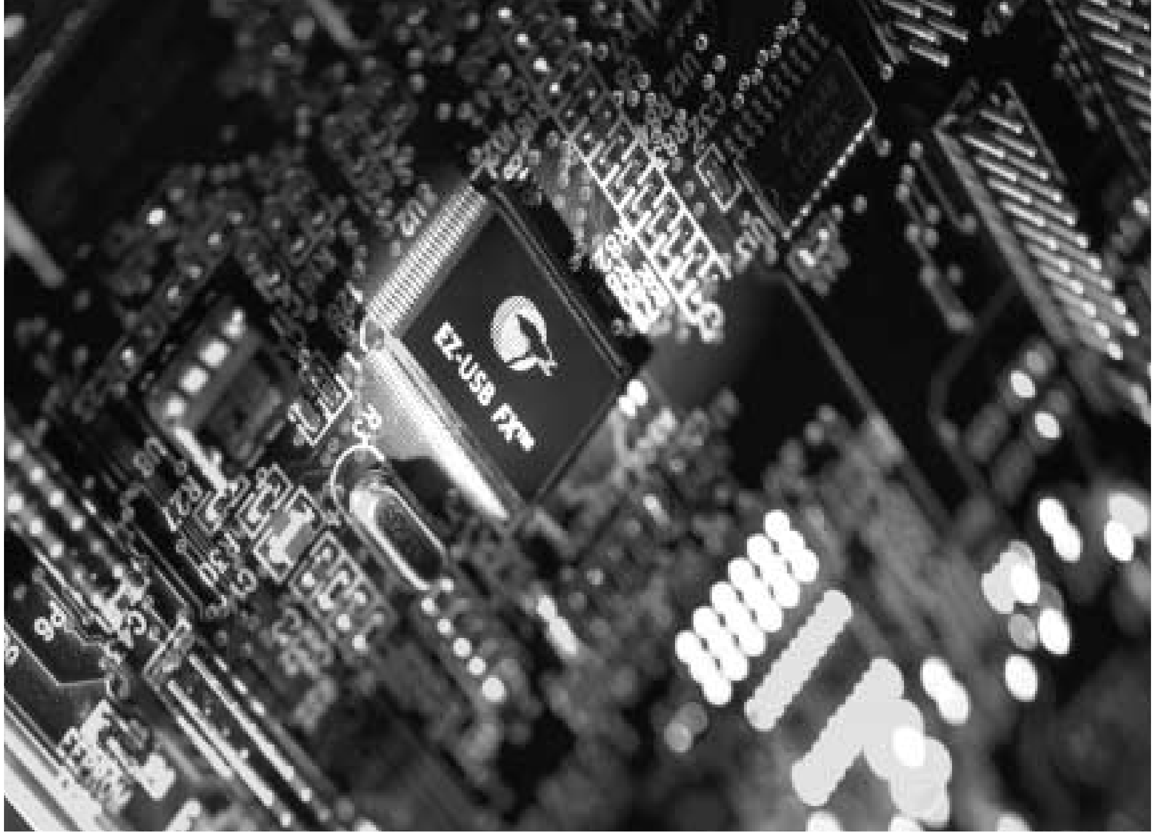
EZ-USB FX
Technical Reference Manual



• Cypress Semiconductor • Interface Products Division •
• 15050 Avenue of Science • Suite 200 • San Diego, CA 92128 •

The EZ-USB FX Technical Reference Manual, Version 1.2.

Table of Contents

List of Figures

(List of Figures)

List of Tables

Chapter 1. Introducing EZ-USB FX

1.1 Introduction

Like a well designed automobile or appliance, a USB peripheral's outward simplicity hides internal complexity. There's a lot going on "under the hood" of a USB device, which gives the user a new level of convenience. For example:

- A USB device can be plugged in anytime, even when the PC is turned on.
- When the PC detects that a USB device has been plugged in, it automatically interrogates the device to learn its capabilities and requirements. From this information, the PC automatically loads the device's driver into the operating system. When the device is unplugged, the operating system automatically logs it off and unloads its driver.
- USB devices do not use DIP switches, jumpers, or configuration programs. There is never an IRQ, DMA, MEMORY, or I/O conflict with a USB device.
- USB expansion hubs make the bus available to dozens of devices.
- USB is fast enough for printers, CD-quality audio, and scanners.

USB is defined in the *Universal Serial Bus Specification Version 1.1*, a 268-page document describing in elaborate detail all aspects of a USB device. The USB Specification is available at **http://usb.org**. The *EZ-USB FX Technical Reference Manual* describes the EZ-USB FX chip along with USB topics that provide help in understanding the USB Specification.

The Cypress Semiconductor EZ-USB FX is a compact, integrated circuit that provides a highly integrated solution for a USB peripheral device. Three key EZ-USB FX features are:

- The EZ-USB FX family provides a *soft* (RAM-based) solution that allows unlimited configuration and upgrades.
- The EZ-USB FX family delivers full USB throughput. Designs that use EZ-USB FX are not limited by number of endpoints, buffer sizes, or transfer speeds.
- The EZ-USB FX family does much of the USB housekeeping in the USB core, simplifying code and accelerating the USB learning curve.

This chapter introduces key USB concepts and terms to make reading this Technical Reference Manual easier.

1.2 EZ-USB FX Block Diagrams

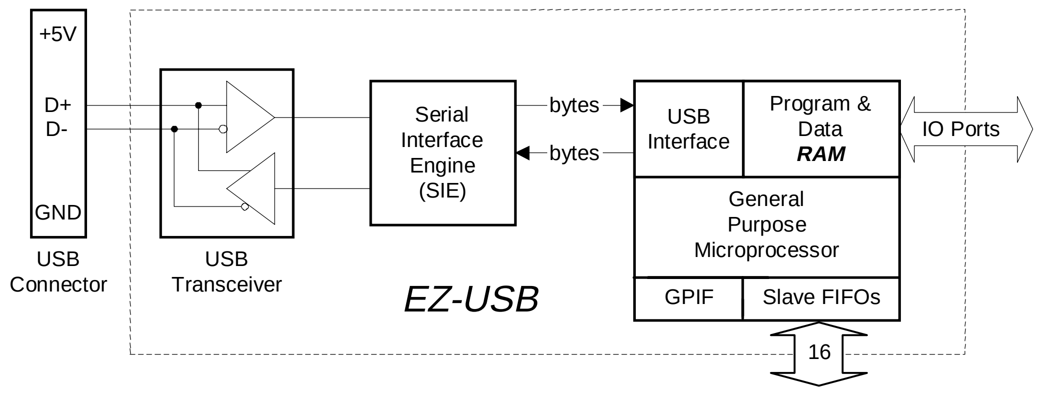


Figure 1-1. CY7C646x3-80NC (80 pin) Simplified Block Diagram

The Cypress Semiconductor EZ-USB FX chip packs the intelligence required by a USB peripheral interface into a compact, integrated circuit. As *Figure 1-1* illustrates, an integrated USB transceiver connects to the USB bus pins D+ and D-. A Serial Interface Engine (SIE) decodes and encodes the serial data and performs error correction, bit stuffing, and other signaling-level details required by USB. Ultimately, the SIE transfers data bytes to and from the USB interface.

The internal microprocessor is an enhanced 8051 with fast execution time and added features. It uses internal RAM for program and data storage, making the EZ-USB FX family a *soft* solution. The USB host downloads 8051 program code and device personality into RAM over the USB bus, and then EZ-USB FX re-connects as the custom device, as defined by the loaded code.

The EZ-USB FX family uses an enhanced SIE/USB interface (USB Core), which has the intelligence to function as a full USB device, even before the 8051 runs. The enhanced core simplifies 8051 code by implementing much of the USB protocol, itself.

EZ-USB FX chips operate at 3.3V. This simplifies the design of bus-powered USB devices, since the 5V power available in the USB connector (USB Specification allows power to be as low as 4.4V) can drive a 3.3V regulator to deliver clean, isolated power to the EZ-USB FX chip.

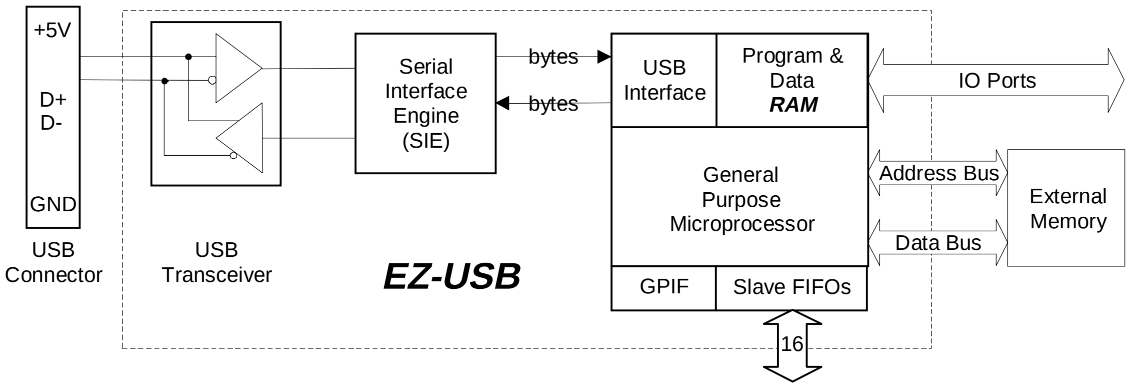


Figure 1-2. CY7C646x3-128NC (128 pin) Simplified Block Diagram

Figure 1-2 illustrates the CY7C646x3-128NC, a 128-pin version of the EZ-USB FX family. In addition to the 40 I/O pins, it contains a 16-bit address bus and an 8-bit data bus for external memory expansion. Slave interface FIFOs and a General Programmable Interface (GPIF) controller provide a flexible, high-bandwidth interface to external logic.

Also included, the DMAEXTFIFO register provides legacy support for invoking the fast transfer mode available on the EZ-USB Series 2100. This allows data to move directly between external logic and internal USB FIFOs. This, along with abundant endpoint resources, allows the EZ-USB FX family to support transfer bandwidths to external logic that exceed the USB delivery/consumption rate.

1.3 The USB Specification

The *Universal Serial Bus Specification Version 1.1* is available on the Internet at **http://usb.org**. Published in January 1998, the USB Specification is the work of a founding committee of seven industry heavyweights: Compaq, DEC, IBM, Intel, Microsoft, NEC, and Northern Telecom. This impressive list of developers secures USB as the low-to-medium speed PC connection method of the future.

A glance at the USB Specification makes it immediately apparent that USB is not nearly as simple as the customary serial or parallel port. The USB Specification uses new terms like “endpoint,” isochronous,” and “enumeration,” and finds new uses for old terms like “configuration,” “interface,” and “interrupt.” Woven into the USB fabric is a software abstraction model that deals with things such as “pipes.” The USB Specification also contains detail about the connector types and wire colors.

1.4 Tokens and PIDs

In this manual, statements like the following appear: “When the host sends an IN token...,” or “The device responds with an ACK.” What do these terms mean? A USB transaction consists of data packets identified by special codes called Packet IDs or PIDs. A PID signifies what kind of packet is being transmitted. There are four PID types, shown in Table 1-1.

Table 1-1. USB PIDs

PID Type	PID Name
Token	IN, OUT, SOF, SETUP
Data	DATA0, DATA1
Handshake	ACK, NAK, STALL
Special	PRE

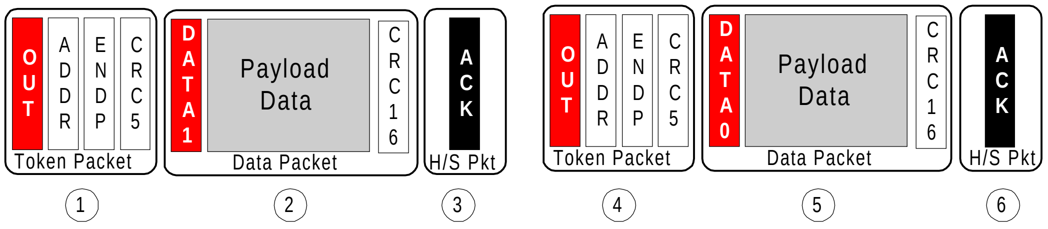


Figure 1-3. USB Packets

Figure 1-3 illustrates a USB transfer. Packet **1** is an OUT token, indicated by the OUT PID. The OUT token signifies that data from the host is about to be transmitted over the bus. Packet **2** contains data, as indicated by the DATA1 PID. Packet **3** is a handshake packet, sent by the device using the ACK (acknowledge) PID to signify to the host that the device received the data error-free.

Continuing with *Figure 1-3*, a second transaction begins with another OUT token **4**, followed by more data **5**, this time using the DATA0 PID. Finally, the device again indicates success by transmitting the ACK PID in a handshake packet **6**.

Why two DATA PIDs, DATA0 and DATA1? It’s because the USB architects took error correction very seriously. As mentioned previously, the ACK handshake is an indication to the host that the peripheral received data without error (the CRC portion of the packet is used to detect errors). But what if the handshake packet itself is garbled in transmission? To detect this, each side (host and device) maintains a *data toggle* bit, which is toggled between data packet transfers. The state of this internal toggle bit is compared with the PID that arrives with the data, either DATA0 or DATA1. When sending data, the host or device sends alternating DATA0-DATA1 PIDs. By comparing the Data PID with the state of the internal toggle bit, the host or device can detect a corrupted handshake packet.

SETUP tokens are unique to CONTROL transfers. They preface eight bytes of data from which the peripheral decodes host Device Requests.

SOF tokens occur once per millisecond, denoting a USB *frame*.

There are three handshake PIDs: ACK, NAK, and STALL.

- ACK means *success*; the data was received error-free.
- NAK means "busy, try again." It's tempting to assume that NAK means "error," but it doesn't. A USB device indicates an error by *not responding*.
- STALL means that something unforeseen went wrong (probably as a result of miscommunication or lack of cooperation between the software and firmware writers). A device sends the STALL handshake to indicate that it doesn't understand a device request, that something went wrong on the peripheral end, or that the host tried to access a resource that wasn't there. It's like HALT, but better, because USB provides a way to recover from a stall.

A PRE (Preamble) PID precedes a low-speed (1.5 Mbps) USB transmission. The EZ-USB FX family supports high-speed (12 Mbps) USB transfers only. It ignores PRE packets and the resultant low-speed transfer.

1.5 Host is Master

This is a fundamental USB concept. There is exactly one master in a USB system: the host computer. **USB devices respond to host requests**. USB devices cannot send information between themselves, as they could if USB were a peer-to-peer topology.

However, there is one case where a USB device can initiate signaling without prompting from the host. After being put into a low-power suspend mode by the host, a device can signal a remote wakeup. A Remote Wakeup is the only method in which the USB becomes the initiator. Everything else happens because the host makes device requests, and the device responds to them.

There's an excellent reason for this host-centric model. The USB architects were keenly mindful of cost, and the best way to make low-cost peripherals is to put most of the smarts into the host side, the PC. If USB had been defined as peer-to-peer, every USB device would have required more intelligence, raising cost.

Here are two important consequences of the "host is master" concept:

1.5.1 Receiving Data from the Host

To send data to a USB peripheral, the host issues an OUT token, followed by the data. If the peripheral has space for the data, and accepts it without error, it returns an ACK to the host. If it is busy, it sends a NAK. If it finds an error, it sends back nothing. For the latter two cases, the host resends the data at a later time.

1.5.2 Sending Data to the Host

A USB device never spontaneously sends data to the host. Nevertheless, in the EZ-USB FX chip, there's nothing to stop the 8051 from loading data for the host into an endpoint buffer (see *"EZ-USB FX Endpoints"*, this chapter) and *arming* it for transfer. However, the data remains in the buffer *until the host sends an IN token to that particular endpoint*. If the host never sends the IN token, the data remains there indefinitely.

1.6 USB Direction

Once you accept that the host is the bus master, it's easy to remember USB direction: OUT means from the host to the device, and IN means from the device to the host. EZ-USB FX nomenclature uses this naming convention. For example, an endpoint that sends data to the host is an IN endpoint. This can be confusing at first, because the 8051 *sends* data by loading an IN endpoint buffer. Keep in mind that an 8051*out* is an IN to the host.

1.7 Frame

The USB host provides a time base to all USB devices by transmitting a SOF (Start Of Frame) packet every millisecond. The SOF packet includes an incrementing, 11-bit frame count. The 8051 can read this frame count from two EZ-USB FX registers. SOF-time has significance for isochronous endpoints; it's the time that the *ping-ponging* buffers switch places. The USB core provides the 8051 with an SOF interrupt request for servicing isochronous endpoint data.

1.8 EZ-USB FX Transfer Types

USB defines four transfer types. These match the requirements of different data types delivered over the bus. (*"EZ-USB FX Endpoints"* explains how the EZ-USB FX family supports the four transfer types.)

1.8.1 Bulk Transfers

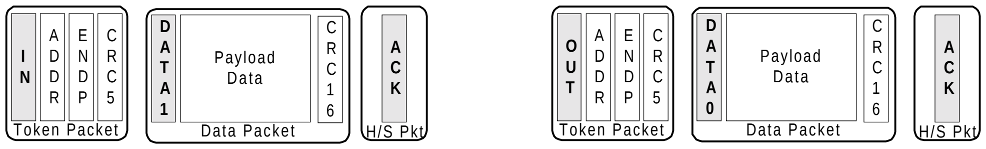


Figure 1-4. Two Bulk Transfers, IN and OUT

Bulk data is *bursty*, traveling in packets of 8, 16, 32, or 64 bytes. Bulk data has guaranteed accuracy, due to an automatic re-try mechanism for erroneous data. The host schedules bulk packets when there is available bus time. Bulk transfers are typically used for printer, scanner, or modem data. Bulk data has built-in flow control provided by handshake packets.

1.8.2 Interrupt Transfers

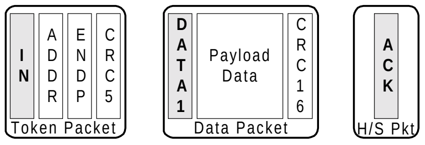


Figure 1-5. An Interrupt Transfer

Interrupt data is like bulk data; it can have packet sizes of 1 through 64 bytes. Interrupt endpoints have an associated polling interval that ensures they will be *pinged* (receive an IN token) by the host on a regular basis.

1.8.3 Isochronous Transfers

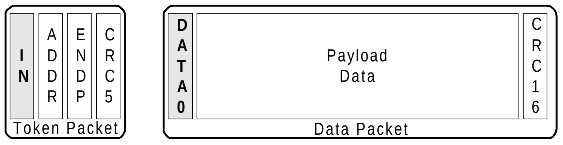


Figure 1-6. An Isochronous Transfer

Isochronous data is time-critical and used to *stream* data like audio and video. Time of delivery is the most important requirement for isochronous data. In every USB frame, a certain amount of USB bandwidth is allocated to isochronous transfers. To lighten the overhead, isochronous transfers have no handshake (ACK/NAK/STALL), and no retries. Error detection is limited to a 16-bit CRC. Isochronous transfers do not use the data toggle mechanism. Isochronous data uses only the DATA0 PID.

1.8.4 Control Transfers

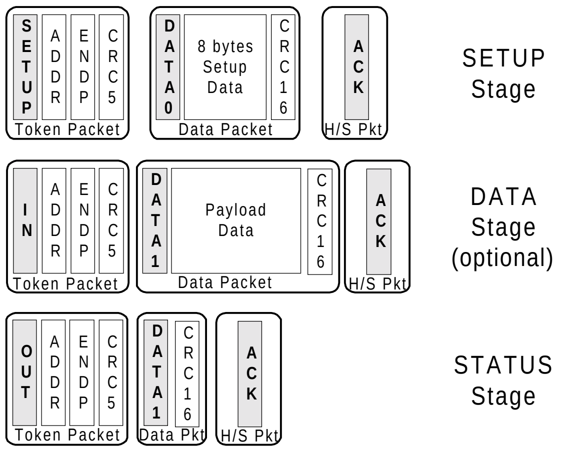


Figure 1-7. A Control Transfer

Control transfers configure and send commands to a device. Being *mission critical*, they employ the most extensive USB error checking USB. Control transfers are delivered on a *best effort* basis by the host (*best effort* is a six-step process defined by the *Universal Serial Bus Specification Version 1.1, "Section 5.5.4"*). The host reserves a part of each USB frame time for Control transfers.

Control transfers consist of two or three stages. The SETUP stage contains eight bytes of USB CONTROL data. An optional DATA stage contains more data, if required. The STATUS (or *handshake*) stage allows the device to indicate successful completion of a control operation.

1.9 Enumeration

Your computer is ON. You plug in a USB device, and the Windowsµ cursor switches to an hourglass, and then back to a cursor. Magically, your device is connected, and its Windowsµ driver is loaded! Anyone who has installed a sound card into a PC and had to configure countless jumpers, drivers, and IO/Interrupt/DMA settings knows that a USB connection is miraculous. We've all *heard* about Plug and Play, but USB delivers the real thing.

How does all this happen automatically? Inside every USB device is a table of *descriptors*. This table is the sum total of the device's requirements and capabilities. When you plug into USB, the host goes through a *sign-on* sequence:

1. The host sends a "Get_Descriptor/Device" request to address zero (devices must respond to address zero when first attached).
2. The device responds to the request by sending ID data back to the host to define itself.
3. The host sends the device a *Set_Address* request, which gives it a unique address to distinguish it from the other devices connected to the bus.
4. The host sends more *Get_Descriptor* requests, asking more device information. From this, it learns everything else about the device, like how many endpoints the device has, its power requirements, what bus bandwidth it requires, and what driver to load.

This sign-on process is called *Enumeration*.

1.10 The USB Core

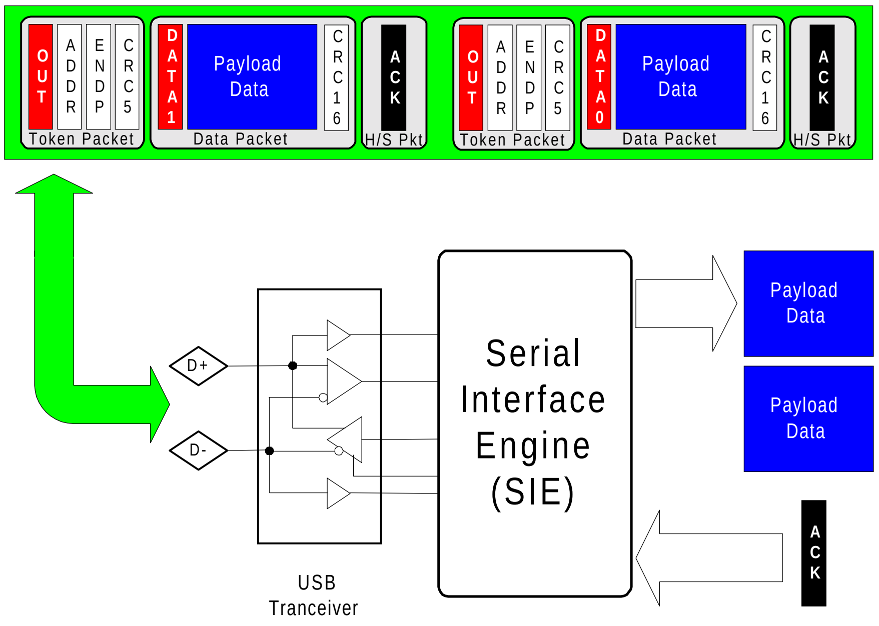


Figure 1-8. What the SIE Does

Every USB device has a Serial Interface Engine (SIE). The SIE connects to the USB data lines D+ and D-, and delivers bytes to and from the USB device. *Figure 1-8* illustrates a USB bulk transfer, with time moving from left to right. The SIE decodes the packet PIDs, performs error checking on the data using the transmitted CRC bits, and delivers payload data to the USB device. If the SIE encounters an error in the data, it automatically indicates *no response* instead of supplying a handshake PID. This instructs the host to re-transmit the data at a later time.

Bulk transfers, such as the one illustrated in *Figure 1-8*, are *asynchronous*, meaning that they include a flow control mechanism using ACK and NAK handshake PIDs. The SIE indicates *busy* to the host by sending a NAK handshake packet. When the peripheral device has successfully transferred the data, it commands the SIE to send an ACK handshake packet, indicating success.

To send data to the host, the SIE accepts bytes and control signals from the USB device, formats it for USB transfer, and sends it over the two-wire USB. Because the USB uses a self-clocking data format (NRZI), the SIE also inserts bits at appropriate places in the bit stream to guarantee a certain number of transitions in the serial data. This is called “bit stuffing,” and is handled transparently by the SIE.

One of the most important features of the EZ-USB FX family is that it is *soft*. Instead of requiring ROM or other fixed memory, it contains internal program/data RAM downloaded over the USB to give the device its unique personality. This makes modifications, specification revisions, and updates a snap.

The EZ-USB FX family can connect as a USB device and download code into internal RAM. All while, its internal 8051 is held in Reset. This is done by an enhanced SIE, which performs all the work shown in *Figure 1-8*, and more. It contains additional logic to perform a full enumeration, using an internal table of descriptors. It also responds to a vendor specific “Firmware Download” device request to load its internal RAM. Additionally, the added SIE functionality is made available to the 8051. This saves 8051 code and processing time.

Throughout this manual, the SIE and its enhancements are referred to as the “USB Core.”

1.11 EZ-USB FX Microprocessor

The EZ-USB FX microprocessor is an enhanced 8051 core. Use of an 8051-compatible processor makes available immediately extensive software support tools to the EZ-USB FX designer. This enhanced 8051 core (described in *Chapter 2. "EZ-USB FX CPU"*, *Chapter 16. "8051 Introduction"*, *Chapter 17. "8051 Architectural Overview"*, and *Chapter 18. "8051 Hardware Description"*) has the following features:

- 4 clocks/cycle, compared to the 12 clocks/cycle of a standard 8051:a 10X speed improvement.
- 48-MHz clock.
- DMA for 48 MB/second memory-to-memory transfers. Dual data pointers for improved XDATA access.
- Two UARTs.

- Three counter-timers.
- Expanded interrupt system.
- 256 bytes of internal register RAM.
- Standard 8051 instruction set—if you know the 8051, you know EZ-USB FX.

The enhanced 8051 core uses on-chip RAM as program and data memory, giving EZ-USB FX its *soft* feature. *Chapter 3. "EZ-USB FX Memory"* describes the various memory options.

The 8051 communicates with the SIE using a set of registers, occupying the top of the on-chip RAM address space. These registers are grouped and described by function in individual chapters of this reference manual and summarized in register order in *Chapter 15. "EZ-USB FX Registers"*.

The EZ-USB 8051 has two duties. First, it participates in the protocol defined in the *Universal Serial Bus Specification Version 1.1, "Chapter 9, USB Device Framework."* Thanks to EZ-USB FX enhancements to the SIE and USB interface, the 8051 firmware associated with USB overhead is simplified, leaving code space and bandwidth available for the 8051's primary duty, to help implement your device. On the device side, abundant input/output resources are available, including I/O ports, UARTs, and an I^2C-compatible bus master controller. These resources are described in *Chapter 4. "EZ-USB FX Input/Output"*

1.12 ReNumeration™

Because the EZ-USB FX chip is *soft*, it can take on the identities of multiple distinct USB devices. The first device downloads your 8051 firmware and USB descriptor tables over the USB cable when the peripheral device is plugged in. Once downloaded, another device comes on as a totally different USB peripheral as defined by the downloaded information. This patented two-step process, called ReNumeration™, happens instantly when the device is plugged in, with no hint that the initial load step has occurred.

Chapter 5. "EZ-USB FX Enumeration & ReNumeration™" describes this feature in detail, along with other EZ-USB FX boot (startup) modes.

1.13 EZ-USB FX Endpoints

The USB Specification defines an endpoint as a source or sink of data. Since USB is a serial bus, a device endpoint is actually a FIFO, which sequentially empties/fills with USB bytes. The host selects a device endpoint by sending a 4-bit address and one direction bit. Therefore, USB can uniquely address 32 endpoints, IN0 through IN15 and OUT0 through OUT15.

From the EZ-USB FX point of view, an endpoint is a buffer full of bytes received or held for transmitted over the bus. The 8051 reads endpoint data from an OUT buffer, and writes endpoint data for transmission over USB to an IN buffer.

There are four USB endpoint types: Bulk, Control, Interrupt, and Isochronous. These endpoint types are described in the following paragraphs:

1.13.1 EZ-USB FX Bulk Endpoints

Bulk endpoints are unidirectional—one endpoint address per direction. Therefore, endpoint 2-IN is addressed differently than endpoint 2-OUT. Bulk endpoints use maximum packet sizes (buffer sizes) of 8, 16, 32, or 64 bytes. EZ-USB FX provides fourteen bulk endpoints, divided into seven IN endpoints (endpoint 1-IN through 7-IN), and seven OUT endpoints (endpoint 1-OUT through 7-OUT). Each of the fourteen endpoints has a 64-byte buffer.

Bulk data is available to the 8051 in RAM or as FIFO data, using a special EZ-USB FX *Autopointer* (*Chapter 6. "EZ-USB FX Bulk Transfers"*).

1.13.2 EZ-USB FX Control Endpoint Zero

Control endpoints transfer mission-critical control information to and from the USB device. The USB Specification requires every USB device to have a default CONTROL endpoint, endpoint zero. Device enumeration, the process that the host initiates when the device is first plugged in, is conducted over endpoint zero. The host sends all USB requests over endpoint zero.

Control endpoints are bi-directional. If you have an endpoint 0 IN CONTROL endpoint, you automatically have an endpoint 0 OUT endpoint. Only Control endpoints accept SETUP PIDs.

A CONTROL transfer consists of a two or three stage sequence:

- SETUP
- DATA (If needed)
- HANDSHAKE

Eight bytes of data in the SETUP portion of the CONTROL transfer have special USB significance, as defined in the *Universal Serial Bus Specification Version 1.1, "Chapter 9."* A USB device must respond properly to the requests described in this chapter to pass USB compliance testing (referred to as the USB "Chapter Nine Test").

Endpoint zero is the only CONTROL endpoint in the EZ-USB FX chip. The 8051 responds to device requests issued by the host over endpoint zero. The USB core is significantly enhanced to simplify the 8051 code required to service these requests. *Chapter 9. "EZ-USB FX Endpoint Zero"* provides a detailed roadmap for writing compliant USB Chapter 9 8051 code.

1.13.3 EZ-USB FX Interrupt Endpoints

Interrupt endpoints are almost identical to bulk endpoints. Fourteen EZ-USB FX endpoints (EP1-EP7, IN, and OUT) may be used as interrupt endpoints. Interrupt endpoints have a maximum packet size 64. They contain a “polling interval” byte in their descriptor to tell the host how often to service them. The 8051 transfers data over interrupt endpoints in exactly the same way as for bulk endpoints. Interrupt endpoints are described in *Chapter 6. "EZ-USB FX Bulk Transfers."*

1.13.4 EZ-USB FX Isochronous Endpoints

Isochronous endpoints deliver high bandwidth, time critical data over USB. Isochronous endpoints are used to stream data to devices such as audio DACs, and from devices such as video cameras. Time of delivery is the most critical requirement, and isochronous endpoints are tailored to this requirement. Once a device has been granted an isochronous bandwidth slot by the host, it is guaranteed the ability to send or receive its data every frame.

EZ-USB FX contains 16 isochronous endpoints, numbered 8-15 (8IN-15IN, and 8OUT-15OUT). 1,024 bytes of FIFO memory are available to the 16 endpoints, and may be divided among them. EZ-USB FX actually contains 2,048 bytes of isochronous FIFO memory to provide double-buffering. Using double buffering, the 8051 reads OUT data from isochronous endpoint FIFOs containing data from the previous frame, while the host writes current frame data into the other buffer. Similarly, the 8051 loads IN data into isochronous endpoint FIFOs that will be transmitted over USB during the next frame, while the host reads current frame data from the other buffer. At every SOF the USB FIFOs and 8051 FIFOs switch, or *ping-pong*.

Isochronous transfers are described in *Chapter 10. "EZ-USB FX Isochronous Transfers."*

1.14 Interrupts

EZ-USB FX adds seven interrupt sources to the standard 8051 interrupt system. Three of the added interrupts are available on device pins: INT4, INT5#, and INT6. The other four are used internally: INT2 is used for all USB interrupts, INT3 is used by the I^{2}C-compatible interface, INT4 is used by the FIFOs and GPIF, and the remaining interrupt is used for remote wakeup indication.

The USB core automatically supplies jump vectors (Autovectors) for its USB and FIFO interrupts to save the 8051 from having to test bits to determine the source of the interrupt. Each INT2 and INT4 interrupt source has its own vector. When an interrupt requires service, the proper ISR (interrupt service routine) is automatically invoked. *Chapter 12. "EZ-USB FX Interrupts"* describes the EZ-USB FX interrupt system.

1.15 Reset and Power Management

The EZ-USB FX chip contains four resets:

- Power-On-Reset (POR)
- USB bus reset
- 8051 reset
- USB Disconnect/Re-connect

The functions of the various EZ-USB FX resets are described in *Chapter 13. "EZ-USB FX Resets"*

A USB peripheral may be put into a low power state when the host signals a *suspend* operation. The USB Specification states that a bus-powered device cannot draw more than 500 ΩA of current from the VBUS wire while in suspend. The EZ-USB FX chip contains logic to turn off its internal oscillator and enter a *sleep* state. A special interrupt, triggered by a wakeup pin or wakeup signaling on the USB bus, starts the oscillator and interrupts the 8051 to resume operation.

Low power operation is described in *Chapter 14. "EZ-USB FX Power Management*".

1.16 Slave FIFOs

The EZ-USB FX contains four 64-byte FIFOs to provide a flexible, high-speed interface to a variety of peripherals. These FIFOs can be *slave FIFOs* that accept RD/WR strobes from an external source, or the *GPIF* can be their bus master. See *"GPIF (General Programmable Interface)"* below for more information. Two FIFOs are provided in the IN direction, and two FIFOs transfer data in the OUT direction.

The FIFO module allows the EZ-USB FX to perform the following functions:

- Act as a FIFO or a small RAM on a microprocessor bus
- Create a 16-bit data path
- Transfer data at speeds up to 96 MB per second (burst).

1.17 GPIF (General Programmable Interface)

The GPIF is a programmable state machine that runs at 48 MHz. It can be used to generate custom bus waveforems and control the FIFOs. It has four programs of seven steps each that execute at 48 MHz. The GPIF program can modify the CTL0-5 lines, branch on the RDY0-5 inputs, and control FIFO data movement.

The GPIF is used to implement many standard interfaces available for the FX, including:

- IDE (ATAPI)
- PC parallel port (EPP)
- Utopia

The GPIF is fully described in *Chapter 8. "General Programmable Interface (GPIF)."*

1.18 EZ-USB FX Product Family

The EZ-USB FX family is available in various pinouts to serve different system requirements and costs. Table 1-2 shows the feature set for each member of the EZ-USB FX family.

Table 1-2. EZ-USB FX Family

Part Number	Package	Ram	ISO Support	I/O	FIFO Width	Addr/Data Bus
CY7C64601-52NC	52-pin PQFP	4 K	No	16	8 Bits	No
CY7C64603-52NC	52-pin PQFP	8 K	No	18	8 Bits	No
CY7C64613-52NC	52-pin PQFP	8 K	Yes	18	8 Bits	No
CY7C64603-80NC	80-pin PQFP	8 K	No	32	16 Bits	No
CY7C64613-80NC	80-pin PQFP	8 K	Yes	32	16 Bits	No
CY7C64603-128NC	128-pin PQFP	8 K	No	40	16 Bits	Yes
CY7C64613-128NC	128-pin PQFP	8 K	Yes	40	16 Bits	Yes

Chapter 2. EZ-USB FX CPU

2.1 Introduction

The EZ-USB FX built-in microprocessor, an enhanced 8051 core, is fully described in *Chapter 16. "8051 Introduction"* , *Chapter 17. "8051 Architectural Overview"* and *Chapter 18. "8051 Hardware Description."* This chapter introduces the processor, its interface to the USB core, and describes architectural differences from a standard 8051.

2.2 8051 Enhancements

The enhanced 8051 core uses the standard 8051 instruction set. Instructions execute faster than with the standard 8051 due to two features:

- Wasted bus cycles are eliminated. A bus cycle uses four clocks, as compared to 12 clocks with the standard 8051.
- The 8051 runs at 24 MHz or 48 MHz.

In addition to speed improvement, the enhanced 8051 core also includes architectural enhancements:

- A second data pointer
- A second UART
- A third, 16-bit timer (TIMER2)
- A high-speed memory interface with a non-multiplexed 16-bit address bus
- Eight additional interrupts (INT2-INT6, WAKEUP, T2, and UART1)
- Variable MOVX timing to accommodate fast/slow RAM peripherals
- 3.3V operation.

2.3 EZ-USB FX Enhancements

EZ-USB FX provides additional enhancements outside the 8051. These include:

- DMA Module
- Fast external transfers (Autopointer, DMAEXTFIFO)
- Vectored USB interrupts (Autovector)
- Separate buffers for SETUP and DATA portions of a CONTROL transfer
- Breakpoint Facility.

2.4 EZ-USB FX Register Interface

The 8051 communicates with the USB core through a set of memory mapped registers. These registers are grouped as follows:

- Endpoint buffers and FIFOs
- Slave FIFOs
- 8051 control
- I/O ports
- DMAEXTFIFO
- I^2C-Compatible Controller
- Interrupts
- USB Functions
- GPIF

These registers and their functions are described throughout this manual. A full description of every register and bit appears in *Chapter 15. "EZ-USB FX Registers."*

2.5 EZ-USB FX Internal RAM

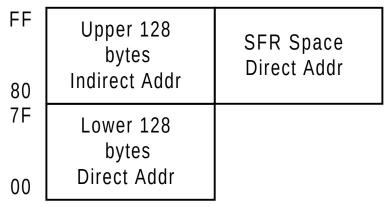


Figure 2-1. 8051 Registers

Like the standard 8051, the EZ-USB 8051 core contains 128 bytes of register RAM at address 00-7F, and a partially populated SFR register space at address 80-FF. An additional 128 indirectly addressed registers (sometimes called “IDATA”) are also available at address 80-FF.

All internal EZ-USB FX RAM, which includes program/data memory, bulk endpoint buffer memory, and the EZ-USB FX register set, is addressed as *add-on* 8051 memory. The 8051 reads or writes these bytes as data using the MOVX (move external) instruction. Even though the MOVX instruction implies external memory, the EZ-USB FX RAM and register set is actually inside the EZ-USB FX chip. External memory attached to the CY7C646x3-128NC address and data busses can also be accessed by the MOVX instruction. The USB core encodes its memory strobe and select signals (RD#, WR#, CS#, and OE#) to eliminate the need for external logic to separate the internal and external memory spaces.

2.6 I/O Ports

A standard 8051 communicates with its I/O ports 0-3 through four Special Function Registers (SFRs). **The USB core implements I/O ports differently than a standard 8051**, as described in *Chapter 4. "EZ-USB FX Input/Output."* The USB core implements a flexible I/O system that is controlled *via* SFRs or *via* the EZ-USB FX register set. Although 8051 SFR bits may be used to control the I/O pins, their addresses and functions are different than in a standard 8051. Each EZ-USB FX I/O pin functions identically, having the following resources:

- An output latch (OUTn). Used when the pin is an output port.
- A register (PINSn) that indicates the state of the I/O pins, regardless of its configuration (input or output).
- An output enable register (OEn) that causes the I/O pin to be driven from the output latch.

Several registers control whether the pin is a port pin or a special function pin. These registers include PORTnCFG, IFCONFIG, PORTACF2, and PORTCGPIF. See *"Port Configuration Tables"* in *Chapter 4. "EZ-USB FX Input/Output"*.

2.7 Interrupts

All standard 8051 interrupts are supported in the enhanced 8051 core. Table 2-1 shows the existing and added 8051 interrupts, and indicates how the added ones are used.

Table 2-1. EZ-USB FX Interrupts

Standard 8051 Interrupts	Enhanced 8051 Interrupts	Used As
INT0		Device Pin INT0#
INT1		Device Pin INT1#
Timer 0		Internal, Timer 0
Timer 1		Internal, Timer 1
Tx0 & Rx0		Internal, UART0
	INT2	Internal, USB
	INT3	Internal, I^{2}C-compatible Controller
	INT4	Internal, FIFO Interrupt
	INT5	Device Pin, PB5/INT5#
	INT6	Device Pin, PB6/INT6
	WAKEUP	Device Pin, USB WAKEUP#
	Tx1 & Rx1	Internal, UART1
	Timer 2	Internal, Timer 2

The EZ-USB FX chip uses 8051 INT2 for 22 different USB interrupts: 17 bulk endpoints plus SOF, Suspend, SETUP Data, SETUP Token, and USB Bus Reset. To help the 8051 determine which interrupt is active, the USB core provides a feature called Autovectoring. The core inserts an address byte into the low byte of the 3-byte jump instruction found at the 8051 INT2 vector address. This second level of vectoring automatically transfers control to the appropriate USB ISR. The Autovector mechanism, as well as the EZ-USB FX interrupt system is the subject of *Chapter 12. "EZ-USB FX Interrupts."*

2.8 Power Control

The USB core implements a power-down mode that allows it to be used in USB bus-powered devices that must draw no more than 500 ΩA when suspended. Power control is accomplished using a combination of 8051 and USB core resources. The mechanism by which EZ-USB FX powers down for suspend, and then re-powers to resume operation, is described in detail in *Chapter 14. "EZ-USB FX Power Management."*

EZ-USB FX responds to USB suspend using three 8051 resources: the *idle* mode and two interrupts. A USB suspend operation is indicated by a lack of bus activity for 3 ms. The USB core detects this, and asserts an interrupt request via the USB interrupt (8051 INT2). The ISR (Interrupt Service Routine) turns off external sub-systems that draw power. When ready to suspend operation, the 8051 sets an SFR bit, PCON.0. This bit causes the 8051 to suspend, waiting for an interrupt.

When the 8051 sets PCON.0, a control signal from the 8051 to the USB core causes the core to shut down the 12-MHz oscillator and internal PLL. This stops all internal clocks to allow the USB core and 8051 to enter a very low power mode.

The suspended EZ-USB FX chip can be awakened two ways: USB bus activity may resume, or an EZ-USB FX pin (WAKEUP#) can be asserted to activate a USB *Remote Wakeup*. Either event triggers the following chain of events:

- The USB core re-starts the 12-MHz oscillator and PLL, and waits for the clocks to stabilize.
- The USB core asserts a high-priority 8051 interrupt to signal a resume interrupt.
- The 8051 vectors to the resume ISR and, upon completion, resumes executing code at the instruction following the instruction that set the PCON.0 bit to 1.

2.9 SFRs

The EZ-USB FX family was designed to keep 8051 coding as standard as possible, to allow easy integration of existing 8051 software development tools. The added 8051 SFR registers and bits are summarized in Table 2-2.

Table 2-2. Added Registers and Bits

8051 Enhancements	SFR	Addr	Function
Dual Data Pointers	DPL0	0x82	Data Pointer 0 Low Addr
	DPH0	0x83	Data Pointer 0 High Addr
	DPL1	0x84	Data Pointer 1 Low Addr
	DPH1	0x85	Data Pointer 1 High Addr
	DPS	0x86	Data Pointer Select (LSB)
	MPAGE	0x92	Replaces standard 8051 Port 2 for indirect external data memory addressing using R0 or R1
Timer 2	T2CON.6-7	0xC8	Timer 2 Control
	RCAP2L	0xCA	T2 Capture/Reload Value L
	RCAP2H	0xCB	T2 Capture/Reload Value H
	T2L	0xCC	T2 Count L
	T2H	0xCD	T2 Count H
	IE.5	0xA8	ET2-Enable T2 Interrupt Bit
	IP.5	0xB8	PT2-T2 Interrupt Priority Control
UART1	SCON1.0-1	0xC0	Serial Port 1 Control
	SBUF1	0xC1	Serial Port 1 Data
	IE.6	0xA8	ES1-SIO1 Interrupt Enable Bit
	IP.6	0xB8	PS1-SIO1 Interrupt Priority Control
	EICON.7	0xD8	SMOD1-SIO1 Baud Rate Doubler
Interrupts			
INT2-INT5	EXIF	0x91	INT2-INT5 Interrupt Flags
	EIE	0xE8	INT2-INT5 Interrupt Enables
	EIP.0-3	0xF8	INT2-INT5 Interrupt Priority Control
INT6	EICON.3	0xD8	INT6 Interrupt Flag
	EIE.4	0xE8	INT6 Interrupt Enable
	EIP.4	0xF8	INT6 Interrupt Priority Control
WAKEUP#	EICON.4	0xD8	WAKEUP# Interrupt Flag
	EICON.5	0xD8	WAKEUP# Interrupt Enable
Expanded SFRs			
I/O Registers	IOA	0x80	Input/Output A
	IOB	0x90	Input/Output B
	IOC	0xA0	Input/Output C
	IOD	0xB0	Input/Output D
	IOE	0xB1	Input/Output E
Interrupt Clears	INT2CLR	0xA1	Interrupt 2 Clear
	INT4CLR	0xA2	Interrupt 4 Clear
Enables	SOEA	0xB2	Output Enable A
	SOEB	0xB3	Output Enable B
	SOEC	0xB4	Output Enable C

8051 Enhancements	SFR	Addr	Function
	SOED	0xB5	Output Enable D
	SOEE	0xB6	Output Enable E
Idle Mode	PCON.0	0x87	EZ-USB FX Power Down (Suspend)

Members of the EZ-USB FX family that supply pins to expand 8051 memory provide separate non-multiplexed 16-bit address and 8-bit data busses. This differs from the standard 8051, which multiplexes eight device pins between three sources: I/O port 0, the external data bus, and the low byte of the address bus. A standard 8051 system with external memory requires a de-multiplexing address latch, strobed by the 8051 ALE (Address Latch Enable) pin. The external latch is not required by the non-multiplexed EZ-USB FX chip, and no ALE signal is provided. In addition to eliminating the customary external latch, the non-multiplexed bus saves one cycle per memory fetch cycle, further improving 8051 performance.

2.10 Internal Bus

The typical 8051 user must choose between using Port 0 as a memory expansion port or as an I/O port. The CY7C646x3-128NC provides a separate I/O system with its own control registers (in external memory space), and provides the I/O port signals on dedicated (not shared) pins. This allows the external data bus to expand memory without sacrificing I/O pins.

The 8051 is the sole master of the memory expansion bus. It provides read and write signals to external memory. The address bus is output-only.

The DMAEXTFIFO register provides legacy support for invoking the fast transfer mode available on the EZ-USB Series 2100. Refer to *"Dummy Register"* in *Chapter 11. "EZ-USB FX DMA System"* for more information about this register.

2.11 Reset

The internal 8051 RESET signal is not directly controlled by the EZ-USB FX RESET pin. Instead, it is controlled by an EZ-USB FX register bit accessible to the USB host. When the EZ-USB FX chip is powered, the 8051 is held in reset. Using the default USB device (enumerated by the USB core), the host downloads code into RAM. Finally, the host clears an EZ-USB FX register bit that takes the 8051 out of reset.

The EZ-USB FX family also operates with external non-volatile memory, in which case the 8051 exits the reset state automatically at power-on. The various EZ-USB FX resets and their effects are described in *Chapter 13. "EZ-USB FX Resets."*

Chapter 3. EZ-USB FX Memory

3.1 Introduction

EZ-USB FX devices divide RAM into two regions: one for code and data, and the other for USB buffers and control registers.

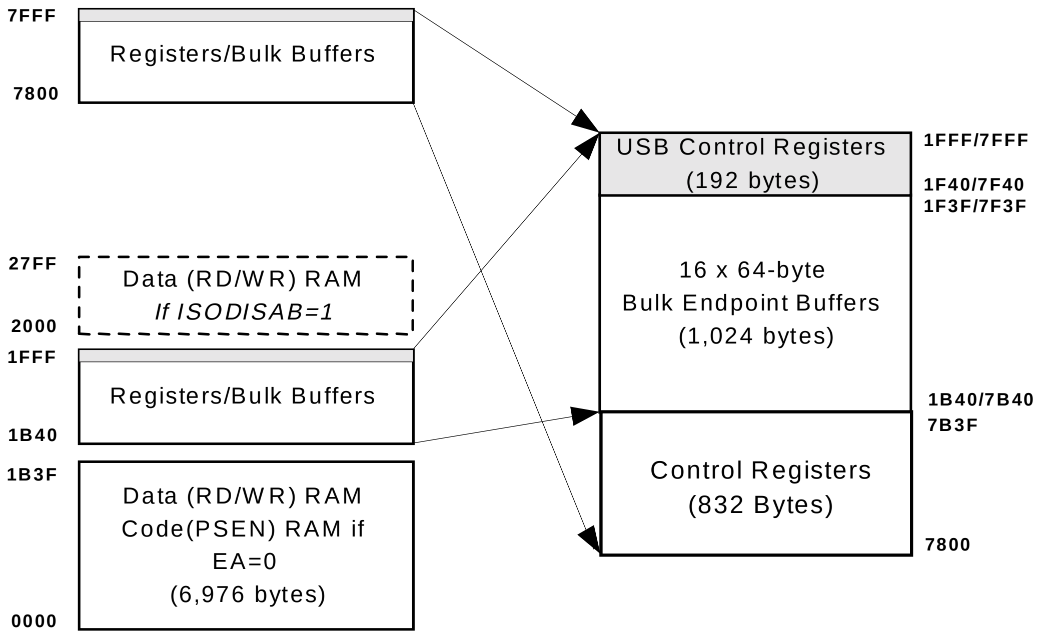


Figure 3-1. EZ-USB FX 8-KB Memory Map - Addresses are in Hexadecimal

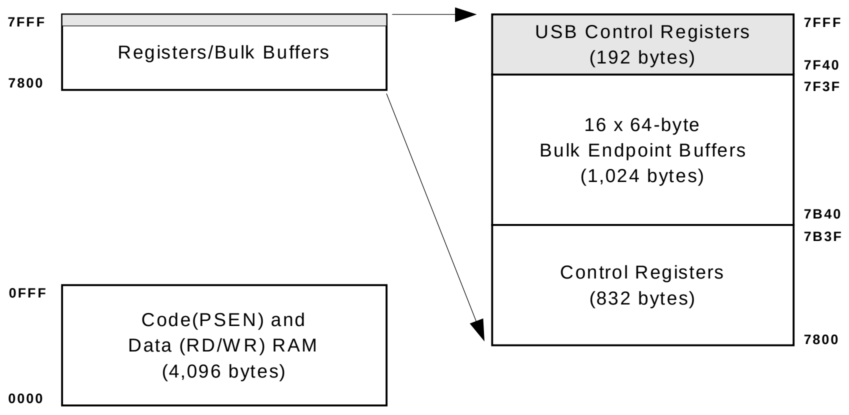


Figure 3-2. EZ-USB FX 4-KB Memory Map - Addresses are in Hexadecimal

3.2 8051 Memory

Figure 3-1 illustrates the two internal EZ-USB FX RAM regions. 6,976 bytes of general-purpose RAM occupy addresses 0x0000-0x1B3F. This RAM is loadable by the USB core or I^{2}C-compatible bus EEPROM, and contains 8051 code and data.

The EZ-USB FX EA (External Access) pin controls the placement of the bottom segment of code (PSEN) memory — inside (EA=0) or outside (EA=1) the EZ-USB FX chip. If the EA pin is tied low, the USB core internally ORs the two 8051 read signals PSEN and RD for this region, so that code and data share the 0x0000-0x1B3F memory space. If EA=1, all code (PSEN) memory is external.

3.2.1 About 8051 Memory Spaces

The 8051 partitions its memory spaces into code memory and data memory. The 8051 reads code memory using the signal PSEN# (Program Store Enable), reads data memory using the signal RD# (Data Read), and writes data memory using the signal WR# (Data Write). The 8051 MOVX (move external) instruction generates RD# or WR# strobes.

On EZ-USB FX, PSEN# is a dedicated pin, while the RD# and WR# signals share pins with two IO port signals: PC7/RD and PC6/WR. Therefore, if expanded memory is used, the port pins PC7 and PC6 are not available to the system.

1,024 bytes of RAM at 0x7B40-0x7F3F implement the sixteen bulk endpoint buffers. 192 additional bytes at 0x7F40-0x7FFF contain the USB control registers. The 8051 reads and writes this memory using the MOVX instruction. In the 8-KB RAM version of EZ-USB FX, the 1,024 bulk endpoint buffer bytes at 0x7B40-0x7F3F also appear at 0x1B40-0x1F3F. This aliasing allows unused bulk endpoint buffer memory to be added contiguously to the data memory, as illustrated Figure 3-3. The memory space at 0x1B40-0x1FFF should not be used.

Even though the 8051 can access EZ-USB FX endpoint buffers at either 0x1B40 or 0x7B40, write the firmware to access this memory only at 0x7B40-0x7FFF to maintain compatibility with future versions of EZ-USB FX that contain more than 8 KB of RAM. Future versions will have the bulk buffer space at 0x7B40-0x7F3F, only.

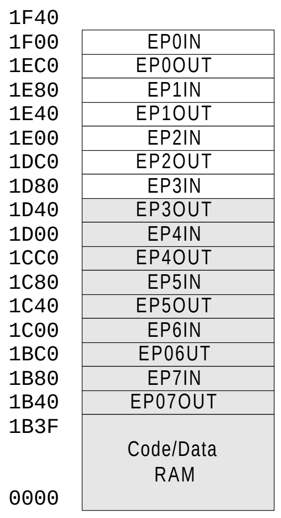


Figure 3-3. Unused Bulk Endpoint Buffers (Shaded) Used as Data Memory

In the example shown in Figure 3-3, only endpoints 0-IN through 3-IN are used for the USB function, so the data RAM (shaded) can be extended to 0x1D7F.

If an application uses *none* of the 16 EZ-USB FX isochronous endpoints, the 8051 can set the ISODISAB bit in the ISOCTL register to disable all 16 isochronous endpoints and make the 2-KB of isochronous FIFO RAM available as 8051 data RAM at 0x2000-0x27FF. **8051 code cannot run in this memory region**.

Setting ISODISAB=1 is an *all or nothing* choice, as all 16 isochronous endpoints are disabled. An application that sets this bit must never attempt to transfer data over an isochronous endpoint.

The memory map figures in the remainder of this chapter assume that ISODISAB=0, the default (and normal) case.

3.3 Expanding EZ-USB FX Memory

The 128-pin EZ-USB FX package provides a 16-bit address bus, an 8-bit bus, and memory control signals PSEN#, RD#, and WR#. These signals are used to expand EZ-USB FX memory.

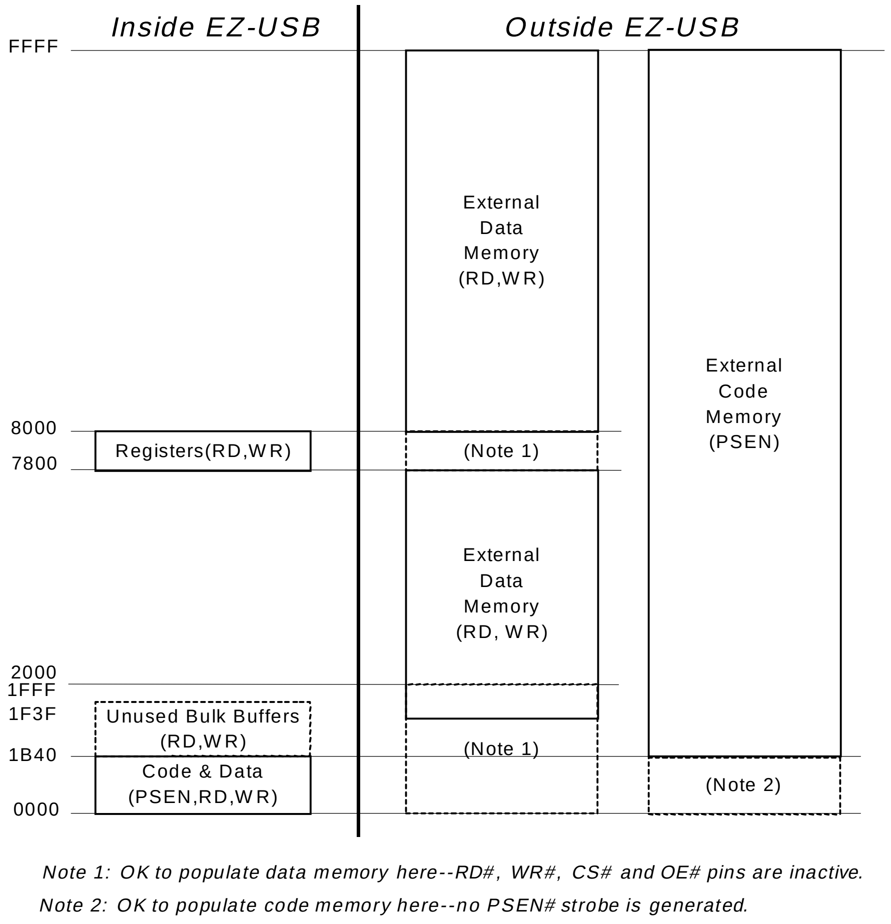

Inside EZ-USB
Outside EZ-USB
FFFF
8000
7800
2000
1FFF
1F3F
1B40
0000
Registers(RD,WR)
Unused Bulk Buffers (RD,WR)
Code & Data (PSEN,RD,WR)
External Data Memory (RD,WR)
(Note 1)
External Data Memory (RD, WR)
(Note 1)
External Code Memory (PSEN)
(Note 2)
Note 1: OK to populate data memory here--RD#, WR#, CS# and OE# pins are inactive.
Note 2: OK to populate code memory here--no PSEN# strobe is generated.

Figure 3-4. EZ-USB FX Memory Map with EA=0

Figure 3-4 shows that when EA=0, the code/data memory is internal at 0x0000-0x1B40. External code memory can be added from 0x0000-0xFFFF, but it appears in the memory map only at 0x1B40-0xFFFF. Addressing external code memory at 0x0000-0x1B3F when EA=0 causes the USB core to inhibit the #PSEN strobe. This allows program memory to be added from 0x0000-0xFFFF without requiring decoding to disable it between 0x0000 and 0x1B3F.

The internal block at 0x7B40-0x7FFF (labeled “Registers”) contains the bulk buffer memory and EZ-USB FX control registers. As previously mentioned, they are aliased at 0x1B40-0x1FFF to allow adding unused bulk buffer RAM to general-purpose memory. 8051 code should access this memory only at the 0x7B40-0x7BFF addresses. External RAM may be added from 0x0000 to 0xFFFF, but the regions shown by Note 1 in Figure 3-4 are ignored; no external strobes or select signals are generated when the 8051 executes a MOVX instruction that addresses these regions.

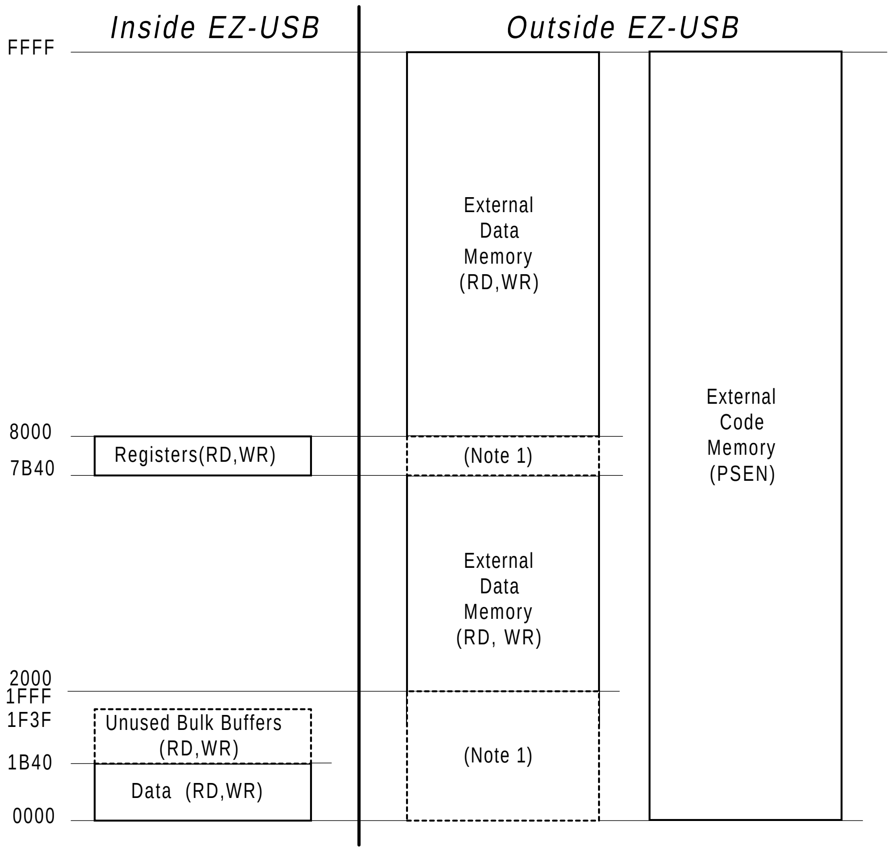
3.4 CS# and OE# Signals

The USB core gates the standard 8051 RD# and WR# signals to exclude selection of external memory that exists internal to the EZ-USB FX part. The PSEN# signal is also available on a pin for connection to external code memory.

Some 8051 systems implement external memory that is used as both data and program memory. These systems must logically OR the PSEN# and RD# signals to qualify the chip enable and output enable signals of the external memory. To save this logic, the USB core provides two additional control signals, CS# and OE#. The equations for these signals are as follows:

- CS# goes low when RD#, WR#, or PSEN# goes low.
- OE# goes low when RD# or PSEN# goes low.

Because the RD#, WR#, and PSEN# signals are already qualified by the addresses allocated to external memory, these strobes are active only when external memory is accessed.


Inside EZ-USB
Outside EZ-USB
FFFF
8000
7B40
2000
1FFF
1F3F
1B40
0000
Registers(RD,WR)
Unused Bulk Buffers (RD,WR)
Data (RD,WR)
External Data Memory (RD,WR)
(Note 1)
External Data Memory (RD, WR)
(Note 1)
External Code Memory (PSEN)

Note 1: OK to populate data memory here--RD#, WR#, CS# and OE# are inactive.

Figure 3-5. EZ-USB FX Memory Map with EA=1

When EA=1 (Figure 3-5), all code (PSEN) memory is external. All internal EZ-USB FX RAM is data memory. This gives the user over 6-KB of general-purpose RAM, accessible by the MOVX instruction.

Figure 3-4 and Figure 3-5 assume that the EZ-USB FX chip uses isochronous endpoints, and therefore that the ISODISAB bit (ISOCTL.0) is LO. If ISODISAB=1, additional data RAM appears internally at 0x2000-0x27FF, and the RD#, WR#, CS#, and OE# signals are modified to exclude this memory space from external data memory.

Chapter 4. EZ-USB FX Input/Output

4.1 Introduction

The EZ-USB FX chip provides two input-output systems:

- A set of programmable I/O pins
- A programmable I^{2}C-compatible Controller

This chapter describes the programmable I/O pins, and shows how they are shared by a variety of 8051 and EZ-USB FX alternate functions, such as UART and timer and interrupt signals. This chapter provides both the programming information for the 8051 I^{2}C-compatible interface, and the operating details of the I^{2}C-compatible boot loader. The role of the boot loader is described in *Chapter 5. "EZ-USB FX Enumeration & ReNumeration™"*.

The I^{2}C-compatible controller uses the SCL and SDA pins, and performs two functions:

- General-purpose 8051 use
- Boot loading from an EEPROM.

Pullup resistors are required on the SDA and SCL lines, even if nothing is connected to the I^{2}C-compatible bus. Each line should be pulled-up to Vcc through a 2.2K ohm resistor.

4.2 I/O Ports

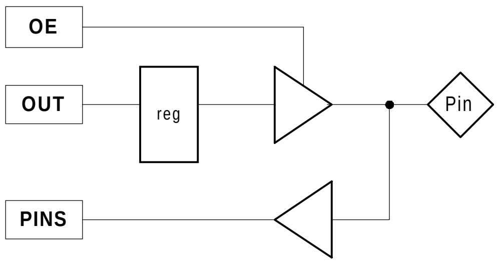


Figure 4-1. EZ-USB FX Input/Output Pin

The EZ-USB FX implements its general purpose I/O ports differently than a standard 8051. Most of the port I/O bits (PINSn and OUTn) are available in bit addressable SFR space or in XDATA space. The OEn bits are also available via SFR registers or XDATA space. See *Figure 4-6* for more information.

Figure 4-1 shows the basic structure of an EZ-USB FX I/O pin. Forty I/O pins are grouped into five 8-bit ports: PORTA, PORTB, PORTC, PORTD, and PORTE. The CY7C646x3-128NC brings out all five port pins. The CY7C646x3-80NC brings out all port pins for PORTA, PORTB, PORTC, and PORTD. The CY7C646x3-52NC brings out two PORTA pins and all pins of PORTB and PORTC. The 8051 accesses I/O pins using the three control bits shown in *Figure 4-1*: OE, OUT, and PINS. The OUT bit writes output data to a register. The OE bit turns on the output buffer. The PINS bit indicates the state of the pin. Section 4.12, *"SFR Addressing"* explains how this basic structure is enhanced to add SFR access to the I/O pins.

If you are using a small package version of EZ-USB FX, it is important to recognize that I/O ports exist inside the part that are not pinned out. Because I/O ports power-up as inputs, the 8051 code should initialize all of the unused ports as outputs to prevent floating internal nodes. Also, users of the 52-pin package should set IFCONFIG.7 (register 784A.7) to 1 to drive other internal nodes to their lowest power states.

To configure a pin as an input, the 8051 sets OE=0 to turn off the output buffer. To configure a pin as an output, the 8051 sets OE=1 to turn on the output buffer, and writes data to the OUT register. The PINS bit reflects the actual pin value, regardless of the value of OE.

A fourth control bit (in PORTACFG, PORTBCFG, PORTCCFG registers) determines whether a port pin is general-purpose Input/Output (GPIO), as shown in *Figure 4-1*, or connected to an alternate 8051 or EZ-USB FX function. Each bit of PORTA, PORTB, and PORTC has a corresponding control bit in PORTACFG, PORTBCFG, and PORTCCFG, respectively. *Figure 4-1* shows the registers and bits associated with the I/O ports shown in Table 4-1 through Table 4-4.

Depending on whether the alternate function is an input or output, the I/O logic is slightly different, as shown in *Figure 4-2* (output) and *Figure 4-3* (input).

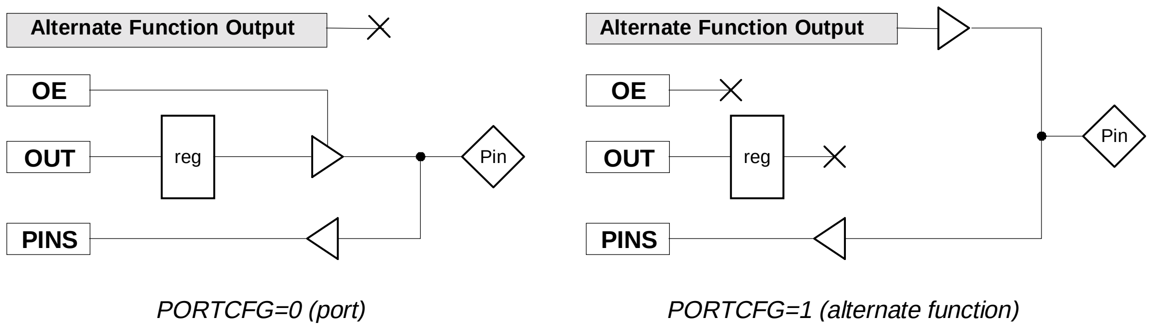

Alternate Function Output
OE
OUT
reg
PINS
Pin
PORTCFG=0 (port)
Alternate Function Output
OE
OUT
reg
PINS
Pin
PORTCFG=1 (alternate function)

Figure 4-2. Alternate Function is an OUTPUT

In *Figure 4-2*, when PORTCFG=0, the I/O port is selected. In this case the alternate function (shaded) is disconnected and the pin functions exactly as shown in
Figure 4-1. When PORTCFG=1, the alternate function is connected to the I/O pin and the output register and buffer are disconnected. Note that the 8051 can still read the state of the pin, and thus the alternate function value.

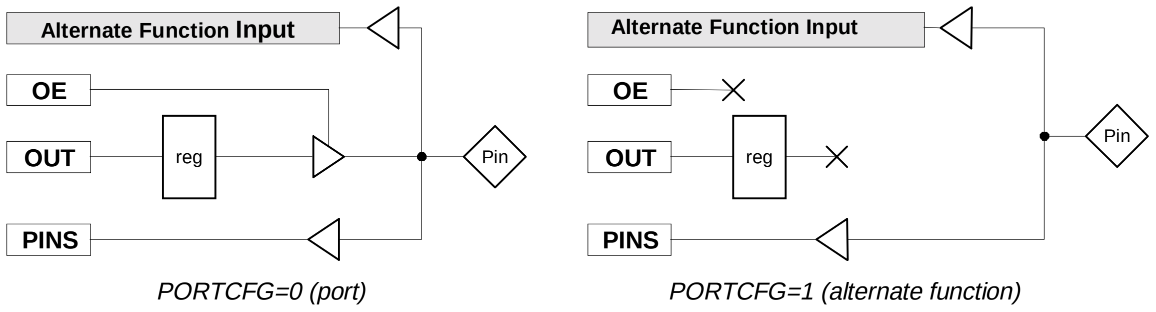


Figure 4-3. Alternate Function is an INPUT

In *Figure 4-3*, when PORTCFG=0, the I/O port is selected. This is the general I/O port shown in *Figure 4-1*, with one important difference—the alternate function is always *listening*. Whether the port pin is set for output or input, the pin signal also drives the alternate function. 8051 firmware should ensure that if the alternate function is not used (if the pin is GPIO only), the alternate input function is disabled in the 8051 Special Function Register (SFR) space.

For example, suppose the PB6/INT6 pin is configured for PB6. The pin signal is also routed to INT6. If INT6 is not used by the application, it should not be enabled. Alternatively, enabling INT6 could be useful, allowing I/O bit PB6 to trigger an interrupt.

When PORTxCFG=1, the alternate function is selected. The output register and buffer are disconnected. The PINS bit can still read the pin, and thus the input to the alternate function.

4.3 Input/Output Port Registers

The port control bits (OUT, OE, and PINS) are contained in the six registers shown in Figures *Figure 4-4* through *Figure 4-6*. Section 4.12, *"SFR Addressing"* explains how this basic structure is enhanced to add SFR access to the I/O pins.

The OUTn registers provide the data that drives the port pin when OE=1 **and** the pin is configured for port output. If the port pin is selected as an input (OE=0), the value stored in the corresponding OUTn bit is stored in an output latch but not used.

The OE registers control the output enables on the tri-state drivers connected to the port pins, unless the corresponding PORTnCFG bit is set to a “1.” When a PORTnCFG bit is set to a “1”, the value of the corresponding OE bit has no effect upon the port pin or the alternate function input.

When the corresponding PORTnCFG bit is “0” and OE=”1”, the corresponding value of OUTn is output to the pin.

When the corresponding PORTnCFG bit is “0” and OE=”0”, the corresponding value of OUTn is not output to the pin; it is tri-stated.

OUTA **Port A Outputs** **7F96**

b7	b6	b5	b4	b3	b2	b1	b0
OUTA7	**OUTA6**	**OUTA5**	**OUTA4**	**OUTA3**	**OUTA2**	**OUTA1**	**OUTA0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

OUTB **Port B Outputs** **7F97**

b7	b6	b5	b4	b3	b2	b1	b0
OUTB7	**OUTB6**	**OUTB5**	**OUTB4**	**OUTB3**	**OUTB2**	**OUTB1**	**OUTB0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

OUTC **Port C Outputs** **7F98**

b7	b6	b5	b4	b3	b2	b1	b0
OUTC7	**OUTC6**	**OUTC5**	**OUTC4**	**OUTC3**	**OUTC2**	**OUTC1**	**OUTC0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

OUTD **Port D Outputs** **7841**

b7	b6	b5	b4	b3	b2	b1	b0
OUTD7	**OUTD6**	**OUTD5**	**OUTD4**	**OUTD3**	**OUTD2**	**OUTD1**	**OUTD0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

OUTE **Port E Outputs** **7845**

b7	b6	b5	b4	b3	b2	b1	b0
OUTE7	**OUTE6**	**OUTE5**	**OUTE4**	**OUTE3**	**OUTE2**	**OUTE1**	**OUTE0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 4-4. Output Port Configuration Registers

PINSA **Port A Pins** **7F99**

b7	b6	b5	b4	b3	b2	b1	b0
PINA7	**PINA6**	**PINA5**	**PINA4**	**PINA3**	**PINA2**	**PINA1**	**PINA0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

PINSB **Port B Pins** **7F9A**

b7	b6	b5	b4	b3	b2	b1	b0
PINB7	**PINB6**	**PINB5**	**PINB4**	**PINB3**	**PINB2**	**PINB1**	**PINB0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

PINSC **Port C Pins** **7F9B**

b7	b6	b5	b4	b3	b2	b1	b0
PINC7	**PINC6**	**PINC5**	**PINC4**	**PINC3**	**PINC2**	**PINC1**	**PINC0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

PINSD **Port D Pins** **7842**

b7	b6	b5	b4	b3	b2	b1	b0
PIND7	**PIND6**	**PIND5**	**PIND4**	**PIND3**	**PIND2**	**PIND1**	**PIND0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

PINSE **Port E Pins** **7846**

b7	b6	b5	b4	b3	b2	b1	b0
PINE7	**PINE6**	**PINE5**	**PINE4**	**PINE3**	**PINE2**	**PINE1**	**PINE0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

Figure 4-5. PINSn Registers

The PINSn registers contain the current value of the port pins, whether they are selected as I/O ports or as alternate functions.

OEA **Port A Output Enable** **7F9C**

b7	b6	b5	b4	b3	b2	b1	b0
OEA7	**OEA6**	**OEA5**	**OEA4**	**OEA3**	**OEA2**	**OEA1**	**OEA0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

OEB **Port B Output Enable** **7F9D**

b7	b6	b5	b4	b3	b2	b1	b0
OEB7	**OEB6**	**OEB5**	**OEB4**	**OEB3**	**OEB2**	**OEB1**	**OEB0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

OEC **Port C Output Enable** **7F9E**

b7	b6	b5	b4	b3	b2	b1	b0
OEC7	**OEC6**	**OEC5**	**OEC4**	**OEC3**	**OEC2**	**OEC1**	**OEC0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

OED **Port D Output Enable** **7843**

b7	b6	b5	b4	b3	b2	b1	b0
OED7	**OED6**	**OED5**	**OED4**	**OED3**	**OED2**	**OED1**	**OED0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

OEE **Port E Output Enable** **7847**

b7	b6	b5	b4	b3	b2	b1	b0
OEE7	**OEE6**	**OEE5**	**OEE4**	**OEE3**	**OEE2**	**OEE1**	**OEE0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 4-6. Output Enable Registers

EZ-USB FX ports A, B, and C have individually selectable, alternate functions for each port pin. Alternate functions, such as UART TxD and RxD, are selected on a pin-by-pin basis for these ports using control bits in registers PORTACFG, PORTBCFG, and PORTCCFG.

Although ports D and E can be used for purposes other than I/O pins, they do not have corresponding, alternate function configuration registers like ports A-C (see Section 15.23, *"PORTA and PORTC Alternate Configurations"*). Instead, their alternate functions are selected in 8-bit groups using a single-interface configuration register called IFCONFIG (see Section 15.22, *"Interface Configuration"*). Two bits, IF[1..0], select four configurations for ports D and E.

4.4 Port Configuration Tables

Table 4-1. Port A Configuration

PORTA Bit 0		
IFCONFIG.3=0		IFCONFIG.3=1
PORT-ACFG.0=0	PORT-ACFG.0=1	
Port pin PA0	**T0out**	**GSTATE[0]**
I/O	O	O

PORTA Bit 1		
IFCONFIG.3=0		IFCONFIG.3=1
PORT-ACFG.1=0	PORT-ACFG.1=1	
Port pin PA1	**T1out**	**GSTATE[1]**
I/O	O	O

PORTA Bit 2		
IFCONFIG.3=0		IFCONFIG.3=1
PORT-ACFG.2=0	PORT-ACFG.2=1	
Port pin PA2	**OE#**	**GSTATE[2]**
I/O	O	O

PORTA Bit 3	
PORT-ACFG.3=0	PORTACFG.3=1
Port pin PA3	**CS#**
I/O	O

Table 4-1. Port A Configuration

PORTA Bit 4			
PORT-ACFG.4=0	PORTACFG.4=1		
	PORTACF2.4=0	PORTACF2.4=1	
		IFCON-FIG[1..0]=10	IFCON-FIG[1..0]=11
Port pin PA4	**FWR#**	**RDY4**	**SLWR**
I/O	O	I	I

PORTA Bit 5			
PORT-ACFG.5=0	PORTACFG.5=1		
	PORTACF2.5=0	PORTACF2.5=1	
		IFCON-FIG[1..0]=10	IFCON-FIG[1..0]=11
Port pin PA5	**FRD#**	**RDY5**	**SLRD**
I/O	O	I	I

PORTA Bit 6	
PORT-ACFG.6=0	PORTACFG.6=1
Port pin PA6	**RxD0out**
I/O	O

PORTA Bit 7	
PORT-ACFG.7=0	PORTACFG.7=1
Port pin PA7	**RxD1out**
I/O	O

Table 4-2. Port B Configuration

PORTB Bit 0				
IFCONFIG[1..0]=00		IFCON-FIG[1..0]=01	IFCON-FIG[1..0]=10	IFCON-FIG[1..0]=11
PORTB-CFG.0=0	PORTB-CFG.0=1			
Port pin PB0	**T2**	**D[0]**	**GDA[0]**	**AFI[0]**
I/O	I	I/O	I/O	I/O

PORTB Bit 1				
IFCONFIG[1..0]=00		IFCON-FIG[1..0]=01	IFCON-FIG[1..0]=10	IFCON-FIG[1..0]=11
PORTB-CFG.1=0	PORTB-CFG.1=1			
Port pin PB1	**T2EX**	**D[1]**	**GDA[1]**	**AFI[1]**
I/O	I	I/O	I/O	I/O

Table 4-2. Port B Configuration

PORTB Bit 2				
IFCONFIG[1..0]=00		IFCON-FIG[1..0]=01	IFCON-FIG[1..0]=10	IFCON-FIG[1..0]=11
PORTB-CFG.2=0	PORTB-CFG.2=1			
Port pin PB2	**RxD1**	**D[2]**	**GDA[2]**	**AFI[2]**
I/O	I	I/O	I/O	I/O

PORTB Bit 3				
IFCONFIG[1..0]=00		IFCON-FIG[1..0]=01	IFCON-FIG[1..0]=10	IFCON-FIG[1..0]=11
PORTB-CFG.3=0	PORTB-CFG.3=1			
Port pin PB3	**TxD1**	**D[3]**	**GDA[3]**	**AFI[3]**
I/O	O	I/O	I/O	I/O

PORTB Bit 4				
IFCONFIG[1..0]=00		IFCON-FIG[1..0]=01	IFCON-FIG[1..0]=10	IFCON-FIG[1..0]=11
PORTB-CFG.4=0	PORTB-CFG.4=1			
Port pin PB4	**INT4**	**D[4]**	**GDA[4]**	**AFI[4]**
I/O	I	I/O	I/O	I/O

PORTB Bit 5				
IFCONFIG[1..0]=00		IFCON-FIG[1..0]=01	IFCON-FIG[1..0]=10	IFCON-FIG[1..0]=11
PORTB-CFG.5=0	PORTB-CFG.5=1			
Port pin PB5	**INT5#**	**D[5]**	**GDA[5]**	**AFI[5]**
I/O	I	I/O	I/O	I/O

PORTB Bit 6				
IFCONFIG[1..0]=00		IFCON-FIG[1..0]=01	IFCON-FIG[1..0]=10	IFCON-FIG[1..0]=11
PORTB-CFG.6=0	PORTB-CFG.6=1			
Port pin PB6	**INT6**	**D[6]**	**GDA[6]**	**AFI[6]**
I/O	I	I/O	I/O	I/O

PORTB Bit 7				
IFCONFIG[1..0]=00		IFCON-FIG[1..0]=01	IFCON-FIG[1..0]=10	IFCON-FIG[1..0]=11
PORTB-CFG.7=0	PORTB-CFG.7=1			
Port pin PB7	**T2OUT**	**D[7]**	**GDA[7]**	**AFI[7]**
I/O	O	I/O	I/O	I/O

Table 4-3. Port C Configuration

PORTC Bit 0			
PORTC-CFG.0=0	PORTCCFG.0=1		
	PORTCCF2.0=0	PORTCCF2.0=1	
		IFCON-FIG[1..0]=10	00, 01, 11 not valid
Port pin PC0	**RxD0**	**RDY0**	**X**
I/O	I	I	

PORTC Bit 1			
PORTC-CFG.1=0	PORTCCFG.1=1		
	PORTCCF2.1=0	PORTCCF2.1=1	
		IFCON-FIG[1..0]=10	00, 01, 11 not valid
Port pin PC1	**TxD0**	**RDY1**	**X**
I/O	O	I	

PORTC Bit 2	
PORTC-CFG.2=0	PORTCCFG.2=1
Port pin PC3	**INT0#**
I/O	I

PORTC Bit 3			
PORTC-CFG.3=0	PORTCCFG.3=1		
	PORTCCF2.3=0	PORTCCF2.3=1	
		IFCON-FIG[1..0]=10	00, 01, 11 not valid
Port pin PC3	**INT1#**	**RDY3**	**X**
I/O	I	I	

PORTC Bit 4			
PORTC-CFG.4=0	PORTCCFG.4=1		
	PORTCCF2.4=0	PORTCCF2.4=1	
		IFCON-FIG[1..0]=10	00, 01, 11 not valid
Port pin PC4	**T0**	**CTL1**	**X**
I/O	I	O	

Table 4-3. Port C Configuration

PORTC Bit 5			
PORTC-CFG.5=0	PORTCCFG.5=1		
	PORTCCF2.5=0	PORTCCF2.5=1	
		IFCON-FIG[1..0]=10	00, 01, 11 not valid
Port pin PC5	**T1**	**CTL3**	**X**
I/O	I	O	

PORTC Bit 6			
PORTC-CFG.6=0	PORTCCFG.6=1		
	PORTCCF2.6=0	PORTCCF2.6=1	
		IFCON-FIG[1..0]=10	00, 01, 11 not valid
Port pin PC6	**WR#**	**CTL4**	**X**
I/O	O	O	

PORTC Bit 7			
PORTC-CFG.7=0	PORTCCFG.7=1		
	PORTCCF2.7=0	PORTCCF2.7=1	
		IFCON-FIG[1..0]=10	00, 01, 11 not valid
Port pin PC7	**RD#**	**CTL5**	**X**
I/O	O	O	

Table 4-4. Port D Bits

PORTD Bits [7..0]					
IFCON-FIG[1..0]=00	IFCON-FIG[1..0]=01	IFCONFIG[1..0]=10		IFCONFIG[1..0]=11	
		IFCONFIG.2=0	IFCONFIG.2=1	IFCONFIG.2=0	IFCONFIG.2=1
Port pins PD[7..0]	**Port pins PD[7..0]**	**Port pins PD[7..0]**	**GDB[7..0]**	**Port pins PD[7..0]**	**BFI[7..0]**
I/O	I/O	I/O	I/O	I/O	I/O

Table 4-5. Port E Bits

Port E Bit 0		
IFCONFIG[1..0]=00, 01	IFCONFIG[1..0]=10	IFCONFIG[1..0]=11
Port pin PE[0]	**adr0**	**BOUTFLAG**
I/O	O	O

Port E Bit 1		
IFCONFIG[1..0]=00, 01	IFCONFIG[1..0]=10	IFCONFIG[1..0]=11
Port pin PE[1]	**adr1**	**AINFULL**
I/O	O	O

Port E Bit 2		
IFCONFIG[1..0]=00, 01	IFCONFIG[1..0]=10	IFCONFIG[1..0]=11
Port pin PE[2]	**adr2**	**BINFULL**
I/O	O	O

Port E Bit 3		
IFCONFIG[1..0]=00, 01	IFCONFIG[1..0]=10	IFCONFIG[1..0]=11
Port pin PE[3]	**adr3**	**AOUTEMTY**
I/O	O	O

Port E Bit 4		
IFCONFIG[1..0]=00, 01	IFCONFIG[1..0]=10	IFCONFIG[1..0]=11
Port pin PE[4]	**adr4**	**BOUTEMTY**
I/O	O	O

Port E Bit 5		
IFCONFIG[1..0]=00, 01	IFCONFIG[1..0]=10	IFCONFIG[1..0]=11
Port pin PE[5]	**CTL3**	**Port pin PE[5]**
I/O	O	I/O

Port E Bit 6		
IFCONFIG[1..0]=00, 01	IFCONFIG[1..0]=10	IFCONFIG[1..0]=11
Port pin PE[6]	**CTL4**	**Port pin PE[6]**
I/O	O	I/O

Port E Bit 7		
IFCONFIG[1..0]=00, 01	IFCONFIG[1..0]=10	IFCONFIG[1..0]=11
Port pin PE[7]	**CTL5**	**Port pin PE[7]**
I/O	O	I/O

4.5 I²C-Compatible Controller

The USB core contains an I^{2}C-compatible controller for boot loading and general-purpose I^{2}C-compatible bus interface. This controller uses the SCL (Serial Clock) and SDA (Serial Data) pins. I2C-compatible controller describes how the boot load operates at power-on to read the contents of an external serial EEPROM to determine the initial EZ-USB FX configuration. The boot loader operates automatically, while the 8051 is held in reset. The last section of this chapter describes the operating details of the boot loader.

After the boot sequence completes and the 8051 is brought out of reset, the general-purpose I^{2}C-compatible controller is available to the 8051 for interface to external I^{2}C-compatible devices, such as other EEPROMS, I/O chips, audio/video control chips, etc.

For I^{2}C-compatible peripherals that support it, the EZ-USB FX I^{2}C-compatible bus can run at 400 KHz. For compatibility, the EZ-USB FX powers-up at the 100-KHz frequency.

4.6 8051 I²C-Compatible Controller

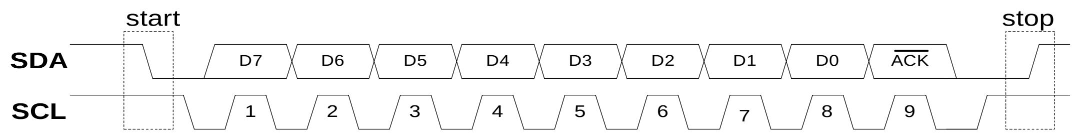


Figure 4-7. General I^{2}C-Compatible Transfer

Figure 4-7 illustrates the waveforms for an I^{2}C-compatible transfer. SCL and SDA are open-drain EZ-USB FX pins, which must be pulled up to Vcc with external resistors. The EZ-USB FX chip is an I^{2}C-compatible bus master only, meaning that it synchronizes data transfers by generating clock pulses on SCL by driving low. Once the master drives SCL low, external slave devices can also drive SCL low to extend clock cycle times.

To synchronize I^{2}C-compatible data, serial data (SDA) is permitted to change state only while SCL is low, and must be valid while SCL is high. Two exceptions to this rule are used to generate START and STOP conditions. A START condition is defined as SDA going low, while SCL is high, and a STOP condition is defined as SDA going high, while SCL is high. Data is sent MSB first. During the last bit time (clock #9 in *Figure 4-7*), the master (EZ-USB FX) floats the SDA line to allow the slave to acknowledge the transfer by pulling SDA low.

***Multiple I^{2}C-Compatible Bus Masters** — The EZ-USB FX chip acts only as an I^{2}C-compatible bus master, never a slave. However, the 8051 can detect a second master by checking for BERR=1 (Section 4.8, "Status Bits").*

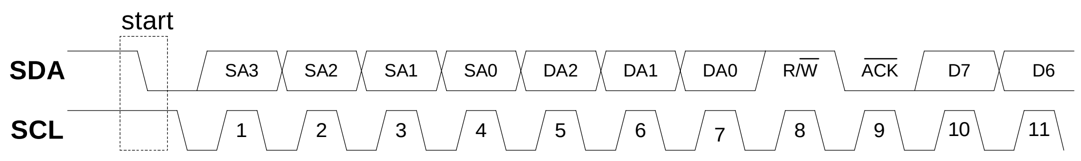


Figure 4-8. Addressing an I^{2}C-compatible Peripheral

The first byte of an I^{2}C-compatible bus transaction contains the address of the desired peripheral. *Figure 4-8* shows the format for this first byte, which is sometimes called a *control* byte.

A master sends the bit sequence shown in *Figure 4-8* after sending a START condition. The master uses this 9-bit sequence to select an I^{2}C-compatible peripheral at a particular address, to establish the transfer direction (using R/W#), and to determine if the peripheral is present by testing for ACK#.

The four most significant bits SA3-SA0 are the peripheral chip's slave address. I^{2}C-compatible devices are pre-assigned slave addresses by device type. For example, slave address 1010 is assigned to EEPROMS. The three bits DA2-DA0 usually reflect the states of I^{2}C-compatible device address pins. Devices with three address pins can be strapped to allow eight distinct addresses for the same device type. The eighth bit (R/W#) sets the direction for the ensuing data transfer, 1 for master read, and 0 for master write. Most address transfers are followed by one or more data transfers, with the STOP condition generated after the last data byte is transferred.

In *Figure 4-8*, a READ transfer follows the address byte (at clock 8, the master sets the R/W# bit high, indicating READ). At clock 9, the peripheral device responds to its address by asserting ACK. At clock 10, the master floats SDA and issues SCL pulses to clock in SDA data supplied by this slave.

Assuming the 12-MHz crystal used by the EZ-USB FX family, the SCL frequency is 90.9 KHz, giving an I^{2}C-compatible transfer rate of 11 microseconds per bit. Operation at four times this rate is available by setting a bit in the boot EEPROM. See Section 5.9, *"Configuration Byte 0"* for details.

I2CS **I^2C-Compatible Control and Status** **7FA5**

b7	b6	b5	b4	b3	b2	b1	b0
START	**STOP**	**LASTRD**	**ID1**	**ID0**	**BERR**	**ACK**	**DONE**
R/W	R/W	R/W	R	R	R	R	R
0	0	0	x	x	0	0	0

I2DAT **I^2C-Compatible Data** **7FA6**

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 4-9. I^2C-compatible Registers

The 8051 uses the two registers shown in *Figure 4-9* to conduct I^2C-compatible transfers. The 8051 transfers data to and from the I^2C-compatible bus by writing and reading the I2DAT register. The I2CS register controls I^2C-compatible transfers and reports various status conditions. The three control bits are START, STOP, and LASTRD. The remaining bits are status bits. Writing to a status bit has no effect.

4.7 Control Bits

4.7.1 START

The 8051 sets the START bit to 1 to prepare an I^2C-compatible bus transfer. If START=1, the next 8051 load to I2DAT generates the start condition followed by the serialized byte of data in I2DAT. The 8051 loads data in the format shown in *Figure 4-7* after setting the START bit. The I^2C-compatible controller clears the START bit during the ACK interval (clock 9 in *Figure 4-7*).

4.7.2 STOP

The 8051 sets STOP=1 to terminate an I^2C-compatible bus transfer. The I^2C-compatible controller clears the STOP bit after completing the STOP condition. If the 8051 sets the STOP bit during a byte transfer, the STOP condition generates immediately following the ACK phase of the byte transfer. If no byte transfer is occurring when the STOP bit is set, the STOP condition is carried out immediately on the bus. Data should not be written to I2CS or I2DAT until the STOP bit returns low. In most versions of CY7C646x3-128NC, an interrupt request is available to signal that STOP bit transmission is complete.

4.7.3 LASTRD

To read data over the I^{2}C-compatible bus, an I^{2}C-compatible master floats the SDA line and issues clock pulses on the SCL line. After every eight bits, the master drives SDA low for one clock to indicate ACK. To signal the last byte of the read transfer, the master floats SDA at ACK time to instruct the slave to stop sending. This is controlled by the 8051 by setting LASTRD=1 before reading the last byte of a read transfer. The I^{2}C-compatible controller clears the LASTRD bit at the end of the transfer (at ACK time).

Setting LASTRD does not automatically generate a STOP condition. The 8051 should also set the STOP bit at the end of a read transfer.

4.8 Status Bits

After a byte transfer, the I^{2}C-compatible controller updates the three status bits BERR, ACK, and DONE. If no STOP condition was transmitted, they are updated at ACK time. If a STOP condition was transmitted they are updated after the STOP condition is transmitted.

4.8.1 DONE

The I^{2}C-compatible controller sets this bit whenever it completes a byte transfer, right after the ACK stage. The controller also generates an I^{2}C-compatible interrupt request (8051 INT3) when it sets the DONE bit. The I^{2}C-compatible controller clears the DONE bit when the 8051 reads or writes the I2DAT register, and it clears the I^{2}C-compatible interrupt request bit whenever the 8051 reads or writes the I2CS or I2DAT register.

4.8.2 ACK

Every ninth SCL of a write transfer, the slave indicates reception of the byte by asserting ACK. The I^{2}C-compatible controller floats SDA during this time, samples the SDA line, and updates the ACK bit with the complement of the detected value. ACK=1 indicates acknowledge, and ACK=0 indicates not-acknowledge. The I^{2}C-compatible controller updates the ACK bit at the same time it sets DONE=1. The ACK bit should be ignored for read transfers on the bus.

4.8.3 BERR

This bit indicates an I^{2}C-compatible bus error. BERR=1 indicates that there was bus contention, which results when an outside device drives the bus LO when it shouldn't, or when another bus master wins arbitration, taking control of the bus. BERR is cleared when the 8051 reads or writes the I2DAT register.

4.8.4 ID1, ID0

These bits are set by the boot loader (Section 4.11, *"I2C-Compatible Boot Loader"*) to indicate whether an 8-bit address or 16-bit address EEPROM at slave address 000 or 001 was detected at power-on. They are normally used only for debug purposes. Table 4-7 shows the encoding for these bits.

4.9 Sending I²C-Compatible Data

To send a multiple byte data record over the I^{2}C-compatible bus, follow these steps:

1. Set the START bit.
2. Write the peripheral address and direction=0 (for write) to I2DAT.
3. Wait for DONE=1*. If BERR=1 or ACK=0, go to step 7.
4. Load I2DAT with a data byte.
5. Wait for DONE=1*. If BERR=1 or ACK=0 go to step 7.
6. Repeat steps 4 and 5 for each byte until all bytes have been transferred.
7. Set STOP=1.

* If the I^{2}C-compatible interrupt (8051 INT3) is enabled, each "Wait for DONE=1" step can be interrupt driven, and handled by an interrupt service routine. See *Chapter 12. "EZ-USB FX Interrupts"* for more details regarding the I^{2}C-compatible interrupt.

4.10 Receiving I²C-Compatible Data

To read a multiple-byte data record, follow these steps:

1. Set the START bit.
2. Write the peripheral address and direction=1 (for read) to I2DAT.
3. Wait for DONE=1*. If BERR=1 or ACK=0, terminate by setting STOP=1.
4. Read I2DAT and discard the data. This initiates the first burst of nine SCL pulses to clock in the first byte from the slave.
5. Wait for DONE=1*. If BERR=1, terminate by setting STOP=1.
6. Read the data from I2DAT. This initiates another read transfer.
7. Repeat steps 5 and 6 for each byte until ready to read the second-to-last byte.
8. Before reading the second-to-last I2DAT byte, set LASTRD=1.

9. Read the data from I2DAT. With LASTRD=1, this initiates the final byte read on the I^{2}C-compatible bus.
10. Wait for DONE=1*. If BERR=1, terminate by setting STOP=1.
11. Set STOP=1.
12. Read the last byte from I2DAT immediately (the next instruction) after setting the STOP bit. This retrieves the last data byte without initiating an extra read transaction (nine more SCL pulses) on the I^{2}C-compatible bus.

* If the I^{2}C-compatible interrupt (8051 INT3) is enabled, each “Wait for DONE=1” step can be interrupt-driven, and handled by an interrupt service routing. See *Chapter 12. "EZ-USB FX Interrupts"* for more details regarding the I^{2}C-compatible interrupt.

4.11 I^{2}C-Compatible Boot Loader

When the EZ-USB FX chip comes out of reset, the EZ-USB FX boot loader checks for the presence of an EEPROM on its I^{2}C-compatible bus. If an EEPROM is detected, the loader reads the first EEPROM byte to determine how to enumerate (specifically, whether to supply ID information from the USB core or from the EEPROM). The various enumeration modes are described in *Chapter 5. "EZ-USB FX Enumeration & ReNumeration™"*.

Prior to reading the first EEPROM byte, the boot loader must set to zero an address counter inside the EEPROM. It does this by sending a control byte (write) to select the EEPROM, followed by a zero address to set the internal EEPROM address pointer to zero. Then, it issues a control byte (read), and reads the first EEPROM byte.

The EZ-USB FX boot loader supports two I^{2}C-compatible EEPROM types:

- EEPROMs with address A[7..4]=1010 that use an 8-bit address, (example: 24LC00, 24LC01/B, 24LC02/B).
- EEPROMs with address A[7..4]=1010 that use a 16-bit address, (example: 24AA64, 24LC128, 24AA256).

EEPROMs with densities up to 256 bytes require loading a single address byte. Larger EEPROMs require loading two address bytes.

The EZ-USB FX I^{2}C-compatible controller needs to determine which EEPROM type is connected—one or two address bytes—so that it can properly reset the EEPROM address pointer to zero before reading the EEPROM. For the single-byte address part, it must send a single zero byte of address, and for the two-byte address part it must send two zero bytes of address.

Because there is no direct way to detect which EEPROM type—single or double address—is connected, the I^{2}C-compatible controller uses the EEPROM address pins A2, A1, and A0 to determine whether to send out one or two bytes of address. This algorithm requires that the EEPROM address lines are strapped as shown in Table 4-6. Single-byte-address EEPROMs are strapped to address 000 and double-byte-address EEPROMs are strapped to address 001.

Table 4-6. Strap Boot EEPROM Address Lines to These Values

Bytes	Example EEPROM	A2	A1	A0
16	24LC00*	N/A	N/A	N/A
128	24LC01	0	0	0
256	24LC02	0	0	0
4K	24LC32	0	0	1
8K	24LC64	0	0	1

* This EEPROM does not have address pins

The I^2C-compatible controller performs a three-step test at power-on to determine whether a one-byte-address or a two-byte-address EEPROM is attached. This test proceeds as follows:

1. The I^2C-compatible controller sends out a “read current address” command to I^2C-compatible sub-address 000 (10100001). If no ACK is returned, the controller proceeds to step 2. If ACK is returned, the one-byte-address device is indicated. The controller discards the data and proceeds to step 3.
2. The I^2C-compatible controller sends out a “read current address” command to I^2C-compatible sub-address 001 (10100011). If ACK is returned, the two-byte-address device is indicated. The controller discards the data and proceeds to step 3. If no ACK is returned, the controller assumes that a valid EEPROM is not connected, assumes the “No Serial EEPROM” mode, and terminates the boot load.
3. The I^2C-compatible controller resets the EEPROM address pointer to zero (using the appropriate number of address bytes), then reads the first EEPROM byte. If it does not read 0xB4 or 0xB6, the controller assumes the “No Serial EEPROM” mode. If it reads either 0xB4 or 0xB6, the controller copies the next six bytes into internal storage. If it reads 0xB6, it proceeds to load the EEPROM contents into internal RAM.

The results of this power-on test are reported in the ID1 and ID0 bits, as shown in Table 4-7.

Table 4-7. Results of Power-On I^2C-Compatible Test

ID1	ID0	Meaning
0	0	No EEPROM detected
0	1	One-byte-address load EEPROM detected
1	0	Two-byte-address load EEPROM detected
1	1	Not used

Other EEPROM devices (with device address of 1010) can be attached to the I^2C-compatible bus for general purpose 8051 use, as long as they are strapped for address other than 000 or 001. If a 24LC00 EEPROM is used, no other EEPROMS with device address 1010 may be used because the 24LC00 responds to all eight sub-addresses.

4.12 SFR Addressing

The 8051 architecture includes a directly-addressable bank of registers from 0x80-0xFF, called Special Function Registers or SFRs. These registers control various 8051 peripheral functions such as the timers, interrupts, and UARTs. Because they are directly addressable, they allow quick transfer of bytes in and out of the 8051 accumulator.

A portion of the 8051 SFR space is bit-addressable. The 8051 architecture assigns 256 bit addresses to individual bits in certain registers, including SFR registers with addresses ending in 0 or 8. The advantage of bit addressing is that special bit manipulation instructions can set, test, or toggle individual bits without dealing with bytes—reading a byte, modifying one bit, or writing back the byte. This bit manipulation is especially useful for I/O, when a single I/O pin needs attention.

The EZ-USB FX preserves the I/O architecture used in EZ-USB Series 2100, where I/O is controlled using memory mapped registers in external RAM space. To allow quick access to the I/O control registers, EZ-USB FX also maps the I/O control registers into SFR registers. In addition, four of the I/O control registers are bit-addressable.

Table 4-8. EZ-USB FX Special Function Registers*

	80	**90**	**A0**	**B0**	**C0**	**D0**	**E0**	**F0**
0	**IOA**	**IOB**	**IOC**	**IOD**	**SCON1**	PSW	ACC	B
1	SP	**EXIF**	**INT2CLR**	**IOE**	**SBUF1**			
2	DPL0	**MPAGE**	**INT4CLR**	**SOEA**				
3	DPH0			**SOEB**				
4	DPL1			**SOEC**				
5	DPH1			**SOED**				
6	DPS			**SOEE**				
7	PCON							
8	TCON	SCON0	IE	IP	T2CON	**EICON**	**EIE**	**EIP**
9	TMOD	SBUF0						
A	TL0				RCAP2L			
B	TL1				RCAP2H			
C	TH0				TL2			
D	TH1				TH2			
E	**CKCON**							
F								

* 8051 enhancements appear in bold. EZ-USB FX SFRs are shaded. Bit-addressable registers (rows 0 and 8) are highlighted.

In the standard 8051, ports 0-3 are addressed using SFRs 80, 90, A0, and B0. Because these ports are not implemented in EZ-USB FX, the SFRs are available. The EZ-USB FX chip maps the input-output data for four of its I/O ports, A-D, into these registers. Also, the I/O register for PORT E and the port output enable registers are mapped into non-bit-addressable SFRs as shown in Table 4-8.

INT2CLR and **INT4CLR** are dummy registers (no data) that provide a fast method for clearing IRQ2 and IRQ4 flags. The 8051 writes any value to these registers to clear the IRQ2 or IRQ4 interrupt request flags.

INT2 is used for all USB interrupts. INT4 is used for all slave FIFO and GPIF interrupts.

Two enable bits turn on the SFR interrupt clearing:

- INT2 SFR clearing is enabled by setting USBBAV.4=1.
- INT4 SFR clearing is enabled by setting INT4SETUP.2=1.

The two code examples (*Figure 4-10* and *Figure 4-11*) illustrate the speed advantage gained by using the INT2CLR SFR to clear a pending USB interrupt request for endpoint 6 OUT. The first example uses the EZ-USB FX method, and the second example uses the new SFR method to clear an interrupt request for bulk endpoint EP6OUT.

```
EP6OUT_ISR_A:
              push   dps
              push   dpl
              push   dph
              push   dpl1
              push   dph1
              push   acc
;
              mov    a,EXIF            ; clear INT2 (USB) IRQ flag
              clr    acc.4
              mov    EXIF,a
;
              mov    dptr,#OUT07IRQ
              mov    a,#01000000b      ; clear OUT6 IRQ bit by writing 1
              movx   @dptr,a

; Do interrupt processing here —set flags, whatever...
;
              pop    acc
              pop    dph1
              pop    dpl1
              pop    dph
              pop    dpl
              pop    dps
              reti
```

Figure 4-10. EZ-USB FX Method, sample code

Because the OUT6 interrupt request bit is in the memory-mapped register OUT07IRQ, the 8051 clears it using the data pointer and a MOVX instruction. Because this is an interrupt service routine, all registers used by the ISR must be saved and restored. It is not known at the time of the interrupt which data pointer is in use, so both of them along with the data pointer select register “dps” are pushed and later restored (popped).

Next, the INT2 request bit is cleared in EXIF.4. It is important to clear INT2 before clearing the individual source of the interrupt—in this example EP6OUT. (This is explained in *Chapter 12. "EZ-USB FX Interrupts"*). Finally, the data pointer is set to OUT07IRQ, and the bit corresponding to OUT6 is set, and written to OUT0IRQ. Writing a “1” clears the OUT6 interrupt request.

```
init:          movx   dptr,#USBBAV
               movx   a,@dptr
               setb   acc.4              ; enable the SFR-clearing feature
               movx   @dptr,a            ; for INT2
;
EP6OUT_ISR_B:
               push   acc
               mov    a,EXIF             ; clear INT2 (USB) IRQ flag
               clr    acc.4
               mov    EXIF,a
;
               mov    INT2CLR,a          ; use whatever value is in acc
;
; Do interrupt processing here
;
               pop    acc
               reti
```

Figure 4-11. SFR Method, sample code

The “init” routine should be included in general initialization code, and is executed only once. Setting bit 4 of USBBAV enables the SFR clearing feature for INT2 (but not INT4).

The ISR clears the INT2 request bit in EXIF.4, as before. But now, only one instruction is required to clear the endpoint 6-OUT IRQ, due to the fact that the SFR is directly addressable.

There are two important points about this operation:

The data in acc is *don’t care*, because the act of writing INT2CLR, and not the data written, actually clears the IRQ. Second, the particular USB interrupt cleared by this instruction is the one currently pending (the interrupt source is displayed in the INT2IVEC register).

4.13 SFR Control of PORTs A-E

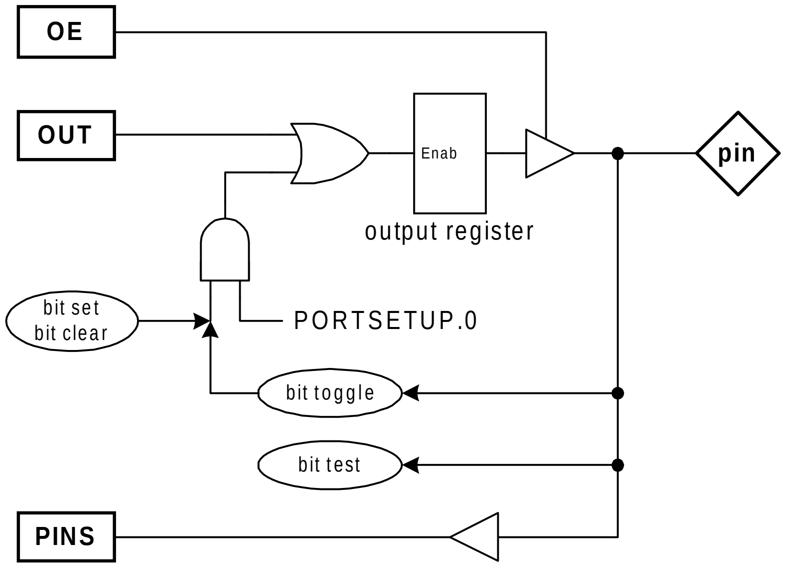


Figure 4-12. EZ-USB FX I/O Structure

Figure 4-12 shows a block diagram of the EZ-USB FX I/O structure. The signals in rectangles, OE, OUT, and PINS, represent the memory mapped register bits that access I/O bits using 8051 MOVX instructions. The ovals represent access via the SFRs. The 8051 sets a single bit, PORTSETUP.0, to enable SFR access to all of the I/O pins.

When PORTSETUP.0=1, both I/O access methods operate simultaneously. Both the MOVX method and SFR addressing method can be used to set the state of an output pin. To elaborate, the following code example sets PA0 using a MOVX instruction, clears it using a bit clear instruction, and then toggles it using a bit toggle instruction.

```
              mov     dptr,#OUTA        ; set PA0 the old way
              movx    a,@dptr           ; get value of OUTA register
              setb    acc.0             ; set bit 0
              movx    @dptr,a           ; write it back
;
              clrb    IOA.0             ; clear PA0 bit the new way
;
              cpl     IOA.0             ; complement PA0 bit the new way
```

Figure 4-13. Use MOVX to Set PA0, sample code

This simple example illustrates two important points. First, both the old (MOVX) and new (SFR) methods can be used on the same I/O bits. Second, the SFR method is much more efficient, because setting the bit using the MOVX takes nine cycles and seven bytes, while a bit set, clear, or toggle instruction takes two cycles and two bytes. In practice, there is no reason to use the first method in EZ-USB FX except for backward compatibility; the example is meant to illustrate that each method can be used independently.

The data registers for I/O ports A, B, C, and D are mapped into SFRs that are bit-addressable (0x80, 0x90, 0xA0, and 0xB0, respectively). Because the 8051 uses the rest of the SFRs, the remaining EZ-USB FX I/O registers (PORTE data and the output enables) are mapped into SFRs that are not bit-addressable. This still gives faster access to these I/O bits because direct addressing takes less time and fewer bytes than MOVX addressing, using the data pointer.

Although not shown in *Figure 4-13*, the output enables are also registered in exactly the same manner as the data register, and the SFR access is enabled using the PORTSETUP.0 bit.

The 8051 can read the state of a pin at any time by:

- Reading a PINS register using a MOVX instruction, or
- Reading the corresponding SFR register or bit.

For the bit-addressable registers IOA, IOB, IOC, or IOD, the bit test instructions (jb, jnb) may be used on individual input pins. Bit test instructions may not be used with IOE (at 0xB1) because it is not bit-addressable. However, SFR access is still faster for the IOE register than MOVX access.

The 8051 can read an I/O pin using SFRs, regardless of the state of the PORTSETUP.0 bit.

The following example code tests the state of PORTC bit 2, and jumps to two different routines depending on the result.

```
checkbit:     jb     IOC.2, process_the_one ; jump if bit set
              jmp    process_the_zero       ; it’s low
```

Figure 4-14. Test the State of PORTC, sample code

Chapter 5. EZ-USB FX Enumeration & ReNumeration™

5.1 Introduction

The EZ-USB FX chip is *soft*. 8051 code and data is stored in internal RAM, which is loaded from the host using the USB interface. Peripheral devices that use the EZ-USB FX chip can operate without ROM, EPROM, or FLASH memory, shortening production lead times and making firmware updates a breeze.

To support the soft feature, the EZ-USB FX chip enumerates automatically as a USB device *without firmware*, so the USB interface itself can download 8051 code and descriptor tables. The USB core performs this initial (power-on) enumeration and code download while the 8051 is held in RESET. This initial USB device, which supports code download, is called the "Default USB Device."

After the code descriptor tables have been downloaded from the host to EZ-USB FX RAM, the 8051 is brought out of reset and begins executing the device code. The EZ-USB FX device enumerates again, this time as the loaded device. This patented enumeration process is called "ReNumeration™." The EZ-USB FX chip accomplishes ReNumeration™ by electrically simulating a physical disconnection and re-connection to the USB.

An EZ-USB FX control bit called "RENUM" (ReNumerated) determines which entity, the core or the 8051, handles device requests over endpoint zero. At power-on, the RENUM bit (USBCS.1) is zero, indicating that the USB core automatically handles device requests. Once the 8051 is running, it can set RENUM to 1 to indicate that user 8051 code handles subsequent device requests using its downloaded firmware. *Chapter 9. "EZ-USB FX Endpoint Zero"* describes how the 8051 handles device requests while RENUM=1.

It is also possible for the 8051 to run with RENUM=0 and have the USB core handle certain endpoint zero requests. (See Info Box below).

This chapter deals with the various EZ-USB FX startup modes, and describes the default USB device that is created at initial enumeration.

Another Use for the Default USB Device

The Default USB Device is established at power-on to set up a USB device capable of downloading firmware into EZ-USB FX RAM. Another useful feature of the EZ-USB FX default device is that 8051 code can be written to support the already-configured Generic USB device. Before bringing the 8051 out of reset, the USB core enables certain endpoints and reports them to the host via descriptors. By utilizing the USB default machine (by keeping RENUM=0), the 8051 can, with very little code, perform meaningful USB transfers that use these default endpoints. This accelerates the USB learning curve. To see an example of how little code is actually necessary, take a look at Section 6.11. "Polled Bulk Transfer Example."

5.2 The Default USB Device

The Default USB Device consists of a single USB configuration containing one interface (interface 0) with three alternate settings, 0, 1, and 2. The endpoints reported for this device are shown in Table 5-1. Note that alternate setting zero consumes no interrupt or isochronous bandwidth, as recommended by the USB Specification.

Table 5-1. EZ-USB FX Default Endpoints

Endpoint	**Type**	**Alternate Setting**		
		0	**1**	**2**
		Maximum Packet Size (Bytes)		
0	CTL	64	64	64
1-IN	INT	0	16	64
2-IN	BULK	0	64	64
2-OUT	BULK	0	64	64
4-IN	BULK	0	64	64
4-OUT	BULK	0	64	64
6-IN	BULK	0	64	64
6-OUT	BULK	0	64	64
8-IN	ISO	0	16	256
8-OUT	ISO	0	16	256
9-IN	ISO	0	16	16
9-OUT	ISO	0	16	16
10-IN	ISO	0	16	16
10 OUT	ISO	0	16	16

For the purpose of downloading 8051 code, the Default USB Device requires only CONTROL endpoint zero. Nevertheless, the USB default machine is enhanced to support other endpoints as shown in *Figure 5-2* (note the alternate settings 1 and 2). This enhancement is provided to allow the developer to get a head start generating USB traffic and learning the USB system. All the descriptors are handled automatically by the USB core, so the developer can immediately start writing code to transfer data over USB using these pre-configured endpoints.

When the USB core establishes the Default USB Device, it also sets the proper endpoint configuration bits to match the descriptor data supplied by the USB core. For example, bulk endpoints 2, 4, and 6 are implemented in the Default USB Device, so the USB core sets the corresponding EPVAL bits. *Chapter 6. "EZ-USB FX Bulk Transfers"* contains a detailed explanation of the EPVAL bits.

Tables 5-9 through 5-13 show the various descriptors returned to the host by the USB core when RENUM=0. These tables describe the USB endpoints defined in Table 5-1, along with other USB details. These tables should help you understand the structure of USB descriptors.

5.3 USB Core Response to EP0 Device Requests

Table 5-2 shows how the USB core responds to endpoint zero requests when RENUM=0.

Table 5-2. How the USB Core Handles EP0 Requests When RENUM=0

bRequest	Name	Action: RENUM=0
0x00	Get Status/Device	Returns two zero bytes
0x00	Get Status/Endpoint	Supplies EP Stall bit for indicated EP
0x00	Get Status/Interface	Returns two zero bytes
0x01	Clear Feature/Device	None
0x01	Clear Feature/Endpoint	Clears Stall bit for indicated EP
0x02	(reserved)	None
0x03	Set Feature/Device	None
0x03	Set Feature/Endpoint	Sets Stall bit for indicated EP
0x04	(reserved)	None
0x05	Set Address	Updates FNADD register
0x06	Get Descriptor	Supplies internal table
0x07	Set Descriptor	None
0x08	Get Configuration	Returns internal value
0x09	Set Configuration	Sets internal value
0x0A	Get Interface	Returns internal value (0-3)
0x0B	Set Interface	Sets internal value (0-3)

Table 5-2. How the USB Core Handles EP0 Requests When RENUM=0

bRequest	Name	Action: RENUM=0
0x0C	Sync Frame	None
Vendor Requests		
0xA0	Firmware Load	Upload/Download RAM
0xA1-0xAF	Reserved	Reserved by Cypress Semiconductor
all other		None

The USB host enumerates by issuing:

- Set_Address
- Get_Descriptor
- Set_Configuration (to 1)

As shown in Table 5-2, after enumeration, the USB core responds to the following host requests:

- Set or clear an endpoint stall (Set/Clear Feature_Endpoint).
- Read the stall status for an endpoint (Get_Status_Endpoint).
- Set/Read an 8-bit configuration number (Set/Get_Configuration).
- Set/Read a 2-bit interface alternate setting (Set/Get_Interface).
- Download or upload 8051 RAM.

5.3.1 Port Configuration Bits

To ensure proper operation of the default Keil Monitor, which uses SIO-1 (RXD1 and TXD1), never change the following Port Config bits from “1”:

- PORTBCFG bits 2 (RXD1) and 3 (TXD1).

To ensure the 8051 processor can access the external SRAM (including the Keil Monitor), do not change the following bits from “1”:

- PORTCCFG bits 6 (WR#) and 7 (RD#).

To ensure that no bits are unintentionally changed, all writes to the PORTxCFG registers should use a read-modify-write series of instructions.

5.4 Firmware Load

The USB Specification provides for *vendor-specific requests* to be sent over CONTROL endpoint zero. The EZ-USB FX chip uses this feature to transfer data between the host and EZ-USB FX RAM. The USB core responds to two “Firmware Load” requests, as shown in Tables 5-3 and 5-4.

Table 5-3. Firmware Download

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**0x40**	Vendor Request, OUT	*None required*
1	bRequest	**0xA0**	“Firmware Load”	
2	wValueL	**AddrL**	Starting Address	
3	wValueH	**AddrH**		
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLenghtL	**LenL**	Number of Bytes	
7	wLengthH	**LenH**		

Table 5-4. Firmware Upload

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**0xC0**	Vendor Request, IN	*None required*
1	bRequest	**0xA0**	“Firmware Load”	
2	wValueL	**AddrL**	Starting Address	
3	wValueH	**AddrH**		
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	**LenL**	Number of Bytes	
7	wLengthH	**LenH**		

These requests are always handled by the USB core (RENUM=0 or 1). The bRequest value 0xA0 is reserved by the EZ-USB FX chip. It should never be used for a vendor request. Cypress Semiconductor also reserves bRequest values 0xA1 through 0xAF. Your system should not use these bRequest values.

A host loader program typically writes 0x01 to the CPUCS register to put the 8051 into RESET, loads all or part of the EZ-USB FX RAM with 8051 code, and finally reloads the CPUCS register with 0 to take the 8051 out of RESET. The CPUCS register is the only USB register that can be written using the Firmware Download command.

Firmware loads are restricted to internal EZ-USB FX memory.

When RENUM=1 at Power-On

At power-on, the RENUM bit is normally set to zero so that the EZ-USB FX to handle device requests over CONTROL endpoint zero. This allows the core to download 8051 firmware and then reconnect as the target device.

At power-on, the USB core checks the I^2C-compatible bus for the presence of an EEPROM. If it finds one, and the first byte of the EEPROM is 0xB6, the core copies the contents of the EEPROM into internal RAM, sets the RENUM bit to 1, and un-RESETS the 8051. The 8051 wakes up ready to run the firmware in RAM. The required data format for this load module is described in Section 5.8. "Serial EEPROM Present, First Byte is 0xB6".

5.5 Enumeration Modes

When the EZ-USB FX chip comes out of RESET, the USB core decides how to enumerate based on the contents of an external EEPROM on its I^2C-compatible bus. Table 5-5 shows the choices. In Table 5-5, PID means Product ID, VID means Version ID, and DID means Device ID.

Table 5-5. USB Core Action at Power-Up

First EEPROM byte	USB Core Action
Not 0xB4 or 0xB6	Supplies descriptors, PID/VID/DID from USB Core. Sets RENUM=0.
0xB4	Supplies descriptors from USB core, PID/VID/DID from EEPROM. Sets RENUM=0.
0xB6	Loads EEPROM into EZ-USB FX RAM. Sets RENUM=1; therefore 8051 supplies descriptors, PID/VID/DID.

If no EEPROM is present, or if one is present but the first byte is neither 0xB4 nor 0xB6, the USB core enumerates using internally stored descriptor data, which contains the Cypress Semiconductor VID, PID, and DID. These ID bytes cause the host operating system to load a Cypress Semiconductor device driver. The USB core also establishes the *Default USB device*. This mode is only used for code development and debug.

If a serial EEPROM is attached to the I^2C-compatible bus and its first byte is 0xB4, the USB core enumerates with the same internally stored descriptor data as for the no-EEPROM case, but with one difference. It supplies the PID/VID/DID data from six bytes in the external EEPROM rather than from the USB core. The custom VID/PID/DID in the EEPROM causes the host operating system to load a device driver that is matched to the EEPROM VID/PID/DID. This EZ-USB FX operating mode provides a *soft* USB device using ReNumeration™

If a serial EEPROM is attached to the I^2C-compatible bus and its first byte is 0xB6, the USB core transfers the contents of the EEPROM into internal RAM. The USB core also sets the RENUM bit to 1 to indicate that the 8051 (and not the USB core) responds to device requests over CONTROL endpoint zero (see the Info Box on page 5-6). Therefore, all descriptor data, including VID/DID/PID values, are supplied by the 8051 firmware. The last byte loaded from the EEPROM (to the CPUCS register) releases the 8051 reset signal, allowing the EZ-USB FX chip to come up as a fully, custom device with firmware in RAM.

The following sections discuss these enumeration methods in detail.

The Other Half of the I^2C-Compatible Story

The EZ-USB FX I^2C-compatible controller serves two purposes. First, as described in this chapter, it manages the serial EEPROM interface that operates automatically at power-on to determine the enumeration method. Second, once the 8051 is up and running, the 8051 can access the I^2C-compatible controller for general-purpose use. This makes a wide range of standard I^2C-compatible peripherals available to an EZ-USB FX system.

Other I^2C-compatible devices can be attached to the SCL and SDA lines of the I^2C-compatible bus as long as there is no address conflict with the serial EEPROM described in this chapter. Chapter 4. "EZ-USB FX Input/Output" describes the general-purpose nature of the I^2C-compatible interface.

5.6 No Serial EEPROM

In the simplest scenario, no serial EEPROM is present on the I^2C-compatible bus or an EEPROM is present, but its first byte is not 0xB4 or 0xB6. In this case, descriptor data is supplied by a table internal to the USB core. The EZ-USB FX chip comes on as the *USB Default Device*, with the ID bytes shown in Table 5-6.

Pullup resistors are required on SCL/SDA, even if no device is connected. The resistors are required to allow EZ-USB FX to detect the “no-EEPROM” condition.

Table 5-6. EZ-USB FX Device Characteristics, No Serial EEPROM

Vendor ID	0x0547 (Cypress Semiconductor/ Anchor Chips)
Product ID	0x2235 (EZ-USB FX)
Device Release	0xXXYY (depends on revision)

The USB host queries the device during enumeration, reads the device descriptor, and uses the bytes described in Table 5-6 to determine which software driver to load into the operating system. This is a major USB feature — drivers are dynamically matched with devices and automatically loaded when a device is plugged in.

The “no EEPROM” scenario is the simplest configuration, and also the most limiting. This mode is used only for code development, utilizing Cypress software tools matched to the ID values in Table 5-6.

Reminder

The USB core uses the data in Table 5-6 for enumeration only if the RENUM bit is zero. If RENUM=1, enumeration data is supplied by 8051 code.

5.7 Serial EEPROM Present, First Byte is 0xB4

Table 5-7. EEPROM Data Format for “B4” Load

EEPROM Address	Contents
0	**0xB4**
1	Vendor ID (VID) L
2	Vendor ID (VID) H
3	Product ID (PID) L
4	Product ID (PID) H
5	Device ID (DID) L
6	Device ID (DID) H
7	Config 0
8	Reserved (set to 0x00)

If at power-on, the USB core detects an EEPROM connected to its I^2C-compatible port with the value **0xB4** at address 0, the USB core copies the Vendor ID (VID), Product ID (PID), and Device ID (DID) from the EEPROM (Table 5-7) into internal storage. The USB core then supplies these bytes to the host as part of the Get_Descriptor-Device request. (These six bytes replace only the VID/PID/DID bytes in the default USB device descriptor.) This causes a driver matched to the VID/PID/DID values in the EEPROM, instead of those in the USB core, to be loaded into the OS.

After initial enumeration, the driver downloads 8051 code and USB descriptor data into EZ-USB FX RAM and starts the 8051. The code then ReNumerates™ and comes on as the fully, custom device.

A recommended EEPROM for this application is the Microchip 24LC00, a small (5-pin SOT package) inexpensive 16-byte serial EEPROM. A 24LC01 (128 bytes) or 24LC02 (256 bytes) may be substituted for the 24LC00, but as with the 24LC00, only the first nine bytes are used.

5.8 Serial EEPROM Present, First Byte is 0xB6

If at power-on, the USB core detects an EEPROM connected to its I^2C-compatible port with the value **0xB6** at address 0, the USB core loads the EEPROM data into EZ-USB FX RAM. It also sets the RENUM bit to 1, causing device requests to be fielded by the 8051 instead of the USB core. The EEPROM data format is shown in Table 5-8.

Table 5-8. EEPROM Data Format for “B6” Load

EEPROM Address	Contents
0	**0xB6**
1*	Vendor ID (VID) L
2*	Vendor ID (VID) H
3*	Product ID (PID) L
4*	Product ID (PID) H
5*	Device ID (DID) L
6*	Device ID (DID) H
7	Config 0
8	Reserved (set to 0x00)
9	Length H
10	Length L
11	StartAddr H
12	StartAddr L
---	Data block

Table 5-8. EEPROM Data Format for "B6" Load

EEPROM Address	Contents
---	Length H
---	Length L
---	StartAddr H
---	StartAddr L
---	Data block

---	0x80
---	0x01
---	0x7F
---	0x92
Last	00000000

* Ignored — see Info Box below.

The first byte tells the USB core to copy EEPROM data into RAM. The next six bytes are ignored (See the Info Box below).

One or more data records follow, starting at EEPROM address 9. The maximum value of Length H is 0x03, allowing a maximum of 1,023 bytes per record. Each data record consists of a length, a starting address, and a block of data bytes. The last data record must have the MSB of its Length H byte set to 1. The last data record consists of a single-byte load to the CPUCS register at 0x7F92. Only the LSB of this byte is significant—8051RES (CPUCS.0) is set to zero to bring the 8051 out of reset.

Serial EEPROM data can be loaded into two EZ-USB FX RAM spaces only.

- 8051 program/data RAM at 0x0000-0x1B3F.
- The CPUCS register at 0x7F92 (only bit 0, 8051 RESET, is host-loadable).

VID/PID/DID in a "B6" EEPROM

Bytes 1-6 of a B6 EEPROM can be loaded with VID/PID/DID bytes if it is desired at some point to run the 8051 program with RENUM=0 (USB core handles device requests), using the EEPROM VID/PID/DID rather than the Cypress Semiconductor values built into the USB core.

5.9 Configuration Byte 0

The first configuration byte, **Config 0,** is valid for both EEPROM load formats; B4 and B6.

Config 0

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**48MHZ**	**CLKINV**	**400KHZ**

Figure 5-1. Configuration 0

Bit 2: **48MHZ** *24- or 48-MHz clock*

If 48MHZ=1, the 8051 operates at a clock rate of 48 MHz, and the CLKOUT pin is a 48-MHz square wave. If 48MHZ=0 the 8051 operates at a clock rate of 24 MHz, and the CLKOUT pin is a 24-MHz square wave. This bit is copied to the CPUCS Register (Bit 3, "24/48"), which is read-only to the 8051. Thus the 8051 clock rate is fixed at 24 or 48 MHz at boot time according to the EEPROM contents, and cannot be changed subsequently by the 8051.

If no EEPROM is present the default value is zero, selecting 24-MHz operation.

Bit 1: **CLKINV** *Invert CLKOUT signal*

If CLKINV=0, the CLKOUT signal is not inverted (as shown in all timing diagrams in this manual). If CLKINV=1, the CLKOUT signal is inverted. This bit is copied to the CPUCS Register Bit 2, which is read-only to the 8051. Thus, the CLKOUT polarity is set to invert or non-invert at boot time according to the EEPROM contents, and cannot be changed subsequently by the 8051.

If no EEPROM is present the default value is zero, selecting non-inverting operation.

Bit 0: **400KHZ** *High-speed I^2C-compatible Bus*

If 400KHZ=0, the I^2C-compatible bus operates at approximately 100 KHz. If 400KHZ=1, the I^2C-compatible bus operates at approximately 400 KHz. This bit is copied to the I2CCTL register bit 0, which is read-write to the 8051. Thus the I^2C-compatible bus speed is initially set by the EEPROM bit, and may be changed subsequently by the 8051.

When the EZ-USB FX comes out of RESET, the I^2C-compatible bus operates at 100 KHz mode, ensuring that a 100 KHz device can be used as the boot EEPROM.

5.10 ReNumeration™

Three EZ-USB FX control bits in the USBCS (USB Control and Status) Register control the ReNumeration™ process: DISCON, DISCOE, and RENUM.

USBCS **USB Control and Status** **7FD6**

b7	b6	b5	b4	b3	b2	b1	b0
WAKESRC	-	-	-	**DISCON**	**DISCOE**	**RENUM**	SIGRSUME
R/W	R	R	R	R/W	R/W	R/W	R/W
0	0	0	0	0	1	0	0

Figure 5-2. USB Control and Status Register

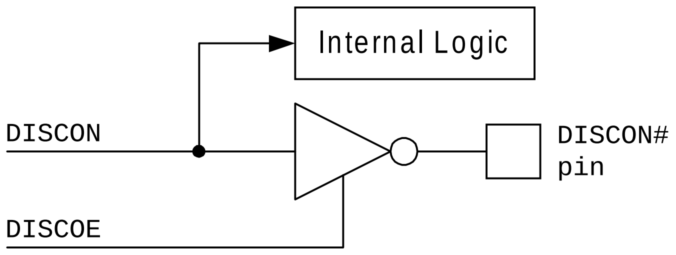


Figure 5-3. Disconnect Pin Logic

The logic for the DISCON and DISCOE bits is shown in *Figure 5-3*. To simulate a USB disconnect, the 8051 writes the value 00001010 to USBCS. This floats the DISCON# pin, and provides an internal DISCON=1 signal to the USB core that causes it to perform disconnect housekeeping.

To re-connect to USB, the 8051 writes the value 00000110 to USBCS. This presents a logic HI to the DISCON# pin, enables the output buffer, and sets the RENUM bit HI to indicate that the 8051 (and not the USB core) is now in control for USB transfers. This arrangement allows connecting the 1,500-ohm resistor directly between the DISCON# pin and the USB D+ line (*Figure 5-4*).

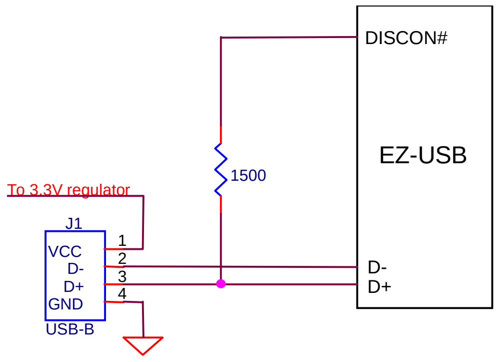


Figure 5-4. Typical Disconnect Circuit

5.11 Multiple ReNumeration™

The 8051 can ReNumerate™ anytime. One use for this capability might be to *fine tune* an isochronous endpoint's bandwidth requests by trying various descriptor values and ReNumerating.

5.12 Default Descriptor

Tables 5-9 through 5-19 show the descriptor data built into the USB core. The tables are presented in the order that the bytes are stored.

Table 5-9. USB Default Device Descriptor

Offset	Field	Description	Value
0	bLength	Length of this Descriptor = 18 bytes	12H
1	bDescriptorType	Descriptor Type = **Device**	01H
2	bcdUSB (L)	USB Specification Version 1.10 (L)	10H
3	bcdUSB (H)	USB Specification Version 1.10 (H)	01H
4	bDeviceClass	Device Class (FF is Vendor-Specific)	FFH
5	bDeviceSubClass	Device Sub-Class (FF is Vendor-Specific)	FFH
6	bDeviceProtocol	Device Protocol (FF is Vendor-Specific)	FFH
7	bMaxPacketSize0	Maximum Packet Size for EP0 = **64 bytes**	40H
8	idVendor (L)	Vendor ID (L) Cypress Semiconductor = 0547H	47H
9	idVendor (H)	Vendor ID (H)	05H
10	idProduct (L)	Product ID (L) EZ-USB FX = 2235H	35H
11	idProduct (H)	Product ID (H)	22H
12	bcdDevice (L)	Device Release Number (BCD,L) (see individual data sheet)	xxH
13	bcdDevice (H)	Device Release Number (BCD,H) (see individual data sheet)	YYH
14	iManufacturer	Manufacturer Index String = None	00H
15	iProduct	Product Index String = None	00H
16	iSerialNumber	Serial Number Index String = None	00H
17	bNumConfigurations	Number of Configurations in this Interface = 1	01H

The Device Descriptor specifies a MaxPacketSize of 64 bytes for endpoint 0, contains Cypress Semiconductor Vendor, Product and Release Number IDs, and uses no string indices. Release Number IDs (*XX* and *YY*) are found in individual Cypress Semiconductor data sheets. The USB core returns this information response to a “Get_Descriptor/Device” host request.

Table 5-10. USB Default Configuration Descriptor

Offset	Field	Description	Value
0	bLength	Length of this Descriptor = 9 bytes	09H
1	bDescriptorType	Descriptor Type = **Configuration**	02H
2	wTotalLength (L)	Total Length (L) Including Interface and Endpoint Descriptors	DAH
3	wTotalLength (H)	Total Length (H)	00H
4	bNumInterfaces	Number of Interfaces in this Configuration	01H
5	bConfigurationValue	Configuration Value Used by Set_Configuration Request to Select this Configuration	01H
6	iConfiguration	Index of String Describing this Configuration = None	00H
7	bmAttributes	Attributes - Bus-Powered, No Wakeup	80H
8	MaxPower	Maximum Power - 100 mA	32H

The configuration descriptor includes a total length field (offset 2-3) that encompasses all interface and endpoint descriptors that follow the configuration descriptor. This configuration describes a single interface (offset 4). The host selects this configuration by issuing a Set_Configuration requests specifying configuration #1 (offset 5).

Table 5-11. USB Default Interface 0, Alternate Setting 0 Descriptor

Offset	Field	Description	Value
0	bLength	Length of the Interface Descriptor	09H
1	bDescriptorType	Descriptor Type = **Interface**	04H
2	bInterfaceNumber	Zero-based Index of this Interface = **0**	00H
3	bAlternateSetting	Alternate Setting Value = **0**	00H
4	bNumEndpoints	Number of Endpoints in this Interface (Not Counting EPO) = **0**	00H
5	bInterfaceClass	Interface Class = Vendor Specific	FFH
6	bInterfaceSubClass	Interface Sub-class = Vendor Specific	FFH
7	bInterfaceProtocol	Interface Protocol = Vendor Specific	FFH
8	iInterface	Index to String Descriptor for this Interface = None	00H

Interface 0, Alternate Setting 0 describes endpoint 0 only. This setting consumes *zero bandwidth*. The interface has no string index.

Table 5-12. USB Default Interface 0, Alternate Setting 1 Descriptor

Offset	Field	Description	Value
0	bLength	Length of the Interface Descriptor	09H
1	bDescriptorType	Descriptor Type = **Interface**	04H
2	bInterfaceNumber	Zero-based Index of this Interface = **0**	00H
3	bAlternateSetting	Alternate Setting Value = **1**	01H
4	bNumEndpoints	Number of Endpoints in this Interface (Not Counting EPO) = **13**	0DH
5	bInterfaceClass	Interface Class = Vendor Specific	FFH
6	bInterfaceSubClass	Interface Sub-class = Vendor Specific	FFH
7	bInterfaceProtocol	Interface Protocol = Vendor Specific	FFH
8	iInterface	Index to String Descriptor for this Interface = None	00H

Interface 0, Alternate Setting 1 has thirteen endpoints, whose individual descriptors follow the interface descriptor. The alternate settings have no string indices.

Table 5-13. Default Interface 0, Alternate Setting 1, INT Endpoint Descriptor

Offset	Field	Description	Value
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **IN1**	81H
3	bmAttributes	XFR Type = **INT**	03H
4	wMaxPacketSize (L)	Maximum Packet Size = **16 Bytes**	10H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds = **10 ms**	0AH

Interface 0, Alternate Setting 1 has one interrupt endpoint, IN1, which has a maximum packet size of 16 and a polling interval of 10 ms.

Table 5-14. Default Interface 0, Alternate Setting 1, Bulk Endpoint Descriptors

Offset	Field	Description	Value
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **IN2**	82H
3	bmAttributes	XFR Type = **BULK**	02H
4	wMaxPacketSize (L)	Maximum Packet Size = **64 Bytes**	40H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds	00H
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **OUT2**	02H
3	bmAttributes	XFR Type = **BULK**	02H
4	wMaxPacketSize (L)	Maximum Packet Size = **64 Bytes**	40H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds	00H
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **IN4**	84H
3	bmAttributes	XFR Type = **BULK**	02H
4	wMaxPacketSize (L)	Maximum Packet Size = **64 Bytes**	40H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds	00H
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **OUT4**	04H
3	bmAttributes	XFR Type = **BULK**	02H
4	wMaxPacketSize (L)	Maximum Packet Size = **64 Bytes**	40H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds	00H
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **IN6**	86H
3	bmAttributes	XFR Type = **BULK**	02H
4	wMaxPacketSize (L)	Maximum Packet Size = **64 Bytes**	40H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds	00H
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **OUT6**	06H
3	bmAttributes	XFR Type = **BULK**	02H
4	wMaxPacketSize (L)	Maximum Packet Size = **64 Bytes**	40H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds	00H

Interface 0, Alternate Setting 1 has six bulk endpoints with max packet sizes of 64 bytes. Even numbered endpoints were chosen to allow endpoint pairing. For more on endpoint pairing, see *Chapter 6. "EZ-USB FX Bulk Transfers"*.

Table 5-15. Default Interface 0, Alternate Setting 1, ISO Endpoint Descriptors

Offset	Field	Description	Value
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **IN8**	88H
3	bmAttributes	XFR Type = **ISO**	01H
4	wMaxPacketSize (L)	Maximum Packet Size = **16 Bytes**	10H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds (1 for iso)	01H
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **OUT8**	08H
3	bmAttributes	XFR Type = **ISO**	01H
4	wMaxPacketSize (L)	Maximum Packet Size = **16 Bytes**	10H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds (1 for iso)	01H
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **IN9**	89H
3	bmAttributes	XFR Type = **ISO**	01H
4	wMaxPacketSize (L)	Maximum Packet Size = **16 Bytes**	10H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds (1 for iso)	01H
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **OUT9**	09H
3	bmAttributes	XFR Type = **ISO**	01H
4	wMaxPacketSize (L)	Maximum Packet Size = **16 Bytes**	10H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds (1 for iso)	01H
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **IN10**	8AH
3	bmAttributes	XFR Type = **ISO**	01H
4	wMaxPacketSize (L)	Maximum Packet Size = **16 Bytes**	10H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds (1 for iso)	01H
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **OUT10**	0AH
3	bmAttributes	XFR Type = **ISO**	01H
4	wMaxPacketSize (L)	Maximum Packet Size = **16 Bytes**	10H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds (1 for iso)	01H

Interface 0, Alternate Setting 1 has six isochronous endpoints with maximum packet sizes of 16 bytes. This is a *low bandwidth* setting.

Table 5-16. USB Default Interface 0, Alternate Setting 2 Descriptor

Offset	Field	Description	Value
0	bLength	Length of the Interface Descriptor	09H
1	bDescriptor Type	Descriptor Type = **Interface**	04H
2	bInterfaceNum-ber	Zero-based Index of this Interface = **0**	00H
3	bAlternateSetting	Alternate Setting Value = **2**	02H
4	bNumEndpoints	Number of Endpoints in this Interface (Not Counting EPO) = **13**	0DH
5	bInterfaceClass	Interface Class = Vendor Specific	FFH
6	bInterfaceSub-Class	Interface Sub-class = Vendor Specific	FFH
7	bInterfaceProto-col	Interface Protocol = Vendor Specific	FFH
8	iInterface	Index to String Descriptor for this Interface = None	00H

Interface 0, Alternate Setting 2 has thirteen endpoints, whose individual descriptors follow the interface descriptor. Alternate Setting 2 differs from Alternate Setting 1 in the maximum packet sizes of its interrupt endpoint and two of its isochronous endpoints (EP8IN and EP8OUT).

Table 5-17. Default Interface 0, Alternate Setting 1, INT Endpoint Descriptor

Offset	Field	Description	Value
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **IN1**	81H
3	bmAttributes	XFR Type = **INT**	03H
4	wMaxPacketSize (L)	Maximum Packet Size = **64 Bytes**	40H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds = **10 ms**	0AH

Alternate Setting 2 for the Interrupt 1-IN increases the maximum packet size for the interrupt endpoint to 64.

Table 5-18. Default Interface 0, Alternate Setting 2, Bulk Endpoint Descriptors

Offset	Field	Description	Value
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptor Type	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **IN2**	82H
3	bmAttributes	XFR Type = **BULK**	02H
4	wMaxPacketSize (L)	Maximum Packet Size = **64 Bytes**	40H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds	00H
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **OUT2**	02H
3	bmAttributes	XFR Type = **BULK**	02H
4	wMaxPacketSize (L)	Maximum Packet Size = **64 Bytes**	40H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds	00H
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **IN4**	84H
3	bmAttributes	XFR Type = **BULK**	02H
4	wMaxPacketSize (L)	Maximum Packet Size = **64 Bytes**	40H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds	00H
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **OUT4**	04H
3	bmAttributes	XFR Type = **ISO**	02H
4	wMaxPacketSize (L)	Maximum Packet Size = **64 Bytes**	40H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds	00H
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **IN6**	86H
3	bmAttributes	XFR Type = **BULK**	02H
4	wMaxPacketSize (L)	Maximum Packet Size = **64 Bytes**	40H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds	00H
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **OUT6**	06H
3	bmAttributes	XFR Type = **BULK**	02H
4	wMaxPacketSize (L)	Maximum Packet Size = **64 Bytes**	40H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds	00H

The bulk endpoints for Alternate Setting 2 are identical to Alternate Setting 1.

Table 5-19. Default Interface 0, Alternate Setting 2, ISO Endpoint Descriptors

Offset	Field	Description	Value
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **IN8**	88H
3	bmAttributes	XFR Type = **ISO**	01H
4	wMaxPacketSize (L)	Maximum Packet Size = **256 Bytes**	00H
5	wMaxPacketSize (H)	Maximum Packet Size - High	01H
6	bInterval	Polling Interval in Milliseconds (1 for iso)	01H
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **OUT8**	08H
3	bmAttributes	XFR Type = **ISO**	01H
4	wMaxPacketSize (L)	Maximum Packet Size = **256 Bytes**	00H
5	wMaxPacketSize (H)	Maximum Packet Size - High	01H
6	bInterval	Polling Interval in Milliseconds (1 for iso)	01H
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **IN9**	89H
3	bmAttributes	XFR Type = **ISO**	01H
4	wMaxPacketSize (L)	Maximum Packet Size = **16 Bytes**	10H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds (1 for iso)	01H
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **OUT9**	09H
3	bmAttributes	XFR Type = **ISO**	01H
4	wMaxPacketSize (L)	Maximum Packet Size = **16 Bytes**	10H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds (1 for iso)	01H
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **IN10**	8AH
3	bmAttributes	XFR Type = **ISO**	01H
4	wMaxPacketSize (L)	Maximum Packet Size = **16 Bytes**	10H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds (1 for iso)	01H
0	bLength	Length of this Endpoint Descriptor	07H
1	bDescriptorType	Descriptor Type = **Endpoint**	05H
2	bEndpointAddress	Endpoint Direction (1 is in) and Address = **OUT10**	0AH
3	bmAttributes	XFR Type = **ISO**	01H
4	wMaxPacketSize (L)	Maximum Packet Size = **16 Bytes**	10H
5	wMaxPacketSize (H)	Maximum Packet Size - High	00H
6	bInterval	Polling Interval in Milliseconds (1 for iso)	01H

The only differences between Alternate Settings 1 and 2 are the maximum packet sizes for EP8IN and EP8OUT. This is a *high-bandwidth* setting.

Chapter 6. EZ-USB FX Bulk Transfers

6.1 Introduction

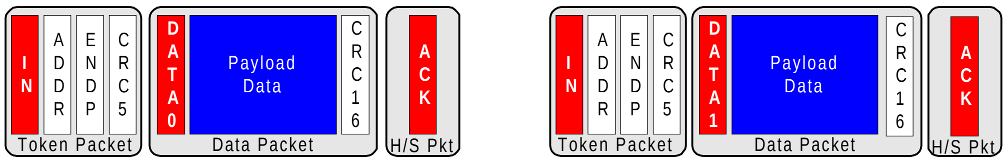


Figure 6-1. Two BULK Transfers, IN and OUT

EZ-USB FX provides sixteen endpoints for BULK, CONTROL, and INTERRUPT transfers, numbered 0-7 as shown in Table 6-1 This chapter describes BULK and INTERRUPT transfers. INTERRUPT transfers are a special case of BULK transfers. EZ-USB FX CONTROL endpoint zero is described in *Chapter 9. "EZ-USB FX Endpoint Zero".*

Table 6-1. EZ-USB FX Bulk, Control, and Interrupt Endpoints

Endpoint	Direction	Type	Size
0	Bidir	Control	64/64
1	IN	Bulk/Int	64
1	OUT	Bulk/Int	64
2	IN	Bulk/Int	64
2	OUT	Bulk/Int	64
3	IN	Bulk/Int	64
3	OUT	Bulk/Int	64
4	IN	Bulk/Int	64
4	OUT	Bulk/Int	64
5	IN	Bulk/Int	64
5	OUT	Bulk/Int	64
6	IN	Bulk/Int	64
6	OUT	Bulk/Int	64
7	IN	Bulk/Int	64
7	OUT	Bulk/Int	64

The USB Specification allows maximum packet sizes of 8, 16, 32, or 64 bytes for bulk data, and 1 - 64 bytes for interrupt data. EZ-USB FX provides the maximum 64 bytes of buffer space for each of its sixteen endpoints: 0-7 IN and 0-7 OUT. Six of the bulk endpoints, 2-IN, 4-IN, 6-IN, 2-OUT, 4-OUT, and 6-OUT may be paired with the next consecutively numbered endpoint to provide double-buffering. This allows one data packet to be serviced by the 8051, while another is in transit over USB. Six *endpoint pairing bits* (USBPAIR Register) control double-buffering.

The 8051 sets fourteen *endpoint valid bits* (IN07VAL, OUT07VAL Registers) at initialization time to tell the USB core which endpoints are active. The default CONTROL endpoint zero is always valid.

Bulk data appears in RAM. Each bulk endpoint has a reserved 64-byte RAM space, a 7-bit count register, and a 2-bit control and status (CS) register. The 8051 can read one bit of the CS Register to determine *endpoint busy*, and write the other to force an endpoint STALL condition.

The 8051 should never read or write an endpoint buffer or byte count register while the endpoint's busy bit is set.

When an endpoint becomes ready for 8051 service, the USB core sets an interrupt request bit. The EZ-USB FX vectored interrupt system separates the interrupt requests by endpoint to automatically transfer control to the ISR (Interrupt Service Routine) for the endpoint requiring service. *Chapter 12. "EZ-USB FX Interrupts"* fully describes this mechanism.

Figure 6-2 illustrates the registers and bits associated with bulk transfers.

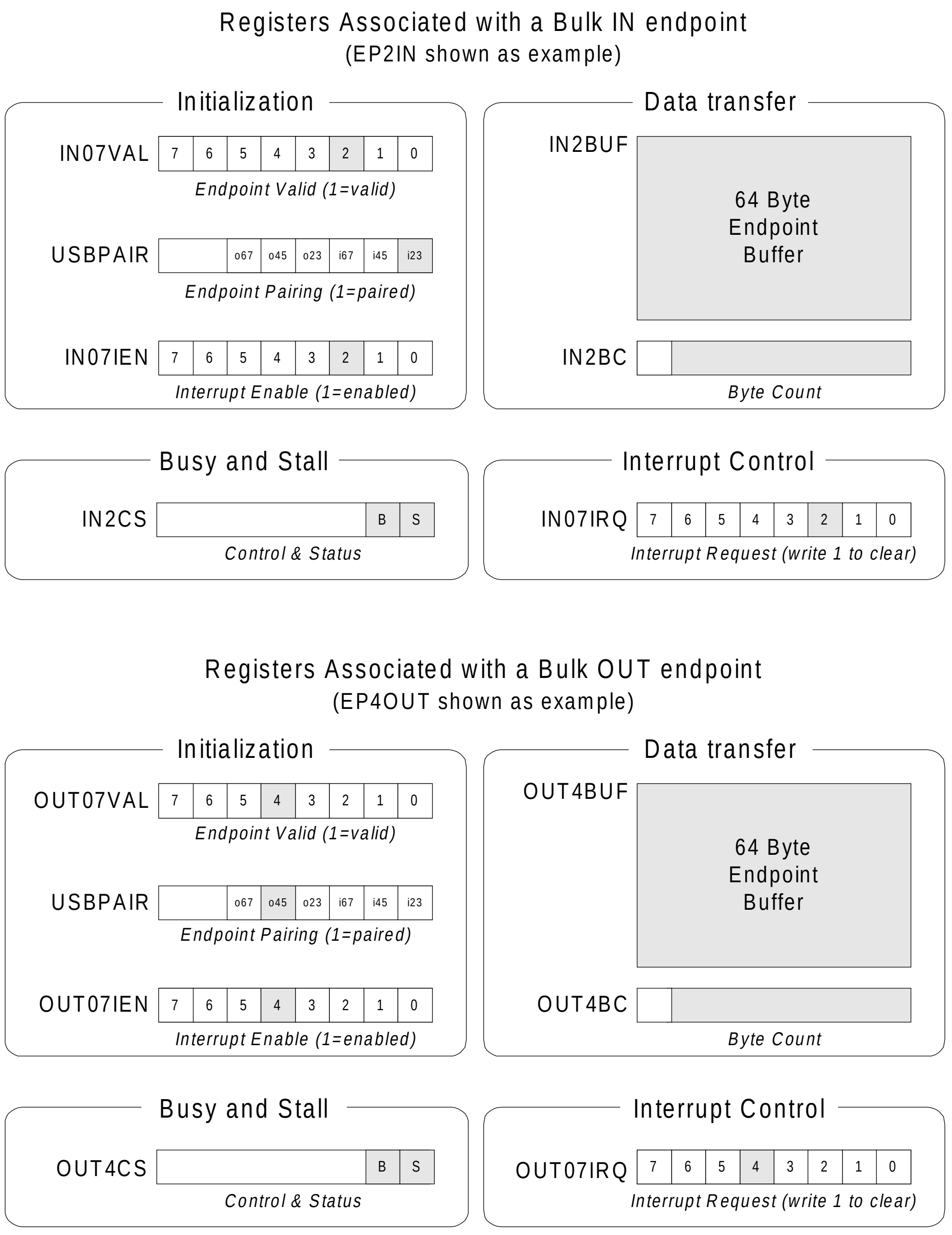


Figure 6-2. Registers Associated with Bulk Endpoints

6.2 Bulk IN Transfers

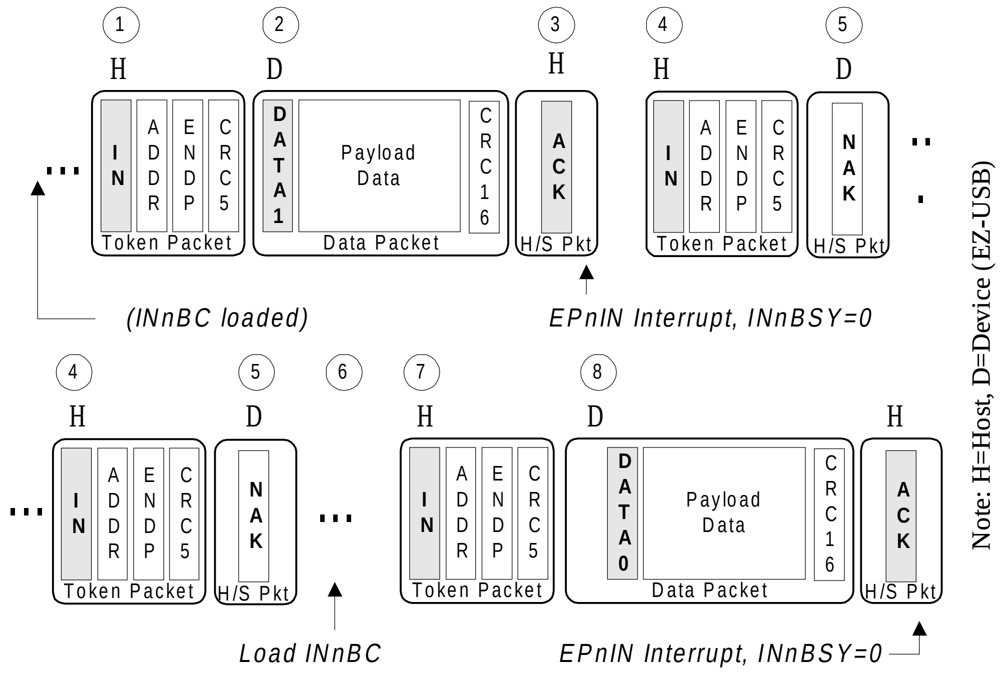


Figure 6-3. Anatomy of a Bulk IN Transfer

USB bulk *IN* data travels from device to host. The host requests an IN transfer by issuing an IN token to the USB core, which responds with data when it is ready. The 8051 indicates *ready* by loading the endpoint's byte count register. If the USB core receives an IN token for an endpoint that is not ready, it responds to the IN token with a *NAK* handshake.

In the bulk IN transfer illustrated in *Figure 6-3*, the 8051 has previously loaded an endpoint buffer with a data packet, and then loaded the endpoint's byte count register with the number of bytes in the packet to arm the next IN transfer. This sets the endpoint's BUSY Bit. The host issues an IN token (1), to which the USB core responds by transmitting the data in the IN endpoint buffer (2). When the host issues an ACK (3), indicating that the data has been received error-free, the USB core clears the endpoint's BUSY Bit and sets its interrupt request bit. This notifies the 8051 that the endpoint buffer is empty. If this is a multi-packet transfer, the host then issues another IN token to get the next packet.

If the second IN token (4) arrives before the 8051 has had time to fill the endpoint buffer, the EZ USB core issues a NAK handshake, indicating *busy* (5). The host continues to send IN tokens (4) and (7) until the data is ready. Eventually, the 8051 fills the endpoint buffer with data, and then loads the endpoint's byte count register (INnBC) with the number of bytes in the packet (6). Loading the byte count re-arms the given endpoint. When the next IN token arrives (7) the USB core transfers the next data packet (8).

6.3 Interrupt Transfers

Interrupt transfers are handled just like bulk transfers.

The only difference between a bulk endpoint and an interrupt endpoint exists in the endpoint descriptor, where the endpoint is identified as type *interrupt*, and a *polling interval* is specified. The polling interval determines how often the USB host issues IN/OUT tokens to the interrupt endpoint.

6.4 EZ-USB FX Bulk IN Example

Suppose 220 bytes are to be transferred to the host using endpoint 6-IN. Further assume that MaxPacketSize of 64 bytes for endpoint 6-IN has been reported to the host during enumeration. Because the total transfer size exceeds the maximum packet size, the 8051 divides the 220-byte transfer into four transfers of 64, 64, 64, and 28 bytes.

After loading the first 64 bytes into IN6BUF (at 0x7C00), the 8051 loads the byte count register IN6BC with the value 64. Writing the byte count register instructs the EZ-USB core to respond to the next host IN token by transmitting the 64 bytes in the buffer. Until the byte count register is loaded to *arm* the IN transfer, any IN tokens issued by the host are answered by EZ-USB FX with NAK (Not-Acknowledge) tokens, telling the USB host that the endpoint is not yet ready with data. The host continues to issue IN tokens to endpoint 6-IN until data is ready for transfer—whereupon the USB core replaces NAKs with valid data.

When the 8051 initiates an IN transfer by loading the endpoint's byte count register, the EZ-USB core sets a busy bit to instruct the 8051 to hold off loading IN6BUF until the USB transfer is finished. When the IN transfer is complete and successfully acknowledged, the EZ-USB core resets the endpoint 6-IN busy bit and generates an endpoint 6-IN interrupt request. If the endpoint 6-IN interrupt is enabled, program control automatically vectors to the data transfer routine for further action (Autovectoring is enabled by setting AVEN=1. Refer to *Chapter 12. "EZ-USB FX Interrupts"*).

The 8051 now loads the next 64 bytes into IN6BUF and then loads the EPINBC Register with 64 for the next two transfers. For the last portion of the transfer, the 8051 loads the final 28 bytes into IN6BUF, and loads IN6BC with 28. This completes the transfer.

Initialization

When the EZ-USB FX chip comes out of RESET, or when the USB host issues a bus reset, the EZ-USB core unarms IN endpoint 1-7 by setting their busy bits to 0. Any IN transfer requests are NAKd until the 8051 loads the appropriate INxBC Register(s). The endpoint valid bits are not affected by an 8051 reset or a USB reset. Chapter 13. "EZ-USB FX Resets" describes the various reset conditions in detail.

The EZ-USB core takes care of USB housekeeping chores, such as handshake verification. When an endpoint 6-IN interrupt occurs, the user is assured that the data loaded by the 8051 into the endpoint buffer was received error-free by the host. The EZ-USB core automatically checks the handshake information from the host and re-transmits the data, if the host indicates an error by not ACKing.

6.5 Bulk OUT Transfers

USB bulk *OUT* data travels from host to device. The host requests an OUT transfer by issuing an OUT token to EZ-USB FX, followed by a packet of data. The USB core then responds with an ACK, if it correctly received the data. If the endpoint buffer is not ready to accept data, the USB core discards the host's OUT data and returns a NAK token, indicating "not ready." In response, the host continues to send OUT tokens *and data* to the endpoint until the USB core responds with an ACK.

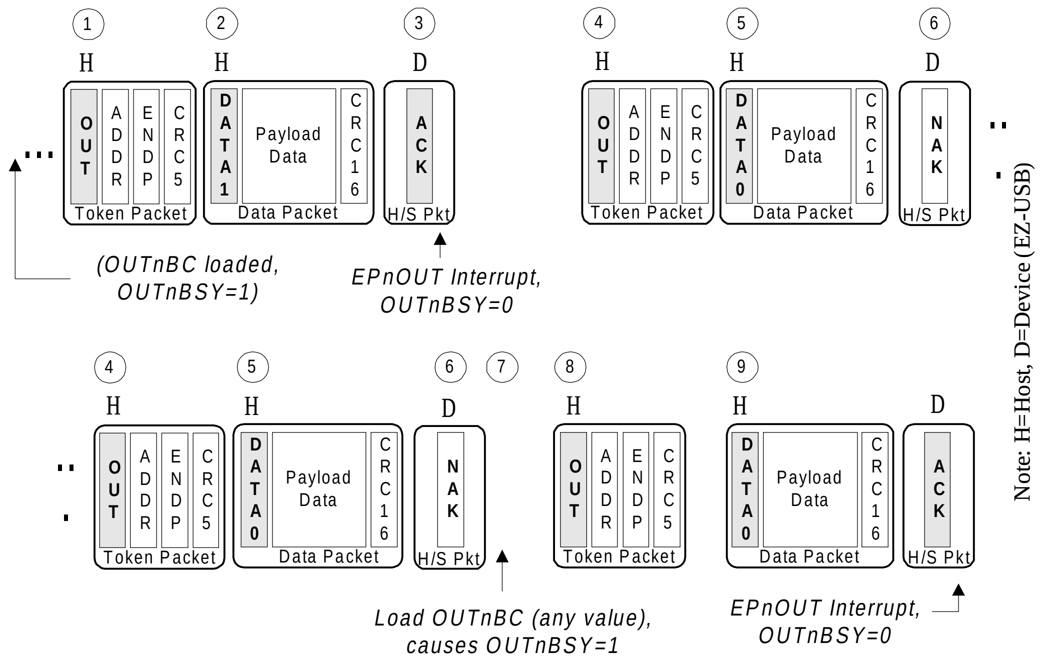


Figure 6-4. Anatomy of a Bulk OUT Transfer

Each EZ-USB FX bulk OUT endpoint has a byte count register, which serves two purposes. The 8051 *reads* the byte count register to determine how many bytes were received during the last OUT transfer from the host. The 8051 *writes* the byte count register (with any value) to tell the USB core that is has finished reading bytes from the buffer, making the buffer available to accept the next OUT transfer. The OUT endpoints come up (after reset) *armed*, so the byte count register writes are required only for OUT transfers after the first one.

In the bulk OUT transfer illustrated in *Figure 6-4*, the 8051 has previously loaded the endpoint's byte count register with any value to arm receipt of the next OUT transfer. Loading the byte count register causes the EZ-USB core to set the OUT endpoint's busy bit to 1, indicating that the 8051 should not use the endpoint's buffer.

The host issues an OUT token (1), followed by a packet of data (2), which the USB core acknowledges, clears the endpoint's busy bit and generates an interrupt request (3). This notifies the 8051 that the endpoint buffer contains valid USB data. The 8051 reads the endpoint's byte count register to find out how many bytes were sent in the packet, and transfers that many bytes out of the endpoint buffer.

In a multi-packet transfer, the host then issues another OUT token (4) along with the next data packet (5). If the 8051 has not finished emptying the endpoint buffer, the EZ-USB FX host issues a NAK, indicating *busy* (6). The data at (5) is shaded to indicate that the USB core discards it, and does not over-write the data in the endpoint's OUT buffer.

The host continues to send OUT tokens (4, 5, and 6) that are greeted by NAKs until the buffer is ready. Eventually, the 8051 empties the endpoint buffer data, and then loads the endpoint's byte count register (7) with any value to re-arm the USB core. Once armed and when the next OUT token arrives (8) the USB core accepts the next data packet (9).

Initializing OUT Endpoints

When the EZ-USB FX chip comes out of reset, or when the USB host issues a bus reset, the USB core arms OUT endpoints 1-7 by setting their busy bits to 1. Therefore, they are initially ready to accept one OUT transfer from the host. Subsequent OUT transfers are NAKd until the appropriate OUTnBC Register is loaded to re-arm the endpoint.

The EZ-USB core takes care of USB housekeeping chores such as CRC checks and data toggle PIDs. When an endpoint 6-OUT interrupt occurs and the busy bit is cleared, the user is assured that the data in the endpoint buffer was received error-free from the host. The USB core automatically checks for errors, and requests the host to re-transmit data if it detects any errors using the built-in USB error checking mechanisms (CRC checks and data toggles).

6.6 Endpoint Pairing

Table 6-2. Endpoint Pairing Bits (in the USB PAIR Register)

Bit	**5**	**4**	**3**	**2**	**1**	**0**
Name	**PR6OUT**	**PR4OUT**	**PR2OUT**	**PR6IN**	**PR4IN**	**PR2IN**
Paired	6 OUT	4 OUT	2 OUT	6 IN	4 IN	2 IN
Endpoints	7 OUT	5 OUT	3 OUT	7 IN	5 IN	3 IN

The 8051 sets endpoint pairing bits to 1 to enable double-buffering of the bulk endpoint buffers. With double-buffering enabled, the 8051 can operate on one data packet while another is being transferred over USB. The endpoint busy and interrupt request bits function identically, so the 8051 code requires little code modification to support double-buffering.

When an endpoint is paired, the 8051 uses only the even-numbered endpoint of the pair. The 8051 should not use the paired odd endpoint. For example, suppose it is desired to use endpoint 2-IN as a double-buffered endpoint. This pairs the IN2BUF and IN3BUF buffers, although the 8051 accesses the IN2BUF buffer only. The 8051 sets PR2IN=1 (in the USBPAIR Register) to enable pairing; sets IN2VAL=1 (in the IN07VAL Register) to make the endpoint valid; and then uses the IN2BUF buffer for all data transfers. The 8051 should not write the IN3VAL Bit, enable IN3 interrupts, access the EP3IN buffer, or load the IN3BC byte count register.

6.7 Paired IN Endpoint Status

INnBSY=1 indicates that *both* endpoint buffers are in use, and the 8051 should not load new IN data into the endpoint buffer. When INnBSY=0, either one or both of the buffers is available for loading by the 8051. The 8051 can keep an internal count that increments on EPnIN interrupts and decrements on byte count loads to determine whether one or two buffers are free. Or, the 8051 can simply check for INnBSY=0 after loading a buffer (and loading its byte count register to re-arm the endpoint) to determine if the other buffer is free.

If an IN endpoint is paired and it is desired to clear the busy bit for that endpoint, do the following: (a) write any value to the even endpoint's byte count register twice, and (b) clear the busy bit for both endpoints in the pair. This is the only code difference between paired and unpaired use of an IN endpoint.

A bulk IN endpoint interrupt request is generated whenever a packet is successfully transmitted over USB. The interrupt request is independent of the busy bit. If both buffers are filled and one is sent, the busy bit transitions from 1 to 0. If one buffer is filled and then sent, the busy bit starts and remains at 0. In either case, an interrupt request is generated to tell the 8051 that a buffer is free.

6.8 Paired OUT Endpoint Status

OUTnBSY=1 indicates that both endpoint buffers are empty, and no data is available to the 8051. When OUTnBSY=0, either one or both of the buffers holds USB OUT data. The 8051 can keep an internal count that increments on EPnOUT interrupts and decrements on byte count loads to determine whether one or two buffers contain data. Or, the 8051 can simply check for OUTnBSY=0 after unloading a buffer (and loading its byte count register to re-arm the endpoint) to determine if the *other* buffer contains data.

6.9 Reusing Bulk Buffer Memory

Table 6-3. EZ-USB FX Endpoint 0-7 Buffer Addresses

Endpoint Buffer	Address	Mirrored
IN0BUF	7F00-7F3F	1F00-1F3F
OUT0BUF	7EC0-7EFF	1EC0-1EFF
IN1BUF	7E80-7EBF	1E80-1EBF
OUT1BUF	7E40-7E7F	1E40-1E7F
IN2BUF	7E00-7E3F	1E00-1E3F
OUT2BUF	7DC0-7DFF	1DC0-1DFF
IN3BUF	7D80-7DBF	1D80-1DBF
OUT3BUF	7D40-7D7F	1D40-1D7F
IN4BUF	7D00-7D3F	1D00-1D3F
OUT4BUF	7CC0-7CFF	1CC0-1CFF
IN5BUF	7C80-7CBF	1C80-1CBF
OUT5BUF	7C40-7C7F	1C40-1C7F
IN6BUF	7C00-7C3F	1C00-1C3F
OUT6BUF	7BC0-7BFF	1BC0-1BFF
IN7BUF	7B80-7BBF	1B80-1BBF
OUT7BUF	7B40-7B7F	1B40-1B7F

Table 6-3 shows the RAM locations for the sixteen 64-byte buffers for endpoints 0-7 IN and OUT. These buffers are positioned at the bottom of the EZ-USB FX register space so that any buffers not used for endpoints can be reclaimed as general purpose data RAM. The top of memory for the 8-KB EZ-USB FX part is at 0x1B3F. However, if the endpoints are allocated in ascending order, starting with the lowest numbered endpoints, the higher numbered unused endpoints can effectively move the top of memory to utilize the unused endpoint buffer RAM as data memory. For example, an application that uses endpoint 1-IN, 2-IN/OUT (paired), 4-IN and 4-OUT can use 0x1B40-0x1CBF as data memory. *Chapter 3. "EZ-USB FX Memory"* provides full details of the EZ-USB FX memory map.

Uploads or Downloads to unused bulk memory can be done only at the Mirrored (low) addresses shown in Table 6-3.

6.10 Data Toggle Control

The EZ-USB core automatically maintains the data toggle bits during bulk, control and interrupt transfers. As explained in *Chapter 1. "Introducing EZ-USB FX"*, the toggle bits are used to detect certain transmission errors so that erroneous data can be re-sent.

In certain circumstances, the host resets its data toggle to “DATA0”:

- After sending a Clear_Feature: Endpoint Stall request to an endpoint.
- After setting a new interface.
- After selecting a new alternate setting.

In these cases, the 8051 can directly clear the data toggle for each of the bulk/interrupt/control endpoints, using the TOGCTL Register (*Figure 6-5*).

TOGCTL **Data Toggle Control** **7FD7**

b7	b6	b5	b4	b3	b2	b1	b0
Q	**S**	**R**	**IO**	**0**	**EP2**	**EP1**	**EP0**
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 6-5. Bulk Endpoint Toggle Control

The I/O bit selects the endpoint direction (1=IN, 0=OUT), and the EP2-EP1-EP0 Bits select the endpoint number. The *Q* Bit, which is read-only, indicates the state of the data toggle for the selected endpoint. Writing R=1 sets the data toggle to DATA0, and writing S=1 sets the data toggle to DATA1.

Currently, there appears to be no reason to set a data toggle to DATA1. The S Bit is provided for generality.

To clear an endpoint's data toggle, the 8051 performs the following sequence:

1. Selects the endpoint by writing the value 000D0EEE binary to the TOGCTL Register, where D is the direction and EEE is the endpoint number.
2. Clears the toggle bit by writing the value 001D0EEE binary to the TOGCTL Register.

After Step 1, the 8051 may read the state of the data toggle by reading the TOGCTL Register checking Bit 7.

6.11 Polled Bulk Transfer Example

The following code sample illustrates the EZ-USB FX registers used for a simple bulk transfer. In this example, 8051 Register R1 keeps track of the number of endpoint 2-IN transfers and Register R2 keeps track of the number of endpoint 2-OUT transfers (mod-256). Every endpoint 2-IN transfer consists of 64 bytes of a decrementing count, with the first byte replaced by the number of IN transfers and the second byte replaced by the number of OUT transfers.

```
start:         mov       SP,#STACK-1         ; set stack
               mov       dptr,#IN2BUF        ; fill EP2IN buffer with
               mov       r7,#64              ; decrementing counter
fill:          mov       a,r7
               movx      @dptr,a
               inc       dptr
               djnz      r7,fill
;
               mov       r1,#0               ; r1 is IN token counter
               mov       r2,#0               ; r2 is OUT token counter
               mov       dptr,#IN2BC         ; Point to EP2 Byte Count Register
               mov       a,#40h              ; 64-byte transfer
               movx      @dptr,a             ; arm the IN2 transfer
;
loop:          mov       dptr,#IN2CS         ; poll the EP2-IN Status
movx           a,@dptr
               jnb       acc.1,serviceIN2    ; not busy--service endpoint
               mov       dptr,#OUT2CS
               movx      a,@dptr
               jb        acc.1,loop          ; EP2OUT is busy--keep looping
;
serviceOUT2:
               inc       r2                  ; OUT packet counter
               mov       dptr,#OUT2BC        ; load byte count Register to re-arm
               movx      @dptr,a             ; (any value)
               sjmp      loop
;
serviceIN2:
               inc       r1                  ; IN packet counter
               mov       dptr,#IN2BUF        ; update the first data byte
               mov       a,r1                ; in EP2IN buffer
               movx      @dptr,a
               inc       dptr                ; second byte in buffer
```

```
34              mov      a,r2              ; get number of OUT packets
35              movx     @dptr,a
36              mov      dptr,#IN2BC       ; point to EP2IN Byte Count Register
37              mov      a,#40h
38              movx     @dptr,a           ; load bc=64 to re-arm IN2
39              sjmp     loop
40 ;
41              END
```

Figure 6-6. Example Code for a Simple (Polled) BULK Transfer

The code at lines 2-7 fills the endpoint 2-IN buffer with 64 bytes of a decrementing count. Two 8-bit counts are initialized to zero at lines 9 and 10. An endpoint 2-IN transfer is *armed* at lines 11-13, which load the endpoint 2-IN byte count register IN2BC with 64. Then, the program enters a polling loop at lines 15-20, where it checks two flags for endpoint 2 servicing. Lines 15-17 check the endpoint 2-IN busy bit in IN2CS Bit 1. Lines 18-20 check the endpoint 2-OUT busy bit in OUT2CS Bit 1. When busy=1, the EZ-USB core is currently using the endpoint buffers, and the 8051 should not access them. When busy=0, new data is ready for service by the 8051.

For both IN and OUT endpoints, the busy bit is set when the EZ-USB core is using the buffers, and cleared by loading the endpoint's byte count register. The byte count value is meaningful for IN transfers because it tells the USB core how many bytes to transfer in response to the next IN token. The 8051 can load any byte count OUT transfers, because only the act of loading the register is significant—loading OUTnBC arms the OUT transfer and sets the endpoint's busy bit.

When an OUT packet arrives in OUT2BUF, the service routine at lines 22-26 increments R2, loads the byte count (any value) into OUT2BC to re-arm the endpoint (lines 24-25), and jumps back to the polling routine. This program does not use OUT2BUF data. It simply counts the number of endpoint 2-OUT transfers.

When endpoint 2-IN is ready for the 8051 to load another packet into IN2BUF, the polling loop jumps to the endpoint 2-IN service routine at lines 28-39. First, R1 is incremented (line 29). The data pointer is set to IN2BUF at line 30, and Register R1 is loaded into the first byte of the buffer (lines 31-32). The data pointer is advanced to the second byte of IN2BUF at line 33, and Register R2 is loaded into the buffer (lines 34-35). Finally, the byte count 40H (64 decimal bytes) is loaded into the byte count Register IN2BC to arm the next IN transfer at lines 36-38, and the routine returns the polling loop.

6.12 Enumeration Note

The code in the example listed above is complete, and it runs on the EZ-USB FX chip. You may be wondering about the *missing step*, which reports the endpoint characteristics to the host during the enumeration process. The reason this code runs without any enumeration code is that the EZ-USB FX chip comes on as a fully-functional USB device with certain endpoints already configured and reported to the host. Endpoint 2 is included in this default configuration. The full default configuration is described in *Chapter 5. "EZ-USB FX Enumeration & ReNumeration™"*.

6.13 Bulk Endpoint Interrupts

All USB interrupts activate the 8051 *INT 2* interrupt. If enabled, INT2 interrupts cause the 8051 to push the current program counter onto the stack, and then execute a jump to location 0x43, where the programmer has inserted a jump instruction to the interrupt service routine (ISR). If the AVEN (Autovector Enable) bit is set, the USB core inserts a special byte at location 0x45, which directs the jump instruction to a table of jump instructions that transfer control the endpoint-specific ISR.

Table 6-4. 8051 INT2 Interrupt Vector

Location	Op-Code	Instruction
0x43	02	LJMP
0x44	AddrH	
0x45	AddrL*	

*Replaced by EZ-USB Core if AVEN=1.

The byte inserted by the EZ-USB core at address 0x45 depends on which bulk endpoint requires service. Table 6-5 shows all INT2 vectors, with the bulk endpoint vectors shaded.

Table 6-5. Byte Inserted by USB Core at Location 0x45 if AVEN=1

Interrupt	Inserted Byte at 0x45
SUDAV	**0x00**
SOF	**0x04**
SUTOK	**0x08**
SUSPEND	**0x0C**
USBRES	**0x10**
Reserved	**0x14**
EP0-IN	**0x18**
EP0-OUT	**0x1C**
EP1-IN	**0x20**
EP1OUT	**0x24**
EP2IN	**0x28**
EP2OUT	**0x2C**
EP3-IN	**0x30**
EP3-OUT	**0x34**
EP4-IN	**0x38**
EP4-OUT	**0x3C**
EP5-IN	**0x40**
EP5-OUT	**0x44**
EP6-IN	**0x48**
EP6-OUT	**0x4C**
EP7-IN	**0x50**
EP7-OUT	**0x54**

The vector values are four bytes apart. This allows the programmer to build a jump table to each of the interrupt service routines. Note that the jump table must begin on a page (256 byte) boundary because the first vector starts at 00. If Autovectoring is not used (AVEN=0), the IVEC Register may be directly inspected to determine the USB interrupt source. (See *Section 12.11. "Autovector Coding"*).

Each bulk endpoint interrupt has an associated interrupt enable bit (in IN07IEN and OUT07IEN), and an interrupt request bit (in IN07IRQ and OUT07IRQ). These IRQ bits can be cleared by writ-ing to the INT2CLR SFR Register.

Any USB ISR should clear the 8051 INT2 interrupt request bit before clearing any of the EZ-USB FX endpoint IRQ bits, to avoid losing interrupts. Interrupts are discussed in more detail in Chapter 12. "EZ-USB FX Interrupts"

Individual interrupt request bits are cleared by writing “1” to them to simplify code. For example, to clear the endpoint 2-IN IRQ, simply write “0000100” to IN07IRQ. This will not disturb the other interrupt request bits. ***Do not read the contents of IN07IRQ, logical-OR the contents with 01, and write it back****. This clears all other pending interrupts because you are writing “1”s to them.*

6.14 Interrupt Bulk Transfer Example

This following simple (but fully-functional) example illustrates the bulk transfer mechanism using interrupts. In the example program, BULK endpoint 6 is used to loop data back to the host. Data sent by the host over endpoint 6-OUT is sent back over endpoint 6-IN.

1. Set up the jump table.

```
                  CSEG    AT 300H       ; any page boundary
USB_Jump_Table:
                  ljmp    SUDAV_ISR     ; SETUP Data Available
                  db      0             ; make a 4-byte entry
                  ljmp    SOF_ISR       ; SOF
                  db      0
                  ljmp    SUTOK_ISR     ; SETUP Data Loading
                  db      0
                  ljmp    SUSP_ISR      ; Global Suspend
                  db      0
                  ljmp    URES_ISR      ; USB Reset
                  db      0
                  ljmp    IBN_ISR
                  db      0
                  ljmp    EP0IN_ISR
                  db      0
                  ljmp    EP0OUT_ISR
                  db      0
                  ljmp    EP1IN_ISR
                  db      0
                  ljmp    EP1OUT_ISR
                  db      0
                  ljmp    EP2IN_ISR
                  db      0
                  ljmp    EP2OUT_ISR
                  db      0
                  ljmp    EP3IN_ISR
                  db      0
                  ljmp    EP3OUT_ISR
                  db      0
                  ljmp    EP4IN_ISR
                  db      0
                  ljmp    EP4OUT_ISR
                  db      0
                  ljmp    EP5IN_ISR
                  db      0
                  ljmp    EP5OUT_ISR
                  db      0
                  ljmp    EP6IN_ISR     ; Used by this example
                  db      0
                  ljmp    EP6OUT_ISR    ; Used by this example
                  db      0
                  ljmp    EP7IN_ISR
                  db      0
                  ljmp    EP7OUT_ISR
                  db      0
```

Figure 6-7. Interrupt Jump Table

This table contains all of the USB interrupts, even though only the jumps for endpoint 6 are used for the example. It is convenient to include this table in any USB application that uses interrupts. Be sure to locate this table on a page boundary.

2. Write the INT2 interrupt vector.

```
; -----------------
; Interrupt Vectors
; -----------------
        org             43h                     ; int2 is the USB vector
        ljmp            USB_Jump_Table          ; Autovector will replace byte 45
```

Figure 6-8. INT2 Interrupt Vector

3. Write the interrupt service routine.

   Put it anywhere in memory and the jump table in step 1 will automatically jump to it.

```
; -----------------------------
; USB Interrupt Service Routine
; -----------------------------
EP6OUT_ISR
             push   acc
;
             mov    a,EXIF          ; clear INT2 (USB) IRQ flag
             clr    acc.4
             mov    EXIF,a
;
             mov    INT2CLR,a       ; use whatever value is in acc
;
             setb   got_EP6-DATA
; Do Interrupt processing here — set flags, whatever . . .
;                                     spend time here or not
             pop    acc
             reti
```

Figure 6-9. Interrupt Service Routine (ISR) for Endpoint 6-OUT

In this example, the ISR simply sets the 8051 flag “got_EP6_data” to indicate to the background program that the endpoint requires service.

4. Write the endpoint 6 transfer program.

```
1  loop:      jnb    got_EP6_data,loop
2             clr    got_EP6_data         ; clear my flag
3  ;
4  ; The user sent bytes to OUT6 endpoint using the USB Control Panel.
5  ; Find out how many bytes were sent.
6  ;
7             mov    dptr,#OUT6BC         ; point to OUT6 byte count register
8             movx   a,@dptr              ; get the value
9             mov    r7,a                 ; stash the byte count
10            mov    r6,a                 ; save here also
11 ;
12 ; Transfer the bytes received on the OUT6 endpoint to the IN6 endpoint
13 ; buffer.  Number of bytes in r6 and r7.
14 ;
15            mov    dptr,#OUT6BUF        ; first data pointer points to EP2OUT buffer
16            inc    dps                  ; select the second data pointer
17            mov    dptr,#IN6BUF         ; second data pointer points to EP2IN buffer
18            inc    dps                  ; back to first data pointer
19 transfer:  movx   a,@dptr              ; get OUT byte
20            inc    dptr                 ; bump the pointer
21            inc    dps                  ; second data pointer
22            movx   @dptr,a              ; put into IN buffer
23            inc    dptr                 ; bump the pointer
24            inc    dps                  ; first data pointer
25            djnz   r7,transfer
26 ;
27 ; Load the byte count into IN6BC.  This arms in IN transfer
28 ;
29            mov    dptr,#IN6BC
30            mov    a,r6                 ; get other saved copy of byte count
31            movx   @dptr,a              ; this arms the IN transfer
32 ;
33 ; Load any byte count into OUT6BC.  This arms the next OUT transfer.
34 ;
35            mov    dptr,#OUT6BC
36            movx   @dptr,a              ; use whatever is in acc
37            sjmp   loop                 ; start checking for another OUT6 packet
```

Figure 6-10. Background Program Transfers Endpoint 6-OUT Data to Endpoint 6-IN

The main program loop tests the “got_EP6_data” flag, waiting until it is set by the endpoint 6 OUT interrupt service routine in *Figure 6-10*. This indicates that a new data packet has arrived in OUT6BUF. Then the service routine is entered, where the flag is cleared in line 2. The number of bytes received in OUT6BUF is retrieved from the OUT6BC Register (Endpoint 6 Byte Count) and saved in Registers R6 and R7 in lines 7-10.

The dual data pointers are initialized to the source (OUT6BUF) and destination (IN6BUF) buffers for the data transfer in lines 15-18. These labels represent the start of the 64-byte buffers for endpoint 6-OUT and endpoint 6-IN, respectively. Each byte is read from the OUT6BUF buffer and written to the IN6BUF buffer in lines 19-25. The saved value of OUT6BC is used as a loop counter in R7 to transfer the exact number of bytes that were received over endpoint 6-OUT.

When the transfer is complete, the program loads the endpoint 6-IN byte count Register IN6BC with the number of loaded bytes (from R6) to *arm* the next endpoint 6-IN transfer in lines 29-31. Finally, the 8051 loads any value into the endpoint 6 OUT byte count Register OUT6BC to arm the next OUT transfer in lines 35-36. Then the program loops back to check for more endpoint 6-OUT data.

DMA cannot be used for this Loopback since the source and destination would be in the same RAM block.

5. Initialize the endpoints and enable the interrupts.

```
start:    mov     SP,#STACK-1       ; set stack
;
; Enable USB interrupts and Autovector
;
          mov     dptr,#USBBAV      ; enable Autovector
          movx    a,@dptr
          setb    acc.0             ; AVEN bit is bit 0
          movx    @dptr,a
;
          movx    dptr,#USBBAV
          movx    a, @dptr
          setb    acc.4             ; enable the SFR-clearing feature
          movx    @dptr, a          ; for INT2
;
          mov     dptr,#OUT07IEN    ; ‘EP0-7 OUT int enables’ Register
;         mov     a,#01000000b      ; set bit 6 for EP6OUT interrupt enable
          movx    @dptr,a           ; enable EP6OUT interrupt
;
; Enable INT2 and 8051 global interrupts
;
          setb    ex2               ; enable int2 (USB interrupt)
          setb    EA                ; enable 8051 interrupts
          clr     got_EP6_data      ; clear my flag
```

Figure 6-11. Initialization Routine

The initialization routine sets the stack pointer, and enables the EZ-USB FX Autovector by setting USBBAV.0 to 1. Then it enables the endpoint 6-OUT interrupt, all USB interrupts (INT2), and the 8051 global interrupt (EA) and finally clears the flag indicating that endpoint 6-OUT requires service.

Once this structure is put into place, it is quite easy to service any or all of the bulk endpoints. To add service for endpoint 2-IN, for example, simply write an endpoint 2-IN interrupt service routine with starting address EP2IN_ISR (to match the address in the jump table in step 1), and add its valid and interrupt enable bits to the “init” routine.

6.15 Enumeration Note

The code in the previous example is complete, and runs on the EZ-USB FX chip. You may be wondering about the *missing step*, which reports the endpoint characteristics to the host during the enumeration process. The reason this code runs without any enumeration code is that the EZ-USB FX chip comes on as a fully-functional USB device with certain endpoints already configured and reported to the host. Endpoint 6 is included in this default configuration. The full default configuration is described in *Chapter 5. "EZ-USB FX Enumeration & ReNumeration™"*.

Portions of the above code are not necessary for the default configuration (such as setting the endpoint valid bits), but the code is included to illustrate all of the EZ-USB FX registers used for bulk transfers

6.16 The Autopointer

Bulk endpoint data is available in 64-byte buffers in EZ-USB FX RAM. In some cases it is preferable to access bulk data as a FIFO register rather than as a RAM. The EZ-USB core provides a special data pointer that automatically increments when data is transferred. Using this Autopointer, the 8051 can access any contiguous block of internal EZ-USB FX RAM or off-chip memory as a FIFO.

AUTOPTRH **Autopointer Address High** **7FE3**

b7	b6	b5	b4	b3	b2	b1	b0
A15	**A14**	**A13**	**A12**	**A11**	**A10**	**A9**	**A8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

AUTOPTRL **Autopointer Address Low** **7FE4**

b7	b6	b5	b4	b3	b2	b1	b0
A7	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

AUTODATA			Autopointer Data				7FE5
b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 6-12. Autopointer Registers

The 8051 first loads AUTOPTRH and AUTOPTRL with a RAM address (for example the address of a bulk endpoint buffer). Then, as the 8051 reads or writes data to the data Register AUTODATA, the address is supplied by AUTOPTRH/L, which automatically increments after every read or write to the AUTODATA Register. The AUTOPTRH/L Registers may be written or read at anytime. These registers maintain the current pointer address, so the 8051 can read them to determine where the next byte will be read or written.

The 8051 code example in *Figure 6-13* uses the Autopointer to transfer a block of eight data bytes from the endpoint 4 OUT buffer to internal 8051 memory.

```
Init:   mov     dptr,#AUTOPTRH
        mov     a,#HIGH(OUT4BUF)  ; High portion of OUT4BUF buffer
        movx    @dptr,a           ; Load AUTOPTRH
        mov     dptr,#AUTOPTRL
        mov     a,#LOW(OUT4BUF)   ; Low portion of OUT4BUF buffer address
        movx    @dptr,a           ; Load AUTOPTRL
        mov     dptr,#AUTODATA    ; point to the ‘fifo’ Register
        mov     r0,#80H           ; store data in upper 128 bytes of 8051 RAM
        mov     r2,#8             ; loop counter
;
loop:   movx    a,@dptr           ; get a ‘fifo’ byte
        mov     @r0,a             ; store it
        inc     r0                ; bump destination pointer
                                  ; (NOTE: no ‘inc dptr’ required here)
        djnz    r2,loop           ; do it eight times
```

Figure 6-13. Use of the Autopointer

As the comment in the second to last line indicates, the Autopointer saves an “inc dptr” instruction that would be necessary if one of the 8051 data pointers were used to access the OUT4BUF RAM data. This improves the transfer time.

The Autopointer works only with internal program/data RAM. It does not work with memory outside the chip, or with internal RAM that is made available when ISODISAB=1. See Section 10.6.1. "Disable ISO" for a description of the ISODISAB bit.

The EZ-USB FX chip should never be a speed bottleneck in a USB system since it can DMA Data at 24Mhz. It also gives the 8051 ample time for other processing duties between endpoint buffer loads.

The Autopointer can be used to quickly move data anywhere in RAM, not just the bulk endpoint buffers. For example, it can be used to good effect in an application that calls for transferring a block of data into RAM, processing the data, and then transferring the data to a bulk endpoint buffer.

Chapter 7. EZ-USB FX Slave FIFOs

7.1 Introduction

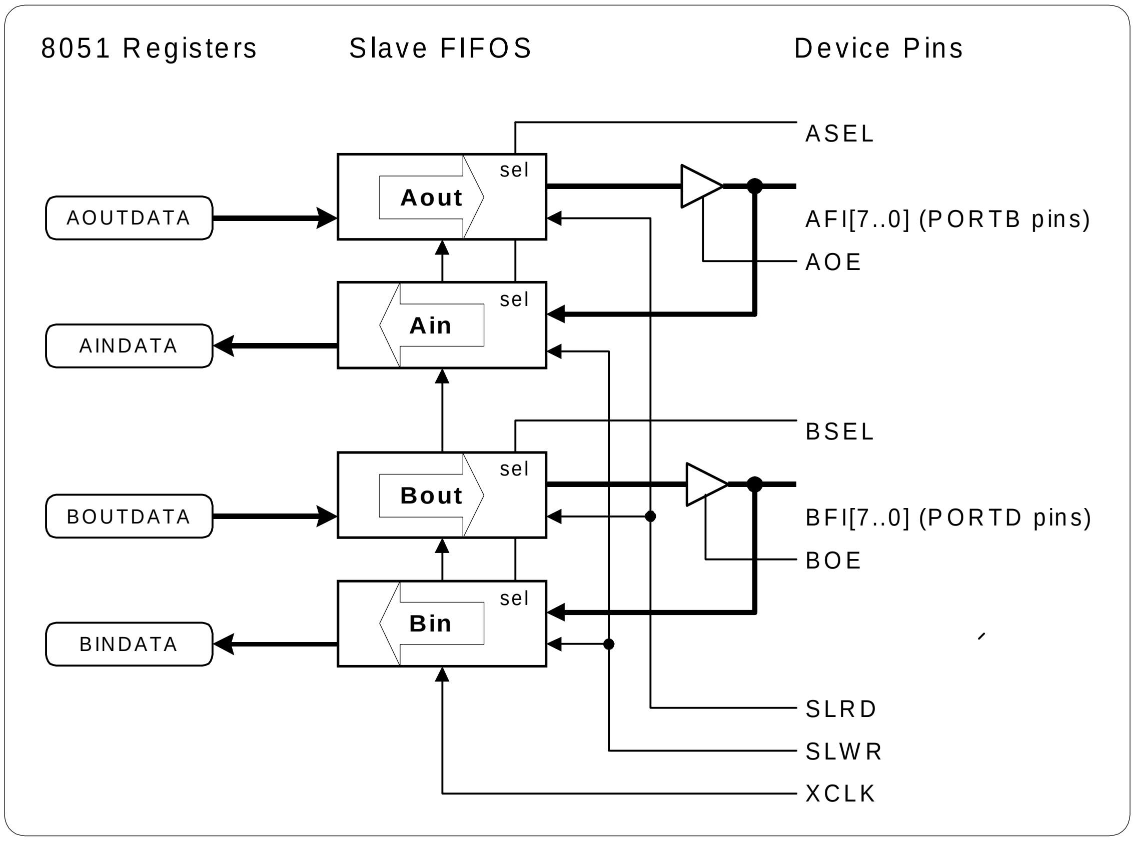


Figure 7-1. The Four 64-Byte Slave FIFOs Configured for 16-Bit Mode

Figure 7-1 illustrates the four slave FIFOs in EZ-USB FX. The slave FIFOs, each 64 bytes in length, serve as general-purpose buffers between external logic and 8051 registers. They are called "slave" FIFOs because the outside logic can supply the timing signals. The FIFOs are grouped into identical *A* and *B* pairs, each pair having an IN and OUT FIFO. *Figure 7-1* illustrates *16-bit mode*, in which outside logic can read or write data either independently or simultaneously from/to the two 8-bit FIFOs.

7.1.1 8051 FIFO Access

The 8051 accesses the slave FIFOs using four registers in XDATA memory: AOUTDATA, AIN-DATA, BOUTDATA, and BINDATA. These registers can be read and written by 8051 code (using the MOVX instruction), or they can serve as sources and destinations for the DMA mechanism, built into the EZ-USB FX. Section 7.2. *"Slave FIFO Register Descriptions"* describes these registers in detail.

7.1.2 External Logic FIFO Access

External logic can access the slave FIFOs either asynchronously or synchronously:

- Asynchronous—SLRD and SLWR pins are read and write strobes.
- Synchronous—SLRD and SLWR pins are enables for the XCLK clock pin.

External logic accesses the FIFOs through two 8-bit data buses, which double as general-purpose I/O ports PORTB and PORTD. When used for FIFO access, the data buses are bi-directional, with output drivers controlled by the AOE and BOE pins.

Two FIFO select signals, ASEL and BSEL, are used to select the FIFO in two modes that use both FIFOs: 8-bit mode, and double-byte mode. These modes and the role of the ASEL and BSEL pins are illustrated in *Figure 7-2* and *Figure 7-3*.

7.1.3 ASEL, BSEL in 8-Bit Mode

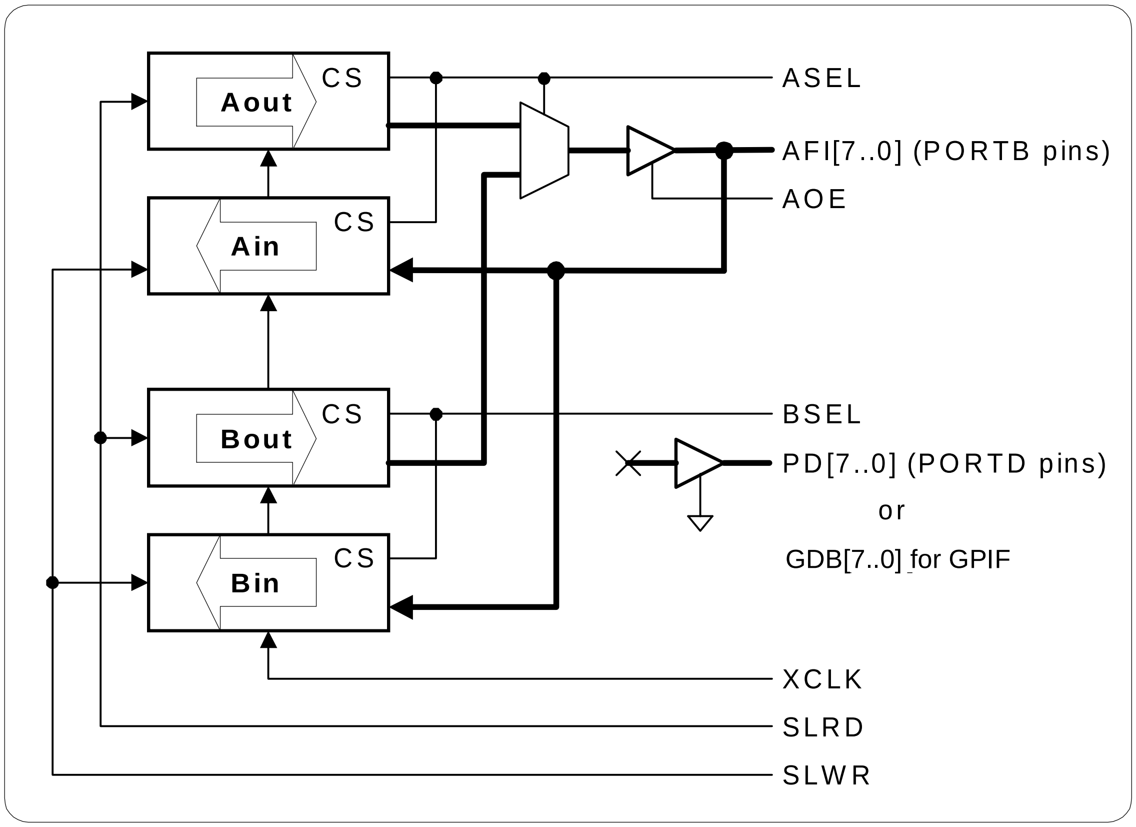


Figure 7-2. Slave FIFOs in 8-Bit Mode

In 8-bit mode, data from the PORTB pins can be read/written from either the A or B FIFOs, as selected by the ASEL and BSEL pins. In 8-bit mode, the input/output port or GPIF data is available on the PORTD pins.

7.1.4 ASEL, BSEL in Double-Byte Mode

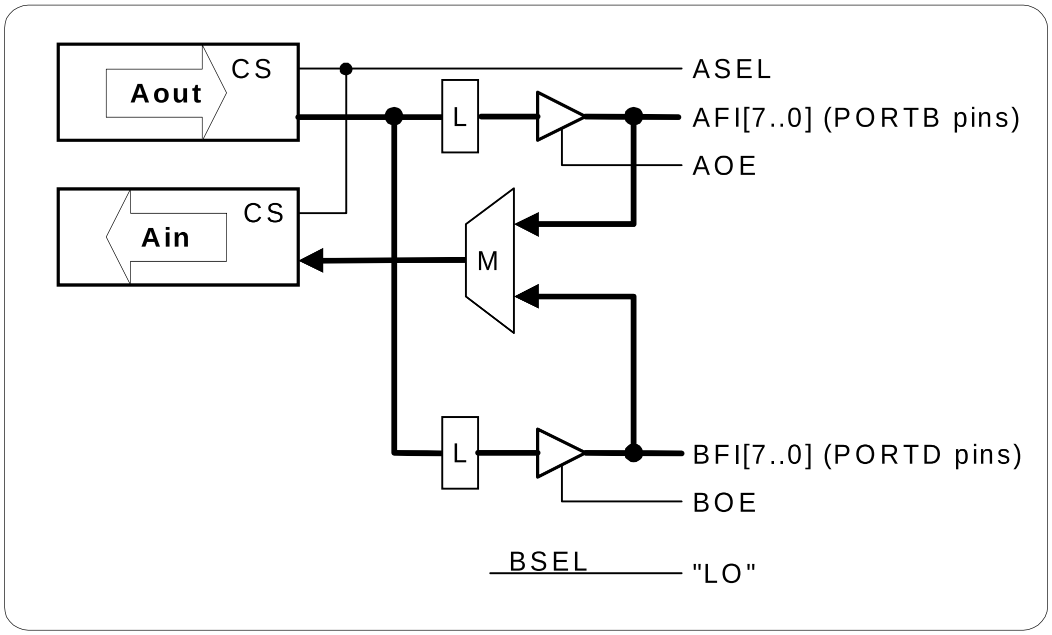


Figure 7-3. Double-Byte Mode with A-FIFO Selected

Figure 7-3 illustrates double-byte mode. For this illustration, signals ASEL, BSEL, AOE, and BOE are programmed to be active high polarity. In this mode, the ASEL and BSEL pins determine which of the FIFO pairs, A or B, accept or transmit interleaved byte data, as follows:

- The IN FIFO receives 16-bit data as double bytes, interleaved from PORTB first and then from PORTD. The data interleaving is automatic, with two bytes written to the FIFO per external write strobe. The interleave order input from the ports is the same whether the destination FIFO is A-IN or B-IN.
- The OUT FIFO transmits two bytes, the first to PORTB and the second to PORTD, per external read strobe. The interleave order output to the ports is the same whether the source FIFO is A-OUT or B-OUT.

7.1.5 FIFO Registers

The 8051 accesses a variety of control and status registers to control the slave FIFO operation. These registers perform the following functions:

- Data registers give the 8051/DMA access to IN FIFO and OUT FIFO data.

- Byte Count registers indicate the number of bytes in each FIFO.
- Flag bits indicate FIFO full, empty, and a programmable level.
- Mode bits control the various FIFO modes.

7.1.6 FIFO Flags and Interrupts

The slave FIFOs have two independent sets of flags, internal and external. The 8051 can directly test the internal flags, or these flags can automatically create 8051 interrupts using INT4. The external flags are available as device pins, to be used by external logic. Two independent sets of programmable flags allow different FIFO *fullness* levels to be set for internal and external use.

The internal FIFO flags are connected to the 8051 interrupt system using INT4. To streamline the 8051 code that deals with these interrupts, the 8051 INT4 vector locations have a special property when a mode bit called “AV4EN” (Autovector 4 Enable) is set. Referring toTable 7-1, when a FIFO flag interrupt occurs with AV4EN=1, internal logic replaces the third byte of the jump instruction at location 0x55 with a different address for each FIFO interrupt source.

Table 7-1. Autovector for INT4*

8051 Addr	Instruction	Notes
0x53	LJMP	Loc 53-55 are the INT4 Interrupt Vector.
0x54	AddrH	
0x55 *	AddrL	**EZ-USB FX logic replaces this byte when AV4EN=1.**

* (Table 7-2 shows bytes inserted at address 55H)

To set up autovectoring, the user places an LJMP instruction at location 0x53. This jumps to a table of instructions that jump to the various FIFO ISRs. Then, every FIFO interrupt automatically vectors to the individual interrupt service routines for the particular FIFO flags. The autovector mechanism saves the 8051 from having to check for the source of each interrupt shared on INT4.

The FIFO interrupts that share INT4 are shown in Table 7-2. The last three are not FIFO-related, and are described in other chapters. The bytes inserted by the EZ-USB FX logic (the low-address byte of the LJMP instruction) are separated by four to allow four bytes per LJMP instruction in the jump table. (An 8051 LJMP instruction requires three bytes).

Note that the bytes inserted for the INT4 autovector start at 0x80, rather than 0x00. This is because another EZ-USB FX autovector, for INT2 (used for all USB interrupts), uses jump table offsets from 0x00 to 0x57. The autovector jump table must start on a page boundary (8051 address XX00). Therefore, separating the two groups of jumps allows a single page of 8051 memory to be used for both INT2 and INT4 jump tables. The INT2 jump table can start at 0x00, and the INT4 jump table can start at 0x80, both in the same page.

Table 7-2. INT4 Autovectors

IVEC4 Value	Byte Inserted at 0x55	Source	Meaning
0x40	0x80	AINPF	A-IN FIFO Programmable Flag
0x44	0x84	BINPF	B-IN FIFO Programmable Flag
0x48	0x88	AOUTPF	A-OUT FIFO Programmable Flag
0x4C	0x8C	BOUTPF	B-OUT FIFO Programmable Flag
0x50	0x90	AINEF	A-IN FIFO Empty Flag
0x54	0x94	BINEF	B-IN FIFO Empty Flag
0x58	0x98	AOUTEF	A-OUT FIFO Empty Flag
0x5C	0x9C	BOUTEF	B-OUT FIFO Empty Flag
0x60	0xA0	AINFF	A-IN FIFO Full Flag
0x64	0xA4	BINFF	B-IN FIFO Full Flag
0x68	0xA8	AOUTFF	A-OUT FIFO Full Flag
0x6C	0xAC	BOUTFF	B-OUT FIFO Full Flag
0x70	0xB0	GPIF-DONE	See *Chapter 8. "General Programmable Interface (GPIF)"*
0x74	0xB4	GPIFWF	See *Chapter 8. "General Programmable Interface (GPIF)"*
0x78	0xB8	DMADONE	See *Chapter 8. "General Programmable Interface (GPIF)"*

The first column shows the value in the IVEC4 Register for each FIFO interrupt source.

If two or more INT4 interrupt requests occur simultaneously, they are serviced in the order shown in Table 7-2, with AINPF having the highest priority and DMADONE the lowest. Interrupt requests remain pending while a higher level interrupt is serviced.

7.2 Slave FIFO Register Descriptions

In the following FIFO diagrams, the 8051-access side is on the left, and the external pins are on the right.

7.2.1 FIFO A Read Data

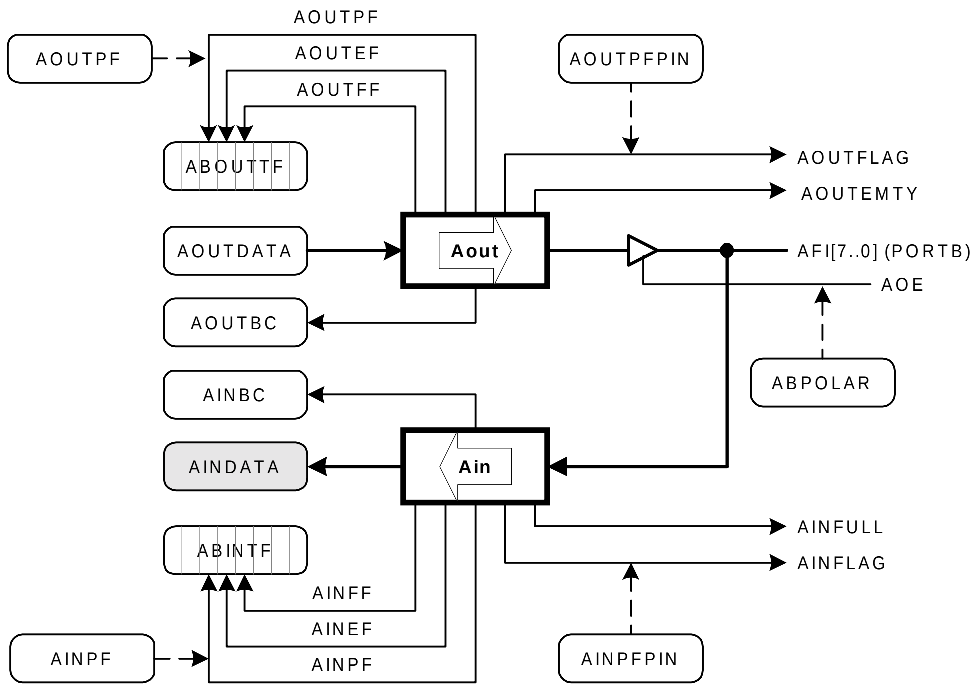


Figure 7-4. AINDATA's Role in the FIFO A Register

AINDATA			FIFO A Read Data				7800
b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

Figure 7-5. FIFO A Read Data

Each time the 8051 reads a byte from this register, the A-IN FIFO advances to the next byte in the FIFO, and the AINBC (byte count) decrements. Reading this register when there is one byte remaining in the A-IN FIFO sets the A-IN FIFO Empty Flag (AINEF, in ABINCS.4). This causes an interrupt request on INT4 (Table 7-2). Reading this register when the A-IN FIFO is empty returns indeterminate data and has no effect on the FIFO flags byte counts.

7.2.2 A-IN FIFO Byte Count

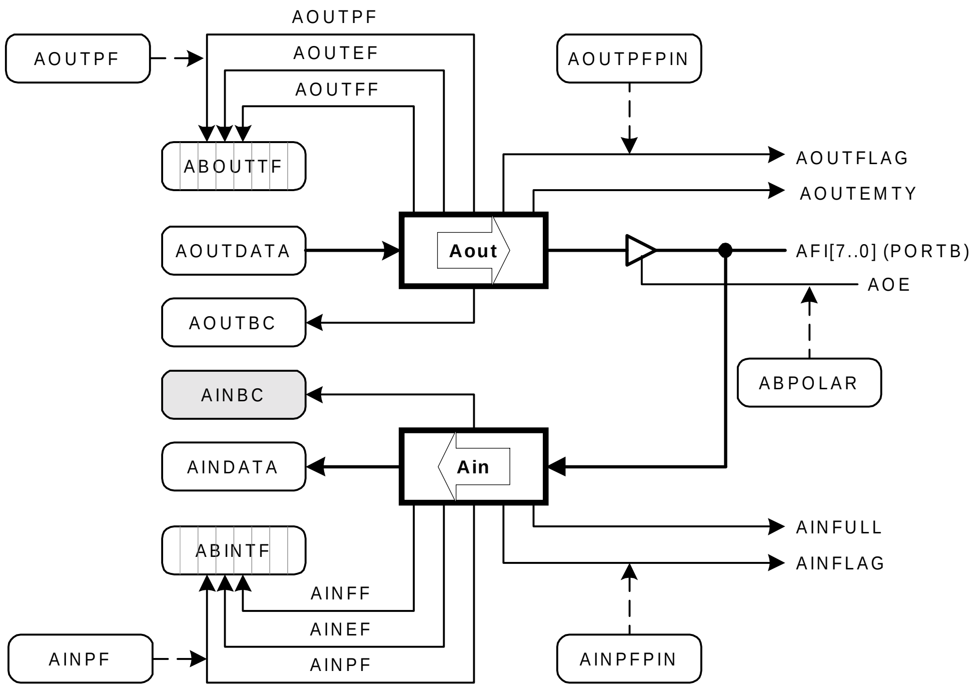


Figure 7-6. AINBC's Role in the FIFO A Register

AINBC **A-IN FIFO Byte Count** **7801**

b7	b6	b5	b4	b3	b2	b1	b0
0	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Figure 7-7. A-IN FIFO Byte Count

This count reflects the number of bytes remaining in the A-IN FIFO. Valid byte counts are 0-64. Every byte written by outside logic increments this count, and every 8051 read of AINDATA decrements this count. If AINBC is zero, an 8051 read of AINDATA returns indeterminate data and results in the byte count in AINBC remaining at zero. Data bytes should never be written to the FIFO from outside logic when the AINFULL flag is HI.

7.2.3 A-IN FIFO Programmable Flag

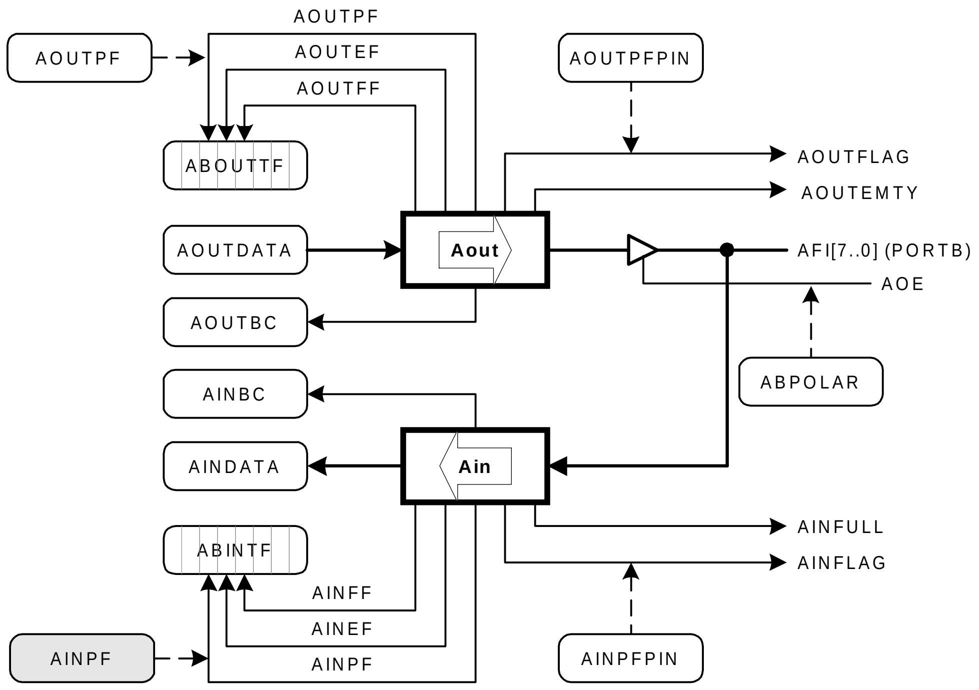


Figure 7-8. AINPF's Role in the FIFO A Register

AINPF			A-IN FIFO Programmable Flag				7802
b7	b6	b5	b4	b3	b2	b1	b0
LTGT	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	1	0	0	0	0	0

Figure 7-9. A-IN FIFO Programmable Flag

This register controls the sense and value for the internal A-IN FIFO programmable flag. This flag is testable by the 8051.

*Another register, AINPF**PIN** (*Section 7.2.3.3. *"A-IN FIFO Pin Programmable Flag") corresponds to an A-IN FIFO programmable flag that drives an output pin, not an internal flag bit.*

The 8051 tests the internal FIFO programmable flag by reading the AINPF Bit in ABINCS.5. This flag can also be enabled to cause an interrupt request on INT4(Table 7-2) when it makes a zero-to-one transition. The default value of the AINPF Register indicates half-empty.

Bit 7: **LTGT** *Less-than, Greater-than flag*

If LTGT=0, the AINPF flag goes true, if the number of bytes in the FIFO is less than or equal to the programmed value in D[6..0].

If LTGT=1, the AINPF flag goes true, if the number of bytes in the FIFO is greater than or equal to the value programmed into D[6..0].

Bit 6-0: **PFVAL** *Programmable Flag Value*

This value, along with the LTGT Bit, determines when the programmable flag for the A-IN FIFO becomes active. The 8051 programs this register to indicate various degrees of A-IN FIFO *fullness* to suit the application. The following two sections show the interaction of the LTGT Bit and the programmed value for two cases, a filling FIFO and an emptying FIFO.

7.2.3.1 Filling FIFO

When a FIFO is filling with data, it is useful to generate an 8051 interrupt when a programmed level is reached. Because the interrupt request is triggered on a zero-to-one transition of the programmable flag AINPF, the LTGT Bit should be set to “1.” In Table 7-3, D[6..0] is set to 48 bytes and the LTGT Bit is set to “1.” When the FIFO reaches 48 bytes, the AINPF Bit goes high, generating an interrupt request.

Table 7-3. Filling FIFO

LTGT	D[6..0]	Bytes in FIFO	AINPF
1	48	45	0
1	48	46	0
1	48	47	0
1	48	48	1
1	48	49	1
1	48	50	1

7.2.3.2 Emptying FIFO

When a FIFO is being emptied of data, the LTGT Bit should be set to “0,” so the zero-to-one transition of the AINPF flag (and therefore the interrupt request) occurs when the byte count descends to below the programmed value.In Table 7-4, D[6..0] is set to 48 bytes, and when the FIFO goes from 49 bytes to 48 bytes, the AINPF Bit goes high, generating an interrupt request.

Table 7-4.   Emptying FIFO

LTGT	D[6..0]	Bytes in FIFO	AINPF
0	48	51	0
0	48	50	0
0	48	49	0
0	48	48	1
0	48	47	1
0	48	46	1

7.2.3.3 A-IN FIFO Pin Programmable Flag

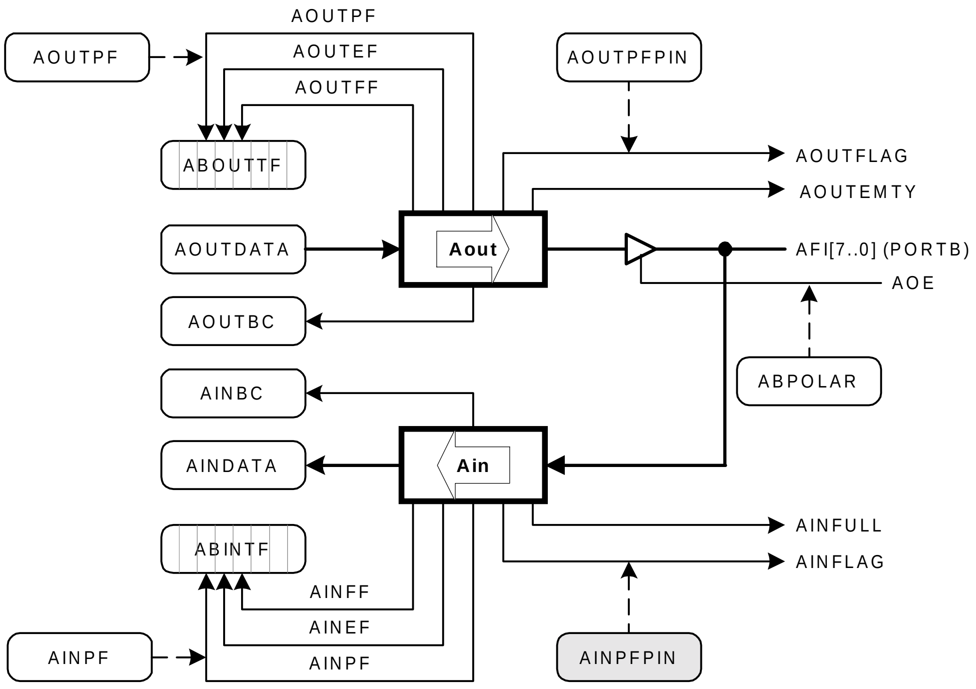


Figure 7-10. AINPFPIN's Role in the FIFO A Register

AINPFPIN **A-IN FIFO Pin Programmable Flag** **7803**

b7	b6	b5	b4	b3	b2	b1	b0
LTGT	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 7-11. A-IN FIFO Pin Programmable Flag

This register controls the sense and value for the A-IN FIFO Programmable Flag that appears on the AINFLAG *pin*. This pin is used by external logic to regulate external writes to the A-IN FIFO. The AINPFPIN Register is programmed with the same data format as the previous register, AINPF. The only operational difference is that the flag drives a hardware pin rather than existing as an internal register bit.

Having separate programmable flags allows the 8051 and external logic to have independent gauges of FIFO fullness. It may be desirable, for example, for one side (8051 or external logic) to have advance notice over the other side about a FIFO becoming full or empty.

The default value of the AINPFPIN Register indicates empty.

7.2.4 B-IN FIFO Read Data

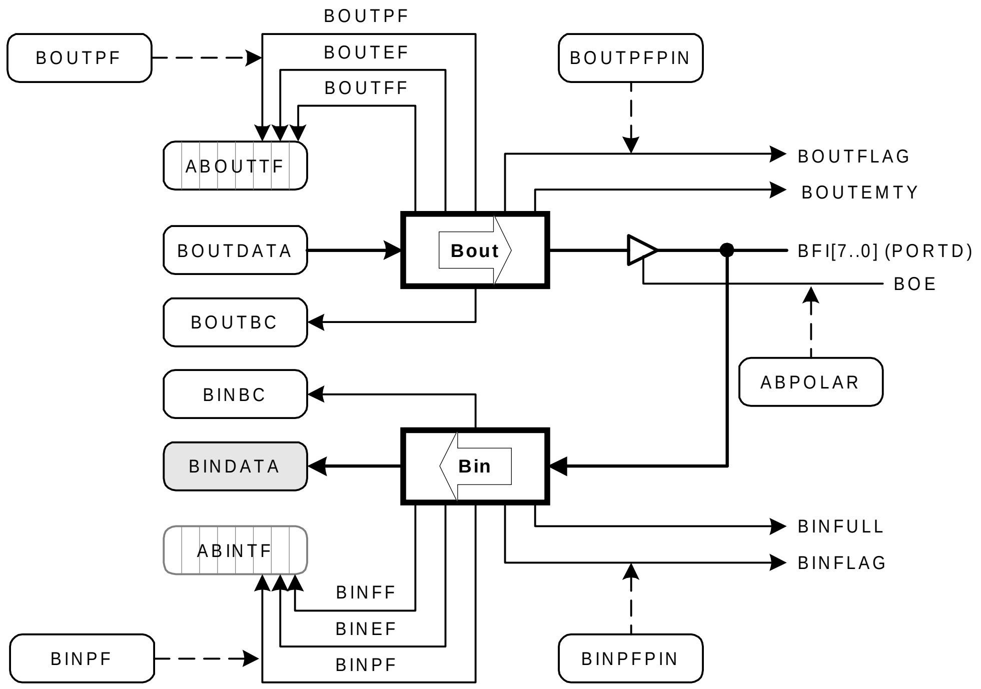


Figure 7-12. BINDATA's Role in the FIFO B Register

BINDATA	B-IN FIFO Read Data	7805

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

Figure 7-13. B-IN FIFO Read Data

Each time the 8051 reads a byte from this register, the B-IN FIFO advances to the next byte in the FIFO, and the BINBC (byte count) decrements. Reading this register when there is one byte remaining in the FIFO sets the B-IN FIFO Empty Flag (BINEF, in ABINCS.1), which causes an INT4 request. Reading this register when the B-IN FIFO is empty returns indeterminate data and has no effect on the FIFO flags or byte count.

7.2.5 B-IN FIFO Byte Count

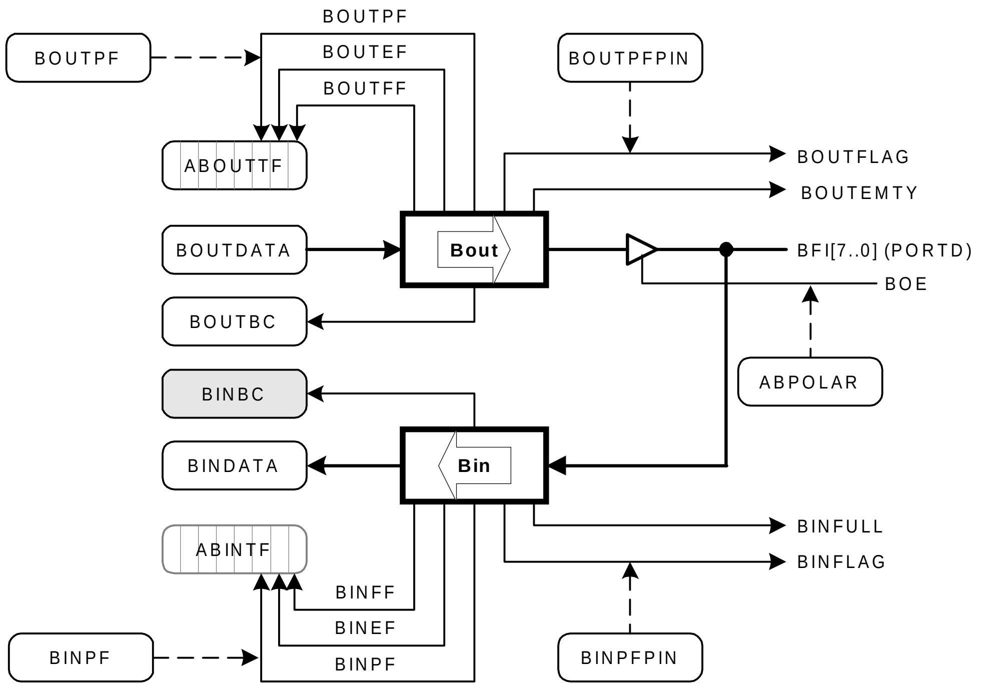


Figure 7-14. BINBC's Role in the FIFO B Register

BINBC			**B-IN FIFO Byte Count**				**7806**
b7	b6	b5	b4	b3	b2	b1	b0
0	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Figure 7-15. B-IN FIFO Byte Count

This count reflects the number of bytes remaining in the B-IN FIFO. Valid byte counts are 0-64. Every byte written by outside logic increments this count, and every 8051 read of BINDATA decrements this count. If BINBC is zero, an 8051 read of BINDATA returns indeterminate data. This results in the byte count in BINBC to remain at zero. Data bytes should never be written to the FIFO from outside logic when the BINFULL flag is HI.

7.2.6 B-IN FIFO Programmable Flag

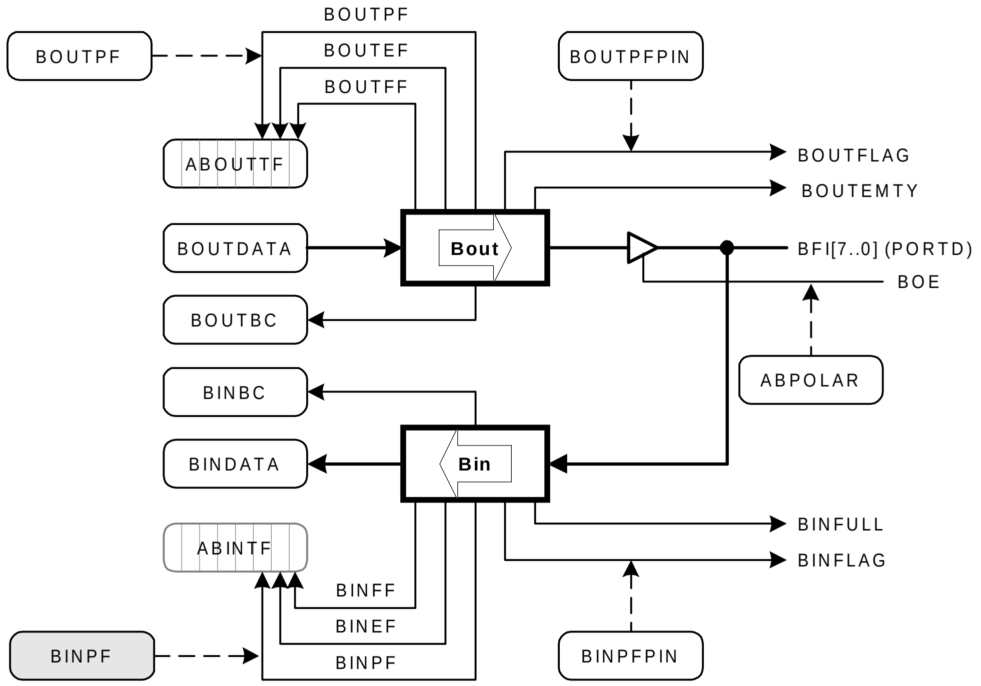


Figure 7-16. BINPF's Role in the FIFO B Register

BINPF	B-IN FIFO Programmable Flag	7807

b7	b6	b5	b4	b3	b2	b1	b0
LTGT	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	1	0	0	0	0	0

Figure 7-17. B-IN FIFO Programmable Flag

This register controls the sense and value for the internal B-IN FIFO programmable flag.

*Another register, BINPF**PIN** (Section 7.2.7. "B-IN FIFO Pin Programmable Flag") corresponds to a B-IN FIFO programmable flag that drives an output pin, not an internal flag bit.*

The 8051 tests the internal FIFO programmable flag by reading the BINPF Bit in ABINCS.2. This flag can also be enabled to cause an interrupt request on INT4(Table 7-2) when it makes a zero-to-one transition. The default value of the BINPF Register indicates half-empty.

Bit 7: **LTGT** *Less-than, Greater-than flag*

If LTGT=0, the BINPF flag goes true if the number of bytes in the FIFO is less than or equal to the programmed value in D[6..0].

If LTGT=1, the BINPF flag goes true if the number of bytes in the FIFO is greater than or equal to the value programmed into D[6..0].

Bit 6-0: **PFVAL** *Programmable Flag Value*

This value, along with the LTGT Bit, determines when the programmable flag for the B-FIFO becomes active. The 8051 programs this register to indicate various degrees of B-FIFO *full-ness* to suit the application. The following two sections in this chapter show the interaction of the LTGT Bit and the programmed value for two cases, a filling FIFO and an emptying FIFO.

7.2.6.1 Filling FIFO

When a FIFO is filling with data, it is useful to generate an 8051 interrupt when a programmed level is reached. Because the interrupt request is triggered on a zero-to-one transition of the programmable flag BINPF, the LTGT Bit should be set to "1." In Table 7-5, D[6..0] is set to 48 bytes and the LTGT Bit is set to "1." When the FIFO reaches 48 bytes, the BINPF Bit goes high, generating an interrupt request.

Table 7-5. Filling FIFO

LTGT	D[6..0]	Bytes in FIFO	BINPF
1	48	45	0
1	48	46	0
1	48	47	0
1	48	48	1
1	48	49	1
1	48	50	1

7.2.6.2 Emptying FIFO

When a FIFO is being emptied of data, the LTGT Bit should be set to "0," so the zero-to-one transition of the BINPF flag (therefore, the interrupt request) occurs when the byte count descends to below the programmed value. In Table 7-6, D[6..0] is set to 48 bytes, and when the FIFO goes from 49 bytes to 48 bytes, the BINPF Bit goes high, generating an interrupt request.

Table 7-6. Emptying FIFO

LTGT	D[6..0]	Bytes in FIFO	BINPF
0	48	51	0
0	48	50	0
0	48	49	0
0	48	48	1
0	48	47	1
0	48	46	1

7.2.7 B-IN FIFO Pin Programmable Flag

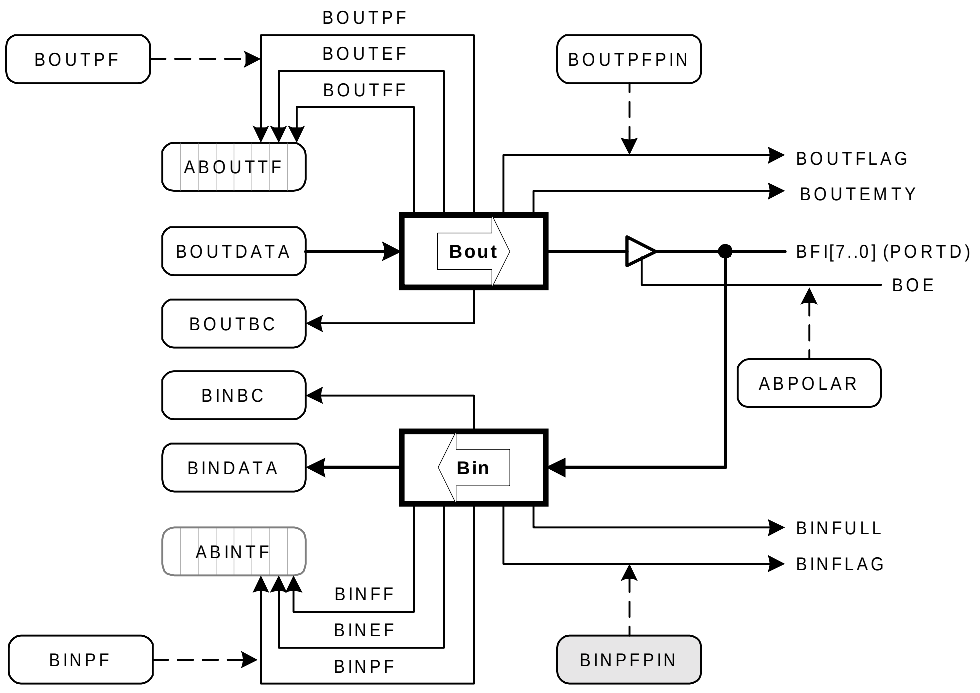


Figure 7-18. BINPFPIN's Role in the FIFO B Register

BINPFPIN **B-IN FIFO Pin Programmable Flag** **7808**

b7	b6	b5	b4	b3	b2	b1	b0
LTGT	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 7-19. B-IN FIFO Pin Programmable Flag

This register controls the sense and value for the B-IN FIFO Programmable Flag that appears on the BINFLAG *pin*. This pin is used by external logic to regulate external writes to the B-IN FIFO. The BINPFPIN Register is programmed with the same data format as the previous register, BINPF. The only operational difference is that the flag drives a hardware pin rather than existing as an internal register bit.

Having separate programmable flags allows the 8051 and external logic to have independent gauges of FIFO fullness. It may be desirable, for example, for one side (8051 or external logic) to have advance notice over the other side about a FIFO becoming full or empty.

The default value of the BINPFPIN Register indicates empty.

7.2.8 Input FIFOs A/B Toggle CTL and Flags

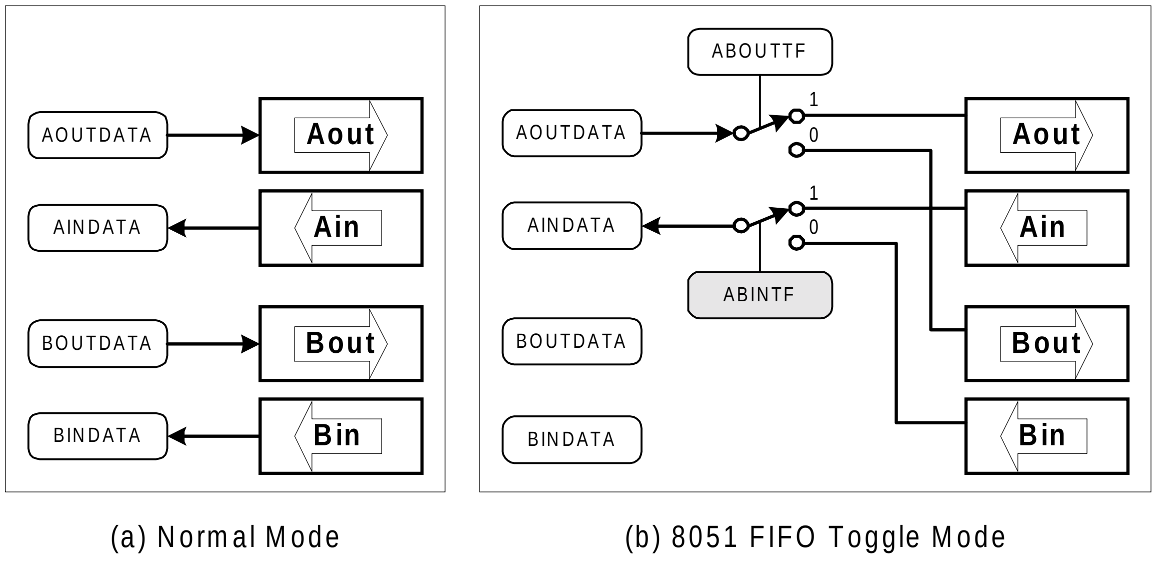


Figure 7-20. 8051 FIFO Toggle Mode vs. Normal Mode Diagram

ABINCS **Input FIFOs A/B Toggle CTL and Flags** **780A**

b7	b6	b5	b4	b3	b2	b1	b0
INTOG	**INSEL**	**AINPF**	**AINEF**	**AINFF**	**BINPF**	**BINEF**	**BINFF**
R/W	R/W	R	R	R	R	R	R
0	1	1	1	0	1	1	0

Figure 7-21. Input FIFOs A/B Toggle CTL and Flags

Bit 7: **INTOG** *Enable Input FIFO Toggle*

A special FIFO toggle mode switches automatically between the A-IN and B-IN FIFOs each time the 8051 reads data from the AINDATA Register. The toggle mechanism works only for programmed 8051 transfers, not DMA transfers.

When INTOG=0, the A-IN and B-IN FIFOs operate in Normal Mode, as illustrated in diagram (a) in *Figure 7-20* on the previous page.

When INTOG=1, the FIFOs operate in Toggle Mode, as illustrated in diagram (b) in *Figure 7-20*. The selected FIFO switches between the A-IN and B-IN FIFOs after every 8051 read of the AINDATA Register. The selected FIFO is indicated by the INSEL Bit (Bit 6).

Bit 6: **INSEL** *Input Toggle Select*

If INTOG=1 when enabling the Toggle Mode:

- This bit selects IN FIFO A or B when the 8051 reads the AINDATA Register. When INSEL=0, the B-IN FIFO is read. When INSEL=1, the A-IN FIFO is read. When INTOG=1, this bit complements automatically (toggles) after every 8051 read of AINDATA. This has the effect of automatically toggling between the A-IN and B-IN FIFOs for successive reads of AINDATA.
- The 8051 can directly write this bit to select *manually* the A-IN or B-IN FIFO. More commonly, the Toggle Mode will be used since it allows 16-bit transfers using the 8051 without requiring the 8051 to switch between the FIFOs.

If INTOG=0 when enabling the Toggle Mode:

- The INSEL Bit has no effect.

Bit 5: **AINPF** *A-IN FIFO Programmable Flag*

AINPF=1 when the A-IN FIFO byte count satisfies the conditions programmed into the programmable FIFO flag register AINPF; otherwise, AINPF=0. A zero-to-one transition of this flag sets the interrupt request bit AINPFIR.

Bit 4: AINEF *A-IN FIFO Empty Flag*

AINEF=1 when the A-IN FIFO is empty; otherwise, AINEF=0. The flag goes active after the 8051 or DMA system reads the last byte in the A-IN FIFO. A zero-to-one transition of this flag sets the interrupt request bit AINEFIR.

AINFF=1 when the A-IN FIFO is full; otherwise, AINFF=0. The flag goes active after external logic writes the 64th byte into the A-IN FIFO. A zero-to-one transition of this flag sets the interrupt request bit AINFFIR.

Bit 2: BINPF *B-IN FIFO Programmable Flag*

BINPF=1 when the number of bytes in the B-IN FIFO satisfies the requirements programmed into the BINPF Register; otherwise, BINPF=0. A zero-to-one transition of this flag sets the interrupt request bit BINPFIR.

Bit 1: BINEF *B-IN FIFO Empty Flag*

BINEF=1 when the B-IN FIFO is empty; otherwise, BINEF=0. The flag goes active after the 8051 or DMA system reads the last byte in the B-IN FIFO. A zero-to-one transition of this flag sets the interrupt request bit BINEFIR.

Bit 0: BINFF *B-IN FIFO Full Flag*

BINFF=1 when the B-IN FIFO is full. The flag goes valid after external logic writes the 64th byte into the B-IN FIFO. A zero-to-one transition of this flag sets the interrupt request bit BINFFIR.

7.2.9 Input FIFOs A/B Interrupt Enables

ABINIE **Input FIFOs A/B Interrupt Enables** **780B**

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**AINPFIE**	**AINEFIE**	**AINFFIE**	**BINPFIE**	**BINEFIE**	**BINFFIE**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 7-22. Input FIFOs A/B Interrupt Enables

Bit 5: AINPFIE *A-IN FIFO Programmable Flag Interrupt Enable*

The 8051 sets AINPFIE=1 to enable an INT4 interrupt when the AINPFIR interrupt request bit makes a zero-to-one transition. This transition indicates that the A-IN FIFO byte count has satisfied the *fullness* level programmed into the programmable FIFO flag register AINPF. The

8051 clears AINPFIE to prevent the associated interrupt request bit from causing an INT4 interrupt.

Bit 4: **AINEFIE** *A-IN FIFO Empty Interrupt Enable*

The 8051 sets AINEFIE=1 to enable an INT4 interrupt when the AINEFIR interrupt request bit makes a zero-to-one transition. This indicates an A-IN FIFO byte count of zero. The 8051 clears AINEFIE to prevent the associated interrupt request bit from causing an INT4 interrupt.

Bit 3: **AINFFIE** *A-IN FIFO Bull Interrupt Enable*

The 8051 sets AINFFIE=1 to enable an INT4 interrupt when the AINFFIR interrupt request bit makes a zero-to-one transition. This indicates an A-IN FIFO byte count of 64. The 8051 clears AINFFIE to prevent the associated interrupt request bit from causing an INT4 interrupt.

Bit 2: **BINPFIE** *B-IN FIFO Programmable Flag Interrupt Enable*

The 8051 sets BINPFIE=1 to enable an INT4 interrupt when the BINPFIR interrupt request bit makes a zero-to-one transition. This transition indicates that the B-IN FIFO byte count has satisfied the *fullness* level programmed into the programmable FIFO flag register BINPF. The 8051 clears BINPFIE to prevent the associated interrupt request bit from causing an INT4 interrupt.

Bit 1: **BINEFIE** *B-IN FIFO Empty Interrupt Enable*

The 8051 sets BINEFIE=1 to enable an INT4 interrupt when the BINEFIR interrupt request bit makes a zero-to-one transition. This indicates a B-IN FIFO byte count of zero. The 8051 clears BINEFIE to prevent the associated interrupt request bit from causing an INT4 interrupt.

Bit 0: **BINFFIE** *B-IN FIFO Full Interrupt Enable*

The 8051 sets BINFFIE=1 to enable an INT4 interrupt when the BINFFIR interrupt request bit makes a zero-to-one transition. This indicates a B-IN FIFO byte count of 64. The 8051 clears BINFFIE to prevent the associated interrupt request bit from causing an INT4 interrupt.

7.2.10 Input FIFOs A/B Interrupt Requests

ABINIRQ	Input FIFOs A/B Interrupt Requests	780C

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**AINPFIR**	**AINEFIR**	**AINFFIR**	**BINPFIR**	**BINEFIR**	**BINFFIR**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 7-23. Input FIFOs A/B Interrupt Requests

Bit 5: AINPFIR *A-IN FIFO Programmable Flag Interrupt Request*

AINPFIR makes a zero-to-one transition when the A-IN FIFO byte count satisfies the required condition programmed into the programmable FIFO flag register AINPF. If enabled by the AINPFIE Bit, this transition causes an INT4 interrupt request.

The 8051 writes a “1” to this bit to clear the interrupt request. The 8051 should clear the 8051 INT4 Bit (EXIF.6) before clearing the AINPFIR Bit in the interrupt service routine to guarantee that pending INT4 interrupts will be recognized.

Bit 4: AINEFIR *A-IN FIFO Empty Interrupt Request*

AINEFIR makes a zero-to-one transition when the A-IN FIFO byte count reaches zero (FIFO empty). If enabled by the AINEFIE Bit, this transition causes an INT4 interrupt request.

The 8051 writes “1” to this bit to clear the interrupt request. The 8051 should clear the 8051 INT4 Bit (EXIF.6) before clearing the AINEFIR Bit in the interrupt service routine to guarantee that pending INT4 interrupts will be recognized.

Bit 3: AINFFIR *A-IN FIFO Full Interrupt Request*

AINFFIR makes a zero-to-one transition when the A-IN FIFO byte count reaches 64 (FIFO full). If enabled by the AINFFIE Bit, this transition causes an INT4 interrupt request.

The 8051 writes a “1” to this bit to clear the interrupt request. The 8051 should clear the 8051 INT4 Bit (EXIF.6) before clearing the AINFFIR Bit in the interrupt service routine to guarantee that pending INT4 interrupts will be recognized.

Bit 2: BINPFIR *B-IN FIFO Programmable Flag Interrupt Request*

BINPFIR makes a zero-to-one transition when the B-IN FIFO byte count satisfies the required condition programmed into the programmable FIFO flag register BINPF. If enabled by the BINPFIE Bit, this transition causes an INT4 interrupt request.

The 8051 writes a “1” to this bit to clear the interrupt request. The 8051 should clear the 8051 INT4 Bit (EXIF.6) before clearing the BINPFIR Bit in the interrupt service routine to guarantee that pending INT4 interrupts will be recognized.

Bit 1: **BINEFIR** *B-IN FIFO Empty Interrupt Request*

BINEFIR makes a zero-to-one transition when the B-IN FIFO byte count reaches zero (FIFO empty). If enabled by the BINEFIE Bit, this transition causes an INT4 interrupt request.

The 8051 writes a “1” to this bit to clear the interrupt request. The 8051 should clear the 8051 INT4 Bit (EXIF.6) before clearing the BINEFIR Bit in the interrupt service routine to guarantee that pending INT4 interrupts will be recognized.

Bit 0: **BINFFIR** *B-IN FIFO Full Interrupt Request*

BINFFIR makes a zero-to-one transition when the B-IN FIFO byte count reaches 64 (FIFO full). If enabled by the BINFFIE Bit, this transition causes an INT4 interrupt request.

The 8051 writes a “1” to this bit to clear the interrupt request. The 8051 should clear the 8051 INT4 Bit (EXIF.6) before clearing the BINFFIR Bit in the interrupt service routine to guarantee that pending INT4 interrupts will be recognized.

7.2.11 FIFO A Write Data

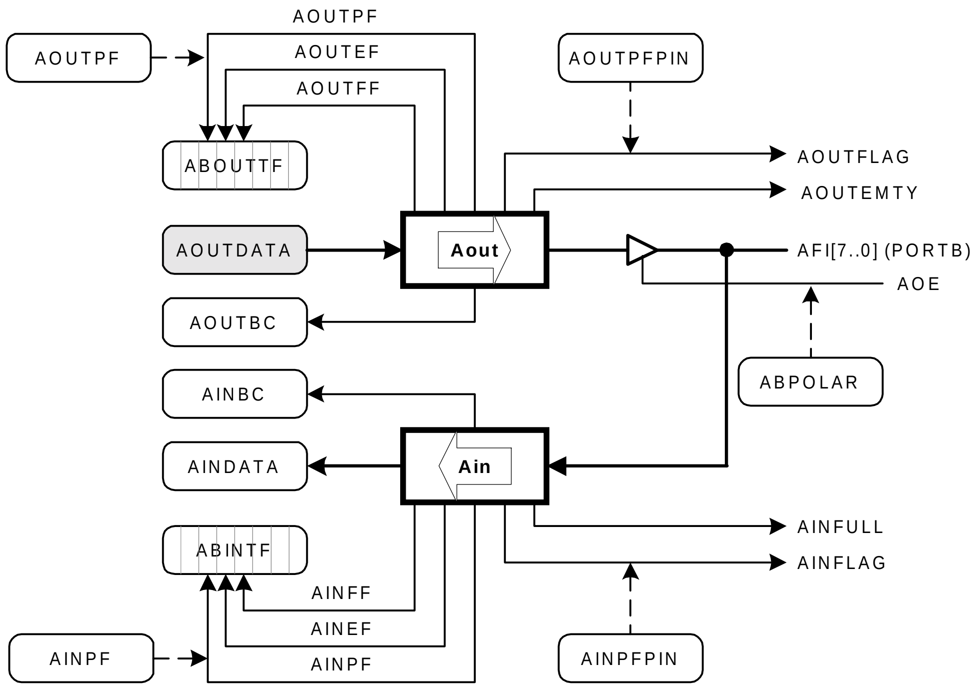


Figure 7-24. AOUTDATA's Role in the FIFO A Register

AOUTDATA **FIFO A Write Data** **780E**

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
W	W	W	W	W	W	W	W
x	x	x	x	x	x	x	x

Figure 7-25. FIFO A Write Data

Each time the 8051/DMA writes a byte to this register, the A-OUT FIFO advances to the next open position in the FIFO and the AOUTBC (byte count) increments. Writing this register when there are 63 bytes remaining in the A-OUT FIFO sets the A-FIFO Full Flag (AOUTFF, in ABOUTCS.3), which causes an INT4 request. Writing this register when the A-OUT FIFO is full (64 bytes) does not update the FIFO or byte count, and has no effect on the FIFO flags or byte count.

7.2.11.1 A-OUT FIFO Byte Count

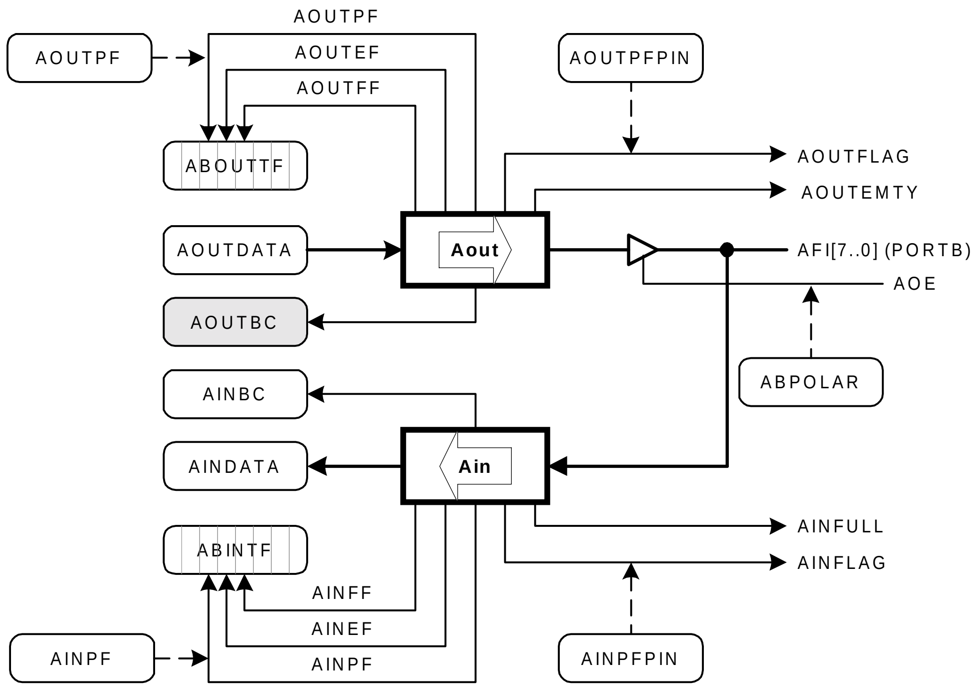


Figure 7-26. AOUTBC's Role in the FIFO A Register

AOUTBC			A-OUT FIFO Byte Count				780F
b7	b6	b5	b4	b3	b2	b1	b0
0	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Figure 7-27. Input FIFOs A/B Interrupt Requests

This count reflects the number of bytes remaining in the A-OUT FIFO. Valid byte counts are 0-64. When non-zero, every byte read by outside logic decrements this count, and every 8051 write of AOUTDATA increments this count. If AOUTBC is zero, reading a data byte by outside logic returns indeterminate data and results in the byte count in AOUTBC remaining at zero.

7.2.12 A-OUT FIFO Programmable Flag

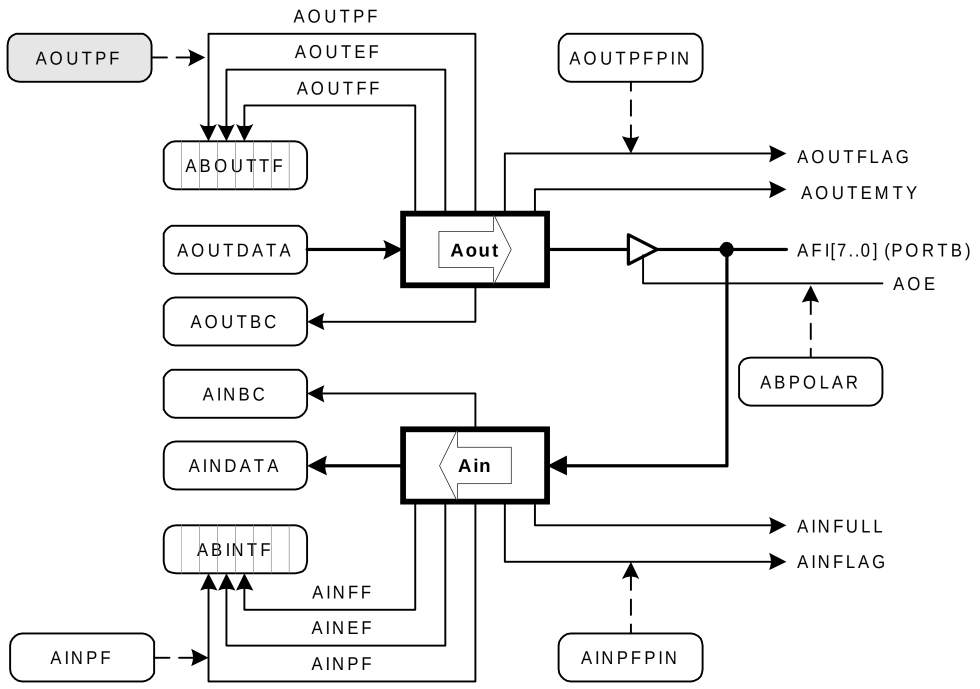


Figure 7-28. AOUTPF's Role in the FIFO A Register

AOUTPF **A-OUT FIFO Programmable Flag** **7810**

b7	b6	b5	b4	b3	b2	b1	b0
LTGT	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	0	1	0	0	0	0	0

Figure 7-29. Input FIFOs A/B Interrupt Requests

This register controls the sense and value for the internal A-OUT FIFO Programmable Flag. The internal flag may be tested by the 8051 and/or enabled to cause an INT4 interrupt request. The default value of the AOUTPF Register indicates a half-full condition.

The 8051 tests the internal FIFO programmable flag by reading the AOUTPF Bit in ABOUTCS.5 (Register at 0x7818).

Bit 7: **LTGT** *Less-than, Greater-than flag*

If LTGT=0, the AOUTPF flag goes true if the number of bytes in the FIFO is less than or equal to the programmed value in D[6..0].

If LTGT=1, the AOUTPF flag goes true if the number of bytes in the FIFO is greater than or equal to the value programmed into D[6..0].

Bit 6-0: **PFVAL** *Programmable Flag Value*

This value, along with the LTGT Bit, determines when the programmable flag for the A-OUT FIFO becomes active. The 8051 programs this register to indicate various degrees of A-OUT FIFO *fullness* to suit the application. The following two sections in this chapter show the interaction of the LTGT Bit and the programmed value for two cases, a filling FIFO and an emptying FIFO.

7.2.12.1 Filling FIFO

When a FIFO is filling with data, it is useful to generate an 8051 interrupt when a programmed level is reached. Because the interrupt request is triggered on a zero-to-one transition of the programmable flag AOUTPF, the LTGT Bit should be set to “1.” In Table 7-7, D[6..0] is set for 48 bytes, and the LTGT Bit is set to “1.” When the FIFO reaches 48 bytes, the AINPF Bit goes high, generating an interrupt request.

Table 7-7. Filling FIFO

LTGT	D[6..0]	Bytes in FIFO	AOUTPF
1	48	45	0
1	48	46	0
1	48	47	0
1	48	48	1
1	48	49	1
1	48	50	1

7.2.12.2 Emptying FIFO

When a FIFO is being emptied of data, the LTGT Bit should be set to “0,” so that the zero-to-one transition of the PF flag (therefore, the interrupt request) occurs when the byte count descends to below the programmed value. In Table 7-8, D[6..0] is set to 48 bytes, and when the FIFO goes from 49 bytes to 48 bytes, the AOUTPF Bit goes high, generating an interrupt request.

Table 7-8. Emptying FIFO

LTGT	D[6..0]	Bytes in FIFO	AOUTPF
0	48	51	0
0	48	50	0
0	48	49	0
0	48	48	1
0	48	47	1
0	48	46	1

7.2.13 A-OUT FIFO Pin Programmable Flag

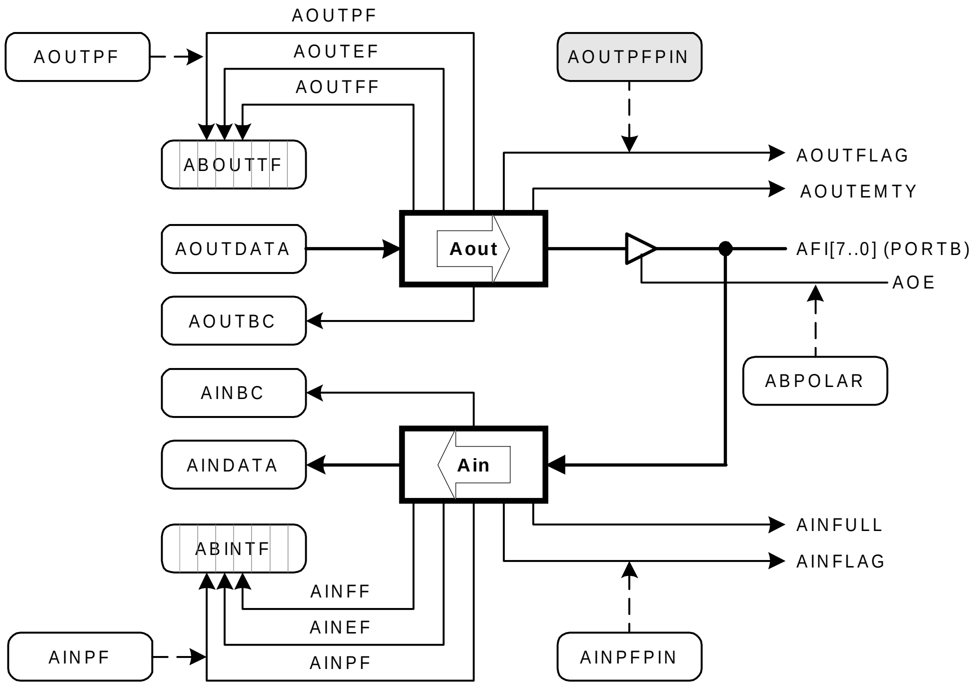


Figure 7-30. AOUTPFPIN's Role in the FIFO A Register

AOUTPFPIN	A-OUT FIFO Pin Programmable Flag	7811

b7	b6	b5	b4	b3	b2	b1	b0
LTGT	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	0	0	0	0	0	0

Figure 7-31. A-OUT FIFO Pin Programmable Flag

This register controls the sense and value for the A-OUT FIFO Programmable Flag that appears on the AOUTFLAG *pin*. This pin is used by external logic to regulate external reads from the A-OUT FIFO. The AOUTPFPIN Register is programmed with the same data format as the previous register, AOUTPF. The only operational difference is that the flag drives a hardware pin, rather than existing as an internal register bit.

Having separate programmable flags allows the 8051 and external logic to have independent gauges of FIFO fullness. It may be desirable, for example, for one side (8051 or external logic) to have advance notice over the other side about a FIFO becoming full or empty.

The default value of the AOUTPFPIN Register indicates full (bytes in FIFO greater than or equal to 64.

7.2.14 B-OUT FIFO Write Data

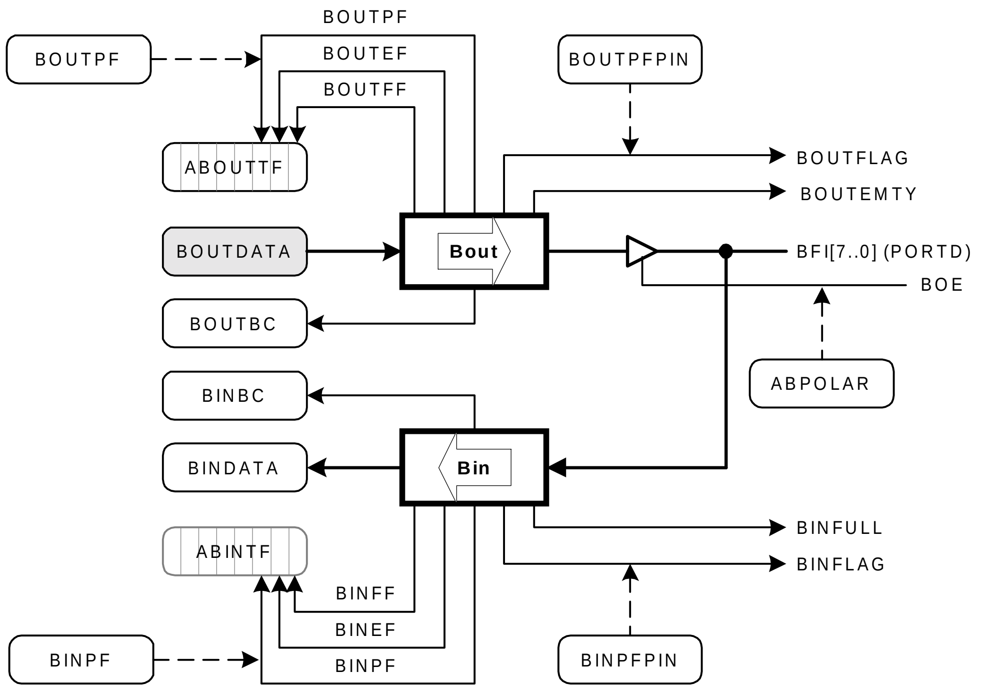


Figure 7-32. BOUTDATA's Role in the FIFO B Register

BOUTDATA			**B-OUT FIFO Write Data**				**7813**
b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
W	W	W	W	W	W	W	W
x	x	x	x	x	x	x	x

Figure 7-33. B-OUT FIFO Write Data

Each time the 8051/DMA writes a byte to this register, the B-OUT FIFO advances to the next open position in the FIFO and the BOUTBC (Byte count) increments. Writing this register when there are 63 bytes remaining in the B-OUT FIFO sets the B-FIFO Full Flag (BOUTFF, in ABOUTCS.0). This causes an INT4 interrupt request. Writing this register when the B-OUT FIFO is full (64 bytes) does not update the FIFO or byte count, and has no effect on the FIFO flags or byte count.

7.2.15 B-OUT FIFO Byte Count

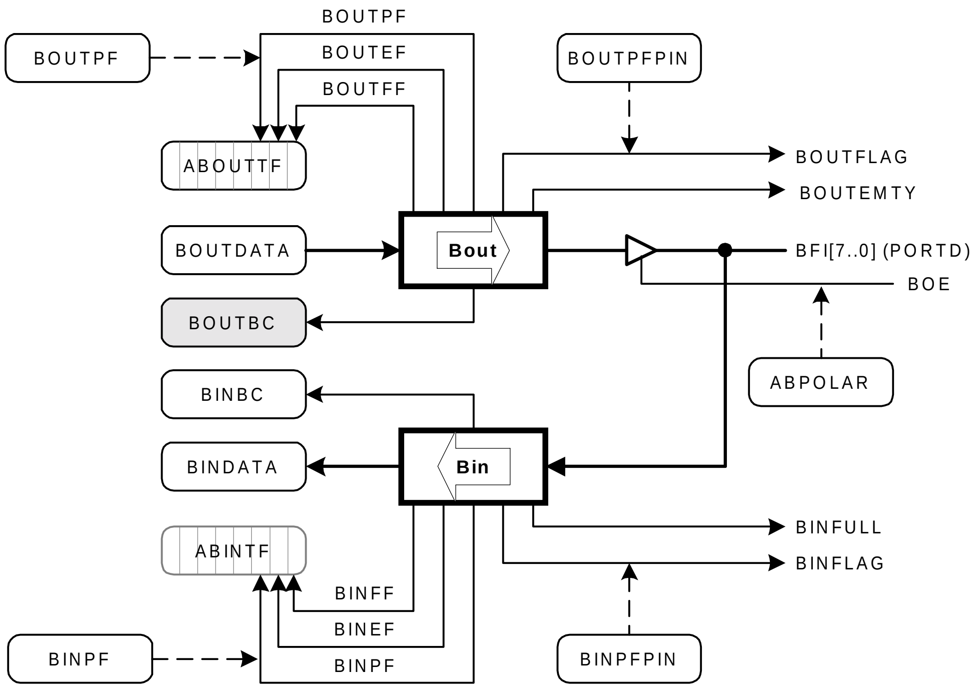


Figure 7-34. BOUTBC's Role in the FIFO B Register

BOUTBC **B-OUT FIFO Byte Count** **7814**

b7	b6	b5	b4	b3	b2	b1	b0
0	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Figure 7-35. B-OUT FIFO Byte Count

This count reflects the number of bytes remaining in the B-OUT FIFO. Valid byte counts are 0-64. When non-zero, every byte read by outside logic decrements this count, and every 8051 write of BOUTDATA increments this count. If BOUTBC is zero, reading a data byte by outside logic returns indeterminate data and results in the byte count in BOUTBC remaining at zero.

7.2.16 B-OUT FIFO Programmable Flag

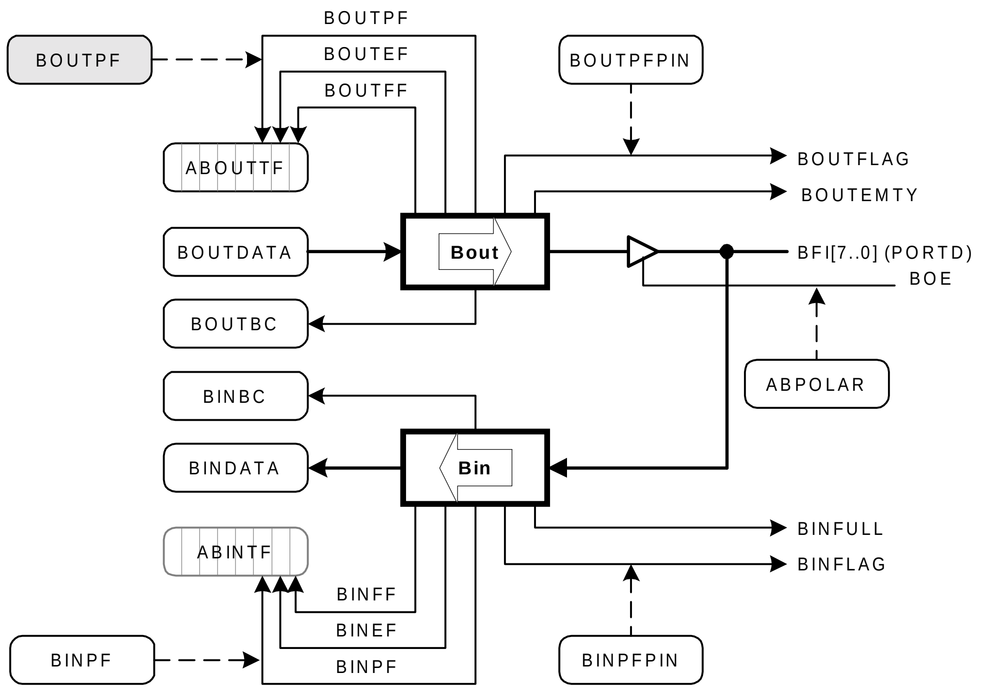


Figure 7-36. BOUTPF's Role in the FIFO B Register

BOUTPF	B-OUT FIFO Programmable Flag						7815
b7	b6	b5	b4	b3	b2	b1	b0
LTGT	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	0	1	0	0	0	0	0

Figure 7-37. B-OUT FIFO Programmable Flag

This register controls the sense and value for the internal B-OUT FIFO Programmable Flag. The internal flag may be tested by the 8051, and/or enabled to cause an INT4 interrupt request. The default value of the BOUTPF Register indicates a half-full condition.

The 8051 tests the internal FIFO programmable flag by reading the BOUTPF Bit in ABOUTCS.2.

Bit 7: **LTGT** *Less-than, Greater-than flag*

If LTGT=0, the BOUTPF flag goes true if the number of bytes in the FIFO is less than or equal to the programmed value in D[6..0].

If LTGT=1, the BOUTPF flag goes true if the number of bytes in the FIFO is greater than or equal to the value programmed into D[6..0].

Bit 6-0: **PFVAL** *Programmable Flag Value*

This value, along with the LTGT Bit, determines when the programmable flag for the B-FIFO becomes active. The 8051 programs this register to indicate various degrees of B-FIFO *fullness* to suit the application. The following two sections of this chapter show the interaction of the LTGT Bit and the programmed value for two cases, a filling FIFO and an emptying FIFO.

7.2.16.1 Filling FIFO

When a FIFO is filling with data, it is useful to generate an 8051 interrupt when a programmed level is reached. Because the interrupt request is triggered on a zero-to-one transition of the programmable flag BOUTPF, the LTGT Bit should be set to “1.” In Table 7-9, D[6..0] is set for 48 bytes and the LTGT Bit is set to “1.” When the FIFO reaches 48 bytes, the BOUTPF Bit goes high, generating an interrupt request.

Table 7-9. Filling FIFO

LTGT	D[6..0]	Bytes in FIFO	BOUTPF
1	48	45	0
1	48	46	0
1	48	47	0
1	48	48	1
1	48	49	1
1	48	50	1

7.2.16.2 Emptying FIFO

When a FIFO is being emptied of data, the LTGT Bit should be set to “0,” so the zero-to-one transition of the BOUTPF flag (therefore, the interrupt request) occurs when the byte count descends to below the programmed value. In Table 7-10, D[6..0] is set to 48 bytes. When the FIFO goes from 49 bytes to 48 bytes, the BOUTPF Bit goes high, generating an interrupt request.

Table 7-10. Emptying FIFO

LTGT	D[6..0]	Bytes in FIFO	BOUTPF
0	48	51	0
0	48	50	0
0	48	49	0
0	48	48	1
0	48	47	1
0	48	46	1

7.2.17 B-OUT FIFO Pin Programmable Flag

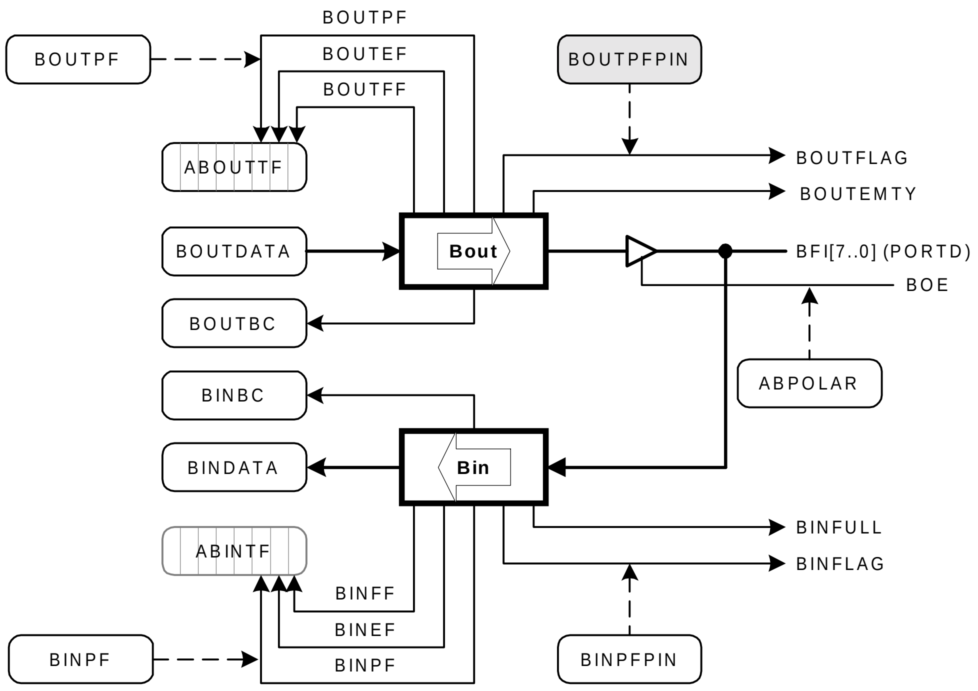


Figure 7-38. BOUTPFPIN's Role in the FIFO B Register

BOUTPFPIN	B-OUT FIFO Pin Programmable Flag	7816

b7	b6	b5	b4	b3	b2	b1	b0
LTGT	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	0	0	0	0	0	0

Figure 7-39. B-OUT FIFO Pin Programmable Flag

This register controls the sense and value for the B-OUT FIFO Programmable Flag that appears on the BOUTFLAG *pin*. This pin is used by external logic to regulate external reads from the B-OUT FIFO. The BOUTPFPIN Register is programmed with the same data format as the previous register, BOUTPF. The only operational difference is that the flag drives a hardware pin rather than existing as an internal register bit.

Having separate, programmable flags allows the 8051 and external logic to have independent gauges of FIFO fullness. It may be desirable, for example, for one side (8051 or external logic) to have advance notice over the other side about a FIFO becoming full or empty.

The default value of the BOUTPFPIN Register indicates full.

7.2.18 Output FIFOs A/B Toggle CTL and Flags

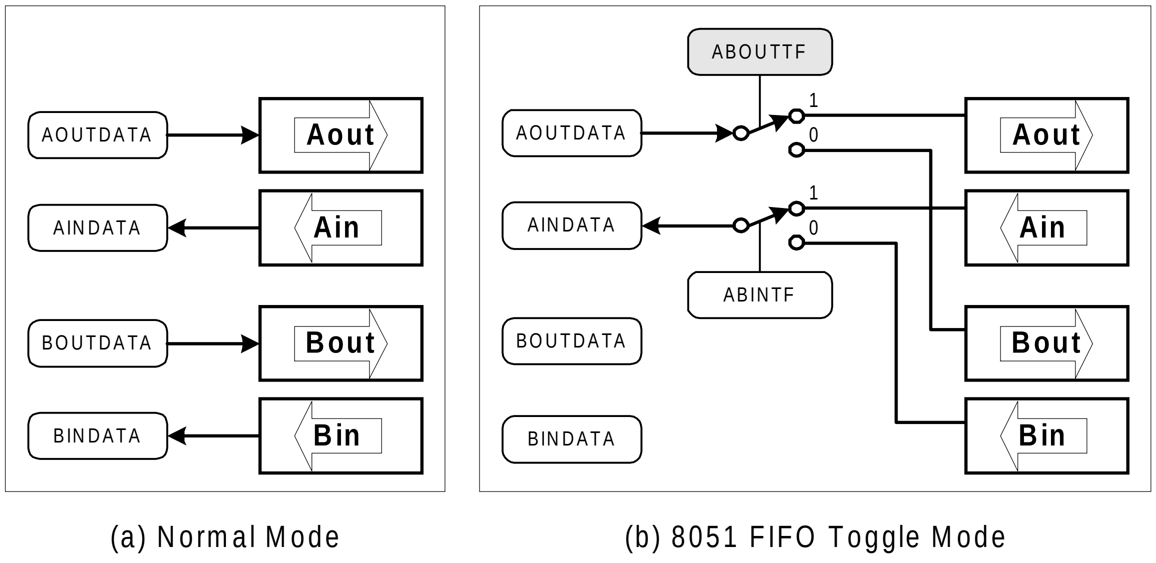


Figure 7-40. 8051 FIFO Toggle Mode vs. Normal Mode Diagram

ABOUTCS	Output FIFOs A/B Toggle CTL and Flags	7818

b7	b6	b5	b4	b3	b2	b1	b0
OUTTOG	**OUTSEL**	**AOUTPF**	**AOUTEF**	**AOUTFF**	**BOUTPF**	**BOUTEF**	**BOUTFF**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	1	0	1	0	0	1	0

Figure 7-41. Output FIFOs A/B Toggle CTL and Flags

Bit 7: OUTTOG *Enable Output FIFO Toggle*

A special FIFO toggle mode switches automatically between the A-OUT and B-OUT FIFOs each time the 8051 writes data to the AOUTDATA Register. The toggle mechanism works only for programmed 8051 transfers, <u>not</u> DMA transfers.

When OUTTOG=0, the A-OUT and B-OUT FIFOs operate in Normal Mode, as shown by diagram (a) in *Figure 7-40* on the previous page.

When OUTTOG=1, the FIFOs operate in Toggle Mode, as shown by diagram (b) in *Figure 7-40*. The selected FIFO switches between the A-OUT and B-OUT FIFOs after every 8051 write to the AOUTDATA Register. The selected FIFO is indicated by the OUTSEL Bit (Bit 6).

Bit 6: OUTSEL *Input Toggle Select*

If OUTTOG=1 when enabling the Toggle Mode:

- This bit selects OUT FIFO A or B when the 8051 writes to the AOUTDATA Register. When OUTSEL=0, the B-OUT FIFO is written. When OUTSEL=1, the A-OUT FIFO is written. When OUTTOG=1, this bit complements automatically (toggles) after every 8051 write to AOUTDATA. This has the effect of automatically toggling between the A-OUT and B-OUT FIFOs for successive 8051 writes to AOUTDATA.
- The 8051 can directly write this bit to select *manually* the A-OUT or B-OUT FIFO. More commonly, the Toggle Mode is used, since it allows 16-bit transfers using the 8051 without requiring the 8051 to switch between the FIFOs.

If OUTTOG=0 when enabling the Toggle Mode:

- The OUTSEL Bit has no effect.

Bit 5: AOUTPF *A-OUT FIFO Programmable Flag*

AOUTPF=1 when the number of bytes in the A-OUT FIFO satisfies the requirements programmed into the AOUTPF Register; otherwise, AOUTPF=0. This bit may be tested by the

8051and/or used to generate an interrupt request. A zero-to-one transition of this flag sets the interrupt request bit AOUTPFIR.

Bit 4: **AOUTEF** *A-OUT FIFO Empty Flag*

AOUTEF=1 when the A-OUT FIFO is empty; otherwise, AOUTEF=0. The flag goes valid after external logic reads the last byte in the A-OUT FIFO. This bit may be tested by the 8051, and/or used to generate an interrupt request. A zero-to-one transition of this flag sets the interrupt request bit AOUTEFIR.

Bit 3: **AOUTFF** *A-OUT FIFO Full Flag*

AOUTFF=1 when the A-OUT FIFO is full; otherwise, AOUTFF=0. The flag goes valid after the 8051/DMA writes the 64th byte into the A-OUT FIFO. A zero-to-one transition of this flag sets the interrupt request bit AOUTFFIR.

Bit 2: **BOUTPF** *B-OUT FIFO Programmable Flag*

BOUTPF=1 when the number of bytes in the B-OUT FIFO satisfies the requirements programmed into the BOUTPF Register; otherwise, BOUTPF=0. A zero-to-one transition of this flag sets the interrupt request bit BOUTPFIR.

Bit 1: **BOUTEF** *B-OUT FIFO Empty Flag*

BOUTEF=1 when the B-OUT FIFO is empty; otherwise, BOUTEF=0. The flag goes valid after external logic reads the last byte in the B-OUT FIFO. A zero-to-one transition of this flag sets the interrupt request bit BOUTEFIR.

Bit 0: **BOUTFF** *B-OUT FIFO Full Flag*

BOUTFF=1 when the B-OUT FIFO is full; otherwise, BOUTFF=0. The flag goes valid after the 8051/DMA writes the 64th byte into the B-OUT FIFO. A zero-to-one transition of this flag sets the interrupt request bit BOUTFFIR.

7.2.19 Output FIFOs A/B Interrupt Enables

ABOUTIE **Input FIFOs A/B Interrupt Enables** **7819**

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**AOUTPFIE**	**AOUTEFIE**	**AOUTFFIE**	**BOUTPFIE**	**BOUTEFIE**	**BOUTFFIE**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 7-42. Output FIFOs A/B Interrupt Enables

Bit 5: **AOUTPFIE** *A-OUT FIFO Programmable Flag Interrupt Enable*

The 8051 sets AOUTPFIE=1 to enable an INT4 interrupt when the AOUTPFIR interrupt request bit makes a zero-to-one transition. This transition indicates that the A-OUT FIFO byte count has satisfied the *fullness* level programmed into the programmable FIFO flag register AOUTPF.

Bit 4: **AOUTEFIE** *A-OUT FIFO Empty Interrupt Enable*

The 8051 sets AOUTEFIE=1 to enable an INT4 interrupt when the AOUTEFIR interrupt request bit makes a zero-to-one transition. This indicates an A-OUT FIFO byte count of zero (FIFO empty).

Bit 3: **AOUTFFIE** *A-OUT FIFO Bull Interrupt Enable*

The 8051 sets AOUTFFIE=1 to enable an INT4 interrupt when the AOUTFFIR interrupt request bit makes a zero-to-one transition. This indicates an A-OUT FIFO byte count of 64 (FIFO full).

Bit 2: **BOUTPFIE** *B-OUT FIFO Programmable Flag Interrupt Enable*

The 8051 sets BOUTPFIE=1 to enable an INT4 interrupt when the BOUTPFIR interrupt request bit makes a zero-to-one transition. This indicates a B-OUT FIFO byte count has satisfied the *fullness* level programmed into the programmable FIFO flag register BOUTPF.

Bit 1: **BOUTEFIE** *B-OUT FIFO Empty Interrupt Enable*

The 8051 sets BOUTEFIE=1 to enable an INT4 interrupt when the BOUTEFIR interrupt request bit makes a zero-to-one transition. This indicates a B-OUT FIFO byte count of zero (FIFO empty).

Bit 0: **BOUTFFIE** *B-OUT FIFO Full Interrupt Enable*

The 8051 sets BOUTFFIE=1 to enable an INT4 interrupt when the BOUTFFIR interrupt request bit makes a zero-to-one transition. This indicates a B-OUT FIFO byte count of 64 (FIFO full).

7.2.20 Output FIFOs A/B Interrupt Requests

ABOUTIRQ **Output FIFOs A/B Interrupt Requests** **781A**

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**AOUTPFIR**	**AOUTEFIR**	**AOUTFFIR**	**BOUTPFIR**	**BOUTEFIR**	**BOUTFFIR**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 7-43. Output FIFOs A/B Interrupt Requests

Bit 5: **AOUTPFIR** *A-OUT FIFO Programmable Flag Interrupt Request*

AOUTPFIR makes a zero-to-one transition when the A-OUT FIFO byte count satisfies the required condition programmed into the programmable FIFO flag register AOUTPF. If enabled by the AOUTPFIE Bit, this transition causes an INT4 interrupt request.

The 8051 writes a “1” to this bit to clear the interrupt request. The 8051 should clear the 8051 INT4 Bit (EXIF.6) before clearing the AOUTPFIR Bit in the interrupt service routine to guarantee that pending INT4 interrupts will be recognized.

Bit 4: **AOUTEFIR** *A-OUT FIFO Empty Interrupt Request*

AOUTEFIR makes a zero-to-one transition when the A-OUT FIFO byte count reaches zero (FIFO empty). If enabled by the AOUTEFIE Bit, this transition causes an INT4 interrupt request.

The 8051 writes “1” to this bit to clear the interrupt request. The 8051 should clear the 8051 INT4 Bit (EXIF.6) before clearing the AOUTEFIR Bit in the interrupt service routine to guarantee that pending INT4 interrupts will be recognized.

Bit 3: **AOUTFFIR** *A-OUT FIFO Full Interrupt Request*

AOUTFFIR makes a zero-to-one transition when the A-OUT FIFO byte count reaches 64 (FIFO full). If enabled by the AOUTFFIE Bit, this transition causes an INT4 interrupt request.

The 8051 writes a “1” to this bit to clear the interrupt request. The 8051 should clear the 8051 INT4 Bit (EXIF.6) before clearing the AOUTFFIR Bit in the interrupt service routine to guarantee that pending INT4 interrupts will be recognized.

Bit 2: **BOUTPFIR** *B-OUT FIFO Programmable Flag Interrupt Request*

BOUTPFIR makes a zero-to-one transition when the B-OUT FIFO byte count satisfies the required condition programmed into the programmable FIFO flag register BOUTPF. If enabled by the BOUTPFIE Bit, this transition causes an INT4 interrupt request.

The 8051 writes a “1” to this bit to clear the interrupt request. The 8051 should clear the 8051 INT4 Bit (EXIF.6) before clearing the BOUTPFIR Bit in the interrupt service routine to guarantee that pending INT4 interrupts will be recognized.

Bit 1: **BOUTEFIR** *B-OUT FIFO Empty Interrupt Request*

BOUTEFIR makes a zero-to-one transition when the B-OUT FIFO byte count reaches zero (FIFO empty). If enabled by the BOUTEFIE Bit, this transition causes an INT4 interrupt request.

The 8051 writes a “1” to this bit to clear the interrupt request. The 8051 should clear the 8051 INT4 Bit (EXIF.6) before clearing the BOUTEFIR Bit in the interrupt service routine to guarantee that pending INT4 interrupts will be recognized.

Bit 0: **BOUTFFIR** *B-OUT FIFO Full Interrupt Request*

BOUTFFIR makes a zero-to-one transition when the B-OUT FIFO byte count reaches 64 (FIFO full). If enabled by the BOUTFFIE Bit, this transition causes an INT4 interrupt request.

The 8051 writes a “1” to this bit to clear the interrupt request. The 8051 should clear the 8051 INT4 Bit (EXIF.6) before clearing the BOUTFFIR Bit in the interrupt service routine to guarantee that pending INT4 interrupts will be recognized.

7.2.21 FIFO A/B Setup

ABSETUP **FIFO A/B Setup** **781C**

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**ASYNC**	**DBLIN**	**0**	**OUTDLY**	**0**	**DBLOUT**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 7-44. FIFO A/B Setup

Bit 5: **ASYNC** *Select SYNC/ASYNC Slave FIFO Clocking*

The ASYNC Bit controls how external logic synchronizes accesses to the A and B FIFOs.

When the 8051 sets ASYNC=1, the A and B FIFOs operate asynchronously, whereby the SLRD (Slave FIFO-READ) and SLWR (Slave FIFO-WRITE) pins are used as direct read and write strobes.

When the 8051 sets ASYNC=0, the A and B FIFOs operate synchronously, whereby the SLRD and SLWR pins are used as enable signals for the externally supplied FIFO clock XCLK. The polarity of the enables, active-high or active-low, is controlled by the ABPOLAR Register (Section 7.2.22. *"FIFO A/B Control Signal Polarities"*).

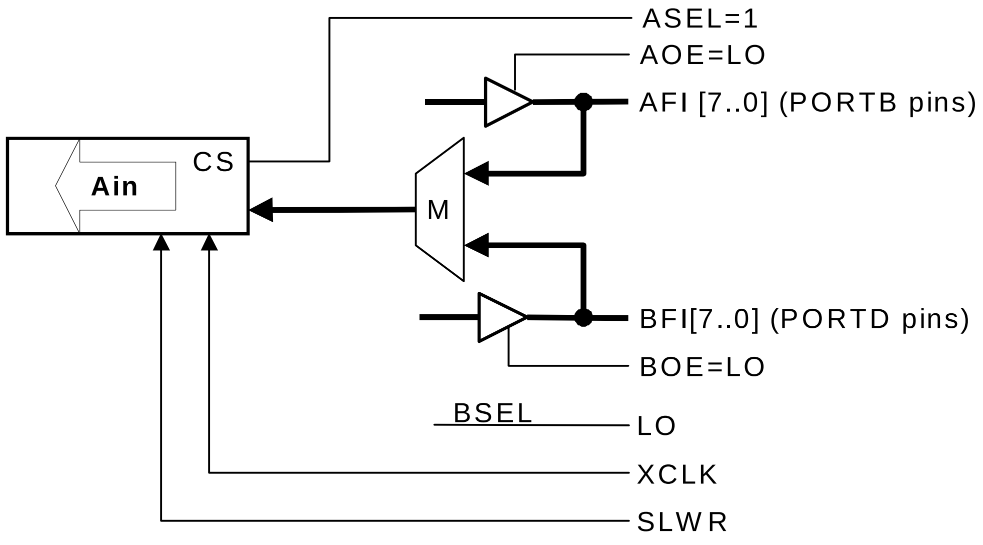


Figure 7-45. A-IN FIFO Double-Byte Mode

Bit 4: **DBLIN** *Enable IN Double-Byte Mode*

The 8051 sets DBLIN=1 to turn on the IN-FIFO double-byte mode. *Figure 7-45* illustrates the double-byte mode for the A-IN FIFO. The B-IN FIFO may also use this mode, in which case the outside logic sets ASEL=0 and BSEL=1. For this illustration, signals ASEL, BSEL, AOE, and BOE are programmed to be active high polarity.

In double-byte mode, external logic writes 16 bits of data into the A-IN or B-IN FIFO each time it asserts the SLWR signal. The double-byte mode automatically writes two bytes for every SLWR pulse in ASYNC mode or two bytes for every clock pulse in SYNC mode. The bytes are taken from PORTD and PORTB, in that order. This provides a very efficient mechanism for transferring 16-bit data into the 8-bit slave FIFOs.

If synchronous clocking is used in double-byte mode, consecutive writes must be separated by at least one XCLK period to give the internal logic time to write both bytes into the FIFO. This clocking restriction applies only to the double-byte mode. In normal operation, one byte per clock can be loaded into a slave IN-FIFO.

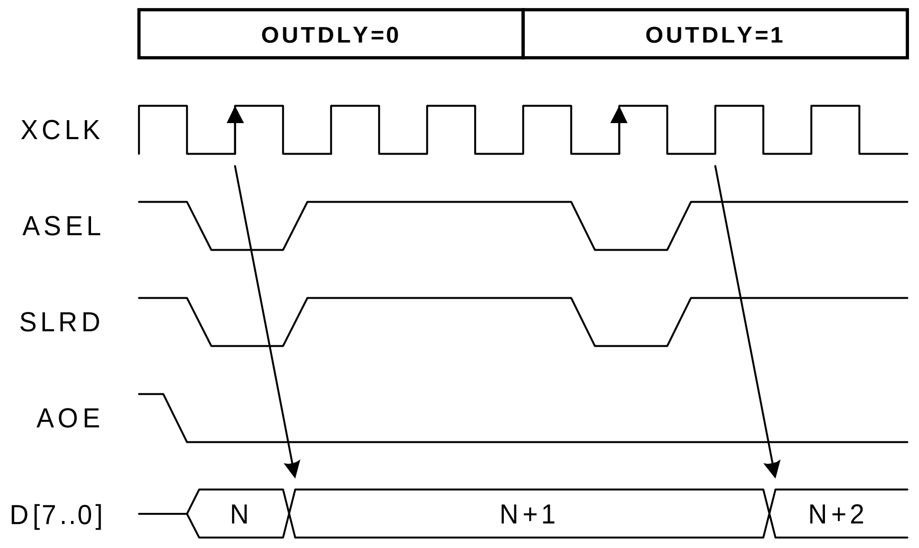


Figure 7-46. A-OUT FIFO Delay Synchronous Reads

Bit 2: **OUTDLY** *Delay Synchronous Reads*

The OUTDLY Bit affects only synchronous reads of a slave FIFO. When OUTDLY=0, output data is valid on the clock edge that corresponds to the SLRD signal being valid. When OUT-DLY=1, the output data is valid one clock later.

Figure 7-46 shows two synchronous reads of the A-OUT FIFO, with the OUTDLY Bit first equal to 0, then equal to 1. For this example, the SLRD, AOE, and ASEL signals are programmed to be active low.

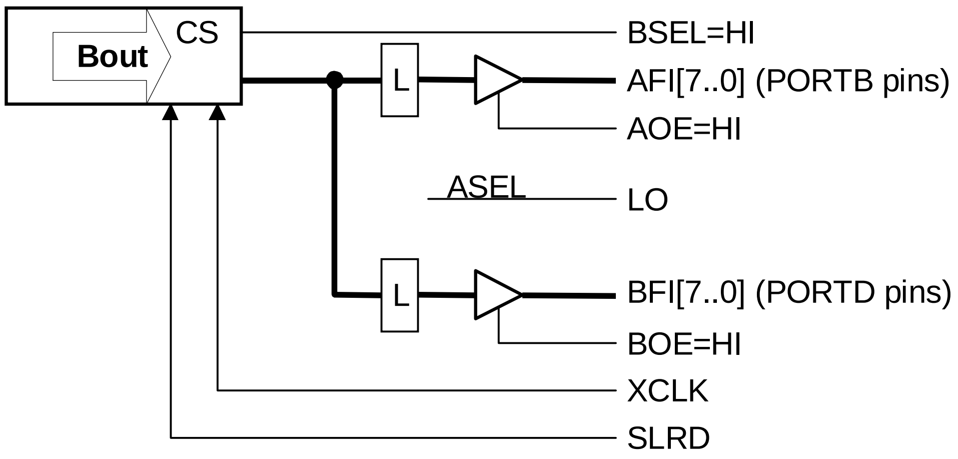


Figure 7-47. B-OUT FIFO Double-Byte Mode

Bit 0: DBLOUT *Enable FIFO-OUT Double-Byte Mode*

The 8051 sets DBLOUT=1 to turn on the OUT-FIFO double-byte mode. *Figure 7-47* illustrates the double-byte mode for the B-OUT FIFO. The A-OUT FIFO may also use this mode, in which case the outside logic sets ASEL=1 and BSEL=0. For this illustration, signals ASEL, BSEL, AOE, and BOE are programmed to be active high polarity.

The double-byte mode automatically provides two FIFO bytes on PORTDand PORTB, in that order, for every SLRD pulse in ASYNC mode or two bytes for every clock pulse in SYNC mode. This provides a very efficient mechanism for transferring 16-bit data out of the 8-bit slave FIFOs.

In SYNC mode, consecutive reads must be separated by at least one XCLK period, to give the internal logic time to retrieve both bytes from the FIFO.

7.2.22 FIFO A/B Control Signal Polarities

ABPOLAR **FIFO A/B Control Signal Polarities** **781D**

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**BOE**	**AOE**	**SLRD**	**SLWR**	**ASEL**	**BSEL**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 7-48. FIFO A/B Control Signal Polarities

These bits define the pin polarities for the indicated signals. The 8051 sets a bit LOW for active low, and HI for active high. The default setting for all FIFO A/B control signals is active low polarity.

7.2.23 FIFO Flag Reset

ABFLUSH	**Reset All FIFO Flags**	**781E**

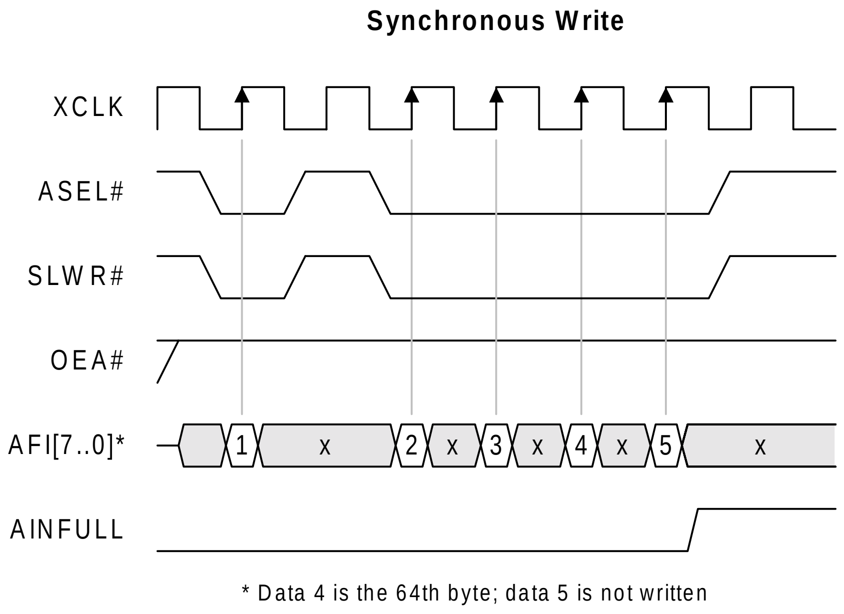
b7	b6	b5	b4	b3	b2	b1	b0
x	**x**	**x**	**x**	**x**	**x**	**x**	**x**
W	W	W	W	W	W	W	W
x	x	x	x	x	x	x	x

Figure 7-49. FIFO Flag Reset

The 8051 writes any value to this register to reset the FIFO byte counts to zero, effectively *flushing* the FIFOs. Consequently, the byte counts are set to zero, the empty flags are set, and the full flags are cleared.

Reading this register returns indeterminate data.

7.3 FIFO Timing



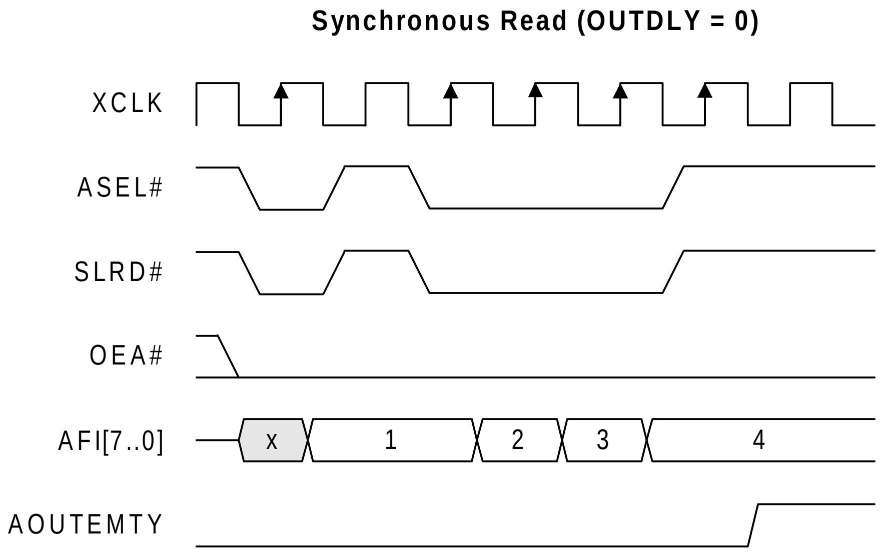
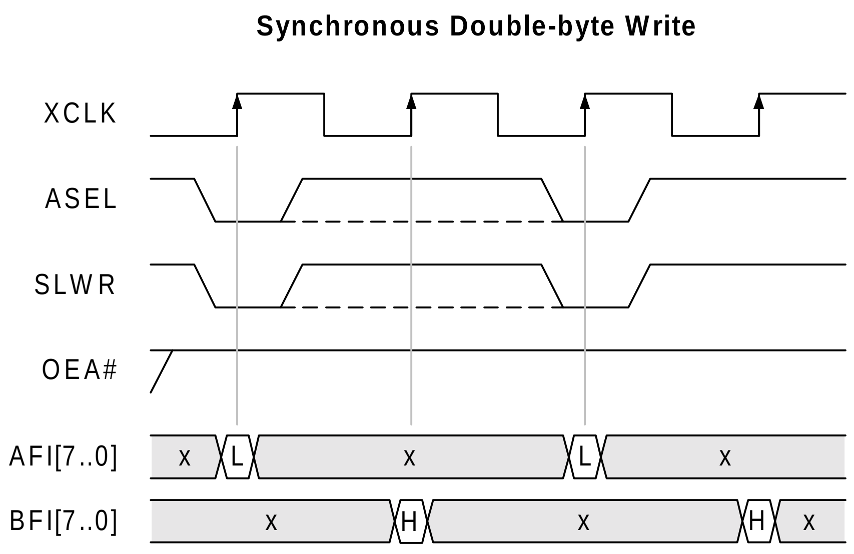


Figure 7-50. Synchronous Write/Read Timing



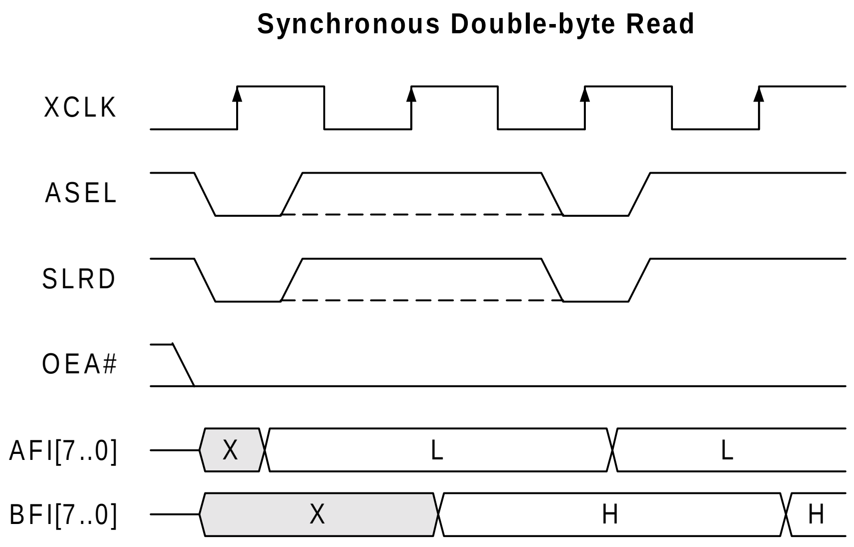


Figure 7-51. Synchronous Double-byte Write/Read

Chapter 8. General Programmable Interface (GPIF)

8.1 What is GPIF?

The **G**eneral **P**rogrammable **I**nter**F**ace (GPIF) is an extremely flexible 8- or 16-bit parallel interface that allows designers to reduce system costs by providing a *glueless* interface between the EZ-USB FX and many different types of external peripherals.

The GPIF allows the EZ-USB FX to perform local bus mastering to external peripherals using a wide variety of protocols. For example, EIDE/ATAPI, printer parallel port (IEEE P1284), Utopia, and other interfaces can be supported using the GPIF block of the EZ-USB FX.

To support a wide range of applications, the GPIF implements an extensive feature set that can be modified to suit the design. As with other highly configurable chips, some initialization steps are required. To support a range of interface styles, the GPIF provides multiple programmable I/O pins and multiple registers to configure those pins.

This chapter provides an overview of GPIF, discusses external connections, and explains the operation of the GPIF *engine*.

Figure 8-1 presents a block diagram illustrating GPIF's place in the FX System.

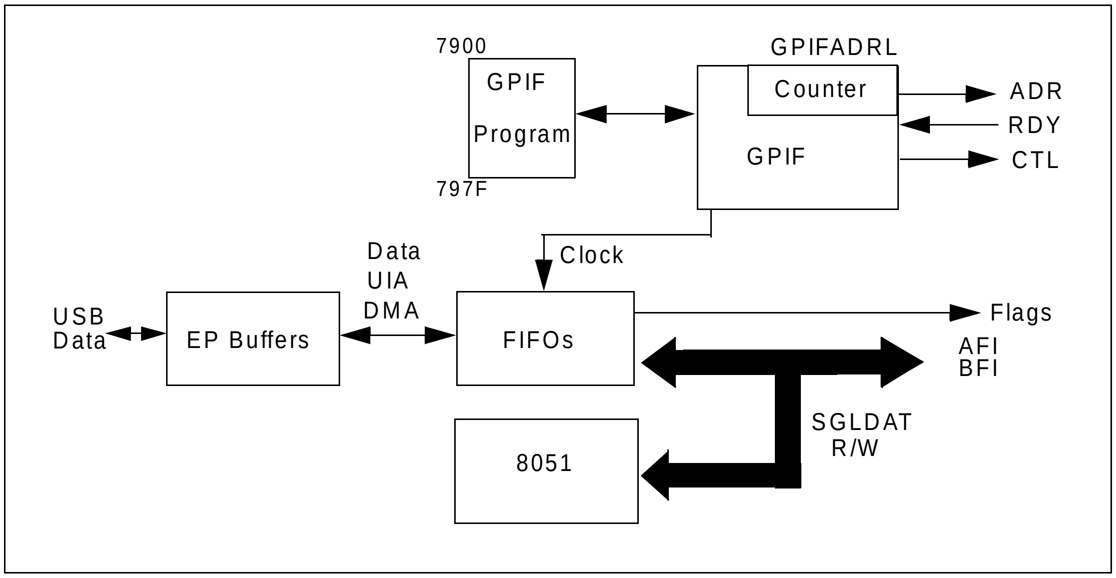


Figure 8-1. GPIF's Place in the FX System

8.2 Applicable Documents and Tools

- EZ-USB FX Data Sheet
- EPP Reference Design
- Mass Storage Reference Design
- GPIF Tool — A Windows application that assists GPIF firmware development. The GPIF Tool can be found on the EZ-USB *FX* Developer's Kit CD.

8.3 Typical GPIF Interface

The GPIF allows the EZ-USB FX connect directly to external peripherals such as ASICs, DSPs, or other digital logic that uses an 8- or 16-bit parallel interface.

The GPIF provides external pins that can operate as outputs (CTL0 to CTL5), inputs (RDY0 to RDY5), Data bus (GDA[7..0] and GDB[7..0]), and Address Lines (ADR0 to ADR5).

A Waveform Descriptor in internal RAM describes the behavior of each of the GPIF signals. The Waveform Descriptor is loaded into the GPIF registers by the 8051 firmware during initialization, and it is then used throughout the execution of the 8051 code to perform transactions over the GPIF interface.

Figure 8-2 shows a block diagram of a typical interface between the EZ-USB FX and a peripheral function.

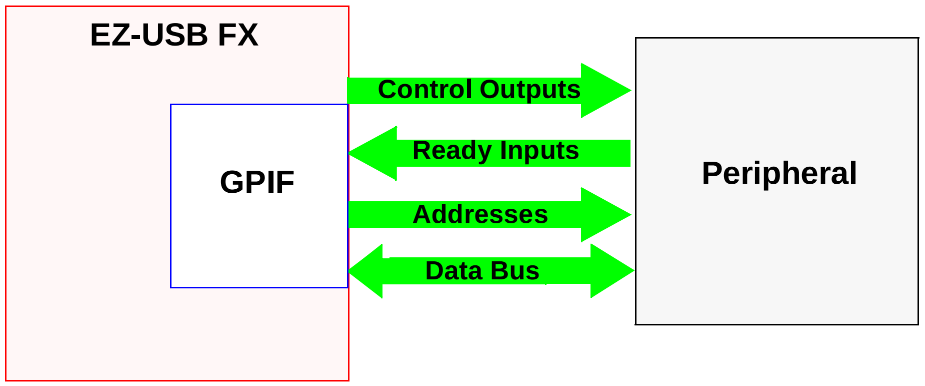


Figure 8-2. EZ-USB FX Interfacing to a Peripheral

The following sections detail the features available and steps needed to create an efficient GPIF design. This includes definition of the external GPIF connections and the internal register settings, along with 8051 firmware needed to execute data transactions over the interface.

8.4 External GPIF Connections

8.4.1 The External GPIF Interface

The GPIF provides many general input and output signals with which to interface your external peripherals *gluelessly* to the EZ-USB FX.

The GPIF interface signals are shown in **Table 8-1**.

Table 8-1. GPIF Pin Descriptions

PIN	IN/OUT	Description
ADR[5:0]	O	Address outputs
GDA[7:0]	I/O	Bidirectional A-FIFO data bus
GDB[7..0]	I/O	Bidirectional B-FIFO data bus
CTL[5:0]	O	Programmable control outputs
RDY[5:0]	I	Sampleable ready inputs

Refer to the figure *EZ-USB FX 128-pin Package* on p. 13 of the *CY7C64603/613 Data Sheet*.

The Control Outputs (CTL0 to CTL5) are intended to be strobes, read/write lines, and other non-bused outputs.

The Ready Inputs (RDY0 to RDY5) sample a signal to allow a transaction to wait (inserting wait states), continue, or repeat until the signal is at the appropriate level.

The GPIF Data Bus is a collection of the GDA[7..0] and GDB[7..0] pins.

- A GPIF interface 8 bits wide uses pins GDA[7..0].
- A GPIF interface 16 bits wide uses pins GDA[7..0] and GDB[7..0].

The GPIF Address lines (ADR0 to ADR5) can generate an automatically incrementing address during a burst transaction. For non-burst transactions, these address lines remain static. For higher-order address lines that may be needed, other non-GPIF I/O signals should be used.

The GPIF Clock can be either an internal 48MHz clock, or an externally-supplied clock from the XCLK pin. If the XCLK_SEL pin is tied high, the GPIF clock is the XCLK pin. Otherwise, the GPIF clock is the 48 MHz internal clock.

8.4.2 Connecting GPIF Signal Pins to Hardware

The first step in creating the interface between the EZ-USB FX GPIF and your peripheral is to define the hardware interconnects. This physical connection of GPIF signals to your interface signals determines the configurations that are necessary by your 8051 firmware.

1. **Determine the proper GPIF Data Bus size.** If your interface's data bus is 8 bits wide, use the GDA[7..0] pins. If your interface's data bus is 16 bits wide, use GDA[7..0] and GDB[7..0].

2. **Assign the CTL signals to your interface**. Make a list of all interface signals to be driven by your peripheral (inputs to the GPIF), and assign them to the RDY0 to RDY5 Inputs. If there are more input signals than available RDY inputs, you need to use other, non-GPIF I/O signals and sample them manually using 8051 firmware. In this case, you should choose the RDY inputs only for signals that must be sampled in the middle of a data transaction.

3. **Assign the RDY signals to your interface.** Make a list of all interface signals to be driven by your peripheral (inputs to the GPIF), and assign them to the RDY0 through RDY5 Inputs. If there are more input signals than available RDY inputs, you will need to use other non-GPIF I/O signals and sample them manually using 8051 firmware. In this case, you should choose to use the RDY inputs only for signals that must be sampled in the middle of a data transaction.

4. **Determine the proper GPIF Address connections**. If your interface uses an Address Bus, use the ADR0 through ADR5 signals for the least significant bits, and other non-GPIF I/O signals for the most significant bits. You may leave these signals unconnected if you do not use address signals, as with a FIFO.

8.4.3 Example GPIF Hardware Interconnect

The following example illustrates the hardware connections that can be made for a standard interface: a 27C256 EPROM.

Table 8-2. Example GPIF Hardware Interconnect

Step	Result	Connection Made
1. Determine the proper GPIF Data Bus size.	8 bits.	Connect GDA[7..0] to D0..D7 of the EPROM.
2. Assign the CTL signals to your interface.	CS# and OE# are inputs to the EPROM.	Connect CTL0 to CS# and CTL1 to OE#.
3. Assign the RDY signals to your interface.	There are no output ready/wait signals from a 27C256 EPROM.	No connection.
4. Determine the proper GPIF Address connections.	16 bits of address.	Connect ADR0..ADR5 to A0..A5 and other I/O ports to A6..A15.

The process is the same for larger and more complicated interfaces.

8.5 Internal GPIF Operation

8.5.1 The Internal GPIF Engine

The GPIF Engine is controlled by two blocks of registers in 8051 XDATA space:

- **GPIF Configuration Registers** — These registers configure the general settings and report the status of the interface to the 8051, a total of 18 registers from 0x7824 to 0x783C. See the *EZ-USB FX Register Summary* and the remainder of this chapter for details.
- **Waveform Memories** — A block of registers loaded by the 8051 with the Waveform Descriptors that program the GPIF interface, a total of 128 bytes from 0x7900 to 0x797F.

The GPIF has 4 Waveform Memories. Each Waveform Memory holds a GPIF program containing up to 7 programmed Intervals. Each Interval is a 32-bit instruction for the GPIF Engine.

The 8051 must load these registers before initiating GPIF operation.

8.5.2 Global GPIF Configuration

The GPIF configuration registers allow for various modes of operation. These modes control the global operation of the GPIF for all waveforms.

8.5.2.1 Data Bus Width

The GPIF can have an 8-bit or 16-bit wide data bus. It is selected by the BUS16 Bit (Bit 2) of the IFCONFIG Register:

- If BUS16 is 1, the data bus is 16 bits wide.
- If BUS16 is 0, the data bus is 8 bits wide.

Refer to the figure *EZ-USB FX 128-pin Package* on p. 13 of the *CY7C64603/613 Data Sheet*.

8.5.2.2 Control Output Modes

The GPIF Control pins (CTL0 to CTL5) have several output modes:

- CTL0 to CTL3 can actively drive CMOS levels 1 and 0, be open-drain, or tristate.
- CTL4 and CTL5 can actively drive CMOS levels 1 and 0, or be open-drain.

  If CTL0 to CTL3 are configured to be tristate-able, CTL4 and CTL5 are not available.

Table 8-3. CTL Output Modes

TRICTL (CTLOUTCFG.7)	CTLOUTCFG[6..0]	CTL0[3..0]	CTL[5..4]
1	1	0, 1, or tristate	Not Available
0	1	0 or open-drain	Open-drain
1	0	0, 1, or tri-stateable	Not Available
0	0	0 or 1	CMOS

Important: *The TRICTL Bit controls the meaning of the OUTPUT field in* ***all*** *currently loaded Waveform Programs.*

8.5.2.3 Synchronous/Asynchronous Mode

The GPIF interface can be operated in Asynchronous or Synchronous mode:

- In Asynchronous mode, the RDY inputs are double-sampled.
- In Synchronous mode, the RDY inputs are single-sampled, improving performance.

The synchronous/asynchronous mode is selected via the SAS Bit (Bit 6) of the READY Register:

If SAS = 1, the RDY[5..0] Inputs are synchronous to the GPIF Clock, sampled at only one rising edge of the GPIF Clock.

If SAS = 0, the RDY[5..0] Inputs are sampled at two rising edges of the GPIF Clock before the appropriate GPIF Branch is taken.

8.5.3 Programming GPIF Waveforms

GPIF Waveforms are programmed by the programmer coding ***Waveform Descriptors.*** Subsequently, the 8051 stores these descriptors into the ***Waveform Memories***. The Waveform Descriptors are the instructions to the GPIF engine — what to do when that Waveform is ***triggered***. There are 4 Waveform Memories in the EZ-USB FX, one for each of the following types of waveforms:

- Single Write Waveform
- Single Read Waveform
- FIFO Write Waveform
- FIFO Read Waveform

(See the WF_SELECT Register in Section 8.5.5.5. *"Waveform Selector".*)

Each Waveform Descriptor consists of up to seven 32-bit instructions that program key transition points for all GPIF interface signals. These instructions are called ***Intervals*** because they have a one-to-one correspondence to intervals of time in an actual waveform. For that portion of the Waveform, an Interval defines, the

- state of the CTL outputs
- state of GDA[7..0] and GDB[7..0] (GPIF Data Bus)
- use of the RDY inputs
- behavior of the GPIF address bus.

Intervals always begin at the rising edge of the GPIF Clock.

8.5.3.1 The GPIF IDLE State

A Waveform consists of *up to* seven programmable Intervals (I0 to I6). **Every Waveform terminates when the GPIF program branches to a special IDLE interval or IDLE state, which is by convention Interval 7 (I7).**

To complete a GPIF transaction, the GPIF program always branches to the IDLE state, **regardless of the interval that the GPIF program is currently executing**. For example, it is possible to have a GPIF Waveform that has 3 Intervals: I0, I1 and I7. The GPIF program branches from I1 (or I0) to I7 when it wishes to terminate.

The IDLE state is the time between two active GPIF transactions.

The state of the GPIF signals during the IDLE state is determined by the contents of the IDLE_CS and IDLE_CTLOUT Registers.

The 8051 programmer must make sure the GPIF is IDLE before starting the next Waveform. The 8051 senses completion of a Waveform (the Waveform Program branching to the GPIF IDLE state) by reading the DONE Bit (Bit 7) in the IDLE_CS Register.

- If DONE is 0, the GPIF is *Busy* generating a Waveform.
- If DONE is 1, the GPIF is *Done* (GPIF is in the IDLE state), ready for the 8051 to start the next GPIF transaction.

Important: *It is illegal to* initiate *any operation (except aborting the current transaction) when the GPIF is Busy. Doing so yields indeterminate behavior, likely to cause data corruption.*

8.5.3.1.1 GPIF Data Bus During IDLE

During the IDLE state, the GPIF Data Bus (GDA[7..0] and GDB[7..0]) can be either driven or tristated, depending on how the 8051 program has set the IDLEDRV bit (bit 0) of the IDLE_CS Register.

- If IDLEDRV is 0, the GPIF Data Bus tristates during IDLE.
- If IDLEDRV is 1, the GPIF Data Bus is actively driven during IDLE. The value driven is the last value driven by any GPIF Waveform program.

8.5.3.1.2 CTL Outputs During IDLE

During the IDLE state, the state of CTL[5..0] depends on the following register bits

- TRICTL Bit (Bit 7) in the CTLOUTCFG Register (as described in the previous Control Output Modes.
- IDLE_CTLOUT[5..0] Register bits.

The combination of TRICTL and IDLE_CTLOUT[5..0] define CTL[5..0] during IDLE as follows:

- If TRICTL is 0, IDLE_CTLOUT[5..0] is the output state of CTL[5..0] during the IDLE state.
- If TRICTL is 1, IDLE_CTLOUT[7..4] are the Output Enables for the CTL0 to CTL3 signals, and IDLE_CTLOUT[3..0] are the Output values for CTL0 to CTL3.

Table 8-4 below illustrates this relationship.

Table 8-4. Control Outputs (CTLn) During the IDLE State

Control Output	CTLOUTCFG.7 TRICTL	IDLE_CTLOUT Output Enable	IDLE_CTLOUT Output Bit
CTL0	0	Not Available	Bit 0 (CTL0)
	1	Bit 4 (OE0)	
CTL1	0	Not Available	Bit 1 (CTL1)
	1	Bit 5 (OE1)	
CTL2	0	Not Available	Bit 2 (CTL2)
	1	Bit 6 (OE2)	
CTL3	0	Not Available	Bit 3 (CTL3)
	1	Bit 7 (OE3)	
CTL4	0	Not Available	Bit 4 (CTL4)
	1	Not Available	Not Available
CTL5	0	Not Available	Bit 5 (CTL5)
	1	Not Available	Not Available

The CTL[5..0] lines are also affected by the corresponding bit in the CTLOUTCFG Register.

- If the IDLE_CTLOUT Register and TRICTL Bit indicate that a 1 is to be driven, then a 1 in the corresponding bit in the CTLOUTCFG Register makes the output an open-drain.
- If the IDLE_CTLOUT Register and TRICTL Bit indicate that a 0 is to be driven, then a 0 in the corresponding bit in the CTLOUTCFG Register makes the output actively drive a CMOS high level.

8.5.3.2 Defining Intervals

Each Waveform is made up of a number of Intervals. A single Interval is defined by a 4-byte Waveform Descriptor, and it is defined as one of two basic types. Each Interval is programmed as either:

- Non-Decision Point (NDP) Interval
- Decision Point (DP) Interval.

It is possible to change only selected bytes of a Waveform Program. It is not necessary to reload the entire program if only a few bytes change. Hence, the 8051 can quickly reconfigure the GPIF when it is not Busy.

8.5.3.2.1 Non-Decision Point (NDP) Intervals

For NDP intervals, the control outputs (CTLn) are defined by the GPIF instruction to be either 1, 0, or tristated during the entire interval. These types of intervals have a programmable fixed duration in units of XCLK cycles.

For write waveforms, the data bus is either driven or tristated during the interval.

For read waveforms, the data bus is either sampled and stored as the read data or not sampled during the interval.

Figure 8-3 below illustrates the basic concept for NDP intervals. A write waveform is shown, and for simplicity all the intervals are shown with equal spacing. There are a total of six programmable outputs, but only one (CTL 0) is shown in the *Figure 8-3*.

Remember that the 4-byte Waveform Descriptor defines the characteristics of each interval. For a detailed definition of the 4-byte Waveform Descriptor, see *Section 8.5.5.5. "Waveform Selector"*.

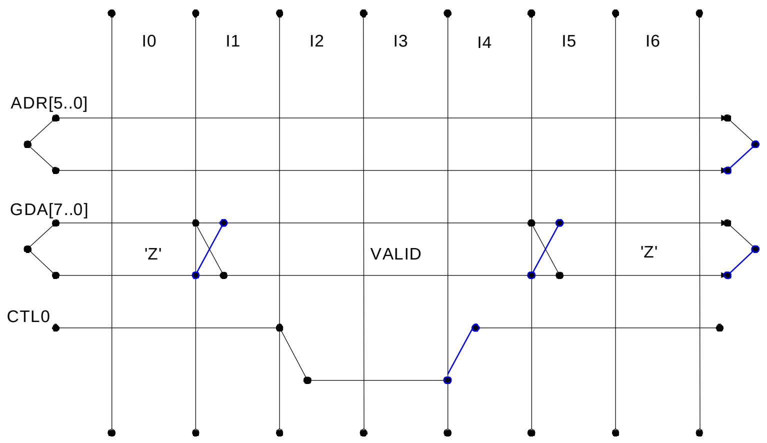


Figure 8-3. Non-Decision Point (NDP) Intervals

The following paragraphs describe *Figure 8-3*,

In interval I0:

- GDA[7..0] is programmed to be tri-stated
- CTL0 is programmed to be driven to a logic 1.

In interval I1:

- GDA[7..0] is programmed to be driven.
- CTL0 is still programmed to be driven to a logic 1.

In interval I2:

- GDA[7..0] is programmed to be driven.
- CTL0 is programmed to be driven to a logic 0.

In interval I3:

- GDA[7..0] is programmed to be driven.
- CTL0 is still programmed to be driven to a logic 0.

In interval I4:

- GDA[7..0] is programmed to be driven.
- CTL0 is programmed to be driven to a logic 1.

In interval I5:

- GDA[7..0] is programmed to be tri-stated
- CTL0 is still programmed to be driven to a logic 1.

In interval I6:

- GDA[7..0] is programmed to be tri-stated
- CTL0 is still programmed to be driven to a logic 1.

Since all intervals in this example are coded as NDP intervals, the GPIF automatically branches from the last interval (I6) to interval 7, the IDLE state. This is the state in which the GPIF waits until the next GPIF waveform is started by the 8051.

8.5.3.2.2 Decision Point (DP) Intervals

The second type of interval is the Decision Point Interval. Any interval can be designated as a DP interval. A DP interval allows the GPIF engine to sample one or more of the RDY inputs (or other internal signals) and branch to other intervals based on the current value.

With a decision point interval, the GPIF can perform simple tasks such as wait until a RDY line is low before continuing to the next interval. Decision point intervals can also be more complex by branching to one interval if the sampled signals result in a logic 1, and a different interval if they result in a logic 0.

In a DP interval, the user specifies which two ‘signals’ to sample. The two ‘signals’ can be selected from any of:

- six external RDY signals
- programmable FIFO flags
- or the INTERNAL_READY Bit in the READY Register.

The user then specifies a logic function (AND, OR, or XOR) to apply to the two selected signals. To select only one signal, simply select the same signal twice and specify the logic function as AND.

In the Waveform Descriptor for the DP interval, the user then specifies which interval to branch to if the resultant logic expression is a 0, and which interval to branch to if the resultant logic expression is a 1.

Below is an example waveform created using one decision point interval (I1) and non-decision point intervals for the rest of the waveform.

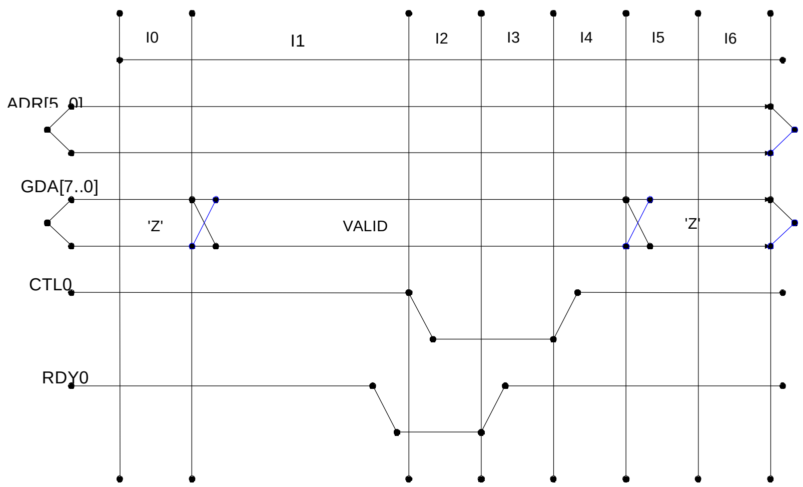


Figure 8-4. One Decision Point: Wait States Inserted Until RDY0 Goes Low

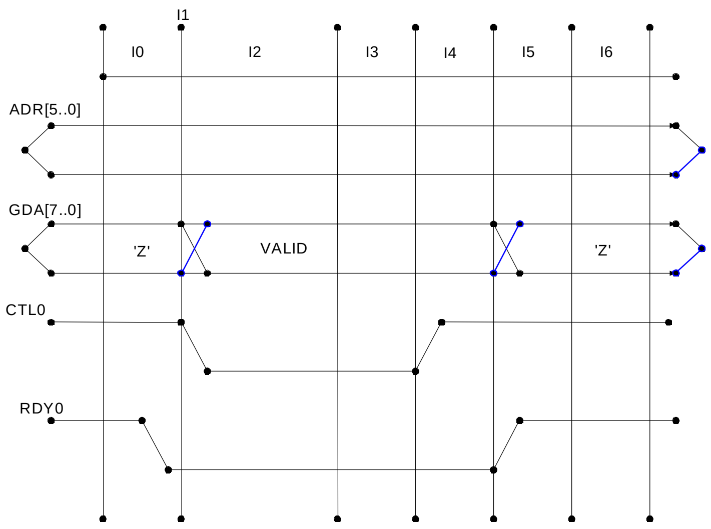


Figure 8-5. One Decision Point: No Wait States Inserted: RDY0 is Already Low at Decision Point I1

In *Figure 8-4* and *Figure 8-5*, there is a single decision point defined as interval I1. In this example, the input ready signal is assumed to be connected to RDY0, and the Waveform Descriptor for I1 is configured to branch to interval I2 if RDY0 is a logic 0 or to branch to interval I1 (wait indefinitely) if RDY0 is a logic 1.

In *Figure 8-4*, the interface remains in I1 until the RDY0 signal is asserted.

In *Figure 8-5*, the RDY0 signal is 0 when I1 is reached, so the GPIF branches to I2.

8.5.3.3 Interval Waveform Descriptor

Each interval must be defined to perform your desired interface. To do this, each GPIF Interval has a Waveform Descriptor made up of a 4-byte word that defines the interval's characteristics.

The four bytes that make up a single interval's Waveform Descriptor are:

- LENGTH/BRANCH
- OPCODE
- LOGIC FUNCTION
- OUTPUT.

Notice that there are 2 definitions for the Waveform Descriptors depending on whether the interval is a decision point (DP = 1) or a non-decision point (DP = 0).

8.5.3.3.1 Non-Decision Point Waveform Descriptor

LENGTH/BRANCH

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Number of XCLK cycles to stay in this interval (0 means 256 cycles)							

OPCODE

7	6	5	4	3	2	1	0
x	x	x	GINT	INCAD	NEXT	DATA	DP = 0

LOGIC FUNCTION

7	6	5	4	3	2	1	0
Not Used							

OUTPUT (if TRICTL Bit = 1)

7	6	5	4	3	2	1	0
OE3	OE2	OE1	OE0	CTL3	CTL2	CTL1	CTL0

OUTPUT (if TRICTL Bit = 0)

7	6	5	4	3	2	1	0
x	x	CTL5	CTL4	CTL3	CTL2	CTL1	CTL0

8.5.3.3.2 Decision Point Waveform Descriptor

LENGTH/BRANCH

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	x	1BRANCH			0BRANCH		

OPCODE

7	6	5	4	3	2	1	0
x	x	x	GINT	INCAD	NEXT	DATA	DP = 1

LOGIC FUNCTION

7	6	5	4	3	2	1	0
LFUNC		TERMA			TERMB		

OUTPUT (if TRICTL Bit = 1)

7	6	5	4	3	2	1	0
OE3	OE2	OE1	OE0	CTL3	CTL2	CTL1	CTL0

OUTPUT (if TRICTL Bit = 0)

7	6	5	4	3	2	1	0
x	x	CTL5	CTL4	CTL3	CTL2	CTL1	CTL0

The following paragraphs describe each of the Waveform Descriptor registers and the fields within each register.

OPCODE Register: This register sets a number of interval characteristics.

DP Bit: indicates whether the interval is a:

1 = Decision Point interval.
0 = Non-Decision Point interval.

DATA Bit: specifies what is to be done with the data bus during this interval.

During a write:
1 = drive the bus with the output data.
0 = tri-state (don’t drive the bus).

During a read:
1 = sample the data bus and store the data.
0 = don’t sample the data bus.

NEXT Bit: at the beginning of this interval, change the output data to the next byte from the outputs FIFO.

1 = move the next data in the output FIFO to the top
0 = do not advance the FIFO

INCAD Bit: specifies whether to increment the GPIF Address lines ADR0..5.

1 = increment the ADR0..ADR5 bus at the beginning of this interval.
0 = do not increment the ADR0..ADR5 signals.

GINT Bit: specifies whether or not to generate a GPIF interrupt to the 8051 during this interval.

1 = generate 8051 GPIF interrupt when this interval is reached.
0 = do not generate interrupt.

OUTPUT Register: This register controls the state of the 6 control outputs (CTL5-0) during the entire interval corresponding to this Waveform Descriptor. For the following example, refer to *8.5.10 "CTLOUTCFG Register"*.

OEn Bit: specifies if the corresponding CTL output signal is tristated.

1 = drive CTLn:

- If the CTLn Bit in the CTLOUTCFG Register is set to 1, the output driver will be and open-drain.
- If the CTLn Bit in the CTLOUTCFG Register is set to 0, the output driver will be driven to CMOS levels.

0 = Tri-state CTLn.

CTLn Bit: specifies the state to set each CTL signal to during this entire interval.

1 = high level
0 = low level

LOGIC FUNCTION Register: This register is used only for decision point Waveform Descriptors. It specifies the inputs (TERMA and TERMB) and the Logic Function (LFUN) to apply to those inputs. Together these define which Branch to take in the LENGTH/BRANCH Register.

TERMA and **TERMB Bits**:

= 000 : RDY0
= 001 : RDY1
= 010 : RDY2
= 011 : RDY3
= 100 : RDY4
= 101 : RDY5
= 110 : Internal programmable FIFO flag
= 111 : INTERNAL_READY (Bit 7 of the READY Register)

LFUN:

= 00 : perform logical AND on selected inputs
= 01 : perform logical OR on selected inputs
= 10 : perform logical XOR on selected inputs
= 11 : perform logical XOR on selected inputs

The Inputs are sampled at each rising edge of the GPIF Clock (XCLK).

The resultant logic function (R_LFUN) is then tested by the GPIF engine to determine which instruction to branch to (see the /LENGTH/BRANCH Register below) before the next rising edge of the GPIF Clock.

This register is meaningful only for DP Instructions, that is, when the DP Bit of the OPCODE Register = 1. (When DP = 0, this register is a *don't care.*)

LENGTH / BRANCH: This register has two different meanings depending on whether the instruction is a decision point or a non-decision point Interval:

- For DP = 0 this is a LENGTH field — the fixed duration of this Interval in units of XCLK (48MHz for internal clocking, XCLK if externally clocked). A value of 0 means an interval length of 256 clocks.
- For DP = 1 this is a BRANCH field - determines the next interval to branch to.

1BRANCH: specifies which interval to branch to if the logic expression equates to a 1. 0 to 6, or 7 (IDLE)

0BRANCH: specifies which interval to branch to if the logic expression equates to a 0. 0 to 6, or 7 (IDLE)

Decision Point Example

To make the GPIF insert wait states (wait for a RDY signal), a reasonable practice is to code a Waveform Descriptor that branches to itself if R_LFUN = 0, and to branch to the next instruction if R_LFUN = 1. This has the effect of extending the Interval (inserting wait states) indefinitely, until the R_LFUN that is programmed is 1.

8.5.3.4 Physical Structure of the Waveform Memories

Up to four different Waveforms can be defined at one time. Each Waveform is made up of up to 7 Waveform Descriptors and loaded into the Waveform Memory as defined in this section. Each Waveform Memory is 32 bytes long, so the 4 Waveform Memories together are 128 bytes long.

Table 8-5. Waveform Memory Types

Waveform Memory	8051 Base XDATA Address
0	0x7900
1	0x7920
2	0x7940
3	0x7960

Within each Waveform Memory, the Waveform Descriptors are packed as described in *Table 8-6, "Waveform Memory Descriptors"*. Waveform Memory 0 is shown as an example. The other Waveform Memories follow exactly the same structure but at higher XDATA addresses.

Table 8-6. Waveform Memory Descriptors

XDATA Address	Contents
0x7900	LENGTH/BRANCH[0] (LENGTH/BRANCH field of Instruction 0 of Waveform Program 0)
0x7901	LENGTH/BRANCH[1] (LENGTH/BRANCH field of Instruction 1 of Waveform Program 0, etc.)
0x7902	LENGTH/BRANCH[2]
0x7903	LENGTH/BRANCH[3]
0x7904	LENGTH/BRANCH[4]
0x7905	LENGTH/BRANCH[5]
0x7906	LENGTH/BRANCH[6]
0x7907	Unused (the Idle state)
0x7908	OPCODE[0] (OPCODE field of Instruction 0 of Waveform Program 0)
0x7909	OPCODE[1] (OPCODE field of Instruction 1 of Waveform Program 0, etc.)
0x790A	OPCODE[2]
0x790B	OPCODE[3]
0x790C	OPCODE[4]
0x790D	OPCODE[5]
0x790E	OPCODE[6]
0x790F	Unused (the Idle state)
0x7910	OUTPUT[0] (OUTPUT field of Instruction 0 of Waveform Program 0)
0x7911	OUTPUT[1] (OUTPUT field of Instruction 1 of Waveform Program 0, etc.)
0x7912	OUTPUT[2]
0x7913	OUTPUT[3]
0x7914	OUTPUT[4]
0x7915	OUTPUT[5]
0x7916	OUTPUT[6]
0x7917	Unused (the Idle state)
0x7918	LOGIC FUNCTION[0] (LOGIC FUNCTION field of Instruction 0 of Waveform Program 0)
0x7919	LOGIC FUNCTION[1] (LOGIC FUNCTION field of Instruction 1 of Waveform Program 0, etc.)
0x791A	LOGIC FUNCTION[2]
0x791B	LOGIC FUNCTION[3]
0x791C	LOGIC FUNCTION[4]
0x791D	LOGIC FUNCTION[5]
0x791E	LOGIC FUNCTION[6]
0x791F	Unused (the Idle state)

8.5.4 Starting GPIF Waveform Transactions

Important: *The following sections are critical to understanding how the GPIF works.*

The 8051 can trigger four types of GPIF Transactions:

- Single Read Transactions
- Single Write Transactions
- FIFO Read Transactions
- FIFO Write Transactions.

Single Transactions produce a single data transfer using one of the GPIF Waveforms you have designed. Single Transactions are typically used to access a control register of a device connected to the GPIF.

FIFO Transactions involve the A and B FIFOs. Multiple bytes of data can be written to a FIFO and a transaction can be started that transfers all of the data using the GPIF Waveforms you have designed. FIFO Transactions are typically used to move bursts of data to or from a device connected to the GPIF.

8.5.4.1 Performing a Single Read Transaction

To perform a Single Read Transaction, follow these steps:

1. Program the 8051 to initialize the GPIF registers and Waveform Descriptors.
2. Program the 8051 to **read** the **SGL**DAT**L**TRIG Register to start a single transaction (using the movx instruction).
3. Program the 8051 to wait for the GPIF to indicate that the transaction is complete.

   A Transaction is complete when either DONE = 1 (in the IDLE_CS Register), or a GPIF Complete interrupt is generated.

4. Depending on the Bus Width Mode and the programmer's desire to start another Transaction, the 8051 can retrieve the data from the **SGL**DAT**H**, **SGL**DAT**L**TRIG, and/or the **SGL**DAT**LN**-TRIG Register.

   In 16-bit mode **only**, the 8051 reads the most significant byte of data from the **SGL**DAT**H** Register.

   In 8- and 16-bit modes, the 8051 reads the least significant byte of data by either:

   - reading **SGL**DAT**L**TRIG, which reads the least significant byte and starts another Single Read Transaction.
   - reading **SGL**DAT**LN**TRIG, which reads the least significant byte but does **not** start another Read Transaction.

The following C program fragment illustrates how to perform a single Read transaction in 8-bit mode:

```
// Declare some byte-wide variables.
unsigned char dummy, inDataLow, inDataHigh ;

// Initiate a (previously set up) GPIF Read transaction by reading
// the SGLDATLTRIG Register (into the variable dummy).

dummy = SGLDATLTRIG;
// Note: we are not yet ready to get the data; this register read
// merely initiates the Read transaction by starting a GPIF
// microcode routine.


// If the GPIF microcode can take longer than one 8051 instruction,
// we must wait for it to complete.
// Otherwise, when reading the data below, we would be reading it
// prematurely, and it would probably be garbage.

while (IDLE_CS & 0x80)              // Spin until the 0x80 bit of
   ;                                // the IDLE_CS Register is
                                    // cleared.

// Get the LS data byte, but do not start a new Read transaction.
inDataLow  = SGLDATLNTRIG;


// For 16-bit mode, we would need to add the following line to
// get the MS data byte.
inDataHigh = SGLDATH;
```

8.5.4.2 Performing a Single Write Transaction

Single Write Transactions are simpler than Single Read Transactions. To run a Single Write Transaction perform the following steps:

1. Program the 8051 to initialize the GPIF registers and Waveform Descriptors.
2. Program the 8051 to **write** the **SGL**DAT**L**TRIG Register to start a single transaction (using the movx instruction).
   - In 16-bit mode, the most significant byte of the data is first written to the**SGL**DAT**H** Register.
   - In both 8- and 16-bit modes, the 8051 starts a Single Write Transaction by **writing** to the **SGL**DAT**L**TRIG Register.
3. Program the 8051 to wait for the GPIF to indicate the transaction is complete.

   A Transaction is complete when either DONE = 1 (in the IDLE_CS Register) or a GPIF Complete interrupt is generated.

8.5.5 GPIF FIFO Transactions

GPIF FIFO transactions allow single or burst accesses to the slave FIFOs. (See *Chapter 7. "EZ-USB FX Slave FIFOs"*).'

***Important:** For GPIF transactions only, the GPIF is master to the slave FIFOs.*

- From the FIFOs point of view, the GPIF is the master that an external FIFO master circuit would have been.
- From the point of view of the GPIF, the GPIF is the master of the slave FIFOs.

8.5.5.1 The GPIF_PF Flag

The slave FIFOs have four Programmable Flags:

- AINPFPIN
- AOUTPFPIN
- BINPFPIN
- BOUTPFPIN

One of these flags is selected to control a given GPIF transaction. The selected flag is called the GPIF_PF flag.

The GPIF_PF flag is selected by the context of the **trigger** for the GPIF transaction.

Table 8-7. Selecting the GPIF_PF Flag

If the GPIF transaction is triggered by . . .	**. . . then the GPIF_PF flag is the**
Reading ATRIG	AINPF flag
Writing ATRIG	AOUTPF flag
Reading BTRIG	BINPF flag
Writing BTRIG	BOUTPF flag

8.5.5.2 Performing a FIFO Read Transaction

To perform a FIFO Read Transaction, do the following:

1. Program the 8051 to either **read** the ATRIG or the BTRIG Register, depending on which FIFO is to receive the data.
2. Program the 8051 to detect completion of the Transaction. As with all GPIF Transactions, Bit 7 of the IDLE_STATE Register (the DONE bit) signals the completeness of the Transaction. (See Step 2 of Section 8.5.4.1. *"Performing a Single Read Transaction"*.)
3. Program the 8051 to move the data from the FIFO to an endpoint (or other 8051-accessible memory) by doing either:

    - direct I/O
    - setting up and starting a DMA transfer.

8.5.5.3 Performing a FIFO Write Transaction

To perform a FIFO Write Transaction, do the following:

1. Program the 8051 to move the data from an endpoint (or other 8051-accessible memory) to the desired FIFO (A or B) by doing either:

    - direct I/O
    - setting up and starting a DMA transfer.

2. Program the 8051 to either **write** the ATRIG or the BTRIG Register, depending on which FIFO has received the data in Step 1.
3. Program the 8051 to detect completion of the Transaction. As with all GPIF Transactions, Bit 7 of the IDLE_STATE Register (the DONE bit) signals the completeness of the Transaction (See Step 2 of Section 8.5.4.1. *"Performing a Single Read Transaction"*.)

8.5.5.4 Burst FIFO Transactions

There are two methods to set up the GPIF to automatically start the next transaction when the previous transaction has finished, repeatedly running Transactions that fetch (in 8-bit mode) a byte at a time. The two methods for controlling and terminating Burst Transactions (always involving FIFOs) are:

- using a FIFO’s GPIF_PF flag
- using a Transaction Count.

Bit 7 of the transaction count registers determine which method to use for the appropriate operation.

There are four Burst Transaction types:

- AIN
- AOUT
- BIN
- BOUT.

Each FIFO Transaction Type is controlled by its own _TC Register.

For example, to perform a burst of reads from a device into the AFIFO using the GPIF_PF flag:

1. Program the 8051 to poll the GPIF_PF flag for the doneness of the Burst of Transactions.
2. Set AIN_TC.7 (FITC Bit) to 1. This tells the GPIF to continue executing read transactions involving the AFIFO until the GPIF_PF flag is set.
3. To start the series of Transactions, read the GPIF_FIFOA_EXEC Register. The Transaction runs. The GPIF runs repeated Transactions. The AFIFO fills with data.
4. As with all GPIF Transactions, the IDLE_CS.7 Bit (the DONE Bit) signals the completion of the Transaction. (See Step 2 of Section 8.5.4.1. *"Performing a Single Read Transaction"*.) In this case the DONE Bit is set by the GPIF Engine when the AFIFO fills the waterline measured by the GPIF_PF flag.

8.5.5.5 Waveform Selector

WF_SELECTOR			Waveform Selector				7824
b7	b6	b5	b4	b3	b2	b1	b0
SNGL_WR_WF		**SNGL_RD_WF**		**FIWR_WF**		**FIRD_WF**	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	0	0	1	0	0

Bits 7-6: **SNGL_WR_WF** *Single Write Waveform Index*

Index to the Waveform Program to run when a “Single Write” is triggered by the 8051.

Bits 5-4: **SNGL_RD_WF** *Single Read Waveform Index*

Index to the Waveform Program to run when a “Single Read” is triggered by the 8051.

Bits 3-2: **FIWR_WF** *FIFO Write Waveform Index*

Index to the Waveform Program to run when a “FIFO Write” is triggered by the 8051.

Bits 1-0: **FIRD_WF** *FIFO Read Waveform Index*

Index to the Waveform Program to run when a “FIFO Read” is triggered by the 8051.

The default for this register is **11 10 01 00**, which points each waveform index to the next Wave-form Memory. In most applications, it is unnecessary to ever access the WF_SELECTOR Register, provided the Waveform Memories are loaded with Waveform Programs in a logical way. (0 is loaded with *FIFO Read*, 1 is loaded with *FIFO Write*, etc.).

Here is an example:

If: the 0th Waveform Memory is loaded with a Waveform Program (*Waveform0*) and the WF_SELECTOR Register is never accessed (all fields set to their defaults)

Then *Waveform0* runs when FIFO Read is triggered by the 8051.

8.5.6 Data/Trigger Registers

SGLDATH			GPIF Data H (16-Bit Mode Only)				7834
b7	b6	b5	b4	b3	b2	b1	b0
D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Bits 7-0: Data to/From GDB[7..0]

16-bit mode: contains the data that gets written to the GDB]7..0] pins

8-bit mode: not used

SGLDATLTRIG			Triggers GPIF Waveform				7835
b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Bits 7-0: Data To/From GDA[7..0]

Execute the single transaction in Waveform Memory.

Writing to this register causes a Single Write transaction to occur. The data sent to the peripheral is the data written to this register.

Reading this register causes a Single Read transaction to occur. The data appears in this register after the read. To avoid triggering an additional transaction, use the SGLDATLNTRIG Register instead of this one.

To avoid triggering an **additional** read transaction, use the SGLDATLTRIG Register instead of this one.

SGLDATLNTRIG			NO GPIF Waveform Trigger				7836
b7	b6	b5	b4	b3	b2	b1	b0
D7	D6	D5	D4	D3	D2	D1	D0
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

Bits 7-0: Data To/From GDA[7..0]

This register contains the data that was read, and mirrors the data in the SGLDATLTRIG Register.

Using this register does **Not** cause additional read transactions to take place, unlike the SGLDATLTRIG Register.

8.5.7 FIFO Operation Trigger Registers

There are two registers that initiate execution of the GPIF state machine using the FIFOs: ATRIG and BTRIG.

Writing to the ATRIG Register causes a write transaction involving A- FIFO. Reading the ATRIG Register causes a read transaction involving A-FIFO.

Similarly, write and read operations involving B-FIFO are initiated by write and read operations to the BTRIG Register.

For information on how the GPIF_PF flag is determined by the FIFO trigger Register, which starts the transactions(s), see *Section 8.5.5.1. "The GPIF_PF Flag".*

.

ATRIG			**Single Write Transaction of A FIFO**				**782E**
b7	b6	b5	b4	b3	b2	b1	b0
Reserved	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**
W	W	W	W	W	W	W	W
x	x	x	x	x	x	x	x

BTRIG			**Single Write Transaction of B FIFO**				**7832**
b7	b6	b5	b4	b3	b2	b1	b0
Reserved	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**
W	W	W	W	W	W	W	W
x	x	x	x	x	x	x	x

Bits 7-0: Single Read/Write FIFO

8.5.8 Transaction Count Registers

Four registers contain the Transaction Counts for FIFO Transactions:

Table 8-8. Addresses of Transaction Count Registers

XDATA Address	Register Name	Description
0x782C	AIN_TC	AIN FIFO Transaction Count
0x782D	AOUT_TC	AOUT FIFO Transaction Count
0x7830	BIN_TC	BIN FIFO Transaction Count
0x7831	BOUT_TC	BOUT FIFO Transaction Count

Each Transaction Count Register has two bit fields:

AIN_TC, AOUT_TC, BIN_TC, BOUT_TC			**Bit Fields of Transaction Count Registers**				**782C, 782D, 7830, 7831**
b7	b6	b5	b4	b3	b2	b1	b0
FITC	**FIFO A in Transaction Count**						
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	1

Bits 7: **FITC** *FIFO T.C.*

1 = Use FIFO GPIF_PF flag.

0 = Use Transaction Count

Bits 6-0: **Transaction Count** *Transaction Count Number*

For information on how the GPIF_PF flag is determined by the FIFO trigger Register, which starts the transactions(s), see *Section 8.5.5.1. "The GPIF_PF Flag".*

8.5.9 READY Register

READY			Internal RDY, Sync/ Async, RDY Pin States				7838
b7	b6	b5	b4	b3	b2	b1	b0
INTRDY	SAS	RDY5	RDY4	RDY3	RDY2	RDY1	RDY0
B	B	R	R	R	R	R	R
1	1	1	0	0	1	0	0

Figure 8-6. Ready Register

Bits 7: **INTERNAL_RDY** *Force Ready Condition*

Bit to allow the 8051 to force a RDY condition to control a Waveform Instruction.

This bit is writable by the 8051. It is one of the bits that can be selected by a DP Instruction to feed TERM_A or TERM_B of a DP Instruction's LOGIC FUNCTION.

Bits 6: **SAS** *Bypass Double Sampling of RDY*

To achieve improved performance, use this bit to bypass double-sampling of synchronous RDY inputs.

1 = Assume RDY[5:0] inputs are synchronous to the GPIF Clock, hence are sampled at only **one** rising edge of the GPIF Clock.

0 = Re-synch RDY[5:0] inputs by double-sampling at **two** rising edges of the GPIF Clock.

Bits 5-0: **RDY_IN[5:0]** *Current State RDY Pins*

The current state of the RDY[5:0] pins, sampled at each rising edge of the GPIF Clock.

8.5.10 CTLOUTCFG Register

CTL_OUT_CFG is a semi-static register (normally set it and forget it) that controls the configuration of the CTL[5:0] outputs.

CTLOUTCFG			**CTL Out Pin Drive**				**7827**
b7	b6	b5	b4	b3	b2	b1	b0
TRICTL	**0**	**CTL5**	**CTL4**	**CTL3**	**CTL2**	**CTL1**	**CT0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bits 7: **TRICTL** *Number active outputs/tristating*

This bit controls how many of the CTL[5:0] outputs are active (4 or 6), and whether those outputs can be tristated. (Tri-stating is only available only if using 4 CTL outputs).

This bit controls the meaning of the OUTPUT field in *all* currently loaded Waveform Programs.

If this bit is 1, Use only the **four** CTL[3:0] pins, where each of the four Outputs can be either driven or tri-stated.

OUTPUT[7:4] of each Waveform Instruction:

- if a bit is 1, tri-state the corresponding CTL[3:0] output.
- if a bit is 0, drive the corresponding CTL[3:0] output.

OUTPUT[3:0] of each Waveform Instruction. Contains the **value** of the corresponding four CTL[3:0] output.

If this bit is 0: Use all six CTL_OUTx signals, none tri-stateable.
See the discussion of the OUTPUT field in *Section 8.5.10. "CTLOUTCFG Register".*

Bits 6: **Reserved** *Reserved*

Bits 5-0: **CTL 5-0** *Active CTL bits*

Configuration of the 6 programmable CTL[5:0] pins.

8.5.11 IDLE State Registers

The IDLE_OUT_CTL and IDLE_CS Registers define the control and data signals during the **IDLE state**, i.e., when the GPIF is not currently involved in a bus transaction, between transactions.

IDLE_CS			**GPIF Done, GPIF IDLE drive mode**				**7825**
b7	b6	b5	b4	b3	b2	b1	b0
DONE	**0**	**0**	**0**	**0**	**0**	**0**	**IDLEDRV**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	0	0	0	0	0	0	0

Bits 7: **DONE** *GPIF Idle State*

0 = Transaction in progress.

1 = Transaction Done (GPIF is idle, hence GPIF is ready for next Transaction). Fires IRQ4 if enabled.

Bits 6-1: **Reserved** *Reserved*

Bits 0: **IDLEDRV** *Set Data Bus when GPIF Idle*

When the GPIF is idle:

0 = Tri-state the Data Bus.

1 = Drive the Data Bus.

IDLE_CTLOUT			**Inactive Bus, CTL States**				**7826**
b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**CTL5**	**CTL4**	**CTL3**	**CTL2**	**CTL1**	**CTL0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	1	1	1	1	1

Figure 8-7. IDLE_CTLOUT 0x7826 Register

Bits 7-6: **Reserved** *Reserved*

Bits 5-0: **CTL[5..0]** *States of CTLOUT signals*

Meaning depends on CTL_OUT_CFG[7]:

If CTL_OUT_CFG[7] == 1

IDLE_CTLOUT[7:4] controls **whether to drive or tri-state** outputs defined in IDLE_CTLOUT[3:0] during the bus IDLE state.

If CTL_CTLOUT[7] == 0:

IDLE_CTLOUT[7:6] = RESERVED = 00

IDLE_CTLOUT[5:0] : Contains the **values** 0 or 1 for the 6 programmable outputs during the bus IDLE state.

The IDLE_CTLOUT and IDLE_CS Registers define the control and data signals during the IDLE state, i.e., when the GPIF is not currently involved in a bus transaction, between transactions.

Important: *the IDLE_OUT_CTL Register contains the default states of the CTL_OUTx signals.*

The IDLE_CS Register contains the 'done' indication that tells the user that it is OK to go ahead with another bus transaction. It also knows whether the data bus is tri-stated or driven when the bus is not currently in use.

Note well: *It is illegal to initiate any operation (except aborting the current transaction) when the 'done' Bit (IDLE_CS[7]) is 0. Doing so yields indeterminate behavior, likely to cause data corruption.*

8.5.12 Address Register GPIFADRL

GPIFADRL			GPIF Address Bus PIns				782A
b7	b6	b5	b4	b3	b2	b1	b0
Reserved	**Reserved**	**ADR5**	**ADR4**	**ADR3**	**ADR2**	**ADR1**	**ADR0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bits 7-6: **Reserved** *Reserved*

Bits 5-0: **ADR[5..0]** *GPIF Address Bus*

Data written to the Register appears as the bus address on the ADR[5:0] pins during a GPIF transaction.

This address does not change until a GPIF program is running. Specifically, the new value does not appear on the Address Bus pins until one of the following is written:

- SGLDATLTRIG Register
- ATRIG Register
- BTRIG Register.

8.5.13 GPIF_ABORT Register

GPIF_ABORT			Abort GPIF Cycles				7839
b7	b6	b5	b4	b3	b2	b1	b0
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
W	W	W	W	W	W	W	W
x	x	x	x	x	x	x	x

Figure 8-8. GPIF Abort Register

Bits 7-0: **D7-D0** *Go GPIF IDLE state. Data is D.C.*

Writing to the GPIF_ABORT Register aborts the current transaction on the bus, returns the DONE Bit to 1, and causes the outputs to go to the state as defined in the IDLE_STATE Register. This is useful in debugging, and in systems that have a bus time-out or watchdog timer.

Chapter 9. EZ-USB FX Endpoint Zero

9.1 Introduction

Endpoint Zero has special significance in a USB system. It is a CONTROL endpoint, and it is required by every USB device. Only CONTROL endpoints accept special SETUP tokens that the host uses to signal transfers that deal with device control. The USB host sends a suite of standard device requests over endpoint zero. These standard requests are fully defined in Chapter 9 of the *USB Specification*. This chapter describes how the EZ-USB FX chip handles endpoint zero requests.

Because EZ-USB FX must enumerate without firmware (see *Chapter 5. "EZ-USB FX Enumeration & ReNumeration™"*), the USB core contains logic to perform enumeration on its own. This hardware assist of endpoint zero operations is made available to the 8051, simplifying the code required to service device requests. This chapter deals with 8051 control of endpoint zero (RENUM=1, *Chapter 5. "EZ-USB FX Enumeration & ReNumeration™"*), and describes EZ-USB FX resources, such as the Setup Data Pointer, that simplify 8051 code handling endpoint zero requests.

Endpoint zero is the only CONTROL endpoint in EZ-USB FX. Although CONTROL endpoints are *bi-directional*, the EZ-USB FX chip provides separate 64-byte buffers, IN0BUF and OUT0BUF, which the 8051 handles exactly like bulk endpoint buffers for the data stages of a CONTROL transfer. A second 8-byte buffer, SETUPDAT, which is unique to endpoint zero, holds data that arrives in the SETUP stage of a CONTROL transfer. This relieves the 8051 programmer of tracking of the three CONTROL transfer phases—SETUP, DATA, and STATUS. The USB core also generates separate interrupt requests for the various transfer phases, further simplifying code.

The IN0BUF and OUT0BUF Buffers have two special properties that result from being used by CONTROL endpoint zero:

- Endpoints 0-IN and 0-OUT are always valid. The valid bits (LSB of IN07VAL and OUT07VAL Registers) are permanently set to 1. Writing any value to these two bits has no effect. Reading these bits always returns a 1.
- Endpoint 0 cannot be paired with endpoint 1, so there is no pairing bit in the USBPAIR Register for endpoint 0 or 1.

9.2 Control Endpoint EP0

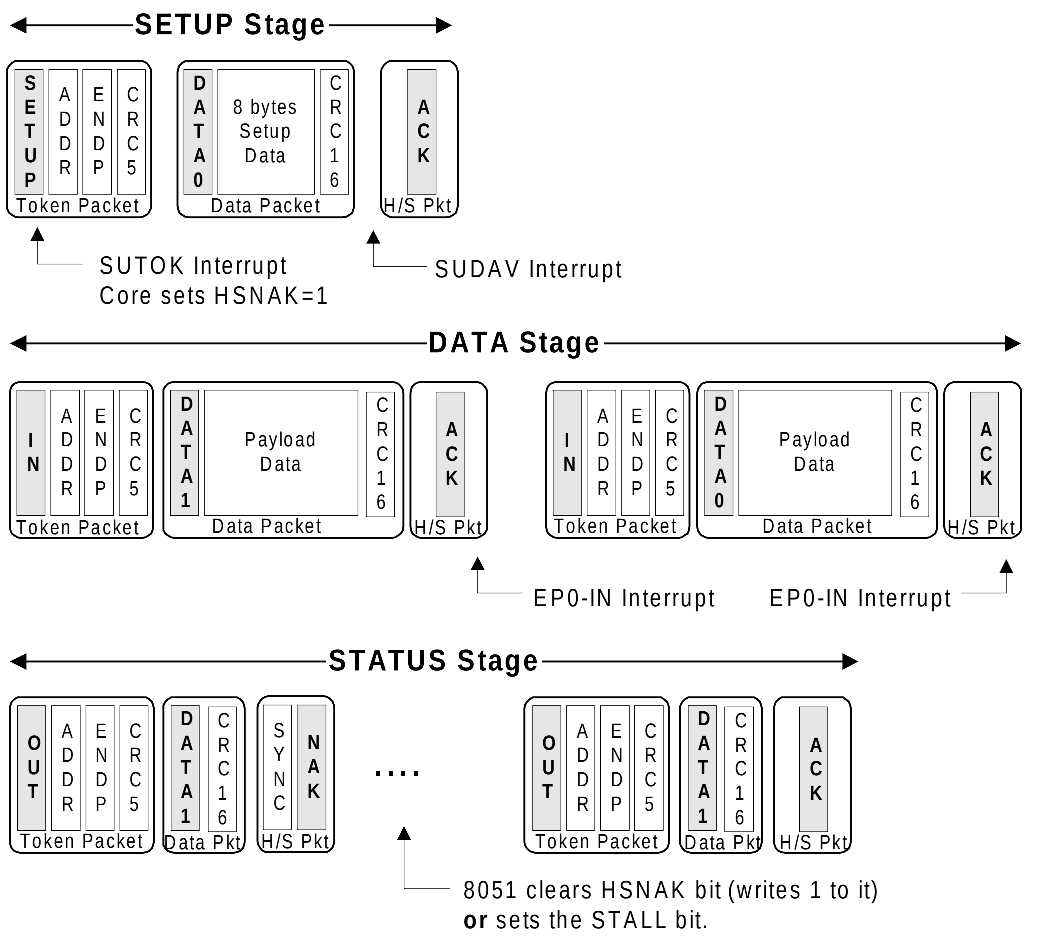


Figure 9-1. A USB Control Transfer (With Data Stage)

Endpoint zero accepts a special SETUP packet, which contains an 8-byte data structure that provides host information about the CONTROL transaction. CONTROL transfers include a final STATUS phase, constructed from standard PIDs (IN/OUT, DATA1, and ACK/NAK).

Some CONTROL transactions include all required data in their 8-byte SETUP Data packet. Other CONTROL transactions require more OUT data than will fit into the eight bytes, or require IN data from the device. These transactions use standard bulk-like transfers to move the data. Note in *Figure 9-1* that the DATA Stage looks exactly like a bulk transfer. As with BULK endpoints, the endpoint zero byte count registers must be loaded to ACK each data transfer stage of a CONTROL transfer.

The STATUS stage consists of an empty data packet with the opposite direction of the data stage, or an IN if there was no data stage. This empty data packet gives the device a chance to ACK or NAK the entire CONTROL transfer. The 8051 writes a “1” to a bit call HSNAK (Handshake NAK) to clear it and instruct the USB core to ACK the STATUS stage.

The HSNAK Bit holds off completing the CONTROL transfer until the device has had time to respond to a request. For example, if the host issues a Set_Interface Request, the 8051 performs various housekeeping chores, such as adjusting internal modes and re-initializing endpoints. During this time, the host issues handshake (STATUS stage) packets to which the USB core responds with NAKs, indicating “busy.” When the 8051 completes the desired operation, it sets HSNAK=1 (by writing a “1” to the bit) to terminate the CONTROL transfer. This handshake prevents the host from attempting to use a partially configured interface.

To perform an endpoint stall for the DATA or STATUS stage of an endpoint zero transfer (the SETUP stage can never stall), the 8051 must set both the STALL and HSNAK Bits for endpoint zero.

Some CONTROL transfers do not have a DATA stage. Therefore, the 8051 code that processes the SETUP data should check the length field in the SETUP data (in the 8-byte buffer at SETUPDAT) and arm endpoint zero for the DATA phase (by loading IN0BC or OUT0BC) only if the length field is non-zero.

Two 8051 interrupts provide notification that a SETUP packet has arrived, as shown in *Figure 9-2*.

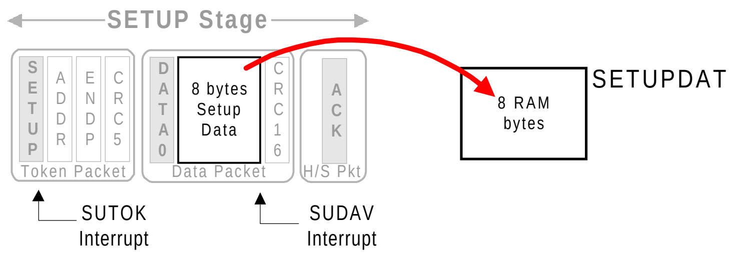


Figure 9-2. Two Interrupts Associated with EP0 CONTROL Transfers

The USB core sets the SUTOKIR Bit (SETUP Token Interrupt Request) when the USB core detects the SETUP token at the beginning of a CONTROL transfer. This interrupt is normally used for debug only.

The USB core sets the SUDAVIR Bit (Setup Data Available Interrupt Request) when the eight bytes of SETUP data have been received error-free and transferred to eight EZ-USB FX Registers starting at SETUPDAT. The USB core takes care of any re-tries if it finds any errors in the SETUP data. These two interrupt request bits are set by the USB core, and must be cleared by firmware.

An 8051 program responds to the SUDAV interrupt request by either directly inspecting the eight bytes at SETUPDAT or by transferring them to a local buffer for further processing. Servicing the

SETUP data should be a high 8051 priority, since the USB Specification stipulates that CONTROL transfers must always be accepted and never NAKd. It is, therefore, possible that a CONTROL transfer could arrive while the 8051 is still servicing a previous one. In this case the previous CONTROL transfer service should be aborted and the new one serviced. The SUTOK interrupt gives advance warning that a new CONTROL transfer is about to over-write the eight SETUPDAT bytes.

If the 8051 stalls endpoint zero (by setting the EP0STALL and HSNAK Bits to 1), the USB core automatically clears this stall bit when the next SETUP token arrives.

Like all EZ-USB FX interrupt requests, the SUTOKIR and SUDAVIR Bits can be directly tested and reset by the CPU (reset by writing a “1”). Thus, if the corresponding interrupt enable bits are set to “0,” the interrupt request conditions can still be directly polled.

Figure 9-3 shows the EZ-USB FX registers that deal with CONTROL transactions over EP0.

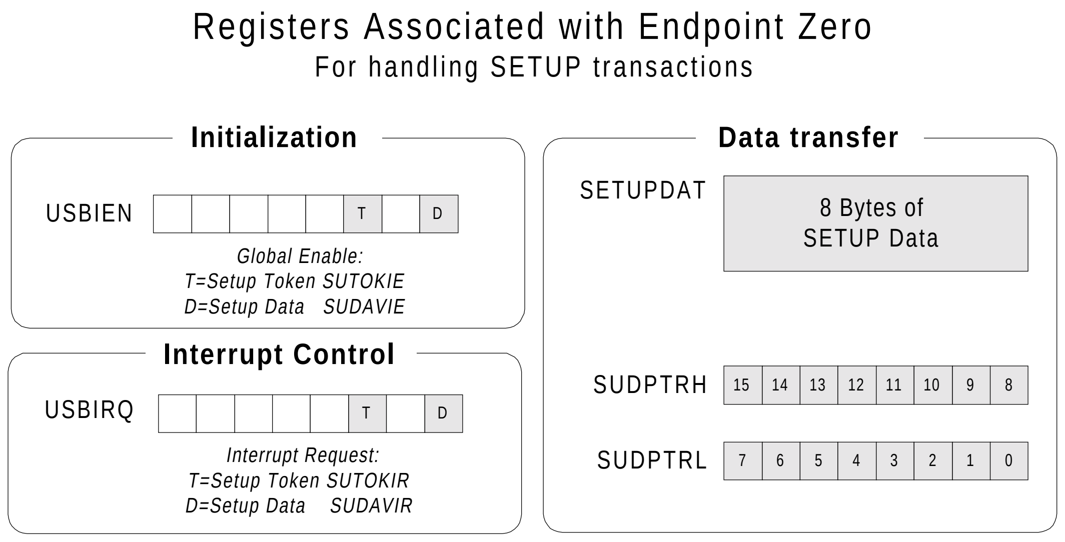


Figure 9-3. Registers Associated with EP0 Control Transfers

These registers augment those associated with normal bulk transfers over endpoint zero, which are described in *Chapter 6. "EZ-USB FX Bulk Transfers"*.

Two bits in the USBIEN (USB Interrupt Enable) Register enable the SETUP Token (SUTOKIE) and SETUP Data interrupts. The actual interrupt request bits are in the USBIRQ (USB Interrupt Requests) Register. They are called STOKIR (SETUP Token Interrupt Request) and SUDAVIR (SETUP Data Interrupt Request).

The USB core transfers the eight SETUP bytes into eight bytes of RAM at SETUPDAT. A 16-bit pointer, SUDPTRH/L provides hardware assistance for handling CONTROL IN transfers, in particular the USB Get_Descriptor Requests, described later in this chapter.

9.3 USB Requests

The *Universal Serial Bus Specification Version 1.1, Chapter 9, "USB Device Framework"* defines a set of *Standard Device Requests*. When the 8051 is in control (RENUM=1), the USB core handles one of these requests (Set Address) directly, and relies on the 8051 to support all of the others. The 8051 acts on device requests by decoding the eight bytes contained in the SETUP packet and available at SETUPDAT (seeTable 9-2). Table 9-1 defines these eight bytes.

Table 9-1. The Eight Bytes in a USB SETUP Packet

Byte	Field	Meaning
0	bmRequestType	Request Type, Direction, and Recipient.
1	bRequest	The actual request (see Table 9-2).
2	wValueL	Word-size value, varies according to bRequest.
3	wValueH	
4	wIndexL	Word-size field, varies according to bRequest.
5	wIndexH	
6	wLengthL	Number of bytes to transfer if there is a data phase.
7	wLengthH	

The **Byte** column in the previous table shows the byte offset from SETUPDAT. The **Field** column shows the different bytes in the request, where the “bm” prefix means bit-map, “b” means byte, and “w” means word (16 bits). Table 9-2 shows the different values defined for bRequest, and how the 8051 responds to each request. The remainder of this chapter describes each of the requests in Table 9-2 in detail.

Table 9-2 applies when RENUM=1, signifying that the 8051, and not the USB core, handles device requests. Table 9-2 shows how the core handles each of these device requests when RENUM=0, for example when the chip is first powered and the 8051 is not running.

Table 9-2. How the 8051 Handles USB Device Requests (RENUM=1)

bRequest	Name	Action	8051 Response
0x00	Get Status	SUDAV Interrupt	Supply RemWU, SelfPwr or Stall Bits
0x01	Clear Feature	SUDAV Interrupt	Clear RemWU, SelfPwr or Stall Bits
0x02	(reserved)	none	Stall EP0
0x03	Set Feature	SUDAV Interrupt	Set RemWU, SelfPwr or Stall Bits
0x04	(reserved)	none	Stall EP0
0x05	Set Address	Update FNADDR Register	none
0x06	Get Descriptor	SUDAV Interrupt	Supply table data over EP0-IN
0x07	Set Descriptor	SUDAV Interrupt	Application dependent
0x08	Get Configuration	SUDAV Interrupt	Send current configuration number
0x09	Set Configuration	SUDAV Interrupt	Change current configuration
0x0A	Get Interface	SUDAV Interrupt	Supply alternate setting No. from RAM
0x0B	Set Interface	SUDAV Interrupt	Change alternate setting No.
0x0C	Sync Frame	SUDAV Interrupt	Supply a frame number over EP0-IN
Vendor Requests			
0xA0 (Firmware Load)		Up/Download RAM	---
0xA1 - 0xAF		SUDAV Interrupt	Reserved by Cypress Semiconductor
All except 0xA0		SUDAV Interrupt	Depends on application

In the ReNumerated condition (RENUM=1), the USB core passes all USB requests, except Set Address, onto the 8051 *via* the SUDAV interrupt. This, in conjunction with the USB disconnect/connect feature, allows a completely new and different USB device (yours) to be characterized by the downloaded firmware.

The USB core implements one vendor-specific request, namely “Firmware Load,” 0xA0. (The bRequest value of 0xA0 is valid only if byte 0 of the request, bmRequestType, is also “x10xxxxx,” indicating a vendor-specific request.) The load request is valid at all times, so even after ReNumeration™ the load feature may be used. If your application implements vendor-specific USB requests, and you do *not* wish to use the Firmware Load feature, be sure to refrain from using the bRequest value 0xA0 for your custom requests. The Firmware Load feature is fully described in *Chapter 5. "EZ-USB FX Enumeration & ReNumeration™"*.

To avoid future incompatibilities, vendor requests A0-AF (hex) are reserved by Cypress Semiconductor.

9.3.1 Get Status

The USB Specification defines three USB status requests. A fourth request, to an interface, is declared in the spec as “reserved.” The four status requests are:

- Remote Wakeup (Device request)
- Self-Powered (Device request)
- Stall (Endpoint request)
- Interface request (reserved).

The USB core activates the SUDAV interrupt request to tell the 8051 to decode the SETUP packet and supply the appropriate status information.

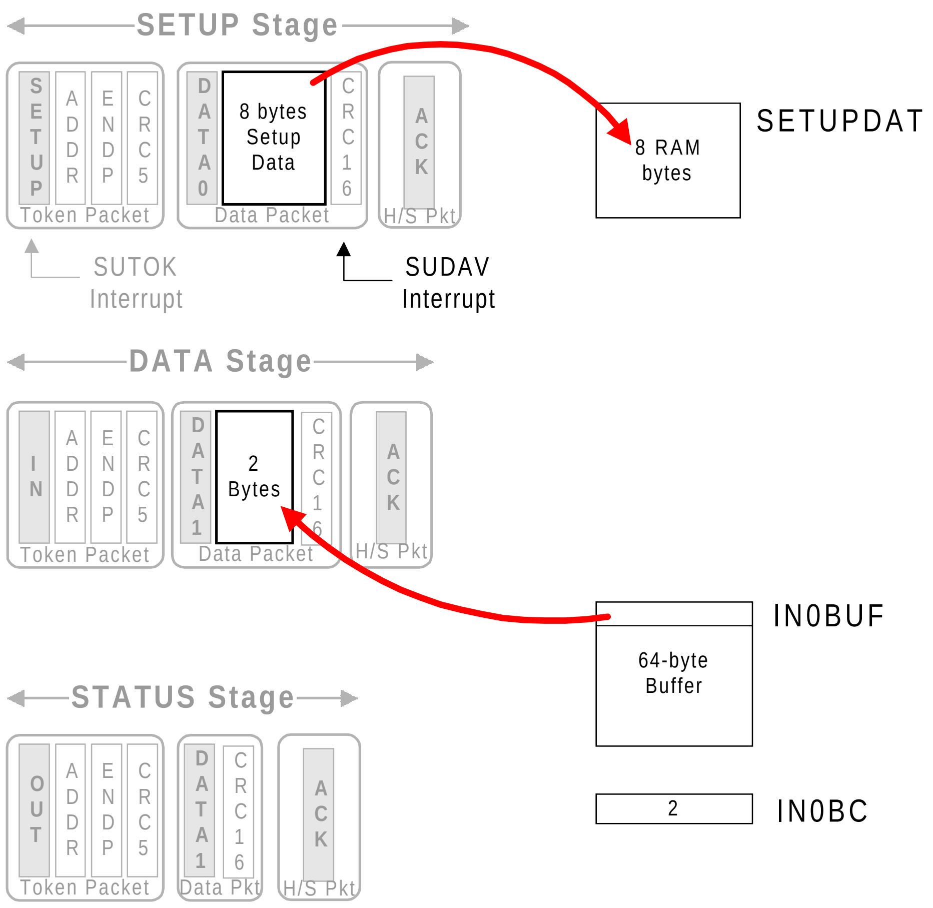


Figure 9-4. Data Flow for a Get_Status Request

As *Figure 9-4* illustrates, the 8051 responds to the SUDAV interrupt by decoding the eight bytes the USB core has copied into RAM at SETUPDAT. The 8051 answers a Get_Status Request (bRequest=0) by loading two bytes into the IN0BUF Buffer and loading the byte count register IN0BC with the value “2.” The USB core transmits these two bytes in response to an IN token. Finally, the 8051 clears the HSNAK Bit (by writing “1” to it) to instruct the USB core to ACK the status stage of the transfer.

The following tables show the eight SETUP bytes for Get_Status Requests.

Table 9-3. Get Status-Device (Remote Wakeup and Self-Powered Bits)

Byte	Field	Value	Meaning	8051 Response
0	bmRequestType	**0x80**	IN, Device	
1	bRequest	**0x00**	"Get Status"	*Load two bytes into IN0BUF*
2	wValueL	0x00		
3	wValueH	0x00		*Byte 0 : bit 0 = Self Powered Bit*
4	wIndexL	0x00		*: bit 1 = Remote Wakeup*
5	wIndexH	0x00		*Byte 1 : zero.*
6	wLengthL	**0x02**	Two bytes requested	
7	wLengthH	**0x00**		

Get_Status-*Device* queries the state of two bits, Remote Wakeup and Self-Powered. The Remote Wakeup Bit indicates whether or not the device is currently enabled to request remote wakeup. Remote wakeup is explained in *Chapter 14. "EZ-USB FX Power Management."* The Self-Powered Bit indicates whether or not the device is self-powered (as opposed to USB bus-powered).

The 8051 returns these two bits by loading two bytes into IN0BUF, and then loading a byte count of two into IN0BC.

Table 9-4. Get Status-Endpoint (Stall Bits)

Byte	Field	Value	Meaning	8051 Response
0	bmRequestType	**0x82**	IN, Endpoint	*Load two bytes into IN0BUF*
1	bRequest	**0x00**	"Get Status"	*Byte 0 : bit 0 = Stall Bit for EP(n)*
2	wValueL	0x00		*Byte 1 : zero*
3	wValueH	0x00		
4	wIndexL	**EP**	Endpoint Number	EP(n):
5	wIndexH	0x00		0x00-0x07: OUT0-OUT7
6	wLengthL	**0x02**	Two bytes requested	0x80-0x87: IN0-IN7.
7	wLengthH	**0x00**		

Each bulk endpoint (IN or OUT) has a STALL Bit in its Control and Status Register (bit 0). If the CPU sets this bit, any requests to the endpoint return a STALL handshake rather than ACK or NAK. The Get Status-Endpoint Request returns the STALL state for the endpoint indicated in byte 4 of the request. Note that bit 7 of the endpoint number EP (byte 4) specifies direction.

Endpoint zero is a CONTROL endpoint, which by USB definition is *bi-directional*. Therefore, it has only one stall bit.

About STALL

The USB STALL handshake indicates that something unexpected has happened. For instance, if the host requests an invalid, alternate setting or attempts to send data to a non-existent endpoint, the device responds with a STALL handshake over endpoint zero instead of ACK or NAK.

Stalls are defined for all endpoint types except ISOCHRONOUS, which do not employ handshakes. Every EZ-USB FX bulk endpoint has its own stall bit. The 8051 sets the stall condition for an endpoint by setting the stall bit in the endpoint's CS Register. The host tells the 8051 to set or clear the stall condition for an endpoint using the Set_Feature/Stall and Clear_Feature/Stall Requests.

An example of the 8051 setting a stall bit is a routine that handles endpoint zero device requests. If an undefined or non-supported request is decoded, the 8051 should stall EP0. (EP0 has a single stall bit because it is a bi-directional endpoint.)

Once the 8051 stalls an endpoint, it should not remove the stall until the host issues a Clear_Feature/Stall Request. An exception to this rule is endpoint 0, which reports a stall condition only for the current transaction, and then automatically clears the stall condition. This prevents endpoint 0, the default CONTROL endpoint, from locking out device requests.

Table 9-5. Get Status-Interface

Byte	Field	Value	Meaning	8051 Response
0	bmRequestType	**0x81**	IN, Endpoint	*Load two bytes into IN0BUF*
1	bRequest	**0x00**	“Get Status”	*Byte 0 : zero*
2	wValueL	0x00		*Byte 1 : zero*
3	wValueH	0x00		
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	**0x02**	Two bytes requested	
7	wLengthH	**0x00**		

Get_Status/Interface is easy: the 8051 returns two zero bytes through IN0BUF and clears the HSNAK Bit. The requested bytes are shown as “Reserved (Reset to zero)” in the USB Specification.

9.3.2 Set Feature

Set Feature is used to enable remote wakeup or stall an endpoint. No data stage is required.

Table 9-6. Set Feature-Device (Set Remote Wakeup Bit)

Byte	Field	Value	Meaning	8051 Response
0	bmRequestType	**0x00**	OUT, Device	*Set the Remote Wakeup Bit*
1	bRequest	**0x03**	“Set Feature”	
2	wValueL	**0x01**	Feature Selector: Remote Wakeup	
3	wValueH	0x00		
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	0x00		
7	wLengthH	0x00		

The only Set_Feature/Device Request presently defined in the USB Specification is to set the remote wakeup bit. This is the same bit reported back to the host as a result of a Get Status-Device Request (Table 9-3). The host uses this bit to enable or disable remote wakeup by the device.

Table 9-7. Set Feature-Endpoint (Stall)

Byte	Field	Value	Meaning	8051 Response
0	bmRequestType	**0x02**	OUT, Endpoint	*Set the STALL Bit for the*
1	bRequest	**0x03**	“Set Feature”	*indicated endpoint:.*
2	wValueL	**0x00**	Feature Selector: STALL	
3	wValueH	0x00		
4	wIndexL	**EP**		EP(n):
5	wIndexH	0x00		0x00-0x07: OUT0-OUT7
6	wLengthL	0x00		0x80-0x87: IN0-IN7
7	wLengthH	0x00		

The only Set_Feature/Endpoint Request presently defined in the USB Specification is to stall an endpoint. The 8051 should respond to this request by setting the stall bit in the Control and Status Register for the indicated endpoint EP (byte 4 of the request). The 8051 can either stall an endpoint on its own or in response to the device request. Endpoint stalls are cleared by the host Clear_Feature/Stall Request.

The 8051 should respond to the Set_Feature/Stall Request by performing the following tasks:

1. Set the stall bit in the indicated endpoint’s CS Register.
2. Reset the data toggle for the indicated endpoint.
3. For an IN endpoint, clear the busy bit in the indicated endpoint’s CS Register.
4. For an OUT endpoint, load any value into the endpoint’s byte count register.
5. Clear the HSNAK Bit in the EP0CS Register (by writing 1 to it) to terminate the Set_Feature/Stall CONTROL transfer.

Steps 3 and 4 restore the stalled endpoint to its default condition, ready to send or accept data after the stall condition is removed by the host (using a Clear_Feature/Stall Request). These steps are also required when the host sends a Set_Interface Request.

Data Toggles

The USB core automatically maintains the endpoint toggle bits to ensure data integrity for USB transfers. The 8051 should directly manipulate these bits only for a very limited set of circumstances:

- Set_Feature/Stall
- Set_Configuration
- Set_Interface

9.3.3 Clear Feature

Clear Feature is used to disable remote wakeup or to clear a stalled endpoint.

Table 9-8. Clear Feature-Device (Clear Remote Wakeup Bit)

Byte	Field	Value	Meaning	8051 Response
0	bmRequestType	**0x00**	OUT, Device	*Clear the remote wakeup bit.*
1	bRequest	**0x01**	“Clear Feature”	
2	wValueL	**0x01**	Feature Selector: Remote Wakeup	
3	wValueH	0x00		
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	0x00		
7	wLengthH	0x00		

Table 9-9. Clear Feature-Endpoint (Clear Stall)

Byte	Field	Value	Meaning	8051 Response
0	bmRequestType	**0x02**	OUT, Endpoint	*Clear the STALL Bit for the*
1	bRequest	**0x01**	“Clear Feature”	*indicated endpoint:.*
2	wValueL	**0x00**	Feature Selector: STALL	
3	wValueH	0x00		
4	wIndexL	**EP**		EP(n):
5	wIndexH	0x00		0x00-0x07: OUT0-OUT7
6	wLengthL	0x00		0x80-0x87: IN0-IN7
7	wLengthH	0x00		

If the USB device supports remote wakeup (reported in its descriptor table when the device enumerates), the Clear_Feature/Remote Wakeup Request disables the wakeup capability.

The Clear_Feature/Stall removes the stall condition from an endpoint. The 8051 should respond by clearing the stall bit in the indicated endpoint’s CS Register.

9.3.4 Get Descriptor

During enumeration, the host queries a USB device to learn its capabilities and requirements using Get_Descriptor Requests. Using tables of *descriptors*, the device sends back (over EP0-IN) such information as what device driver to load, how many endpoints it has, its different configurations, alternate settings it may use, and informative text strings about the device.

The USB core provides a special *Setup Data Pointer* to simplify 8051 service for Get_Descriptor Requests. The 8051 loads this 16-bit pointer with the beginning address of the requested descriptor, clears the HSNAK Bit (by writing "1" to it), and the USB core does the rest.

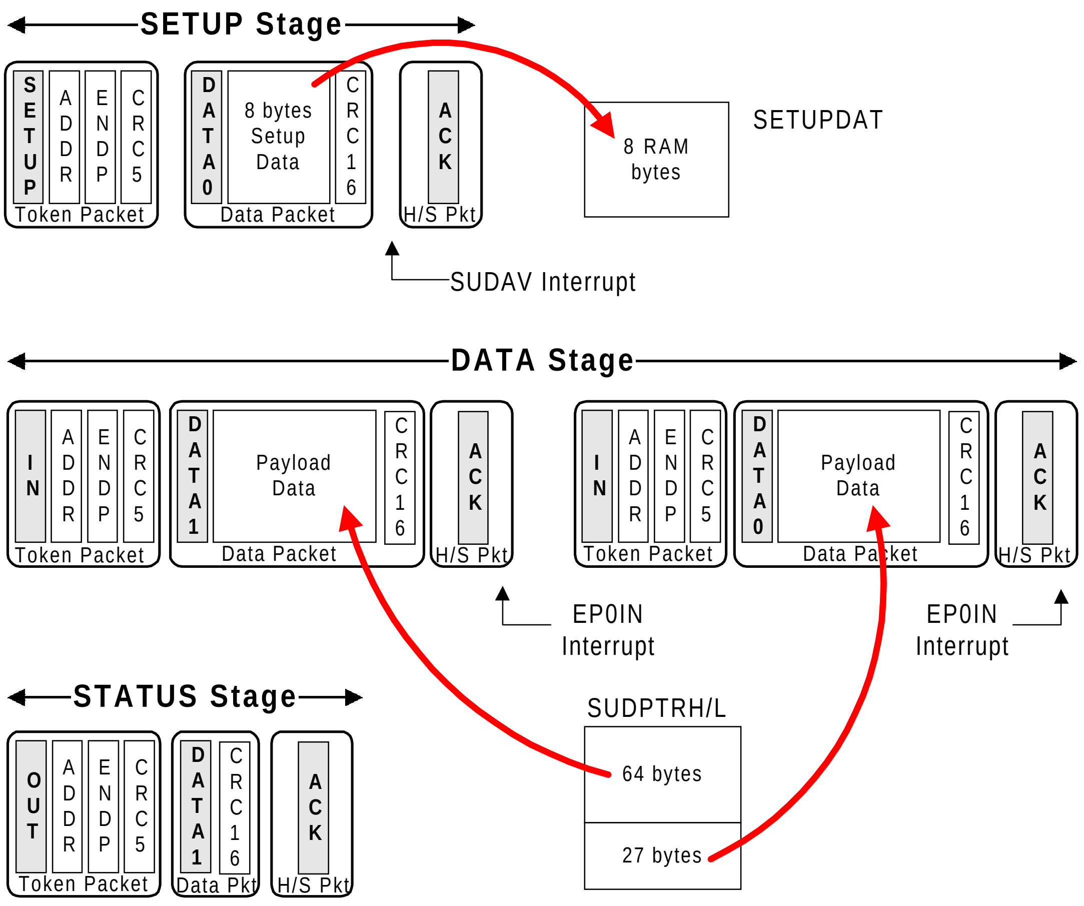


Figure 9-5. Using Setup Data Pointer (SUDPTR) for Get_Descriptor Requests

Figure 9-5 illustrates use of the Setup Data Pointer. This pointer is implemented as two registers, SUDPTRH and SUDPTRL. Most Get_Descriptor Requests involve transferring more data than fits into one packet. In the *Figure 9-5* example, the descriptor data consists of 91 bytes.

The CONTROL transaction starts in the usual way, with the USB core transferring the eight bytes in the SETUP packet into RAM at SETUPDAT and activating the SUDAV interrupt request. The 8051 decodes the Get_Descriptor Request, and responds by clearing the HSNAK Bit (by writing "1" to it), and then loading the SUDPTR Registers with the address of the requested descriptor. Loading the SUDPTRL Register causes the USB core to automatically respond to two IN transfers with 64 bytes and 27 bytes of data using SUDPTR as a base address, and then to respond to (ACK) the STATUS stage.

The usual endpoint zero interrupts, SUDAV and EP0IN, remain active during this automated transfer. The 8051 normally disables these interrupts because the transfer requires no 8051 intervention.

Three types of descriptors are defined: Device, Configuration, and String.

9.3.4.1 Get Descriptor-Device

Table 9-10. Get Descriptor-Device

Byte	Field	Value	Meaning	8051 Response
0	bmRequestType	**0x80**	IN, Device	*Set SUDPTR H-L to start of Device Descriptor table in RAM.*
1	bRequest	**0x06**	“Get_Descriptor”	
2	wValueL	0x00		
3	wValueH	**0x01**	Descriptor Type: Device	
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	**LenL**		
7	wLengthH	**LenH**		

As illustrated in *Figure 9-5*, the 8051 loads the 2-byte SUDPTR with the starting address of the Device Descriptor table. When SUDPTRL is loaded, the USB core performs the following operations:

1. Reads the requested number of bytes for the transfer from bytes 6 and 7 of the SETUP packet (**LenL** and **LenH** in Table 9-10).
2. Reads the requested descriptor’s length field to determine the actual descriptor length.
3. Sends the smaller of (a) the requested number of bytes or (b) the actual number of bytes in the descriptor, over IN0BUF using the Setup Data Pointer as a data table index. This constitutes the second phase of the three-phase CONTROL transfer. The core packetizes the data into multiple data transfers, as necessary.
4. Automatically checks for errors and re-transmits data packets if necessary.
5. Responds to the third (handshake) phase of the CONTROL transfer to terminate the operation.

The Setup Data Pointer can be used for any Get_Descriptor Request; for example, Get_Descriptor-String. It can also be used for vendor-specific requests (you define), as long as bytes 6-7 contain the number of bytes in the transfer (Step 1).

It is possible for the 8051 to do *manual* CONTROL transfers, directly loading the IN0BUF Buffer with the various packets and keeping track of which SETUP phase is in effect. This is a good USB training exercise, but not necessary due to the hardware support built into the USB core for CONTROL transfers.

For DATA stage transfers of fewer than 64 bytes, moving the data into the IN0BUF Buffer and then loading the EP0INBC Register with the byte count would be equivalent to loading the Setup Data Pointer. However, this would waste 8051 overhead because the Setup Data Pointer requires no byte transfers into the IN0BUF Buffer.

9.3.4.2 Get Descriptor-Configuration

Table 9-11. Get Descriptor-Configuration

Byte	Field	Value	Meaning	8051 Response
0	bmRequestType	**0x80**	IN, Device	*Set SUDPTR H-L to start of requested Configuration Descriptor table in RAM*
1	bRequest	**0x06**	“Get_Descriptor”	
2	wValueL	**CFG**	Config Number	
3	wValueH	**0x02**	Descriptor Type: Configuration	
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	**LenL**		
7	wLengthH	**LenH**		

9.3.4.3 Get Descriptor-String

Table 9-12. Get Descriptor-String

Byte	Field	Value	Meaning	8051 Response
0	bmRequestType	**0x80**	IN, Device	*Set SUDPTR H-L to start of requested string Descriptor table in RAM.*
1	bRequest	**0x06**	“Get_Descriptor”	
2	wValueL	**STR**	String Number	
3	wValueH	**0x03**	Descriptor Type: String	
4	wIndexL	0x00	(Language ID L)	
5	wIndexH	0x00	(Language ID H)	
6	wLengthL	**LenL**		
7	wLengthH	**LenH**		

Configuration and string descriptors are handled similarly to device descriptors. The 8051 firmware reads byte 2 of the SETUP data to determine which configuration or string is being requested, then loads the corresponding table pointer into SUDPTRH-L. The USB core does the rest.

9.3.5 Set Descriptor

Table 9-13. Set Descriptor-Device

Byte	Field	Value	Meaning	8051 Response
0	bmRequestType	**0x00**	OUT, Device	*Read device descriptor data over OUT0BUF.*
1	bRequest	**0x07**	“Set_Descriptor”	
2	wValueL	0x00		
3	wValueH	0x01	Descriptor Type: Device	
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	**LenL**		
7	wLengthH	**LenH**		

Table 9-14. Set Descriptor-Configuration

Byte	Field	Value	Meaning	8051 Response
0	bmRequestType	**0x00**	OUT, Device	*Read configuration descriptor data over OUT0BUF.*
1	bRequest	**0x07**	“Set_Descriptor”	
2	wValueL	0x00		
3	wValueH	0x02	Descriptor Type: Configuration	
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	**LenL**		
7	wLengthH	**LenH**		

Table 9-15. Set Descriptor-String

Byte	Field	Value	Meaning	8051 Response
0	bmRequestType	**0x00**	IN, Device	*Read string descriptor data over OUT0BUF.*
1	bRequest	**0x07**	“Get_Descriptor”	
2	wValueL	0x00	Config Number	
3	wValueH	0x03	Descriptor Type: String	
4	wIndexL	0x00	(Language ID L)	
5	wIndexH	0x00	(Language ID H)	
6	wLengthL	**LenL**		
7	wLengthH	**LenH**		

The 8051 handles Set_Descriptor Requests by clearing the HSNAK Bit (by writing “1” to it), then reading descriptor data directly from the OUT0BUF Buffer. The USB core keeps track of the number of byes transferred from the host into OUT0BUF, and compares this number with the length field in bytes 6 and 7. When the proper number of bytes has been transferred, the USB core automatically responds to the status phase, which is the third and final stage of the CONTROL transfer.

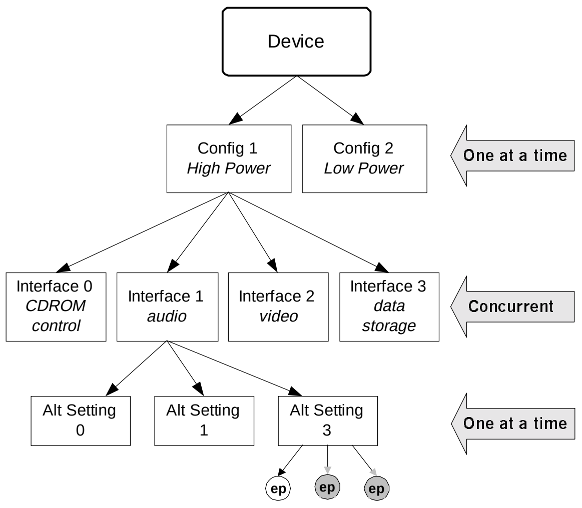
The 8051 controls the flow of data in the Data Stage of a Control Transfer. After the 8051 processes each OUT packet, it loads any value into the OUT endpoint’s byte count register to re-arm the endpoint.

Configurations, Interfaces, and Alternate Settings

A USB device has one or more **configurations**. Only one configuration is active at any time.

A configuration has one or more **interfaces**, all of which are concurrently active. Multiple interfaces allow different host-side device drivers to be associated with different portions of a USB device.

Each interface has one or more **alternate settings**. Each alternate setting has a collection of one or more endpoints.



This structure is a software model; the USB core takes no action when these settings change. However, the 8051 **must re-initialize endpoints** when the host changes configurations or interfaces alternate settings.

As far as 8051 firmware is concerned, a *configuration* is simply a byte variable that indicates the current setting.

The host issues a Set_Coniguration Request to select a configuration, and a Get_Configuration Request to determine the current configuration.

9.3.5.1 Set Configuration

Table 9-16. Set Configuration

Byte	Field	Value	Meaning	8051 Response
0	bmRequestType	**0x00**	OUT, Device	*Read and stash byte 2, change configurations in firmware.*
1	bRequest	**0x09**	“Set_Configuration”	
2	wValueL	**CFG**	Config Number	
3	wValueH	0x00		
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	0x00		
7	wLengthH	0x00		

When the host issues the Set_Configuration Request, the 8051 saves the configuration number (byte 2 inTable 9-16), performs any internal operations necessary to support the configuration, and finally clears the HSNAK Bit (by writing “1” to it) to terminate the Set_Configuration CONTROL transfer.

After setting a configuration, the host issues Set_Interface commands to set up the various interfaces contained in the configuration.

9.3.6 Get Configuration

Table 9-17. Get Configuration

Byte	Field	Value	Meaning	8051 Response
0	bmRequestType	**0x80**	IN, Device	*Send CFG over IN0BUF after re-configuring.*
1	bRequest	**0x08**	“Get_Configuration”	
2	wValueL	0x00		
3	wValueH	0x00		
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	**1**	LenL	
7	wLengthH	**0**	LenH	

The 8051 returns the current configuration number. It loads the configuration number into EP0IN, loads a byte count of one into EP0INBC, and finally clears the HSHAK Bit (by writing “1” to it) to terminate the Set_Configuration CONTROL transfer.

9.3.7 Set Interface

This confusingly named USB command actually sets *alternate settings* for a specified interface.

USB devices can have multiple concurrent interfaces. For example, a device may have an audio system that supports different sample rates, and a graphic control panel that supports different languages. Each interface has a collection of endpoints. Except for endpoint 0, which each interface uses for device control, endpoints may not be shared between interfaces.

Interfaces may report alternate settings in their descriptors. For example, the audio interface may have setting 0, 1, and 2 for 8-KHz, 22-KHz, and 44-KHz sample rates. The panel interface may have settings 0 and 1 for English and Spanish. The Set/Get_Interface Requests select between the various alternate settings in an interface.

Table 9-18. Set Interface (Actually, Set Alternate Setting AS for Interface IF)

Byte	Field	Value	Meaning	8051 Response
0	bmRequestType	**0x00**	OUT, Device	*Read and stash byte 2 (AS) for Interface IF, change setting for Interface IF in firmware.*
1	bRequest	**0x0B**	“Set_Interface”	
2	wValueL	**AS**	Alt Setting Number	
3	wValueH	0x00		
4	wIndexL	**IF**	For this interface	
5	wIndexH	0x00		
6	wLengthL	0x00		
7	wLengthH	0x00		

The 8051 should respond to a Set_Interface Request by performing the following steps:

1. Perform the internal operation requested (such as adjusting a sampling rate).
2. Reset the data toggles for every endpoint in the interface.
3. For an IN endpoint, clear the busy bit for every endpoint in the interface.
4. For an OUT endpoint, load any value into the byte count register for every endpoint in the interface.
5. Clear the HSNAK Bit (by writing “1” to it) to terminate the Set_Feature/Stall CONTROL transfer.

9.3.8 Get Interface

Table 9-19. Get Interface (Actually, Get Alternate Setting AS for interface IF)

Byte	Field	Value	Meaning	8051 Response
0	bmRequestType	**0x81**	IN, Device	*Send AS for Interface IF over OUT0BUF (1 byte).*
1	bRequest	**0x0A**	“Get_Interface”	
2	wValueL	0x00		
3	wValueH	0x00		
4	wIndexL	**IF**	For this interface	
5	wIndexH	0x00		
6	wLengthL	**1**	LenL	
7	wLengthH	**0**	LenH	

The 8051 simply returns the alternate setting for the requested interface IF, and clears the HSNAK Bit by writing “1” to it.

9.3.9 Set Address

When a USB device is first plugged in, it responds to device address 0 until the host assigns it a unique address using the Set_Address Request. The USB core copies this device address into the FNADDR (Function Address) Register, and subsequently responds only to requests to this address. This address is in effect until the USB device is unplugged, the host issues a USB Reset, or the host powers down.

The FNADDR Register can be read, but not written by the 8051. Whenever the USB core ReNumerates™, it automatically resets the FNADDR to zero, allowing the device to come back as *new*.

An 8051 program does not need to know the device address, because the USB core automatically responds only to the host-assigned FNADDR value. The USB core makes it readable by the 8051 for debug/diagnostic purposes.

9.3.10 Sync Frame

Table 9-20. Sync Frame

Byte	Field	Value	Meaning	8051 Response
0	bmRequestType	**0x82**	IN, Endpoint	*Send a frame number over*
1	bRequest	**0x0C**	“Sync_Frame”	*IN0BUF to synchronize endpoint.*
2	wValueL	0x00		*EP*
3	wValueH	0x00		
4	wIndexL	**EP**	Endpoint number	
5	wIndexH	0x00		EP(n):
6	wLengthL	**2**	LenL	0x08-0x0F: OUT8-OUT15
7	wLengthH	**0**	LenH	0x88-0x8F: IN8-IN15

The Sync_Frame Request is used to establish a marker in time so the host and USB device can synchronize multi-frame transfers over isochronous endpoints.

Suppose an isochronous transmission consists of a repeating sequence of five 300 byte packets transmitted from host to device over EP8-OUT. Both host and device maintain sequence counters that count repeatedly from 1 to 5 to keep track of the packets inside a transmission. To start up in sync, both host and device need to reset their counts to “0” at the same time (in the same frame).

To get in sync, the host issues the Sync_Frame Request with EP=EP-OUT (byte 4). The 8051 firmware responds by loading IN0BUF with a two-byte frame count for some future time; for example, the current frame plus 20. This marks frame “current+20” as the sync frame, during which both sides initialize their sequence counters to “0.” The 8051 reads the current frame count in the USBFRAMEL and USBFRAMEH Registers.

Multiple isochronous endpoints can be synchronized in this manner. The 8051 would keep separate internal sequence counts for each endpoint.

About USB Frames

The USB host issues a SOF (Start Of Frame) packet once every millisecond. Every SOF packet contains an 11-bit (mod-2048) frame number. The 8051 services all isochronous transfers at SOF time, using a single SOF interrupt request and vector. If the USB core detects a missing SOF packet, it uses an internal counter to generate the SOF interrupt.

9.3.11 Firmware Load

The USB endpoint zero protocol provides a mechanism for mixing vendor-specific requests with standard device requests. Bits 6:5 of the bmRequestType field are set to 00 for a standard device request, and to 10 for a vendor request.

Table 9-21. Firmware Download

Byte	Field	Value	Meaning	8051 Response
0	bmRequestType	**0x40**	Vendor Request, OUT	*None required.*
1	bRequest	**0xA0**	“Firmware Load”	
2	wValueL	**AddrL**	Starting address	
3	wValueH	**AddrH**		
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	**LenL**	Number of bytes	
7	wLengthH	**LenH**		

Table 9-22. Firmware Upload

Byte	Field	Value	Meaning	8051 Response
0	bmRequestType	**0xC0**	Vendor Request, IN	*None Required.*
1	bRequest	**0xA0**	“Firmware Load”	
2	wValueL	**AddrL**	Starting address	
3	wValueH	**AddrH**		
4	wIndexL	0x00		
5	wIndexH	0x00		
6	wLengthL	**LenL**	Number of Bytes	
7	wLengthH	**LenH**		

The USB core responds to two endpoint zero vendor requests, RAM Download and RAM Upload. These requests are active in all modes (RENUM=0 or 1).

Because bit 7 of the first byte of the SETUP packet specifies direction, only one bRequest value (0xA0) is required for the upload and download requests. These RAM load commands are available to any USB device that uses the EZ-USB FX chip.

A host loader program typically writes 0x01 to the CPUCS Register to put the 8051 into RESET, loads all or part of the EZ-USB FX internal RAM with 8051 code, and finally reloads the CPUCS Register with 0 to take the 8051 out of RESET. The CPUCS Register is the only USB register that can be written using the Firmware Download command.

Chapter 10. EZ-USB FX Isochronous Transfers

10.1 Introduction

Isochronous endpoints typically handle time-critical, streamed data delivered or consumed in byte-sequential order. Examples are audio data sent to a DAC over USB or teleconferencing video data sent from a camera to the host. Due to the byte-sequential nature of this data, the EZ-USB FX chip makes isochronous data available as a single byte that represents the head or tail of an endpoint FIFO.

The EZ-USB FX chips that support isochronous transfers implement sixteen isochronous endpoints, IN8-IN15 and OUT8-OUT15. 1,024 bytes of FIFO memory can be distributed over the 16 endpoint addresses. FIFO sizes for the isochronous endpoints are programmable.

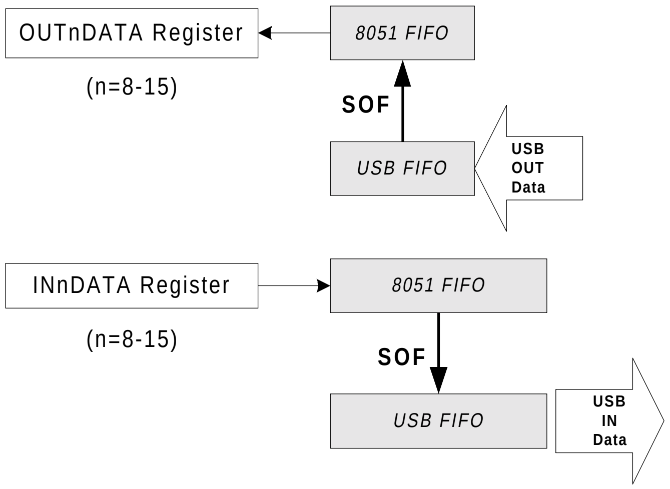


Figure 10-1. EZ-USB FX Isochronous Endpoints 8-15

The 8051 reads or writes isochronous data using sixteen FIFO data registers, one per endpoint. These FIFO registers are shown in *Figure 10-1* as INnDATA (Endpoint n IN Data) and OUTnDATA (Endpoint n OUT Data).

The USB core provides a total of 2,048 bytes of FIFO memory (1,024 bytes, double-buffered) for ISO endpoints. This memory is in addition to the 8051 program/data memory, and normally exists outside of the 8051 memory space. The 1,024 FIFO bytes may be divided among the sixteen isochronous endpoints. The 8051 writes sixteen EZ-USB FX registers to allocate the FIFO buffer space to the isochronous endpoints. The 8051 also sets *endpoint valid* bits to enable isochronous endpoints.

10.2 Isochronous IN Transfers

IN transfers travel from device to host. *Figure 10-2* shows the EZ-USB FX registers and bits associated with isochronous IN transfers.

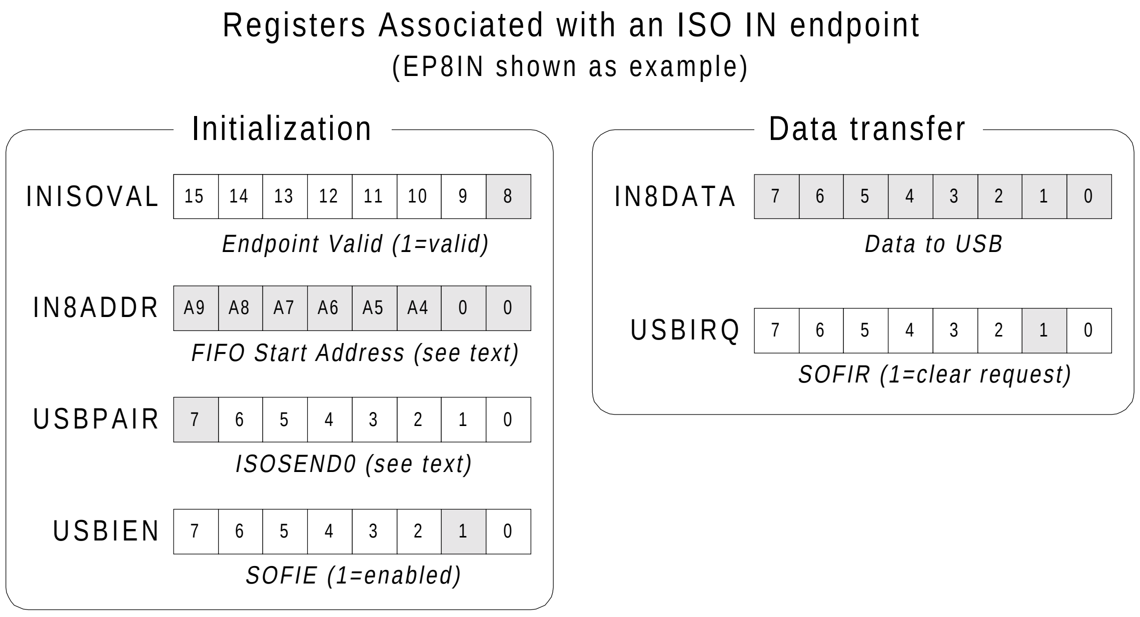


Figure 10-2. Isochronous IN Endpoint Registers

10.2.1 Initialization

To initialize an isochronous IN endpoint, the 8051 performs the following:

1. Sets the endpoint valid bit for the endpoint.
2. Sets the endpoint's FIFO size by loading a starting address (Section 10.4. *"Setting Isochronous FIFO Sizes"*).
3. Sets the ISOSEND0 Bit in the USBPAIR Register for the desired response.

4. Enables the SOF Interrupt. All isochronous endpoints are serviced in response to the SOF Interrupt.
5. Sets the INT2SFR Bit in USBBAV to enable INT 2 clearing via the INT2CLR SFR Register.

The USB core uses the ISOSEND0 Bit to determine what to do if:

- The 8051 does not load any bytes to an INnDATA Register during the previous frame, and
- An IN token for that endpoint arrives from the host.

If ISOSEND0=0 (the default value), the USB core does not respond to the IN token. If ISOSEND0=1, the USB core sends a zero-length data packet in response to the IN token. The action to take depends on the overall system design. The ISOSEND0 Bit applies to all of the isochronous IN endpoints, EP8IN through EP15IN.

10.2.2 IN Data Transfers

When an SOF Interrupt occurs, the 8051 is presented with empty IN FIFOs that it fills with data to be transferred to the host during the next frame. The 8051 has 1 ms to transfer data into these FIFOs before the next SOF Interrupt arrives.

To respond to the SOF Interrupt, the 8051 clears the USB Interrupt (8051 INT2) by clearing EXIF.4, and clears the SOFIR (Start Of Frame Interrupt Request) Bit by writing any value to the INT2CLR Register. Then, the 8051 loads data into the appropriate isochronous INnDATA FIFO Register(s). The USB core keeps track of the number of bytes the 8051 loads to each INnDATA Register, and subsequently transfers the correct number of bytes in response to the USB IN token during the next frame.

The isochronous FIFO swap occurs every SOF, even if during the previous frame the host did not issue an IN token to read the isochronous FIFO data, or if the host encountered an error in the data. USB isochronous data has no *re-try* mechanism, like bulk data.

10.3 Isochronous OUT Transfers

OUT transfers travel from host to device. *Figure 10-3* shows the EZ-USB FX registers and bits associated with isochronous OUT transfers.

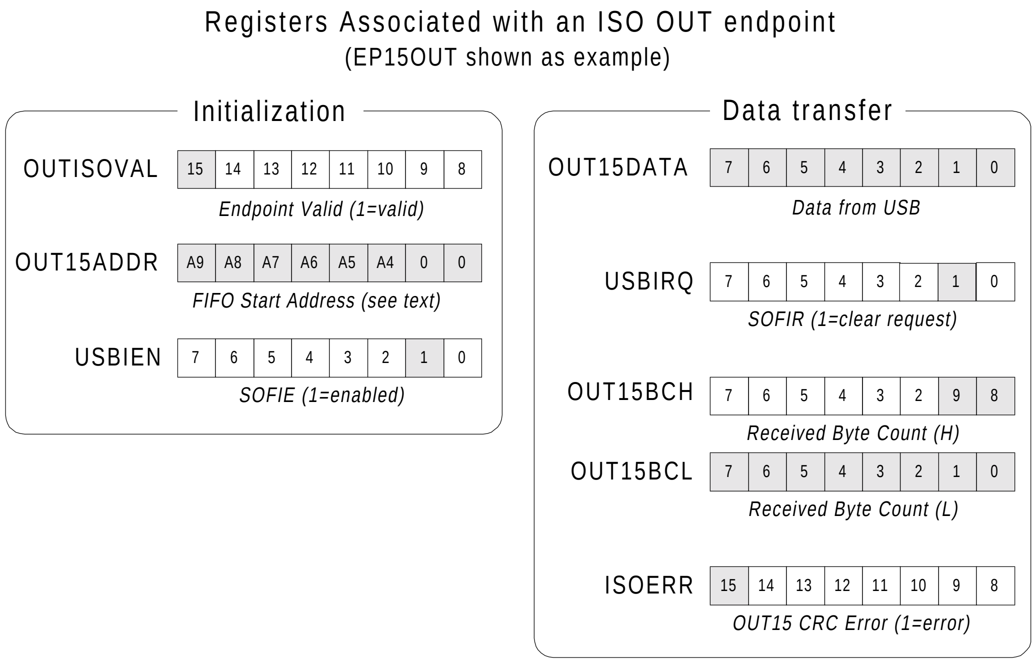


Figure 10-3. Isochronous OUT Registers

10.3.1 Initialization

To initialize an isochronous OUT endpoint, the 8051:

- Sets the endpoint valid bit for the endpoint.
- Sets the endpoint's FIFO size by loading a starting address (Section 10.4. *"Setting Isochronous FIFO Sizes"*).
- Enables the SOF Interrupt. All isochronous endpoints are serviced in response to the SOF Interrupt.
- Sets the INT2SFR Bit in USBBAV to enable INT 2 clearing via the INT2CLR SFR.

10.3.2 OUT Data Transfer

When an SOF Interrupt occurs, the 8051 is presented with FIFOs containing OUT data sent from the host in the previous frame, along with 10-bit byte counts, indicating how many bytes are in the FIFOs. The 8051 has 1 ms to transfer data out of these FIFOs before the next SOF Interrupt arrives.

To respond to the SOF Interrupt, the 8051 clears the USB Interrupt (8051 INT2), and clears the SOFIR Bit by writing “1” to it. Then, the 8051 reads data from the appropriate OUTnDATA FIFO Register(s). The 8051 can check an error bit in the ISOERR Register to determine if a CRC error occurred for the endpoint data. Isochronous data is never re-sent, so the firmware must decide what to do with *bad-CRC* data.

10.4 Setting Isochronous FIFO Sizes

Up to sixteen EZ-USB FX isochronous endpoints share an EZ-USB FX 1,024-byte RAM, which can be configured as one to sixteen FIFOs. The 8051 initializes the endpoint FIFO sizes by specifying the starting address for each FIFO within the 1,024 bytes, starting at address zero. The isochronous FIFOs can exist anywhere in the 1,024 bytes, but the user must take care to ensure that there is sufficient space between start addresses to accommodate the endpoint FIFO size.

Sixteen start address registers set the isochronous FIFO sizes(Table 10-1). The USB core constructs the address writing the 1,024 byte range from the register value as shown in *Figure 10-4*.

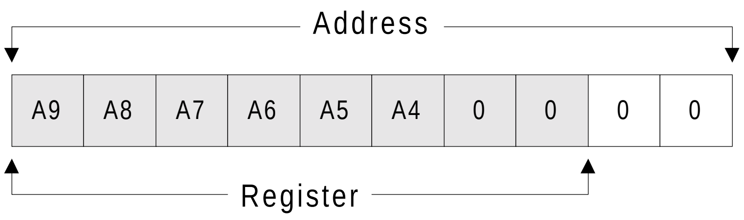


Figure 10-4. FIFO Start Address Format

Table 10-1. Isochronous Endpoint FIFO Starting Address Registers

Register	Function	b7	b6	b5	b4	b3	b2	b1	b0
OUT8ADDR	Endpoint 8 OUT Start Address	A9	A8	A7	A6	A5	A4	0	0
OUT9ADDR	Endpoint 9 OUT Start Address	A9	A8	A7	A6	A5	A4	0	0
OUT10ADDR	Endpoint 10 OUT Start Address	A9	A8	A7	A6	A5	A4	0	0
OUT11ADDR	Endpoint 11 OUT Start Address	A9	A8	A7	A6	A5	A4	0	0
OUT12ADDR	Endpoint 12 OUT Start Address	A9	A8	A7	A6	A5	A4	0	0
OUT13ADDR	Endpoint 13 OUT Start Address	A9	A8	A7	A6	A5	A4	0	0
OUT14ADDR	Endpoint 14 OUT Start Address	A9	A8	A7	A6	A5	A4	0	0
OUT15ADDR	Endpoint 15 OUT Start Address	A9	A8	A7	A6	A5	A4	0	0
IN8ADDR	Endpoint 8 IN Start Address	A9	A8	A7	A6	A5	A4	0	0
IN9ADDR	Endpoint 9 IN Start Address	A9	A8	A7	A6	A5	A4	0	0
IN10ADDR	Endpoint 10 IN Start Address	A9	A8	A7	A6	A5	A4	0	0
IN11ADDR	Endpoint 11 IN Start Address	A9	A8	A7	A6	A5	A4	0	0
IN12ADDR	Endpoint 12 IN Start Address	A9	A8	A7	A6	A5	A4	0	0
IN13ADDR	Endpoint 13 IN Start Address	A9	A8	A7	A6	A5	A4	0	0
IN14ADDR	Endpoint 14 IN Start Address	A9	A8	A7	A6	A5	A4	0	0
IN15ADDR	Endpoint 15 IN Start Address	A9	A8	A7	A6	A5	A4	0	0

The size of an isochronous endpoint FIFO is determined by subtracting consecutive addresses in Table 10-1, and multiplying by four. Values written to these registers must have the two LSBs set to zero. The last endpoint, EP15IN, has a size of 1,024 minus IN15ADDR times four [(1024-IN15ADDR)4]. Because the 10-bit effective address has the four LSBs set to zero (*Figure 10-4*), the FIFO sizes are allocated in increments of 16 bytes. For example, if OUT8ADDR=0x00 and OUT9ADDR=0x04, EP8OUT has a FIFO size of the difference multiplied by four or 16 bytes.

An 8051 assembler or C compiler may be used to translate FIFO sizes into starting addresses. The assembler example in *Figure 10-5* shows a block of equates for the 16 isochronous FIFO sizes, followed by assembler equations to compute the corresponding FIFO relative address values. To initialize all sixteen FIFO sizes, the 8051 merely copies the table starting at 8OUTAD to the sixteen EZ-USB FX registers starting at OUT8ADDR.

```
0100   EP8INSZ     equ   256     ; Iso FIFO sizes in bytes
0100   EP8OUTSZ    equ   256
0010   EP9INSZ     equ   16
0010   EP9OUTSZ    equ   16
0010   EP10INSZ    equ   16
0010   EP10OUTSZ   equ   16
0000   EP11INSZ    equ   0
0000   EP11OUTSZ   equ   0
0000   EP12INSZ    equ   0
0000   EP12OUTSZ   equ   0
0000   EP13INSZ    equ   0
0000   EP13OUTSZ   equ   0
0000   EP14INSZ    equ   0
0000   EP14OUTSZ   equ   0
0000   EP15INSZ    equ   0
0000   EP15OUTSZ   equ   0
;
0000   8OUTAD      equ   0       ; Load these 16 bytes into ADDR regs starting OUT8ADDR
0040   9OUTAD      equ   8OUTAD  + Low(EP8OUTSZ/4)
0044   10OUTAD     equ   9OUTAD  + Low(EP9OUTSZ/4)
0048   11OUTAD     equ   10OUTAD + Low(EP10OUTSZ/4)
0048   12OUTAD     equ   11OUTAD + Low(EP11OUTSZ/4)
0048   13OUTAD     equ   12OUTAD + Low(EP12OUTSZ/4)
0048   14OUTAD     equ   13OUTAD + Low(EP13OUTSZ/4)
0048   15OUTAD     equ   14OUTAD + Low(EP14OUTSZ/4)
0048   8INAD       equ   15OUTAD + Low(EP15OUTSZ/4)
0088   9INAD       equ   8INAD   + Low(EP8INSZ/4)
008C   10INAD      equ   9INAD   + Low(EP9INSZ/4)
0090   11INAD      equ   10INAD  + Low(EP10INSZ/4)
0090   12INAD      equ   11INAD  + Low(EP11INSZ/4)
0090   13INAD      equ   12INAD  + Low(EP12INSZ/4)
0090   14INAD      equ   13INAD  + Low(EP13INSZ/4)
0090   15INAD      equ   14INAD  + Low(EP14INSZ/4)
```

Figure 10-5. Using Assembler to Translate the FIFO Sizes to Addresses

The assembler computes starting addresses (*Figure 10-5*) by adding the previous endpoint's address to the desired size shifted right twice. This aligns A9 with bit 7 as shown in Table 10-1. The LOW operator takes the low byte of the resulting 16-bit expression

The user of this code must ensure that the sizes given in the first equate block are all multiples of 16. This is easy to tell by inspection—the least significant digit of the hex values in the first column should be zero.

10.5 Isochronous Transfer Speed

The amount of data USB can transfer during a 1-ms frame is slightly more than 1,000 bytes per frame (1,500 bytes theoretical, without accounting for USB overhead and bus utilization). A device's actual isochronous transfer bandwidth is usually determined by how fast the CPU can move data in and out of its isochronous endpoint FIFOs.

The 8051 code example in *Figure 10-6* shows a typical transfer loop for moving external FIFO data into an IN endpoint FIFO. This code assumes that the 8051 is moving data from an external FIFO attached to the EZ-USB FX data bus and strobed by the RD signal, into an internal isochronous IN FIFO.

```
mov   dptr, #DMASRCH    ; (3) Set up the DMA source to be 0x8000 in external memory
mov   a, #080h          ; (2)
movx  @dptr, a          ; (2)
mov   dptr, #DMASRCL    ; (2)
clr   a                 ; (1)
movx  @dptr, a          ; (2)
mov   dptr, #DMADESTH   ; (3) Set up the DMA destination to be the IN8DATA fifo
mov   a, #low(IN8DATA)  ; (2)
movx  @dptr, a          ; (2)
mov   dptr, #DMADESTL   ; (2)
mov   a, #hi(IN8DATA)   ; (2)
movx  @dptr, a          ; (2)
mov   dptr, #DMALEN     ; (3) Load the length register. Note that this code
                        ;  will only support up to 0xff bytes.
mov   a, #nbytes        ; (2)
movx  @dptr, a          ; (2)
mov   dptr, #DMAGO      ; (3) Start the DMA. It will run in parallel with the
                        ; 8051 code for nbytes/4 8051 cyclesexternal
                        ; Note that this code must be running in internal memory and
                        ; that memory cannot be used during the DMA (Maximum 64 8051
                        ; clocks long)
movx  @dptr, a          ; (2)
```

Figure 10-6. 8051 Data Transfer to Isochronous FIFO (IN8DATA) w/DMA

This code sets up a transfer in thirty-five 8051 cycles and then completes the transfer via DMA in 64 clocks for 256 bytes. Using 99 cycles for 256 bytes provides a net ISO transfer rate of 31Mbytes/second (with a 48Mhz 8051). In other words, performing the ISO transfer only uses 3.3% of the processor MIPS, leaving the entire 8051 available for other processing.

10.6 Other Isochronous Registers

Two additional registers, ISOCTL and ZBCOUT, provide additional isochronous endpoint features.

10.6.1 Disable ISO

ISOCTL **Isochronous Control** **7FA1**

b7	b6	b5	b4	b3	b2	b1	b0
-	**-**	**-**	**-**	**PPSTAT**	**MBZ**	**MBZ**	**ISODISAB**

Figure 10-7. ISOCTL Register

Bit zero of the ISOCTL Register is called ISODISAB. When the 8051 sets ISODISAB=1, all sixteen of EZ-USB FX endpoints are disabled. If ISODISAB=1, EP8IN-EP15IN and EP8OUT-EP15OUT should not be used. ISODISAB is cleared at power-on.

When ISODISAB=1, the 2,048 bytes of RAM normally used for isochronous buffers is available to the 8051 as XDATA RAM (*not* program memory), from 0x2000 to 0x27FF in internal memory. When ISODISAB=1, the behavior of the RD# and WR# strobe signals changes to reflect the additional 2 KB of memory inside the EZ-USB FX chip. This is shown in Table 10-2.

Table 10-2. Addresses for RD# and WR# vs. ISODISAB Bit

ISODISAB	RD#, WR#
0 (default)	2000-7B40, 8000-FFFF
1	**2800**-7B40, 8000-FFFF

ISOCTL Register bits listed as MBZ (must be zero) in *Figure 10-7* must be written with zeros. The PPSTAT Bit toggles every SOF, and may be written with any value (no effect). Therefore, to disable the isochronous endpoints, the 8051 should write the value 0x01 to the ISOCTL Register.

***Caution!** — If you use this option, be absolutely certain that the host never sends isochronous data to your device. Isochronous data directed to a disabled isochronous endpoint system causes unpredictable operation.*

The Autopointer is not usable from 0x2000-0x27FF (the reclaimed ISO buffer RAM) when ISO-DISAB=1.

10.6.2 Zero Byte Count Bits

ZBCOUT **Zero Byte Count OUT** **7FA2**

b7	b6	b5	b4	b3	b2	b1	b0
EP15	**EP14**	**EP13**	**EP12**	**EP11**	**EP10**	**EP9**	**EP8**

Figure 10-8. ZBCOUT Register

When the SOF Interrupt is asserted, the 8051 normally checks the isochronous OUT endpoint FIFOs for data. Before reading the byte count registers and unloading an isochronous FIFO, the firmware may wish to check for a zero byte count. In this case, the 8051 can check bits in the ZBCOUT Register. Any endpoint bit set to “1” indicates that no OUT bytes were received for that endpoint during the previous frame. *Figure 10-8* shows this register.

The USB core updates these bits every SOF.

10.7 ISO IN Response with No Data

USBPAIR **USB Endpoint Pairing** **7FDD**

b7	b6	b5	b4	b3	b2	b1	b0
ISOEND0	**-**	**PR6OUT**	**PR4OUT**	**PR2OUT**	**PR6IN**	**PR4IN**	**PR2IN**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	x	0	0	0	0	0	0

Figure 10-9. ISOIN Register

The ISOSEND0 Bit (bit 7 in the USBPAIR Register) is used when the EZ-USB FX chip receives an isochronous IN token while the IN FIFO is empty. If ISOSEND0=0 (the default value), the USB core does not respond to the IN token. If ISOSEND0=1, the USB core sends a zero-length data packet in response to the IN token. The action to take depends on the overall system design. The ISOSEND0 Bit applies to all of the isochronous IN endpoints, IN-8 through IN-15.

10.8 Restrictions Near SOF

The EZ-USB FX does not restrict FIFO accesses near SOF events.

Chapter 11. EZ-USB FX DMA System

11.1 Introduction

The EZ-USB FX incorporates a Direct Memory Access (DMA) system that transfers byte data between on-chip or off-chip resources without 8051 intervention. Data can be transferred very quickly (as fast as one byte per 48-MHz clock) using the following sources and destinations:

- Isochronous endpoint buffers
- Bulk endpoint buffers
- Internal Slave FIFOs (A and B)
- External FIFOs
- Internal RAM
- External RAM.

The 8051 sets up a DMA transfer by initializing registers with a source address, a destination address, and a byte transfer count. Up to 256 bytes can be programmed per transfer. Then the 8051 writes a control register to initiate the DMA transfer. The DMA unit signals end-of-transfer with a vectored DMADONE Interrupt request through 8051 INT4.

Most source-destination pairs are supported. The exceptions are explained in this chapter.

Normally a RAM or ROM is connected to the EZ-USB FX address and data bus, which uses the RD# and WR# pins for strobes. It is also possible to connect an external FIFO to the data bus and use a second set of strobe signals, FRD# (Fast Read) and FWR# (Fast Write).

11.2 DMA Register Descriptions

11.2.1 Source, Destination, Transfer Length Address Registers

DMASRCH **DMA Source Address (H)** **784F**

b7	b6	b5	b4	b3	b2	b1	b0
A15	**A14**	**A13**	**A12**	**A11**	**A10**	**A9**	**A8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 11-1. Upper Byte of the DMA Source Address

DMASRCL **DMA Source Address (L)** **7850**

b7	b6	b5	b4	b3	b2	b1	b0
A7	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 11-2. Lower Byte of the DMA Source Address

DMADESTH **DMA Destination Address (H)** **7851**

b7	b6	b5	b4	b3	b2	b1	b0
A15	**A14**	**A13**	**A12**	**A11**	**A10**	**A9**	**A8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 11-3. Upper Byte of the DMA Destination Address

DMADESTL	DMA Destination Address (L)	7852

b7	b6	b5	b4	b3	b2	b1	b0
A7	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 11-4. Lower Byte of the DMA Destination Address

DMALEN	DMA Transfer Length	7854

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 11-5. DMA Transfer Length (0=256 Bytes, 1=1 Byte, ... 255=255 Bytes)

There are some restrictions on DMA source-destination pairs. Table 11-1 shows all possible sources and destinations, and which transfers are permitted. The main restriction is that transfers may not occur in the same 2-KB RAM block in which the 8051 code is running.

To elaborate, the internal 8-KB RAM is divided into four 2-KB blocks, as follows:

Block0	0000-07FF
Block1	0800-0FFF
Block2	1000-17FF
Block3	1800-1FFF

Because the internal RAM combines 8051 program and data RAM, the 8051 is normally *running* in (fetching code from) one of these RAM blocks. *The RAM block in which the 8051 code is running during a DMA transfer may not be used simultaneously as a DMA source or destination.*

Block 3 has some special considerations. While doing DMA transfers in or out of Block 3 (a common case, because Block 3 contains the endpoint buffers), you may not simultaneously run code in this block, or access the following resources:

- Bulk Endpoint buffer memory
- Bulk Endpoint byte count registers.

When doing DMA transfers in Block 3, you *may* simultaneously access:

- Block 3 RAM
- Any of the other USB control/status registers.

Some care should be exercised to observe the rule that you can't run 8051 code in RAM that is simultaneously involved in a DMA transfer. For example, if the 8051 is running code in Block 1 and doing a DMA transfer to/from Block 0, an 8051 interrupt would skip to Block 0 to fetch the interrupt vector, and violate the rule.

The safest way to do internal RAM DMA transfers is to put your data buffer into Block 2 or Block 3, and the background code (which runs during the DMA transfer) in Block 0 or Block 1.

This restriction applies only when internal RAM is used as a DMA source or destination.

Table 11-1. DMA Sources and Destinations

DMA Source	RAM 0	RAM 1	RAM 2	RAM3/ Bulk Buffers	WF Desc	ISO IN	EXT MEM	Recl ISO	AOUT FIFO	BOUT FIFO	EXT FIFO
RAM0	**N**	*	*	*	*	*	*	*	*	*	*
RAM1	*	**N**	*	*	*	*	*	*	*	*	*
RAM2	*	*	**N**	*	*	*	*	*	*	*	*
RAM3/Bulk Buffers	*	*	*	**N**	*	*	*	*	*	*	*
ISO OUT	*	*	*	*		**N**	*	**N**			
EXT MEM	*	*	*	*	*	*	**N**	*	*	*	**N**
Reclaimed ISO	*	*	*	*		**N**	*	**N**			
AIN FIFO	*	*	*	*			*				
BIN FIFO	*	*	*	*			*				
EXT FIFO	*	*	*	*			**N**				**N**

* A cell with an asterisk is permitted as long as the 8051 is not executing code in either the source or destination memory block.

N A shaded cell with an "N" is not permitted. The results are unpredictable.

DMA transfers involving mixed internal and external access are not permitted. For example, transferring across the 0x2800 boundary with ISODISAB=1 is illegal. So is transferring across the 0xFFFF (external) to 0x00 (internal) boundary and crossing the 7FFF to 8000 boundary.

Table 11-1 shows all DMA source and destinations, and indicates which DMA transfers are permitted. Table 11-2 explains the legends used in Table 11-1.

- A blank cell indicates a legal source and destination under all conditions.
- RAM3 contains the bulk buffers. Executing code RAM3 precludes DMA access to the bulk buffers. The bulk buffer area may not be used as program space.

Table 11-2. Legends Used in Table 11-1

Source/ Destination	Description
RAM0	RAM Block 0 from 0000-07FF.
RAM1	RAM Block 0 from 0800-0FFF.
RAM2	RAM Block 0 from 1000-17FF.
RAM3	RAM Block 0 from 1800-1FFF.
Bulk Buffers	XDATA Registers at 7B40-7FFF. In the 8-KB part, these registers also appear at 1B40-1FFF.
WF Descriptors	**Destination only.** GPIF waveform descriptors at 7900-797F.*
ISO OUT	**Source only.** Endpoint 8OUT-15OUT FIFO Registers, which are XDATA Registers at 7F60-7F67.
ISO IN	**Destination only.** Endpoint 8IN-15IN Registers, which are XDATA Registers at 7F68-7F6F.
EXT MEM	If isochronous endpoints are disabled, 2000-FFFF (ISODISAB = 1). If any isochronous endpoints are used, 2800-FFFF (ISODISAB = 0).
Reclaimed ISO	RAM at 2000-27FF is available as XDATA memory if ISODISAB=1.
AIN FIFO	**Source only.** Slave FIFO A-IN. FIFO register is at XDATA 7800 (AINDATA).
AOUT FIFO	**Destination only.** Slave FIFO A-OUT. FIFO register is at XDATA 780E (AOUTDATA).
BIN FIFO	**Source only.** Slave FIFO B-IN. FIFO register is at XDATA 7805 (BINDATA).
BOUT FIFO	**Destination only.** Slave FIFO B-OUT. FIFO register is at XDATA 7813 (BOUTDATA).
EXT FIFO	XDATA location 7858, used to access external RAM as a FIFO.

* Accessible only if IFCONFIG[1..0]=10.

11.2.2 DMA Start and Status Register

DMAGO			DMA Start and Status				7855
b7	b6	b5	b4	b3	b2	b1	b0
DONE	**0**	**0**	**0**	**0**	**0**	**0**	**0**
R/W	x	x	x	x	x	x	x
0	x	x	x	x	x	x	x

Figure 11-6. DMA Start and Status Register

The 8051 writes any value to this register to initiate a DMA transfer.

Bit 7: **DONE** *DMA Transfer Done*

This read-only bit indicates that the DMA transfer specified by the source and destination addresses and byte count has completed. DONE=0 indicates DMA is in progress; DONE=1 indicates completion. If enabled, a vectored INT4 Interrupt request is generated on a zero-to-one transition of the DONE Bit.

11.2.3 DMA Synchronous Burst Enables Register

FASTXFR			Fast Transfer Control*				7FE2
b7	b6	b5	b4	b3	b2	b1	b0
FISO	FBLK	RPOL	RMOD1	RMOD0	WPOL	WMOD1	WMOD0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

* This register is shown only for reference.

Figure 11-7. Fast Transfer Control Register

DMABURST			Synchronous Burst Enables				7857
b7	b6	b5	b4	b3	b2	b1	b0
x	**x**	**x**	**DSTR2**	**DSTR1**	**DSTR0**	**RB**	**WB**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	1	0	0

Figure 11-8. Synchronous Burst Enables

This register enables synchronous burst reads and writes on the 8051 address/data bus (RB and WB), and selects cycle stretch values for the read and write strobes (DSTR[2..0]). Refer to Section 11.4.1. *"DMA External Writes"* and Section 11.4.2. *"DMA External Reads"* for more information on DMA Stretch.

Bit 4-2: **DSTR[2..0]** *DMA Stretch*

The read and write strobes used in external DMA transfers are controlled by the following bits:

For DMA Reads:

- One of four read strobe waveforms (synonymous with *modes*) is selected by the RMOD[1..0] Bits in the FASTXFR Register.
- Read strobe polarity is set by the RPOL Bit in the FASTXFR Register.
- Read strobe duration is set by the DMA Stretch Bits (DSTR[2..0]) in the DMABURST Register. This overrides the 8051 stretch bits in PCON that set RD/WR pulse widths (DMA only).

For DMA Writes:

- One of four write strobe waveforms is selected by the WMOD[1..0] Bits in the FASTXFR Register.
- Write strobe polarity is set by the WPOL Bit in the FASTXFR Register.
- Write strobe duration is set by the DSTR[2..0] Bits in the DMABURST Register.

The external DMA read and write stretch values are the same, as set by the DSTR[2..0] Bits.

Bit 1: **RB** *DMA Burst Read*

When RB=1, a burst read DMA transfer is performed over the EZ-USB FX data bus. The term “burst” means that the read strobe stays low during multiple-byte transfers, and the CLKOUT signal (“CLK24/48” in *Figure 11-8.* and *Figure 11-9.*) synchronizes the data. DMA burst reads work only for two read waveforms, Mode 0 and Mode 1. The stretch value DSTR[2..0] has no effect in burst read mode.

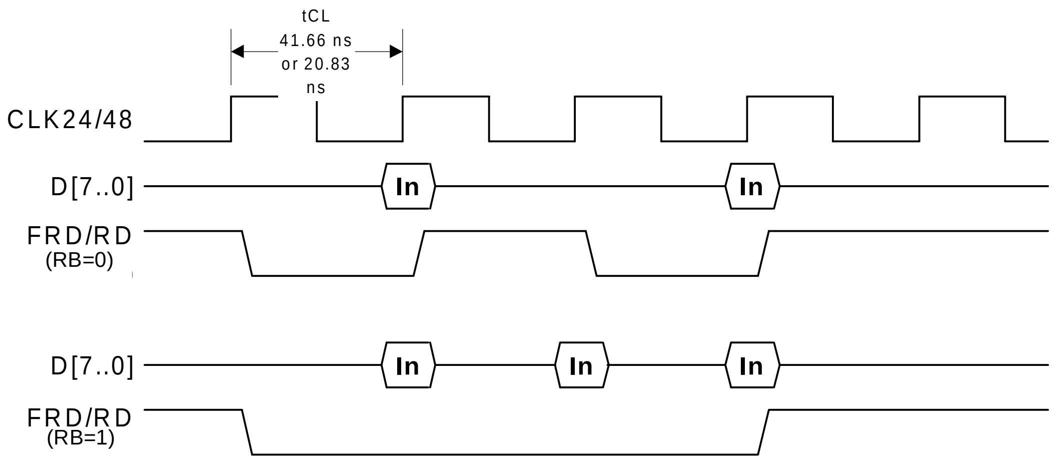


Figure 11-9. Effect of the RB Bit on DMA Mode 0 Reads

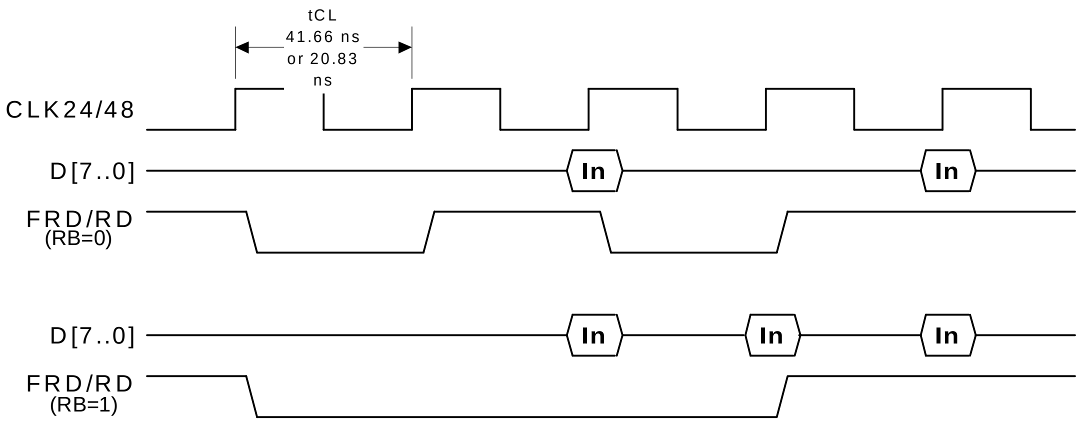


Figure 11-10. Effect of the RB Bit on DMA Mode 1 Reads

In non-burst mode (RB=0), the DMA read transfers occur every other clock, and a read strobe is generated for each byte. In burst mode (RB=1), the read strobe remains low for the entire multi-byte transfer, and a byte is transferred every clock. The only difference between Mode 0 and Mode 1 is that the data is sampled one clock later in Mode 1.

Bit 0: **WB** *DMA Burst Write*

When WB=1, a burst write DMA transfer is performed over the EZ-USB FX data bus. The term “burst” means that the write strobe stays low during multiply-byte transfers, and the CLKOUT signal (shown as “CLK24/48” in *Figure 11-10.*) synchronizes the output data. DMA burst writes work only for the Mode 0 waveform (WMOD[1..0] must be programmed to 00). The DMA stretch value DSTR[2..0] has no effect in burst write.

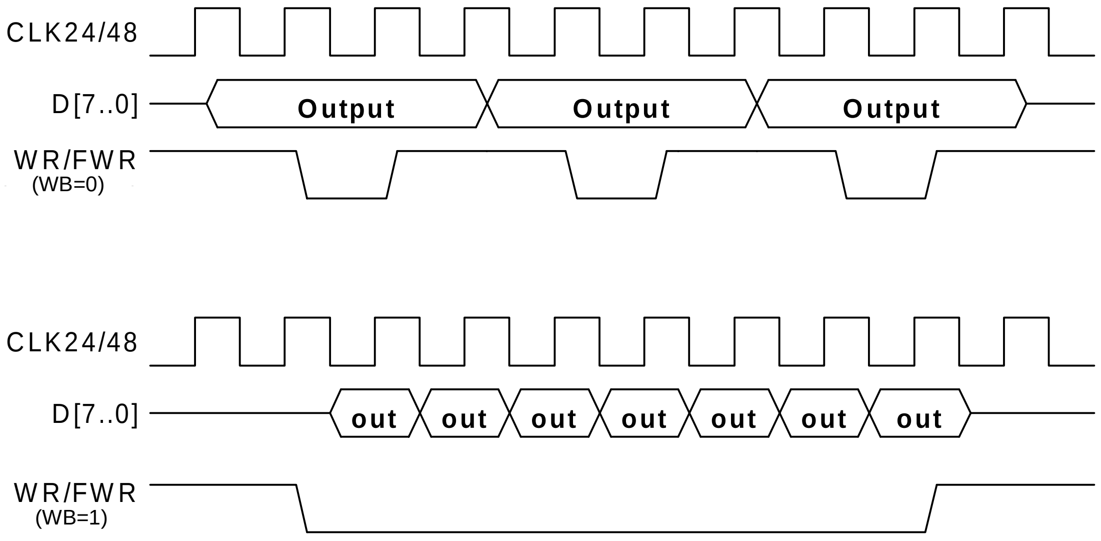


Figure 11-11. Effect of the WB Bit on DMA Mode 0 Writes

In non-burst mode (DMAWR=0), the DMA write transfers occur every third clock, and a write strobe is generated for every byte. In burst mode (DMAWR=1), the write strobe remains low for the entire multi-byte transfer, and a byte is transferred every clock.

11.2.4 Dummy Register

DMAEXTFIFO **Use A/D Buses as External FIFO** **7858**

b7	b6	b5	b4	b3	b2	b1	b0
n/a	**n/a**	**n/a**	**n/a**	**n/a**	**n/a**	**n/a**	**n/a**
n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a

Figure 11-12. DMAEXTFIFO Register. Data is "Don't Care".

This is a *dummy* register, containing no data bits. Programming the DMAEXTFIFO address into the DMA source or destination address causes the EZ-USB FX data bus to be used in the following way for DMA cycles:

- The FRD# or FWR# pins are used as strobes instead of the RD# and WR# pins.
- The address bus follows the 8051 program counter rather than incrementing.
- The typical use for this transfer type is to connect a FIFO to the EZ-USB FX data bus.

11.3 External DMA Transfers - Strobes

11.3.1 Selection of RD/FRD and WR/FWR DMA Strobes

The preceding section explains how to select waveforms and timing for read and write strobes to external memory. These strobes can correspond with two different sets of read-write signals, RD#/WR#, the normal 8051 external bus strobes, or FRD#/FWR#, the EZ-USB FX fast transfer strobes. Which of the read and write strobes is active is controlled by the DMA address loaded into a DMA source or destination register, according to Table 11-3. The table also shows what the address bus does for the two strobe types.

Table 11-3. DMA External RAM Control

DMA Source/Destination	Active Strobes	Address Bus
Any external memory address	RD#, WR#	Increments
DMAEXTFIFO (0x7858)	FRD#, FWR#	Follows 8051 PC

For external DMA transfers, the timing of the DMA strobe signals is governed by two registers, FASTXFR and DMABURST, as previously described. Internal DMA transfers (both source and destination inside the EZ-USB FX), occur at one byte per clock, independent of stretch value or settings of the FASTXFR or DMABURST Registers.

For normal (non-DMA) 8051 external transfers (these use the MOVX instruction), the timing of the RD# and WR# strobes is governed by the stretch value programmed into the 8051 CKCON Register CKCON[2..0].

11.4 Interaction of DMA Strobe Waveforms and Stretch Bits

Figure 11-13 and *Figure 11-14* illustrate how the DMA read and write strobes are lengthened using the DSTR[2..0] Bits in the DMABURST Register, for each of the four waveforms that can be selected by the read and write mode bits in the FASTXFR Register.

11.4.1 DMA External Writes

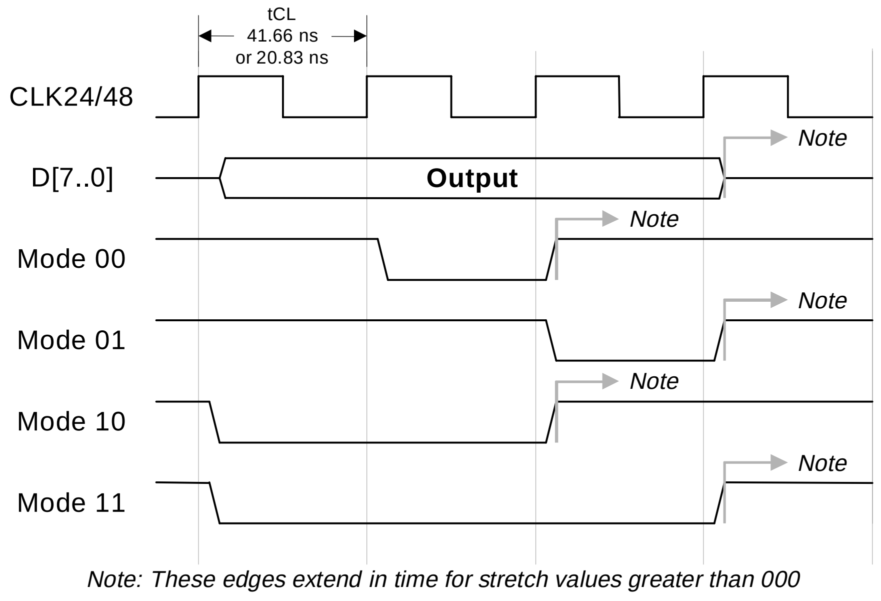


Figure 11-13. DMA Write Strobe Timing: 4 Modes Selected by FASTXFR[4..3]

The DMA Write strobes (WR# or FWR#) exhibit the four basic waveforms shown in *Figure 11-13.* for the four modes selected by FASTXFR[1..0]. The waveforms in *Figure 11-13.* apply for a stretch value of 000, with one exception: Mode 3 requires a stretch value of 001 or higher. When DMA burst writes are enabled (WB=1), the stretch values DSTR[2..0] and waveform select values (WMOD[1.0]) have no effect.

The timing relationship between the *onset* of output data and the *leading edges* of the write strobe (*Figure 11-11.*) are independent of stretch value. The effect of stretch values greater than zero is to lengthen the *data valid time* and the *trailing edges* of the write strobes, as shown in Table 11-4.

Table 11-4. Effect of Stretch Values on a Write Strobe

DSTR[2..0]	Data Valid Time	Write Strobe Width
000	3 clocks	(*Figure 11-13.*) [only modes 0-2 apply]
001	6 clocks	Add 2 clocks
010	10 clocks	Add 6 clocks
011	14 clocks	Add 10 clocks
100	18 clocks	Add 14 clocks
101	22 clocks	Add 18 clocks
110	26 clocks	Add 22 clocks
111	30 clocks	Add 26 clocks

11.4.2 DMA External Reads

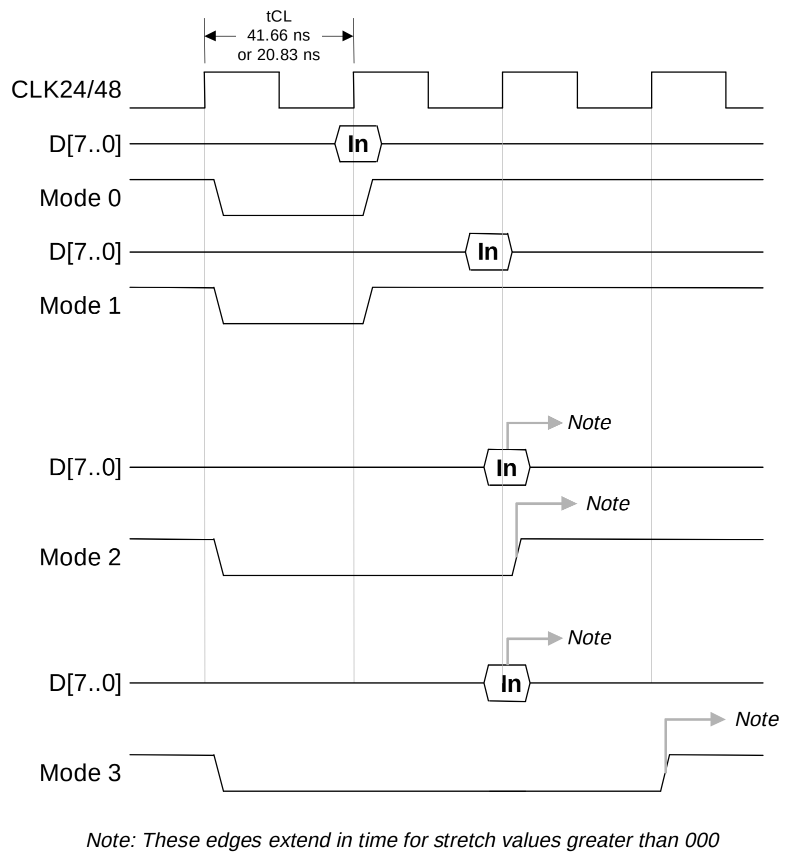


Figure 11-14. DMA Read Strobe Timing: 4 Modes Selected by FASTXFR[4..3]

The DMA Read strobes (RD# or FRD#) exhibit the four basic waveforms shown in *Figure 11-14.* for the four modes selected by FASTXFR[4..3].

11.4.2.1 Modes 0 and 1

Mode 0 and Mode 1 read strobes are intended for synchronous memories, and are unaffected by the programmed stretch value. Note that the only difference between Mode 0 and Mode 1 is that Mode 1 captures the data one clock later than Mode 0.

11.4.2.2 Modes 2 and 3

Mode 2 and Mode 3 read strobes are intended for asynchronous memories. The strobe widths are affected by the stretch values programmed into the DSTR[2..0]. Stretch values greater than zero have the effect of lengthening the data sampling time and the trailing edge of the read strobe, as shown in Table 11-5. In Mode 2, the data is always captured on the same clock edge on which the read strobe goes inactive. In Mode 3, the data is always captured one clock earlier than in Mode 2.

Table 11-5. Effect of Stretch Values on a Write Strobe

DSTR[2..0]	Mode 2 Read Strobe Width	Mode 3 Read Strobe Width
000	2 clocks	3 clocks
001	4 clocks	5 clocks
010	8 clocks	9 clocks
011	12 clocks	13 clocks
100	16 clocks	17 clocks
101	20 clocks	21 clocks
110	24 clocks	25 clocks
111	28 clocks	29 clocks

Chapter 12. EZ-USB FX Interrupts

12.1 Introduction

The EZ-USB FX enhanced 8051 responds to the interrupts shown in Table 12-1. Interrupt sources that are not present in the standard 8051 are marked with an “X” in the *New* column of the table. The three interrupts used by the USB core are shown in **bold** type.

Table 12-1. EZ-USB FX Interrupts

New	8051 Interrupt (IRQ name)	Source	Vector (hex)	Natural Priority
	IE0	INT0# Pin	03	1
	TF0	Timer 0 Overflow	0B	2
	IE1	INT1# Pin	13	3
	TF1	Timer 1 Overflow	1B	4
	RI_0 & TI_0	UART0 Rx & Tx	23	5
X	TF2	Timer 2 Overflow	2B	6
X	**Resume (WAKEUP)**	**WAKEUP# Pin or USB Core**	33	0
X	RI_1 & TI_1	UART1 Rx & Tx	3B	7
X	**USB (INT2)**	**USB Core**	43	8
X	**I^{2}C-compatible (INT3)**	**USB Core**	4B	9
X	IE4 (FIFOs)	Slave FIFOs/INT4 pin	53	10
X	IE5	INT5# Pin	5B	11
X	IE6	INT6 Pin	63	12

The **Natural Priority** column in Table 12-1 shows the 8051 interrupt priorities. As explained in *Chapter 18. "8051 Hardware Description"*, the 8051 can assign each interrupt to a high or low priority group. The 8051 resolves priorities within the groups using the natural priorities.

12.2 USB Core Interrupts

The USB core provides four interrupt groups types, which are described in the following sections:

- **Wakeup** — After the EZ-USB FX chip detects USB suspend and the 8051 has entered its idle state, the USB core responds to an external signal on its WAKEUP# pin or resumption of USB bus activity by re-starting the EZ-USB FX oscillator and resuming 8051 operation.
- **USB Signaling** — These include 16 bulk endpoint interrupts, three interrupts not specific to a particular endpoint (SOF, Suspend, USB Reset), and two interrupts for CONTROL transfers (SUTOK, SUDAV). These interrupts share the USB interrupt (INT2). Also included is an interrupt indicating that a bulk packet was NAKd.
- **I^{2}C-compatible Transfers** — (INT3).
- **Slave FIFO Flags** — (INT4)

12.3 Resume Interrupt

Chapter 14. "EZ-USB FX Power Management" describes suspend-resume signaling in detail, and presents a code example that uses the Wakeup Interrupt.

12.4 USB Signaling Interrupts

Figure 12-1. shows the 21 USB requests that share the 8051 USB (INT2) Interrupt. The bottom IRQ, EP7-OUT, is expanded in the diagram to show the logic associated with each USB interrupt request.

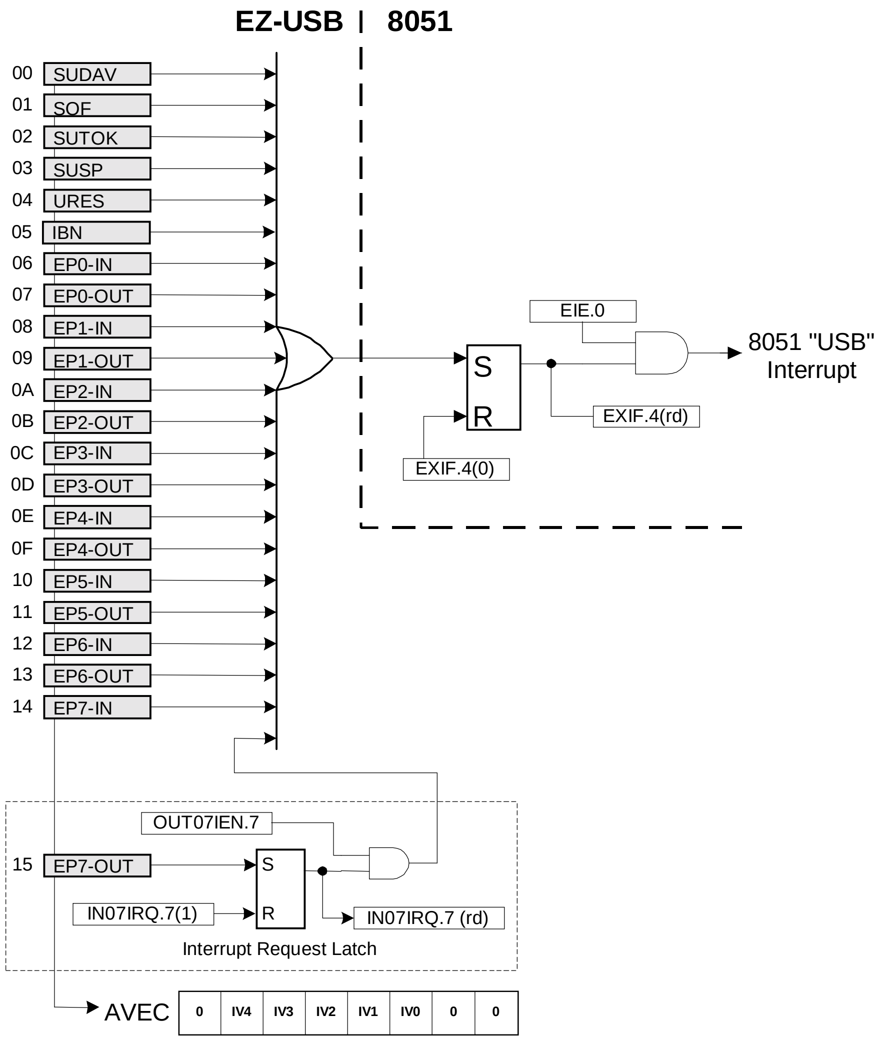


Figure 12-1. USB Interrupts

Referring to the logic inside the dotted lines, each USB interrupt source has an interrupt request latch. The USB core sets an IRQ Bit, and the 8051 clears an IRQ Bit by writing a “1” to it. The output of each latch is ANDed with an IEN (Interrupt Enable) Bit and then ORed with all the other USB interrupt request sources.

The USB core prioritizes the USB interrupts and constructs an Autovector, which appears in the AVEC register. The interrupt vector values IV[4..0] are shown to the left of the interrupt sources (shaded boxes). 00 is the highest priority, 15 is the lowest. If two USB interrupts occur simultaneously, the prioritization affects which one is first indicated in the AVEC register. If the 8051 has enabled Autovectoring, the AVEC Byte replaces byte 0x45 in 8051 program memory. This causes the USB interrupt to automatically vector to different addresses for each USB interrupt source. This mechanism is explained in detail in Section 12.10 *"USB Autovectors"*.

Due to the OR gate in *Figure 12-1.*, any of the USB interrupt sources sets the 8051 *USB* interrupt request latch, whose state appears as an interrupt request in the 8051 SFR Bit EXIF.4. The 8051 enables the USB interrupt by setting SFR Bit EIE.0.To clear the USB interrupt request the 8051 writes a zero to the EXIF.4 Bit. Note that this is the opposite of clearing any of the individual USB interrupt sources, which the 8051 does by writing a “1” to the IRQ Bit.

When a USB resource requires service (for example, a SOF token arrives or an OUT token arrives on a BULK endpoint), two things happen. First, the corresponding Interrupt Request Latch is set. Second, a pulse is generated, ORd with the other USB interrupt logic, and routed to the 8051 INT2 input. The pulse is required because INT2 is edge triggered.

When the 8051 finishes servicing a USB interrupt, it clears the particular IRQ Bit by writing a “1” to it. If any other USB interrupts are pending, the act of clearing the IRQ causes the USB core logic to generate another pulse for the highest-priority pending interrupt. If more that one is pending, they are serviced in the priority order shown in *Figure 12-1.*, starting with SUDAV (priority 00) as the highest priority, and ending with EP7-OUT (priority 15) as the lowest.

Important

It is important in any USB Interrupt Service Routine (ISR) to clear the 8051 INT2 Interrupt **before** clearing the particular USB interrupt request latch. This is because as soon as the USB interrupt is cleared, any pending USB interrupt will pulse the 8051 INT2 Input. If the INT2 Interrupt Request latch has not been previously cleared the pending interrupt is lost.

Figure 12-2. illustrates a typical USB ISR for endpoint 2-IN.

```
USB_ISR:   push   dps
           push   dpl
           push   dph
           push   dpl1
           push   dph1
           push   acc
;
           mov    a,EXIF         ; FIRST clear the USB (INT2) interrupt request
           clr    acc.4
           mov    EXIF,a         ; Note:  EXIF reg is not 8051 bit-addressable
;
           mov    dptr,#IN07IRQ  ; now clear the USB interrupt request
           mov    a,#00000100b   ; use IN2 as example
           movx   @dptr,a
;
; (perform interrupt routine stuff)
;
           pop    acc
           pop    dph1
           pop    dpl1
           pop    dph
           pop    dpl
           pop    dps
;
           reti
```

Figure 12-2. The Order of Clearing Interrupt Requests is Important

IN07IRQ — Endpoints 0-7 IN Interrupt Requests — 7FA9

b7	b6	b5	b4	b3	b2	b1	b0
IN7IR	**IN6IR**	**IN5IR**	**IN4IR**	**IN3IR**	**IN2IR**	**IN1IR**	**IN0IR**

OUT07IRQ — Endpoints 0-7 OUT Interrupt Requests — 7FAA

b7	b6	b5	b4	b3	b2	b1	b0
OUT7IR	**OUT6IR**	**OUT5IR**	**OUT4IR**	**OUT3IR**	**OUT2IR**	**OUT1IR**	**OUT0IR**

USBIRQ — USB Interrupt Request — 7FAB

b7	b6	b5	b4	b3	b2	b1	b0
-	**-**	**IBNIR**	**URESIR**	**SUSPIR**	**SUTOKIR**	**SOFIR**	**SUDAVIR**

IN07IEN — Endpoints 0-7 IN Interrupt Enables — 7FAC

b7	b6	b5	b4	b3	b2	b1	b0
IN7IEN	**IN6IEN**	**IN5IEN**	**IN4IEN**	**IN3IEN**	**IN2IEN**	**IN1IEN**	**IN0IEN**

OUT07IEN — Endpoints 0-7 OUT Interrupt Enables — 7FAD

b7	b6	b5	b4	b3	b2	b1	b0
OUT7IEN	**OUT6IEN**	**OUT5IEN**	**OUT4IEN**	**OUT3IEN**	**OUT2IEN**	**OUT1IEN**	**OUT0IEN**

USBIEN — USB Interrupt Enables — 7FAE

b7	b6	b5	b4	b3	b2	b1	b0
-	**-**	**IBNIE**	**URESIE**	**SUSPIE**	**SUTOKIE**	**SOFIE**	**SUDAVIE**

Figure 12-3. EZ-USB FX Interrupt Registers

Figure 12-3. shows the registers associated with the USB interrupts. Each interrupt source has an enable (IEN) and a request (IRQ) Bit. The 8051 sets the IEN Bit to enable the interrupt. The USB core sets an IRQ Bit high to request an interrupt, and the 8051 clears an IRQ Bit by writing a “1” to it.

The USBIEN and USBIRQ registers control the first five interrupts shown in *Figure 12-3.*. The IN07IEN and OUT07 registers control the remaining 16 USB interrupts, which correspond to the 16 bulk endpoints IN0-IN7 and OUT0-OUT7.

The following sections describe the USB interrupts in detail.

12.5 SUTOK, SUDAV Interrupts

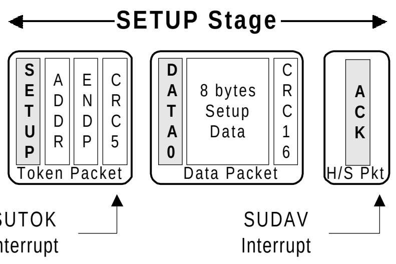


Figure 12-4. SUTOK and SUDAV Interrupts

SUTOK and SUDAV are supplied to the 8051 by EZ-USB FX CONTROL endpoint zero. The first portion of a USB CONTROL transfer is the SETUP stage shown in *Figure 12-4.*. (A full CONTROL transfer is shown in *Figure 9-1.*.) When the USB core decodes a SETUP packet, it asserts the SUTOK (SETUP Token) Interrupt Request. After the USB core has received the eight bytes error-free and copied them into eight internal registers at SETUPDAT, it asserts the SUDAV Interrupt Request.

The 8051 program responds to the SUDAV Interrupt by reading the eight SETUP data bytes in order to decode the USB request (*Chapter 9. "EZ-USB FX Endpoint Zero"*).

The SUTOK Interrupt is provided to give advance warning that the eight register bytes at SETUP-DAT are about to be over-written. It is useful for debug and diagnostic purposes.

12.6 SOF Interrupt

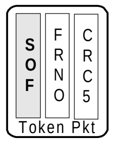


Figure 12-5. A Start Of Frame (SOF) Packet

USB Start of Frame Interrupt Requests occur every millisecond. When the USB core receives an SOF packet, it copies the eleven-bit frame number (FRNO in *Figure 12-5.*) into the USBFRAMEH and USBFRAMEL registers, and activates the SOF Interrupt Request. The 8051 services all isochronous endpoint data as a result of the SOF Interrupt.

12.7 Suspend Interrupt

If the EZ-USB FX detects 3 ms of no bus activity, it activates the SUSP (Suspend) Interrupt Request. A full description of Suspend-Resume signaling appears in *Chapter 14. "EZ-USB FX Power Management".*

12.8 USB RESET Interrupt

The USB signals a bus reset by driving both D+ and D- low for at least 10 ms. When the USB core detects the onset of USB bus reset, it activates the URES Interrupt Request.

12.9 Bulk Endpoint Interrupts

The remaining 16 USB interrupt requests are indexed to the 16 EZ-USB FX bulk endpoints. The USB core activates a bulk interrupt request when the endpoint buffer requires service. For an OUT endpoint, the interrupt request signifies that OUT data has been sent from the host, validated by the USB core, and is sitting in the endpoint buffer memory. For an IN endpoint, the interrupt request signifies that the data previously loaded by the 8051 into the IN endpoint buffer has been read and validated by the host, making the IN endpoint buffer ready to accept new data.

The USB core sets an endpoint's interrupt request bit when the endpoint's busy bit (in the endpoint CS register) goes low, indicating that the endpoint buffer is available to the 8051. For example, when endpoint 4-OUT receives a data packet, the busy bit in the OUT4CS register goes low, and OUT07IRQ.4 goes high, requesting the endpoint 4-OUT Interrupt.

12.10 USB Autovectors

The USB interrupt is shared by 22 interrupt sources.To save the code and processing time required to sort out which USB interrupt occurred, the USB core provides a second level of inter-rupt vectoring, called *Autovectoring*. When the 8051 takes a USB interrupt, it pushes the program counter onto its stack, and then executes a jump to address 43, where it expects to find a jump instruction to the INT2 service routine. The 8051 jump instruction is encoded as follows:

Table 12-2. 8051 JUMP Instruction

Address	Op-Code	Hex Value
0043	Jump	0x02
0044	AddrH	0xHH
0045	AddrL	0xLL

If Autovectoring is enabled (AVEN=1 in the USBBAV register), the USB core substitutes its AVEC Byte for the byte at address 0x0045. Therefore, if the programmer pre-loads the high byte (“page”) of a jump table address at location 0x0044, the core-inserted byte at 0x45 will automatically direct the JUMP to one of 21 addresses within the page. In the jump table, the programmer then puts a series of jump instructions to each particular ISR.

Table 12-3. A Typical USB Jump Table

Table Offset	Instruction
00	JMP SUDAV_ISR
04	JMP SOF_ISR
08	JMP SUTOK_ISR
0C	JMP SUSPEND_ISR
10	JMP USBRESET_ISR
14	JMP IBN_ISR
18	JMP EP0IN _ISR
1C	JMP EP0OUT_ISR
20	JMP IN1BUF_ISR
24	JMP EP1OUT_ISR
28	JMP EP2IN_ISR
2C	JMP EP2OUT_ISR
30	JMP EP3IN_ISR
34	JMP EP3OUT_ISR
38	JMP EP4IN_ISR
3C	JMP EP4OUT_ISR
40	JMP EP5IN_ISR
44	JMP EP5OUT_ISR
48	JMP EP6IN_ISR
4C	JMP EP6OUT_ISR
50	JMP EP7IN_ISR
54	JMP EP7OUT_ISR

12.11 Autovector Coding

A detailed example of a program that uses Autovectoring is presented in Section 6.14 *"Interrupt Bulk Transfer Example"*. The coding steps are summarized here.To employ EZ-USB FX Autovectoring:

1. Insert a jump instruction at 0x43 to a table of jump instructions to the various USB interrupt service routines. Make sure the jump table starts on a 0x100 byte page boundary.
2. Code the jump table with jump instructions to each individual USB interrupt service routine. This table has two important requirements, arising from the format of the AVEC Byte (zero-based, with 2 LSBs set to 0):

- It must begin on a page boundary (address 0xNN00).
- The jump instructions must be four bytes apart.

3. The interrupt service routines can be placed anywhere in memory.
4. Write initialization code to enable the USB interrupt (INT2) and Autovectoring.

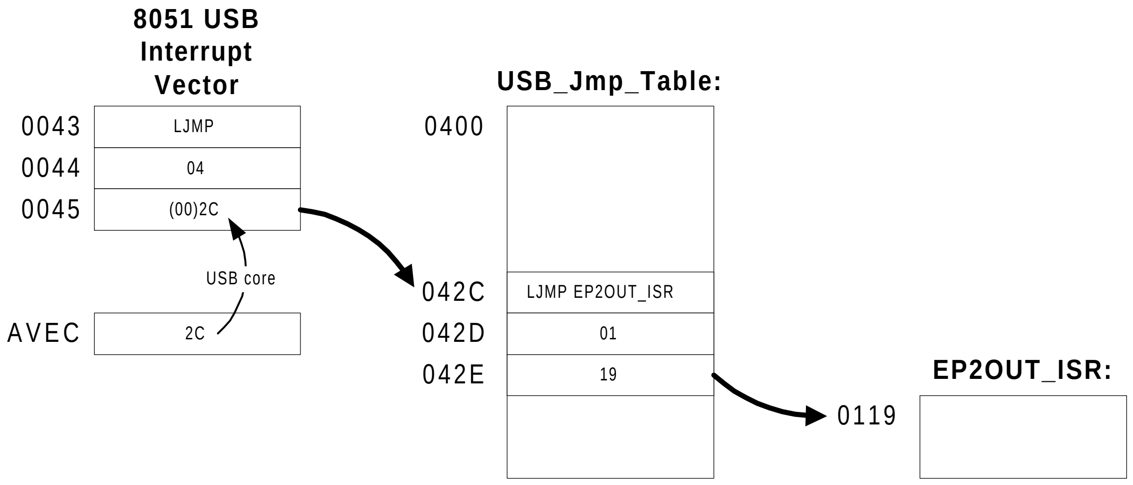
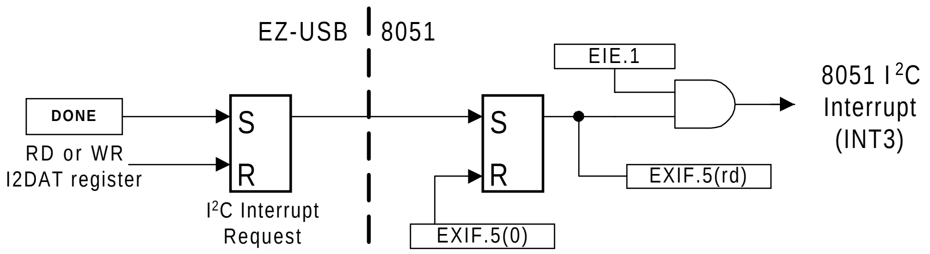


Figure 12-6. The Autovector Mechanism in Action

Figure 12-6. illustrates an ISR that services endpoint 2-OUT. When endpoint 2-OUT requires service, the USB core activates the USB interrupt request, vectoring the 8051 to location 0x43. The jump instruction at this location, which was originally coded as “LJMP 04-**00**” becomes “LJMP 04-**2C**” due to the USB core substituting **2C** as the Autovector byte for Endpoint 2-OUT (Table 12-3). The 8051 jumps to 042C, where it executes the jump instruction to the endpoint 2-OUT ISR, shown in this example at address 0119. Once the 8051 takes the vector at 0043, initiation of the endpoint-specific ISR takes only eight 8051 cycles.

12.12 I²C-Compatible Interrupt



I2CS	START	STOP	LASTRD	ID1	ID0	BERR	ACK	DONE
I2DAT	D7	D6	D5	D4	D3	D2	D1	D0

Figure 12-7. I²C-Compatible Interrupt Enable Bits and Registers

Chapter 4. "EZ-USB FX Input/Output" describes the 8051 interface to the EZ-USB FX I^{2}C-compatible controller. The 8051 uses two registers, I2CS (I^{2}C-compatible Control and Status) and I2DAT (I^{2}C-compatible Data) to transfer data over the I^{2}C-compatible bus. The USB core signals completion of a byte transfer by setting the DONE Bit (I2CS.0) high, which also sets an I^{2}C-compatible Interrupt Request latch (*Figure 12-7.*). This interrupt request is routed to the 8051 INT3 Interrupt.

The 8051 enables the I^{2}C-compatible interrupt by setting EIE.1=1. The 8051 determines the state of the interrupt request flag by reading EXIF.5, and resets the INT3 Interrupt Request by writing a zero to EXIF.5. Any 8051 read or write to the I2DAT or I2CS register automatically clears the I^{2}C-compatible Interrupt Request.

12.13 In Bulk NAK Interrupt

The EZ-USB FX family responds to an IN token from the host by transmitting bytes that the 8051 has loaded into an IN endpoint buffer, and *armed* by loading the IN endpoint's byte count register. After the host successfully receives the IN data, the 8051 receives an EP-IN Interrupt, signifying that the IN endpoint buffer is once again ready to accept data.

In some situations, the host may send IN tokens before the 8051 has loaded and armed an IN endpoint. To alert the 8051 that an IN endpoint is being *pinged*, a set of interrupts, one per IN endpoint, indicates that an IN endpoint just sent a NAK to the host. This happens when the host sends an IN token and the IN endpoint does not have data (yet) for the host. This set of interrupts is called “IBN” (IN Bulk NAK). Its INT2 Autovector is 05, which was previously reserved in the EZ-USB family.

The IBN Interrupt Requests and enables are controlled by two registers:

IBNIRQ **IN Bulk NAK Interrupt Requests** **7FB0**

b7	b6	b5	b4	b3	b2	b1	b0
EP7IR	**EP6IR**	**EP5IR**	**EP4IR**	**EP3IR**	**EP2IR**	**EP1IR**	**EP0IR**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 12-8. IN Bulk NAK Interrupt Request Register

IBNEN **IN Bulk NAK Interrupt Enables** **7FB1**

b7	b6	b5	b4	b3	b2	b1	b0
EP7IE	**EP6IE**	**EP5IE**	**EP4IE**	**EP3IE**	**EP2IE**	**EP1IE**	**EP0IE**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	0	0	0	0	0	0	0

Figure 12-9. IN Bulk NAK Interrupt Enable Register

As with all other EZ-USB FX interrupt requests, the 8051 clears an IBNIRQ Bit by writing a “1” to it. Each of the individual IN endpoints may be enabled for an IBN Interrupt using the IBNEN register.

12.14 I^{2}C-Compatible STOP Complete Interrupt

I2CMODE **I^{2}C-Compatible Mode** **7FA7**

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	0	0	**STOPIE**	**400KHZ**
R	R	R	R	R	R	R/W	R/W
0	0	0	0	0	0	0	0

Figure 12-10. I^{2}C-Compatible Mode Register

The I^{2}C-compatible interrupt includes one additional interrupt source, a 1-0 transition of the STOP Bit. To enable this interrupt, set the STOPIE Bit in the I2CMODE register. The 8051 determines the interrupt source by checking the DONE and STOP Bits in the I2CS register.

I2CS **I^{2}C-Compatible Control and Status** **7FA5**

b7	b6	b5	b4	b3	b2	b1	b0
START	**STOP**	**LASTRD**	**ID1**	**ID0**	**BERR**	**ACK**	**DONE**
R/W	R/W	R/W	R	R	R	R	R
0	0	0	X	X	0	0	0

Figure 12-11. I^{2}C-Compatible Control and Status Register

I2DAT **I^{2}C-Compatible Data** **7FA6**

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 12-12. I^{2}C-Compatible Data

The two registers that the 8051 uses to control I^{2}C-compatible transfers are shown above. In the EZ-USB family, an I^{2}C-compatible Interrupt Request occurs on INT3 whenever:

- The DONE Bit (I2CS.0) makes a zero-to-one transition, or
  (This interrupt signals the 8051 that the I^{2}C-compatible controller is ready for another command.)
- The STOP Bit (I2CS.6) makes a one-to-zero transition.

The 8051 concludes I^{2}C-compatible transfers by setting the STOP Bit (I2CS.6). When the STOP condition has been sent over the I^{2}C-compatible bus, the I^{2}C-compatible controller resets I2CS.6 to zero. *During the time the I^{2}C-compatible controller is generating the stop condition, it ignores accesses to the I2CS and I2DAT registers.* The 8051 code should therefore check the STOP Bit for zero before writing new data to I2CS or I2DAT.

The STOP Bit completion interrupt is enabled by setting I2CMODE.1 to “1.”

12.15 Slave FIFO Interrupt (INT4)

INT4IVEC **Interrupt 4 Autovector** **785D**

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**I4V3**	**I4V2**	**I4V1**	**I4V0**	**0**	**0**
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Figure 12-13. Interrupt 4 Autovector

The EZ-USB FX slave FIFOs contain various flags to alert the 8051 when a FIFO needs attention. These flags are encoded into the INT4 autovector, which the 8051 can read in the INT4IVEC register. The encoded values for each INT4 source are shown in Table 12-4.

INT4SETUP **Interrupt 4 Setup** **785E**

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**INT4SFR**	**INTRNL**	**AV4EN**
R	R	R	R	R	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 12-14. Interrupt 4 Setup

Bit 0: **AV4EN** *Enable INT4 Autovector*

To streamline the 8051 code that deals with the FIFO interrupts, the 8051 INT4 vector locations use the same autovectoring mechanism as the USB (INT2) Interrupt when the AV4EN Bit (INT4SETUP.0) is set. Referring to Table 12-4, when a FIFO flag interrupt occurs with AV4EN=1, internal logic replaces the third byte of the jump instruction at location 0x53 with a different address for each FIFO interrupt source.

Table 12-4. Autovector for INT4*

8051 Addr	Instruction	Notes
0x53	LJMP	Loc 53-55 are the INT4 Interrupt Vector
0x54	AddrH	
0x55	AddrL	EZ-USB FX logic replaces this byte when **AV4EN=1**

*Table 12-5 shows bytes inserted at Address 55H

To set up autovectoring, the user places an LJMP instruction at location 0x53, which jumps to a table of instructions, which jump to the various FIFO ISRs. Then, every FIFO interrupt automatically vectors to the individual interrupt service routine for the particular FIFO flag. The autovector mechanism saves the 8051 from having to check for the source of each interrupt shared on INT4.

Table 12-4 shows the IVEC4 values for the various FIFO interrupts. The last three interrupt vectors in Table 12-4 are not FIFO related, and are described in other chapters. The bytes inserted by the EZ-USB FX logic (the low-address byte of the LJMP instruction) are separated by four to allow four bytes per LJMP instruction in the jump table. (An 8051 LJMP instruction requires 3 bytes).

Table 12-5. INT4 Autovectors

IVEC4 Value	Byte Inserted at 0x55	Source	Meaning
0x40	0x80	AINPF	A-IN FIFO Programmable Flag
0x44	0x84	BINPF	B-IN FIFO Programmable Flag
0x48	0x88	AOUTPF	A-OUT FIFO Programmable Flag
0x4C	0x8C	BOUTPF	B-OUT FIFO Programmable Flag
0x50	0x90	AINEF	A-IN FIFO Empty Flag
0x54	0x94	BINEF	B-IN FIFO Empty Flag
0x58	0x98	AOUTEF	A-OUT FIFO Empty Flag
0x5C	0x9C	AINEF	B-OUT FIFO Empty Flag
0x60	0xA0	AINFF	A-IN FIFO Full Flag
0x64	0xA4	BINFF	B-IN FIFO Full Flag
0x68	0xA8	AOUTFF	AOUT FIFO Full Flag
0x6C	0xAC	BOUTFF	B-OUT FIFO Full Flag
0x70	0xB0	GPIFDONE	(See *Chapter 8. "General Programmable Interface (GPIF)"*)
0x74	0xB4	GPIFWR	(See *Chapter 8. "General Programmable Interface (GPIF)"*
0x78	0xB8	DMADONE	(See *Chapter 8. "General Programmable Interface (GPIF)"*)

Note that the bytes inserted for the INT4 autovector start at0x80, rather that 0x00. This is because another EZ-USB FX autovector, for INT2 (used for all USB interrupts), uses jump table offsets from 0x00 to 0x57. The autovector jump table must start on a page boundary (8051 address XX00). Therefore, separating the two groups of jumps allows a single page of 8051 memory to be used for both INT2 and INT4 jump tables. The INT2 jump table can start at 0x00, and the INT4 jump table can start at 0x80, sharing the same page.

If two or more INT4 Interrupt Requests occur simultaneously, they are serviced in the order shown in Table 12-5, with AINPF having the highest priority and DMADONE the lowest. Pending interrupt requests remain pending while a higher level interrupt is serviced.

As with the USB (INT2) Interrupts, the INT4 Interrupt Request must be cleared in the ISR (Interrupt Service Routine) before clearing the individual slave FIFO interrupt request bit.

Bit 1: **INTRNL** *INT4 Source*

This bit selects the interrupt source for 8051 INT4. If INTRNL=0, INT4 is supplied from the EZ-USB FX INT4 pin. If INTRNL=1, INT4 is supplied from the slave FIFO interrupt unit, with the interrupt sources shown in Table 12-5.

To enable the INT4 pin instead of PB4, set PORTBCFG.4=1.

Bit 2: **INT4SFR** *Enable SFR clearing of INT4*

The 8051 sets INT4SFR=1 to enable clearing of the pending INT4 Interrupt Request currently being serviced by writing any value to INT4CLR (SFR at 0xA2).

Chapter 13. EZ-USB FX Resets

13.1 Introduction

The EZ-USB FX chip has three resets:

- A Power-On Reset (POR), which turns on the EZ-USB FX chip in a known state.
- An 8051 reset, controlled by the USB core.
- A USB bus reset, sent by the host to reset a device.

This chapter describes the effects of these three resets.

13.2 EZ-USB FX Power-On Reset (POR)

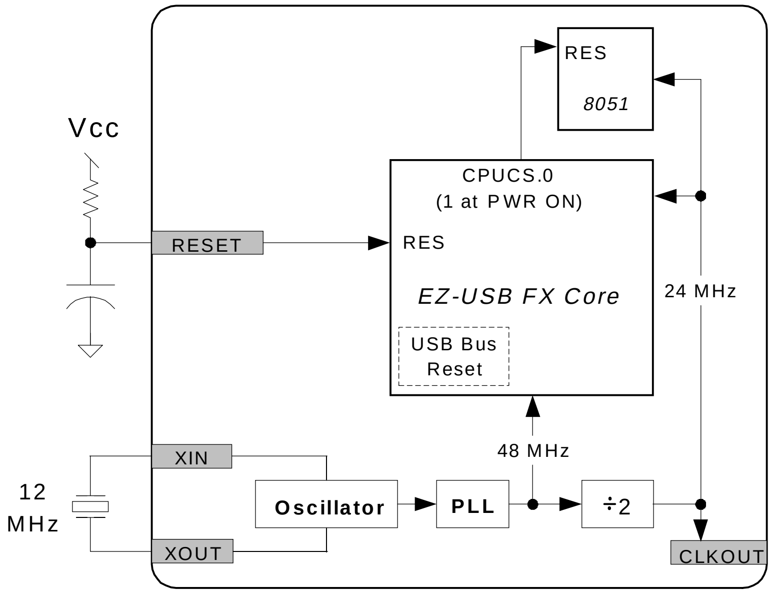


Figure 13-1. EZ-USB FX Resets

When power is first applied to the EZ-USB FX chip, the external R-C circuit holds the USB core in reset until the on-chip PLL stabilizes. The CLKOUT pin is active as soon as power is applied. The 8051 may clear an EZ-USB FX control bit, CLKOUTOE (CPUCS.1), to inhibit the CLKOUT output pin for EMI-sensitive applications that do not need this signal. External logic can force a chip reset by pulling the active-low RESET pin LO. The RESET pin is normally connected to GND through a 1 ΩF capacitor and to Vcc through a 10-K resistor (*Figure 13-1*). The oscillator and PLL are unaffected by the state of the RESET pin.

The CLKOUT signal is active while RESET = LO. When RESET returns HI, the activity on the CLKOUT pin depends on whether or not the EZ-USB FX chip is in suspend state. If in suspend, CLKOUT stops. Resumption of USB bus activity or asserting the WAKEUP# pin LO re-starts the CLKOUT signal.

Power-on default values for all EZ-USB FX register bits are shown in *Chapter 15. "EZ-USB FX Registers"*. Table 13-1 summarizes reset states that affect USB device operation. Note that the term “Power-On Reset” refers to a reset initiated by application of power, *or* by assertion of the RESET pin.

Table 13-1. EZ-USB FX States After Power-On Reset (POR)

Item	Register	Default Value	Comment
1	Endpoint Data	`xxxxxxxx`	
2	Byte Counts	`xxxxxxxx`	
3	CPUCS	`rrrr0011`	rrrr=rev number, b1 =CLKOUTOE, b0=8051RES
4	PORT Configs	`00000000`	IO, not alternate functions
5	PORT Registers	`xxxxxxxx`	
6	PORT OEs	`00000000`	Inputs
7	Interrupt Enables	`00000000`	Disabled
8	Interrupt Reqs	`00000000`	Cleared
9	Bulk IN C/S	`00000000`	Bulk IN endpoints not busy (unarmed)
10	Bulk OUT C/S*	`00000000`	Bulk OUT endpoints busy (armed)
11	Toggle Bits	`00000000`	Data toggles = 0
12	USBCS	`00000100`	RENUM=0, DISCOE=1 (Discon pin drives)
13	FNADDR	`00000000`	USB Function Address
14	IN07VAL	`01010111`	EP0,1,2,4,6 IN valid
15	OUT07VAL	`01010101`	EP0,2,4,6 OUT valid
16	INISOVAL	`00000111`	EP8,9,10 IN valid
17	OUTISOVAL	`00000111`	EP8,9,10 OUT valid
18	USBPAIR	`0x000000`	ISOsend0 (b7) = 0, no pairing
19	USBBAV	`00000000`	Break condition cleared, no Autovector
20	Configuration	00000000	Internal USB core value
21	Alternate Setting	00000000	Internal USB core value

* In this state (8051 reset) an OUT endpoint ACKs all OUT requests.

* When the 8051 is released from reset, the EZ-USB FX automatically arms the Bulk OUT endpoints by setting their CS Registers to 000000010b. This causes the next OUT request to be ACK'd and subsequent OUT transfers to be NAK'd until the 8051 loads the endpoint byte count register.

Table 13-1 documents the following summary list of states at power on.

- Endpoint data buffers and byte counts are un-initialized (1,2).
- The 8051 is held in reset, and the CLKOUT pin is enabled (3).
- All port pins are configured as input ports (4-6).
- USB interrupts are disabled, and USB interrupt requests are cleared (7-8).
- Bulk IN and OUT endpoints are unarmed, and their stall bits are cleared (9). The USB core will NAK IN tokens and ACK OUT tokens while the 8051 is reset. OUT endpoints are enabled for one OUT transfer when the 8051 is released from reset.
- Endpoint toggle bits are cleared (11).
- The RENUM Bit is cleared. This means that the USB core, and not the 8051, initially responds to USB device requests (12).
- The USB function address register is set to zero (13).
- The endpoint valid bits are set to match the endpoints used by the default USB device (14-17).
- Endpoint pairing is disabled. Also, ISOSend0=0, meaning that if an Isochronous endpoint receives an IN token without being loaded by the 8051 in the previous frame, the USB core does not generate any response (18).
- The breakpoint condition is cleared, and autovectoring is turned off (19).
- Configuration Zero, Alternate Setting Zero is in effect (20-21).

13.3 Releasing the 8051 Reset

The EZ-USB FX register bit CPUCS.0 resets the 8051. This bit is LO at power-on, initially holding the 8051 in reset. There are three ways to release the 8051 from reset:

- By the host, as the final step of a RAM download.

- Automatically, at the end of an EEPROM load (assuming the EEPROM is correctly programmed).
- Automatically, when external ROM is used (EA=1).

13.3.1 RAM Download

Once enumerated, the host can download code into the EZ-USB FX RAM using the “Firmware Load” vendor request (*Chapter 9. "EZ-USB FX Endpoint Zero"*). The last packet loaded writes 0 to the CPUCS Register, which clears the 8051 RESET Bit.

The other bit in the CPUCS Register, CLKOUTOE, is writable only by the 8051. The host writing a zero byte to this register does not turn off the CLKOUT signal.

13.3.2 EEPROM Load

Chapter 5 describes the EEPROM boot loads in detail. Briefly, at power-on, the USB core checks for the presence of an EEPROM on its I^2C-compatible bus. If found, it reads the first EEPROM Byte. If it reads 0xB6 as the first byte, the USB core downloads 8051 code from the EEPROM into internal RAM. The last operation in a “B6” load writes 0x00 to the CPUCS Register (at 0x7F92), which releases the 8051 from reset.

13.3.3 External ROM

EZ-USB FX systems can use external program memory containing 8051 code and USB device descriptors, which include the VID/DID/PID Bytes. Because these systems do not require an I^2C-compatible EEPROM to supply the VID/DID/PID, the USB core automatically releases 8051 reset when:

- EA=1 (External code memory), *and*
- No “B4/B6” EEPROM is detected on the I^2C-compatible bus.

Under these conditions, the USB core also sets the RENUM Bit to “1,” giving USB control to the 8051.

13.4 8051 Reset Effects

Once the 8051 is running, the USB host may reset the 8051 by downloading the value 0x01 to the CPUCS Register. The host might do this in preparation for loading code overlays, effectively magnifying the size of the internal EZ-USB FX RAM. For such applications it is important to know the state of the EZ-USB FX chip during and after an 8051 reset. In this section, this particular reset is called an “8051 Reset,” and should not be confused with the POR described in Section 13.2. *"EZ-USB FX Power-On Reset (POR)"*. This discussion applies only to the condition where the EZ-USB FX chip is powered, and the 8051 is reset by the host setting the CPUCS Register to 0.

The basic USB device configuration remains intact through an 8051 reset. Valid endpoints remain valid, the USB function address remains the same, and the I/O ports retain their configurations and values. Stalled endpoints remain stalled, and data toggles don’t change. The only effects of an 8051 reset are as follows:

- USB (INT2) interrupts are disabled, but pending interrupt requests remain pending.

  When the 8051 comes out of reset, pending interrupts are kept pending, but disabled. This gives the firmware writer the choice of acting on pre-8051-reset USB events, or ignoring them by clearing the pending interrupt(s) before enabling INT2.

- **During** the 8051 Reset, all bulk IN endpoints are unarmed, causing the USB core to NAK IN tokens; OUT tokens are ACK’d.
- **After** the 8051 Reset is removed, the OUT bulk endpoints are automatically armed. OUT endpoints are thus ready to accept *one* OUT packet before 8051 intervention is required.
- The breakpoint condition is cleared.

  USBBAV.3, the breakpoint BREAK Bit, is cleared.

The other bits in the USBBAV Register are unaffected. The RENUM Bit is not affected by an 8051 reset.

13.5 USB Bus Reset

The host signals a USB Bus Reset by driving an SE0 state (both D+ and D- data lines low) for a minimum of 10 ms. The USB core senses this condition, requests the 8051 USB Interrupt (INT2), and supplies the interrupt vector for a USB Reset. A USB reset affects the EZ-USB FX resources as shown in Table 13-2.

Table 13-2. EZ-USB FX States After a USB Bus Reset

Item	Register	Default Value	Comment
1	Endpt Data	uuuuuuuu	u = unchanged
2	Byte Counts	uuuuuuuu	
3	CPUCS	uuuuuuuu	
4	PORT Configs	uuuuuuuu	
5	PORT Registers	uuuuuuuu	
6	PORT OEs	uuuuuuuu	
7	Interrupt Enables	uuuuuuuu	
8	Interrupt Reqs	uuuuuuuu	
9	Bulk IN C/S	00000000	unarm
10	Bulk OUT C/S	uuuuuuuu	retain armed/unarmed state
11	Toggle Bits	00000000	
12	USBCS	uuuuuuuu	RENUM Bit unchanged
13	FNADDR	00000000	USB Function Address
14	IN07VAL	uuuuuuuu	
15	OUT07VAL	uuuuuuuu	
16	INISOVAL	uuuuuuuu	
17	OUTISOVAL	uuuuuuuu	
18	USBPAIR	uuuuuuuu	
19	Configuration	00000000	
20	Alternate Setting	00000000	

A USB bus reset leaves most EZ-USB FX resources unchanged. From Table 13-2, after USB bus reset:

- The USB core *unarms* all Bulk IN endpoints (9). Data loaded by the 8051 into an IN endpoint buffer remains there, and the 8051 firmware can either re-send it by loading the endpoint byte count register to re-arm the transfer, or send new data by re-loading the IN buffer before re-arming the endpoint.
- Bulk OUT endpoints retain their *busy* states (10). Data sent by the host to an OUT endpoint buffer remains in the buffer, and the 8051 firmware can either read the data or reject it as *stale* simply by not reading it. In either case, the 8051 loads a dummy value to the endpoint byte count Register to re-arm OUT transfers.
- Toggle bits are cleared (11).
- The device address is reset to zero (13).

Note from item 12 ofTable 13-2 that the RENUM Bit is unchanged after a USB bus reset. Therefore, if a device has ReNumerated™ and loaded a new personality, it retains the new personality through a USB bus reset.

13.6 EZ-USB FX Disconnect

Table 13-3. Effects of an EZ-USB FX Disconnect and Re-connect

Item	Register	Default Value	Comment
1	Endpt Data	uuuuuuuu	u = unchanged
2	Byte Counts	uuuuuuuu	
3	CPUCS	uuuuuuuu	
4	PORT Configs	uuuuuuuu	
5	PORT Registers	uuuuuuuu	
6	PORT OEs	uuuuuuuu	
7	Interrupt Enables	uuuuuuuu	
8	Interrupt Reqs	uuuuuuuu	
9	Bulk IN C/S	00000000	unarm, clear stall bit
10	Bulk OUT C/S	00000010	Arm, clear stall bit
11	Toggle Bits	00000000	reset
12	USBCS	uuuuuuuu	RENUM Bit unchanged
13	FNADDR	00000000	USB Function Address
14	IN07VAL	uuuuuuuu	
15	OUT07VAL	uuuuuuuu	
16	INISOVAL	uuuuuuuu	
17	OUTISOVAL	uuuuuuuu	
18	USBPAIR	uuuuuuuu	
19	Configuration	00000000	
20	Alternate Setting	00000000	

Although not strictly a "reset," when the EZ-USB FX simulates a disconnect-reconnect to ReNumerate™, there are effects on the USB core:

- Bulk IN endpoints are unarmed, and bulk OUT endpoints are armed (9-10).
- Endpoint STALL Bits are cleared (9-10).
- Data toggles are reset (11).
- The function address is reset to zero (13).

- The configuration is reset to zero (19).
- Alternate settings are reset to zero (20).

13.7 Reset Summary

Table 13-4. Effects of Various EZ-USB FX Resets (“U” Means “Unaffected”)

Resource	RESET pin	USB Bus Reset	Disconnect	8051 Reset
8051 Reset	reset	U	U	N/A
EP0-7 IN EPs	unarm	unarm	unarm	unarm
EP0-7 OUT EPs	arm	U	arm	arm all /arm one
Breakpoint	reset	U	U	reset
Stall Bits	reset	U	reset	U
Interrupt Enables	reset	U	U	reset
Interrupt Reqs	reset	U	U	U
CLKOUT	run	U	U	U
Data Toggles	reset	reset	reset	U
Function Address	reset	reset	reset	U
Configuration	0	0	0	U
ReNum	0	U	U	U

Table 13-4 summarizes the effects of the four EZ-USB FX resets.

The I^{2}C-compatible controller is not reset for any of the conditions laid out in Table 13-4. *Only the EZ-USB FX RESET pin resets it.*

Chapter 14. EZ-USB FX Power Management

14.1 Introduction

The USB host can suspend a device to put it into power-down mode. When the USB signals a SUSPEND operation, the EZ-USB FX chip goes through a sequence of steps to allow the 8051 to first turn off external power-consuming subsystems, and then enter an ultra-low-power mode by turning off its oscillator, its D+ and D- drivers, and other SIE circuits. Once suspended, the EZ-USB FX chip is awakened either by resumption of USB bus activity or by assertion of its WAKEUP# pin. This chapter describes the suspend-resume mechanism.

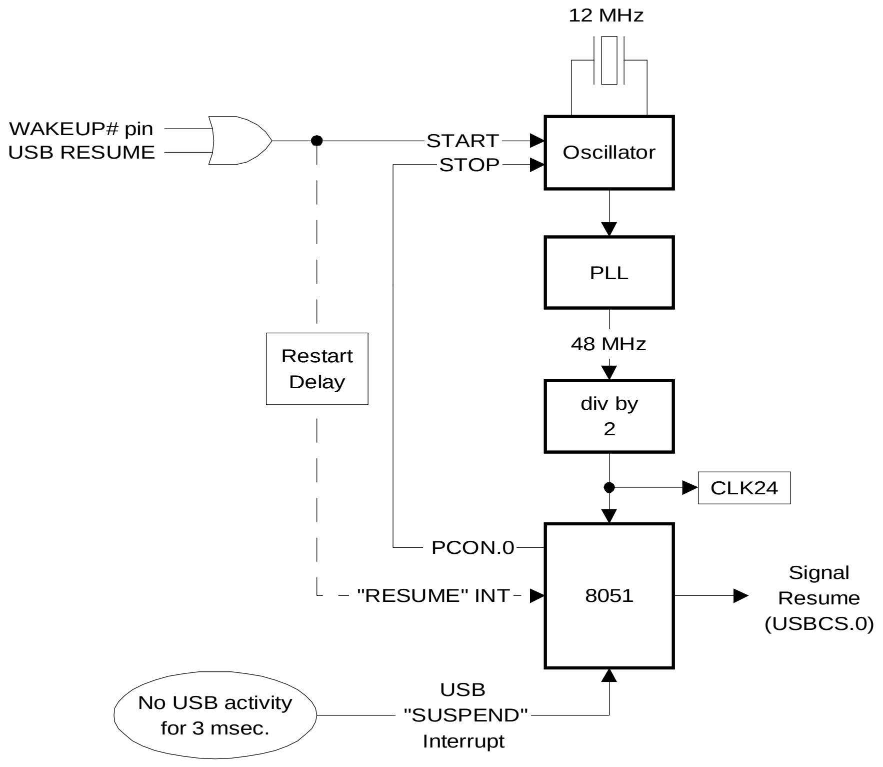


Figure 14-1. Suspend-Resume Control

Figure 14-1 illustrates the EZ-USB FX logic that implements USB suspend and resume. These operations are explained in the next sections.

14.2 Suspend

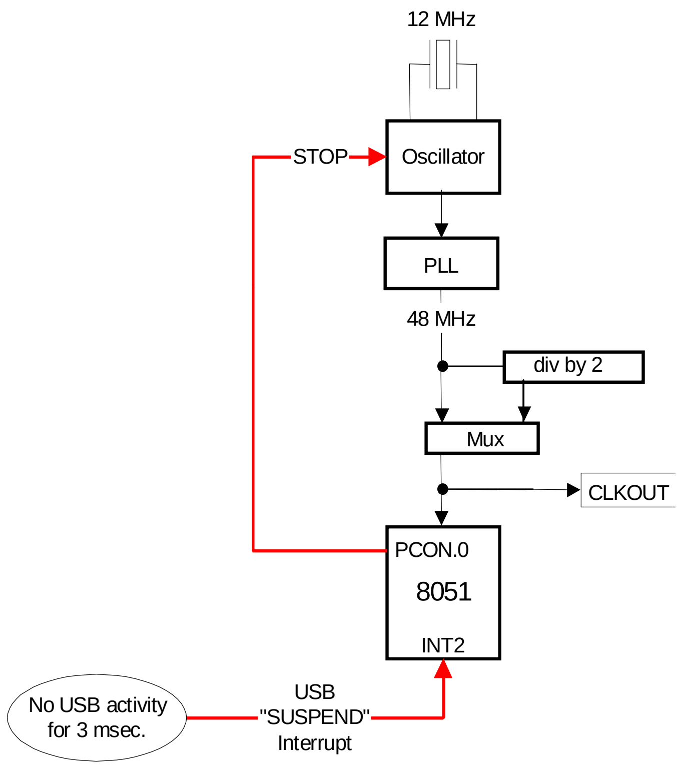


Figure 14-2. EZ-USB FX Suspend Sequence

A USB device recognizes SUSPEND as 3 ms of a bus idle (“J”) state. The USB core alerts the 8051 by asserting the USB (INT2) Interrupt and the SUSPEND Interrupt vector. This gives the 8051 code a chance to perform power conservation housekeeping before shutting down the oscillator (by setting PCON.0=1).

For bus-powered devices, the 8051 code must respond to the SUSPEND Interrupt by taking the following steps:

1. Perform any necessary housekeeping, such as shutting off external power-consuming devices.
2. Set bit 0 of the PCON SFR (Special Function Register). This has two effects:

    - The 8051 enters its *idle* mode.
    - The 8051 sends an internal signal to the USB core, causing it to turn off the oscillator and PLL.

These actions put the EZ-USB FX chip into a low-power mode, as required by the USB Specification.

Self-powered devices may perform the above power saving steps, but it is not required by the USB Specification.

14.3 Resume

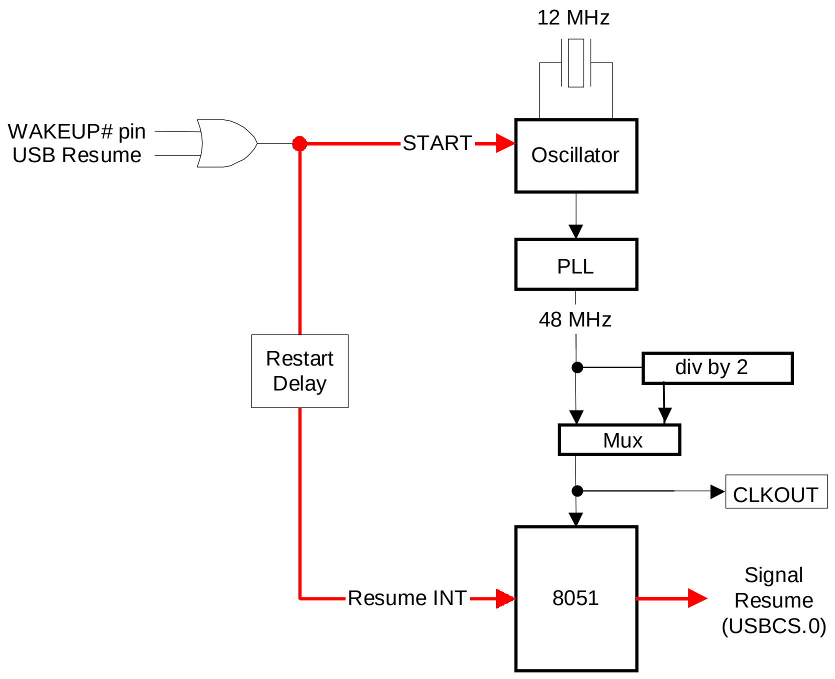


Figure 14-3. EZ-USB FX Resume Sequence

When the 8051 sets PCON.0, it enters an idle state. 8051 execution is resumed by activation of its RESUME Interrupt (Vector 33). When external logic pulls WAKEUP# low (for example, when a keyboard key is pressed or a modem receives a ring signal) or USB bus activity resumes, the USB core re-starts the 12-MHz oscillator, allowing the 8051 to recognize the RESUME Interrupt and continue executing instructions.

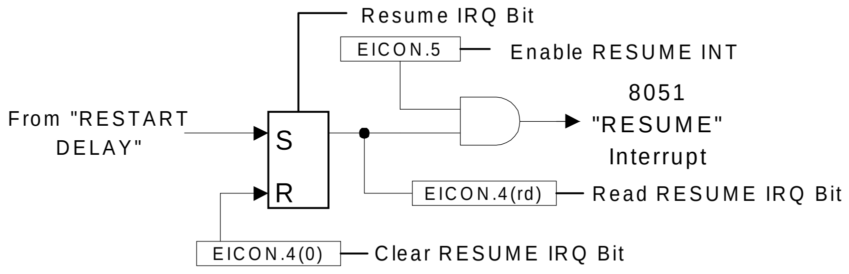


Figure 14-4. EZ-USB FX RESUME Interrupt

Figure 14-4 shows the two 8051 EICON SFR Bits associated with the RESUME Interrupt. The USB core asserts the resume signal when the USB core senses a USB Global Resume, or when the EZ-USB FX WAKEUP# pin is pulled low.

The 8051 enables the RESUME Interrupt by setting EICON.5.

```
setb     EICON.5     ; enable Resume Int
```

The 8051 can read the RESUME Interrupt request bit in EICON.4.

The 8051 clears the interrupt request bit by writing a zero to EICON.4.

```
Resume_isr:   clr      EICON.4     ; clear the RESUME
                                   ; Interrupt Request Bit
              reti
```

The EZ-USB FX oscillator re-starts when:

- USB bus activity resumes (labelled “USB Resume” in *Figure 14-3*), or
- External logic asserts the EZ-USB FX WAKEUP# pin low.

After an oscillator stabilization time, the USB core asserts the 8051 Resume Interrupt. (See *Figure 12-1*). This causes the 8051 to exit its *idle* mode. The Resume Interrupt is the highest priority 8051 interrupt. *It is always enabled, unaffected by the EA Bit. However, it is affected by the EICON.5 Bit (ERESI).*

The resume ISR clears the interrupt request flag, and executes a "reti" (return from interrupt) instruction. This causes the 8051 to continue program execution at the instruction following the one that set PCON.0 to initiate the suspend operation.

About the 'Resume' Interrupt

The 8051 enters the idle mode when PCON.0 is set to "1." Although the 8051 exits its idle state when *any* interrupt occurs, the EZ-USB FX logic supports only the RESUME Interrupt for the USB resume operation. This is because the USB core asserts this particular interrupt after restarting the 8051 clock.

14.4 Remote Wakeup

USBCS	**USB Control and Status**	**7FD6**

b7	b6	b5	b4	b3	b2	b1	b0
WAKESRC	-	-	-	DISCON	DISCOE	RENUM	**SIGRSUME**

Figure 14-5. USB Control and Status Register

Any device can ask to be enabled by the host to wakeup the host. (Additional information is provided in the section *"Remote Wakeup: The Big Picture"*, later in this chapter.

Two bits in the USBCS Register are used for remote wakeup, WAKESRC and SIGRSUME.

After exiting the idle state, the 8051 reads the WAKESRC Bit in the USBCS Register to discover how the wakeup was initiated. WAKESRC=1 indicates assertion of the WAKEUP# pin, and WAKESRC=0 indicates a resumption of USB bus activity. **The 8051 clears the WAKESRC Bit by writing a "1" to it.**

More about Wakeup

If your design does not assert remote wakeup (that is your device is not one of the pre-enabled devices that can wakeup the host), tie the WAKEUP# pin high.

Holding the WAKEUP# pin low inhibits the EZ-USB FX chip from suspending.

Suspend Flop. There is a flip flop in the FX that is set when the FX has seen the 3 ms of no SOFs on the USB. It's called Suspend Flop.

The WAKEUP# pin is tied to the clear of the Suspend Flop. Then the WAKEUP#pin is driven low:

The Suspend Flop clears, and the oscillator and Restart Delay start.

When the Suspend Flop is set (among other FX-local power saving steps), the D+ and D- drivers are powered down. Hence, until the Suspend Flop is cleared, the FX cannot signal the host.

When the FX wants to signal a Remote Wakeup Pulse to the host, the Suspend Flop must be cleared (or never allowed to set) to allow the D+ and D- drivers to be powered up **before** asserting the Remote Wakeup Pulse on the USB. In other words:

- *Holding* WAKEUP# low prevents the Suspend Flop from ever setting. (As it says above, “Holding the WAKEUP#pin low inhibits the FX chip from suspending.”)
- *Pulsing* WAKEUP# low clears the Suspend Flop.

This applies especially to self-powered devices where the designer elects to *not* save power during suspend, devices such as those powered from a wall transformer (as opposed to a battery.) You should tie the WAKEUP#pin low for these “power-rich” devices that *will* signal a remote wakeup.

In summary, for all devices that wish to assert a K-state wakeup pulse to the host (Remote Wakeup), the Suspend Flop must be cleared before asserting the K-state wakep pulse.

When a USB device is suspended, the upstream driver is tri-stated, and the bus pullup and pull-down resistors cause the bus to assume the “J,” or idle state. A suspended device signals a remote wakeup by asserting the “K” state for 1-15 ms. The 8051 controls this using the SIGR-SUME Bit in the USBCS Register.

If the 8051 finds WAKESRC=1 after exiting the idle mode, it must drive the “K” state for 1-15 ms to signal the USB remote wakeup. It does this by setting SIGRSUME=1, waiting 10-15 ms, and then setting SIGRSUME=0. The resume routine should also write a “1” to the WAKESRC Bit to clear it.

J and K States

The USB Specification uses differential data signals D+ and D-. Instead of defining a logical "1" and "0," it defines the "J" and "K" states. For a high speed device, the "J" state means (D+ > D-).

The USB Default device does not support remote wakeup. This fact is reported at enumeration time in byte 7 of the built-in Configuration Descriptor (Table 5-10).

Remote Wakeup: The Big Picture

Additional factors besides the EZ-USB FX suspend-resume mechanism described in this section determine whether remote wakeup is possible. These are:

- The device must report that it is capable of signaling a remote wakeup in the "bAttributes" field of its Configuration Descriptor. For an example of this description, see Table 5-10.
- The host must issue a "Set_Feature/Device" request with the feature selector field set to 0x01 to enable remote wakeup. For a detailed request, see Table 9-6.

Chapter 15. EZ-USB FX Registers

15.1 Introduction

This section describes the EZ-USB FX registers in the order they appear in the EZ-USB FX memory map. The registers are named according to the following conventions.

Most registers deal with endpoints. The general register format is **DDDnFFF**, where:

DDD is endpoint direction, IN or OUT with respect to the USB host.

n is the endpoint number, where:

- "07" refers to endpoints 0-7 as a group.
- 0-7 refers to each individual BULK/INTERRUPT/CONTROL endpoint.
- "ISO" indicates isochronous endpoints as a group.

FFF is the function, where:

- CS is a control and status register
- IRQ is an Interrupt Request Bit
- IE is an Interrupt Enable Bit
- BC, BCL, and BCH are byte count registers. BC is used for single byte counts, and BCL/H are used as the low and high bytes of 16-bit byte counts.
- DATA is a single-register access to a FIFO.
- BUF is the start address of a buffer.

15.1.1 Example Register Formats

- IN7BC is the Endpoint 7 IN byte count.

- OUT07IRQ is the register containing interrupt request bits for OUT endpoints 0-7.
- INISOVAL contains valid bits for the isochronous IN endpoints (EP8IN-EP15IN).

15.1.2 Other Conventions

USB	Indicates a global (not endpoint-specific) USB function.
ADDR	Is an address.
VAL	Means valid.
FRAME	Is a frame count.
PTR	Is an address pointer.

Register Name	Register Function	Address

b7	b6	b5	b4	b3	b2	b1	b0
bitname	**bitname**	**bitname**	**bitname**	**bitname**	**bitname**	**bitname**	**bitname**
R, W access	R, W access	R, W access	R, W access	R, W access	R, W access	R, W access	R, W access
Default val	Default val	Default val	Default val	Default val	Default val	Default val	Default val

Figure 15-1. Register Description Format

Figure 15-1. illustrates the register description format used in this chapter.

- The top line shows the register name, functional description, and address in the EZ-USB FX memory.
- The second line shows the bit position in the register.
- The third line shows the name of each bit in the register.
- The fourth line shows 8051 accessibility: R(ead), W(rite), or R/W.
- The fifth line shows the default value. These values apply after a Power-On-Reset (POR).

15.2 Slave FIFO Registers

15.2.1 FIFO A Read Data

AINDATA	FIFO A Read Data	7800

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

Figure 15-2. FIFO A Read Data

Each time the 8051 reads a byte from this register, the A-IN FIFO advances to the next byte in the FIFO and the AINBC (byte count) decrements. Reading this register when there is one byte remaining in the A-IN FIFO sets the A-IN FIFO Empty Flag (AINEF, in ABINCS.4), which causes an interrupt request on INT4 (Table 2). Reading this register when the A-IN FIFO is empty returns indeterminate data and has no effect on the FIFO flags byte counts. For more information, see Section 7.2.1. *"FIFO A Read Data"*.

15.2.2 A-IN FIFO Byte Count

AINBC	A-IN FIFO Byte Count	7801

b7	b6	b5	b4	b3	b2	b1	b0
0	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Figure 15-3. A-IN FIFO Byte Count

This count reflects the number of bytes remaining in the A-IN FIFO. Valid byte counts are 0-64. When non-zero, every byte written by outside logic increments this count, and every 8051 read of AINDATA decrements this count. If AINBC is zero, an 8051 read of AINDATA returns indeterminate data and results in the byte count in AINBC remaining at zero. Data bytes should never be written to the FIFO from outside logic when the AINFULL flag is HI. For more information, see Section 7.2.2. *"A-IN FIFO Byte Count"*.

15.2.3 A-IN FIFO Programmable Flag

AINPF	**A-IN FIFO Programmable Flag**	**7802**

b7	b6	b5	b4	b3	b2	b1	b0
LTGT	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-4. A-IN FIFO Programmable Flag

This register controls the sense and value for the internal A-IN FIFO programmable flag. This flag is testable by the 8051. For more information including a list of bit definitions for this register, see Section 7.2.3. *"A-IN FIFO Programmable Flag"*.

15.2.4 A-IN FIFO Pin Programmable Flag

AINPFPIN	**A-IN FIFO Pin Programmable Flag**	**7803**

b7	b6	b5	b4	b3	b2	b1	b0
LTGT	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-5. A-IN FIFO Pin Programmable Flag

This register controls the sense and value for the A-IN FIFO Programmable Flag that appears on the AINFLAG *pin*. This pin is used by external logic to regulate external writes to the A-IN FIFO. The AINPFPIN Register is programmed with the same data format as the previous register, AINPF. The only operational difference is that the flag drives a hardware pin rather than existing as an internal register bit. For more information, see Section 7.2.3.3. *"A-IN FIFO Pin Programmable Flag"*.

15.2.5 B-IN FIFO Read Data

BINDATA	B-IN FIFO Read Data	7805

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

Figure 15-6. B-IN FIFO Read Data

Each time the 8051 reads a byte from this register, the B-IN FIFO advances to the next byte in the FIFO and the BINBC (byte count) decrements. Reading this register when there is one byte remaining in the FIFO sets the B-IN FIFO Empty Flag (BINEF, in ABINCS.1), which causes an INT4 Request. Reading this register when the B-IN FIFO is empty returns indeterminate data and has no effect on the FIFO flags or byte count. For more information, see Section 7.2.4. *"B-IN FIFO Read Data"*.

15.2.6 B-IN FIFO Byte Count

BINBC	B-IN FIFO Byte Count	7806

b7	b6	b5	b4	b3	b2	b1	b0
0	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Figure 15-7. B-IN FIFO Byte Count

This count reflects the number of bytes remaining in the B-IN FIFO. Valid byte counts are 0-64. When non-zero, every byte written by outside logic increments this count, and every 8051 read of BINDATA decrements this count. If BINBC is zero, an 8051 read of BINDATA returns indeterminate data and results in the byte count in BINBC remaining at zero. Data bytes should never be written to the FIFO from outside logic when the BINFULL flag is HI. For more information, see Section 7.2.5. *"B-IN FIFO Byte Count."*

15.2.7 B-IN FIFO Programmable Flag

BINPF **B-IN FIFO Programmable Flag** **7807**

b7	b6	b5	b4	b3	b2	b1	b0
LTGT	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-8. B-IN FIFO Programmable Flag

This register controls the sense and value for the internal B-IN FIFO programmable flag. For more information including a list of bit descriptions, see Section 7.2.6. *"B-IN FIFO Programmable Flag."*

15.2.8 B-IN FIFO Pin Programmable Flag

BINPFPIN **B-IN FIFO Pin Programmable Flag** **7808**

b7	b6	b5	b4	b3	b2	b1	b0
LTGT	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-9. B-IN FIFO Pin Programmable Flag

This register controls the sense and value for the B-IN FIFO Programmable Flag that appears on the BINFLAG *pin*. This pin is used by external logic to regulate external writes to the B-IN FIFO. The BINPFPIN Register is programmed with the same data format as the previous register, BINPF. The only operational difference is that the flag drives a hardware pin rather than existing as an internal register bit. For more information see Section 7.2.7. *"B-IN FIFO Pin Programmable Flag."*

15.2.9 Input FIFOs A/B Toggle CTL and Flags

ABINTCS **Input FIFOs A/B Toggle CTL and Flags** **780A**

b7	b6	b5	b4	b3	b2	b1	b0
INTOG	**INSEL**	**AINPF**	**AINEF**	**AINFF**	**BINPF**	**BINEF**	**BINFF**
R/W	R/W	R	R	R	R	R	R
0	1	0	1	0	0	1	0

Figure 15-10. Input FIFOs A/B Toggle CTL and Flags

For information about this register, including a list of bit descriptions, see Section 7.2.8. *"Input FIFOs A/B Toggle CTL and Flags."*

15.2.10 Input FIFOs A/B Interrupt Enables

ABINIE **Input FIFOs A/B Interrupt Enables** **780B**

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**AINPFIE**	**AINEFIE**	**AINFFIE**	**BINPFIE**	**BINEFIE**	**BINFFIE**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-11. Input FIFOs A/B Interrupt Enables

For information about this register, including a list of bit descriptions, see Section 7.2.9. *"Input FIFOs A/B Interrupt Enables."*

15.2.11 Input FIFOs A/B Interrupt Requests

ABINIRQ **Input FIFOs A/B Interrupt Enables** **780C**

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**AINPFIR**	**AINEFIR**	**AINFFIR**	**BINPFIR**	**BINEFIR**	**BINFFIR**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-12. Input FIFOs A/B Interrupt Requests

For information about the ABINIRQ Register, including a list of bit descriptions, see Section 7.2.10. *"Input FIFOs A/B Interrupt Requests."*

15.2.12 FIFO A Write Data

AOUTDATA			**FIFO A Write Data**				**780E**
b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

Figure 15-13. FIFO A Write Data

A-OUT FIFO Write Data. Each time the 8051/DMA writes a byte to this register, the A-OUT FIFO advances to the next open position in the FIFO and the AOUTBC (byte count) increments. Writing this register when there are 63 bytes remaining in the A-OUT FIFO sets the A-FIFO Full Flag (AOUTFF, in ABOUTCS.3), which causes an INT4 Request. Writing this register when the A-OUT FIFO is full (64 bytes) does not update the FIFO or byte count, and has no effect on the FIFO flags or byte count. For more information, see Section 7.2.11. *"FIFO A Write Data."*

15.2.13 A-OUT FIFO Byte Count

AOUTBC			**A-OUT FIFO Byte Count**				**780F**
b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Figure 15-14. Input FIFOs A/B Interrupt Requests

This count reflects the number of bytes remaining in the A-OUT FIFO. Valid byte counts are 0-64. When non-zero, every byte read by outside logic decrements this count, and every 8051 write of AOUTDATA increments this count. If AOUTBC is zero, reading a data byte by outside logic returns indeterminate data and results in the byte count in AOUTBC remaining at zero. For more information, see Section 7.2.11.1. *"A-OUT FIFO Byte Count."*

15.2.14 A-OUT FIFO Programmable Flag

AOUTPF	A-OUT FIFO Programmable Flag	7810

b7	b6	b5	b4	b3	b2	b1	b0
LTGT	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	0	1	0	0	0	0	0

Figure 15-15. Input FIFOs A/B Interrupt Requests

This register controls the sense and value for the internal A-OUT FIFO Programmable Flag. The internal flag may be tested by the 8051, and/or enabled to cause an INT4 Interrupt Request. The 8051 tests the internal FIFO programmable flag by reading the AOUTPF Bit in ABOUTCS.5 (register at 0x7818). For more information including a list of bit descriptions, see Section 7.2.12. *"A-OUT FIFO Programmable Flag."*

15.2.15 A-OUT FIFO Pin Programmable Flag

AOUTPFPIN	A-OUT FIFO Pin Programmable Flag	7811

b7	b6	b5	b4	b3	b2	b1	b0
LTGT	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	0	0	0	0	0	0

Figure 15-16. A-OUT FIFO Pin Programmable Flag

This register controls the sense and value for the A-OUT FIFO Programmable Flag that appears on the AOUTFLAG *pin*. This pin is used by external logic to regulate external reads from the A-OUT FIFO. The AOUTPFPIN Register is programmed with the same data format as the previous register, AOUTPF. The only operational difference is that the flag drives a hardware pin rather than existing as an internal register bit. For more information, see Section 7.2.13. *"A-OUT FIFO Pin Programmable Flag."*

15.2.16 B-OUT FIFO Write Data

BOUTDATA	B-OUT FIFO Write Data	7813

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

Figure 15-17. B-OUT FIFO Write Data

Each time the 8051/DMA writes a byte to this register, the B-OUT FIFO advances to the next open position in the FIFO and the BOUTBC (Byte count) increments. Writing this register when there are 63 bytes remaining in the B-OUT FIFO sets the B-FIFO Full Flag (BOUTFF, in ABOUTCS.0), which causes an INT4 Interrupt Request. Writing this register when the B-OUT FIFO is full (64 bytes) does not update the FIFO or byte count, and has no effect on the FIFO flags or byte count. For more information, see Section 7.2.14. *"B-OUT FIFO Write Data."*

15.2.17 B-OUT FIFO Byte Count

BOUTBC	B-OUT FIFO Byte Count	7814

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Figure 15-18. B-OUT FIFO Byte Count

This count reflects the number of bytes remaining in the B-OUT FIFO. Valid byte counts are 0-64. When non-zero, every byte read by outside logic decrements this count, and every 8051 write of BOUTDATA increments this count. If BOUTBC is zero, reading a data byte by outside logic returns indeterminate data and results in the byte count in BOUTBC remaining at zero. For more information, see Section 7.2.15. *"B-OUT FIFO Byte Count."*

15.2.18 B-OUT FIFO Programmable Flag

BOUTPF	B-OUT FIFO Programmable Flag	7815

b7	b6	b5	b4	b3	b2	b1	b0
LTGT	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	0	1	0	0	0	0	0

Figure 15-19. B-OUT FIFO Programmable Flag

This register controls the sense and value for the internal B-OUT FIFO Programmable Flag. The internal flag may be tested by the 8051, and/or enabled to cause an INT4 Interrupt Request. For more information including a list of bit descriptions, see Section 7.2.16. *"B-OUT FIFO Programmable Flag."*

15.2.19 B-OUT FIFO Pin Programmable Flag

BOUTPFPIN	B-OUT FIFO Pin Programmable Flag	7816

b7	b6	b5	b4	b3	b2	b1	b0
LTGT	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-20. B-OUTFIFO Pin Programmable Flag

This register controls the sense and value for the B-OUT FIFO Programmable Flag that appears on the BOUTFLAG *pin*. This pin is used by external logic to regulate external reads from the B-OUT FIFO. The BOUTPFPIN Register is programmed with the same data format as the previous register, BOUTPF. The only operational difference is that the flag drives a hardware pin rather than existing as an internal register bit. For more information including a list of bit descriptions, see Section 7.2.17. *"B-OUT FIFO Pin Programmable Flag."*

15.2.20 Output FIFOs A/B Toggle CTL and Flags

ABOUTCS	Output FIFOs A/B Toggle CTL and Flags	7818

b7	b6	b5	b4	b3	b2	b1	b0
OUTINTOG	**OUTSEL**	**AOUTPF**	**AOUTEF**	**AOUTFF**	**BOUTPF**	**BOUTEF**	**BOUTFF**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	1	0	1	0	0	1	0

Figure 15-21. Output FIFOs A/B Toggle CTL and Flags

For information about this register, including a list of bit descriptions, see Section 7.2.18. *"Output FIFOs A/B Toggle CTL and Flags."*

15.2.21 Output FIFOs A/B Interrupt Enables

ABOUTIE	Output FIFOs A/B Interrupt Enables	7819

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**AOUTPFIE**	**AOUTEFIE**	**AOUTFFIE**	**BOUTPFIE**	**BOUTEFIE**	**BOUTFFIE**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-22. Output FIFOs A/B Interrupt Enables

For information about this register, including a list of bit descriptions, see Section 7.2.19. *"Output FIFOs A/B Interrupt Enables."*

15.2.22 Output FIFOs A/B Interrupt Requests

ABOUTIRQ	Output FIFOs A/B Interrupt Requests	781A

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**AOUTPFIR**	**AOUTEFIR**	**AOUTFFIR**	**BOUTPFIR**	**BOUTEFIR**	**BOUTFFIR**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-23. Output FIFOs A/B Interrupt Requests

For information about this register, including a list of bit descriptions, see Section 7.2.20. *"Output FIFOs A/B Interrupt Requests."*

15.2.23 FIFO A/B Setup

ABSETUP			**FIFO A/B Setup**				**781C**
b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**ASYNC**	**DBLIN**	**0**	**OUTDLY**	**0**	**DBLOUT**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-24. FIFO A/B Setup

For information about this register, including a list of bit descriptions, see Section 7.2.21. *"FIFO A/B Setup."*

15.2.24 FIFO A/B Control Signal Polarities

ABPOLAR			**FIFO A/B Control Signal Polarities**				**781D**
b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**BOE**	**AOE**	**SLRD**	**SLWR**	**ASEL**	**BSEL**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-25. FIFO A/B Control Signal Polarities

These bits define the pin polarities for the indicated signals. The 8051 sets a bit LOW for active low, and HI for active high. For more information including a list of bit descriptions, see Section 7.2.22. *"FIFO A/B Control Signal Polarities."*

15.2.25 FIFO Flag Reset

ABFLUSH	Reset All FIFO Flags	781E

b7	b6	b5	b4	b3	b2	b1	b0
x	**x**	**x**	**x**	**x**	**x**	**x**	**x**
W	W	W	W	W	W	W	W
x	x	x	x	x	x	x	x

Figure 15-26. FIFO Flag Reset

The 8051 writes any value to this register to reset the FIFO byte counts to zero, effectively *flushing* the FIFOs. Consequently, the byte counts are set to zero, the empty flags are set, and the full flags are cleared. Reading this register returns indeterminate data. For more information including a list of bit descriptions, see Section 7.2.23. *"FIFO Flag Reset."*

15.3 Waveform Selector

WFSELECT	Waveform Selector	7824

b7	b6	b5	b4	b3	b2	b1	b0
SINGLEWR0-3		**SINGLERD0-3**		**FIFOWR0-3**		**FIFORD0-3**	
W	W	R	R	W	W	R	R
x	x	x	x	x	x	x	x

Figure 15-27. Waveform Selector

For detailed information, see *Chapter 8. "General Programmable Interface (GPIF)"*.

15.4 GPIF Done, GPIF IDLE Drive Mode

IDLECS **GPIF Done, GPIF IDLE Drive Mode** **7825**

b7	b6	b5	b4	b3	b2	b1	b0
DONE	**0**	**0**	**0**	**0**	**0**	**0**	**IDLEDRV**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-28. GPIF Done, GPIF IDLE Drive Mode

For detailed information, see *Chapter 8. "General Programmable Interface (GPIF)"*.

15.5 Inactive Bus, CTL States

IDLECTLOUT **Inactive Bus, CTL States** **7826**

b7	b6	b5	b4	b3	b2	b1	b0
IOE3	**IOE2**	**IOE1/CTL5**	**IOE0/CTL4**	**CTL3**	**CTL2**	**CTL1**	**CTL0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-29. Inactive Bus, CTL States

CTLOUTCFG **CTLOUT Pin Drive** **7827**

b7	b6	b5	b4	b3	b2	b1	b0
TRICTL	**0**	**CTL5**	**CTL4**	**CTL3**	**CTL2**	**CTL1**	**CTL0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-30. CTLOUT Pin Drive

15.6 GPIF Address LSB

GPIFADRL	GPIF Address Low	782A

b7	b6	b5	b4	b3	b2	b1	b0
x	**x**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-31. GPIF Address Low

15.7 FIFO A IN Transaction Count

AINTC	FIFO A IN Transaction Count	782C

b7	b6	b5	b4	b3	b2	b1	b0
FITC	**AINTC6**	**AINTC5**	**AINTC4**	**AINTC3**	**AINTC2**	**AINTC1**	**AINTC0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-32. FIFO A IN Transaction Count

For detailed information, see *Chapter 8. "General Programmable Interface (GPIF)"*.

15.8 FIFO A OUT Transaction Count

AOUTTC	FIFO A OUT Transaction Count	782D

b7	b6	b5	b4	b3	b2	b1	b0
FITC	**AOUTTC6**	**AOUTTC5**	**AOUTTC4**	**AOUTTC3**	**AOUTTC2**	**AOUTTC1**	**AOUTTC0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-33. FIFO A OUT Transaction Count

See *Chapter 8. "General Programmable Interface (GPIF)"* for detailed information.

15.9 FIFO A Transaction Trigger

ATRIG	FIFO A Transaction Trigger	782E

b7	b6	b5	b4	b3	b2	b1	b0
x	**x**	**x**	**x**	**x**	**x**	**x**	**x**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-34. FIFO A Transaction Trigger

See *Chapter 8. "General Programmable Interface (GPIF)"* for detailed information.

15.10 FIFO B IN Transaction Count

BINTC	**FIFO B IN Transaction Count**	**7830**

b7	b6	b5	b4	b3	b2	b1	b0
BINTC7	**BINTC6**	**BINTC5**	**BINTC4**	**BINTC3**	**BINTC2**	**BINTC1**	**BINTC0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-35. FIFO B IN Transaction Count

For detailed information, see *Chapter 8. "General Programmable Interface (GPIF)"*.

15.11 FIFO B OUT Transaction Count

BOUTTC	**FIFO B OUT Transaction Count**	**7831**

b7	b6	b5	b4	b3	b2	b1	b0
BOUTTC7	**BOUTTC6**	**BOUTTC5**	**BOUTTC4**	**BOUTTC3**	**BOUTTC2**	**BOUTTC1**	**BOUTTC0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-36. FIFO B OUT Transaction Count

For detailed information, see *Chapter 8. "General Programmable Interface (GPIF)"*.

15.12 FIFO B Transaction Trigger

BTRIG	FIFO B Transaction Trigger	7832

b7	b6	b5	b4	b3	b2	b1	b0
x	**x**	**x**	**x**	**x**	**x**	**x**	**x**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-37. FIFO B Transaction

For detailed information, see *Chapter 8. "General Programmable Interface (GPIF)"*.

15.13 GPIF Data H (16-bit mode only)

SGLDATH	GPIF Data H (16-bit mode only)	7834

b7	b6	b5	b4	b3	b2	b1	b0
D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-38. GPIF Data H (16-bit mode only)

15.14 Read or Write GPIF Data L and Trigger Read Transaction

SGLDATLTRIG	R/W GPIF DataL/Trig Rd Transaction	7835

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-39. Read or Write GPIF Data L and Trigger Read Transaction

15.15 Read GPIF Data L, No Read Transaction Trigger

SGLDATLNTRIG **Rd GPIF Data L/No Trig Rd Transaction** **7836**

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

Figure 15-40. Read GPIF Data L, No Read Transaction Trigger

15.16 Internal READY, Sync/Async, READY Pin States

READY **Internal Rdy, Sync/Async, Rdy Pin States** **7838**

b7	b6	b5	b4	b3	b2	b1	b0
INTRDY	**SAS**	**RDY5**	**RDY4**	**RDY3**	**RDY2**	**RDY1**	**RDY0**
R/W	R/W	R	R	R	R	R	R
x	x	x	x	x	x	x	x

Figure 15-41. Internal READY, Sync/Async, READY Pin States

15.17 Abort GPIF Cycles

ABORT **Abort GPIF Cycles** **7839**

b7	b6	b5	b4	b3	b2	b1	b0
x	**x**	**x**	**x**	**x**	**x**	**x**	**x**
W	W	W	W	W	W	W	W
x	x	x	x	x	x	x	x

Figure 15-42. Abort GPIF Cycles

15.18 General Purpose I/F Interrupt Enable

GENIE	General Purpose I/F Interrupt Enable	783B

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**DMADONE**	**GPIFWF**	**GPIFDONE**
W	W	W	W	W	W	W	W
x	x	x	x	x	x	x	x

Figure 15-43. Generic Interrupt Enable

15.19 Generic Interrupt Request

GENIRQ	Generic Interrupt Request	783C

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**DMADONE**	**GPIFWF**	**GPIFDONE**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-44. Generic Interrupt Request

15.20 Input/Output Port Registers D and E

For more information, see Section 4.3. *"Input/Output Port Registers"*.

15.20.1 Port D Outputs

OUTD			**Port D Outputs**				**7841**
b7	b6	b5	b4	b3	b2	b1	b0
OUTD7	**OUTD6**	**OUTD5**	**OUTD4**	**OUTD3**	**OUTD2**	**OUTD1**	**OUTD0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-45. Port D Outputs

15.20.2 Input Port D Pins

PINSD			**Port D Pins**				**7842**
b7	b6	b5	b4	b3	b2	b1	b0
PIND7	**PIND6**	**PIND5**	**PIND4**	**PIND3**	**PIND2**	**PIND1**	**PIND0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

Figure 15-46. Input Port D Pins

15.20.3 Port D Output Enable

OED			**Port D Output Enable**				**7843**
b7	b6	b5	b4	b3	b2	b1	b0
OED7	**OED6**	**OED5**	**OED4**	**OED3**	**OED2**	**OED1**	**OED0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-47. Port D Output Enable Register

15.20.4 Port E Outputs

OUTE	Port E Outputs	7845

b7	b6	b5	b4	b3	b2	b1	b0
OUTE7	**OUTE6**	**OUTE5**	**OUTE4**	**OUTE3**	**OUTE2**	**OUTE1**	**OUTE0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-48. Port E Outputs

15.20.5 Input Port E Pins

PINSE	Port E Pins	7846

b7	b6	b5	b4	b3	b2	b1	b0
PINE7	**PINE6**	**PINE5**	**PINE4**	**PINE3**	**PINE2**	**PINE1**	**PINE0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

Figure 15-49. Input Port E Pins

15.20.6 Port E Output Enable

OEE	Port E Output Enable	7847

b7	b6	b5	b4	b3	b2	b1	b0
OEE7	**OEE6**	**OEE5**	**OEE4**	**OEE3**	**OEE2**	**OEE1**	**OEE0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-50. Port E Output Enable Register

15.21 PORTSETUP

PORTSETUP **Timer0 Clock Source, Port to SFR Mapping** **7849**

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**0**	**TOCLK**	**SFRPORT**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-51. PORTSETUP

15.22 Interface Configuration

IFCONFIG **Interface Configuration** **784A**

b7	b6	b5	b4	b3	b2	b1	b0
52ONE	**0**	**0**	**0**	**GSTATE**	**BUS16**	**IF1**	**IF0**
R/W	R	R	R	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-52. Interface Configuration

Bit 7: **52ONE** *Set to "1" for the 52-pin package*

This bit must be set to "1" for the 52-pin versions of EZ-USB FX. This ensures that certain signals that are driven properly for EZ-USB FX low power operation.

Bit 6-4: **Reserved** *Reserved bits, read as 0*

Bit 3: **GSTATE** *Output GSTATE*

When GSTATE=1, three bits in Port A take on the signals shown in Table 15-1. The GSTATE bits, which indicate GPIF states, are used for diagnostic purposes.

Table 15-1. Port A Alternate Functions When GSTATE=1.

IO Pin	Alternate Function
PA0	GSTATE[0]
PA1	GSTATE[1]
PA2	GSTATE[2]

Bit 2: **BUS16** *8- or 16-Bit Slave FIFO Operation*

This bit selects 8-bit (BUS16=0) or 16-bit (BUS16=1) operation for slave FIFOs A and B. See *Chapter 7. "EZ-USB FX Slave FIFOs"* for full details.

Bit 1-0: **Interface Select** *Reconfigure I/O ports*

These bits, along with the BUS16 bit, select different groups of signals for various EZ-USB FX pins. Table 15-2 shows the selections.

Table 15-2. Pin Configurations Based on IFCONFIG[1..0]

IFCONFIG[1..0]					
00	**01**	**10**		**11**	
		BUS16=1	**BUS16=0**	**BUS16=1**	**BUS16-=0**
PE0	PE0	adr0	adr0	BOUTFLAG	BOUTFLAG
PE1	PE1	adr1	adr1	AINFULL	AINFULL
PE2	PE2	adr2	adr2	BINFULL	BINFULL
PE3	PE3	adr3	adr3	AOUTEMTY	AOUTEMTY
PE4	PE4	adr4	adr4	BOUTEMTY	BOUTEMTY
PE5	PE5	CTL3	CTL3	PE5	PE5
PE6	PE6	CTL4	CTL4	PE6	PE6
PE7	PE7	CTL5	CTL5	PE7	PE7
NC	NC	CTL0	CTL0	AINFLAG	AINFLAG
NC	NC	CTL1	CTL1	BINFLAG	BINFLAG
NC	NC	CTL2	CTL2	AOUTFLAG	AOUTFLAG
Strap	Strap	RDY0	RDY0	ASEL	ASEL
Strap	Strap	RDY1	RDY1	BSEL	BSEL
Strap	Strap	RDY2	RDY2	AOE	AOE
Strap	Strap	RDY3	RDY3	BOE	BOE
Strap	Strap	RDY4	RDY4	SLWR	SLWR
Strap	Strap	RDY5	RDY5	SLRD	SLRD
Strap	Strap	adr5	adr5	X	X
Strap	Strap	XCLK	XCLK	XCLK	XCLK
PORTB	**D[7..0]**	**GDA[7..0]**	**GDA7..0]**	**AFI[7..0]**	**AFI[7..0]**
PORTD	**PORTD**	**GDB[7..0]**	**PORTD**	**BFI[7..0]**	**PORTD**

NC -Package pin must be left unconnected.
Strap -Package pin must be either pulled-up to V_{DD} or pulled-down to GND.

15.23 PORTA and PORTC Alternate Configurations

15.23.1 Port A Alternate Configuration #2

PORTACF2	PORTA Alternate Configuration #2	784B

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**SLRD**	**SLWR**	**0**	**0**	**0**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-53. Port A Alternate Configuration #2

Bit 5: **SLRD** *Select SLRD/RDY5 signal on PA5 pin*

This bit, in conjunction with the PORTACFG.5 Bit and the IFCONFIG[1..0] bits, determines the function of PA5, as shown in Table 15-3.

Table 15-3. Port A Bit 5

PORTA Bit 5			
PORT-ACFG.5=0	PORTACFG.5=1		
	PORTACF2.5=0	PORTACF2.5=1	
		IFCONFIG[1..0]=10	IFCONFIG[1..0]=11
Port pin PA5	**FRD#**	**RDY5**	**SLRD**

Bit 4: **SLWR** *Select SLWR/RDY4 signal on PA4 pin*

This bit, in conjunction with the PORTACFG.4 Bit and the IFCONFIG[1..0] bits, determines the function of PA4, as shown in Table 15-4.

Table 15-4. Port A Bit 4

PORTA Bit 4			
PORT-ACFG.4=0	PORTACFG.4=1		
	PORTACF2.4=0	PORTACF2.4=1	
		IFCONFIG[1..0]=10	IFCONFIG[1..0]=11
Port pin PA4	**FWR#**	**RDY4**	**SLWR**

15.23.2 Port C Alternate Configuration #2

PORTCCF2	PORTC Alternate Configuration #2	784C

b7	b6	b5	b4	b3	b2	b1	b0
CTL5	**CTL4**	**CTL3**	**CTL1**	**RDY3**	**0**	**RDY1**	**RDY0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-54. Port C Alternate Configuration #2

Bit 7: **CTL5** *Select CTL5 on PC7 pin*

This bit, in conjunction with the PORTCCFG.7 Bit, determines the function of PC7, as shown in Table 15-5.

Table 15-5. Port C Bit 7

PORTC Bit 7			
PORTCCFG.7=0	PORTCCFG.7=1		
	PORTCCF2.7=0	PORTCCF2.7=1	
		IFCONFIG[1..0]=10	00, 01, 11 not valid
Port pin PC7	**RD#**	**CTL5**	**X**

Bit 6: **CTL4** *Select CTL4 on PC6 pin*

This bit, in conjunction with the PORTCCFG.6 Bit, determines the function of PC6, as shown in Table 15-6.

Table 15-6. Port C Bit 6

PORTC Bit 6			
PORTCCFG.6=0	PORTCCFG.6=1		
	PORTCCF2.6=0	PORTCCF2.6=1	
		IFCONFIG[1..0]=10	00, 01, 11 not valid
Port pin PC6	**WR#**	**CTL4**	**X**

Bit 5: CTL3 *Select CTL3 on PC5 pin*

This bit, in conjunction with the PORTCCFG.5 Bit, determines the function of PC5, as shown in Table 15-7.

Table 15-7. Port C Bit 5

PORTC Bit 5			
PORTCCFG.5=0	PORTCCFG.5=1		
	PORTCCF2.5=0	PORTCCF2.5=1	
		IFCONFIG[1..0]=10	00, 01, 11 not valid
Port pin PC5	**T1**	**CTL3**	**X**

Bit 4: CTL1 *Select CTL1 on PC4 pin*

This bit, in conjunction with the PORTCCFG.4 Bit, determines the function of PC4, as shown in Table 15-8.

Table 15-8. Port C Bit 4

PORTC Bit 4			
PORTCCFG.4=0	PORTCCFG.4=1		
	PORTCCF2.4=0	PORTCCF2.4=1	
		IFCONFIG[1..0]=10	00, 01, 11 not valid
Port pin PC4	**T0**	**CTL1**	**X**

Bit 3: RDY3 *Select RDY3 on PC3 pin*

This bit, in conjunction with the PORTCCFG.3 Bit, determines the function of PC3, as shown in Table 15-9.

Table 15-9. Port C Bit 3

PORTC Bit 3			
PORTCCFG.3=0	PORTCCFG.3=1		
	PORTCCF2.3=0	PORTCCF2.3=1	
		IFCONFIG[1..0]=10	00, 01, 11 not valid
Port pin PC3	**INT1**	**RDY3**	**X**

Bit 2: **Reserved** *Reads as 0*

Bit 1: **RDY1** *Select RDY1 on PC1 pin*

This bit, in conjunction with the PORTCCFG.1 Bit, determines the function of PC1, as shown in Table 15-10.

Table 15-10. Port C Bit 1

PORTC Bit 1			
PORTCCFG.1=0	PORTCCFG.1=1		
	PORTCCF2.1=0	PORTCCF2.1=1	
		IFCONFIG[1..0]=10	00, 01, 11 not valid
Port pin PC1	**TxD0**	**RDY1**	**X**

Bit 0: **CTL5** *Select CTL5 on PC0 pin*

This bit, in conjunction with the PORTCCFG.0 Bit, determines the function of PC0, as shown in Table 15-11.

Table 15-11. Port C Bit 0

PORTC Bit 0			
PORTCCFG.0=0	PORTCCFG.0=1		
	PORTCCF2.0=0	PORTCCF2.0=1	
		IFCONFIG[1..0]=10	00, 01, 11 not valid
Port pin PC0	**RxD0**	**RDY0**	**X**

15.24 DMA Registers

For more information on these DMA registers, see *Section 11.2. "DMA Register Descriptions"*.

15.24.1 Source, Destination, Transfer Length Address Registers

DMASRCH **DMA Source Address (H)** **784F**

b7	b6	b5	b4	b3	b2	b1	b0
A15	**A14**	**A13**	**A12**	**A11**	**A10**	**A9**	**A8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-55. Upper Byte of the DMA Source Address

DMASRCL **DMA Source Address (L)** **7850**

b7	b6	b5	b4	b3	b2	b1	b0
A7	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-56. Lower Byte of the DMA Source Address

DMADESTH **DMA Destination Address (H)** **7851**

b7	b6	b5	b4	b3	b2	b1	b0
A15	**A14**	**A13**	**A12**	**A11**	**A10**	**A9**	**A8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-57. Upper Byte of the DMA Destination Address

DMADESTL	DMA Destination Address (L)	7852

b7	b6	b5	b4	b3	b2	b1	b0
A7	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-58. Lower Byte of the DMA Destination Address

DMALEN	DMA Transfer Length	7854

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-59. DMA Transfer Length (0=256 Bytes, 1=1 Byte, ... 255=255 Bytes)

15.24.2 DMA Start and Status Register

For further information on DMA Registers, see *Section 11.2. "DMA Register Descriptions".*

DMAGO	DMA Start and Status	7855

b7	b6	b5	b4	b3	b2	b1	b0
DONE	**0**	**0**	**0**	**0**	**0**	**0**	**0**
R/W	x	x	x	x	x	x	x
0	x	x	x	x	x	x	x

Figure 15-60. DMA Start and Status Register

15.24.3 DMA Synchronous Burst Enables Register

DMABURST	Synchronous Burst Enables	7857

b7	b6	b5	b4	b3	b2	b1	b0
x	**x**	**x**	**DSTR2**	**DSTR1**	**DSTR0**	**BR**	**BW**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	1	0	0

Figure 15-61. Synchronous Burst Enables

15.24.4 Select 8051 A/D busses as External FIFO

DMAEXTFIFO	Use A/D Buses as External FIFO	7858

b7	b6	b5	b4	b3	b2	b1	b0
n/a	**n/a**	**n/a**	**n/a**	**n/a**	**n/a**	**n/a**	**n/a**
n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a

Figure 15-62. Dummy Register

15.25 Slave FIFO Interrupt (INT4)

The EZ-USB FX slave FIFOs contain various flags to alert the 8051 when a FIFO needs attention. These flags are encoded into the INT4 Autovector, which the 8051 can read in the INT4IVEC Register. The encoded values for each INT4 source are shown in Table 12-4. For more information including bit descriptions, see Section 12.15. *"Slave FIFO Interrupt (INT4)"*.

15.25.1 Interrupt 4 Autovector

INT4IVEC **Interrupt 4 Autovector** **785D**

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**I4V3**	**I4V2**	**I4V1**	**I4V0**	**0**	**0**
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Figure 15-63. Interrupt 4 Autovector

15.25.2 Interrupt 4 Autovector

INT4SETUP **Interrupt 4 Setup** **785E**

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**INT4FC**	**INTRNL**	**AV4EN**
R	R	R	R	R	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-64. Interrupt 4 Setup

15.26 Waveform Descriptors

WFDESC	Waveform Descriptors	7900

b7	b6	b5	b4	b3	b2	b1	b0
x	**x**	**x**	**x**	**x**	**x**	**x**	**x**
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Figure 15-65. Waveform Descriptors

For detailed information, see *Section 8.1. "What is GPIF?"*.

15.27 Bulk Data Buffers

INnBUF,OUTnBUF	Endpoint 0-7 IN/OUT Data Buffers	7B40-7F3F*

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

* See Table 15-12 for individual endpoint buffer addresses.

Figure 15-66. Bulk Data Buffers

Table 15-12. Bulk Endpoint Buffer Memory Addresses

Address	Address	Name	Size
1F00-1F3F	7F00-7F3F	IN0BUF	64
1EC0-1EFF	7EC0-7EFF	OUT0BUF	64
1E80-1EBF	7E80-7EBF	IN1BUF	64
1E40-1E7F	7E40-7E7F	OUT1BUF	64
1E00-1E3F	7E00-7E3F	IN2BUF	64
1DC0-1DFF	7DC0-7DFF	OUT2BUF	64
1D80-1DBF	7D80-7DBF	IN3BUF	64
1D40-1D7F	7D40-7D7F	OUT3BUF	64
1D00-1D3F	7D00-7D3F	IN4BUF	64
1CC0-1CFF	7CC0-7CFF	OUT4BUF	64
1C80-1CBF	7C80-7CBF	IN5BUF	64
1C40-1C7F	7C40-7C7F	OUT5BUF	64
1C00-1C3F	7C00-7C3F	IN6BUF	64
1BC0-1BFF	7BC0-7BFF	OUT6BUF	64
1B80-1BBF	7B80-7BBF	IN7BUF	64
1B40-1B7F	7B40-7B7F	OUT7BUF	64

Sixteen 64-byte bulk data buffers appear at 0x1B40 **and** 0x7B40 in the 8K version of EZ-USB FX. The endpoints are ordered to permit the reuse of the buffer space as contiguous RAM when the higher numbered endpoints are not used. These registers default to unknown states.

15.28 Isochronous Data FIFOs

OUTnDATA	EP8OUT-EP15OUT FIFO Registers	7F60-7F67*

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

INnDATA	EP8IN-EP15IN FIFO Registers	7F68-7F6F*

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
W	W	W	W	W	W	W	W
x	x	x	x	x	x	x	x

* See Table 15-13 for individual endpoint buffer addresses.

Figure 15-67. Isochronous Data FIFOs

Table 15-13. Isochronous Endpoint FIFO Register Addresses

Address	Isochronous Data	Name
7F60	Endpoint 8 OUT Data	OUT8DATA
7F61	Endpoint 9 OUT Data	OUT9DATA
7F62	Endpoint 10 OUT Data	OUT10DATA
7F63	Endpoint 11 OUT Data	OUT11DATA
7F64	Endpoint 12 OUT Data	OUT12DATA
7F65	Endpoint 13 OUT Data	OUT13DATA
7F66	Endpoint 14 OUT Data	OUT14DATA
7F67	Endpoint 15 OUT Data	OUT15DATA
7F68	Endpoint 8 IN Data	IN8DATA
7F69	Endpoint 9 IN Data	IN9DATA
7F6A	Endpoint 10 IN Data	IN10DATA
7F6B	Endpoint 11 IN Data	IN11DATA
7F6C	Endpoint 12 IN Data	IN12DATA
7F6D	Endpoint 13 IN Data	IN13DATA
7F6E	Endpoint 14 IN Data	IN14DATA
7F6F	Endpoint 15 IN Data	IN15DATA

Sixteen addressable data registers hold data from the eight isochronous IN endpoints and the eight isochronous OUT endpoints. Reading a data register reads a FIFO byte (USB OUT data); writing a Data Register loads a FIFO byte (USB IN data).

15.29 Isochronous Byte Counts

OUTnBCH **OUT(8-15) Byte Count High** **7F70-7F7F***

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**0**	**BC9**	**BC8**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

INnBCL **OUT(8-15) Byte Count Low** **7F70-7F7F***

b7	b6	b5	b4	b3	b2	b1	b0
BC7	**BC6**	**BC5**	**BC4**	**BC3**	**BC2**	**BC1**	**BC0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

* See Table 15-14 for individual endpoint buffer addresses.

Figure 15-68. Isochronous Byte Counts

Table 15-14. Isochronous Endpoint Byte Count Register Addresses

Address	Isochronous Data	Name
7F70	Endpoint 8 Byte Count High	OUT8BCH
7F71	Endpoint 8 Byte Count Low	OUT8BCL
7F72	Endpoint 9 Byte Count High	OUT9BCH
7F73	Endpoint 9 Byte Count Low	OUT9BCL
7F74	Endpoint 10 Byte Count High	OUT10BCH
7F75	Endpoint 10 Byte Count Low	OUT10BCL
7F76	Endpoint 11 Byte Count High	OUT11BCH
7F77	Endpoint 11 Byte Count Low	OUT11BCL
7F78	Endpoint 12 Byte Count High	OUT12BCH
7F79	Endpoint 12 Byte Count Low	OUT12BCL
7F7A	Endpoint 13 Byte Count High	OUT13BCH
7F7B	Endpoint 13 Byte Count Low	OUT13BCL
7F7C	Endpoint 14 Byte Count High	OUT14BCH
7F7D	Endpoint 14 Byte Count Low	OUT14BCL
7F7E	Endpoint 15 Byte Count High	OUT15BCH
7F7F	Endpoint 15 Byte Count Low	OUT15BCL

The USB core uses the byte count registers to report isochronous data payload sizes for OUT data transferred from the host to the USB core. Ten bits of byte count data allow payload size up to 1,023 bytes. A byte count of zero is valid, meaning that the host sent no isochronous data during the previous frame. The default values of these registers are unknown.

Byte counts are valid only for OUT endpoints. The byte counts indicate the number of bytes remaining in the endpoint's OUT FIFO. Every time the 8051 reads a byte from the ISODATA Register, the byte count decrements by one.

To read USB OUT data, the 8051 first reads byte count registers OUTnBCL and OUTnBCH to determine how many bytes to transfer out of the OUT FIFO. (The 8051 can also quickly test ISO output endpoints for zero byte counts using the ZBCOUT Register.) Then, the CPU reads that number of bytes from the ISODATA Register. Separate byte counts are maintained for each endpoint, so the CPU can read the FIFOs in a discontinuous manner. For example, if EP8 indicates a byte count of 100, and EP9 indicates a byte count of 50, the CPU could read 50 bytes from EP8, then read 10 bytes from EP9, and resume reading EP8.

There are no byte count registers for the IN endpoints. The USB core automatically tracks the number of bytes loaded by the 8051.

If the 8051 does not load an IN isochronous endpoint FIFO during a 1-ms frame, and the host requests data from that endpoint during the next frame (IN token), the USB core responds according to the setting of the ISOSEND0 Bit (USBPAIR.7). If ISOSEND0=1, the core returns a zero-length data packet in response to the host IN token. If ISOSEND=0, the core does not respond to the IN token.

It is the responsibility of the 8051 programmer to ensure that the number of bytes written to the IN FIFO does not exceed the maximum packet size as reported during enumeration.

15.30 CPU Registers

CPUCS **CPU Control and Status** **7F92**

b7	b6	b5	b4	b3	b2	b1	b0
RV3	**RV2**	**RV1**	**RV0**	**24/48**	**CLKINV**	**CLKOE**	**8051RES**
R	R	R	R	R	R	R/W	R
RV3	RV2	RV1	RV0	0	0	1	1

Figure 15-69. CPU Control and Status Register

This register enables the CLKOE output and permits the host to reset the 8051 using a firmware download.

Bit 7-4: **RV[3..0]** *Silicon Revision*

These register bits define the silicon revision. Consult individual Cypress Semiconductor data sheets for values.

Bit 3: **24/48** *8051 Clock Frequency*

This read-only bit indicates that the 8051 clock rate is 24 or 48 MHz. This bit is set at power-on according to a bit in the EEPROM connected to the EZ-USB FX I^2C-compatible bus. If no EEPROM is connected, the EZ-USB FX defaults to a 24-MHz 8051 clock. Once running (after boot), the 8051 cannot change the clock rate.

Bit 2: **CLKINV** *Invert the CLKOUT signal*

This read-only bit indicates that the CLKOUT signal is inverted. This bit is set at power-on according to a bit in the EEPROM connected to the EZ-USB FX I^2C-compatible bus. If no EEPROM is connected, the EZ-USB FX defaults to a non-inverted 24-MHz 8051 clock.

When CLKINV=0, the clock has the polarity shown in all the timing diagrams in this manual. When CLKINV=1, the clock is inverted.

Bit 1: **CLKOE** *CLKOUT pin output enable*

The CLKOUT signal may be disabled by floating the CLKOUT pin. The 8051 does this by clearing CLKOE. This is a good idea if the CLKOUT pin is not used since it reduces EMI.

Bit 0: **8051RES** *8051 reset*

The USB host writes "1" to this bit to reset the 8051, and "0" to run the 8051. Only the USB host can write this bit.

15.31 Port Configuration

PORTACFG **I/O Port A Configuration** **7F93**

b7	b6	b5	b4	b3	b2	b1	b0
RxD1OUT	**RxD0OUT**	**FRD**	**FWR**	**CS**	**OE**	**T1OUT**	**T0OUT**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PORTBCFG **I/O Port B Configuration** **7F94**

b7	b6	b5	b4	b3	b2	b1	b0
T2OUT	**INT6**	**INT5**	**INT4**	**TXD1**	**RXD1**	**T2EX**	**T2**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PORTCCFG **I/O Port C Configuration** **7F95**

b7	b6	b5	b4	b3	b2	b1	b0
RD	**WR**	**T1**	**T0**	**INT1**	**INT0**	**TXD0**	**RXD0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-70. I/O Port Configuration Registers

These three registers select between I/O ports and various alternate functions for I/O ports PORTA, PORTB, and PORTC. They are read/write by the 8051.

When PORTnCFG=0, the port pin functions as I/O, using the OUT, PINS, and OE control bits. Data written to an OUTn Registers appears on an I/O Port pin if the corresponding output enable bit (OEn) is HI.

When PORTnCFG=1, the pin assumes the alternate function shown in Table 15-15 on the following page.

For more information, see *Section 4.3. "Input/Output Port Registers".*

These registers are used in conjunction with the IFCONFIG PORTACF2 Registers to define the pin functions.

Table 15-15. I/O Pin Alternate Functions

I/O	Name	Alternate Functions
PA0	T0OUT	Timer 0 Output
PA1	T1OUT	Timer 1 Output
PA2	OE#	External Memory Output Enable
PA3	CS#	External Memory Chip Select
PA4	FWR#	Fast Access Write Strobe
PA5	FRD#	Fast Access Read Strobe
PA6	RXD0OUT	Mode 0: UART0 Synchronous Data Output
PA7	RXD1OUT	Mode 0: UART1 Synchronous Data Output
PB0	T2	Timer/Counter 2 Clock Input
PB1	T2EX	Timer/Counter 2 Capture/Reload Input
PB2	RxD1	Serial Port 1 Input
PB3	TxD1	Mode 0: Clock Output Modes 1-3: Serial Port 1 Data Output
PB4	INT4	INT4 Interrupt Request
PB5	INT5#	INT5 Interrupt Request
PB6	INT6	INT6 Interrupt Request
PB7	T2OUT	Timer/Counter 2 Overflow Indication
PC0	RxD0	Serial Port 0 Input
PC1	TxD0	Mode 0: Clock Output Modes 1-3: Serial Port 0 Data Output
PC2	INT0#	INT0 Interrupt Request
PC3	INT1#	INT1 Interrupt Request
PC4	T0	Timer/Counter 0 External Input
PC5	T1	Timer/Counter 1 External Input
PC6	WR#	External Memory Write Strobe
PC7	RD#	External Memory Read Strobe

15.32 Input/Output Port Registers A - C

For more information, see Section 4.3. *"Input/Output Port Registers."*

15.32.1 Outputs

The OUTn Registers provide the data that drives the port pin when OE=1 **and** the pin is configured for port output. If the port pin is selected as an input (OE=0), the value stored in the corresponding OUTn Bit is stored in an output latch but not used.

OUTA **Port A Outputs** **7F96**

b7	b6	b5	b4	b3	b2	b1	b0
OUTA7	**OUTA6**	**OUTA5**	**OUTA4**	**OUTA3**	**OUTA2**	**OUTA1**	**OUTA0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-71. Port A Outputs

OUTB **Port B Outputs** **7F97**

b7	b6	b5	b4	b3	b2	b1	b0
OUTB7	**OUTB6**	**OUTB5**	**OUTB4**	**OUTB3**	**OUTB2**	**OUTB1**	**OUTB0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-72. Port B Outputs

OUTC **Port C Outputs** **7F98**

b7	b6	b5	b4	b3	b2	b1	b0
OUTC7	**OUTC6**	**OUTC5**	**OUTC4**	**OUTC3**	**OUTC2**	**OUTC1**	**OUTC0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-73. Port C Outputs

15.32.2 Pins

The PINSn Registers contain the current value of the port pins, whether they are selected as I/O ports or as alternate functions.

PINSA **Port A Pins** **7F99**

b7	b6	b5	b4	b3	b2	b1	b0
PINA7	**PINA6**	**PINA5**	**PINA4**	**PINA3**	**PINA2**	**PINA1**	**PINA0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

Figure 15-74. Port A Pins

PINSB **Port B Pins** **7F9A**

b7	b6	b5	b4	b3	b2	b1	b0
PINB7	**PINB6**	**PINB5**	**PINB4**	**PINB3**	**PINB2**	**PINB1**	**PINB0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

Figure 15-75. Port B Pins

OUTC **Port C Pins** **7F98**

b7	b6	b5	b4	b3	b2	b1	b0
PINC7	**PINC6**	**PINC5**	**PINC4**	**PINC3**	**PINC2**	**PINC1**	**PINC0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

Figure 15-76. Port C Pins

15.32.3 Output Enables

The OE Registers control the output enables on the tri-state drivers connected to the port pins. When these bits are “1,” the port is an output, unless the corresponding PORTnCFG Bit is set to a “1.”

OEA **Port A Output Enable** **7F9C**

b7	b6	b5	b4	b3	b2	b1	b0
OEA7	**OEA6**	**OEA5**	**OEA4**	**OEA3**	**OEA2**	**OEA1**	**OEA0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-77. Port A Output Enable

OEB **Port B Output Enable** **7F9D**

b7	b6	b5	b4	b3	b2	b1	b0
OEB7	**OEB6**	**OEB5**	**OEB4**	**OEB3**	**OEB2**	**OEB1**	**OEB0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-78. Port B Output Enable

OEC **Port C Output Enable** **7F9E**

b7	b6	b5	b4	b3	b2	b1	b0
OEC7	**OEC6**	**OEC5**	**OEC4**	**OEC3**	**OEC2**	**OEC1**	**OEC0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-79. Port C Output Enable

15.33 Isochronous Control/Status Registers

ISOERR	Isochronous OUT EP Error	7FA0

b7	b6	b5	b4	b3	b2	b1	b0
ISO15ERR	**ISO14ERR**	**ISO13ERR**	**ISO12ERR**	**ISO11ERR**	**ISO10ERR**	**ISO9ERR**	**ISO8ERR**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

Figure 15-80. Isochronous OUT Endpoint Error Register

The ISOERR bits are updated at every SOF. They indicate that a CRC error was received on a data packet for the current frame. The ISOERR Bit status refers to the USB data received in the previous frame, and which is currently in the endpoint's OUT FIFO. If the ISOERR Bit = 1, indicating a bad CRC check, the data is still available in the OUTnDATA Register.

ISOCTL	Isochronous Control	7FA1

b7	b6	b5	b4	b3	b2	b1	b0
-	**-**	**-**	**-**	**PPSTAT**	**MBZ**	**MBZ**	**ISODISAB**
R	R	R	R	R	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-81. Isochronous Control Register

Bit 3: **PPSTAT** *Ping-Pong Status*

This bit indicates the isochronous buffer currently in use by the USB core. It is used only for diagnostic purposes.

Bits 2,1: **MBZ** *Must be zero*

These bits must always be written with zeros.

Bit 0: **ISODISAB** *ISO Endpoints Disable*

ISODISAB=0 enables all 16 isochronous endpoints

ISODISAB=1 disables *all 16* isochronous endpoints, making the 2,048 bytes of isochronous FIFO memory available as 8051 data memory at 0x2000-0x27FF.

ZBCOUT **Zero Byte Count Bits** **7FA2**

b7	b6	b5	b4	b3	b2	b1	b0
EP15	**EP14**	**EP13**	**EP12**	**EP11**	**EP10**	**EP9**	**EP8**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

Figure 15-82. Zero Byte Count Register

Bits 0-7: **EP(n)** *Zero Byte Count for ISO OUT Endpoints*

The 8051 can check these bits as a fast way to check all of the OUT isochronous endpoints at once for no data received during the previous frame. A “1” in any bit position means that a zero byte Isochronous OUT packet was received for the indicated endpoint.

15.34 I^2C-Compatible Registers

† **Read/write latency note:** These registers need the equivalent of 2 instruction clock cycles of time between performing the following instructions back-to-back: (1) write-write (2) write-read.

I2CS **I^2C-Compatible Control and Status** **7FA5†**
Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
START	**STOP**	**LASTRD**	**ID1**	**ID0**	**BERR**	**ACK**	**DONE**
R/W	R/W	R/W	R	R	R	R	R
0	0	0	x	x	0	0	0

I2DAT **I^2C-Compatible Data** **7FA6†**
Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-83. I^2C-Compatible Transfer Registers

The 8051 uses these registers to transfer data over the EZ-USB FX I^{2}C-compatible bus. For I^{2}C-compatible peripherals that support it, the EZ-USB FX I^{2}C-compatible bus can run at 400 KHz. For compatibility, the EZ-USB FX powers-up at the 100-KHz frequency.

In the EZ-USB FX family, an I^{2}C-compatible Interrupt Request occurs on INT3 whenever the DONE Bit (I2CS.0) makes a zero-to-one transition. This interrupt signals the 8051 that the I^{2}C-compatible controller is ready for another command. For more information on the I^{2}C-compatible interrupt, see Section 12.14. *"I2C-Compatible STOP Complete Interrupt."*

Bit 7: **START** *Signal START condition*

The 8051 sets the START Bit to “1” to prepare an I^{2}C-compatible bus transfer. If START=1, the next 8051 load to I2DAT will generate the start condition followed by the serialized byte of data in I2DAT. The 8051 loads byte data into I2DAT after setting the START Bit. The I^{2}C-compatible controller clears the START Bit during the ACK interval.

Bit 6: **STOP** *Signal STOP condition*

The 8051 sets STOP=1 to terminate an I^{2}C-compatible bus transfer. The I^{2}C-compatible controller clears the STOP Bit after completing the STOP condition. If the 8051 sets the STOP Bit during a byte transfer, the STOP condition will be generated immediately following the ACK phase of the byte transfer. If no byte transfer is occurring when the STOP Bit is set, the STOP condition will be carried out immediately on the bus. Data should not be written to I2CS or I2DAT until the STOP Bit returns low.

Bit 5: **LASTRD** *Last Data Read*

To read data over the I^{2}C-compatible bus, an I^{2}C-compatible master floats the SDA line and issues clock pulses on the SCL line. After every eight bits, the master drives SDA low for one clock to indicate ACK. To signal the last byte of the read transfer, the master floats SDA at ACK time to instruct the slave to stop sending. This is controlled by the 8051 by setting LastRD=1 before reading the last byte of a read transfer. The I^{2}C-compatible controller clears the LastRD Bit at the end of the transfer (at ACK time).

Setting LastRD does not automatically generate a STOP condition. The 8051 should also set the STOP Bit at the end of a read transfer.

Bit 4-3: **ID1,ID0** *Boot EEPROM ID*

These bits are set by the boot loader to indicate whether an 8-bit address or 16-bit address EEPROM at slave address 000 or 001 was detected at power-on. Normally, they are used for debug purposes only.

Bit 2: **BERR** *Bus Error*

This bit indicates an I^{2}C-compatible bus error. BERR=1 indicates that there was bus contention, which results when an outside device drives the bus LO when it shouldn't, or when another bus master wins arbitration, taking control of the bus. BERR is cleared when 8051 reads or writes the IDATA Register.

Bit 1: **ACK** *Acknowledge Bit*

Every ninth SCL or a write transfer the slave indicates reception of the byte by asserting ACK. The EZ-USB FX controller floats SDA during this time, samples the SDA line, and updates the ACK Bit with the complement of the detected value. ACK=1 indicates acknowledge, and ACK=0 indicates not-acknowledge. The USB core updates the ACK Bit at the same time it sets DONE=1. The ACK Bit should be ignored for read transfers on the bus.

Bit 0: **DONE** *I^{2}C-CompatibleTransfer DONE*

The I^{2}C-compatible controller sets this bit whenever it completes a byte transfer, right after the ACK stage. The controller also generates an I^{2}C-compatible Interrupt Request (8051 INT3) when it sets the DONE Bit. The I^{2}C-compatible controller automatically clears the DONE Bit and the I^{2}C-compatible Interrupt Request bit whenever the 8051 reads or writes the I2DAT Register.

I2CMODE **I^{2}C-Compatible Mode** **7FA7**†
Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	0	0	**STOPIE**	0
R	R	R	R	R	R	R/W	R
0	0	0	0	0	0	0	0

Figure 15-84. I^{2}C-Compatible Mode Register

The I^{2}C-compatible STOP Bit Interrupt Request is activated when the STOP Bit makes a 1-0 transition. To enable this interrupt, set the STOPIE Bit in the I2CMODE Register. The 8051 determines the interrupt source by checking the DONE and STOP bits in the I2CS Register.

15.35 Interrupts

† **Read/write latency note:** These registers need the equivalent of 2 instruction clock cycles of time between performing the following instructions back-to-back: (1) write-write (2) write-read.

IVEC **Interrupt Vector** **7FA8**

b7	b6	b5	b4	b3	b2	b1	b0
0	**IV4**	**IV3**	**IV2**	**IV1**	**IV0**	**0**	**0**
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Figure 15-85. Interrupt Vector Register

IVEC indicates the source of an interrupt from the USB Core. When the USB core generates an INT2 (USB) Interrupt Request, it updates IVEC to indicate the source of the interrupt. The interrupt sources are encoded on IV[4..0] as shown in *Figure 12-1.*

IN07IRQ **Endpoint 0-7 IN Interrupt Request** **7FA9**†
Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
IN7IR	**IN6IR**	**IN5IR**	**IN4IR**	**IN3IR**	**IN2IR**	**IN1IR**	**IN0IR**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

OUT07IRQ **Endpoint 0-7 OUT Interrupt Requests** **7FAA**†
Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
OUT7IR	**OUT6IR**	**OUT5IR**	**OUT4IR**	**OUT3IR**	**OUT2IR**	**OUT1IR**	**OUT0IR**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-86. IN/OUT Interrupt Request (IRQ) Registers

These interrupt request (IRQ) registers indicate the pending interrupts for each bulk endpoint. An interrupt request (IR) Bit becomes active when the BSY Bit for an endpoint makes a transition from one to zero (when the endpoint becomes *un-busy*, giving access to the 8051). The IR bits function independently of the Interrupt Enable (IE) bits, so interrupt requests are held whether or not the interrupts are enabled.

The 8051 clears an interrupt request bit by writing a “1” to it. (See the following Note).

*Do **not** clear an IRQ Bit by reading an IRQ Register, ORing its contents with a bit mask, and writing back the IRQ Register. This will clear ALL pending interrupts. Instead, simply write the bit mask value (with a “1” in the bit position of the IRQ you want to clear) directly to the IRQ Register.*

USBIRQ **USB Interrupt Request** **7FAB**†
Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
-	**-**	**IBNIR**	**USESIR**	**SUSPIR**	**SUTOKIR**	**SOFIR**	**SUDAVIR**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-87. USB Interrupt Request (IRQ) Registers

USBIRQ indicates the interrupt request status of the USB reset, suspend, setup token, start of frame, and setup data available interrupts.

Bit 5: IBNIR *IN Bulk NAK Interrupt Request*

The USB core sets this bit when any of the IN bulk endpoints responds to an IN token with a NAK. This interrupt occurs when the host sends an IN token to a bulk IN endpoint which has not been armed by the 8051 writing its byte count register. Individual enables and requests (per endpoint) are controlled by the IBNIRQ and IBNIEN Registers (7FB0, 7FB1). Write a “1” to this bit to clear the interrupt request.

Bit 4: URESIR *USB Reset Interrupt Request*

The USB core sets this bit to “1” when it detects a USB bus reset.

Because this bit can change state while the 8051 is in reset, it may be active when the 8051 comes out of reset, although it is reset to “0” by a power-on reset. Write a “1” to this bit to clear the interrupt request. See *Chapter 13. "EZ-USB FX Resets"* for more information about this bit.

Bit 3: **SUSPIR** *USB Suspend Interrupt Request*

The USB core sets this bit to “1” when it detects USB SUSPEND signaling (no bus activity for 3 ms). Write a “1” to this bit to clear the interrupt request.

Because this bit can change state while the 8051 is in reset, it may be active when the 8051 comes out of reset, although it is reset to “0” by a power-on reset. See *Chapter 14. "EZ-USB FX Power Management"* for more information about this bit.

Bit 2: **SUTOKIR** *SETUP Token Interrupt Request*

The USB core sets this bit to “1” when it receives a SETUP token. Write a “1” to this bit to clear the interrupt request. See *Chapter 9. "EZ-USB FX Endpoint Zero"* for more information on the handling of SETUP tokens.

Because this bit can change state while the 8051 is in reset, it may be active when the 8051 comes out of reset, although it is reset to “0” by a power-on reset.

Bit 1: **SOFIR** *Start of frame Interrupt Request*

The USB core sets this bit to “1” when it receives a SOF packet. Write a “1” to this bit to clear the interrupt request.

Because this bit can change state while the 8051 is in reset, it may be active when the 8051 comes out of reset, although it is reset to “0” by a power-on reset.

Bit 0: **SUDAVIR** *SETUP data available Interrupt Request*

The USB core sets this bit to “1” when it has transferred the eight data bytes from an endpoint zero SETUP packet into internal registers (at SETUPDAT). Write a “1” to this bit to clear the interrupt request.

Because this bit can change state while the 8051 is in reset, it may be active when the 8051 comes out of reset, although it is reset to “0” by a power-on reset.

IN07EN **Endpoint 0-7 IN Interrupt Enables** **7FAC†**
Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
IN7IEN	**IN6IEN**	**IN5IEN**	**IN4IEN**	**IN3IEN**	**IN2IEN**	**IN1IEN**	**IN0IEN**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

OUT07IEN **Endpoint 0-7 OUT Interrupt Enables** **7FAD†**
Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
OUT7IEN	**OUT6IEN**	**OUT5IEN**	**OUT4IEN**	**OUT3IEN**	**OUT2IEN**	**OUT1IEN**	**OUT0IEN**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-88. IN/OUT Interrupt Enable Registers

The Endpoint Interrupt Enable individually enable the BULK endpoint interrupts. They do not affect the endpoint action, only the generation of an interrupt in response to endpoint events.

When the IEN Bit for an endpoint is “0,” the interrupt request bit for that endpoint is ignored, but saved. When the IEN Bit for an endpoint is “1,” any IRQ Bit equal to “1” generates an 8051 INT2 Request.

The INT2 interrupt (EIE.0) and the 8051 global interrupt enable (EA) must be enabled for the endpoint interrupts to propagate to the 8051. Once the INT2 interrupt is active, it must be cleared by software.

USBIEN **USB Interrupt Enable** **7FAE†**
Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
-	**-**	**IBNIE***	**URESIE**	**SUSPIE**	**SUTOKIE**	**SOFIE**	**SUDAVIE**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-89. USB Interrupt Enable Register

USBIEN bits gate the interrupt request to the 8051 for USB reset, suspend, SETUP token, start of frame, and SETUP data available.

Bit 5: **IBNIE** *IN bulk NAK Interrupt Enable*

The 8051 sets this bit to enable the IN-bulk-NAK interrupt. This interrupt occurs when the host sends an IN token to a bulk IN endpoint which has not been *armed* by the 8051 writing its byte count register. Individual enables and requests (per endpoint) are controlled by the IBNIRQ and IBNIEN Registers (7FB0, 7FB1).

Bit 4: **URESIE** *USB Reset Interrupt Enable*

This bit is the interrupt mask for the URESIR Bit. When this bit is “1,” the interrupt is enabled, when it is “0,” the interrupt is disabled.

Bit 3: **SUSPIE** *USB Suspend Interrupt Enable*

This bit is the interrupt mask for the SUSPIR Bit. When this bit is “1,” the interrupt is enabled, when it is “0,” the interrupt is disabled.

Bit 2: **SUTOKIE** *SETUP Token Interrupt Enable*

This bit is the interrupt mask for the SUTOKIR Bit. When this bit is “1,” the interrupt is enabled, when it is “0,” the interrupt is disabled.

Bit 1: **SOFIE** *Start of frame Interrupt Enable*

This bit is the interrupt mask for the SOFIE Bit. When this bit is “1,” the interrupt is enabled, when it is “0,” the interrupt is disabled.

Bit 0: **SUDAVIE** *SETUP data available Interrupt Enable*

This bit is the interrupt mask for the SUDAVIE Bit. When this bit is “1,” the interrupt is enabled, when it is “0,” the interrupt is disabled.

USBBAV **Breakpoint and Autovector** **7FAF**

b7	b6	b5	b4	b3	b2	b1	b0
-	**-**	**-**	**INT2SFC**	**BREAK**	**BPPULSE**	**BPEN**	**AVEN**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-90. Breakpoint and Autovector Register

Bit 4: **INT2SFC** *Interrupt 2 cleared by SFR*

If INT2SFC=1, the IRQ2 flag can be quickly cleared by writing any value to the INT2CLR SFR .

Bit 3: **BREAK** *Breakpoint enable*

The BREAK Bit is set when the 8051 address bus matches the address held in the bit breakpoint address registers (7FB2, 7FB3). The BKPT pin reflects the state of this bit. The 8051 writes a “1” to the BREAK Bit to clear it. It is not necessary to clear the BREAK Bit if the pulse mode bit (BPPULSE) is set.

Bit 2: **BPPULSE** *Breakpoint pulse mode*

The 8051 sets this bit to “1” to pulse the BREAK Bit (and BKPT pin) high for 8 CLKOUT cycles when the 8051 address bus matches the address held in the breakpoint address registers. When this bit is set to “0,” the BREAK Bit (and BKPT pin) remains high until it is cleared by the 8051.

Bit 1: **BPEN** *Breakpoint enable*

If this bit is “1,” a BREAK signal is generated whenever the 16-bit address lines match the value in the Breakpoint Address Registers (BPADDRH/L). The behavior of the BREAK Bit and associated BKPT pin signal is either latched or pulsed, depending on the state of the BPPULSE Bit.

Bit 0: **AVEN** *Auto-vector enable*

If this bit is “1,” the EZ-USB FX Auto-vector feature is enabled for USB (INT2) interrupts. If it is 0, the auto-vector feature is disabled. See *Chapter 12. "EZ-USB FX Interrupts"* for more information on the auto-vector feature. Note: a separate bit, AV4EN in the INT4SETUP (785E) enables the INT4 autovector.

IBNIRQ **IN Bulk NAK Interrupt Requests** **7FB0†**
Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
-	**EP6IR**	**EP5IR**	**EP4IR**	**EP3IR**	**EP2IR**	**EP1IR**	**EP0IR**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-91. IN Bulk NAK Interrupt Request Register

These bits are set when a bulk IN endpoint (0-6) received an IN token while the endpoint was not armed for data transfer. In this case the SIE automatically sends a NAK response, and sets the corresponding IBNIRQ Bit. If the IBN interrupt is enabled (USBIEN.5=1), and the endpoint interrupt is enabled in the IBNIEN Register, an interrupt is request generated. The 8051 can test the

IBNIRQ Register to determine which of the endpoints caused the interrupt. The 8051 clears an IBNIRQ Bit by writing a “1” to it.

IBNIEN **IN Bulk NAK Interrupt Enables** **7FB1**†
Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
-	**EP6IE**	**EP5IE**	**EP4IE**	**EP3IE**	**EP2IE**	**EP1IE**	**EP0IE**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	0	0	0	0	0	0

Figure 15-92. IN Bulk NAK Interrupt Enable Register

Each of the individual IN endpoints may be enabled for an IBN interrupt using the IBNEN Register. The 8051 sets an interrupt enable bit to 1 to enable the corresponding interrupt.

BPADDRH **Breakpoint Address High** **7FB2**

b7	b6	b5	b4	b3	b2	b1	b0
A15	**A14**	**A13**	**A12**	**A11**	**A10**	**A9**	**A8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

BPADDRL **Breakpoint Address Low** **7FB3**

b7	b6	b5	b4	b3	b2	b1	b0
A7	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-93. IN/OUT Interrupt Enable Registers

When the current 16-bit address (code or XDATA) matches the BPADDRH/BPADDRL address, a breakpoint event occurs. The BPPULSE and BPEN bits in the USBBAV Register control the action taken on a breakpoint event.

If the BPEN Bit is “0,” address breakpoints are ignored. If BPEN is “1” and BPPULSE is “1,” an 8 CLKOUT wide pulse appears on the BKPT pin. If BPEN is “1” and BPPULSE is “0,” the BKPT pin remains active until the 8051 clears the BREAK Bit by writing “1” to it.

15.36 Endpoint 0 Control and Status Registers

† **Read/write latency note:** These registers need the equivalent of 2 instruction clock cycles of time between performing the following instructions back-to-back: (1) write-write (2) write-read.

EP0CS **Endpoint Zero Control and Status** **7FB4†**
Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
-	**-**	**-**	**-**	**OUTBSY**	**INBSY**	**HSNAK**	**EP0STALL**
R	R	R	R	R	R	R/W	R/W
0	0	0	0	1	0	0	0

IN0BC **Endpoint Zero IN Byte Count** **7FB5†**
Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
-	**BC6**	**BC5**	**BC4**	**BC3**	**BC2**	**BC1**	**BC0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

OUT0BC **Endpoint Zero OUT Byte Count** **7FC5†**
Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
-	**BC6**	**BC5**	**BC4**	**BC3**	**BC2**	**BC1**	**BC0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Figure 15-94. Port Configuration Registers

These registers control EZ-USB FX CONTROL endpoint zero. Because endpoint zero is a bi-directional endpoint, the IN and OUT functionality is controlled by a single control and status (CS) register, unlike endpoints 1-7, which have separate INCS and OUTCS Registers.

Bit 3: **OUTBSY** *OUT Endpoint Busy*

OUTBSY is a read-only bit that is automatically cleared when a SETUP token arrives. The 8051 sets the OUTBSY Bit by writing a byte count to EPOUTBC.

If the CONTROL transfer uses an OUT data phase, the 8051 must load a dummy byte count into OUT0BC to arm the OUT endpoint buffer. Until it does, the USB core will NAK the OUT tokens.

Bit 2: **INBSY** *IN Endpoint Busy*

INBSY is a read-only bit that is automatically cleared when a SETUP token arrives. The 8051 sets the INBSY Bit by writing a byte count to IN0BC.

If the CONTROL transfer uses an IN data phase, the 8051 loads the requested data into the IN0BUF buffer, and then loads the byte count into IN0BC to arm the data phase of the CONTROL transfer. Alternatively, the 8051 can arm the data transfer by loading an address into the Setup Data Pointer Registers SUDPTRH/L. Until armed, the USB core will NAK the IN tokens.

Bit 1: **HSNAK** *Handshake NAK*

HSNAK (Handshake NAK) is a read/write bit that is automatically set when a SETUP token arrives. The 8051 clears HSNAK by writing a “1” to the register bit.

While HSNAK=1, the USB core NAKs the handshake (status) phase of the CONTROL transfer. When HSNAK=0, it ACKs the handshake phase. The 8051 can clear HSNAK at any time during a CONTROL transfer.

Bit 0: **EP0STALL** *Endpoint Zero Stall*

EP0STALL is a read/write bit that is automatically cleared when a SETUP token arrives. The 8051 sets EP0STALL by writing a “1” to the register bit.

While EP0STALL=1, the USB core sends the STALL PID for any IN or OUT token. This can occur in either the data or handshake phase of the CONTROL transfer.

To indicate an endpoint stall on endpoint zero, set both EP0STALL and HSNAK bits. Setting the EP0STALL Bit alone causes endpoint zero to NAK forever because the host keeps the control transfer pending.

15.37 Endpoint 1-7 Control and Status Registers

† **Read/write latency note:** These registers need the equivalent of 2 instruction clock cycles of time between performing the following instructions back-to-back: (1) write-write (2) write-read.

Endpoints 1-7 IN and OUT are used for bulk or interrupt data.Table 15-16 shows the addresses for the control/status and byte count registers associated with these endpoints. The bi-directional CONTROL endpoint zero registers are described in Section 15.36. *"Endpoint 0 Control and Status Registers."*

Table 15-16. Control and Status Register Addresses for Endpoints 0-7

Address	Function	Name
7FB4†	Control and Status - Endpoint IN0	EP0CS
7FB5†	Byte Count - Endpoint IN0	IN0BC
7FB6†	Control and Status - Endpoint IN1	IN1CS
7FB7†	Byte Count - Endpoint IN1	IN1BC
7FB8†	Control and Status - Endpoint IN2	IN2CS
7FB9†	Byte Count - Endpoint IN2	IN2BC
7FBA†	Control and Status - Endpoint IN3	IN3CS
7FBB†	Byte Count - Endpoint IN3	IN3BC
7FBC†	Control and Status - Endpoint IN4	IN4CS
7FBD†	Byte Count - Endpoint IN4	IN4BC
7FBE†	Control and Status - Endpoint IN5	IN5CS
7FBF†	Byte Count - Endpoint IN5	IN5BC
7FC0†	Control and Status - Endpoint IN6	IN6CS
7FC1†	Byte Count - Endpoint IN6	IN6BC
7FC2†	Control and Status - Endpoint IN7	IN7CS
7FC3†	Byte Count - Endpoint IN7	IN7BC
7FC4	*Reserved*	
7FC5†	Byte Count - Endpoint OUT0	OUT0BC
7FC6†	Control and Status - Endpoint OUT1	OUT1CS
7FC7†	Byte Count - Endpoint OUT1	OUT1BC
7FC8†	Control and Status - Endpoint OUT2	OUT2CS
7FC9†	Byte Count - Endpoint OUT2	OUT2BC
7FCA†	Control and Status - Endpoint OUT3	OU37CS
7FCB†	Byte Count - Endpoint OUT3	OUT3BC
7FCC†	Control and Status - Endpoint OUT4	OUT4CS
7FCD†	Byte Count - Endpoint OUT4	OUT4BC
7FCE†	Control and Status - Endpoint OUT5	OUT5CS
7FCF†	Byte Count - Endpoint OUT5	OUT5BC
7FD0†	Control and Status - Endpoint OUT6	OUT6CS
7FD1†	Byte Count - Endpoint OUT6	OUT6BC
7FD2†	Control and Status - Endpoint OUT7	OUT7CS
7FD3†	Byte Count - Endpoint OUT7	OUT7BC

INnCS	Endpoint (1-7) IN Control and Status	7FB6-7FC2*† Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
-	**-**	**-**	**-**	**-**	**-**	**INnBSY**	**INnSTL**
R	R	R	R	R	R	R/W	R/W
0	0	0	0	0	0	0	0

* See Table 15-16 for individual control/status register addresses.

Figure 15-95. IN Control and Status Registers

Bit 1: INnBSY *IN Endpoint (1-7) Busy*

The BSY Bit indicates the status of the endpoint's IN Buffer INnBUF. The USB core sets BSY=0 when the endpoint's IN buffer is empty and ready for loading by the 8051. The 8051 causes BSY=1 by loading the endpoint's byte count register.

When BSY=1, the 8051 should not write data to an IN endpoint buffer, because the endpoint FIFO could be in the act of transferring data to the host over the USB. BSY=0 when the USB IN transfer is complete and endpoint RAM data is available for 8051 access. USB IN tokens for the endpoint are NAKd while BSY=0 (the 8051 is still loading data into the endpoint buffer).

A 1-to-0 transition of BSY (indicating that the 8051 can access the buffer) generates an interrupt request for the IN endpoint. After the 8051 writes the data to be transferred to the IN endpoint buffer, it loads the endpoint's byte count register with the number of bytes to transfer, which automatically sets BSY=1. This enables the IN transfer of data to the host in response to the next IN token. Again, the CPU should never load endpoint data while BSY=1.

The 8051 writes a “1” to an IN endpoint busy bit to disarm a previously armed endpoint. (This sets BSY=0.) The 8051 program should do this only after a USB bus reset, or when the host selects a new interface or alternate setting that uses the endpoint. This prevents stale data from a previous setting from being accepted by the host's first IN transfer that uses the new setting.

Even though the register description shows bit 1 as “R/W,” the 8051 can only clear this bit by writing a “1” to it. The 8051 can not directly set this bit.

To disarm a paired IN endpoint, write a “1” to the busy bit for *both* endpoints in the pair.

Bit 0: INnSTL *IN Endpoint (1-7) Stall*

The 8051 sets this bit to “1” to *stall* an endpoint, and to “0” to clear a stall.

When the stall bit is “1,” the USB core returns a STALL Handshake for all requests to the endpoint. This notifies the host that something unexpected has happened.

The 8051 sets an endpoint’s stall bit under two circumstances:

1. The host sends a “Set_Feature—Endpoint Stall” Request to the specific endpoint.
2. The 8051 encounters any *show stopper* error on the endpoint, and sets the stall bit to tell the host to halt traffic to the endpoint.

The 8051 clears an endpoint’s stall bit under two circumstances:

1. The host sends a “Clear_Feature—Endpoint Stall” Request to the specific endpoint.
2. The 8051 receives some other indication from the host that the stall should be cleared (this is referred to as “host intervention” in the USB Specification). This indication could be a USB bus reset.

All stall bits are automatically cleared when the EZ-USB FX chip ReNumerates™ by pulsing the DISCON Bit HI.

INnBC **Endpoint (1-7) IN Byte Count** **7FB7-7FC3*†**

Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
-	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

* See Table 15-16 for individual byte count register addresses.

Figure 15-96. IN Byte Count Registers

The 8051 writes this register with the number of bytes it loaded into the IN endpoint buffer INnBUF. Writing this register also *arms* the endpoint by setting the endpoint BSY Bit to 1.

Legal values for these registers are 0-64. A zero transfer size is used to terminate a transfer that is an integral multiple of MaxPacketSize. For example, a 256-byte transfer with maxPacketSize = 64, would require four packets of 64 bytes each plus one packet of 0 bytes.

The IN byte count should never be written while the endpoint’s BUSY Bit is set.

When the register pairing feature is used (*Chapter 6. "EZ-USB FX Bulk Transfers"*) IN2BC is used for the EP2/EP3 pair, IN4BC is used for the EP4/EP5 pair, and IN6BC is used for the EP6/EP7 pair. In the *paired* (double-buffered) mode, after the first write to the even-numbered byte count register, the endpoint BSY Bit remains at 0, indicating that only one of the buffers is full, and the other is still empty. The odd numbered byte count register is not used when endpoints are paired.

OUTnCS	Endpoint (1-7) OUT Control and Status	7FC6-7FD2*† Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
-	**-**	**-**	**-**	**-**	**-**	**OUTnBSY**	**OUTnSTL**
R	R	R	R	R	R	R	R/W
0	0	0	0	0	0	0	0

* See Table 15-16 for individual control/status register addresses.

Figure 15-97. OUT Control and Status Registers

Bit 1: **OUTnBSY** *OUT Endpoint (1-7) Busy*

The BSY Bit indicates the status of the endpoint's OUT Buffer OUTnBUF. The USB core sets BSY=0 when the host data is available in the OUT buffer. The 8051 sets BSY=1 by loading the endpoint's byte count register.

When BSY=1, endpoint RAM data is invalid--the endpoint buffer has been emptied by the 8051 and is waiting for new OUT data from the host, or it is the process of being loaded over the USB. BSY=0 when the USB OUT transfer is complete and endpoint RAM data in OUTn-BUF is available for the 8051 to read. USB OUT tokens for the endpoint are NAKd while BSY=1 (the 8051 is still reading data from the OUT endpoint).

A 1-to-0 transition of BSY (indicating that the 8051 can access the buffer) generates an interrupt request for the OUT endpoint. After the 8051 reads the data from the OUT endpoint buffer, it loads the endpoint's byte count register with any value to re-arm the endpoint, which automatically sets BSY=1. This enables the OUT transfer of data from the host in response to the next OUT token. The CPU should never read endpoint data while BSY=1.

Bit 0: **OUTnSTL** *OUT Endpoint (1-7) Stall*

The 8051 sets this bit to "1" to *stall* an endpoint, and to "0" to clear a stall.

When the stall bit is "1," the USB core returns a STALL Handshake for all requests to the endpoint. This notifies the host that something unexpected has happened.

The 8051 sets an endpoint's stall bit under two circumstances:

1. The host sends a "Set_Feature—Endpoint Stall" Request to the specific endpoint.
2. The 8051 encounters any *show stopper* error on the endpoint, and sets the stall bit to tell the host to halt traffic to the endpoint.

The 8051 clears an endpoint's stall bit under two circumstances:

1.The host sends a "Clear_Feature—Endpoint Stall" Request to the specific endpoint.
2.The 8051 receives some other indication from the host that the stall should be cleared (this is referred to as "host intervention" in the USB Specification).

All stall bits are automatically cleared when the EZ-USB FX chip ReNumerates™.

OUTnBC **Endpoint (1-7) OUT Byte Count** **7FC7-7FD3*†**
Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
-	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R	R	R	R	R	R	R	R/W
0	0	0	0	0	0	0	0

* See Table 15-16 for individual control/status register addresses.

Figure 15-98. OUT Byte Count Registers

The 8051 reads this register to determine the number of bytes sent to an OUT endpoint. Legal sizes are 0 - 64 bytes.

Each EZ-USB FX bulk OUT endpoint has a byte count register, which serves two purposes. The 8051 *reads* the byte count register to determine how many bytes were received during the last OUT transfer from the host. The 8051 *writes* the byte count register (with any value) to tell the USB core that it has finished reading bytes from the buffer, making the buffer available to accept the next OUT transfer. Writing the byte count register sets the endpoint's BSY Bit to "1."

When the register-pairing feature is used, OUT2BC is used for the EP2/EP3 pair, OUT4BC is used for the EP4/EP5 pair, and OUT6BC is used for the EP6/EP7 pair. The odd-numbered byte count registers should not be used. When the 8051 writes a byte to the even numbered byte count register, the USB core switches buffers. If the other buffer already contains data to be read by the 8051, the OUTnBSY Bit remains at "0."

All OUT tokens are NAKd until the 8051 is released from RESET, whereupon the ACK/NAK behavior is based on pairing.

15.38 Global USB Registers

† **Read/write latency note:** These registers need the equivalent of 2 instruction clock cycles of time between performing the following instructions back-to-back: (1) write-write (2) write-read.

SUDPTRH **Setup Data Pointer High** **7FD4†**
Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
A15	**A14**	**A13**	**A12**	**A11**	**A10**	**A9**	**A8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

SUDPTRL **Setup Data Pointer Low** **7FD5†**
Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
A7	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-99. Setup Data Pointer High/Low Registers

When the EZ-USB FX chip receives a “Get_Descriptor” Request on endpoint zero, it can instruct the USB core to handle the multi-packet IN transfer by loading these registers with the address of an internal table containing the descriptor data. The descriptor data tables may be placed in internal program/data RAM or in unused *Endpoint 0-7 RAM*. The SUDPTR does not operate with external memory. The SUDPTR Registers should be loaded in HIGH/LOW order.

In addition to loading SUDPTRL, the 8051 must also clear the HSNAK Bit in the EP0CS Register (by writing a “1” to it) to complete the CONTROL transfer.

Any host request that uses the EZ-USB FX Setup Data Pointer to transfer IN data must indicate the number of bytes to transfer in bytes 6 (wLenghthL) and 7 (wLengthH) of the SETUP packet. These bytes are pre-assigned in the USB Specification to be length bytes in all standard device requests such as “Get_Descriptor.” If vendor-specific requests are used to transfer large blocks of data using the Setup Data Pointer, they must include this pre-defined length field in bytes 6-7 to tell the USB core how many bytes to transfer using the Setup Data Pointer.

The USB core transfers the lesser of (a) the bytes requested in the SETUP packet, and (b) the bytes in the length field of the descriptor pointed to by the Setup Data Pointer.

USBCS **USB Control and Status** **7FD6†**

Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
WAKESRC	**-**	**-**	**-**	**DISCON**	**DISCOE**	**RENUM**	**SIGRSUME**
R/W	R	R	R	R/W	R/W	R/W	R/W
0	0	0	0	0	1	0	0

Figure 15-100. USB Control and Status Registers

Bit 7: **WAKESRC** *Wakeup source*

This bit indicates that a high to low transaction was detected on the WAKEUP# pin. Writing a “1” to this bit resets it to “0.”

Bit 3: **DISCON** *Signal a Disconnect on the DISCON# pin*

The EZ-USB FX DISCON# pin reflects the complement of this bit. This bit is normally set to 0.

Bit 2: **DISCOE** *Disconnect Output Enable*

DISCOE controls the output buffer on the DISCON# pin. When DISCOE=0, the pin floats, and when DISCOE=1, it drives to the complement of the DISCON Bit (above).

DISCOE is used in conjunction with the RENUM Bit to perform ReNumeration™, (*Chapter 5. "EZ-USB FX Enumeration & ReNumeration™"*).

Bit 1: **RENUM** *ReNumerate*

This bit controls which entity, the USB core or the 8051, handles USB device requests. When RENUM=0, the USB core handles all device requests. When RENUM=1, the 8051 handles all device requests except Set_Address.

The 8051 sets RENUM=1 during a bus disconnect to transfer USB control to the 8051. The USB core automatically sets RENUM=1 under two conditions:

1. Completion of a “B6” boot load (*Chapter 5. "EZ-USB FX Enumeration & ReNumeration™"*.
2. When external memory is used (EA=1) and no boot I^{2}C-compatible EEPROM is used (see Section 13.3.3. *"External ROM"*).

Bit 0: **SIGRSUME** *Signal remote device resume*

The 8051 sets SIGRSUME=1 to drive the “K” state onto the USB bus. This should be done only by a device that is capable of remote wakeup, and then only during the SUSPEND state. To signal RESUME, the 8051 sets SIGRSUME=1, waits 10-15 ms, then sets SIGRSUME=0.

TOGCTL	**Data Toggle Control**	**7FD7†** **Read/write latency applies**

b7	b6	b5	b4	b3	b2	b1	b0
Q	**S**	**R**	**IO**	**0**	**EP2**	**EP1**	**EP0**
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-101. Data Toggle Control Register

Bit 7: **Q** *Data Toggle Value*

Q=0 indicates DATA0 and Q=1 indicates DATA1, for the endpoint selected by the I/O and EP[2..0] bits. The 8051 writes the endpoint select bits (IO and EP[2..0]), before reading this value.

Bit 6: **S** *Set Data Toggle to DATA1*

After selecting the desired endpoint by writing the endpoint select bits (IO and EP[2..0]) the 8051 sets S=1 to set the data toggle to DATA1. The endpoint selection bits should not be changed while this bit is written.

At this writing there is no known reason to set an endpoint data toggle to 1. This bit is provided for generality and testing only.

Bit 5: **R** *Set Data Toggle to DATA0*

After selecting the desired endpoint by writing the endpoint select bits (IO and EP[2..0]) the 8051 sets R=1 to set the data toggle to DATA0. The endpoint selection bits should not be changed while this bit is written. For advice on when to reset the data toggle, see *Chapter 9. "EZ-USB FX Endpoint Zero"*.

Bit 4: **IO** *Select IN or OUT endpoint*

The 8051 sets this bit to select an endpoint direction prior to setting its R or S Bit. IO=0 selects an OUT endpoint, IO=1 selects an IN endpoint.

Bit 2-0: **EP** *Select endpoint*

The 8051 sets these bits to select an endpoint prior to setting its R or S Bit. Valid values are 0-7 to correspond to bulk endpoints IN0-IN7 and OUT0-OUT7.

USBFRAMEL **USB Frame Count Low** **7FD8**

b7	b6	b5	b4	b3	b2	b1	b0
FC7	**FC6**	**FC5**	**FC4**	**FC3**	**FC2**	**FC1**	**FC0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

USBFRAMEH **USB Frame Count High** **7FD9**

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**FC10**	**FC9**	**FC8**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

Figure 15-102. USB Frame Count High/Low Registers

Every millisecond the host sends a SOF token indicating “Start Of Frame,” along with an 11-bit incrementing frame count. The EZ-USB FX copies the frame count into these registers at every SOF. One use of the frame count is to respond to the USB SYNC_FRAME Request (*Chapter 9. "EZ-USB FX Endpoint Zero"*).

If the USB core detects a missing or garbledSOF, it generates an internal SOF and increments USBFRAMEL-USBRAMEH.

FNADDR **Function Address** **7FDB**

b7	b6	b5	b4	b3	b2	b1	b0
0	**FA6**	**FA5**	**FA4**	**FA3**	**FA2**	**FA1**	**FA0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

Figure 15-103. Function Address Register

During the USB enumeration process, the host sends a device a unique 7-bit address, which the USB core copies into this register. There is normally no reason for the CPU to know its

USB device address because the USB Core automatically responds only to its assigned address.

During ReNumeration™ the USB Core sets register to 0 to allow the EZ-USB FX chip to respond to the default address 0.

USBPAIR **USB Endpoint Pairing** **7FDD**†

Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
ISOSEND0	**-**	**PR6OUT**	**PR4OUT**	**PR2OUT**	**PR6IN**	**PR4IN**	**PR2IN**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	x	0	0	0	0	0	0

Figure 15-104. USB Endpoint Pairing Register

Bit 7: **ISOSEND0** *Isochronous Send Zero Length Data Packet*

The ISOSEND0 Bit is used when the EZ-USB FX chip receives an isochronous IN token while the IN FIFO is empty. If ISOSEND0=0 (the default value), the USB core does not respond to the IN token. If ISOSEND0=1, the USB core sends a zero-length data packet in response to the IN token. Which action to take depends on the overall system design. The ISOSEND0 Bit applies to all of the isochronous IN endpoints, IN8BUF through IN15BUF.

Bit 5-3: **PRnOUT** *Pair Bulk OUT Endpoints*

Set the endpoint pairing bits (PRxOUT) to “1” to enable double-buffering of the bulk OUT endpoint buffers. With double buffering enabled, the 8051 can operate on one buffer while another is being transferred over USB. The endpoint busy and interrupt request bits function identically, so the 8051 code requires no code modification to support double buffering.

When an endpoint is paired, the 8051 uses only the even-numbered endpoint of the pair. The 8051 should not use the paired odd endpoint’s IRQ, IEN, VALID bits or the buffer associated with the odd numbered endpoint.

Bit 2-0: **PRnIN** *Pair Bulk IN Endpoints*

Set the endpoint pairing bits (PRxIN) to “1” to enable double-buffering of the bulk IN endpoint buffers. With double buffering enabled, the 8051 can operate on one buffer while another is being transferred over USB.

When an endpoint is paired, the 8051 should access only the even-numbered endpoint of the pair. The 8051 should not use the IRQ, IEN, VALID bits or the buffer associated with the odd numbered endpoint.

IN07VAL **Endpoints 0-7 IN Valid Bits** **7FDE**†

Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
IN7VAL	**IN6VAL**	**IN5VAL**	**IN4VAL**	**IN3VAL**	**IN2VAL**	**IN1VAL**	**IN0VAL**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	1	0	1	0	1	1	1

OUT07VAL **Endpoints 0-7 OUT Valid Bits** **7FDF**†

Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
OUT7VAL	**OUT6VAL**	**OUT5VAL**	**OUT4VAL**	**OUT3VAL**	**OUT2VAL**	**OUT1VAL**	**OUT0VAL**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	1	0	1	0	1	0	1

Figure 15-105. IN/OUT Valid Bits Register

The 8051 sets VAL=1 for any active endpoints, and VAL=0 for inactive endpoints. These bits instruct the USB core to return a “no response” if an invalid endpoint is addressed, instead of a NAK.

The default values of these registers are set to support all endpoints that exist in the default USB device (see Table 5-1).

INISOVAL **Isochronous IN Endpoint Valid Bits** **7FE0†**

Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
IN15VAL	**IN14VAL**	**IN13VAL**	**IN12VAL**	**IN11VAL**	**IN10VAL**	**IN9VAL**	**IN8VAL**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	1	1	1

OUTISOVAL **Isochronous OUT Endpoint Valid Bits** **7FE1†**

Read/write latency applies

b7	b6	b5	b4	b3	b2	b1	b0
OUT15VAL	**OUT14VAL**	**OUT13VAL**	**OUT12VAL**	**OUT11VAL**	**OUT10VAL**	**OUT9VAL**	**OUT8VAL**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	1	1	1

Figure 15-106. Isochronous IN/OUT Endpoint Valid Bits Register

The 8051 sets VAL=1 for active endpoints, and VAL=0 for inactive endpoints. These bits instruct the USB core to return a “no response” if an invalid endpoint is addressed.

The default values of these registers are set to support all endpoints that exist in the default USB device (Table 5-1).

15.39 Fast Transfers

FASTXFR **Fast Transfer Control** **7FE2**

b7	b6	b5	b4	b3	b2	b1	b0
FISO	**FBLK**	**RPOL**	**RMOD1**	**RMOD0**	**WPOL**	**WMOD1**	**WMOD0**
R	R	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-107. Fast Transfer Control Register

The USB core provides a fast transfer mode designed for attaching external FIFOs to the isochronous and bulk endpoint buffers. The FASTXFR Register enables the modes for bulk and/or isochronous transfers, and selects the timing waveforms for the FRD# and FWR# signals.

Bit 7: **FISO** *Enable Fast ISO Transfers*

The 8051 sets FISO=1 to enable fast isochronous transfers for all16 isochronous endpoint FIFOs. When FISO=0, fast transfers are disabled for all 16 isochronous endpoints.

Bit 6: **FBLK** *Enable Fast BULK Transfers*

The 8051 sets FBLK=1 to enable fast bulk transfers using the Autopointer (see Section 15.40. *"SETUP Data"*) with BULK endpoints. When FBLK=0 fast transfers are disabled for BULK endpoints.

Bit 5: **RPOL** *FRD# Pulse Polarity*

The 8051 sets RPOL=0 for active-low FRD# pulses, and RPOL=1 for active high FRD# pulses.

Bit 4-3: **RMOD** *FRD# Pulse Mode*

These bits select the phasing and width of the FRD# pulse.

Bit 2: **WPOL** *FWR# Pulse Polarity*

The 8051 sets WPOL=0 for active-low FWR# pulses, and WPOL=1 for active high FWR# pulses.

Bit 1-0: **WMOD** *FWR# Pulse Mode*

These bits select the phasing and width of the FWR# pulse.

AUTOPTRH	Auto Pointer Address High	7FE3

b7	b6	b5	b4	b3	b2	b1	b0
A15	**A14**	**A13**	**A12**	**A11**	**A10**	**A9**	**A8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

AUTOPTRL	Auto Pointer Address Low	7FE4

b7	b6	b5	b4	b3	b2	b1	b0
A7	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

AUTODATA	Auto Pointer Data	7FE5

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

Figure 15-108. Auto Pointer Registers

These registers control the EZ-USB FX *Autopointer.*

15.39.1 AUTOPTRH/L

The 8051 loads a 16-bit address into the AUTOPTRH/L Registers. Subsequent reads or writes to the AUTODATA Register increment the 16-bit value in these registers. The loaded address must be in internal EZ-USB FX RAM. The 8051 can read these registers to determine the address of the next byte to be accessed via the AUTODATA Register.

15.39.2 AUTODATA

8051 data read or written to the AUTODATA Register accesses the memory addressed by the AUTOPTRH/L Registers, and increments the address *after* the read or write.

These registers allow FIFO access to the bulk endpoint buffers, as well as being useful for internal data movement. *Chapter 6. "EZ-USB FX Bulk Transfers"* and *Chapter 10. "EZ-USB FX Isochronous Transfers"* explain how to use the Autopointer for fast transfers to and from the EZ-USB FX endpoint buffers.

15.40 SETUP Data

SETUPBUF	SETUP Data Buffer (8 Bytes)	7FE8-7FEF

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

Figure 15-109. SETUP Data Buffer

This buffer contains the 8 bytes of SETUP packet data from the most recently received CONTROL transfer.

The data in SETUPBUF is valid when the SUDAVIR (Setup Data Available Interrupt Request) Bit is set.

15.41 Isochronous FIFO Sizes

OUTnADDR	ISO OUT Endpoint Start Address	7FF0-7FF7*

b7	b6	b5	b4	b3	b2	b1	b0
A9	**A8**	**A7**	**A6**	**A5**	**A4**	**0**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

INnADDR	ISO IN Endpoint Start Address	7FF8-7FFF*

b7	b6	b5	b4	b3	b2	b1	b0
A9	**A8**	**A7**	**A6**	**A5**	**A4**	**0**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

* See Table 15-17 for individual start address register addresses.

Figure 15-110. SETUP Data Buffer

Table 15-17. Isochronous FIFO Start Address Registers

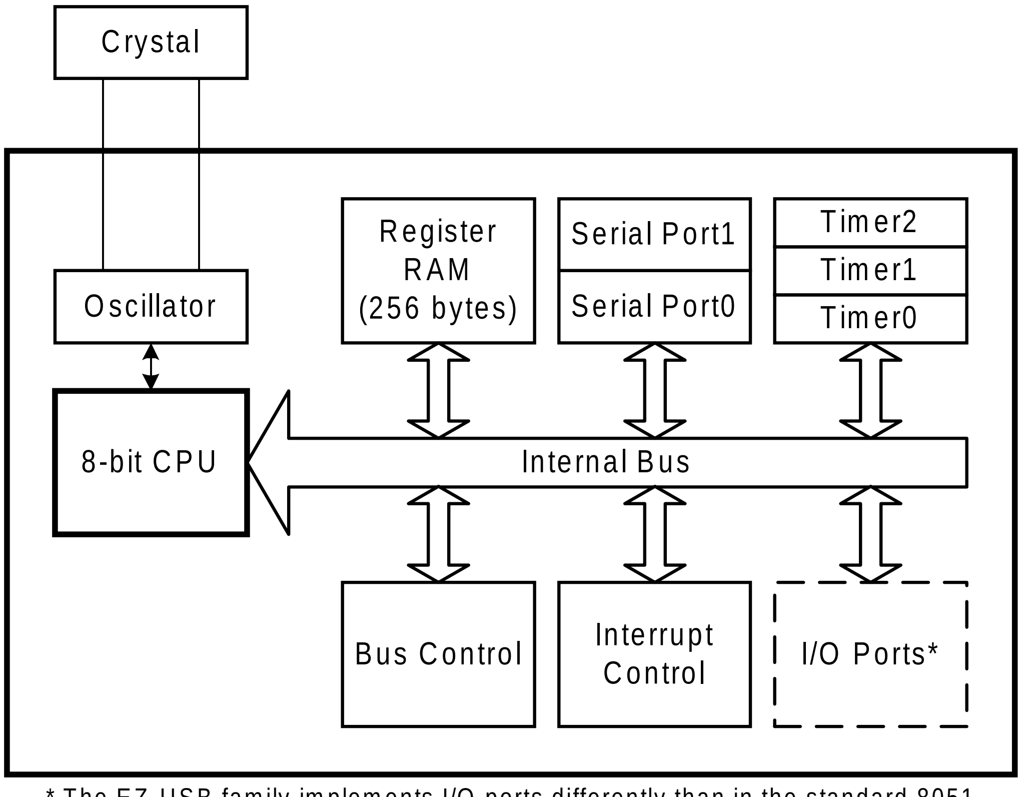
Address	Endpoint Start Address
7FF0	Endpoint 8 OUT Start Address
7FF1	Endpoint 9 OUT Start Address
7FF2	Endpoint 10 OUT Start Address
7FF3	Endpoint 11 OUT Start Address
7FF4	Endpoint 12 OUT Start Address
7FF5	Endpoint 13 OUT Start Address
7FF6	Endpoint 14 OUT Start Address
7FF7	Endpoint 15 OUT Start Address
7FF8	Endpoint 8 IN Start Address
7FF9	Endpoint 9 IN Start Address
7FFA	Endpoint 10 IN Start Address
7FFB	Endpoint 11 IN Start Address
7FFC	Endpoint 12 IN Start Address
7FFD	Endpoint 13 IN Start Address
7FFE	Endpoint 14 IN Start Address
7FFF	Endpoint 15 IN Start Address

EZ-USB FX Isochronous endpoints use a pool of 1,024 double-buffered FIFO bytes. The 1,024 FIFO bytes can be divided between any or all of the isochronous endpoints. The 8051 sets isochronous endpoint FIFO sizes by writing starting addresses to these registers, starting with address 0. Address bits A3-A0 are internally set to zero, so the minimum FIFO size is 16 bytes.

Chapter 16. 8051 Introduction

16.1 Introduction

The EZ-USB FX contains an 8051 core that is binary-compatible with the industry standard 8051 instruction set.



* The EZ-USB family implements I/O ports differently than in the standard 8051

Figure 16-1. 8051 Features

This chapter provides an overview of the 8051 core features. The topics are:

- New 8051 Features
- Performance Overview
- Software Compatibility
- 803x/805x Feature Comparison
- 8051/DS80C320 Differences.

16.2 8051 Features

The 8051 core provides the following design features and enhancements to the standard 8051 micro-controller:

- Compatible with industry standard 803x/805x:
  - Standard 8051 instruction set
  - Two full-duplex serial ports
  - Three timers
- High-speed architecture:
  - 4 clocks/instruction cycle
  - 2.5X average improvement in instruction execution time over the standard 8051
  - Wasted bus cycles eliminated
  - Dual data pointers
- 256 Bytes internal data RAM
- High-speed external memory interface with 16-bit address bus
- Variable length `MOVX` to access fast/slow RAM peripherals
- Supports industry standard compilers, assemblers, emulators, and ROM monitors

16.3 Performance Overview

The 8051 core has been designed to offer increased performance by executing instructions in a 4-clock bus cycle, as opposed to the 12-clock bus cycle in the standard 8051. (See *Figure 16-2*). The shortened bus timing improves the instruction execution rate for most instructions by a factor of three over the standard 8051 architectures.

Some instructions require a different number of instruction cycles on the 8051 core than they do on the standard 8051. In the standard 8051, all instructions except for `MUL` and `DIV` take one or two instruction cycles to complete. In the 8051 core, instructions can take between one and five instruction cycles to complete. The average speed improvement for the entire instruction set is approximately 2.5X. Table 16-1 catalogs the speed improvements.

Table 16-1. 8051/Standard 8051 Speed Comparison

Number of Opcodes	Speed Improvement
150	3.0X
51	1.5X
43	2.0X
2	2.4X
Total: 255	Average: 2.5X
Note: Comparison is for 8051 and standard 8051 running at the same clock frequency.	

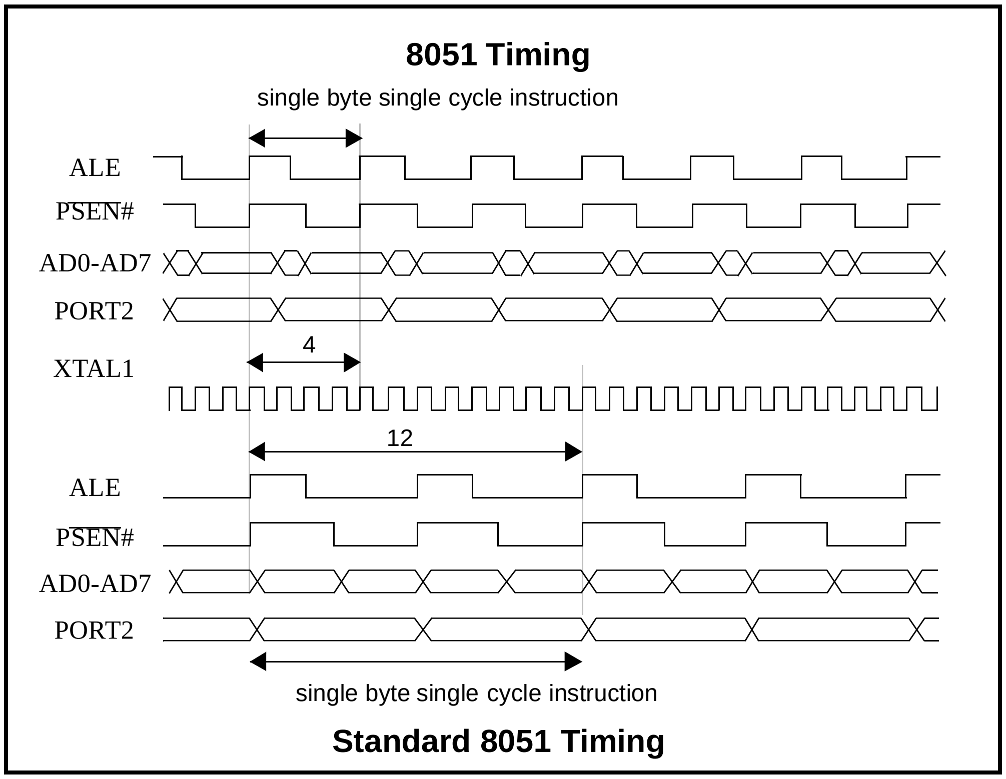


Figure 16-2. 8051/Standard 8051 Timing Comparison

16.4 Software Compatibility

The 8051 core is object code compatible with the industry standard 8051 micro-controller. That is, object code compiled with an industry standard 8051 compiler or assembler executes on the 8051 core and is functionally equivalent. However, because the 8051 core uses a different instruction timing than the standard 8051, existing code with timing loops may require modification.

The "Instruction Set" in Table 16-2 lists the number of instruction cycles required to perform each instruction on the 8051 core. The 8051 instruction cycle timing and number of instruction cycles required for each instruction are compatible with the Dallas Semiconductor DS80C320.

16.5 803x/805x Feature Comparison

Table 16-2 provides a feature-by-feature comparison of the 8051 core and several common 803x/ 805x configurations.

Table 16-2. Features of 8051 Core & Common 803x/805x Configurations

Feature	Intel				Dallas DS80C320	Anchor 8051
	8031	8051	80C32	80C52		
Clocks per instruction cycle	12	12	12	12	4	4
Program / Data Memory	-	4 KB ROM	-	8 KB ROM	-	8 K RAM
Internal RAM	128 bytes	128 bytes	256 bytes	256 bytes	256 bytes	256 bytes
Data Pointers	1	1	1	1	2	2
Serial Ports	1	1	1	1	2	2
16-bit Timers	2	2	3	3	3	3
Interrupt sources (total of int. and ext.)	5	5	6	6	13	13
Stretch memory cycles	no	no	no	no	yes	yes

16.6 8051 Core/DS80C320 Differences

The 8051 core is similar to the DS80C320 in terms of hardware features and instruction cycle timing. However, there are some important differences between the 8051 core and the DS80C320.

16.6.1 Serial Ports

The 8051 core does not implement serial port framing error detection and does not implement slave address comparison for multiprocessor communications. Therefore, the 8051 core also does not implement the following SFRs: SADDR0, SADDR1, SADEN0, and SADEN1.

16.6.2 Timer 2

The 8051 core does not implement Timer 2 downcounting mode or the downcount enable bit (TMOD2, Bit 0). Also, the 8051 core does not implement Timer 2 output enable (T2OE) bit (TMOD2, Bit 1). Therefore, the TMOD2 SFR is also not implemented in the 8051 core.

Also, the 8051 core Timer 2 overflow output is active for one clock cycle. In the DS80C320, the Timer 2 overflow output is a square wave with a 50% duty cycle.

It is possible to float the T2OUT pin by setting OEB.7=0 and PORTBCFG.7=0. This selects the PORTB (not T2OUT) signal, and turns off its output buffer.

16.6.3 Timed Access Protection

The 8051 core does not implement timed access protection and therefore, does not implement the TA SFR.

16.6.4 Watchdog Timer

The EZ-USB FX/8051 does not implement a watchdog timer. It also does not implement I/O ports 0-3. Instead, it uses ports A-E.

Chapter 17. 8051 Architectural Overview

17.1 Introduction

This chapter provides a technical overview and description of the 8051 core architecture.

17.1.1 Memory Organization

Memory organization in the 8051 core is similar to that of the industry standard 8051. There are three distinct memory areas: registers, program memory, and data memory.

17.1.1.1 Registers

Register memory is implemented inside the 8051 core. The 8051 accesses registers in two regions using direct addressing, providing the fastest available 8051 data access. The two directly addressable regions are 128 general purpose registers at addresses 00-7F, and 128 bytes of Special Function Registers (SFRs) at 80-FF. The SFR address space, which is not fully populated, contains 8051 control and status registers, plus added EZ-USB FX control and status registers.

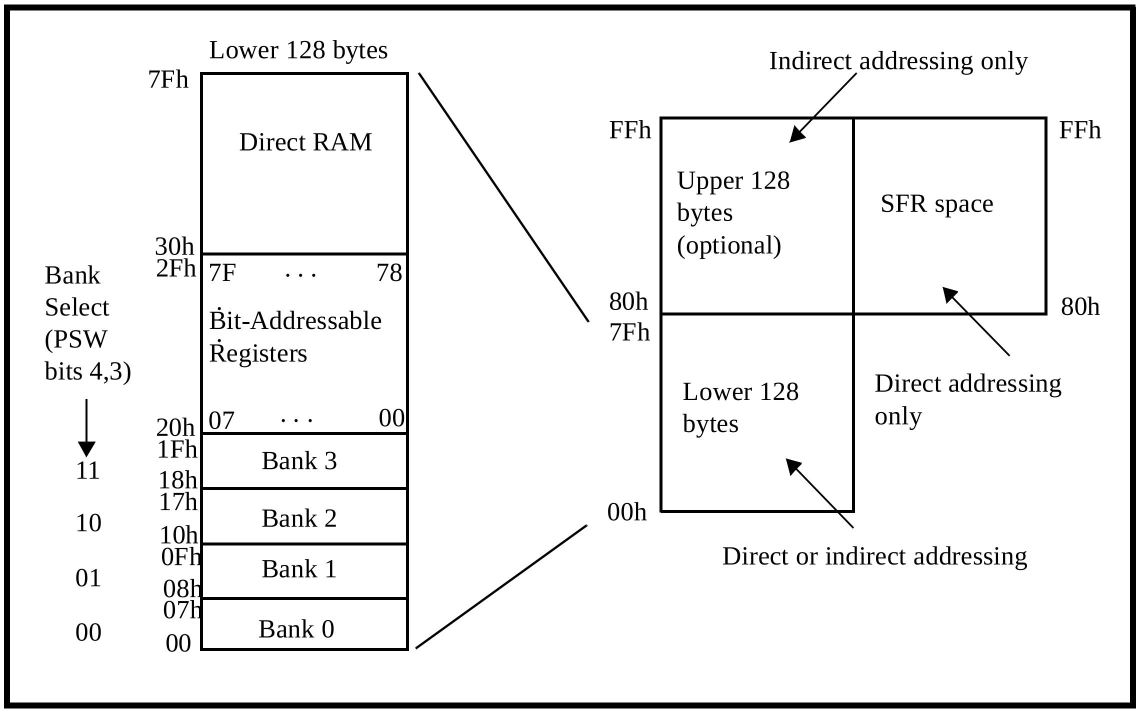


Figure 17-1. Internal RAM Organization

Some examples of direct addressing are:

- MOV A,22H ; load accumulator from register at address 22
- MOV A,IOE ; read the EZ-USB FX PORTE pins (added SFR)
- MOV IOD,A ; write the PORTD Bits (added SFR)

An additional 128 registers overlap the SFRs at addresses 80-FF. The 8051 keeps these separate from the SFRs by using a different addressing mode, 8-bit indirect, to access them. For example, to read the register at location 90(hex):

- MOV R0,#90H ;point to register RAM at 90(hex)
- MOV A,@R0 ;read it using 8-bit indirect addressing

The 8051 uses two registers, R0 and R1, to hold the 8-bit index. This addressing mode may also access register memory from 0-127, although it is faster and more efficient to use the direct addressing available in this lower region.

Since the 8051 stack is internally accessed using indirect addressing, it is a good idea to put the stack in the upper 128 bytes of register memory, which is addressable using indirect addressing only. This frees the lower 128 register bytes for use by the more efficient direct addressing.

17.1.1.2 Program Memory

The 8051 has separate address spaces for program and data memory. Program memory can only be read, not written. The read strobe for program memory is PSEN (Program Store Enable). The 8051 generates PSEN strobes for two conditions, instruction fetches and the MOVC (move code memory into the accumulator) instruction.

17.1.1.3 Data Memory

Data memory occupies a separate address space from program memory. Data memory can be read or written, using the RD and WR strobes. Up to 64 KB of data memory can be added to the EZ-USB FX versions that bring out the 8051 address and data bus pins. As the next section explains, a portion of this *external* data memory is actually implemented inside the FX chip.

17.1.1.4 EZ-USB FX Program/Data Memory

The EZ-USB FX family contains internal RAM, which in most systems provides all the memory for a single-chip solution. Therefore, this internal RAM must serve both as 8051 program and data memory. To accomplish this, the 8051 reads internal RAM using the logical OR of the PSEN and RD strobes. It is the responsibility of the system designer to ensure that the program and data memory spaces do not overlap. This is done using linker directives that place the code and data modules.

It is possible to add external program and data memory to the EZ-USB FX parts that provide the 8051 address and data bus pins. To avoid conflict with the internal combined program/data memory, the EZ-USB FX logic gates the memory strobes to be inactive when the 8051 accesses internal memory. These strobes include the RD#, WR#, CS#, and OE# pins. Because of this internal gating, a 64-KB memory (data and/or program) can be added without requiring external logic to inhibit access to the bottom 8 KB that are inside the FX part. Note that the PSEN and RD signals are available on separate pins, so the program and data spaces are not combined as they are inside the FX part.

The EA (external access) pin allows *all* external memory to be program memory. When EA is tied high, the 8051 reads the internal RAM using only the RD strobe—the combining of RD and PSEN is disabled. With EA=1, the internal RAM becomes data memory only, and program memory starts at 0000 in external memory. The other effect of tying the EA pin high is that the 8051 powers up running (not in RESET), ready to run the external code.

17.1.1.5 Accessing Data Memory

The 8051 reads and writes data memory using the MOVX instruction. Either an 8-bit or a 16-bit index (address pointer) can be used. 8-bit addressing uses either R0 or R1 to supply the lower address byte, and the MPAGE Register (SFR address 92H) to supply the high address byte. 16-bit addressing uses the 16-bit data pointer (DPTR) to supply the full address.

EZ-USB FX registers exist in the upper portion of internal memory. The 8051 accesses them using the MOVX instruction. A limited set of EZ-USB FX registers are in the SFR address space to provide fastest possible access.

17.1.2 Instruction Set

All 8051 instructions are binary code compatible and perform the same functions as they do with the industry standard 8051. The effects of these instructions on bits, flags, and other status functions is identical to the industry standard 8051. However, the timing of the instructions is different, both in terms of number of clock cycles per instruction cycle and timing within the instruction cycle.

Table 17-2 lists the 8051 instruction set and the number of instruction cycles required to complete each instruction. Table 17-1 defines the symbols and mnemonics used in Table 17-2.

Table 17-1. Legend for Instruction Set Table

Symbol	Function
A	Accumulator
Rn	Register R7–R0
direct	Internal register address
@Ri	Internal register pointed to by R0 or R1
rel	Two’s complement offset byte
bit	Direct bit address
#data	8-bit constant
#data 16	16-bit constant
addr 16	16-bit destination address
addr 11	11-bit destination address

Table 17-2. 8051 Instruction Set

Mnemonic	Description	Byte	Instr. Cycles	Hex Code
Arithmetic				
ADD A, Rn	Add register to A	1	1	28-2F
ADD A, direct	Add direct byte to A	2	2	25
ADD A, @Ri	Add data memory to A	1	1	26-27
ADDC A, #data	Add immediate to A	2	2	24
ADDC A, Rn	Add register to A with carry	1	1	38-3F
ADDC A, direct	Add direct byte to A with carry	2	2	35
ADDC A, @Ri	Add data memory to A with carry	1	1	36-37
ADDC A, #data	Add immediate to A with carry	2	2	34
SUBB A, Rn	Subtract register from A with borrow	1	1	98-9F
SUBB A, direct	Subtract direct byte from A with borrow	2	2	95
SUBB A, @Ri	Subtract data memory from A with borrow	1	1	96-97
SUBB A, #data	Subtract immediate from A with borrow	2	2	94
INC A	increment A	1	1	04
INC Rn	Increment register	1	1	08-0F
INC direct	Increment direct byte	2	2	05
INC @ Ri	Increment data memory	1	1	06-07
DEC A	Decrement A	1	1	14
DEC Rn	Decrement Register	1	1	18-1F
DEC direct	Decrement direct byte	2	2	15
DEC @Ri	Decrement data memory	1	1	16-17
INC DPTR	Increment data pointer	1	3	A3
MUL AB	Multiply A by B	1	5	A4
DIV AB	Divide A by B	1	5	84
DA A	Decimal adjust A	1	1	D4
Logical				
ANL, Rn	AND register to A	1	1	58-5F
ANL A, direct	AND direct byte to A	2	2	55
ANL A, @Ri	AND data memory to A	1	1	56-57
ANL A, #data	AND immediate to A	2	2	54
ANL direct, A	AND A to direct byte	2	2	52
ANL direct, #data	AND immediate data to direct byte	3	3	53
ORL A, Rn	OR register to A	1	1	48-4F
ORL A, direct	OR direct byte to A	2	2	45
ORL A, @Ri	OR data memory to A	1	1	46-47
ORL A, #data	OR immediate to A	2	2	44
ORL direct, A	OR A to direct byte	2	2	42

Table 17-2. 8051 Instruction Set

Mnemonic	Description	Byte	Instr. Cycles	Hex Code
ORL direct, #data	OR immediate data to direct byte	3	3	43
XORL A, Rn	Exclusive-OR register to A	1	1	68-6F
XORL A, direct	Exclusive-OR direct byte to A	2	2	65
XORL A, @Ri	Exclusive-OR data memory to A	1	1	66-67
XORL A, #data	Exclusive-OR immediate to A	2	2	64
XORL direct, A	Exclusive-OR A to direct byte	2	2	62
XORL direct, #data	Exclusive-OR immediate data to direct byte	3	3	63
CLR A	Clear A	1	1	E4
CPL A	Complement A	1	1	F4
SWAP A	Swap nibbles of a	1	1	C4
RL A	Rotate A left	1	1	23
RLC A	Rotate A left through carry	1	1	33
RRA	Rotate A right	1	1	03
RRC A	Rotate A right through carry	1	1	13
Data Transfer				
MOV A, Rn	Move register to A	1	1	E8-EF
MOV A, direct	Move direct byte to A	2	2	E5
MOV A, @Ri	Move data memory to A	1	1	E6-E7
MOV A, #data	Move immediate to A	2	2	74
MOV Rn, A	Move A to register	1	1	F8-FF
MOV Rn, direct	Move direct byte to register	2	2	A8-AF
MOV Rn, #data	Move immediate to register	2	2	78-7F
MOV direct, A	Move A to direct byte	2	2	F5
MOV direct, Rn	Move register to direct byte	2	2	88-8F
MOV direct, direct	Move direct byte to direct byte	3	3	85
MOV direct, @Ri	Move data memory to direct byte	2	2	86-87
MOV direct, #data	Move immediate to direct byte	3	3	75
MOV @Ri, A	MOV A to data memory	1	1	F6-F7
MOV @Ri, direct	Move direct byte to data memory	2	2	A6-A7
MOV @Ri, #data	Move immediate to data memory	2	2	76-77
MOV DPTR, #data	Move immediate to data pointer	3	3	90
MOVC A, @A+DPTR	Move code byte relative DPTR to A	1	3	93
MOVC A, @A+PC	Move code byte relative PC to A	1	3	83
MOVX A, @Ri	Move external data (A8) to A	1	2-9*	E2-E3
MOVX A, @DPTR	Move external data (A16) to A	1	2-9*	E0
MOVX @Ri, A	Move A to external data (A8)	1	2-9*	F2-F3
MOVX @DPTR, A	Move A to external data (A16)	1	2-9*	F0
PUSH direct	Push direct byte onto stack	2	2	C0

Table 17-2. 8051 Instruction Set

Mnemonic	Description	Byte	Instr. Cycles	Hex Code
POP direct	Pop direct byte from stack	2	2	D0
XCH A, Rn	Exchange A and register	1	1	C8-CF
XCH A, direct	Exchange A and direct byte	2	2	C5
XCH A, @Ri	Exchange A and data memory	1	1	C6-C7
XCHD A, @Ri	Exchange A and data memory nibble	1	1	D6-D7
** Number of cycles is user-selectable. See Section 17.1.5. "Stretch Memory Cycles (Wait States)".*				
Boolean				
CLR C	Clear carry	1	1	C3
CLR bit	Clear direct bit	2	2	C2
SETB C	Set carry	1	1	D3
SETB bit	Set direct bit	2	2	D2
CPL C	Complement carry	1	1	B3
CPL bit	Complement direct bit	2	2	B2
ANL C, bit	AND direct bit to carry	2	2	82
ANL C, /bit	AND direct bit inverse to carry	2	2	B0
ORL C, bit	OR direct bit to carry	2	2	72
ORL C, /bit	OR direct bit inverse to carry	2	2	A0
MOV C, bit	Move direct bit to carry	2	2	A2
MOV bit, C	Move carry to direct bit	2	2	92
Branching				
ACALL addr 11	Absolute call to subroutine	2	3	11-F1
LCALL addr 16	Long call to subroutine	3	4	12
RET	Return from subroutine	1	4	22
RETI	Return from interrupt	1	4	32
AJMP addr 11	Absolute jump unconditional	2	3	01-E1
LJMP addr 16	Long jump unconditional	3	4	02
SJMP rel	Short jump (relative address)	2	3	80
JC rel	Jump on carry = 1	2	3	40
JNC rel	Jump on carry = 0	2	3	50
JB bit, rel	Jump on direct bit = 1	3	4	20
JNB bit, rel	Jump on direct bit = 0	3	4	30
JBC bit, rel	Jump on direct bit = 1 and clear	3	4	10
JMP @ A+DPTR	Jump indirect relative DPTR	1	3	73
JZ rel	Jump on accumulator = 0	2	3	60
JNZ rel	Jump on accumulator /= 0	2	3	70
CJNE A, direct, rel	Compare A, direct JNE relative	3	4	B5
CJNE A, #d, rel	Compare A, immediate JNE relative	3	4	B4
CJNE Rn, #d, rel	Compare reg, immediate JNE relative	3	4	B8-BF

Table 17-2. 8051 Instruction Set

Mnemonic	Description	Byte	Instr. Cycles	Hex Code
CJNE @ Ri, #d, rel	Compare Ind, immediate JNE relative	3	4	B6-B7
DJNZ Rn, rel	Decrement register, JNZ relative	2	3	D8-DF
DJNZ direct, rel	Decrement direct byte, JNZ relative	3	4	D5
Miscellaneous				
NOP	No operation	1	1	00
There is an additional reserved opcode (A5) that performs the same function as NOP. All mnemonics are copyrighted. Intel Corporation 1980.				

17.1.3 Instruction Timing

Instruction cycles in the 8051 core are 4 clock cycles in length, as opposed to the 12 clock cycles per instruction cycle in the standard 8051. This translates to a 3X improvement in execution time for most instructions.

Some instructions require a different number of instruction cycles on the 8051 core than they do on the standard 8051. In the standard 8051, all instructions except for `MUL` and `DIV` take one or two instruction cycles to complete. In the 8051 core, instructions can take between one and five instruction cycles to complete.

For example, in the standard 8051, the instructions `MOVX A, @DPTR` and `MOV direct, direct` each take 2 instruction cycles (24 clock cycles) to execute. In the 8051 core, `MOVX A, @DPTR` takes two instruction cycles (8 clock cycles) and `MOV direct, direct` takes three instruction cycles (12 clock cycles). Both instructions execute faster on the 8051 core than they do on the standard 8051, but require different numbers of clock cycles.

For timing of real-time events, use the numbers of instruction cycles from Table 17-2 to calculate the timing of software loops. The bytes column indicates the number of memory accesses (bytes) needed to execute the instruction. In most cases, the number of bytes is equal to the number of instruction cycles required to complete the instruction. However, as indicated, there are some instructions (for example, `DIV` and `MUL`) that require a greater number of instruction cycles than memory accesses.

By default, the 8051 core timer/counters run at 12 clock cycles per increment so that timer-based events have the same timing as with the standard 8051. The timers can also be configured to run at 4 clock cycles per increment to take advantage of the higher speed of the 8051 core.

17.1.4 CPU Timing

As previously stated, an 8051 core instruction cycle consists of 4 *CLKOUT* cycles. Each *CLKOUT* cycle forms a CPU cycle. Therefore, an instruction cycle consists of 4 CPU cycles: C1, C2, C3, and C4, as illustrated in *Figure 17-2*. Various events occur in each CPU cycle, depending on the type of instruction being executed. The labels C1, C2, C3, and C4 in timing descriptions refer to the 4 CPU cycles within a particular instruction cycle.

The execution for instruction *n* is performed during the fetch of instruction *n*+1. Data writes occur during fetch of instruction *n*+2. The level sensitive interrupts are sampled with the rising edge of *CLKOUT* at the end of C3.

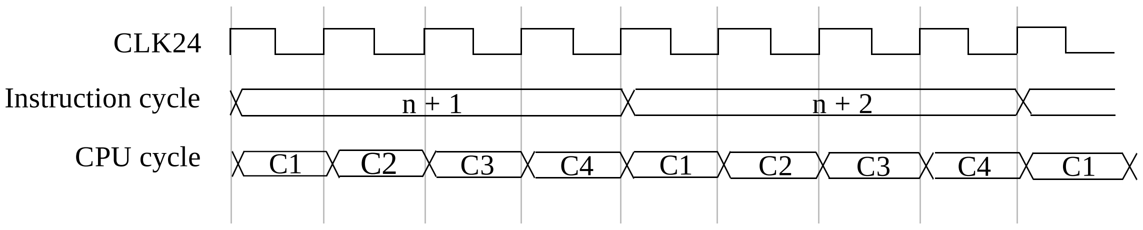


Figure 17-2. CPU Timing for Single-Cycle Instruction

17.1.5 Stretch Memory Cycles (Wait States)

The stretch memory cycle feature enables application software to adjust the speed of *data memory* (not code memory) access. The 8051 core can execute the MOVX instruction in as few as 2 instruction cycles. However, it is sometimes desirable to stretch this value; for example to access slow memory or slow memory-mapped peripherals such as UARTs or LCDs.

The three LSBs of the Clock Control Register (at SFR location 8Eh) control the stretch value. You can use stretch values between zero and seven. A stretch value of zero adds zero instruction cycles, resulting in `MOVX` instructions executing in two instruction cycles. A stretch value of seven adds seven instruction cycles, resulting in `MOVX` instructions executing in nine instruction cycles. The stretch value can be changed dynamically under program control.

By default, the stretch value resets to one (three cycle `MOVX`). For full-speed data memory access, the software must set the stretch value to zero. The stretch value affects only data memory access (<u>not</u> program memory).

The stretch value affects the width of the read/write strobe and all related timing. Using a higher stretch value results in a wider read/write strobe, which allows the memory or peripheral more time to respond.

Table 17-3 lists the data memory access speeds for stretch values zero through seven. MD2–0 are the three LSBs of the Clock Control Register (CKCON.2–0).

Table 17-3. Data Memory Stretch Values

MD2	MD1	MD0	Memory Cycles	Read/Write Strobe Width (Clocks)	Strobe Width @ 24MHz	Strobe Width @ 48MHz
0	0	0	2	2	83.3 ns	41.65 ns
0	0	1	3 (default)	4	166.7 ns	83.35 ns
0	1	0	4	8	333.3 ns	166.66 ns
0	1	1	5	12	500 ns	250 ns
1	0	0	6	16	666.7 ns	333.35 ns
1	0	1	7	20	833.3 ns	416.65 ns
1	1	0	8	24	1000 ns	500 ns
1	1	1	9	28	1166.7 ns	583.35 ns

17.1.6 Dual Data Pointers

The 8051core employs dual data pointers to accelerate data memory block moves. The standard 8051 data pointer (DPTR) is a 16-bit value used to address external data RAM or peripherals. The 8051 maintains the standard data pointer as DPTR0 at SFR locations 82h (DPL0) and 83h (DPH0). It is not necessary to modify existing code to use DPTR0.

The 8051 core adds a second data pointer (DPTR1) at SFR locations 84h (DPL1) and 85h (DPH1). The SEL Bit in the DPTR Select Register, DPS (SFR 86h), selects the active pointer. When SEL = 0, instructions that use the DPTR will use DPL0 and DPH0. When SEL = 1, instructions that use the DPTR will use DPL1 and DPH1. SEL is the bit 0 of SFR location 86h. No other bits of SFR location 86h are used.

All DPTR-related instructions use the data pointer selected by the SEL Bit. To switch the active pointer, toggle the SEL Bit. The fastest way to do so is to use the increment instruction (`INC DPS`). This requires only one instruction to switch from a source address to a destination address, saving application code from having to save source and destination addresses when doing a block move.

Using dual data pointers provides significantly increased efficiency when moving large blocks of data.

The SFR locations related to the dual data pointers are:

82h	DPL0	DPTR0 low byte
83h	DPH0	DPTR0 high byte
84h	DPL1	DPTR1 low byte
85h	DPH1	DPTR1 high byte
86h	DPS	DPTR Select (Bit 0)

17.1.7 Special Function Registers

The Special Function Registers (SFRs) control several of the features of the 8051 and EZ-USB FX. Most of the 8051 core SFRs are identical to the standard 8051 SFRs. However, there are additional SFRs that control features that are not available in the standard 8051, plus some EZ-USB FX features.

Table 17-4 lists the 8051 core SFRs and indicates which SFRs are not included in the standard 8051 SFR space. See Section 4.12. *"SFR Addressing"* for the EZ-USB FX added SFRs.

In Table 17-5, SFR Bit positions that contain a 0 or a 1 cannot be written to and, when read, always return the value shown (0 or 1). SFR Bit positions that contain “-” are available but not used. Table 17-5 lists the reset values for the SFRs.

The following SFRs are related to CPU operation and program execution:

81h	SP	Stack Pointer
D0h	PSW	Program Status Word ()
E0h	ACC	Accumulator Register
F0h	B	B Register

Table 17-6 lists the functions of the bits in the PSW SFR. Detailed descriptions of the remaining SFRs appear with the associated hardware descriptions in *Chapter 4. "EZ-USB FX Input/Output"* of this manual.

Table 17-4. Special Function Registers

Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Addr
SP									81h
DPL0									82h
DPH0									83h
DPL1(1)									84h
DPH1(1)									85h
DPS(1)	0	0	0	0	0	0	0	SEL	86h
PCON	SMOD0	-	1	1	GF1	GF0	STOP	IDLE	87h
TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	88h
TMOD	GATE	C/T	M1	M0	GATE	$C/\overline{T}$	M1	M0	89h
TL0									8Ah
TL1									8Bh
TH0									8Ch
TH1									8Dh
CKCON(1)	-	-	T2M	T1M	T0M	MD2	MD1	MD0	8Eh
SPC_FNC(1)	0	0	0	0	0	0	0	WRS	8Fh
EXIF(1)	IE5	IE4	I2CINT	USBINT	1	0	0	0	91h
MPAGE(1)									92h
SCON0	SM0_0	SM1_0	SM2_0	REN_0	TB8_0	RB8_0	TI_0	RI_0	98h
SBUF0									99h
IE	EA	ES1	ET2	ES0	ET1	EX1	ET0	EX0	A8h
IP	1	PS1	PT2	PS0	PT1	PX1	PT0	PX0	B8h
SCON1(1)	SM0_1	SM1_1	SM2_1	REN_1	TB8_1	RB8_1	TI_1	RI_1	C0h
SBUF1(1)									C1h
T2CON	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	$C/\overline{T2}$	$CP/\overline{RL2}$	C8h
RCAP2L									CAh
RCAP2H									CBh
TL2									CCh
TH2									CDh
PSW	CY	AC	F0	RS1	RS0	OV	F1	P	D0h
EICON(1)	SMOD1	1	ERESI	RESI	INT6	0	0	0	D8h
ACC									E0H
EIE(1)	1	1	1	EWDI	EX5	EX4	EI2C	EUSB	E8h
B									F0h
EIP(1)	1	1	1	PX6	PX5	PX4	PI2C	PUSB	F8h
(1) Not part of standard 8051 architecture.									

Table 17-5. Special Function Register Reset Values

Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Addr
SP	0	0	0	0	0	1	1	1	81h
DPL0	0	0	0	0	0	0	0	0	82h
DPH0	0	0	0	0	0	0	0	0	83h
DPL1(1)	0	0	0	0	0	0	0	0	84h
DPH1(1)	0	0	0	0	0	0	0	0	85h
DPS(1)	0	0	0	0	0	0	0	0	86h
PCON	0	0	1	1	0	0	0	0	87h
TCON	0	0	0	0	0	0	0	0	88h
TMOD	0	0	0	0	0	0	0	0	89h
TL0	0	0	0	0	0	0	0	0	8Ah
TL1	0	0	0	0	0	0	0	0	8Bh
TH0	0	0	0	0	0	0	0	0	8Ch
TH1	0	0	0	0	0	0	0	0	8Dh
CKCON(1)	0	0	0	0	0	0	0	1	8Eh
SPC_FNC(1)	0	0	0	0	0	0	0	0	8Fh
EXIF(1)	0	0	0	0	1	0	0	0	91h
MPAGE(1)	0	0	0	0	0	0	0	0	92h
SCON0	0	0	0	0	0	0	0	0	98h
SBUF0	0	0	0	0	0	0	0	0	99h
IE	0	0	0	0	0	0	0	0	A8h
IP	1	0	0	0	0	0	0	0	B8h
SCON1(1)	0	0	0	0	0	0	0	0	C0h
SBUF1(1)	0	0	0	0	0	0	0	0	C1h
T2CON	0	0	0	0	0	0	0	0	C8h
RCAP2L	0	0	0	0	0	0	0	0	CAh
RCAP2H	0	0	0	0	0	0	0	0	CBh
TL2	0	0	0	0	0	0	0	0	CCh
TH2	0	0	0	0	0	0	0	0	CDh
PSW	0	0	0	0	0	0	0	0	D0h
EICON(1)	0	1	0	0	0	0	0	0	D8h
ACC	0	0	0	0	0	0	0	0	E0H
EIE(1)	1	1	1	0	0	0	0	0	E8h
B	0	0	0	0	0	0	0	0	F0h
EIP(1)	1	1	1	0	0	0	0	0	F8h

(1) Not part of standard 8051 architecture.

Table 17-6. PSW Register - SFR D0h

Bit	Function
PSW.7	**CY** - Carry flag. This is the **unsigned** carry bit. The CY flag is set when an arithmetic operation results in a carry from bit 7 to bit 8, and cleared otherwise. In other words, it acts as a virtual bit 8. The CY flag is cleared on multiplication and division.
PSW.6	**AC** - Auxiliary carry flag. Set to 1 when the last arithmetic operation resulted in a carry into (during addition) or borrow from (during subtraction) the high order nibble, otherwise cleared to 0 by all arithmetic operations.
PSW.5	**F0** - User flag 0. Bit-addressable, general purpose flag for software control.
PSW.4	**RS1** - Register bank select bit 1. used with RS0 to select a register bank in internal RAM.
PSW.3	**RS0** - Register bank select bit 0, decoded as: <u>RS1RS0</u> <u>Banks Selected</u> 00 Register bank 0, addresses 00h-07h 01 Register bank 1, addresses 08h-0Fh 10 Register bank 2, addresses 10h-17h 11 Register bank 3, addresses 18h-1Fh
PSW.2	**OV** - Overflow flag. This is the **signed** carry bit. The OV flag is set when a positive sum exceeds 7fh, or a negative sum (in two's compliment notation) exceeds 80h. On a multiply, if OV = 1, the result of the multiply is greater than FFh. On a divide, OV = 1 on a divide by 0.
PSW.1	**F1** - User flag 1. Bit-addressable, general purpose flag for software control.
PSW.0	**P** - Parity flag. Set to 1 when the modulo-2 sum of the 8 bits in the accumulator is 1 (odd parity), cleared to 0 on even parity.

Chapter 18. 8051 Hardware Description

18.1 Introduction

This chapter provides technical data about the 8051 core hardware operation and timing. The topics are:

- Timers/Counters
- Serial Interface
- Interrupts
- 8051 Reset
- Power Saving Modes.

18.2 Timers/Counters

The 8051 core includes three timer/counters (Timer 0, Timer 1, and Timer 2). Each timer/counter can operate as either a timer with a clock rate based on the *CLKOUT* pin or as an event counter clocked by the *T0* pin (Timer 0), *T1* pin (Timer 1), or the *T2* pin (Timer 2).

Each timer/counter consists of a 16-bit register that is accessible to software as two SFRs:

- Timer 0 — TL0 and TH0
- Timer 1 — TL1 and TH1
- Timer 2 — TL2 and TH2.

18.2.1 803x/805x Compatibility

The implementation of the timers/counters is similar to that of the Dallas Semiconductor DS80C320. Table 18-7 summarizes the differences in timer/counter implementation between the Intel 8051, the Dallas Semiconductor DS80C320, and the 8051 core.

Table 18-7. Timer/Counter Implementation Comparison

Feature	Intel 8051	Dallas DS80C320	8051
Number of timers	2	3	3
Timer 0/1 overflow available as output signals	not implemented	not implemented	T0OUT, T1OUT (one CLKOUT pulse)
Timer 2 output enable	n/a	implemented	not directly implemented
Timer 2 downcount enable	n/a	implemented	not implemented
Timer 2 overflow available as output signal	n/a	implemented	T2OUT (one CLKOUT pulse)

18.2.2 Timers 0 and 1

Timers 0 and 1 each operate in four modes, as controlled through the TMOD SFR (Table 18-8) and the TCON SFR (Table 18-9). The four modes are:

- 13-bit timer/counter (mode 0)
- 16-bit timer/counter (mode 1)
- 8-bit counter with auto-reload (mode 2)
- Two 8-bit counters (mode 3, Timer 0 only)

18.2.2.1 Mode 0

Mode 0 operation, illustrated in *Figure 18-3*, is the same for Timer 0 and Timer 1. In mode 0, the timer is configured as a 13-bit counter that uses bits 0-4 of TL0 (or TL1) and all 8 bits of TH0 (or TH1). The timer enable bit (TR0/TR1) in the TCON SFR starts the timer. The C/$\overline{T}$ Bit selects the timer/counter clock source, CLKOUT or the T0/T1 pins.

The timer counts transitions from the selected source as long as the GATE Bit is 0, or the GATE Bit is 1 and the corresponding interrupt pin (INT0# or INT1#) is 1.

When the 13-bit count increments from 1FFFh (all ones), the counter rolls over to all zeros, the TF0 (or TF1) Bit is set in the TCON SFR, and the T0OUT (or T1OUT) pin goes high for one clock cycle.

The upper 3 bits of TL0 (or TL1) are indeterminate in mode 0 and must be masked when the software evaluates the register.

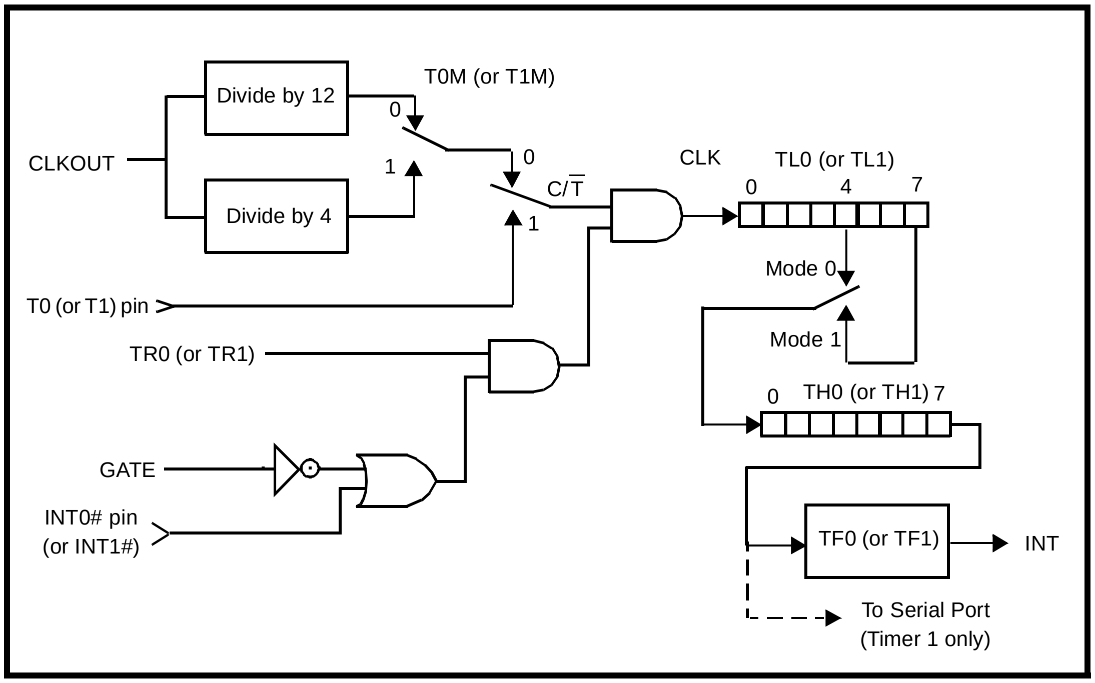


Figure 18-3. Timer 0/1 - Modes 0 and 1

18.2.2.2 Mode 1

Mode 1 operation is the same for Timer 0 and Timer 1. In mode 1, the timer is configured as a 16-bit counter. As illustrated in *Figure 18-3*, all 8 bits of the LSB Register (TL0 or TL1) are used. The counter rolls over to all zeros when the count increments from FFFFh. Otherwise, mode 1 operation is the same as mode 0.

Table 18-8. TMOD Register — SFR 89h

Bit	Function
TMOD.7	**GATE** - Timer 1 gate control. When GATE = 1, Timer 1 will clock only when INT1# = 1 and TR1 (TCON.6) = 1. When GATE = 0, Timer 1 will clock only when TR1 = 1, regardless of the state of INT1#.
TMOD.6	**C/$\overline{\text{T}}$** - Counter/Timer select. When C/$\overline{\text{T}}$ = 0, Timer 1 is clocked by CLKOUT/4 or CLKOUT/12, depending on the state of T1M (CKCON.4). When C/$\overline{\text{T}}$ = 1, Timer 1 is clocked by the T1 pin.
TMOD.5	**M1** - Timer 1 mode select bit 1.
TMOD.4	**M0** - Timer 1 mode select bit 0, decoded as: M1 M0 Mode 0 0 Mode 0 : 13-bit counter 0 1 Mode 1 : 16-bit counter 1 0 Mode 2 : 8-bit counter with auto-reload 1 1 Mode 3 : Timer 1 stopped
TMOD.3	**GATE** - Timer 0 gate control, When GATE = 1, Timer 0 will clock only when INT0 = 1 and TR0 (TCON.4) = 1. When GATE = 0, Timer 0 will clock only when TR0 = 1, regardless of the state of INT0.
TMOD.2	**C/$\overline{\text{T}}$** - Counter/Timer select. When C/$\overline{\text{T}}$ = 0, Timer 0 is clocked by CLKOUT/4 or CLKOUT/12, depending on the state of T0M (CKCON.3). When C/$\overline{\text{T}}$ = 1, Timer 0 is clocked by the T0 pin.
TMOD.1	**M1** - Timer 0 mode select bit 1.
TMOD.0	**M0** - Timer 0 mode select bit 0, decoded as: M1 M0 Mode 0 0 Mode 0 : 13-bit counter 0 1 Mode 1 : 16-bit counter 1 0 Mode 2 : 8-bit counter with auto-reload 1 1 Mode 3 : Two 8-bit counters

Table 18-9. TCON Register — SRF 88h

Bit	Function
TCON.7	**TF1** - Timer 1 overflow flag. Set to 1 when the Timer 1 count overflows and cleared when the processor vectors to the interrupt service routine.
TCON.6	**TR1** - Timer 1 run control. Set to 1 to enable counting on Timer 1.
TCON.5	**TF0** - Timer 0 overflow flag. Set to 1 when the Timer 0 count overflows and cleared when the processor vectors to the interrupt service routine.
TCON.4	**TR0** - Timer 0 run control. Set to 1 to enable counting on Timer 0.
TCON.3	**IE1** - Interrupt 1 edge detect. If external interrupt 1 is configured to be edge-sensitive (IT1 = 1), IE1 is set by hardware when a negative edge is detected on the INT1 pin and is automatically cleared when the CPU vectors to the corresponding interrupt service routine. In this case, IE1 can also be cleared by software. If external interrupt 1 is configured to be level-sensitive (IT1 = 0), IE1 is set when the INT1# pin is 0 and cleared when the INT1# pin is 1. In level-sensitive mode, software cannot write to IE1.
TCON.2	**IT1** - Interrupt 1 type select. INT1 is detected on falling edge when IT1 = 1; INT1 is detected as a low level when IT1 = 0.
TCON.1	**IE0** - Interrupt 0 edge detect. If external interrupt 0 is configured to be edge-sensitive (IT0 = 1), IE0 is set by hardware when a negative edge is detected on the INT0 pin and is automatically cleared when the CPU vectors to the corresponding interrupt service routine. In this case, IE0 can also be cleared by software. If external interrupt 0 is configured to be level-sensitive (IT0 = 0), IE0 is set when the INT0# pin is 0 and cleared when the INT0# pin is 1. In level-sensitive mode, software cannot write to IE0.
TCON.0	**IT0** - Interrupt 0 type select. INT0 is detected on falling edge when IT0 = 1; INT0 is detected as a low level when IT0 = 0.

18.2.2.3 Mode 2

Mode 2 operation is the same for Timer 0 and Timer 1. In mode 2, the timer is configured as an 8-bit counter, with automatic reload of the start value. The LSB Register (TL0 or TL1) is the counter and the MSB Register (TH0 or TH1) stores the reload value.

As illustrated in *Figure 18-4*, mode 2 counter control is the same as for mode 0 and mode 1. However, in mode 2, when TL*n* increments from FFh, the value stored in TH*n* is reloaded into TL*n*.

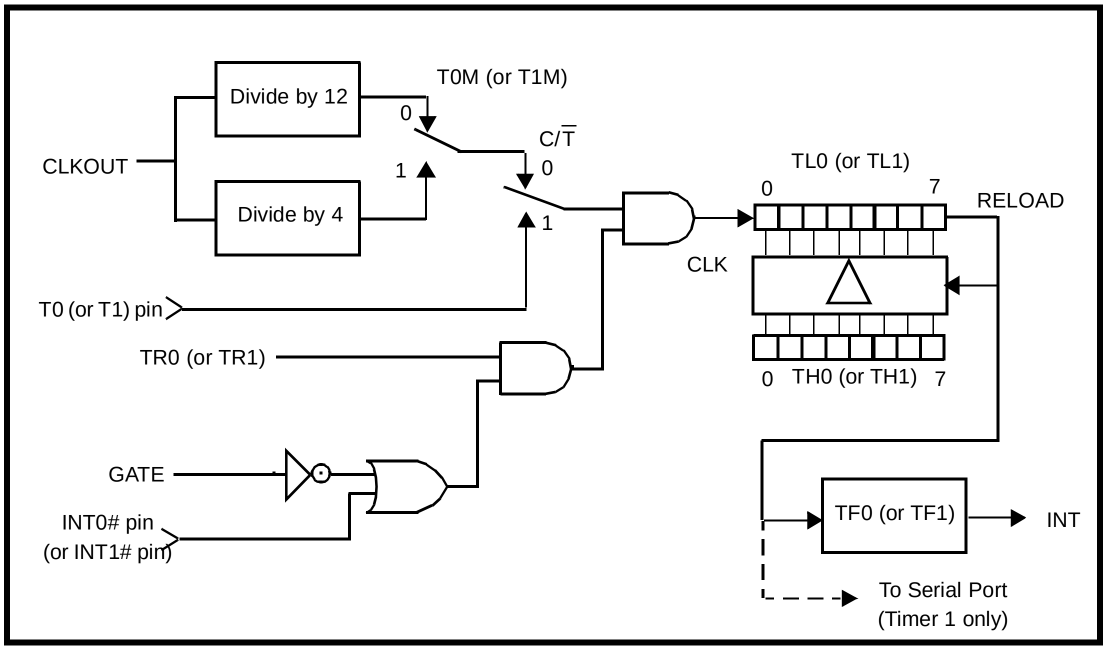


Figure 18-4. Timer 0/1 - Mode 2

18.2.2.4 Mode 3

In mode 3, Timer 0 operates as two 8-bit counters and Timer 1 stops counting and holds its value.

As shown in *Figure 18-5*, TL0 is configured as an 8-bit counter controlled by the normal Timer 0 control bits. TL0 can either count CLKOUT cycles (divided by 4 or by 12) or high-to-low transitions on *T0*, as determined by the C/$\overline{T}$ Bit. The GATE function can be used to give counter enable control to the INT0# pin.

TH0 functions as an independent 8-bit counter. However, TH0 can only count CLKOUT cycles (divided by 4 or by 12). The Timer 1 control and flag bits (TR1 and TF1) are used as the control and flag bits for TH0.

When Timer 0 is in mode 3, Timer 1 has limited usage because Timer 0 uses the Timer 1 control bit (TR1) and interrupt flag (TF1). Timer 1 can still be used for baud rate generation and the Timer 1 count values are still available in the TL1 and TH1 Registers.

Control of Timer 1 when Timer 0 is in mode 3 is through the Timer 1 mode bits. To turn Timer 1 on, set Timer 1 to mode 0, 1, or 2. To turn Timer 1 off, set it to mode 3. The Timer 1 C/$\overline{T}$ Bit and T1M Bit are still available to Timer 1. Therefore, Timer 1 can count CLKOUT/4, CLKOUT/12, or high-to-low transitions on the T1 pin. The Timer 1 GATE function is also available when Timer 0 is in mode 3.

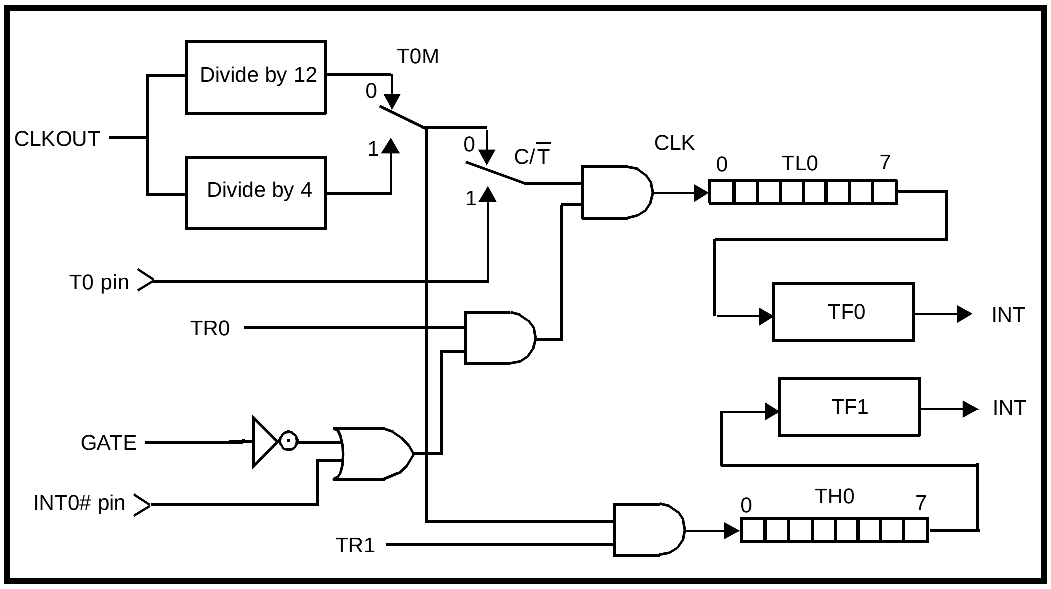


Figure 18-5. Timer 0 - Mode 3

18.2.3 Timer Rate Control

The default timer clock scheme for the 8051 timers is 12 CLKOUTcycles per increment, the same as in the standard 8051. However, in the 8051, the instruction cycle is 4 CLKOUT cycles.

Using the default rate (12 clocks per timer increment) allows existing application code with real-time dependencies, such as baud rate, to operate properly. However, applications that require fast timing can set the timers to increment every 4 CLKOUT cycles by setting bits in the Clock Control Register (CKCON) at SFR location 8Eh. (SeeTable 18-10).

The CKCON Bits that control the timer clock rates are:

CKCON Bit	Counter/Timer
5	Timer 2
4	Timer 1
3	Timer 0

When a CKCON Register Bit is set to 1, the associated counter increments at 4-CLKOUT intervals. When a CKCON Bit is cleared, the associated counter increments at 12-CLKOUT intervals. The timer controls are independent of each other. The default setting for all three timers is 0 (12-CLKOUT intervals). These bits have no effect in counter mode.

Table 18-10. CKCON Register — SRF 8Eh

Bit	Function
CKCON.7,6	Reserved
CKCON.5	**T2M** - Timer 2 clock select. When T2M = 0, Timer 2 uses CLKOUT/12 (for compatibility with 80C32); when T2M = 1, Timer 2 uses CLKOUT/4. This bit has no effect when Timer 2 is configured for baud rate generation.
CKCON.4	**T1M** - Timer 1 clock select. When T1M = 0, Timer 1 uses CLKOUT/12 (for compatibility with 80C32); when T1M = 1, Timer 1 uses CLKOUT/4.
CKCON.3	**T0M** - Timer 0 clock select. When T0M = 0, Timer 0 uses CLKOUT/12 (for compatibility with 80C32); when T0M = 1, Timer 0 uses CLKOUT/4.
CKCON.2-0	**MD2, MD1, MD0** - Control the number of cycles to be used for external MOVX instructions.

18.2.4 Timer 2

Timer 2 runs only in 16-bit mode and offers several capabilities not available with Timers 0 and 1. The modes available with Timer 2 are:

- 16-bit timer/counter
- 16-bit timer with capture
- 16-bit auto-reload timer/counter
- Baud rate generator.

The SFRs associated with Timer 2 are:

- T2CON — SFR C8h (Table 18-12)
- RCAP2L — SFR CAh - Used to capture the TL2 value when Timer 2 is configured for capture mode, or as the LSB of the 16-bit reload value when Timer 2 is configured for auto-reload mode.
- RCAP2H — SFR CBh - Used to capture the TH2 value when Timer 2 is configured for capture mode, or as the MSB of the 16-bit reload value when Timer 2 is configured for auto-reload mode.
- TL2 - SFR CCh — Lower 8 bits of the 16-bit count.
- TH2 - SFR CDh — Upper 8 bits of the 16-bit count.

18.2.4.1 Timer 2 Mode Control

Table 18-11 summarizes how the SFR Bits determine the Timer 2 mode.

Table 18-11. Timer 2 Mode Control Summary

RCLK	TCLK	CP/$\overline{\text{RL2}}$	TR2	Mode
0	0	1	1	16-bit timer/counter with capture
0	0	0	1	16-bit timer/counter with auto-reload
1	X	X	1	Baud rate generator
X	1	X	1	Baud rate generator
X	X	X	0	Off
X = Don't care.				

18.2.5 16-Bit Timer/Counter Mode

Figure 18-6 illustrates how Timer 2 operates in timer/counter mode with the optional capture feature. The C/$\overline{\text{T2}}$ Bit determines whether the 16-bit counter counts CLKOUT cycles (divided by 4 or 12), or high-to-low transitions on the T2 pin. The TR2 Bit enables the counter. When the count increments from FFFFh, the TF2 flag is set, and the T2OUT pin goes high for one CLKOUT cycle.

Table 18-12. T2CON Register — SFR C8h

Bit	Function
T2CON.7	**TF2** - Timer 2 overflow flag. Hardware will set TF2 when the Timer 2 overflows from FFFFh. TF2 must be cleared to 0 by the software. TF2 will only be set to a 1 if RCLK and TCLK are both cleared to 0. Writing a 1 to TF2 forces a Timer 2 interrupt if enabled.
T2CON.6	**EXF2** - Timer 2 external flag. Hardware will set EXF2 when a reload or capture is caused by a high-to-low transition on the T2EX pin, and EXEN2 is set. EXF2 must be cleared to 0 by the software. Writing a 1 to EXF2 forces a Timer 2 interrupt if enabled.
T2CON.5	**RCLK** - Receive clock flag. Determines whether Timer 1 or Timer 2 is used for Serial Port 0 timing of received data in serial mode 1 or 3. RCLK =1 selects Timer 2 overflow as the receive clock. RCLK =0 selects Timer 1 overflow as the receive clock.
T2CON.4	**TCLK** - Transmit clock flag. Determines whether Timer 1 or Timer 2 is used for Serial Port 0 timing of transmit data in serial mode 1 or 3. RCLK =1 selects Timer 2 overflow as the transmit clock. RCLK =0 selects Timer 1 overflow as the transmit clock.
T2CON.3	**EXEN2** - Timer 2 external enable. EXEN2 = 1 enables capture or reload to occur as a result of a high-to-low transition on the T2EX pin, if Timer 2 is not generating baud rates for the serial port. EXEN2 = 0 causes Timer 2 to ignore all external events on the T2EX pin.
T2CON.2	**TR2** - Timer 2 run control flag. TR2 = 1 starts Timer 2. TR2 = 0 stops Timer 2.
T2CON.1	**C/$\overline{\text{T2}}$** - Counter/timer select. C/$\overline{\text{T2}}$ = 0 selects a timer function for Timer 2. C/$\overline{\text{T2}}$ = 1 selects a counter of falling transitions on the T2 pin. When used as a timer, Timer 2 runs at 4 clocks per tick or 12 clocks per tick as programmed by CKCON.5, in all modes except baud rate generator mode. When used in baud rate generator mode, Timer 2 runs at 2 clocks per tick, independent of the state of CKCON.5.
T2CON.0	**CP/$\overline{\text{RL2}}$** - Capture/reload flag. When CP/$\overline{\text{RL2}}$ = 1, Timer 2 captures occur on high-to-low transitions of the T2EX pin, if EXEN2 = 1. When CP/$\overline{\text{RL2}}$ = 0, auto-reloads occur when Timer 2 overflows or when high-to-low transitions occur on the T2EX pin, if EXEN2 = 1. If either RCLK or TCLK is set to 1, CP/$\overline{\text{RL2}}$ will not function and Timer 2 will operate in auto-reload mode following each overflow.

18.2.5.1 6-Bit Timer/Counter Mode with Capture

The Timer 2 capture mode (*Figure 18-6*) is the same as the 16-bit timer/counter mode, with the addition of the capture registers and control signals.

The CP/$\overline{\text{RL2}}$ Bit in the T2CON SFR enables the capture feature. When CP/$\overline{\text{RL2}}$ = 1, a high-to-low transition on the T2EX pin when EXEN2 = 1 causes the Timer 2 value to be loaded into the capture registers RCAP2L and RCAP2H.

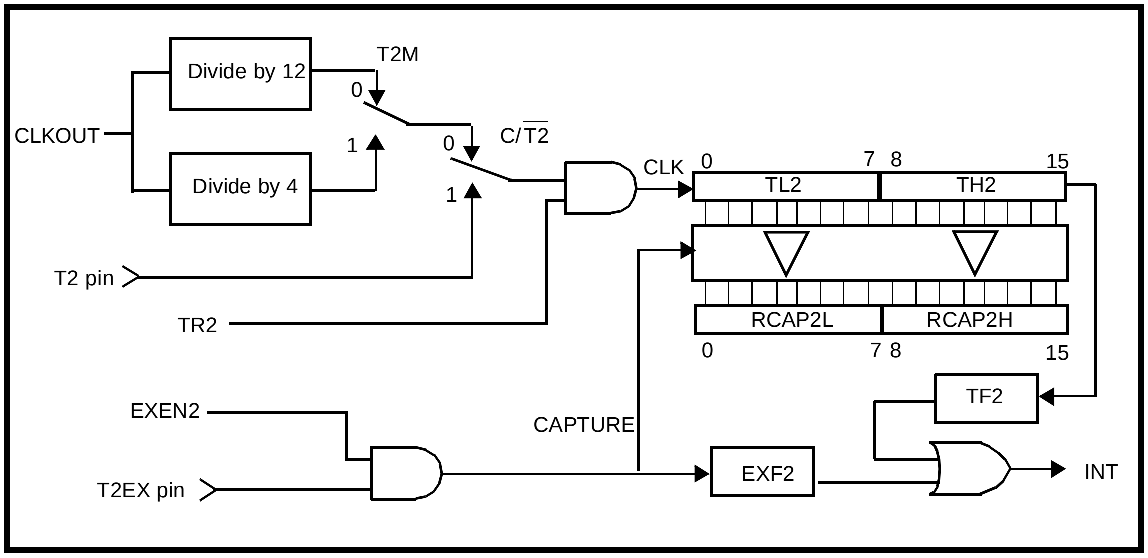


Figure 18-6. Timer 2 - Timer/Counter with Capture

18.2.6 16-Bit Timer/Counter Mode with Auto-Reload

When CP/$\overline{\text{RL2}}$ = 0, Timer 2 is configured for the auto-reload mode illustrated in *Figure 18-7* Control of counter input is the same as for the other 16-bit counter modes. When the count increments from FFFFh, Timer 2 sets the TF2 flag and the starting value is reloaded into TL2 and TH2. The software must preload the starting value into the RCAP2L and RCAP2H Registers.

When Timer 2 is in auto-reload mode, a reload can be forced by a high-to-low transition on the T2EX pin, if enabled by EXEN2 = 1.

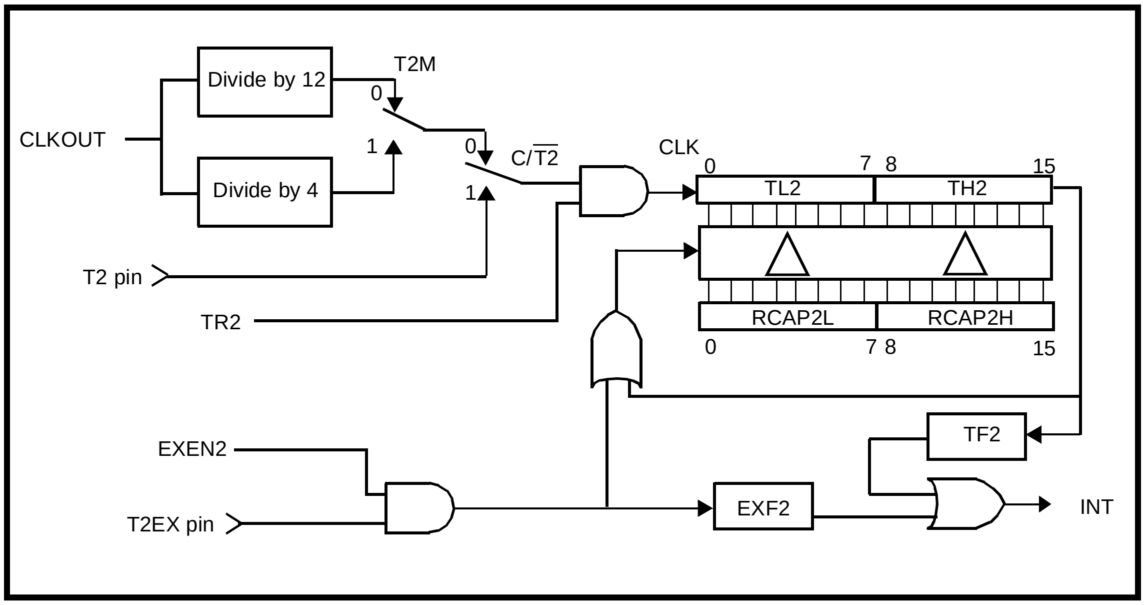


Figure 18-7. Timer 2 - Timer/Counter with Auto Reload

18.2.7 Baud Rate Generator Mode

Setting either RCLK or TCLK to 1 configures Timer 2 to generate baud rates for Serial Port 0 in serial mode 1 or 3. In baud rate generator mode, Timer 2 functions in auto-reload mode. However, instead of setting the TF2 flag, the counter overflow is used to generate a shift clock for the serial port function. As in normal auto-reload mode, the overflow also causes the pre-loaded start value in the RCAP2L and RCAP2H Registers to be reloaded into the TL2 and TH2 Registers.

When either TCLK = 1 or RCLK = 1, Timer 2 is forced into auto-reload operation, regardless of the state of the CP/$\overline{\text{RL2}}$ Bit.

When operating as a baud rate generator, Timer 2 does not set the TF2 Bit. In this mode, a Timer 2 interrupt can only be generated by a high-to-low transition on the T2EX pin setting the EXF2 Bit, and only if enabled by EXEN2 = 1.

The counter time base in baud rate generator mode is CLKOUT/2. To use an external clock source, set C/$\overline{\text{T2}}$ to 1 and apply the desired clock source to the T2 pin.

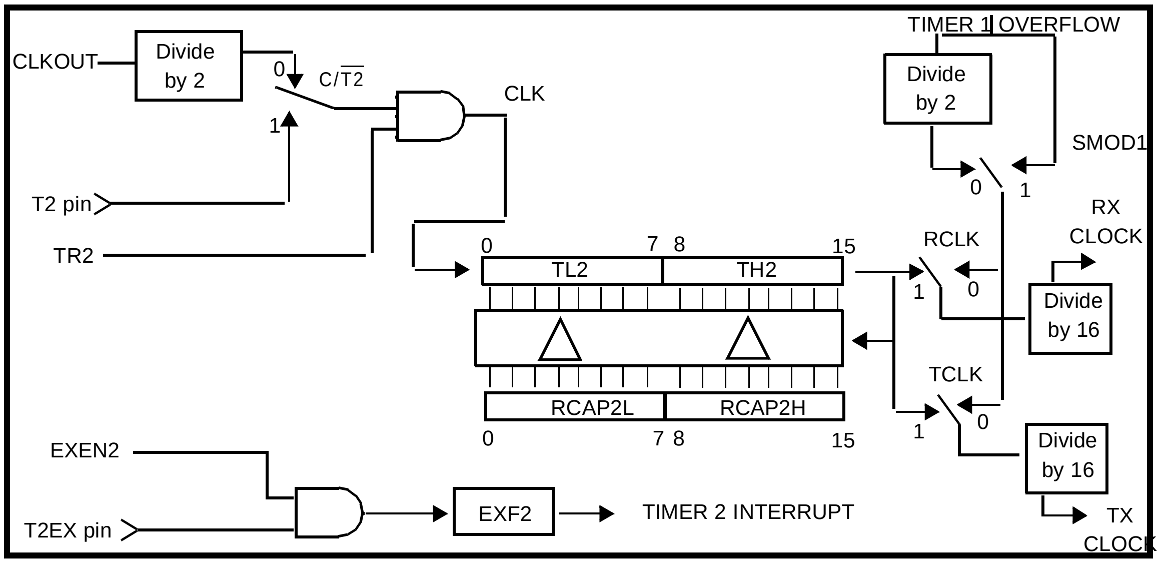


Figure 18-8. Timer 2 - Baud Rate Generator Mode

18.3 Serial Interface

The 8051 core provides two serial ports. Serial Port 0 is identical in operation to the standard 8051 serial port. Serial Port 1 is identical to Serial Port 0, except that Timer 2 cannot be used as the baud rate generator for Serial Port 1.

Each serial port can operate in synchronous or asynchronous mode. In synchronous mode, 8051 generates the serial clock and the serial port operates in half-duplex mode. In asynchronous mode, the serial port operates in full-duplex mode. In all modes, 8051 buffers received data in a holding register, enabling the UART to receive an incoming byte before the software has read the previous value.

Each serial port can operate in one of four modes, as outlined in Table 18-13.

Table 18-13. Serial Port Modes

Mode	Sync/ Async	Baud Clock	Data Bits	Start/Stop	9th Bit Function
0	Sync	CLKOUT/4 or CLKOUT/12	8	None	None
1	Async	Timer 1 or Timer 2[1]	8	1 start, 1 stop	None
2	Async	CLKOUT/32 or CLKOUT/64	9	1 start, 1 stop	0, 1, parity
3	Async	Timer 1 or Timer 2[1]	9	1 start, 1 stop	0, 1, parity
(1) Timer 2 available for Serial Port 0 only.					

The SFRs associated with the serial ports are:

- SCON0 - SFR 98h — Serial Port 0 control (Table 18-14).
- SBUF0 - SFR 99h — Serial Port 0 buffer.
- SCON1 - SFR C0h — Serial Port 1 control (Table 18-15).
- SBUF1 - SFR C1h — Serial Port 1 buffer.

18.3.1 803x/805x Compatibility

The implementation of the serial interface is similar to that of the Intel 8052.

18.3.2 Mode 0

Serial mode 0 provides synchronous, half-duplex serial communication. For Serial Port 0, serial data output occurs on the RXD0OUT pin, serial data is received on the RXD0 pin, and the TXD0 pin provides the shift clock for both transmit and receive. For Serial Port 1, the corresponding pins are RXD1OUT, RXD1, and TXD1.

The serial mode 0 baud rate is either CLKOUT/12 or CLKOUT/4, depending on the state of the SM2_0 Bit (or SM2_1 for Serial Port 1). When SM2_0 = 0, the baud rate is CLKOUT/12, when SM2_0 = 1, the baud rate is CLKOUT/4.

Mode 0 operation is identical to the standard 8051. Data transmission begins when an instruction writes to the SBUF0 (or SBUF1) SFR. The UART shifts the data, LSB first, at the selected baud rate, until the 8-bit value has been shifted out.

Mode 0 data reception begins when the REN_0 (or REN_1) Bit is set and the RI_0 (or RI_1) Bit is cleared in the corresponding SCON SFR. The shift clock is activated and the UART shifts data in on each rising edge of the shift clock until 8 bits have been received. One machine cycle after the 8th bit is shifted in, the RI_0 (or RI_1) Bit is set and reception stops until the software clears the RI Bit.

Figure 18-9 through *Figure 18-12* illustrate Serial Port Mode 0 transmit and receive timing for both low-speed (CLKOUT/12) and high-speed (CLKOUT/4) operation.

Table 18-14. SCON0 Register — SFR 98h

Bit	Function
SCON0.7	**SM0_0** - Serial Port 0 mode bit 0.
SCON0.6	**SM1_0** - Serial Port 0 mode bit 1, decoded as: SM0_0 SM1_0 Mode 0 0 0 0 1 1 1 0 2 1 1 3
SCON0.5	**SM2_0** - Multiprocessor communication enable. In modes 2 and 3, this bit enables the multi-processor communication feature. If SM2_0 = 1 in mode 2 or 3, then RI_0 will not be activated if the received 9th bit is 0. If SM2_0=1 in mode 1, then RI_0 will only be activated if a valid stop is received. In mode 0, SM2_0 establishes the baud rate: when SM2_0=0, the baud rate is CLKOUT/12; when SM2_0=1, the baud rate is CLKOUT/4.
SCON0.4	**REN_0** - Receive enable. When REN_0=1, reception is enabled.
SCON0.3	**TB8_0** - Defines the state of the 9th data bit transmitted in modes 2 and 3.
SCON0.2	**RB8_0** - In modes 2 and 3, RB8_0 indicates the state of the 9th bit received. In mode 1, RB8_0 indicates the state of the received stop bit. In mode 0, RB8_0 is not used.
SCON0.1	**TI_0** - Transmit interrupt flag. indicates that the transmit data word has been shifted out. In mode 0, TI_0 is set at the end of the 8th data bit. In all other modes, TI_0 is set when the stop bit is placed on the TXD0 pin. **TI_0 must be cleared by firmware.**
SCON0.0	**RI_0** - Receive interrupt flag. Indicates that serial data word has been received. In mode 0, RI_0 is set at the end of the 8th data bit. In mode 1, RI_0 is set after the last sample of the incoming stop bit, subject to the state of SM2_0. In modes 2 and 3, RI_0 is set at the end of the last sample of RB8_0. **RI_0 must be cleared by firmware.**

Table 18-15. SCON1 Register — SFR C0h

Bit	Function
SCON1.7	**SM0_1** - Serial Port 1 mode bit 0.
SCON1.6	**SM1_1** - Serial Port 1 mode bit 1, decoded as: SM0_1 SM1_1 Mode 0 0 0 0 1 1 1 0 2 1 1 3
SCON1.5	**SM2_1** - Multiprocessor communication enable. In modes 2 and 3, this bit enables the multiprocessor communication feature. If SM2_1 = 1 in mode 2 or 3, then RI_1 will not be activated if the received 9th bit is 0. If SM2_1=1 in mode 1, then RI_1 will only be activated if a valid stop is received. In mode 0, SM2_1 establishes the baud rate: when SM2_1=0, the baud rate is CLKOUT/12; when SM2_1=1, the baud rate is CLKOUT/4.
SCON1.4	**REN_1** - Receive enable. When REN_1=1, reception is enabled.
SCON1.3	**TB8_1** - Defines the state of the 9th data bit transmitted in modes 2 and 3.
SCON1.2	**RB8_1** - In modes 2 and 3, RB8_0 indicates the state of the 9th bit received. In mode 1, RB8_1 indicates the state of the received stop bit. In mode 0, RB8_1 is not used.
SCON1.1	**TI_1** - Transmit interrupt flag. indicates that the transmit data word has been shifted out. In mode 0, TI_1 is set at the end of the 8th data bit. In all other modes, TI_1 is set when the stop bit is placed on the TXD0 pin. TI_1 must be cleared by the software.
SCON1.0	**RI_1** - Receive interrupt flag. Indicates that serial data word has been received. In mode 0, RI_1 is set at the end of the 8th data bit. In mode 1, RI_1 is set after the last sample of the incoming stop bit, subject to the state of SM2_1. In modes 2 and 3, RI_1 is set at the end of the last sample of RB8_1. RI_1 must be cleared by the software.

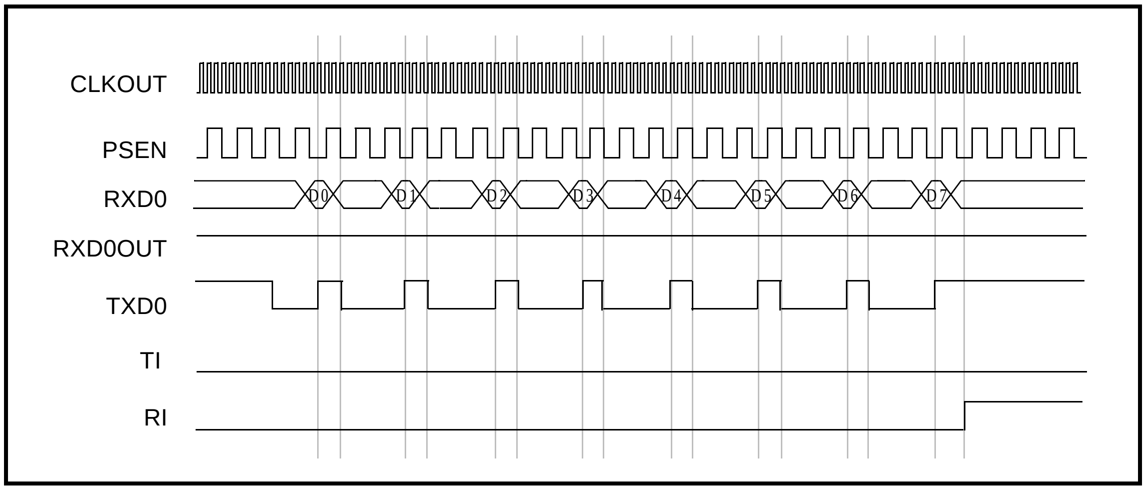


Figure 18-9. Serial Port Mode 0 Receive Timing - Low Speed Operation

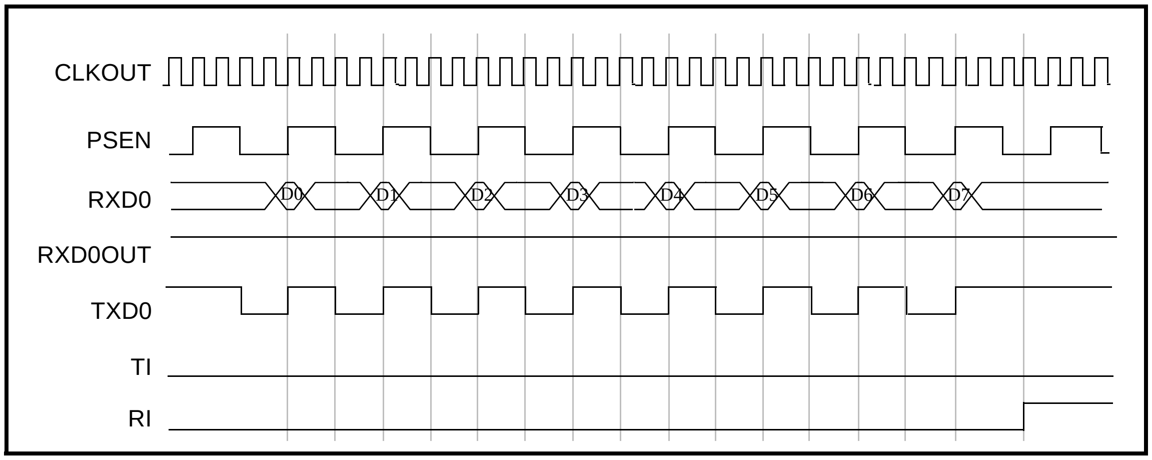


Figure 18-10. Serial Port Mode 0 Receive Timing - High Speed Operation

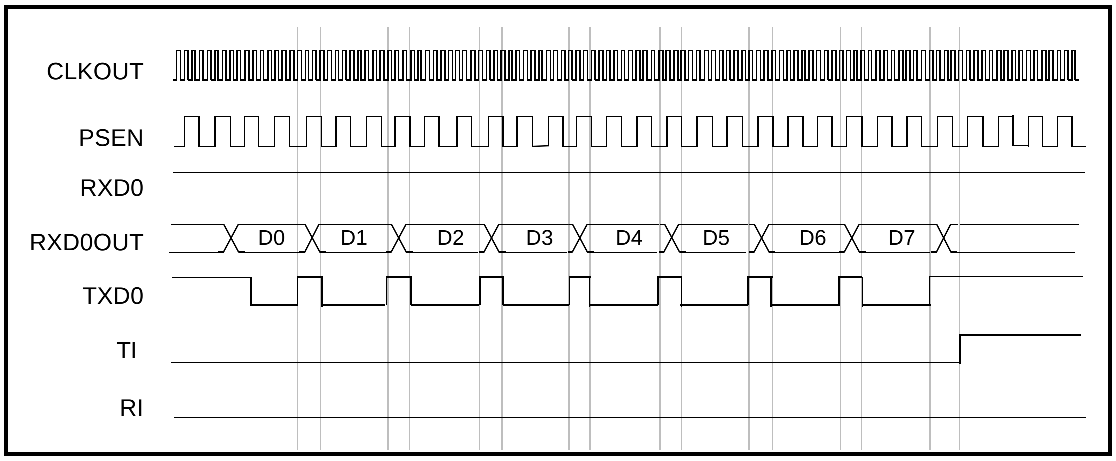


Figure 18-11. Serial Port Mode 0 Transmit Timing - Low Speed Operation

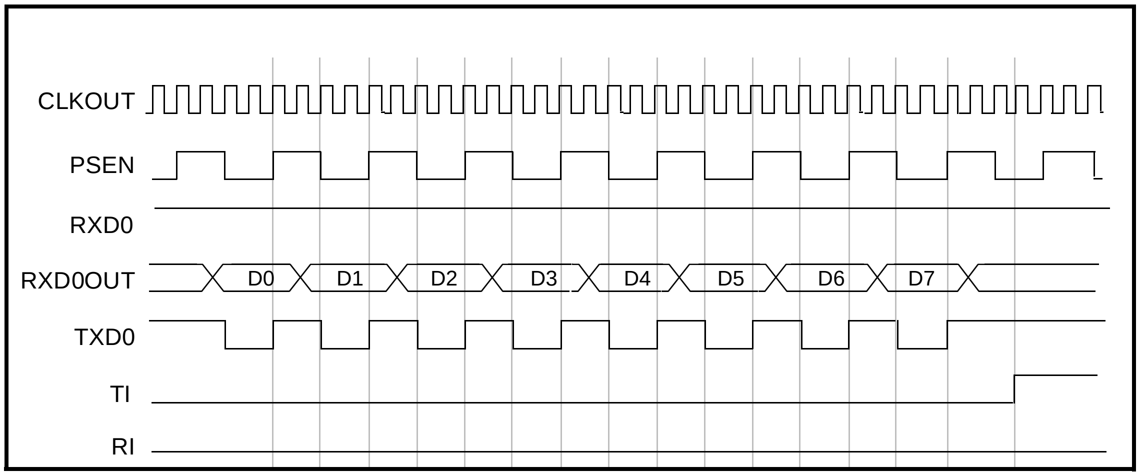


Figure 18-12. Serial Port Mode 0 Transmit Timing - High Speed Operation

18.3.3 Mode 1

Mode 1 provides standard asynchronous, full-duplex communication, using a total of 10 bits: 1 start bit, 8 data bits, and 1 stop bit. For receive operations, the stop bit is stored in RB8_0 (or RB8_1). Data bits are received and transmitted LSB first.

18.3.3.1 Mode 1 Baud Rate

The mode 1 baud rate is a function of timer overflow. Serial Port 0 can use either Timer 1 or Timer 2 to generate baud rates. Serial Port 1 can only use Timer 1. The two serial ports can run at the same baud rate if they both use Timer 1, or different baud rates if Serial Port 0 uses Timer 2 and Serial Port 1 uses Timer 1.

Each time the timer increments from its maximum count (FFh for Timer 1 or FFFFh for Timer 2), a clock is sent to the baud rate circuit. The clock is then divided by 16 to generate the baud rate.

When using Timer 1, the SMOD0 (or SMOD1) Bit selects whether or not to divide the Timer 1 roll-over rate by 2. Therefore, when using Timer 1, the baud rate is determined by the equation:

$$\text{Baud Rate} = \frac{2^{\text{SMODx}}}{32} \times \text{Timer 1 Overflow}$$

SMOD0 is SFR Bit PCON.7; SMOD1 is SFR Bit EICON.7.

When using Timer 2, the baud rate is determined by the equation:

$$\text{Baud Rate} = \frac{\text{Timer 2 Overflow}}{16}$$

To use Timer 1 as the baud rate generator, it is best to use Timer 1 mode 2 (8-bit counter with auto-reload), although any counter mode can be used. The Timer 1 reload is stored in the TH1 Register, which makes the complete formula for Timer 1:

$$\text{Baud Rate} = \frac{2^{\text{SMODx}}}{32} \times \frac{\text{CLKOUT}}{12 \times (256 - \text{TH1})}$$

The 12 in the denominator in the above equation can be changed to 4 by setting the T1M Bit in the CKCON SFR. To derive the required TH1 value from a known baud rate (when TM1 = 0), use the equation:

$$\text{TH1} = 256 - \frac{2^{\text{SMODx}} \times \text{CLKOUT}}{384 \times \text{Baud Rate}}$$

You can also achieve very low serial port baud rates from Timer 1 by enabling the Timer 1 interrupt, configuring Timer 1 to mode 1, and using the Timer 1 interrupt to initiate a 16-bit software reload. Table 18-16 lists sample reload values for a variety of common serial port baud rates.

More accurate baud rates are achieved by using Timer 2 as the baud rate generator (next section).

Table 18-16. Timer 1 Reload Values for Common Serial Port Mode 1 Baud Rates

Nominal Rate	24 MHz Divisor	Reload Value	Actual Rate	Error
57600	6	FA	62500	8.5%
38400	10	F6	37500	-2.3%
28800	13	F3	28846	+0.16%
19200	20	EC	18750	-2.3%
9600	39	D9	9615	+0.16%
4800	78	B2	4807	+0.15%
2400	156	64	2403	+.13%
Settings: SMOD =1, C/$\overline{\text{T}}$=0, Timer1 mode=2, TIM=1 ***Note:*** Using rates that are off by 2.3% or more will not work in all systems.				

To use Timer 2 as the baud rate generator, configure Timer 2 in auto-reload mode and set the TCLK and/or RCLK Bits in the T2CON SFR. TCLK selects Timer 2 as the baud rate generator for the transmitter; RCLK selects Timer 2 as the baud rate generator for the receiver. The 16-bit reload value for Timer 2 is stored in the RCAP2L and RCA2H SFRs, which makes the equation for the Timer 2 baud rate:

$$\text{Baud Rate} = \frac{\text{CLKOUT}}{32 \times (65536 - \text{RCAP2H,RCAP2L})}$$

where RCAP2H,RCAP2L is the content of RCAP2H and RCAP2L taken as a 16-bit unsigned number.

The 32 in the denominator is the result of CLKOUT being divided by 2 and the Timer 2 overflow being divided by 16. Setting TCLK or RCLK to 1 automatically causes CLKOUT to be divided by 2, as shown in *Figure 18-8*, instead of the 4 or 12 as determined by the T2M Bit in the CKCON SFR.

To derive the required RCAP2H and RCAP2L values from a known baud rate, use the equation:

$$\text{RCAP2H,RCAP2L} = 65536 - \frac{\text{CLKOUT}}{32 \times \text{Baud Rate}}$$

When either RCLK or TCLK is set, the TF2 flag is not set on a Timer 2 roll over, and the T2EX reload trigger is disabled.

Table 18-17. Timer 2 Reload Values for Common Serial Port Mode 1 Baud Rates

Nominal Rate	C/T2	Divisor	Reload Val	Actual Rate	Error
57600	0	13	F3	57692.31	0.16%
38400	0	20	EC	37500	-2.34%
28800	0	26	E6	28846.15	0.16%
19200	0	39	D9	19230.77	0.16%
9600	0	78	B2	9615.385	0.16%
4800	0	156	64	4807.692	0.16%
2400	0	312	FEC8	2403.846	0.16%
Note: using rates that are off by 2.3% or more will not work in all systems.					

18.3.3.2 Mode 1 Transmit

Figure 18-13 illustrates the mode 1 transmit timing. In mode 1, the UART begins transmitting after the first roll over of the divide-by-16 counter after the software writes to the SBUF0 (or SBUF1) Register. The UART transmits data on the TXD0 (or TXD1) pin in the following order: start bit, 8 data bits (LSB first), stop bit. The TI_0 (or TI_1) Bit is set 2 CLKOUT cycles after the stop bit is transmitted.

18.3.4 Mode 1 Receive

Figure 18-14 illustrates the mode 1 receive timing. Reception begins at the falling edge of a start bit received on the RXD0 (or RXD1) pin, when enabled by the REN_0 (or REN_1) Bit. For this purpose, the RXD0 (or RXD1) pin is sampled 16 times per bit for any baud rate. When a falling edge of a start bit is detected, the divide-by-16 counter used to generate the receive clock is reset to align the counter roll over to the bit boundaries.

For noise rejection, the serial port establishes the content of each received bit by a majority decision of 3 consecutive samples in the middle of each bit time. This is especially true for the start bit. If the falling edge on the RXD0 (or RXD1) pin is not verified by a majority decision of 3 consecutive samples (low), then the serial port stops reception and waits for another falling edge on the RXD0 (or RXD1) pin.

At the middle of the stop bit time, the serial port checks for the following conditions:

- RI_0 (or RI_1) = 0, and
- If SM2_0 (or SM2_1) = 1, the state of the stop bit is 1.
  (If SM2_0 (or SM2_1) = 0, the state of the stop bit doesn't matter.)

If the above conditions are met, the serial port then writes the received byte to the SBUF0 (or SBUF1) Register, loads the stop bit into RB8_0 (or RB8_1), and sets the RI_0 (or RI_1) Bit. If the above conditions are not met, the received data is lost, the SBUF Register and RB8 Bit are not loaded, and the RI Bit is not set.

After the middle of the stop bit time, the serial port waits for another high-to-low transition on the (RXD0 or RXD1) pin.

Mode 1 operation is identical to that of the standard 8051 when Timers 1 and 2 use CLKOUT/12 (the default).

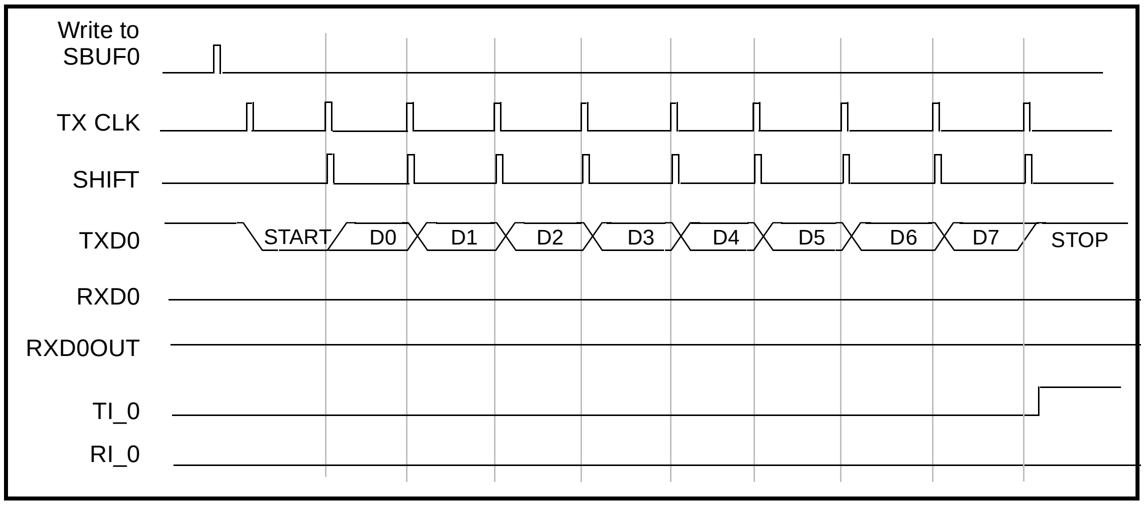


Figure 18-13. Serial Port 0 Mode 1 Transmit Timing

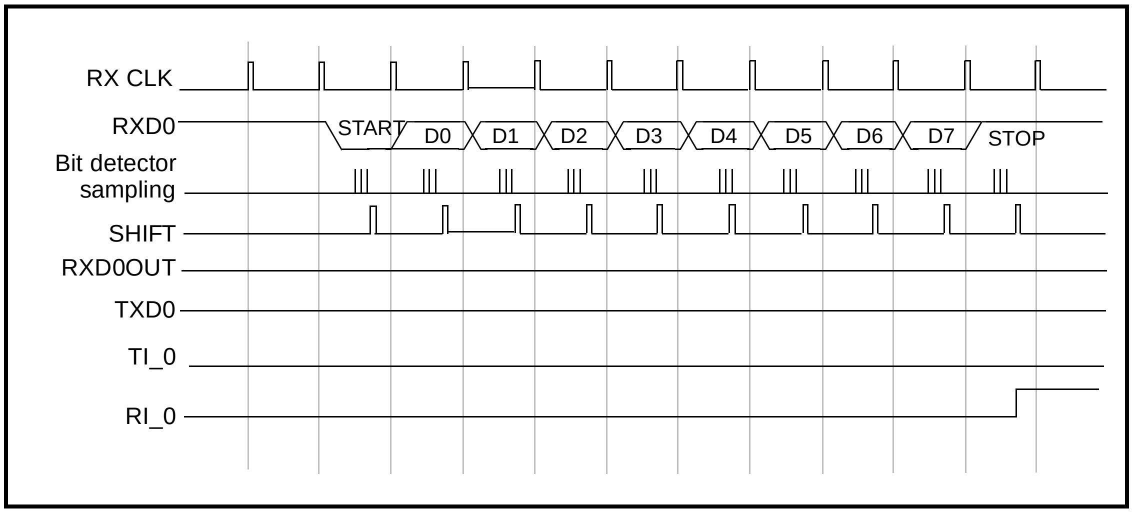


Figure 18-14. Serial Port 0 Mode 1 Receive Timing

18.3.5 Mode 2

Mode 2 provides asynchronous, full-duplex communication, using a total of 11 bits: 1 start bit, 8 data bits, a programmable 9th bit, and 1 stop bit. The data bits are transmitted and received LSB first. For transmission, the 9th bit is determined by the value in TB8_0 (or TB8_1). To use the 9th bit as a parity bit, move the value of the P Bit (SFR PSW.0) to TB8_0 (or TB8_1).

The mode 2 baud rate is either CLKOUT/32 or CLKOUT/64, as determined by the SMOD0 (or SMOD1) Bit. The formula for the mode 2 baud rate is:

$$\text{Baud Rate} = \frac{2^{\text{SMODx}} \times \text{CLKOUT}}{64}$$

Mode 2 operation is identical to the standard 8051.

18.3.5.1 Mode 2 Transmit

Figure 18-15 illustrates the mode 2 transmit timing. Transmission begins after the first roll over of the divide-by-16 counter following a software write to SBUF0 (or SBUF1). The UART shifts data out on the TXD0 (or TXD1) pin in the following order: start bit, data bits (LSB first), 9th bit, stop bit. The TI_0 (or TI_1) Bit is set when the stop bit is placed on the TXD0 (or TXD1) pin.

18.3.5.2 Mode 2 Receive

Figure 18-16 illustrates the mode 2 receive timing. Reception begins at the falling edge of a start bit received on the RXD0 (or RXD1) pin, when enabled by the REN_0 (or REN_1) Bit. For this purpose, the RXD0 (or RXD1) pin is sampled 16 times per bit for any baud rate. When a falling edge of a start bit is detected, the divide-by-16 counter used to generate the receive clock is reset to align the counter roll over to the bit boundaries.

For noise rejection, the serial port establishes the content of each received bit by a majority decision of 3 consecutive samples in the middle of each bit time. This is especially true for the start bit. If the falling edge on the RXD0 (or RXD1) pin is not verified by a majority decision of 3 consecutive samples (low), then the serial port stops reception and waits for another falling edge on the RXD0 (or RXD1) pin.

At the middle of the stop bit time, the serial port checks for the following conditions:

- RI_0 (or RI_1) = 0, and
- If SM2_0 (or SM2_1) = 1, the state of the stop bit is 1.
  (If SM2_0 (or SM2_1) = 0, the state of the stop bit doesn't matter.)

If the above conditions are met, the serial port then writes the received byte to the SBUF0 (or SBUF1) Register, loads the stop bit into RB8_0 (or RB8_1), and sets the RI_0 (or RI_1) Bit. If the above conditions are not met, the received data is lost, the SBUF Register and RB8 Bit are not loaded, and the RI Bit is not set. After the middle of the stop bit time, the serial port waits for another high-to-low transition on the RXD0 *(*or RXD1) pin.

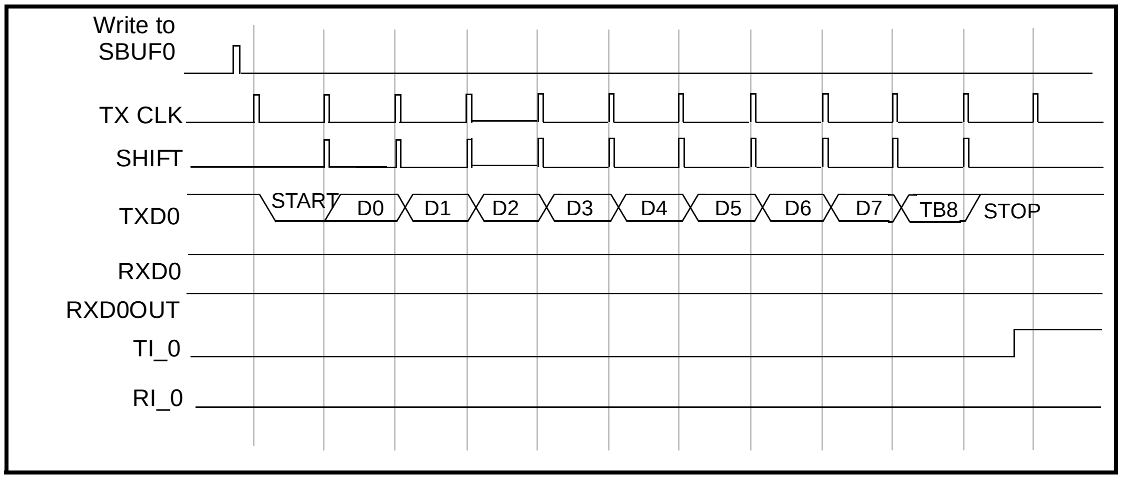


Figure 18-15. Serial Port 0 Mode 2 Transmit Timing

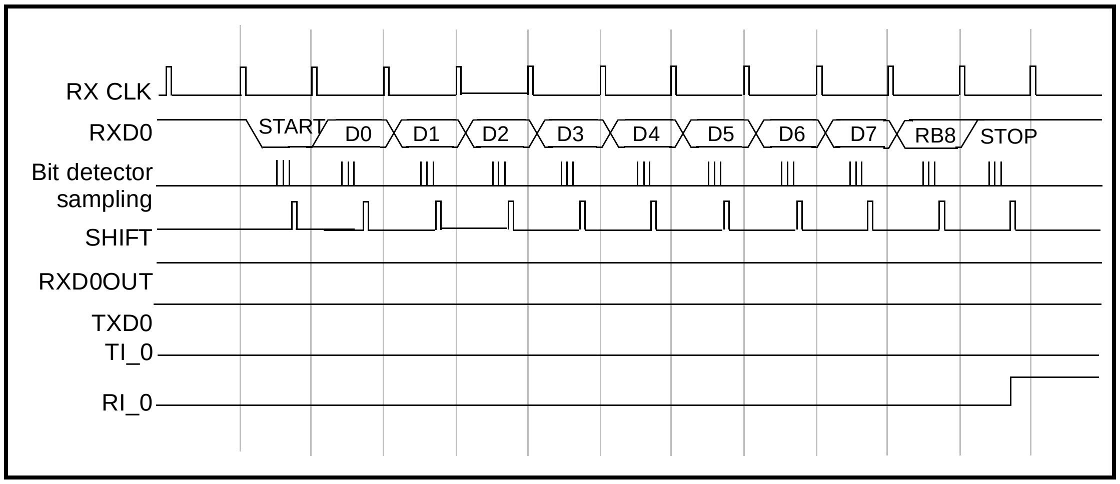


Figure 18-16. Serial Port 0 Mode 2 Receive Timing

18.3.6 Mode 3

Mode 3 provides asynchronous, full-duplex communication, using a total of 11 bits: 1 start bit, 8 data bits, a programmable 9th bit, and 1 stop bit. The data bits are transmitted and received LSB first.

The mode 3 transmit and operations are identical to mode 2. The mode 3 baud rate generation is identical to mode 1. That is, mode 3 is a combination of mode 2 protocol and mode 1 baud rate. *Figure 18-17* illustrates the mode 3 transmit timing.

Mode 3 operation is identical to that of the standard 8051 when Timers 1 and 2 use CLKOUT/12 (the default).

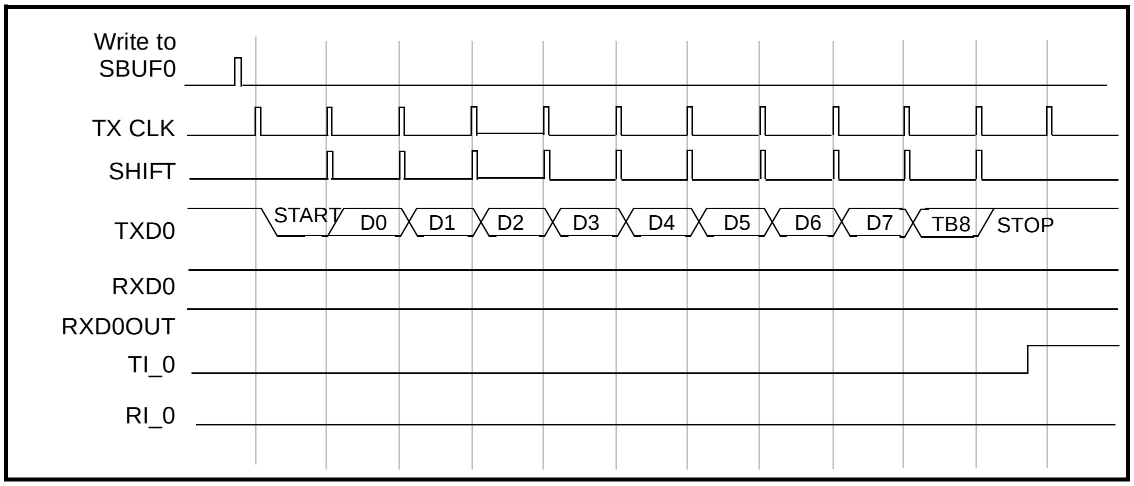


Figure 18-17. Serial Port 0 Mode 3 Transmit Timing

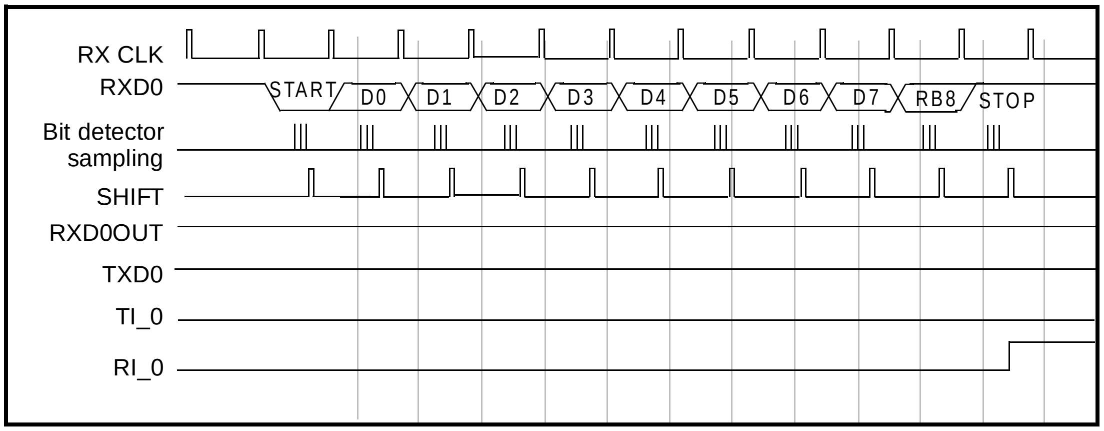


Figure 18-18. Serial Port 0 Mode 3 Receive Timing

18.3.7 Multiprocessor Communications

The multiprocessor communication feature is enabled in modes 2 and 3 when the SM2 Bit is set in the SCON SFR for a serial port (SM2_0 for Serial Port 0, SM2_1 for Serial Port 1). In multiprocessor communication mode, the 9th bit received is stored in RB8_0 (or RB8_1) and, after the stop bit is received, the serial port interrupt is activated only if RB8_0 (or RB8_1) = 1.

A typical use for the multiprocessor communication feature is when a master wants to send a block of data to one of several slaves. The master first transmits an address byte that identifies the target slave. When transmitting an address byte, the master sets the 9th bit to 1; for data bytes, the 9th bit is 0.

With SM2_0 (or SM2_1) = 1, no slave will be interrupted by a data byte. However, an address byte interrupts all slaves so that each slave can examine the received address byte to determine whether that slave is being addressed. Address decoding must be done by software during the interrupt service routine. The addressed slave clears its SM2_0 (or SM2_1) Bit and prepares to receive the data bytes. The slaves that are not being addressed leave the SM2_0 (or SM2_1) Bit set and ignore the incoming data bytes.

18.3.8 Interrupt SFRs

The following SFRs are associated with interrupt control:

- IE - SFR A8h (Table 18-18)
- IP - SFR B8h (Table 18-19)
- EXIF - SFR 91h (Table 18-20)
- EICON - SFR D8h (Table 18-21)
- EIE - SFR E8h (Table 18-22)
- EIP - SFR F8h (Table 18-23).

The IE and IP SFRs provide interrupt enable and priority control for the standard interrupt unit, as with the standard 8051. Additionally, these SFRs provide control bits for the Serial Port 1 interrupt. These bits (ES1 and PS1) are available only when the extended interrupt unit is implemented (ext_intr=1). Otherwise, they are read as 0.

Bits ES0, ES1, ET2, PS0, PS1, and PT2 are present, but not used, when the corresponding module is not implemented.

The EXIF, EICON, EIE and EIP Registers provide flags, enable control, and priority control for the optional extended interrupt unit.

Table 18-18. IE Register — SFR A8h

Bit	Function
IE.7	**EA** - Global interrupt enable. Controls masking of all interrupts except USB wakeup (resume). EA = 0 disables all interrupts except USB wakeup. When EA = 1, interrupts are enabled or masked by their individual enable bits.
IE.6	**ES1** - Enable Serial Port 1 interrupt. ES1 = 0 disables Serial port 1 interrupts (TI_1 and RI_1). ES1 = 1 enables interrupts generated by the TI_1 or TI_1 flag.
IE.5	**ET2** - Enable Timer 2 interrupt. ET2 = 0 disables Timer 2 interrupt (TF2). ET2=1 enables interrupts generated by the TF2 or EXF2 flag.
IE.4	**ES0** - Enable Serial Port 0 interrupt. ES0 = 0 disables Serial Port 0 interrupts (TI_0 and RI_0). ES0=1 enables interrupts generated by the TI_0 or RI_0 flag.
IE.3	**ET1** - Enable Timer 1 interrupt. ET1 = 0 disables Timer 1 interrupt (TF1). ET1=1 enables interrupts generated by the TF1 flag.
IE.2	**EX1** - Enable external interrupt 1. EX1 = 0 disables external interrupt 1 (INT1). EX1=1 enables interrupts generated by the INT1# pin.
IE.1	**ET0** - Enable Timer 0 interrupt. ET0 = 0 disables Timer 0 interrupt (TF0). ET0=1 enables interrupts generated by the TF0 flag.
IE.0	**EX0** - Enable external interrupt 0. EX0 = 0 disables external interrupt 0 (INT0). EX0=1 enables interrupts generated by the INT0# pin.

Table 18-19. IP Register — SFR B8h

Bit	Function
IP.7	**Reserved**. Read as 1.
IP.6	**PS1** - Serial Port 1 interrupt priority control. PS1=0 sets Serial Port 1 interrupt (TI_1 or RI_1) to low priority. PS1=1 sets Serial port 1 interrupt to high priority.
IP.5	**PT2** - Timer 2 interrupt priority control. PT2=0 sets Timer 2 interrupt (TF2) to low priority. PT2=1 sets Timer 2 interrupt to high priority.
IP.4	**PS0** - Serial Port 0 interrupt priority control. PS0=0 sets Serial Port 0 interrupt (TI_0 or RI_0) to low priority. PS0=1 sets Serial Port 0 interrupt to high priority.
IP.3	**PT2** - Timer 1 interrupt priority control. PT1 = 0 sets Timer 1 interrupt (TF1) to low priority. PT1=1 sets Timer 1 interrupt to high priority.
IP.2	**PX1** - External interrupt 1 priority control. PX 1= 0 sets external interrupt 1 (INT1) to low priority. PT1 = 1 sets external interrupt 1 to high priority.
IP.1	**PT0** - Timer 0 interrupt priority control. PT0 = 0 sets Timer 0 interrupt (TF0) to low priority. PT0=1 sets Timer 0 interrupt to high priority.
IP.0	**PX0** - External interrupt 0 priority control. PX0 = 0 sets external interrupt 0 (INT0) to low priority. PX0=1 sets external interrupt 0 to high priority.

Table 18-20. EXIF Register — SFR 91h

Bit	Function
EXIF.7	**IE5** - External interrupt 5 flag. IE 5= 1 indicates a falling edge was detected at the INT5# pin. IE5 must be cleared by software. Setting IE5 in software generates an interrupt, if enabled.
EXIF.6	**IE4** - External interrupt 4 flag. IE4 indicates a rising edge was detected at the INT4 pin. IE4 must be cleared by software. Setting IE4 in software generates an interrupt, if enabled.
EXIF.5	**I2CINT** - External interrupt 3 flag. The “INT3” interrupt is internally connected to the EZ-USB FX I^2C controller and renamed “I2CINT”. I2CINT = 1 indicates an I^2C interrupt. I2CINT must be cleared by software. Setting I2CINT in software generates an interrupt, if enabled.
EXIF.4	**USBINT** - External interrupt 2 flag. The “INT2” interrupt is internally connected to the EZ-USB FX interrupt and renamed “USBINT”. USBINT = 1 indicates an USB interrupt. USBINT must be cleared by software. Setting USBINT in software generates an interrupt, if enabled.
EXIF.3	**Reserved**. Read as 1.
EXIF.2-0	**Reserved**. Read as 0.

Table 18-21. EICON Register — SFR D8h

Bit	Function
EICON.7	**SMOD1** - Serial Port 1 baud rate doubler enable. When SMOD1 = 1 the baud rate for Serial Port is doubled.
EICON.6	**Reserved**. Read as 1.
EICON.5	**ERESI** - Enable resume interrupt. ERESI = 0 disables resume interrupt (RESI). ERESI = 1 enables interrupts generated by the resume event.
EICON.4	**RESI** - Wakeup interrupt flag. EICON.4 = 1 indicates a negative transition was detected at the WAKEUP# pin, or that USB has activity resumed from the suspended state. EICON.4 = 1 must be cleared by software before exiting the interrupt service routine, otherwise the interrupt occurs again. Setting EICON.4=1 in software generates a wakeup interrupt, if enabled.
EICON.3	**INT6** - External interrupt 6. When INT6 = 1, the INT6 pin has detected a low to high transition. INT6 will remain active until cleared by writing a 0 to this bit. Setting this bit in software generates an INT6 interrupt in enabled.
EICON.2-0	**Reserved**. Read as 0.

Table 18-22. EIE Register — SFR E8h

Bit	Function
EIE.7-5	Reserved. Read as 1.
EIE.4	EX6 - Enable external interrupt 6. EX6 = 0 disables external interrupt 6 (INT6). EX6 = 1 enables interrupts generated by the INT6 pin.
EIE.3	EX5 - Enable external interrupt 5. EX5 = 0 disables external interrupt 5 (INT5). EX5 = 1 enables interrupts generated by the INT5# pin.
EIE.2	EX4 - Enable external interrupt 4. EX4 = 0 disables external interrupt 4 (INT4). EX4 = 1 enables interrupts generated by the INT4 pin.
EIE.1	EI2C - Enable external interrupt 3. EI2C = 0 disables external interrupt 3 (INT3). EI2C = 1 enables interrupts generated by the I^{2}C interface.
EIE.0	EUSB - Enable USB interrupt. EUSB = 0 disables USB interrupts. EUSB = 1 enables interrupts generated by the USB Interface.

Table 18-23. EIP Register — SFR F8h

Bit	Function
EIP.7-5	**Reserved**. Read as 1.
EIP.4	**PX6** - External interrupt 6 priority control. PX6 = 0 sets external interrupt 6 (INT6) to low priority. PX6 = 1 sets external interrupt 6 to high priority.
EIP.3	**PX5** - External interrupt 5 priority control. PX5 = 0 sets external interrupt 5 (INT5#) to low priority. PX5=1 sets external interrupt 5 to high priority.
EIP.2	**PX4** - External interrupt 4 priority control. PX4 = 0 sets external interrupt 4 (INT4) to low priority. PX4=1 sets external interrupt 4 to high priority.
EIP.1	**PI2C** - External interrupt 3 priority control. PI2C = 0 sets I^{2}C interrupt to low priority. PI2C=1 sets I^{2}C interrupt to high priority.
EIP.0	**PUSB** - External interrupt 2 priority control. PUSB = 0 sets USB interrupt to low priority. PUSB=1 sets USB interrupt to high priority.

18.4 Interrupt Processing

When an enabled interrupt occurs, the 8051 core vectors to the address of the interrupt service routine (ISR) associated with that interrupt, as listed in Table 18-24. The 8051 core executes the ISR to completion unless another interrupt of higher priority occurs. Each ISR ends with a `RETI` (return from interrupt) instruction. After executing the `RETI`, the CPU returns to the next instruction that would have been executed if the interrupt had not occurred.

An ISR can only be interrupted by a higher priority interrupt. That is, an ISR for a low-level interrupt can only be interrupted by high-level interrupt. An ISR for a high-level interrupt can only be interrupted by the resume interrupt.

The 8051 core always completes the instruction in progress before servicing an interrupt. If the instruction in progress is RETI, or a write access to any of the IP, IE, EIP, or EIE SFRs, the 8051 core completes one additional instruction before servicing the interrupt.

18.4.1 Interrupt Masking

The EA Bit in the IE SFR (IE.7) is a global enable for all interrupts except the USB wakeup (resume) interrupt. When EA = 1, each interrupt is enabled or masked by its individual enable bit. When EA = 0, all interrupts are masked, except the USB wakeup interrupt.

Table 18-25 provides a summary of interrupt sources, flags, enables, and priorities.

Table 18-24. Interrupt Natural Vectors and Priorities

Interrupt	Description	Natural Priority	Interrupt Vector
RESUME	USB Wakeup (resume) interrupt	0	33h
INT0	External interrupt 0	1	03h
TF0	Timer 0 interrupt	2	0Bh
INT1	External interrupt 1	3	13h
TF1	Timer 1 interrupt	4	1Bh
TI_0 or RI_0	Serial port 0 interrupt	5	23h
TF2 or EXF2	Timer 2 interrupt	6	2Bh
TI_1 or RI_1	Serial port 1 interrupt	7	3Bh
INT2	USB interrupt	8	43h
INT3	I^2C interrupt	9	4Bh
INT4	External interrupt 4	4	53h
INT5	External interrupt 5	11	5Bh
INT6	External interrupt 6	12	63H

18.4.1.1 Interrupt Priorities

There are two stages of interrupt priority assignment, interrupt level and natural priority. The interrupt level (highest, high, or low) takes precedence over natural priority. The USB wakeup interrupt, if enabled, always has highest priority and is the only interrupt that can have highest priority. All other interrupts can be assigned either high or low priority.

In addition to an assigned priority level (high or low), each interrupt also has a natural priority, as listed in Table 18-24. Simultaneous interrupts with the same priority level (for example, both high) are resolved according to their natural priority. For example, if INT0 and INT2 are both programmed as high priority, INT0 takes precedence due to its higher natural priority.

Once an interrupt is being serviced, only an interrupt of higher priority level can interrupt the service routine of the interrupt currently being serviced.

Table 18-25. Interrupt Flags, Enables, and Priority Control

Interrupt	Description	Flag	Enable	Priority Control
RESUME	Resume interrupt	EICON.4	EICON.5	N/A
INT0	External interrupt 0	TCON.1	IE.0	IP.0
TF0	Timer 0 interrupt	TCON.5	IE.1	IP.1
INT1	External interrupt 1	TCON.3	IE.2	IP.2
TF1	Timer 1 interrupt	TCON.7	IE.3	IP.3
TI_0 or RI_0	Serial port 0 transmit or receive	SCON0.0 (RI.0), SCON0.1 (Ti_0)	IE.4	IP.4
TF2 or EXF2	Timer 2 interrupt	T2CON.7 (TF2), T2CON.6 (EXF2)	IE.5	IP.5
TI_1 or RI_1	Serial port 1 transmit or receive	SCON1.0 (RI_1), SCON1.1 (TI_1)	IE.6	IP.6
USB (INT2)	USB interrupt	EXIF.4	EIE.0	EIP.0
I^{2}C (INT3)	I^{2}C interrupt	EXIT.5	EIE.1	EIP.1
INT4	External interrupt 4	EXIF.6	EIE.2	EIP.2
INT5	External interrupt 5	EXIF.7	EIE.3	EIP.3
INT6	External INT 6	EICON.3	EIE.4	EIP.4

18.4.2 Interrupt Sampling

The internal timers and serial ports generate interrupts by setting their respective SFR interrupt flag bits. External interrupts are sampled once per instruction cycle.

INT0 and INT1 are both active low and can be programmed to be either edge-sensitive or level-sensitive, through the IT0 and IT1 Bits in the TCON SFR. For example, when IT0 = 0, INT0 is level-sensitive and the 8051 core sets the IE0 flag when the INT0# pin is sampled low. When IT0 = 1, INT0 is edge-sensitive and the 8051 sets the IE0 flag when the INT0# pin is sampled high then low on consecutive samples.

The remaining five interrupts (INT 4-6, USB & I^{2}C Interrupts) are edge-sensitive only. INT6 and INT4 are active high and INT5 is active low.

To ensure that edge-sensitive interrupts are detected, the corresponding ports should be held high for 4 CLKOUT cycles and then low for 4 CLKOUT cycles. Level-sensitive interrupts are not latched and must remain active until serviced.

18.4.3 Interrupt Latency

Interrupt response time depends on the current state of the 8051. The fastest response time is 5 instruction cycles: 1 to detect the interrupt, and 4 to perform the `LCALL` to the ISR.

The maximum latency (13 instruction cycles) occurs when the 8051 is currently executing a `RETI` instruction followed by a `MUL` or `DIV` instruction. The 13 instruction cycles in this case are: 1 to detect the interrupt, 3 to complete the `RETI`, 5 to execute the `DIV` or `MUL`, and 4 to execute the `LCALL` to the ISR. For the maximum latency case, the response time is 13 x 4 = 52 CLKOUT cycles.

18.4.4 Single-Step Operation

The 8051 interrupt structure provides a way to perform single-step program execution. When exiting an ISR with an `RETI` instruction, the 8051 will always execute at least one instruction of the task program. Therefore, once an ISR is entered, it cannot be re-entered until at least one program instruction is executed.

To perform single-step execution, program one of the external interrupts (for example,INT0) to be level-sensitive and write an ISR for that interrupt the terminates as follows:

```
    JNB  TCON.1,$   ; wait for high on INT0# pin
    JB   TCON.1,$   ; wait for low on INT0# pin
    RETI            ; return for ISR
```

The CPU enters the ISR when the INT0# pin goes low, then waits for a pulse on INT0#. Each time INT0# is pulsed, the CPU exits the ISR, executes one program instruction, then re-enters the ISR.

18.5 Reset

The 8051 RESET pin is internally connected to an EZ-USB FX register bit that is controllable through the USB host. See *Chapter 13. "EZ-USB FX Resets"* for details.

18.6 Power Saving Modes

18.6.1 Idle Mode

An instruction that sets the IDLE Bit (PCON.0) causes the 8051 to enter idle mode when that instruction completes. In idle mode, CPU processing is suspended, and internal registers maintain their current data. When the 8051 core is in idle, the USB core enters suspend mode and shuts down the 24 MHz oscillator. See *Chapter 14. "EZ-USB FX Power Management"* for a full description of the Suspend/Resume process.

Table 18-26. PCON Register — SFR 87h

Bit	Function
PCON.7	**SMOD0** - Serial Port 0 baud rate double enable. When SMOD0 = 1, the baud rate for Serial Port 0 is doubled.
PCON.6-4	**Reserved**.
PCON.3	**GF1** - General purpose flag 1. Bit-addressable, general purpose flag for software control.
PCON.2	**GF0** - General purpose flag 0. Bit-addressable, general purpose flag for software control.
PCON.1	This bit should always be set to 0.
PCON.0	**IDLE** - Idle mode select. Setting the IDLE Bit places the 8051 in idle mode.

If the EZ-USB FX WAKEUP# pin is tied low, setting PCON.0 high does not put the 8051 into IDLE state.

EZ-USB FX Register Summary

The following table is a summary of all the EZ-USB FX Registers.

EZ-USB FX Register Summary

Addr	Name	Description	D7	D6	D5	D4	D3	D2	D1	D0	Default	Access	Notes		
		FIFO A-IN													
7800	AINDATA	Read Data from FIFO A	D7	D6	D5	D4	D3	D2	D1	D0	xxxxxxxx	R	R, r = read-only,		
7801	AINBC	Input FIFO A Byte Count	0	D6	D5	D4	D3	D2	D1	D0	00000000	R			
7802	AINPF	FIFO A-IN Programmable Flag (internal bit)	LTGT	D6	D5	D4	D3	D2	D1	D0	00100000	RW	Default: half empty		
7803	AINPFPIN	FIFO A-IN Programmable Flag (external pin)	LTGT	D6	D5	D4	D3	D2	D1	D0	00000000	RW	LTGT = 0: Flag is true (1) if Bytes in FIFO <= BCNT		
7804		(reserved)											LTGT = 1: Flag is true (1) if Bytes in FIFO >= BCNT		
		FIFO B-IN											Default: empty		
7805	BINDATA	Read Data from FIFO B	D7	D6	D5	D4	D3	D2	D1	D0	xxxxxxxx	R	W, w = write-only		
7806	BINBC	Input FIFO B Byte Count	0	D6	D5	D4	D3	D2	D1	D0	00000000	R			
7807	BINPF	FIFO B-IN Programmable Flag (internal bit)	LTGT	D6	D5	D4	D3	D2	D1	D0	00100000	RW	Default: half empty		
7808	BINPFPIN	FIFO B-IN Programmable Flag (external pin)	LTGT	D6	D5	D4	D3	D2	D1	D0	00000000	RW	LTGT = 0: Flag is true (1) if Bytes in FIFO <= BCNT		
7809		(reserved)											LTGT = 1: Flag is true (1) if Bytes in FIFO >= BCNT		
		FIFO A/B-IN Control											Default: empty		
780A	ABINCS	Input FIFOS Toggle control and flags	INTOG	INSEL	AINPF	AINEF	AINFF	BINPF	BINEF	BINFF	01110110	bbrrrrrr	FF=Full Flag, EF=Empty Flag, PF=Programmable Flag		
780B	ABINIE	Input FIFO Interrupt Enables	0	0	AINPF	AINEF	AINFF	BINPF	BINEF	BINFF	00000000	RW	INSEL: 1=A-FIFO, 0=B-FIFO		
780C	ABINIRQ	Input FIFO Interrupt Requests	0	0	AINPF	AINEF	AINFF	BINPF	BINEF	BINFF	xxxxxxxx	RW			
780D		(reserved)													
		FIFO A-OUT													
780E	AOUTDATA	Load Output FIFO A	D7	D6	D5	D4	D3	D2	D1	D0	xxxxxxxx	W			
780F	AOUTBC	Output FIFO A Byte Count	0	D6	D5	D4	D3	D2	D1	D0	00000000	R			
7810	AOUTPF	FIFO A-OUT Programmable Flag (internal bit)	LTGT	D6	D5	D4	D3	D2	D1	D0	10100000	RW	"INT" suffix means internal, 8051-accessible bits. Default: half-full		
7811	AOUTPFPIN	FIFO A-OUT Programmable Flag (external pin)	LTGT	D6	D5	D4	D3	D2	D1	D0	11000000	RW	Default: full		
7812		(reserved)													
		FIFO B-OUT													
7813	BOUTDATA	Load Output FIFO B	D7	D6	D5	D4	D3	D2	D1	D0	xxxxxxxx	W	RW = Read or Write,		
7814	BOUTBC	Output FIFO B Byte Count	0	D6	D5	D4	D3	D2	D1	D0	00000000	R			
7815	BOUTPF	FIFO B-OUT Programmable Flag (internal bit)	LTGT	D6	D5	D4	D3	D2	D1	D0	10100000	RW	"PIN" suffix means external pin flags. Default: half-full		
7816	BOUTPFPIN	FIFO B-OUT Programmable Flag (external pin)	LTGT	D6	D5	D4	D3	D2	D1	D0	11000000	RW	Default: full		
7817		(reserved)													
		FIFO A/B OUT Control													
7818	ABOUTCS	Output FIFOS Toggle control and flags	OUTTOG	OUTSEL	AOUTPF	AOUTEF	AOUTFF	BOUTPF	BOUTEF	BOUTFF	01010010	RW	PF=Programmable Flag,		
7819	ABOUTIE	Output FIFO Interrupt Enables	0	0	AOUTPF	AOUTEF	AOUTFF	BOUTPF	BOUTEF	BOUTFF	00000000	RW			
781A	ABOUTIRQ	Output FIFO Interrupt Requests	0	0	AOUTPF	AOUTEF	AOUTFF	BOUTPF	BOUTEF	BOUTFF	xxxxxxxx	RW			
781B		(reserved)													
		FIFO A/B Global Control													

EZ-USB FX Register Summary

Addr	Name	Description	D7	D6	D5	D4	D3	D2	D1	D0	Default	Access	Notes		
781C	ABSETUP	FIFO Setup	0	0	ASYNC	DBLIN	0	OUTDLY	0	DBLOUT	00000000	RW	ASYNC=1: async FIFOS, DBL=1:double stuff,OUTDLY=1: clock delay		
781D	ABPOLAR	FIFO Control Signals Polarity	0	0	BOE	AOE	SLRD	SLWR	ASEL	BSEL	00000000	RW	0=active LO, 1-active HI		
781E	ABFLUSH	Write (data=x) to reset all flags	x	x	x	x	x	x	x	x	xxxxxxxx	W	Flag reset: Empty=1, Full=0, PF=?		
781F		(reserved)													
7820		(reserved)													
7821		(reserved)													
7822		(reserved)													
7823		(reserved)													
		GPIF													
7824	WFSELECT	Waveform Selector	SINGLEWR 0-3		SINGLERD 0-3		FIFOWR 0-3		FIFORD 0-3		11100100	RW	Select waveform 0[00], 1[01], 2[10] or 3[11]		
7825	IDLE_CS	GPIF Done, GPIF IDLE drive mode	DONE	0	0	0	0	0	0	IDLEDRV	10000000	RW	DONE=1: GPIF done (IRQ4). IDLEDRV=1:drive bus, 0:TS		
7826	IDLE_CTLOUT	Inactive Bus, CTL states	0	0	CTL5	CTL4	CTL3	CTL2	CTL1	CTL0	11111111	RW			
7827	CTLOUTCFG	CTL OUT pin drive	TRICTL	0	CTL5	CTL4	CTL3	CTL2	CTL1	CTL0	00000000	RW	0=CMOS, 1=open drn.		
7828		(reserved)													
7829		(reserved)													
782A	GPIFADRL	GPIF Address L	x	x	A5	A4	A3	A2	A1	A0	00000000	RW			
782B		(reserved)													
782C	AINTC	FIFO A IN T.C.	FITC	FIFO A IN Transaction Count [6:0]							00000001	RW	FITC=1: Use FIFO flags		
782D	AOUTTC	FIFO A OUT T.C.	FITC	FIFO A OUT Transaction Count [6:0]							00000001	RW	FITC=0: Use Transac Count.		
782E	ATRIG	***Write:*** write FIFO A. ***Read:*** start RD trans.	x	x	x	x	x	x	x	x	xxxxxxxx	RW	Note: read the data in ?		
782F		(reserved)													
7830	BINTC	FIFO B IN T.C.	FITC	FIFO B IN Transaction Count [6:0]							00000001	RW			
7831	BOUTTC	FIFO B OUT T.C.	FITC	FIFO B OUT Transaction Count [6:0]							00000001	RW			
7832	BTRIG	***Write***: write FIFO B***. Read:*** start RD trans.	x	x	x	x	x	x	x	x	xxxxxxxx	RW	?		
7833		(reserved)													
7834	SGLDATH	GPIF Data H (16-bit mode only)	D15	D14	D13	D12	D11	D10	D9	D8	xxxxxxxx	RW			
7835	SGLDATLTRIG	Read or Write GPIF Data L & trigger rd transac	D7	D6	D5	D4	D3	D2	D1	D0	xxxxxxxx	RW	Triggers a GPIF Read Waveform		
7836	SGLDATLNTRIG	Read GPIF Data L, no rd transac trigger	D7	D6	D5	D4	D3	D2	D1	D0	xxxxxxxx	R	No GPIF Waveform		
7837		(reserved)													
7838	READY	Internal RDY,Sync/Async, RDY pin states	INTRDY	SAS	RDY5	RDY4	RDY3	RDY2	RDY1	RDY0	00xxxxxx	bbrrrrrr	SAS=1: synchrnous, 0:asynchronous		
7839	ABORT	Abort GPIF cycles	x	x	x	x	x	x	x	x	xxxxxxxx	W	Go To GPIF IDLE state. Data is D.C.		
783A		(reserved)													
783B	GENIE		0	0	0	0	0	DMADN	GPWF	GPDONE	00000000	RW			
783C	GENIRQ		0	0	0	0	0	DMADN	GPWF	GPDONE	00000xxx	RW			
783D		(reserved)													
783E		(reserved)													

EZ-USB FX Register Summary

Addr	Name	Description	D7	D6	D5	D4	D3	D2	D1	D0	Default	Access	Notes		
783F		(reserved)													
7840		(reserved)													
		IO Ports D,E													
7841	OUTD	Output Port D	OUTD7	OUTD6	OUTD5	OUTD4	OUTD3	OUTD2	OUTD1	OUTD0	xxxxxxxx	W			
7842	PINSD	Input Port D pins	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	xxxxxxxx	R			
7843	OED	Port D Output Enable	0ED7	0ED6	0ED5	0ED4	0ED3	0ED2	0ED1	0ED0	00000000	RW			
7844		(reserved)													
7845	OUTE	Output Port E	OUTE7	OUTE6	OUTE5	OUTE4	OUTE3	OUTE2	OUTE1	OUTE0	xxxxxxxx	W			
7846	PINSE	Input Port E pins	PINE7	PINE6	PINE5	PINE4	PINE3	PINE2	PINE1	PINE0	xxxxxxxx	R			
7847	OEE	Port E Output Enable	OEE7	OEE6	OEE5	OEE4	OEE3	OEE2	OEE1	OEE0	00000000	RW			
7848		(reserved)													
7849	PORTSETUP	Timer0 Clock source, Port-to-SFR mapping	0	0	0	0	0	0	T0CLK	SFRPORT	00000000	RW	T0CLK (0) Normal Timer0 clock; (1) CPU clock/13		
784A	IFCONFIG	Select 8/16 bit data bus, confugure busses (IF)	*52ONE*	0	0		GSTATE	BUS16	IF1	IF0	00000000	brrrbbbb	SFRPORT (1) IO Ports SFR mapped, (0) not		
784B	PORTACF2	Port A GPIF signals	0	0	SLRD	SLWR	0	0	0	0	00000000	RW	*52ONE*: Set to 1 for 52-pin part (drive internal RDY signals HI)		
784C	PORTCCF2	Port C GPIF signals	CTL5	CTL4	CTL3	CTL1	RDY3	0	RDY1	RDY0	00000000	RW			
784D		(reserved)													
784E		(reserved)													
		DMA Control													
784F	DMASRCH	DMA SourceH	A15	A14	A13	A12	A11	A10	A9	A8	00000000	RW			
7850	DMASRCL	DMA Source L	A7	A6	A5	A4	A3	A2	A1	A0	00000000	RW			
7851	DMADESTH	DMA Destination H	A15	A14	A13	A12	A11	A10	A9	A8	00000000	RW			
7852	DMADESTL	DMA Destination L	A7	A6	A5	A4	A3	A2	A1	A0	00000000	RW			
7853		(reserved)													
7854	DMALEN	DMA Transfer Length	D7	D6	D5	D4	D3	D2	D1	D0	00000001	RW	0=256, 1=1…255=255		
7855	DMAGO	Start DMA Transfer	DONE	x	x	x	x	x	x	x	xxxxxxxx	rxxxxxxx	Write this register to start a DMA transfer		
7856		(reserved)											DSTR[2..0] set stretch values for external DMA transfers		
7857	DMABURST		x	x	x	DSTR2	DSTR1	DSTR0	RB	WB	00000100	RW	RB/WB enable synchronous burst transfers on 8051 data bus		
7858	DMAEXTFIFO		n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	xxxxxxxx	N/A	Use this DMA address to select 8051 A/D busses as external FIFO		
7859		(reserved)													
785A		(reserved)											Note: DSTRn are stretch values for DMA FRD# and FWR# signals		
785B		(reserved)													
785C		(reserved)													
		Interrupt 4 Vector Control													
785D	INT4IVEC	Interrupt 4 Vector	0	1	I4V3	I4V2	I4V1	I4V0	0	0	01000000	R	See bottom of page 2 for vector coding		
785E	INT4SETUP	Interrupt 4 Setup	0	0	0	0	0	INT4SFC	INTERNAL	AV4EN	00000000	RW	INTERNAL: 0-INT4 from pin, 1-INT4 from FIFO unit		
785F-78FF		(reserved)													

EZ-USB FX Register Summary

Addr	Name	Description	D7	D6	D5	D4	D3	D2	D1	D0	Default	Access	Notes		
7900	WFDESC(0)	**Waveform Descriptors (128 bytes)**									xxxxxxxx	RW			
797F	WFDESC(127)	(last wafeform descriptor byte)													
7980-7B3F		(reserved)													
		Endpoint 0-7 Data Buffers											R=RD, W=WR, RW or b=BOTH		
7B40	OUT7BUF	(64 bytes)	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7B80	IN7BUF	(64 bytes)	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7BC0	OUT6BUF	(64 bytes)	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW	[6] EP6OUT and EP7OUT pairable for double buffering		
7C00	IN6BUF	(64 bytes)	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW	[5] EP6IN and EP7IN pairable for double buffering		
7C40	OUT5BUF	(64 bytes)	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7C80	IN5BUF	(64 bytes)	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7CC0	OUT4BUF	(64 bytes)	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW	[4] EP4OUT and EP5OUT pairable for double buffering		
7D00	IN4BUF	(64 bytes)	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW	[3] EP4IN and EP5IN pairable for double buffering		
7D40	OUT3BUF	(64 bytes)	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7D80	IN3BUF	(64 bytes)	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7DC0	OUT2BUF	(64 bytes)	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW	[2] EP2-IN and EP3IN pairable for double buffering		
7E00	IN2BUF	(64 bytes)	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW	[1] EP2-IN and EP3IN pairable for double buffering		
7E40	OUT1BUF	(64 bytes)	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7E80	IN1BUF	(64 bytes)	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7EC0	OUT0BUF	(64 bytes)	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7F00	IN0BUF	(64 bytes)	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7F40-7F5F		(reserved)													
		Isochronous Data													
7F60	OUT8DATA	Endpoint 8 OUT Data	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	R			
7F61	OUT9DATA	Endpoint 9 OUT Data	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	R			
7F62	OUT10DATA	Endpoint 10 OUT Data	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	R			
7F63	OUT11DATA	Endpoint 11 OUT Data	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	R			
7F64	OUT12DATA	Endpoint 12 OUT Data	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	R			
7F65	OUT13DATA	Endpoint 13 OUT Data	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	R			
7F66	OUT14DATA	Endpoint 14 OUT Data	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	R			
7F67	OUT15DATA	Endpoint 15 OUT Data	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	R			
7F68	IN8DATA	Endpoint 8 IN Data	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	W			
7F69	IN9DATA	Endpoint 9 IN Data	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	W			
7F6A	IN10DATA	Endpoint 10 IN Data	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	W			
7F6B	IN11DATA	Endpoint 11 IN Data	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	W			
7F6C	IN12DATA	Endpoint 12 IN Data	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	W			

EZ-USB FX Register Summary

Addr	Name	Description	D7	D6	D5	D4	D3	D2	D1	D0	Default	Access	Notes		
7F6D	IN13DATA	Endpoint 13 IN Data	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	W			
7F6E	IN14DATA	Endpoint 14 IN Data	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	W			
7F6F	IN15DATA	Endpoint 15 IN Data	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	W			
		Isochronous Byte Counts													
7F70	OUT8BCH	EP8 Out Byte Count H	0	0	0	0	0	0	d9	d8	xxxxxxxx	R			
7F71	OUT8BCL	EP8 Out Byte Count L	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	R			
7F72	OUT9BCH	EP9 Out Byte Count H	0	0	0	0	0	0	d9	d8	xxxxxxxx	R			
7F73	OUT9BCL	EP9 Out Byte Count L	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	R			
7F74	OUT10BCH	EP10 Out Byte Count H	0	0	0	0	0	0	d9	d8	xxxxxxxx	R			
7F75	OUT10BCL	EP10 Out Byte Count L	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	R			
7F76	OUT11BCH	EP11 Out Byte Count H	0	0	0	0	0	0	d9	d8	xxxxxxxx	R			
7F77	OUT11BCL	EP11 Out Byte Count L	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	R			
7F78	OUT12BCH	EP12 Out Byte Count H	0	0	0	0	0	0	d9	d8	xxxxxxxx	R			
7F79	OUT12BCL	EP12 Out Byte Count L	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	R			
7F7A	OUT13BCH	EP13 Out Byte Count H	0	0	0	0	0	0	d9	d8	xxxxxxxx	R			
7F7B	OUT13BCL	EP13 Out Byte Count L	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	R			
7F7C	OUT14BCH	EP14 Out Byte Count H	0	0	0	0	0	0	d9	d8	xxxxxxxx	R			
7F7D	OUT14BCL	EP14 Out Byte Count L	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	R			
7F7E	OUT15BCH	EP15 Out Byte Count H	0	0	0	0	0	0	d9	d8	xxxxxxxx	R			
7F7F	OUT15BCL	EP15 Out Byte Count L	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	R			
7F80-7F91		(reserved)													
		CPU Registers													
7F92	CPUCS	Control & Status	rv3	rv2	rv1	rv0	24/48	CLKINV	CLKOE	8051RES	00000010	rrrrrrbr	Bits 7:2 are chip rev number		
7F93	PORTACFG	Port A Configuration	RxD1out	RxD0out	FRD	FWR	CS	OE	T1out	T0out	00000000	RW	CFG: 0=port (default), 1=alternate function		
7F94	PORTBCFG	Port B Configuration	T2OUT	INT6	INT5	INT4	TxD1	RxD1	T2EX	T2	00000000	RW			
7F95	PORTCCFG	Port C Configuration	RD	WR	T1	T0	INT1	INT0	TxD0	RxD0	00000000	RW			
		Input-Output Port Registers													
7F96	OUTA	Output Register A	OUTA7	OUTA6	OUTA5	OUTA4	OUTA3	OUTA2	OUTA1	OUTA0	00000000	RW	WR: output FF; RD: register outputs		
7F97	OUTB	Output Register B	OUTB7	OUTB6	OUTB5	OUTB4	OUTB3	OUTB2	OUTB1	OUTB0	00000000	RW			
7F98	OUTC	Output Register C	OUTC7	OUTC6	OUTC5	OUTC4	OUTC3	OUTC2	OUTC1	OUTC0	00000000	RW			
7F99	PINSA	Port Pins A	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	xxxxxxxx	R	WR: no effect; RD: pin states (reg if OE=1, pin if OE=0)		
7F9A	PINSB	Port Pins B	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	xxxxxxxx	R			
7F9B	PINSC	Port Pins C	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	xxxxxxxx	R			
7F9C	OEA	Output Enable A	OEA7	OEA6	OEA5	OEA4	OEA3	OEA2	OEA1	OEA0	00000000	RW	1=output enabled		
7F9D	OEB	Output Enable B	OEB7	OEB6	OEB5	OEB4	OEB3	OEB2	OEB1	OEB0	00000000	RW			
7F9E	OEC	Output Enable C	OEC7	OEC6	OEC5	OEC4	OEC3	OEC2	OEC1	OEC0	00000000	RW	RW pins enabled		
7F9F		reserve													

EZ-USB FX Register Summary

Addr	Name	Description	D7	D6	D5	D4	D3	D2	D1	D0	Default	Access	Notes		
		Isochronous Control/Status Registers													
7FA0	ISOERR	ISO OUT Endpoint Error	ISO15ERR	ISO14ERR	ISO13ERR	ISO12ERR	ISO11ERR	ISO10ERR	ISO9ERR	ISO8ERR	xxxxxxxx	R	CRC Error		
7FA1	ISOCTL	Isochronous Control	*	*	*	*	PPSTAT	MBZ	MBZ	ISODISAB	0000x000	rrrrrbbb			
7FA2	ZBCOUT	Zero Byte Count bits	EP15	EP14	EP13	EP12	EP11	EP10	EP9	EP8	xxxxxxxx	R			
7FA3		(reserved)													
7FA4		(reserved)													
		I^{2}C Registers													
7FA5†	I2CS	Control & Status	START	STOP	LASTRD	ID1	ID0	BERR	ACK	DONE	000xx000	bbbrrrrr			
7FA6†	I2DAT	Data	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7FA7†	I2CMODE	I2C STOP interrupt enable	0	0	0	0	0	0	STOPIE	400kHz	00000000	RW			
		Interrupts													
7FA8	IVEC	Interrupt Vector	0	IV4	IV3	IV2	IV1	IV0	0	0	00000000	R			
7FA9†	IN07IRQ	EPIN Interrupt Request	IN7IR	IN6IR	IN5IR	IN4IR	IN3IR	IN2IR	IN1IR	IN0IR	00000000	RW	1 = clear request, 0= no effect		
7FAA†	OUT07IRQ	EPOUT Interrupt Request	OUT7IR	OUT6IR	OUT5IR	OUT4IR	OUT3IR	OUT2IR	OUT1IR	OUT0IR	xxxxxxxx	RW	1 = clear request, 0= no effect		
7FAB†	USBIRQ	USB Interrupt Request	0	0	IBNIR	URESIR	SUSPIR	SUTOKIR	SOFIR	SUDAVIR	xxxxxxxx	RW	1 = clear request, 0= no effect		
7FAC†	IN07IEN	EP0-7IN Int Enables	IN7IEN	IN6IEN	IN5IEN	IN4IEN	IN3IEN	IN2IEN	IN1IEN	IN0IEN	00000000	RW	1=enabled, 0=disabled		
7FAD†	OUT07IEN	EP0-7OUT Int Enables	OUT7IEN	OUT6IEN	OUT5IEN	OUT4IEN	OUT3IEN	OUT2IEN	OUT1IEN	OUT0IEN	00000000	RW	1=enabled, 0=disabled		
7FAE†	USBIEN	USB Int Enables	0	0	IBNIE	URESIE	SUSPIE	SUTOKIE	SOFIE	SUDAVIE	00000000	RW	1=enabled, 0=disabled		
7FAF	USBBAV	Breakpoint & Autovector	*****	*****	*****	INT2SFC	BREAK	BPPULSE	BPEN	AVEN	xxx0xx00	RW	Breakpoint and Autovector		
7FB0†	IBNIRQ	IBN Interrupt request		EP6IN	EP5IN	EP4IN	EP3IN	EP2IN	EP1IN	EP0IN	xxxxxxxx	RW			
7FB1†	IBNIE	IBN Interrupt Enable		EP6IN	EP5IN	EP4IN	EP3IN	EP2IN	EP1IN	EP0IN	00000000	RW			
7FB2	BPADDRH	Breakpoint Address H	A15	A14	A13	A12	A11	A10	A9	A8	00000000	RW			
7FB3	BPADDRL	Breakpoint Address L	A7	A6	A5	A4	A3	A2	A1	A0	00000000	RW			
		Bulk Endpoints 0-7													
7FB4†	EP0CS	Control & Status	*	*	*	*	OUTBSY	INBSY	HSNAK	EP0STALL	00001000	rrrrrrbb			
7FB5†	IN0BC	Byte Count	*	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7FB6†	IN1CS	Control & Status	*	*	*	*	*	*	in1bsy	in1stl	00000000	rrrrrrbb			
7FB7†	IN1BC	Byte Count	*	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7FB8†	IN2CS	Control & Status	*	*	*	*	*	*	in2bsy	in2stl	00000000	rrrrrrbb			
7FB9†	IN2BC	Byte Count	*	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7FBA†	IN3CS	Control & Status	*	*	*	*	*	*	in3bsy	in3stl	00000000	rrrrrrbb			
7FBB†	IN3BC	Byte Count	*	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7FBC†	IN4CS	Control & Status	*	*	*	*	*	*	in4bsy	in4stl	00000000	rrrrrrbb			
7FBD†	IN4BC	Byte Count	*	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7FBE†	IN5CS	Control & Status	*	*	*	*	*	*	in5bsy	in5stl	00000000	rrrrrrbb			
7FBF†	IN5BC	Byte Count	*	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			

† **Read/write latency note:** These registers need the equivalent of 2 instruction clock cycles of time between performing the following instructions back-to-back: (1) write-write (2) write-read.

EZ-USB FX Register Summary

Addr	Name	Description	D7	D6	D5	D4	D3	D2	D1	D0	Default	Access	Notes		
7FC0†	IN6CS	Control & Status	*	*	*	*	*	*	in6bsy	in6stl	00000000	rrrrrrbb			
7FC1†	IN6BC	Byte Count	*	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7FC2†	IN7CS	Control & Status	*	*	*	*	*	*	in7bsy	in7stl	00000000	rrrrrrbb			
7FC3†	IN7BC	Byte Count	*	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7FC4		(reserved)													
7FC5†	OUT0BC	Byte Count	*	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7FC6†	OUT1CS	Control & Status	*	*	*	*	*	*	out1bsy	out1stl	00000010	rrrrrrrb			
7FC7†	OUT1BC	Byte Count	*	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7FC8†	OUT2CS	Control & Status	*	*	*	*	*	*	out2bsy	out2stl	00000010	rrrrrrrb			
7FC9†	OUT2BC	Byte Count	*	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7FCA†	OUT3CS	Control & Status	*	*	*	*	*	*	out3bsy	out3stl	00000010	rrrrrrrb			
7FCB†	OUT3BC	Byte Count	*	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7FCC†	OUT4CS	Control & Status	*	*	*	*	*	*	out4bsy	out4stl	00000010	rrrrrrrb			
7FCD†	OU4TBC	Byte Count	*	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7FCE†	OUT5CS	Control & Status	*	*	*	*	*	*	out5bsy	out5stl	00000010	rrrrrrrb			
7FCF†	OUT5BC	Byte Count	*	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7FD0†	OUT6CS	Control & Status	*	*	*	*	*	*	out6bsy	out6stl	00000010	rrrrrrrb			
7FD1†	OUT6BC	Byte Count	*	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
7FD2†	OUT7CS	Control & Status	*	*	*	*	*	*	out7bsy	out7stl	00000010	rrrrrrrb			
7FD3†	OUT7BC	Byte Count	*	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	RW			
		Global USB Registers													
7FD4†	SUDPTRH	Setup Data Ptr H	A15	A14	A13	A12	A11	A10	A9	A8	xxxxxxxx	RW			
7FD5†	SUDPTRL	Setup Data Ptr L	A7	A6	A5	A4	A3	A2	A1	A0	xxxxxxxx	RW			
7FD6†	USBCS	USB Control & Status	WakeSRC	*	*	*	DisCon	DiscOE	ReNum	SIGRSUME	00000100	brrrbbbb			
7FD7†	TOGCTL	Toggle Control	Q	S	R	IO	0	EP2	EP1	EP0	xxxxxxxx	rbbbbbbb	Endpoint Toggle bits		
7FD8	USBFRAMEL	Frame Number L	FC7	FC6	FC5	FC4	FC3	FC2	FC1	FC0	xxxxxxxx	R			
7FD9	USBFRAMEH	Frame Number H	0	0	0	0	0	FC10	FC9	FC8	xxxxxxxx	R			
7FDA		(reserved)													
7FDB	FNADDR	Function Address	0	FA6	FA5	FA4	FA3	FA2	FA1	FA0	xxxxxxxx	R			
7FDC		(reserved)													
7FDD†	USBPAIR	Endpoint Control	ISOsend0	*	PR6OUT	PR4OUT	PR2OUT	PR6IN	PR4IN	PR2IN	0x000000	RW	1 = Pair endpoint (double-buffer)		
7FDE†	IN07VAL	Input Endpoint 0-7 valid	IN7VAL	IN6VAL	IN5VAL	IN4VAL	IN3VAL	IN2VAL	IN1VAL	1	01010111	RW	1=valid, 0=not valid		
7FDF†	OUT07VAL	Output Endpoint 0-7 valid	OUT7VAL	OUT6VAL	OUT5VAL	OUT4VAL	OUT3VAL	OUT2VAL	OUT1VAL	1	01010101	RW	(a not-valid EP returns 'no response' instead of NAK)		
7FE0†	INISOVAL	Input EP 8-15 valid	IN15VAL	IN14VAL	IN13VAL	IN12VAL	IN11VAL	IN10VAL	IN9VAL	IN8VAL	00000111	RW	NOTE: EP0 is always valid		
7FE1†	OUTISOVAL	Output EP 8-15 valid	OUT15VAL	OUT14VAL	OUT13VAL	OUT12VAL	OUT11VAL	OUT10VAL	OUT9VAL	OUT8VAL	00000111	RW			
7FE2	FASTXFR	Fast Transfer Mode	FISO	FBLK	RPOL	RMOD1	RMOD0	WPOL	WMOD1	WMOD0	xxxxxxxx	RW			

† **Read/write latency note:** These registers need the equivalent of 2 instruction clock cycles of time between performing the following instructions back-to-back: (1) write-write (2) write-read.

EZ-USB FX Register Summary

Addr	Name	Description	D7	D6	D5	D4	D3	D2	D1	D0	Default	Access	Notes		
7FE3	AUTOPTRH	Auto-Pointer H	A15	A14	A13	A12	A11	A10	A9	A8	xxxxxxxx	RW			
7FE4	AUTOPTRL	Auto-Pointer L	A7	A6	A5	A4	A3	A2	A1	A0	xxxxxxxx	RW			
7FE5	AUTODATA	Auto Pointer Data	D7	D6	D5	D4	D3	D2	D1	D0	xxxxxxxx	RW			
7FE6		(reserved)													
7FE7		(reserved)													
		Setup Data													
7FE8	SETUPDAT	8 bytes of SETUP data	d7	d6	d5	d4	d3	d2	d1	d0	xxxxxxxx	R	3 LSB's of address are 000		
		Isochronous FIFO Sizes													
7FF0	OUT8ADDR	Endpt 8 OUT Start Addr	A9	A8	A7	A6	A5	A4	0	0	xxxxxxxx	RW			
7FF1	OUT9ADDR	Endpt 9 OUT Start Addr	A9	A8	A7	A6	A5	A4	0	0	xxxxxxxx	RW			
7FF2	OUT10ADDR	Endpt 10 OUT Start Addr	A9	A8	A7	A6	A5	A4	0	0	xxxxxxxx	RW			
7FF3	OUT11ADDR	Endpt 11 OUT Start Addr	A9	A8	A7	A6	A5	A4	0	0	xxxxxxxx	RW			
7FF4	OUT12ADDR	Endpt 12 OUT Start Addr	A9	A8	A7	A6	A5	A4	0	0	xxxxxxxx	RW			
7FF5	OUT13ADDR	Endpt 13 OUT Start Addr	A9	A8	A7	A6	A5	A4	0	0	xxxxxxxx	RW			
7FF6	OUT14ADDR	Endpt 14 OUT Start Addr	A9	A8	A7	A6	A5	A4	0	0	xxxxxxxx	RW			
7FF7	OUT15ADDR	Endpt 15 OUT Start Addr	A9	A8	A7	A6	A5	A4	0	0	xxxxxxxx	RW			
7FF8	IN8ADDR	Endpt 8 IN Start Addr	A9	A8	A7	A6	A5	A4	0	0	xxxxxxxx	RW			
7FF9	IN9ADDR	Endpt 9 IN Start Addr	A9	A8	A7	A6	A5	A4	0	0	xxxxxxxx	RW			
7FFA	IN19ADDR	Endpt 10 IN Start Addr	A9	A8	A7	A6	A5	A4	0	0	xxxxxxxx	RW			
7FFB	IN11ADDR	Endpt 11 IN Start Addr	A9	A8	A7	A6	A5	A4	0	0	xxxxxxxx	RW			
7FFC	IN12ADDR	Endpt 12 IN Start Addr	A9	A8	A7	A6	A5	A4	0	0	xxxxxxxx	RW			
7FFD	IN13ADDR	Endpt 13 IN Start Addr	A9	A8	A7	A6	A5	A4	0	0	xxxxxxxx	RW			
7FFE	IN14ADDR	Endpt 14 IN Start Addr	A9	A8	A7	A6	A5	A4	0	0	xxxxxxxx	RW			
7FFF	IN15ADDR	Endpt 15 IN Start Addr	A9	A8	A7	A6	A5	A4	0	0	xxxxxxxx	RW			

EZ-USB FX Register Summary

		Interrupt 4 Sources and INT4IVEC values													
	20	AINPF	**0**	**1**	0	0	0	0	0	0					
	21	BINPF	**0**	**1**	0	0	0	1	0	0					
	22	AOUTPF	**0**	**1**	0	0	1	0	0	0					
	23	BOUTPF	**0**	**1**	0	0	1	1	0	0					
	24	AINEF	**0**	**1**	0	1	0	0	0	0					
	25	BINEF	**0**	**1**	0	1	0	1	0	0					
	26	AOUTEF	**0**	**1**	0	1	1	0	0	0					
	27	BOUTEF	**0**	**1**	0	1	1	1	0	0					
	28	AINFF	**0**	**1**	1	0	0	0	0	0					
	29	BINFF	**0**	**1**	1	0	0	1	0	0					
	2A	AOUTFF	**0**	**1**	1	0	1	0	0	0					
	2B	BOUTFF	**0**	**1**	1	0	1	1	0	0					
	2C	GPIFDONE	**0**	**1**	1	1	0	0	0	0					
	2D	GPIFWF	**0**	**1**	1	1	0	1	0	0					
	2E	DMADONE	**0**	**1**	1	1	1	0	0	0					
		New Interrupt 2 source and vector													
	05	IBN	0	0	0	1	0	1	0	0			Bulk IN token arrived and was NAK'd		