- Data movement of a byte to a word location with either sign or zero byte extension: MOVBS MOVBZ

*Note: The data movement instructions can be used with a big number of different addressing modes including indirect addressing and automatic pointer in-/decrementing.*

### System Stack Instructions

- Pushing of a word onto the system stack: PUSH
- Popping of a word from the system stack: POP
- Saving of a word on the system stack, and then updating the old word with a new value (provided for register bank switching): SCXT

Instruction Set Summary

**Jump Instructions**

- Conditional jumping to an either absolutely, indirectly, or relatively addressed target instruction within the current code segment: JMPA JMPI JMPR
- Unconditional jumping to an absolutely addressed target instruction within any code segment: JMPS
- Conditional jumping to a relatively addressed target instruction within the current code segment depending on the state of a selectable bit: JB JNB
- Conditional jumping to a relatively addressed target instruction within the current code segment depending on the state of a selectable bit with a post-inversion of the tested bit in case of jump taken (semaphore support): JBC JNBS

**Call Instructions**

- Conditional calling of an either absolutely or indirectly addressed subroutine within the current code segment: CALLA CALLI
- Unconditional calling of a relatively addressed subroutine within the current code segment: CALLR
- Unconditional calling of an absolutely addressed subroutine within any code segment: CALLS
- Unconditional calling of an absolutely addressed subroutine within the current code segment plus an additional pushing of a selectable register onto the system stack: PCALL
- Unconditional branching to the interrupt or trap vector jump table in code segment 0: TRAP

**Return Instructions**

- Returning from a subroutine within the current code segment: RET
- Returning from a subroutine within any code segment: RETS
- Returning from a subroutine within the current code segment plus an additional popping of a selectable register from the system stack: RETP
- Returning from an interrupt service routine: RETI

### System Control Instructions

- Resetting the C161CS/JC/JI via software: SRST
- Entering the Idle mode: IDLE
- Entering the Power Down mode: PWRDN
- Servicing the Watchdog Timer: SRVWDT
- Disabling the Watchdog Timer: DISWDT
- Signifying the end of the initialization routine  
(pulls pin RSTOUT high, and disables the effect of  
any later execution of a DISWDT instruction): EINIT

### Miscellaneous

- Null operation which requires 2 Bytes of  
storage and the minimum time for execution: NOP
- Definition of an unseparable instruction sequence: ATOMIC
- Switch 'reg', 'bitoff' and 'bitaddr' addressing modes  
to the Extended SFR space: EXTR
- Override the DPP addressing scheme  
using a specific data page instead of the DPPs,  
and optionally switch to ESFR space: EXTP          EXTTPR
- Override the DPP addressing scheme  
using a specific segment instead of the DPPs,  
and optionally switch to ESFR space: EXTS          EXTSTR

*Note: The ATOMIC and EXT\* instructions provide support for uninterruptable code sequences e.g. for semaphore operations. They also support data addressing beyond the limits of the current DPPs (except ATOMIC), which is advantageous for bigger memory models in high level languages. Refer to [Chapter 24](#) for examples.*

### Protected Instructions

Some instructions of the C161CS/JC/JI which are critical for the functionality of the controller are implemented as so-called Protected Instructions. These protected instructions use the maximum instruction format of 32 bits for decoding, while the regular instructions only use a part of it (e.g. the lower 8 bits) with the other bits providing additional information like involved registers. Decoding all 32 bits of a protected doubleword instruction increases the security in cases of data distortion during instruction fetching. Critical operations like a software reset are therefore only executed if the complete instruction is decoded without an error. This enhances the safety and reliability of a microcontroller system.

## 27 Device Specification

The device specification describes the electrical parameters of the device. It lists DC characteristics like input, output or supply voltages or currents, and AC characteristics like timing characteristics and requirements.

Other than the architecture, the instruction set or the basic functions of the C161CS/JC/JI core and its peripherals, these DC and AC characteristics are subject to changes due to device improvements or specific derivatives of the standard device.

Therefore these characteristics are not contained in this manual, but rather provided in a separate Data Sheet, which can be updated more frequently.

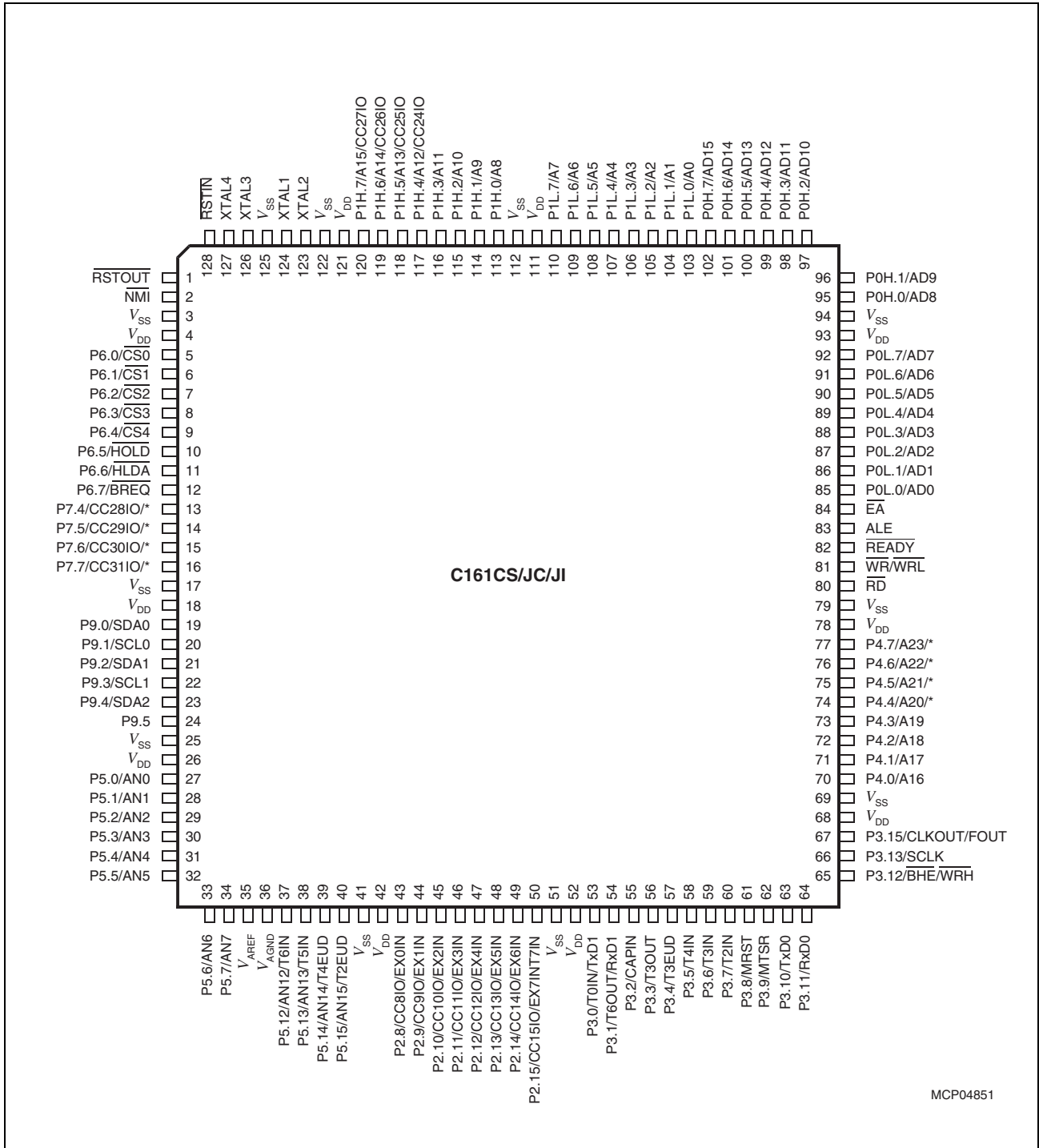
Please refer to the current version of the Data Sheet of the respective device for all electrical parameters.

*Note: In any case the specific characteristics of a device should be verified, before a new design is started. This ensures that the used information is up to date.*

**Figure 27-1** shows the pin diagram of the C161CS/JC/JI. It shows the location of the different supply and IO pins. A detailed description of all the pins is also found in the Data Sheet.

*Note: Not all alternate functions shown in **Figure 27-1** are supported by all derivatives. Please refer to the corresponding descriptions in the data sheets.*





MCP04851

**Figure 27-1 Pin Configuration P-TQFP-128 Package (top view)**

*Note: The CAN and/or SDLM interface lines can be assigned to the indicated pins of Port 4 or Port 7.*

## 28 Keyword Index

This section lists a number of keywords which refer to specific details of the C161CS/JC/JI in terms of its architecture, its functional units or functions. This helps to quickly find the answer to specific questions about the C161CS/JC/JI.

### A

- Access to X-Peripherals 9-39
- Acronyms 1-8
- Adapt Mode 22-15
- ADC 2-17, 18-1
- ADCIC, ADEIC 18-15
- ADCON 18-3
- ADDAT, ADDAT2 18-6
- Address
  - Arbitration 9-27
  - Area Definition 9-26
  - Boundaries 3-12
  - Segment 9-9, 22-18
- ADDRSELx 9-25, 9-27
- ALE length 9-13
- Alternate signals 7-9
- ALU 4-16
- Analog/Digital Converter 2-17, 18-1
- Arbitration
  - Address 9-27
  - External Bus 9-31
  - Master/Slave mode 9-35
  - Signals 9-32
- ASC0 11-1
  - Asynchronous mode 11-5
  - Baudrate 11-11
  - Error Detection 11-10
  - Interrupts 11-15
  - Synchronous mode 11-8
- ASC1 12-1
  - Asynchronous mode 12-5
  - Baudrate 12-6
  - Error Detection 12-6
  - Interrupts 12-7
  - Port control 12-6

- Synchronous mode 12-5
- Asynchronous Serial Interface
  - (->ASC0) 11-1
- Asynchronous Serial Interface
  - (->ASC1) 12-1
- Auto Scan conversion 18-7

### B

- Baudrate
  - ASC0 11-11
  - ASC1 12-6
  - Bootstrap Loader 16-6
  - IIC Bus 21-8
  - SSC 13-13
- BHE 7-30, 9-9
- Bidirectional reset 22-4
- Bit
  - addressable memory 3-4
  - Handling 4-10
  - Manipulation Instructions 26-2
  - protected 2-21, 4-10
  - reserved 2-12
- Bootstrap Loader 16-1, 22-16
- Boundaries 3-12
- BTR 19-12
- BUFFCON 20-36
- BUFFSTAT 20-30
- Bus
  - Arbitration 9-31
  - CAN 2-14, 19-1, 19-37
  - Demultiplexed 9-5
  - Idle State 9-30
  - IIC 2-14, 2-15
  - Mode Configuration 9-3, 22-17
  - Multiplexed 9-4
  - Physical IIC 21-4

BUSCONx 9-22, 9-27

BUSSTAT 20-34

## C

CAN Interface 2-14, 19-1

activation 19-31

port control 19-39

second 19-36

CAPCOM 2-19

interrupt 17-19

timer 17-4

unit 17-1

Capture Mode

CAPCOM 17-13

GPT1 10-20

GPT2 (CAPREL) 10-34

Capture/Compare unit 17-1

CCM0, CCM1, CCM2, CCM3 17-10

CCM4, CCM5, CCM6, CCM7 17-11

CCxIC 17-19

Chip Select

Configuration 9-10, 22-18

Latched/Early 9-11

CLKDIV 20-28

Clock

distribution 6-2, 23-14

generator modes 6-10, 22-19

output signal 23-17

Code memory handling 24-16

Compare modes 17-14

Concatenation of Timers 10-16, 10-33

Configuration

Address 9-9, 22-18

Bus Mode 9-3, 22-17

Chip Select 9-10, 22-18

default 22-20

PLL 6-10, 22-19

Reset 22-7, 22-12

special modes 22-16

Write Control 22-17

Context

Pointer 4-25

Context Switching 5-19

Conversion

analog/digital 18-1

Auto Scan 18-7

timing control 18-13

Count direction 10-4

Counter 10-8, 10-14, 10-29, 10-31

CP 4-25

CPU 2-2, 4-1

CRIC 10-38

CSP 4-20

CSR 19-7

## D

Data Page 4-22, 24-14

boundaries 3-12

Default startup configuration 22-20

Delay

Read/Write 9-16

Demultiplexed Bus 9-5

Development Support 1-7

Direct Drive 6-9

Direction

count 10-4

Disable

Interrupt 5-16

Peripheral 23-14

Segmentation 4-15

Division 4-30, 24-1

DP0L, DP0H 7-13

DP1L, DP1H 7-17

DP2 7-21

DP3 7-26

DP4 7-32, 7-40, 7-46

DP9 7-50

DPP 4-22, 24-14

Driver characteristic (ports) 7-5

## E

Early chip select 9-11

Early WR control 9-16

Edge characteristic (ports) 7-6

Emulation Mode 22-14

- Enable
  - Interrupt 5-16
  - Peripheral 23-14
  - Segmentation 4-15
  - XBUS peripherals 9-38
- Error Detection
  - ASC0 11-10
  - ASC1 12-6
  - CAN 19-4
  - SSC 13-15
- ERRSTAT 20-35
- EXICON 5-28
- EXISEL 5-29
- External
  - Bus 2-10
  - Bus Characteristics 9-12–9-19
  - Bus Idle State 9-30
  - Bus Modes 9-3–9-8
  - Fast interrupts 5-28
  - Host Mode (->EHM) 22-16
  - Interrupt source control 5-29
  - Interrupts 5-26
  - Interrupts during sleep mode 5-30
  - startup configuration 22-13
- F**
- Fast external interrupts 5-28
- FLAGRST 20-38
- Flags 4-16–4-19
- FOCON 23-18
- Frequency output signal 23-17
- Full Duplex 13-7
- G**
- GLOBCON 20-26
- GMS 19-15
- GPR 3-6, 4-25, 25-2
- GPT 2-18
- GPT1 10-1
- GPT2 10-22
- H**
- Half Duplex 13-10
- Hardware
  - Reset 22-2
  - Traps 5-31
- Hold State
  - Entry 9-33
  - Exit 9-34
- I**
- ICADR 21-10
- ICCFG 21-8
- ICCON 21-9
- ICRTB 21-10
- ICST 21-11
- Idle
  - Mode 23-3
  - State (Bus) 9-30
- IFR 20-29
- IIC Bus Module 21-1
- IIC Interface 2-14
- Incremental Interface 10-9
- Indication of reset source 14-6
- Input threshold 7-2
- Instruction 24-1, 26-1
  - Bit Manipulation 26-2
  - Branch 4-4
  - Pipeline 4-3
  - protected 26-4
  - Timing 4-11
  - unseparable 24-14
- INTCON 20-39
- Interface
  - CAN 2-14, 19-1
  - External Bus 9-1
  - IIC 2-14
  - IIC Bus 21-1
  - J1850 2-15, 20-1
  - serial async. (->ASC0) 11-1
  - serial async. (->ASC1) 12-1
  - serial sync. (->SSC) 13-1
- Internal RAM 3-4
- Interrupt
  - CAPCOM 17-19
  - during sleep mode 5-30

- Enable/Disable 5-16
- External 5-26
- Fast external 5-28
- Handling CAN 19-9
- Node Sharing 5-25
- Priority 5-8
- Processing 5-1, 5-6
- Response Times 5-20
- RTC 15-3
- source control 5-29
- Sources 5-2
- System 2-7, 5-2
- Vectors 5-2
- IP 4-19
- IPCR 20-25
- IRAM 3-4
  - status after reset 22-8
- ISNC 5-25
- J**
- J1850 Interface (->SDLM) 2-15, 20-1
- L**
- LARn 19-21
- Latched chip select 9-11
- LGML 19-16
- LMLM 19-17
- M**
- Management
  - Peripheral 23-14
  - Power 23-1
- Master mode
  - External bus 9-35
  - IIC Bus 21-6
- MCFGn 19-22
- MCRn 19-19
- MDC 4-32
- MDH 4-30
- MDL 4-31
- Memory 2-8
  - bit-addressable 3-4
  - Code memory handling 24-16
  - External 3-11
  - RAM/SFR 3-4
  - ROM area 3-3
  - Tri-state time 9-15
  - XRAM 3-9
- Memory Cycle Time 9-14
- Multimaster mode
  - IIC Bus 21-7
- Multiplexed Bus 9-4
- Multiplication 4-30, 24-1
- N**
- NMI 5-1, 5-34
- Noise filter (Ext. Interrupts) 5-30
- O**
- ODP2 7-22, 7-41, 7-47
- ODP3 7-27
- ODP4 7-33
- ONES 4-33
- Open Drain Mode 7-4
- Oscillator
  - circuitry 6-3, 6-5
  - measurement 6-3
  - selection 6-6
  - Watchdog 6-11, 22-19
- P**
- P0L, P0H 7-12
- P1L, P1H 7-16
- P2 7-21
- P3 7-26
- P4 7-32, 7-40, 7-46
- P5 7-37
- P5DIDIS 7-39
- P9 7-50
- PCIR 19-10
- PEC 2-8, 3-7, 5-12
  - Response Times 5-23
- PECCx 5-12
- Peripheral
  - enable on XBUS 9-38
  - Enable/Disable 23-14

Management 23-14  
 Summary 2-11  
 Phase Locked Loop (->PLL) 6-1  
 PICON 7-2  
 Pins 8-1  
   in Idle and Power Down mode 23-9  
 Pipeline 4-3  
   Effects 4-6  
 PLL 6-1, 22-19  
 POCONx 7-7  
 Port 2-16  
   driver characteristic 7-5  
   edge characteristic 7-6  
   input threshold 7-2  
 Power Down Mode 23-6  
 Power Management 2-19, 23-1  
 Prescaler 6-9  
 Protected  
   Bits 2-21, 4-10  
   instruction 26-4  
 PSW 4-16, 5-10

## **R**

RAM  
   extension 3-9  
   internal 3-4  
 Read/Write Delay 9-16  
 READY 9-18  
 Real Time Clock (->RTC) 15-1  
 Registers 25-1  
   sorted by address 25-14  
   sorted by name 25-4  
 Reserved bits 2-12  
 Reset 22-1  
   Bidirectional 22-4  
   Configuration 22-7, 22-12  
   Hardware 22-2  
   Output 22-8  
   Software 22-3  
   Source indication 14-6  
   Values 22-5  
   Watchdog Timer 22-3  
 RSTCON 22-23

RTC 2-17, 15-1  
 RXCNT 20-43  
 RXCNTB 20-44  
 RXCPU 20-43  
 RXD00-RXD010 20-45  
 RXD10-RXD110 20-46

## **S**

S0BG 11-11  
 S0CON 11-2, 12-3  
 S0EIC, S0RIC, S0TIC, S0TBIC 11-15  
 S0RBUF 11-7, 11-9  
 S0TBUF 11-7, 11-9  
 S1EIC (XP6IC), S1RIC (XP5IC), S1TIC  
   (XP4IC) 12-7  
 SDLM 2-15, 20-1  
   port control 20-23  
 Security Mechanism 23-22  
 Segment  
   Address 9-9, 22-18  
   boundaries 3-12  
 Segmentation 4-20  
   Enable/Disable 4-15  
 Serial Data Link Module 20-1  
 Serial Interface 2-13, 11-1, 12-1  
   Asynchronous 11-5, 12-5  
   CAN 2-14, 19-1  
   IIC 2-14  
   J1850 2-15, 20-1  
   Synchronous 11-8, 12-5, 13-1  
 SFR 3-8, 25-4, 25-14  
 Sharing X-Peripherals 9-39  
 Single Chip Mode 9-2  
   startup configuration 22-20  
 Slave mode  
   External bus 9-35  
   IIC Bus 21-7  
 Sleep Mode 23-5  
 Slow Down Mode 23-10  
 SOFPTR 20-44  
 Software  
   Reset 22-3  
   system configuration 22-22

Traps 5-31  
Source  
  Interrupt 5-2  
  Reset 14-6  
SP 4-27  
Special operation modes (config.) 22-16  
SSC 13-1  
  Baudrate generation 13-13  
  Error Detection 13-15  
  Full Duplex 13-7  
  Half Duplex 13-10  
SSCBR 13-13  
SSCCON 13-2, 13-4  
SSCEIC, SSCRIC, SSCTIC 13-17  
SSCRB, SSCTB 13-8  
Stack 3-5, 4-27, 24-4  
Startup Configuration 22-7, 22-12  
  external reset 22-13  
  single-chip 22-20  
  via software 22-22  
STKOV 4-28  
STKUN 4-29  
Subroutine 24-10  
Synchronous Serial Interface  
  (->SSC) 13-1  
SYSCON 4-13, 9-20  
SYSCON1 23-6  
SYSCON2 23-12  
SYSCON3 23-15

**T**

T01CON 17-5  
T2CON 10-12  
T2IC, T3IC, T4IC 10-21  
T3CON 10-3  
T4CON 10-12  
T5CON 10-30  
T5IC, T6IC 10-38  
T6CON 10-24  
T78CON 17-5  
T7IC 17-9  
T8IC 17-9  
TFR 5-33

Threshold 7-2  
Timer 2-18, 10-1, 10-22  
  Auxiliary Timer 10-12, 10-30  
  CAPCOM 17-4  
  Concatenation 10-16, 10-33  
  Core Timer 10-3, 10-24  
Tools 1-7  
TRANSSTAT 20-32  
Traps 5-31  
Tri-State Time 9-15  
TXCNT 20-40  
TXCPU 20-41  
TXD0-TXD10 20-41  
TxDelay 20-29

**U**

UARn 19-21  
UGML 19-16  
UMLM 19-17  
Unlock Sequence 23-22  
Unseparable instructions 24-14

**V**

Visible mode 9-38

**W**

Waitstate  
  Memory Cycle 9-14  
  Tri-State 9-15  
  XBUS peripheral 9-38  
Watchdog 2-16, 14-1  
  after reset 22-5  
  Oscillator 6-11, 22-19  
  Reset 22-3

WDT 14-2  
WDTCN 14-4

**X**

XBUS 2-10, 9-37  
  enable peripherals 9-38  
  external access 9-39  
  waitstates 9-38  
XP0IC 21-13

XP1IC 21-13

XP7IC 20-48

XPER-Share mode 9-39

XRAM

    on-chip 3-9

    status after reset 22-8

xxIC 5-7

## **Z**

ZEROS 4-33



## Infineon goes for Business Excellence

“Business excellence means intelligent approaches and clearly defined processes, which are both constantly under review and ultimately lead to good operating results.

Better operating results and business excellence mean less idleness and wastefulness for all of us, more professional success, more accurate information, a better overview and, thereby, less frustration and more satisfaction.”

Dr. Ulrich Schumacher

<http://www.infineon.com>