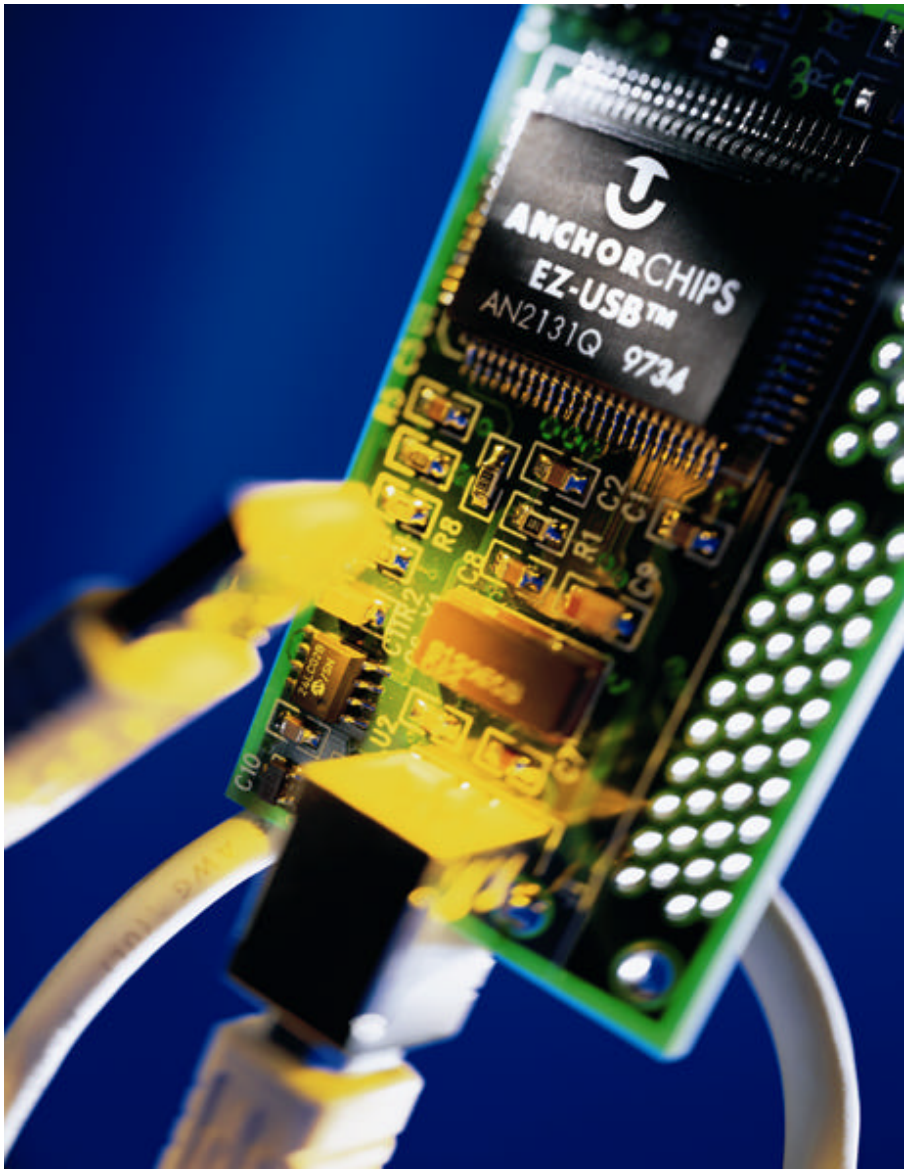
EZ-USB Technical Reference Manual
Version 1.51



Anchor Chips Incorporated

12396 World Trade Drive
M/S 212
San Diego, CA 92128
(619) 613-7900 Fax (619) 676-6896

The EZ-USB Technical Reference Manual

Version 1.51

Table of Contents

Figures

Tables

1 Introducing EZ-USB

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 IO 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 1.0" (**http://usb.org**), a 268 page document which describes all aspects of a USB device in elaborate detail. This EZ-USB Technical Reference Manual describes the EZ-USB chip along with USB topics that should provide help in understanding the Specification.

The Anchor Chips EZ-USB is a compact integrated circuit that provides a highly integrated solution for a USB peripheral device. Three key EZ-USB features are:

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

This chapter introduces some key USB concepts and terminology that should make reading the rest of this Technical Reference Manual easier.

1.2 EZ-USB Block Diagrams

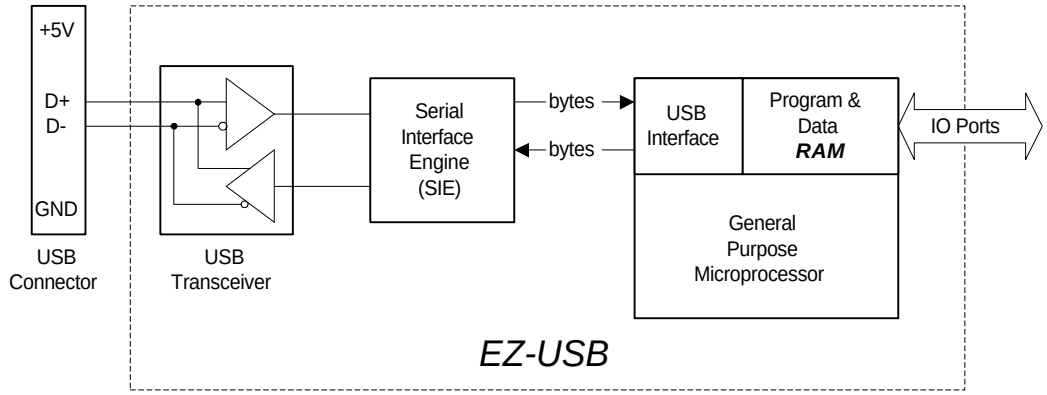


Figure 1-1. AN2131S (44 pin) Simplified Block Diagram

The Anchor Chips EZ-USB 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, and ultimately transfers data bytes to and from the USB interface.

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

The EZ-USB family uses an enhanced SIE/USB interface (called the Anchor "core") which has the intelligence to function as a full USB device even before the 8051 is running. This allows the USB host to download code into EZ-USB RAM, and then start the 8051. The enhanced core simplifies 8051 code by implementing much of the USB protocol itself.

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

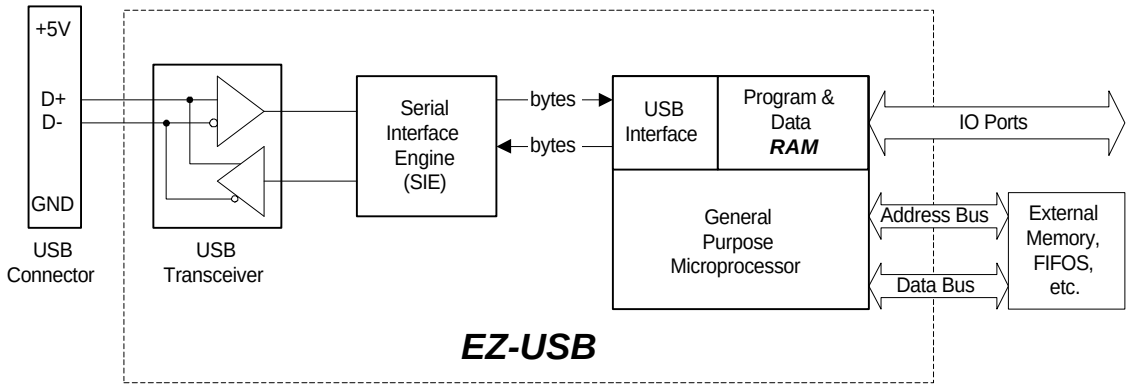


Figure 1-2. AN2131Q (80 pin) Simplified Block Diagram

Figure 1-2 illustrates the AN2131Q, an 80-pin version of the EZ-USB family. In addition to the 24 IO pins, it contains a 16-bit address bus and 8-bit data bus for external memory expansion.

A special "fast transfer" mode moves data directly between external logic and internal USB FIFOS. The fast transfer mode, along with abundant endpoint resources, allows the EZ-USB family to support transfer bandwidths beyond the maximum required by the USB specification.

1.3 The USB Specification

The USB Specification Version 1.0 is available on the Internet at **http://usb.org**. Published in January 1996, the 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 implementers assures 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 spec 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 spec even contains detail about the connector types and wire colors.

1.4 Tokens and PIDS

In this manual, you'll read statements like, "When the host sends an IN token…". Or, "The device responds with a ACK". What do these terms mean?

A USB transaction consists of data packets, identified by special codes called Packet ID's or PIDS. A PID signifies what kind of packet is being transmitted. There are four PID types, as 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

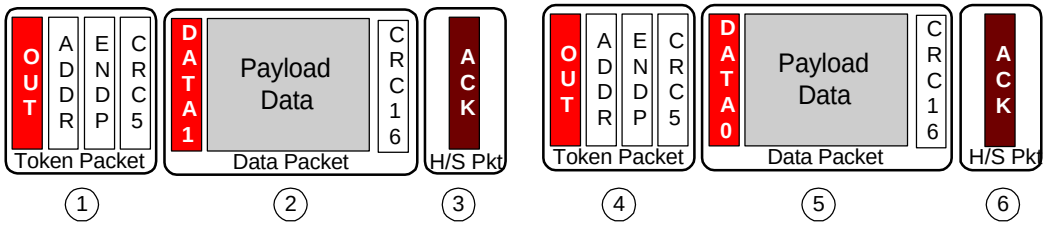


Figure 1-3. USB Packets

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? Because the USB architects took error correction very seriously. As mentioned above, the ACK handshake is a signal to the host that the peripheral received data without error (the CRC portions of the packet are used to detect errors). But what if a 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 mis-communication 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 isn’t there. It’s like “halt”, but better, since USB provides a way to recover from a stall.

A PRE (Preamble) PID precedes a low-speed (1.5 Mbit/sec) USB transmission. The EZ-USB family supports high speed (12 Mbit/sec) USB transfers only, so it ignores PRE packets and the subsequent 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.

Actually, 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. But that’s the only way to yank the host’s chain. 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 instead sends a NAK. If it finds an error, it sends nothing back. For the latter two cases, the host re-sends 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 chip, there’s nothing to stop the 8051 from loading data for the host into an endpoint

buffer (Section 1.13) and "arming" it for transfer. But the data will sit in the buffer *until the host sends an IN token to that particular endpoint.* If the host never sends the IN token, the data sits there forever.

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 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, since the 8051 *sends* data by loading an IN endpoint buffer, but keeping in mind that an 8051 "out" is an IN to the host, it makes sense.

1.7 Frame

The USB host provides a time base to all USB devices by transmitting an 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 registers. SOF-time has significance for isochronous endpoints; it's the time that the "ping-ponging" buffers switch places. The EZ-USB core provides the 8051 with an SOF interrupt request for servicing isochronous endpoint data.

1.8 USB Transfer Types

USB defines four transfer types. These match the requirements of different data types delivered over the bus. (Section 1.13, "Endpoints", explains how the EZ-USB family supports the four transfer types.)

1.8.1 Bulk Transfers

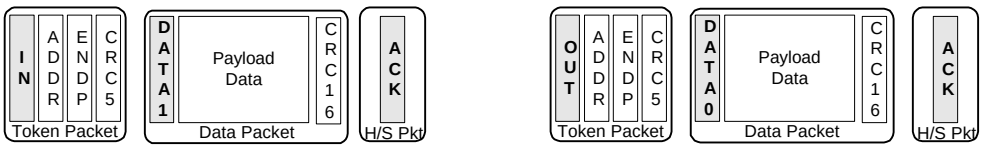


Figure 1-4. Two bulk transfers, IN and OUT

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

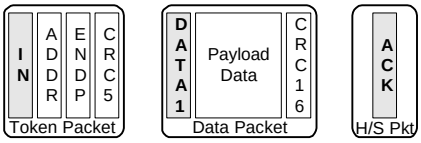


Figure 1-5. An Interrupt Transfer

Interrupt data is like bulk data, but exists only for IN endpoints in the Version 1.0 USB Specification. Interrupt data can have packet sizes of 1-64 bytes. Interrupt endpoints have an associated polling interval that insures that they will be "pinged" (will receive an IN token) by the host on a regular basis.

Note:
At this writing, OUT interrupt endpoint capability is added in the draft version of the USB 1.1 Specification. Due to the general purpose nature of the EZ-USB bulk/interrupt endpoints, the EZ-USB family also accommodates OUT interrupt endpoints.

1.8.3 Isochronous Transfers

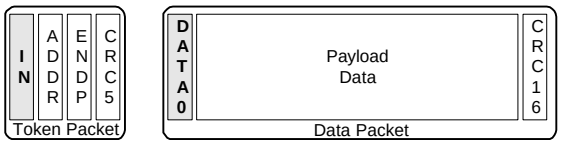


Figure 1-6. An Isochronous transfer

Isochronous data is time-critical, used for "streaming" 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

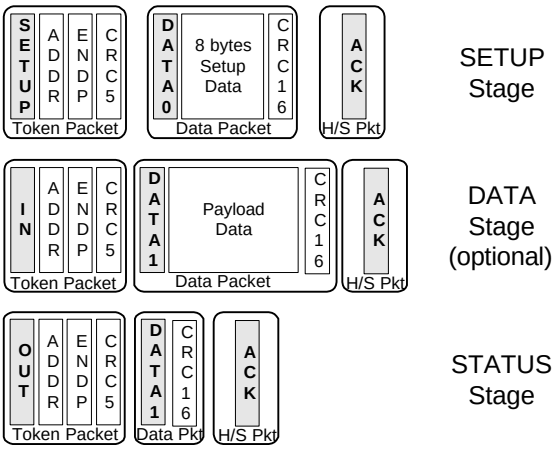


Figure 1-7. A Control Transfer

Control transfers are used to configure and send commands to a device. Being "mission critical", they employ the most extensive error checking USB offers. Control transfers are delivered on a "best effort" basis by the host ("best effort" is defined by a six step process in the USB Specification, 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 Windows™ cursor switches to an hourglass, and then back to a cursor. And magically, your device is connected and its Windows driver is loaded! Anyone who has installed a sound card into a PC and futzed with jumpers, drivers and IO/Interrupt/DMA settings knows that a USB connection can be akin to a miracle. We've all heard about "plug and play", and USB delivers the real deal.

How does all this happen automatically? Inside every USB device is a table of 'descriptors' that are 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 dutifully responds to this request by sending ID bytes data back to the host telling what it is.

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 for 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 Anchor Core

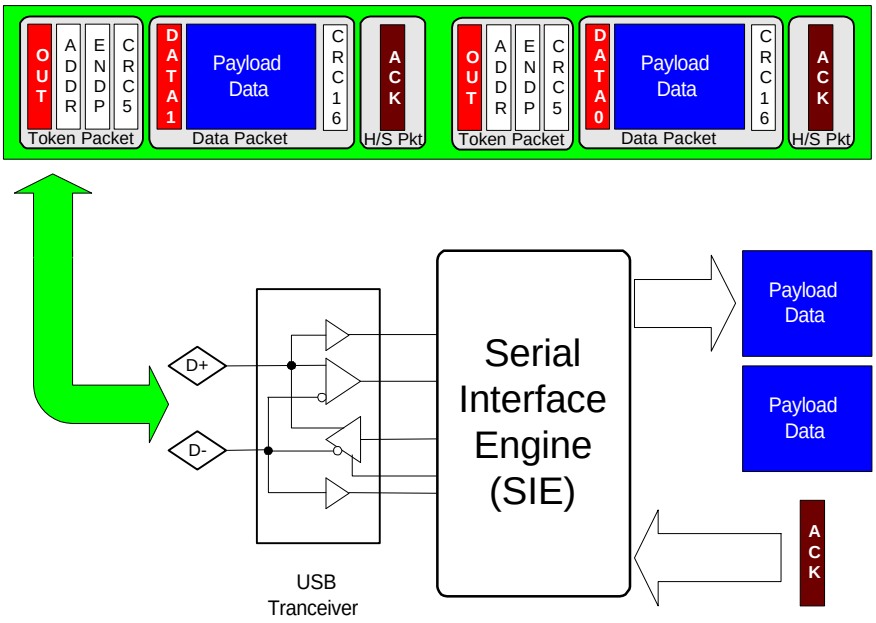


Figure 1-8. What the SIE does

Every USB device has a Serial Interface Engine, or 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 transparently handled by the SIE.

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

The EZ-USB 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 does all of the work shown in Figure 1-8, plus more. It contains additional logic to perform a full enumeration, using an internal table of descriptors. It also responds to a vendor specific "Anchor download" device request to load its internal RAM. An added bonus is that the added SIE functionality is also 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 "Anchor Core".

1.11 EZ-USB Microprocessor

The EZ-USB microprocessor is an enhanced 8051 core. Use of an 8051 compatible processor makes extensive software support tools immediately available to the EZ-USB designer. This enhanced 8051 core, described in Chapter 2 and Appendices A-C, has the following features:

- 4-clock cycle, as compared to the 12-clock cycle of a standard 8051, giving a 3X speed improvement.
- Dual data pointers for faster memory-to-memory transfers.
- Two UARTS.
- Three counter-timers.
- An expanded interrupt system.
- 24 MHz clock.
- 256 bytes of internal register RAM.
- Standard 8051 instruction set—if you know the 8051, you know EZ-USB.

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

The 8051 communicates with the SIE using a set of registers, which occupy 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 12, "EZ-USB Registers".

The EZ-USB 8051 has two duties. First, it participates in the protocol defined in Chapter 9 of the USB Specification, "USB Device Framework". Thanks to EZ-USB 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 IO ports, UARTS, and an I^2C bus master controller. These resources are described in Chapter 4, "EZ-USB Input/Output".

1.12 ReNumeration™

Because it is "soft", the EZ-USB chip 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 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, "Enumeration/ReNumeration" describes this feature in detail, along with other EZ-USB boot (startup) modes.

1.13 EZ-USB 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 1 direction bit. Therefore USB can uniquely address 32 endpoints, IN0 through IN15 and OUT0 through OUT15.

From the EZ-USB point of view, an endpoint is a buffer full of bytes received or to be 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.

Four USB endpoint types are defined, Bulk, Control, Interrupt and Isochronous.

1.13.1 EZ-USB 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 (and therefore buffer sizes) of 8, 16, 32 or 64 bytes. EZ-USB 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 form, or as FIFO data using a special EZ-USB *Autopointer* (Chapter 6, "Bulk Transfers").

1.13.2 EZ-USB 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, so if you have an endpoint 0 IN CONTROL endpoint you automatically have an endpoint 0 OUT endpoint. Control endpoints alone 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 Chapter Nine of the USB specification. A USB device must respond properly to the requests described in this chapter to pass USB compliance testing (usually referred to as the USB "Chapter Nine Test").

Endpoint zero is the only CONTROL endpoint in the EZ-USB chip. The 8051 responds to device requests issued by the host over endpoint zero. The EZ-USB core is significantly enhanced to simplify the 8051 code required to service these requests. Chapter 7, "Endpoint Zero" provides a detailed roadmap for writing USB Chapter 9 compliant 8051 code.

1.13.3 EZ-USB Interrupt Endpoints

Interrupt endpoints are almost identical to bulk endpoints. Fourteen EZ-USB endpoints (EP1-EP7, IN and OUT) may be used as interrupt endpoints. Interrupt endpoints have maximum packet sizes up to 64, and 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, "Bulk Transfers".

1.13.4 EZ-USB 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 cameras and scanners. 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 to be able to send or receive its data every frame.

EZ-USB contains sixteen isochronous endpoints, numbered 8-15 (8IN-15IN, and 8OUT-15OUT). 1024 bytes of FIFO memory are available to the sixteen endpoints, and may be divided among them. The EZ-USB chip actually contains 2048 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 8.

1.14 Fast Transfer Modes

The following versions of the EZ-USB have a fast transfer mode: AN2125SC, AN2126SC, AN2135SC, AN2136SC, AN2131QC, and AN2141QC. The fast transfer mode minimizes the transfer time from EZ-USB FIFOS and external memory. In this mode the EZ-USB core monitors transfers between the 8051 accumulator and internal FIFOS, and substitutes data from the external data bus for the transfer. The EZ-USB core also supplies external FIFO read and write strobes to synchronize the transfers.

Using the fast transfer mode, the 8051 transfers a byte of data between an internal FIFO and the external bus using a single 8051 "MOVX" instruction, which takes 2 cycles or 333 nanoseconds. Both Isochronous and Bulk endpoints can use this fast transfer mode.

1.15 Interrupts

The EZ-USB enhanced 8051 adds seven interrupt sources to the standard 8051 interrupt system. Three of the added interrupts are used internally, and the others are available on device pins. INT2 is used for all USB interrupts. INT3 is used by the I^2C interface. A third interrupt is used for remote wakeup indication.

The EZ-USB core automatically supplies jump vectors ("Autovectors") for its USB interrupts to save the 8051 from having to test bits to determine the source of the interrupt. Each BULK/CONTROL/INTERRUPT endpoint has its own vector, so when an endpoint requires service the proper interrupt service routine is automatically invoked. The 8051 services all isochronous endpoints in response to an SOF (Start Of Frame) interrupt request. Chapter 9, "EZ-USB Interrupts" describes the EZ-USB interrupt system.

1.16 Reset and Power Management

The EZ-USB chip contains four resets:

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

The functions of the various EZ-USB resets are described in Chapter 10, “EZ-USB 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 microamps of current from the Vcc wire while in suspend. The EZ-USB chip contains logic to turn off its internal oscillator and enter a “sleep” state. A special interrupt, triggered by a wakeup pin or wake-up signaling on the USB bus, starts the oscillator and interrupts the 8051 to resume operation.

Low power operation is described in Chapter 11, “EZ-USB Power Management”.

1.17 EZ-USB Product Family

The EZ-USB family is available in various pinouts to serve different system requirements and costs.

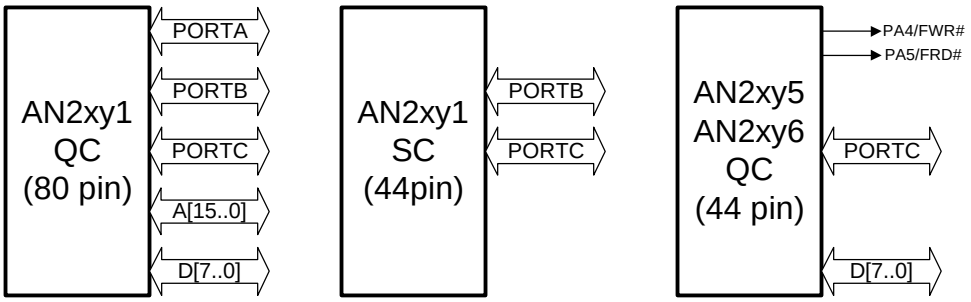


Figure 1-9. EZ-USB Family Members

Figure 1-9 illustrates the basic EZ-USB family options. Referring to the part numbers, x=1 for RAM-based versions, and x=3 for ROM based versions. The “y” option indicates the amount of internal memory, equal to 2^y kilobytes. Part numbers ending in 1 or 5 have isochronous endpoints; part numbers ending in 6 do not.

1.18 Pin Description

The tables on the following pages describe the EZ-USB pins, both for the 44 pin packages and the 80 pin package for the entire EZ-USB family.

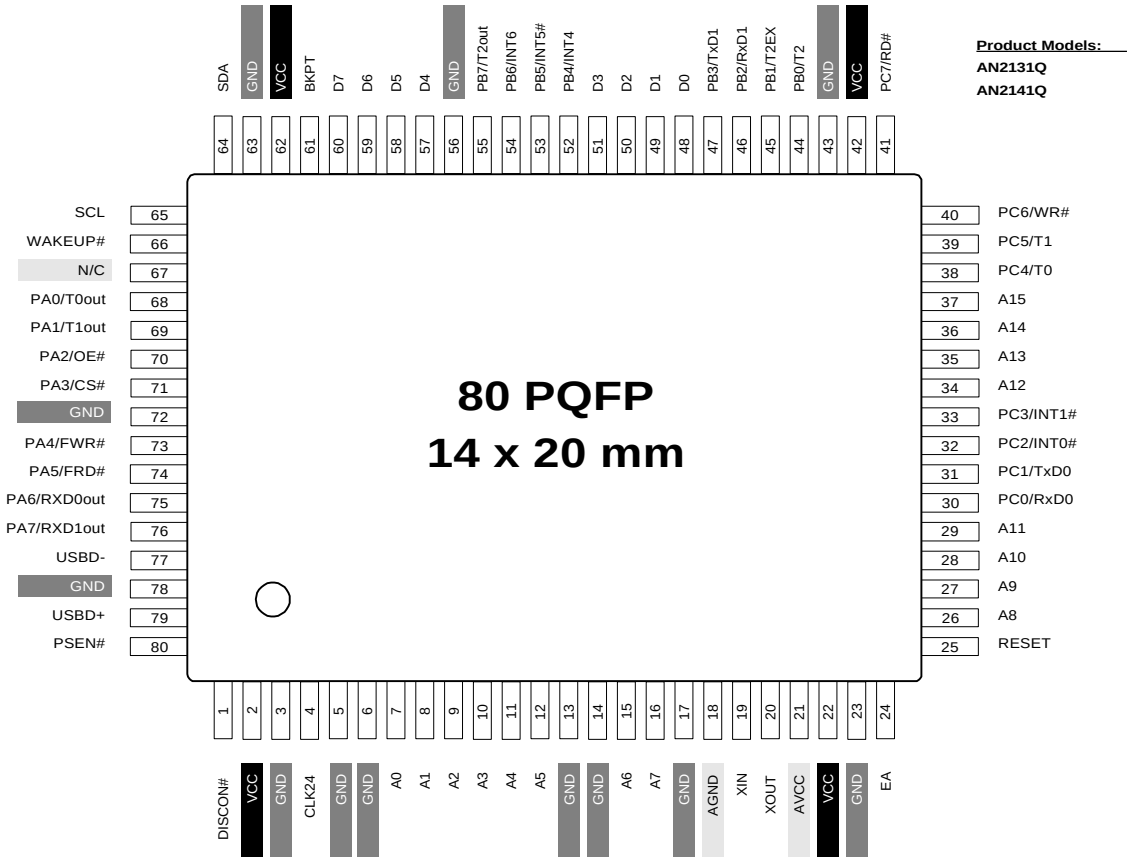


Figure 1-10. 80-pin PQFP Package (AN2131Q and AN2141Q)

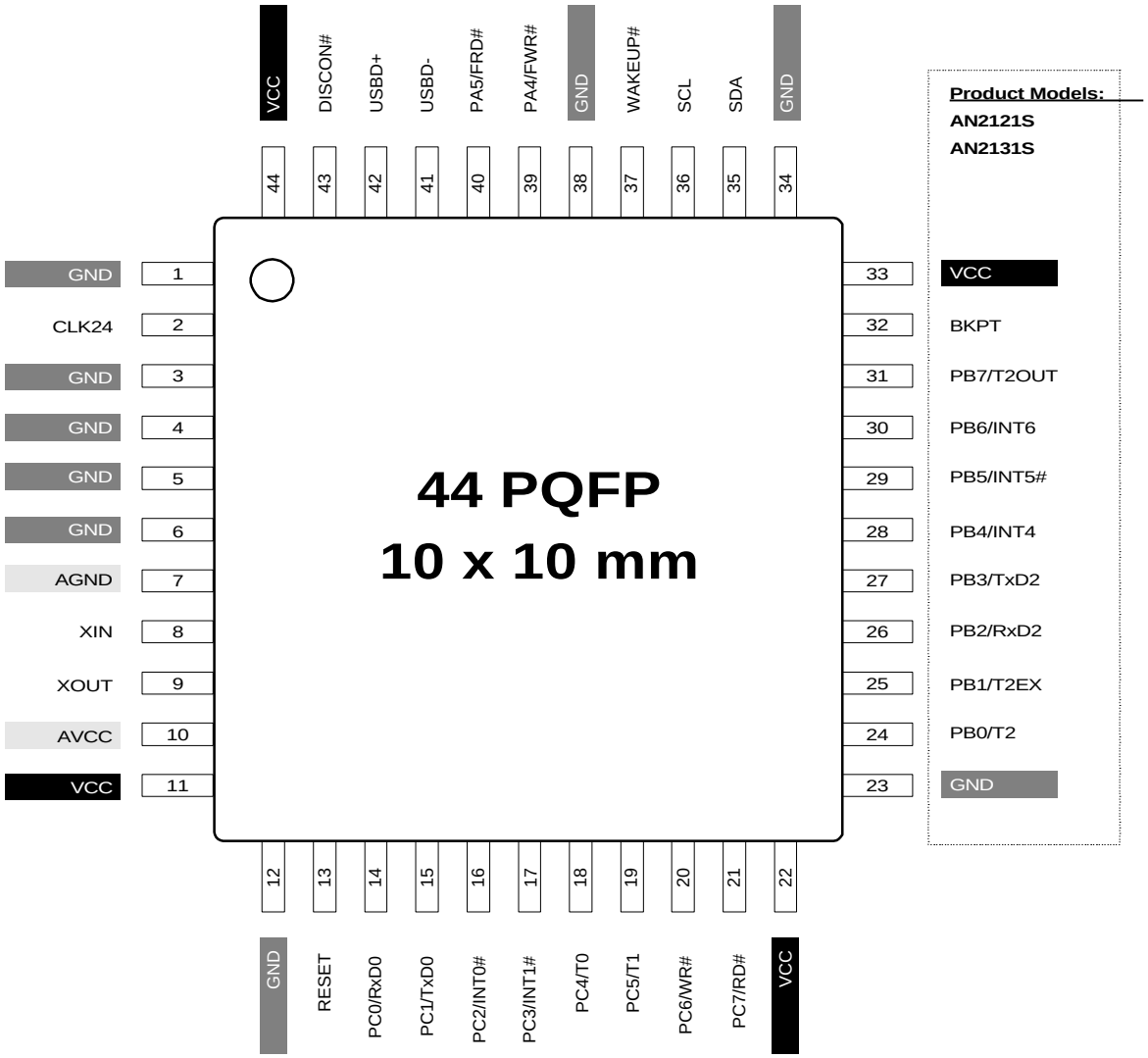


Figure 1-11. 44-pin PQFP Package (AN2121S and AN2131S)

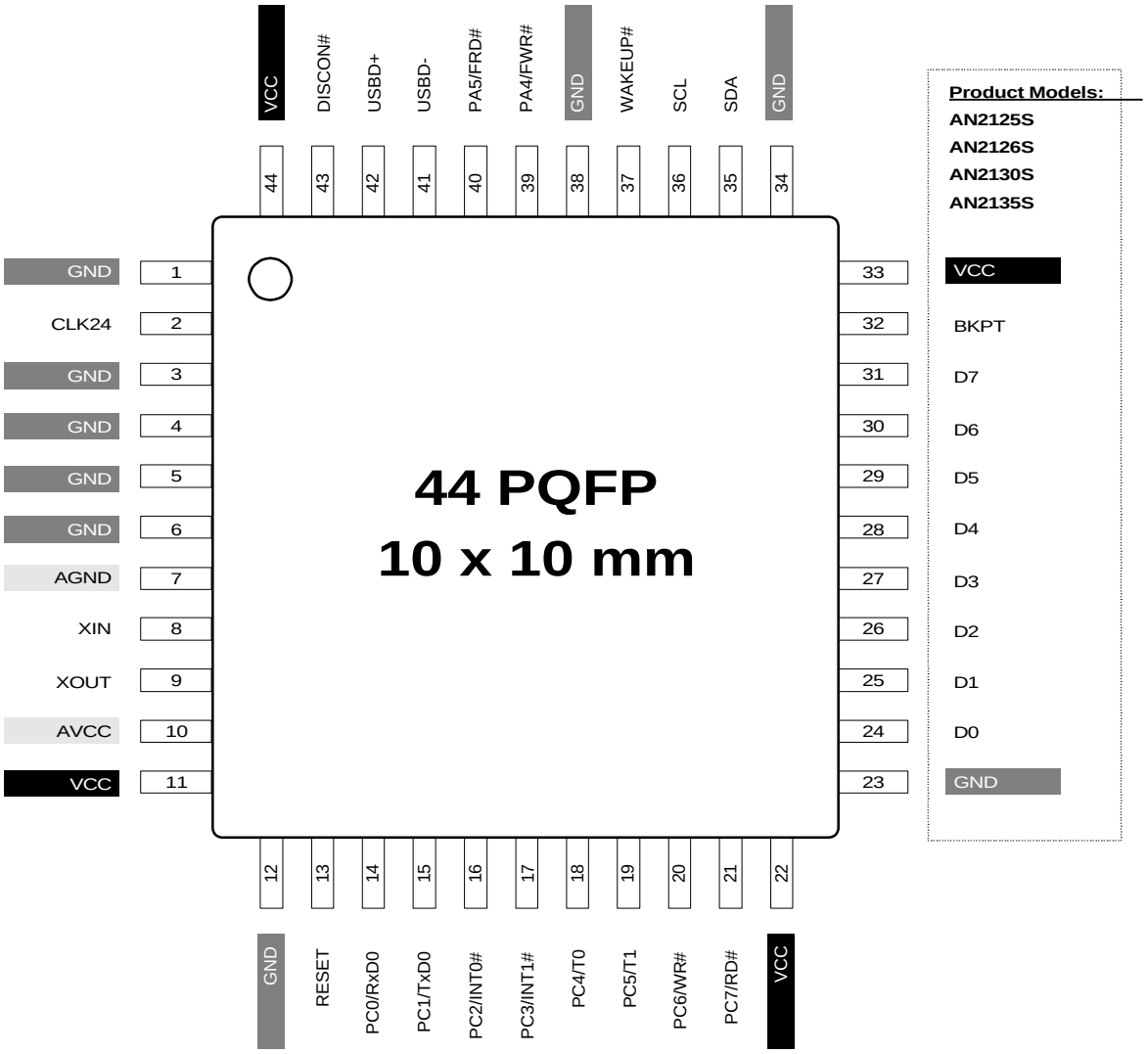


Figure 1-12. 44-pin PQFP Package (AN2125S, AN2126S, AN2130S, AN2135S)

2125SC 2126SC 2135SC 2136SC	2121SC 2131SC	2131QC 2141QC	Name	Type	Default	Description
10	10	21	AVCC	Power	N/A	**Analog VCC**. This signal provides power to the analog section of the chip.
7	7	18	AGND	Power	N/A	**Analog Ground**. Connect to ground with as short a path as possible.
43	43	1	DISCON#	Output	HI	**Disconnect**. This signal drives low when the 8051 sets the DISCON bit HI (USBCS.3). When the DISCON bit is LO, the DISCON# pin either drives high or floats, depending on the state of the DISCOE bit (USBCS.2).
41	41	77	USBD-	I/O/Z	Z	**USB D- signal**. Connect to the USB D- signal through a 24 ohm resistor.
42	42	79	USBD+	I/O/Z	Z	**USB D+ signal**. Connect to the USB D+ pin through a 24 ohm resistor.
N/A	N/A	7-12, 15, 16, 26-29, 34-37	A0-A5, A6, A7 A8-A11 A12-A15	Output	0x0000	**8051 Address bus**. This bus is driven at all times. When the 8051 is addressing internal RAM it reflects the internal address.
24-27 28-31	N/A	48-51, 57-60	D0-D3, D4-D7	I/O/Z	Z	**8051 Data bus**. This bi-directional bus is high-impedance when inactive, input for bus reads, and output for bus writes. The data bus is also used to transfer data directly to and from internal EZ-USB FIFOS under control of the FRD# and FWR# strobes. D0-D7 are active only for external bus accesses, and are driven low in suspend.
N/A	N/A	80	PSEN#	Output	H	**Program Store Enable**. This active-low signal indicates a code fetch from external memory. It is active for program memory fetches above 0x1B40 when the EA pin is LO, or above 0x0000 when the EA pin is HI.
32	32	61	BKPT	Output	0	**Breakpoint**. This pin goes active (high) when the 8051 address bus matches the BPADDRH/L registers and breakpoints are enabled in the USBBAV register (BPEN=1). If the BPPULSE bit in the USBBAV register is HI, this signal pulses high for eight 24 Mhz clocks. If the BPPULSE bit is LO, the signal remains high until the 8051 clears the BREAK bit (by writing 1 to it) in the USBBAV register.
13	13	25	RESET	Input	N/A	**Active high Reset**. Resets the 8051 and the USB SIE. This pin is normally tied to ground through a 10K resistor, and to VCC through a 1 uF capacitor.
N/A	N/A	24	EA	Input	N/A	**External Access**. If this signal is active (high), the 8051 fetches code from external memory instead of the internal program RAM. If EA=0, the 8051 fetches code from 0x1B40 and up (AN2131).
8	8	19	XIN	Input	N/A	**Crystal input**. Connect this signal to a 12Mhz series resonant, fundamental mode crystal and 22-33 pF capacitor to GND.

2125SC 2126SC 2135SC 2136SC	2121SC 2131SC	2131QC 2141QC	Name	Type	Default	Description
9	9	20	XOUT	Output	N/A	**Crystal output**. Connect this signal to a 12Mhz series resonant, fundamental mode crystal and 22-33 pF capacitor to GND.
N/A	N/A	68	PA0 or T0OUT	I/O	I (PA0)	Multiplexed pin whose function is selected by the "T0OUT" bit of the PORTACFG register. If T0OUT=0, the pin is the bi-directional I/O port bit PA0. If T0OUT=1 the pin is the active-high T0OUT signal from 8051 Timer-counter0. T0OUT outputs a high level for one CLK24 clock cycle when Timer 0 overflows. If Timer 0 is operated in mode 3 (two separate timer/counters), T0OUT is active when the low byte timer/counter overflows.
N/A	N/A	69	PA1 or T1OUT	I/O	I (PA1)	Multiplexed pin whose function is selected by the "T1OUT" bit of the PORTACFG register. If T1OUT=0, the pin is the bi-directional I/O port bit PA1. If T1OUT=1 the pin is the active-high T1OUT signal from 8051 Timer-counter1. T1OUT outputs a high level for one CLK24 clock cycle when Timer1 overflows. If Timer 1 is operated in mode 3 (two separate timer/counters), T1OUT is active when the low byte timer/counter overflows.
N/A	N/A	70	PA2 or OE#	I/O	I (PA2)	Multiplexed pin whose function is selected by the "OE" bit of the PORTACFG register. If OE=0, the pin is the bi-directional I/O port pin PA2. If OE=1 the pin is an active-low output enable for external memory. If the OE# pin is used, it should be externally pulled up to VCC to insure that the write strobe is inactive (high) at power-on.
N/A	N/A	71	PA3 or CS#	I/O	I (PA3)	Multiplexed pin whose function is selected by the "CS" bit of the PORTACFG register. If CS=0, the pin is the bi-directional I/O port pin PA3. If CS=1 the pin is an active-low chip select for external memory. If the CS# pin is used, it should be externally pulled up to VCC to insure that the write strobe is inactive (high) at power-on.
39	39	73	PA4 or FWR#	I/O	I (PA4)	Multiplexed pin whose function is selected by the "FWR" (Fast Write) bit of the PORTACFG register. If FWR=0, the pin is the bi-directional I/O port pin PA4. If FWR=1 the pin is the write strobe for an external FIFO. If the FWR# pin is used, it should be externally pulled up to VCC to insure that the write strobe is inactive at power-on.

2125SC 2126SC 2135SC 2136SC	2121SC 2131SC	2131QC 2141QC	Name	Type	Default	Description
40	40	74	PA5 or FRD#	I/O	I (PA5)	Multiplexed pin whose function is selected by the "FRD" (Fast Read) bit of the PORTACFG register. If FRD=0, the pin is the bi-directional I/O port pin PA5. If FRD=1 the pin is the read strobe for an external FIFO. If the FRD# pin is used, it should be externally pulled up to VCC to insure that the write strobe is inactive (high) at power-on.
N/A	N/A	75	PA6 or RXD0OUT	I/O	I (PA6)	Multiplexed pin whose function is selected by the "RXD0OUT" bit of the PORTACFG register. If RXD0OUT=0 (default), the pin is the bi-directional I/O port bit PA6. If RXD0OUT=1 the pin is the active-high RXD0OUT signal from 8051 UART0. If RXD0OUT is selected and UART0 is in mode 0, this pin provides the output data for UART0 only when it is in sync mode. Otherwise it is a 1.
N/A	N/A	76	PA7 or RXD1OUT	I/O	I (PA7)	Multiplexed pin whose function is selected by the "RXD1OUT" bit of the PORTACFG register. If RXD1OUT=0 (default), the pin is the bi-directional I/O port bit PA7. If RXD1OUT =1 the pin is the active-high RXD1OUT signal from 8051 UART1. When RXD1OUT is selected and UART1 is in mode 0, this pin provides the output data for UART1 only when it is in sync mode. In modes 1, 2 and 3, this pin is a 1.
N/A	24	44	PB0 or T2	I/O	I (PB0)	Multiplexed pin whose function is selected by the "T2" bit of the PORTBCFG register. If T2=0, the pin is the bi-directional I/O port bit PB0. If T2=1 the pin is the active-high T2 signal from 8051 Timer2, which provides the input to Timer2 when C/T2=1. When C/T2=0, Timer2 does not use this pin.
N/A	25	45	PB1 or T2EX	I/O	I (PB1)	Multiplexed pin whose function is selected by the "T2EX" bit of the PORTBCFG register. If T2EX=0, the pin is the bi-directional I/O port bit PB1. If T2EX=1 the pin is the active-high T2EX signal from 8051 Timer2. T2EX – Input pin that reloads timer 2 on its falling edge. Only active if the EXEN2 bit is set in T2CON.
N/A	26	46	PB2 or RXD1	I/O	I (PB2)	Multiplexed pin whose function is selected by the "RXD1" bit of the PORTBCFG register. If RXD1=0, the pin is the bi-directional I/O port bit PB2. If RXD1=1 the pin is the active-high RXD1 input signal for 8051 UART1, which provides data to the UART in all modes.

2125SC 2126SC 2135SC 2136SC	2121SC 2131SC	2131QC 2141QC	Name	Type	Default	Description
N/A	27	47	PB3 or TXD1	I/O	I (PB3)	Multiplexed pin whose function is selected by the "TXD1" bit of the PORTBCFG register. If TXD1=0, the pin is the bi-directional I/O port bit PB3. If TXD1=1 the pin is the active-high TXD1 output pin for 8051 UART1, which provides the output clock in sync mode, and the output data in async mode.
N/A	28	52	PB4 or INT4	I/O	I (PB4)	Multiplexed pin whose function is selected by the "INT4" bit of the PORTBCFG register. If INT4=0, the pin is the bi-directional I/O port bit PB4. If INT4=1 the pin is the 8051 INT4 interrupt request signal. The INT4 pin is edge-sensitive, active high.
N/A	29	53	PB5 or INT5#	I/O	I (PB5)	Multiplexed pin whose function is selected by the "INT5" bit of the PORTBCFG register. If INT5=0, the pin is the bi-directional I/O port bit PB5. If INT5=1 the pin is the INT5# interrupt request signal. The INT5# pin is edge-sensitive, active low.
N/A	30	54	PB6 or INT6	I/O	I (PB6)	Multiplexed pin whose function is selected by the "INT6" bit of the PORTBCFG register. If INT6=0, the pin is the bi-directional I/O port bit PB6. If INT6=1 the pin is the INT6 interrupt request signal. The INT6 pin is edge-sensitive, active high.
N/A	31	55	PB7 or T2OUT	I/O	I (PB7)	Multiplexed pin whose function is selected by the "T2OUT" bit of the PORTBCFG register. If T2OUT=0, the pin is the bi-directional I/O port bit PB7. If T2OUT=1 the pin is the active-high T2OUT signal from 8051 Timer2. T2OUT is active (high) for one clock cycle when Timer/Counter 2 overflows.
14	14	30	PC0 or RXD0	I/O	I (PC0)	Multiplexed pin whose function is selected by the "RXD0" bit of the PORTCCFG register. If RXD0=0, the pin is the bi-directional I/O port bit PC0. If RXD0=1 the pin is the active-high RXD0 from 8051 UART0, which provides data to the UART in all modes.
15	15	31	PC1 or TXD0	I/O	I (PC1)	Multiplexed pin whose function is selected by the "TXD0" bit of the PORTCCFG register. If TXD0=0, the pin is the bi-directional I/O port bit PC1. If TXD0=1 the pin is the active-high TXD0 signal for 8051 UART0, which provides the output clock in sync mode, and the output data in async mode.

2125SC 2126SC 2135SC 2136SC	2121SC 2131SC	2131QC 2141QC	Name	Type	Default	Description
16	16	32	PC2 or INT0#	I/O	I (PC2)	Multiplexed pin whose function is selected by the "INT0" bit of the PORTCCFG register. If INT0=0, the pin is the bi-directional I/O port bit PC2. If INT0=1 the pin is the active-low 8051 INT0 interrupt input signal, which is either edge triggered (IT0 = 1) or level triggered (IT0 = 0).
17	17	33	PC3 or INT1#	I/O	I (PC3)	Multiplexed pin whose function is selected by the "INT1" bit of the PORTCCFG register. If INT1=0, the pin is the bi-directional I/O port bit PC3. If INT1=1 the pin is the active-low 8051 INT1 interrupt input signal, which is either edge triggered (IT1 = 1) or level triggered (IT1 = 0).
18	18	38	PC4 or T0	I/O	I (PC4)	Multiplexed pin whose function is selected by the "T0" bit of the PORTCCFG register. If T0=0, the pin is the bi-directional I/O port bit PC4. If T0=1 the pin is the active-high T0 signal for 8051 Timer0, which provides the input to Timer0 when C/T0 is 1. When C/T0 is 0, Timer0 does not use this bit.
19	19	39	PC5 or T1	I/O	I (PC5)	Multiplexed pin whose function is selected by the "T1" bit of the PORTCCFG register. If T1=0, the pin is the bi-directional I/O port bit PC5. If T1=1 the pin is the active-high T1 signal from 8051 Timer1, which provides the input to Timer1 when C/T1 is 1. When C/T0 is 0, Timer1 does not use this bit.
20	20	40	PC6 or WR#	I/O	I (PC6)	Multiplexed pin whose function is selected by the "WR" bit of the PORTCCFG register. If WR=0, the pin is the bi-directional I/O port bit PC6. If WR=1 the pin is the active-low write signal for external memory. If the WR# signal is used, it should be externally pulled up to VCC to insure that the write strobe is inactive at power-on.
21	21	41	PC7 or RD#	I/O	I (PC7)	Multiplexed pin whose function is selected by the "RD" bit of the PORTCCFG register. If RD=0, the pin is the bi-directional I/O port bit PC7. If RD=1 the pin is the active-low read signal for external memory. If the RD# signal is used, it should be externally pulled up to VCC to insure that the read strobe is inactive at power-on.
2	2	4	CLK24	Output		**24Mhz clock**, phase locked to the 12Mhz input clock. Output is disabled by setting the OUTCLKEN bit = 0 in the CPUCS register.
37	37	66	WAKEUP#	Input	N/A	**USB Wakeup**. If the 8051 is in suspend, a high to low edge on this pin starts up the oscillator and interrupts the 8051 to allow it to exit the suspend mode. Holding WAKEUP# LOW inhibits the EZ-USB chip from suspending.
36	36	65	SCL	OD	Z	**I^2C Clock**. Connect to VCC with a 2.2K resistor.
35	35	64	SDA	OD	Z	**I^2C Data**. Connect to VCC with a 2.2K resistor.

2125SC 2126SC 2135SC 2136SC	2121SC 2131SC	2131QC 2141QC	Name	Type	Default	Description
11, 22, 33, 44	11, 22, 33, 44	2, 22, 42, 62	VCC		N/A	**VCC**. 3.3 V power source.
1, 3, 4, 5, 6, 12, 23, 34, 38	1, 3, 4, 5, 6, 12, 23, 34, 38	3, 5, 6, 13, 14, 17, 23, 43, 56, 63, 72, 78	GND		N/A	**Ground**. Note: On the 80-pin package, pins 5,6,13,14 and 72 are test pins that must be grounded for normal operation. Driving pin 72 high floats all functional pins for automated board test. The corresponding pins on the 44-pin package are pins 3,4,5, 6 and 38. Driving pin 38 high floats all functional pins for automated board test.
N/A	N/A	67	NC		N/A	This pin must be left unconnected.

2 EZ-USB CPU

2.1 Introduction

The EZ-USB built-in microprocessor, an enhanced 8051 core, is fully described in Appendices A-C. This chapter introduces the processor, its interface to the EZ-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 4 clocks, as compared to 12 clocks with the standard 8051.
- The 8051 runs at 24 MHz.

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

1. A second data pointer.
2. A second UART.
3. A third, 16-bit timer (TIMER2).
4. A high speed memory interface with a non-multiplexed 16-bit address bus.
5. Eight additional interrupts (INT2-INT6, PFI, T2, UART1).
6. Variable length MOVX timing to accommodate fast/slow RAM peripherals.
7. 3.3 Volt operation.

2.3 EZ-USB Enhancements

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

- Fast “DMA-like” external transfers (Autopointer, Fast Transfer Mode)
- Vectored USB interrupts (Autovector)
- Separate buffers for SETUP and DATA portions of a CONTROL transfer.
- Breakpoint Facility

2.4 EZ-USB Register Interface

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

- Endpoint buffers and FIFOS
- 8051 control
- IO Ports
- Fast Transfer
- I^2C Controller
- Interrupts
- USB functions

These registers and their functions are described throughout this manual. A full description of every register and bit appears in Chapter 12, "EZ-USB Registers".

2.5 EZ-USB Internal RAM

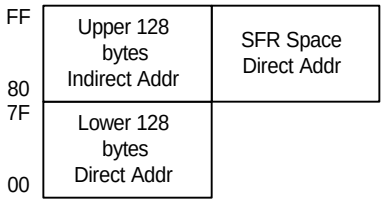


Figure 2-1. 8051 Registers

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

All internal EZ-USB RAM, which includes program/data memory, bulk endpoint buffer memory, and the EZ-USB 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 RAM and register set is actually inside the EZ-USB chip. External memory attached to the AN2131Q address and data busses is also accessed by the MOVX instruction. The EZ-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 IO Ports

A standard 8051 communicates with its IO ports 0-3 through four Special Function Registers (SFR). Standard 8051 IO pins are "quasi-bidirectional", with weak pullups that briefly drive high only when the pin makes a zero to one transition.

The EZ-USB core implements IO ports differently than a standard 8051, as described in Chapter 4, "EZ-USB Input/Output". Instead of using the 8051 IO ports and SFR's, the EZ-USB core implements a flexible IO system that is controlled *via* the EZ-USB register set. Each EZ-USB IO pin functions identically, having the following resources:

- An output latch. Used when the pin is an output port.
- A bit that indicates the state of the IO pin, regardless of its configuration (input or output).
- An output enable bit that causes the IO pin to be driven from the output latch.
- An alternate function bit that determines whether the pin is general IO or a special 8051 or EZ-USB function.

The SFRs associated with 8051 ports 0-3 are not implemented in EZ-USB. These SFR addresses include P0 (0x80), P1 (0x90), P2 (0xA0) and P3 (0xB0). **NOTE:** Because the address bus is non-multiplexed, the "MOVX @R0/R1" instructions are not supported.

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 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^2C Controller
	INT4	Device Pin, PB4/INT4
	INT5	Device Pin, PB5/INT5#
	INT6	Device Pin, PB6/INT6
	PFI	Device Pin, USB WAKEUP#
	Tx1 & Rx1	Internal, UART1
	Timer 2	Internal, Timer 2

The EZ-USB chip uses 8051 INT2 for 21 different USB interrupts: sixteen bulk endpoints plus SOF, Suspend, SETUP Data, SETUP Token and USB Bus Reset. To help the 8051 determine which interrupt is active, the EZ-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 interrupt system is the subject of Chapter 9, "EZ-USB Interrupts".

2.8 Power Control

The EZ-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 microamps when suspended. Power control is accomplished using a combination of 8051 and EZ-USB core resources. The mechanism by which EZ-USB powers down for suspend, and then re-powers to resume operation, is described in detail in Chapter 11, "EZ-USB Power Management".

A suspend operation uses three 8051 resources, the "idle" mode and two interrupts. Many enhanced 8051 architectures provide power control similar (or identical) to the EZ-USB enhanced 8051 core.

A USB suspend operation is indicated by a lack of bus activity for 3 milliseconds. The EZ-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 EZ-USB core causes the core to shut down the 12 MHz oscillator and internal PLL. This stops all internal clocks to allow the EZ-USB core and 8051 to enter a very low power mode.

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

1. The EZ-USB core re-starts the 12 MHz oscillator and PLL, and waits for the clocks to stabilize.
2. The EZ-USB core asserts a special, high-priority 8051 interrupt to signal a 'resume' interrupt.
3. 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 SFR's

The EZ-USB 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 SFR Registers and Bits

8051 Enhancement	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)
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
Idle Mode	PCON.0	0x87	EZ-USB Power Down (Suspend)

2.10 Internal bus

Members of the EZ-USB family that provide 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: IO 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 chip, and no ALE signal is needed. In addition to eliminating the customary external latch, the non-multiplexed bus saves one cycle per memory fetch cycle, further improving 8051 performance.

A standard 8051 user must choose between using Port 0 as a memory expansion port or an IO port. The AN2131Q provides a separate IO system with its own control registers (in external memory space), and provides the IO port signals on dedicated (not shared) pins. This allows the external data bus to be used to expand memory without sacrificing IO 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.

A special “fast transfer” mode gives the EZ-USB family the capability to transfer data to and from external memory over the expansion bus using a single ‘movx’ instruction, which takes only two cycles. (8 clocks).

2.11 RESET

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

The EZ-USB 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 resets and their effects are described in Chapter 10, “EZ-USB Resets”.

3 EZ-USB Memory

3.1 Introduction

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

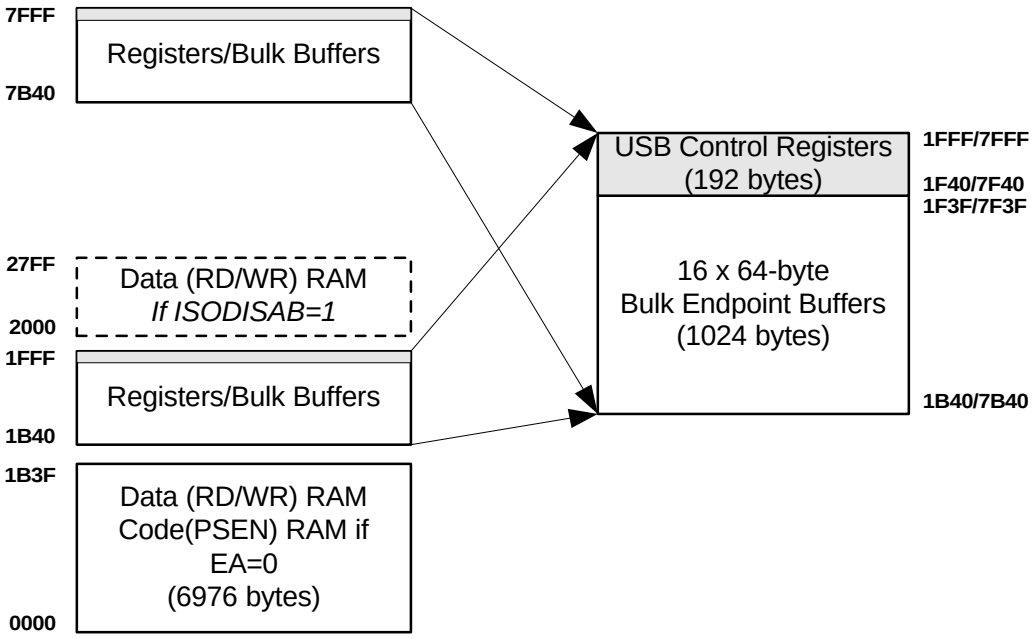


Figure 3-1. EZ-USB 8K Memory Map. Addresses are in Hexadecimal

3.2 8051 Memory

Figure 3-1 illustrates the two internal EZ-USB RAM regions. 6976 bytes of general-purpose RAM occupy addresses 0x0000-0x1B3F. This RAM is loadable by the EZ-USB core or I^2C bus EEPROM, and contains 8051 code and data.

The EZ-USB EA (External Access) pin controls where the bottom segment of code (PSEN) memory is located—inside (EA=0) or outside (EA=1) the EZ-USB chip. If the EA (External Access) pin is tied low, the EZ-USB core internally OR's 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.

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.

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.

1024 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 (move external) instruction. In the 8K RAM EZ-USB version, the 1024 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 in Figure 3-2. The memory space at 0x1F40-0x1FFF should not be used.

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

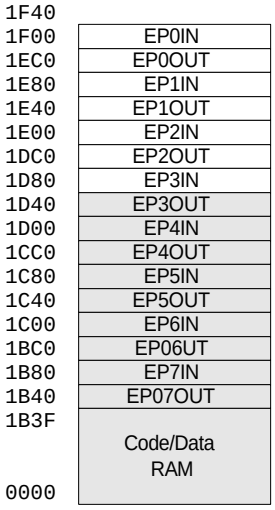


Figure 3-2. Unused bulk endpoint buffers (shaded) used as data memory

In the example shown in Figure 3-2, only endpoints 0-IN through 3-IN are used for the USB function, so the data RAM (shaded) can be extended to 0x1D7F.

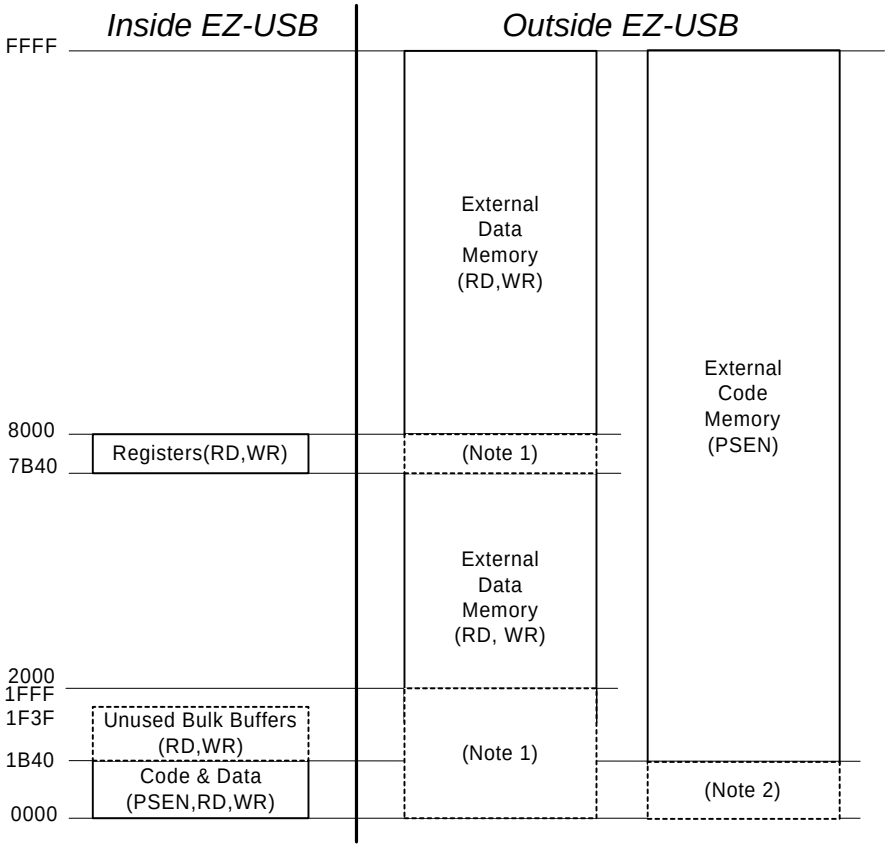
If an application uses *none* of the sixteen EZ-USB isochronous endpoints, the 8051 can set the ISODISAB bit in the ISOCTL register to disable all sixteen isochronous endpoints, and make the 2 Kilobytes of isochronous FIFO RAM available as 8051 data RAM at 0x2000-0x27FF.

Setting ISODISAB=1 is an “all or nothing” choice, as all sixteen 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 Memory

The 80 pin EZ-USB package provides a 16-bit address bus, an 8-bit data bus, and memory control signals PSEN#, RD# and WR#. These signals are used to expand EZ-USB memory.



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 3-3. EZ-USB Memory Map with EA=0

Figure 3-3 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 EZ-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 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 are

ignored; no external strobes or select signals are generated when the 8051 executes a MOVX instruction that addresses these regions.

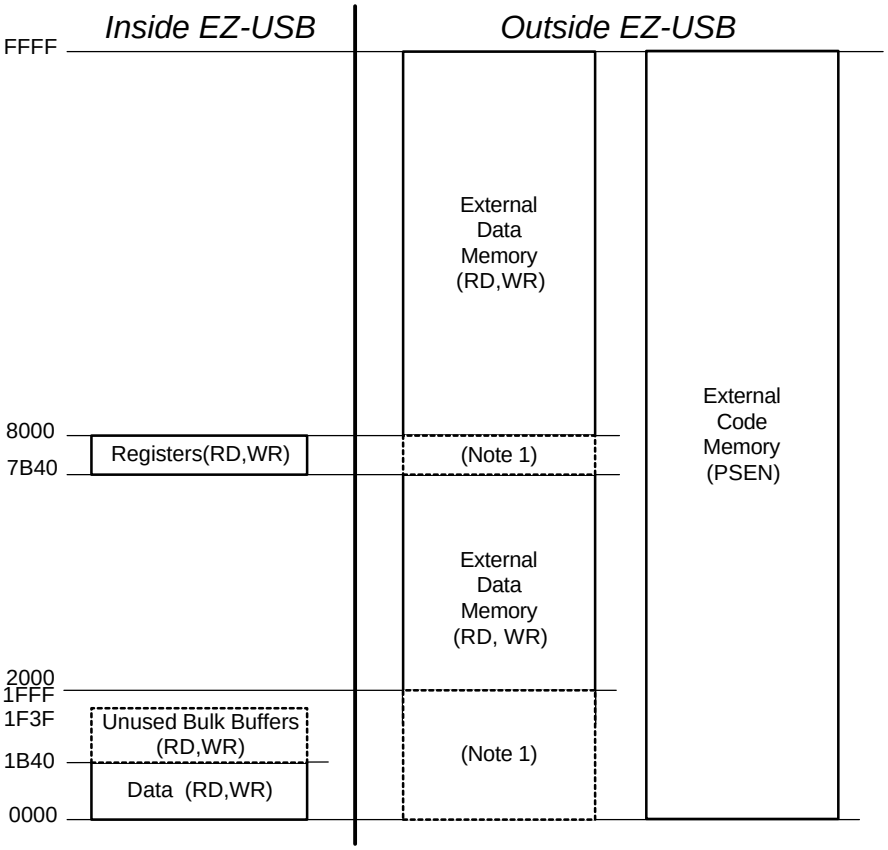
3.4 CS# and OE# Signals

The EZ-USB core automatically gates the standard 8051 RD# and WR# signals to exclude selection of external memory that exists internal to the EZ-USB 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 EZ-USB core provides two additional control signals, CS# and OE#. The equations for these signals are as follows:

- CS# = RD# or WR# or PSEN#
- OE# = RD# or PSEN#

Since 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.



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

Figure 3-4. EZ-USB Memory Map with EA=1

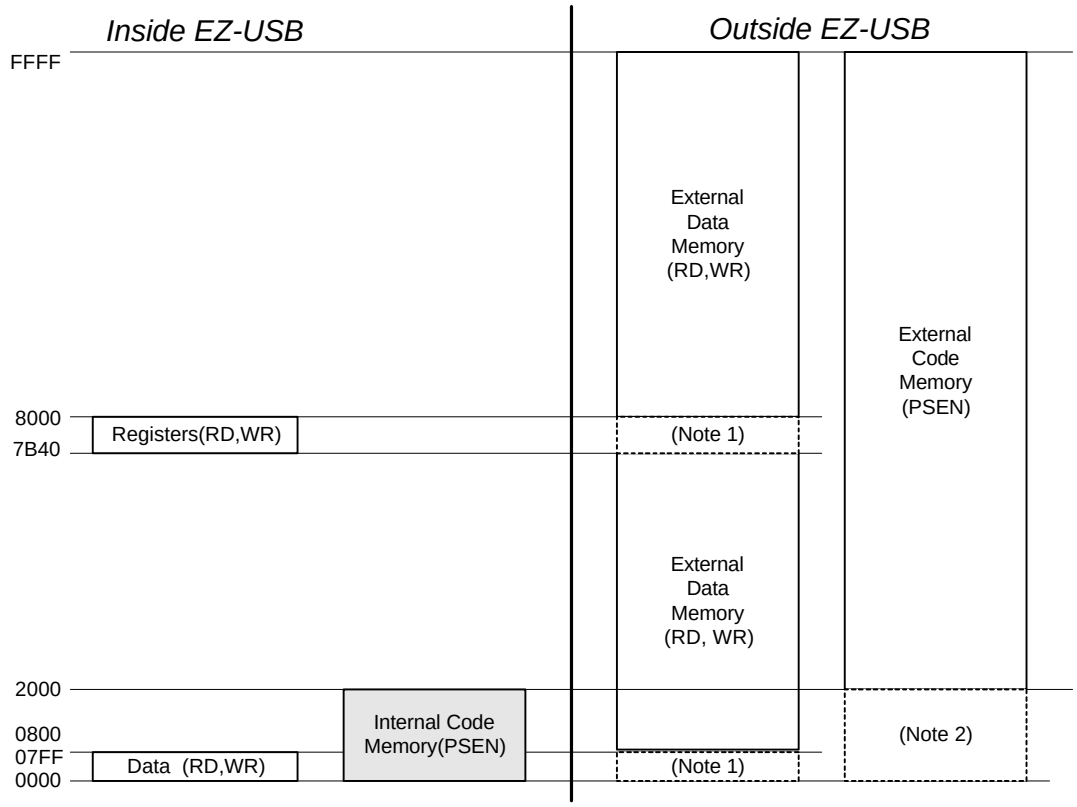
When EA=1 (Figure 3-4), all code (PSEN) memory is external. All internal EZ-USB RAM is data memory. This gives the user over 6 Kilobytes of general-purpose RAM, accessible by the 'movx' instruction.

NOTE:

Figure 3-3 and Figure 3-4 assume that the EZ-USB 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.

3.5 EZ-USB ROM Versions

The EZ-USB 8K Masked ROM and 32K Masked ROM memory maps are shown in Figure 3-5 and Figure 3-6.



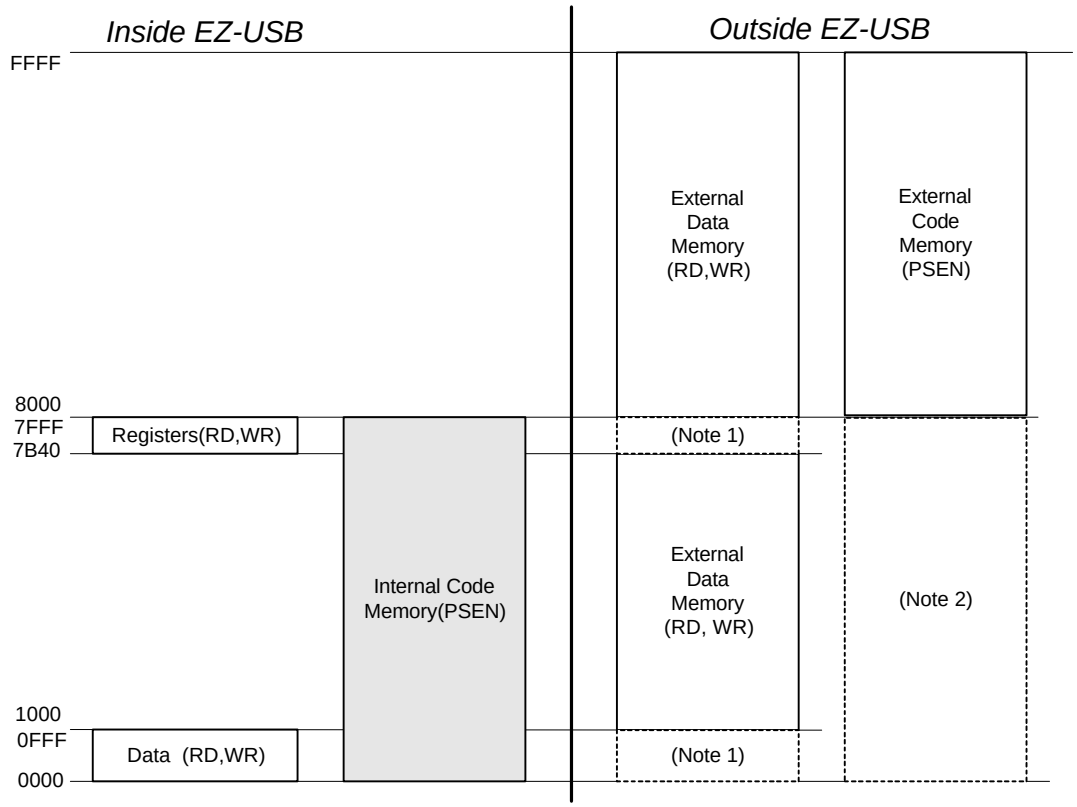
Note 1: OK to populate data memory here, but no RD# or WR# strobes are generated.

Note 2: OK to populate code memory here, but no PSEN# strobe is generated.

Figure 3-5. 8K ROM, 2K RAM version

EZ-USB ROM versions contain program memory starting at 0x0000. In these versions, the internal RAM is implemented as data-only memory.

Code for this ROM version can be developed and tested using the AN2131Q with an external code memory (EA=1, Figure 3-4). As long as the 8051 limits internal RAM access to 0x0000-0x07FF and accesses the EZ-USB registers and bulk data at 0x7B40-0x7FFF, the code in the external memory will be the identical image of the code that will ultimately be internal at 0x0000-0x1FFF in the ROM version.



Note 1: OK to populate data memory here, but no RD# or WR# strobes are generated.

Note 2: OK to populate code memory here, but no PSEN# strobe is generated.

Figure 3-6. 32K ROM, 4K RAM version

The EZ-USB 32K ROM version contains program memory from 0x0000 through 0x7FFF, and data memory from 0x0000 through 0x0FFF.

Code for this ROM version can be developed and tested using the AN2131Q with an external code memory (EA=1, Figure 3-4). As long as the 8051 limits internal RAM access to 0x0000-0x0FFF and accesses the EZ-USB registers and bulk data at 0x7B40-0x7FFF, the code in the external memory will be the identical image of the code that will ultimately be internal at 0x0000-0x7FFF in the ROM version.

4 EZ-USB Input/Output

4.1 Introduction

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

- A set of programmable IO pins
- A programmable I^2C Controller

This chapter begins with a description of the programmable IO pins, and shows how they are shared by a variety of 8051 and EZ-USB alternate functions such as UART, timer and interrupt signals.

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

- General purpose 8051 use
- Boot loading from an EEPROM

This chapter describes both the programming information for the 8051 I^2C interface, and the operating details of the I^2C boot loader. The role of the boot loader is described in Chapter 5, "Enumeration/ReNumeration™"

4.2 IO Ports

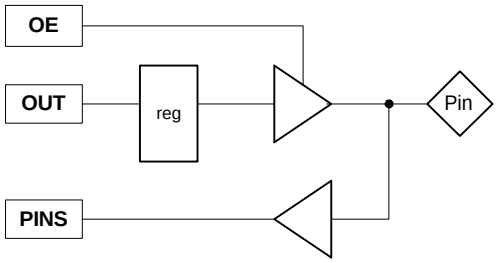


Figure 4-1. EZ-USB Input/Output Pin

The EZ-USB family implements its IO ports using memory-mapped registers. This is in contrast to a standard 8051, which uses SFR bits for input/output.

Figure 4-1 shows the basic structure of an EZ-USB IO pin. Twenty-four IO pins are grouped into three 8-bit ports named PORTA, PORTB and PORTC. The AN2131Q has all three ports, while the AN2131S has PORTB, PORTC, and two PORTA bits. The 8051 accesses IO 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, and the PINS bit indicates the state of the pin.

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 as shown in Figure 4-1, or connected to an alternate 8051 or EZ-USB function. Table 4-1 lists the alternate functions available on the IO pins. Figure 4-4 shows the registers and bits associated with the IO ports.

Table 4-1. IO pin functions for PORTxCFG=0 and PORTxCFG=1

PORTxCFG bit = 0	**PORTxCFG bit = 1**			
Signal	**Signal**	**Direc.**	**Description**	**Fig.**
PA0	T0out	OUT	Timer0 Overflow Pulse	4-2
PA1	T1out	OUT	Timer1 Overflow Pulse	4-2
PA2	OE#	OUT	EZ-USB Output Enable	4-2
PA3	CS#	OUT	EZ-USB Chip Select	4-2
PA4	FWR#	OUT	EZ-USB Fast Write Strobe	4-2
PA5	FRD#	OUT	EZ-USB Fast Read Strobe	4-2
PA6	RxD0out	OUT	UART0 mode 0 Data Out	4-2
PA7	RxD1out	OUT	UART1 mode 0 Data Out	4-2
PB0	T2	IN	Timer2 Clock Input	4-3
PB1	T2EX	IN	Timer2 Capture/Reload	4-3
PB2	RxD1	IN	UART1 Receive Data	4-3
PB3	TxD1	OUT	UART1 Transmit Data	4-2
PB4	INT4	IN	Interrupt 4	4-3
PB5	INT5#	IN	Interrupt 5	4-3
PB6	INT6	IN	Interrupt 6	4-3
PB7	T2OUT	OUT	Timer2 Overflow Pulse	4-2
PC0	RxD0	IN	UART0 Receive Data	4-3
PC1	TxD0	OUT	UART0 Transmit Data	4-2
PC2	INT0#	IN	Interrupt 0	4-3
PC3	INT1#	IN	Interrupt 1	4-3
PC4	T0	IN	Timer0 Clock Input	4-3
PC5	T1	IN	Timer1 Clock Input	4-3
PC6	WR#	OUT	Write Strobe	4-2
PC7	RD#	OUT	Read Strobe	4-2

Depending on whether the alternate function is an input or output, the IO logic is slightly different, as shown in Figure 4-2 (output) and Figure 4-3 (input). The last column of Table 4-1 indicates which figure applies to each pin.

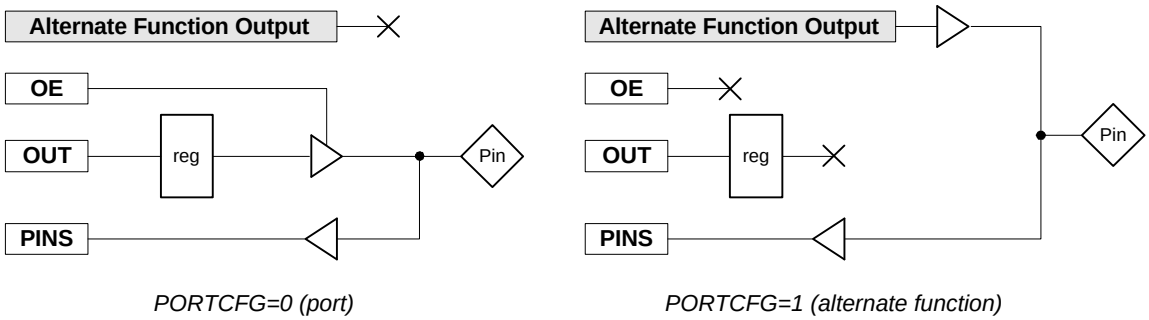


Figure 4-2. Alternate Function is an OUTPUT

Referring to Figure 4-2, when PORTCFG=0, the IO 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 IO 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.

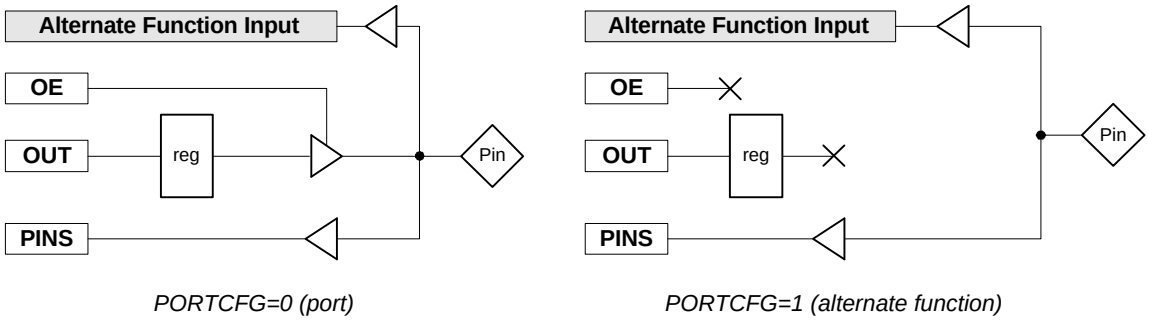


Figure 4-3. Alternate Function is an INPUT

Referring to Figure 4-3, when PORTCFG=0, the IO port is selected. This is the general IO 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 insure that if the alternate function is not used (if the pin is general-purpose IO only), the alternate input function is disabled.

For example, suppose the PB4/INT4 pin is configured for PB4. The pin signal is also routed to INT4. If INT4 is not used by the application, it should not be enabled. Alternatively, enabling INT4 could be useful, allowing IO bit PB4 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 IO Port Registers

PORTACFG	RxD1out	RxD0out	FRD	FWR	CS	OE	T1out	T0out
OUTA	D7	D6	D5	D4	D3	D2	D1	D0
PINSA	D7	D6	D5	D4	D3	D2	D1	D0
OEA	D7	D6	D5	D4	D3	D2	D1	D0
PORTBCFG	T2OUT	INT6	INT5	INT4	TxD1	RxD1	T2EX	T2
OUTB	D7	D6	D5	D4	D3	D2	D1	D0
PINSB	D7	D6	D5	D4	D3	D2	D1	D0
OEB	D7	D6	D5	D4	D3	D2	D1	D0
PORTCCFG	RD	WR	T1	T0	INT1	INT0	TxD0	RxD0
OUTC	D7	D6	D5	D4	D3	D2	D1	D0
PINSC	D7	D6	D5	D4	D3	D2	D1	D0
OEC	D7	D6	D5	D4	D3	D2	D1	D0

Figure 4-4. Registers associated with PORTS A,B,C

Figure 4-4 shows the registers associated with the EZ-USB IO ports. The power-on default for the PORTCFG bits is 0, selecting the IO port function. The power-on default for the OE bits is 0, selecting the input direction.

4.4 I²C Controller

The EZ-USB core contains an I²C controller for boot loading and general-purpose I²C bus interface. This controller uses the SCL (Serial Clock) and SDA (Serial Data) pins. Chapter 5, "Enumeration-ReNumeration™", describes how the boot load operates at power-on to read the contents of an external serial EEPROM in order to determine the initial EZ-USB 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 I²C controller is available to the 8051 for interface to external I²C devices such as other EEPROMS, IO chips, audio/video control chips, etc.

4.5 8051 I²C Controller

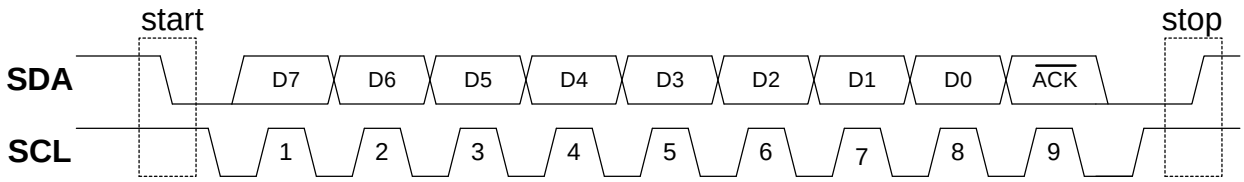


Figure 4-5. General I²C Transfer

Figure 4-5 illustrates the waveforms for an I²C transfer. SCL and SDA are open-drain EZ-USB pins, which must be pulled up to Vcc with external resistors. The EZ-USB chip is an I²C 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 I²C 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 4-5) the master (EZ-USB) floats the SDA line to allow the slave to acknowledge the transfer by pulling SDA low.

Multiple I²C Bus Masters

The EZ-USB chip acts only as an I²C bus master, never a slave. However, the 8051 can detect a second master by checking for BERR=1 (Section 4.5).

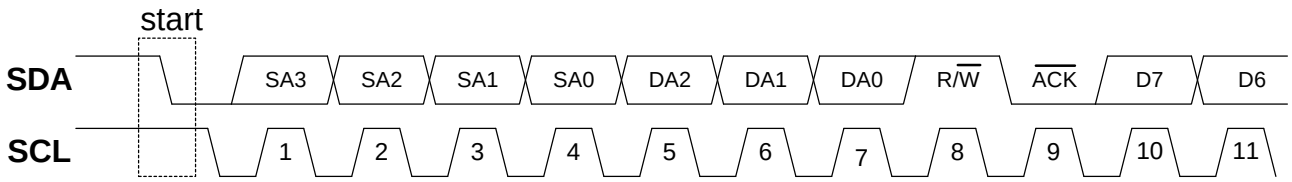


Figure 4-6. Addressing an I²C Peripheral

The first byte of an I²C bus transaction contains the address of the desired peripheral. Figure 4-6 shows the format for this first byte, which is sometimes called a *control* byte.

A master sends the bit sequence shown in Figure 4-6 after sending a START condition. The master uses this 9-bit sequence to select an I²C 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. I²C devices are pre-assigned slave addresses by device type, for example the slave address "1010" is assigned to EEPROMS. The three bits DA2-DA0 usually reflect the states of I²C 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 4-6, 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 the slave.

Assuming the 12.0 MHz crystal used by the EZ-USB family, the SCL frequency is 90.9 KHz, giving an I²C transfer rate of 11 microseconds per bit.

I2CS **I²C Control and Status** **7FA5**

b7	b6	b5	b4	b3	b2	b1	b0
START	**STOP**	**LASTRD**	**ID1**	**ID0**	**BERR**	**ACK**	**DONE**

I2DAT **I²C Data** **7FA6**

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

Figure 4-7. I²C Registers

The 8051 uses the two registers shown in Figure 4-7 to conduct I²C transfers. The 8051 transfers data to and from the I²C bus by writing and reading the I2DAT register. The I2CS register controls I²C 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.

4.6 Control Bits

4.6.1 START

The 8051 sets the START bit to 1 to prepare an I²C 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 data in the format shown in Figure 4-5 after setting the START bit. The I²C controller clears the START bit during the ACK interval (clock 9 in Figure 4-5).

4.6.2 STOP

The 8051 sets STOP=1 to terminate an I²C bus transfer. The I²C 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.

4.6.3 LASTRD

To read data over the I²C bus, an I²C 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^2C controller clears the LastRD bit at the end of the transfer (at ACK time).

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

4.7 Status Bits

After a byte transfer the EZ-USB 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.

4.7.1 DONE

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

4.7.2 ACK

Every ninth SCL of a write transfer the slave indicates reception of the byte by asserting ACK. The EZ-USB 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 EZ-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.

4.7.3 BERR

This bit indicates an I^2C 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.

4.7.4 ID1,ID0

These bits are set by the boot loader (Section 4.10) 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 4-3 on page 50 shows the encoding for these bits.

4.8 Sending I²C Data

To send a multiple byte data record over the I²C 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 4,5 for each byte until all bytes have been transferred.
7. Set STOP=1

*If the I²C interrupt (8051 INT 3) is enabled, each "Wait for DONE=1" step can be interrupt driven, and handled by an interrupt service routine. See "I²C Interrupt" in Chapter 9 for more details regarding the I²C interrupt.

4.9 Receiving I²C 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 then 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 then terminate by setting STOP=1.
6. Read the data from I2DAT. This initiates another read transfer.
7. Repeat steps 5,6 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 I²C bus.
10. Wait for DONE=1*. If BERR=1 then 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 I²C bus.

*If the I²C interrupt (8051 INT 3) is enabled, each "Wait for DONE=1" step can be interrupt-driven, and handled by an interrupt service routine. See "I²C Interrupt" in Chapter 9 for more details regarding the I²C interrupt.

4.10 I²C Boot Loader

When the EZ-USB chip comes out of reset, the EZ-USB boot loader checks for the presence of an EEPROM on its I²C 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 EZ-USB core or from the EEPROM). The various enumeration modes are described in Chapter 5, "Enumeration/ReNumeration".

Prior to reading the first EEPROM byte, the boot loader must set an address counter inside the EEPROM to zero. 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 boot loader supports two I²C EEPROM types:

1. EEPROMs with address A[7..4]=1010 that use an 8-bit address (example: 24LC00, LC01/A, LC02/A).
2. EEPROMs with address A[7..4]=1010 that use a 16-bit address (example: 24LC32, 24LC64).

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

The EZ-USB I²C 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.

Since there is no direct way to detect which EEPROM type—single or double address—is connected, the I²C 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 that the EEPROM address lines are strapped as shown in Table 4-2. Single-byte-address EEPROMS are strapped to address 000, and double-byte-address EEPROMS are strapped to address 001.

Table 4-2. 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 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 controller sends out a "read current address" command to I^2C 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 controller sends out a "read current address" command to I^2C 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 controller resets the EEPROM address pointer to zero (using the appropriate number of address bytes), then reads the first EEPROM byte. If does not read 0xB0 or 0xB2, the controller assumes the "No Serial EEPROM" mode. If it reads either 0xB0 or 0xB2, the controller copies the next six bytes into internal storage, and if it reads 0xB2, 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 4-3.

Table 4-3. Results of power-on I^2C 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 addresses of 1010) can be attached to the I^2C bus for general purpose 8051 use, as long as they are strapped for addresses other than 000 or 001. If a 24LC00 EEPROM is used, no other EEPROMS with device address 1010 may be used, since the 24LC00 responds to all eight sub-addresses.

5 EZ-USB Enumeration and ReNumeration™

5.1 Introduction

The EZ-USB 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 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 chip automatically enumerates as a USB device *without firmware*, so the USB interface itself may be used to download 8051 code and descriptor tables. The EZ-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 Anchor Device*.

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

An EZ-USB control bit called ReNum (for "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 EZ-USB core automatically handles device requests. Once the 8051 is running, it can set ReNum=1 to indicate that user 8051 code handles subsequent device requests using its downloaded firmware. Chapter 7, "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 EZ-USB core handle certain endpoint zero requests (see box, "Another Use for the Default Anchor Device").

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

Another Use for the Default Anchor Device

The Default Anchor Device is established at power-on to set up a USB device capable of downloading firmware into EZ-USB RAM. Another useful feature of the EZ-USB default device is that 8051 code can be written to support the already-configured Anchor Generic USB device. Before bringing the 8051 out of reset, the EZ-USB core enables certain endpoints and reports them to the host via descriptors. By utilizing the default Anchor 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 the polled bulk transfer example in the "EZ-USB Bulk Transfers" chapter.

5.2 The Default Anchor Device

The Default Anchor 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 uses no interrupt or isochronous bandwidth, as recommended by the USB specification.

Table 5-1. EZ-USB Default Endpoints

Endpoint	Type	Alternate Setting		
		0	**1**	**2**
		Max 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 purposes of downloading 8051 code, the Default Anchor Device requires only CONTROL endpoint zero. Nevertheless, the Anchor default machine is enhanced to support other endpoints as shown in Table 5-1 (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 automatically handled by the EZ-USB core, so the developer can immediately start writing code to transfer data over USB using these preconfigured endpoints.

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

Table 5-9 through Table 5-19 (pages 63-71) show the various descriptors returned to the host by the EZ-USB core when ReNum=0. These tables describe the USB endpoints defined in Table 5-1, along with other USB details, and should be useful to help understand the structure of USB descriptors.

5.3 EZ-USB Core Response to EP0 Device Requests

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

Table 5-2. How the EZ-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)
0x0C	Sync Frame	None
Vendor Requests		
0xA0	***Anchor Load***	Upload/Download RAM
0xA1-0xAF	***Reserved***	Reserved by Anchor Chips
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 EZ-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.

5.4 Anchor Load

The USB specification provides for *vendor-specific requests* to be sent over CONTROL endpoint zero. The EZ-USB chip uses this feature to transfer data between the host and EZ-USB RAM. The EZ-USB core responds to two "Anchor Load" requests, as shown in Table 5-3 and Table 5-4 below:

Table 5-3. Anchor Download

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**0x40**	Vendor Request, OUT	*None required*
1	bRequest	**0xA0**	"Anchor 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 5-4. Anchor Upload

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**0xC0**	Vendor Request, IN	*None required*
1	bRequest	**0xA0**	"Anchor 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 EZ-USB core (ReNum=0 or 1). This means that 0xA0 is 'reserved' by the EZ-USB chip, and therefore should never be used for a vendor request. Anchor Chips also reserves bRequest values 0xA1 through 0xAF, so 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 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 Anchor Download command.

Anchor loads are restricted to internal EZ-USB memory.

When Renum=1 at Power-On

At power-on, the ReNum bit is normally set to zero so that the EZ-USB handles 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 EZ-USB core checks the I^2C bus for the presence of an EEPROM. If it finds one, and the first byte of the EEPROM is 0xB2, 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 firmware in RAM. The required data format for this load module is described in the next section.

5.5 Enumeration Modes

When the EZ-USB chip comes out of reset, the EZ-USB core makes a decision about how to enumerate based on the contents of an external EEPROM on its I^2C 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. EZ-USB Core Action at Power-Up

First EEPROM byte	EZ-USB Core Action
Not 0xB0 or 0xB2	Supplies descriptors, PID/VID/DID from EZ-USB Core. Sets ReNum=0.
0xB0	Supplies descriptors from EZ-USB core, PID/VID/DID from EEPROM. Sets ReNum=0.
0xB2	Loads EEPROM into EZ-USB 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 0xB0 nor 0xB2, the EZ-USB core enumerates using internally stored descriptor data, which contains the Anchor Chips VID, PID and DID. These ID bytes cause the host operating system to load an Anchor Chips device driver. The EZ-USB core also establishes the *Anchor Default USB device*. This mode is only used for code development and debug.

If a serial EEPROM is attached to the I^2C bus and its first byte is 0xB0, the EZ-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 EZ-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 operating mode provides a *soft* USB device using ReNumeration™.

If a serial EEPROM is attached to the I^2C bus and its first byte is 0xB2, the EZ-USB core transfers the contents of the EEPROM into internal RAM. The EZ-USB core also sets the ReNum bit to 1 to indicate that the 8051 (and not the EZ-USB core) responds to device requests over CONTROL endpoint zero (see box, “When ReNum=1 at Power-On”). 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 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 Story

The EZ-USB I^2C 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 controller for general-purpose use. This makes a wide range of standard I^2C peripherals available to an EZ-USB system.

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

5.6 No Serial EEPROM

In the simplest case, no serial EEPROM is present on the I^2C bus, or an EEPROM is present but its first byte is not 0xB0 or 0xB2. In this case, descriptor data is supplied by a table internal to the EZ-USB core. The EZ-USB chip comes on as the *Anchor Default Device*, with the ID bytes shown in Table 5-6.

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

Vendor ID	0x0547 (Anchor Chips)
Product ID	0x2131 (EZ-USB)
Device Release	0xXXYY(depends on revision)

The USB host queries the device during enumeration, reads the device descriptor, and uses the Table 5-6 bytes 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 case is the simplest configuration, but also the most limiting. This mode is used only for code development, utilizing Anchor software tools matched to the ID values in Table 5-6.

Reminder:

The EZ-USB core uses the Table 5-6 data 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 0xB0

Table 5-7. EEPROM Data Format for "B0" Load

EEPROM Address	Contents
0	**0xB0**
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-	not used

If at power-on, the EZ-USB core detects an EEPROM connected to its I^2C port with the value **0xB0** at address 0, the EZ-USB core copies the Vendor ID (VID), Product ID (PID) and Device ID (DID) from the EEPROM (Table 5-7) into internal storage. The EZ-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 Anchor device descriptor). This causes a driver matched to the VID/PID/DID values in the EEPROM, instead of those in the EZ-USB core, to be loaded into the OS.

After initial enumeration, the driver downloads 8051 code and USB descriptor data into EZ-USB RAM and starts the 8051. The code then ReNumerates™ to come 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 seven bytes are used.

5.8 Serial EEPROM Present, First Byte is 0xB2

If at power-on, the EZ-USB core detects an EEPROM connected to its I^2C port with the value **0xB2** at address 0, the EZ-USB core loads the EEPROM data into EZ-USB RAM. It also sets the ReNum bit to 1, causing device requests to be fielded by the 8051 instead of the EZ-USB core. The EEPROM data format is shown in Table 5-8.

Table 5-8. EEPROM Data Format for "B2" Load

EEPROM Address	Contents
0	**0xB2**
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	Length H
8	Length L
9	StartAddr H
10	StartAddr L
---	Data block

---	Length H
---	Length L
---	StartAddr H
---	StartAddr L
---	Data block

---	0x80
---	0x01
---	0x7F
---	0x92
last	00000000

The first byte tells the EZ-USB core to copy EEPROM data into RAM. The next six bytes (1-6) are ignored (see box).

One or more data records follow, starting at EEPROM address 7. 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 RAM spaces only:

1. 8051 program/data RAM at 0x0000-0x1B40
2. The CPUCS register at 0x7F92 (only bit 0, 8051 RESET, is host-loadable).

VID/PID/DID in a "B2" EEPROM

Bytes 1-6 of a "B2" EEPROM can be loaded with VID/PID/DID bytes if it is desired at some point to run the 8051 program with ReNum=0 (EZ-USB core handles device requests), using the EEPROM VID/PID/DID rather than the Anchor Chips values built into the EZ-USB core.

5.9 ReNumeration™

Three EZ-USB 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 5-1, USB Control and Status Register

The inverted value of DISCON appears on the EZ-USB DISCON# pin. The DISCOE pin tri-states the DISCON# pin when DISCOE=0.

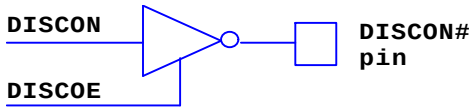


Figure 5-2. Disconnect pin logic

To simulate a USB disconnect, the 8051 first sets DISCON=0 to present a logic HI on the DISCON# pin when the output is enabled. When DISCOE=1, the DISCON# pin drives high, supplying the required voltage to the 1500 ohm pullup resistor on the D+ line. To simulate a disconnect, the 8051 sets DISCOE=0 to float the DISCON# pin. Then the 8051 sets RENUM=1 to indicate that the 8051, and not the USB core, will handle USB requests. Then the 8051 re-connects by setting DISCOE=1 which drives the DISCON# pin HI and re-indicates the device presence on the bus. This arrangement allows connecting the 1500 ohm resistor directly between the DISCON# pin and the USB D+ line (Figure 5-3).

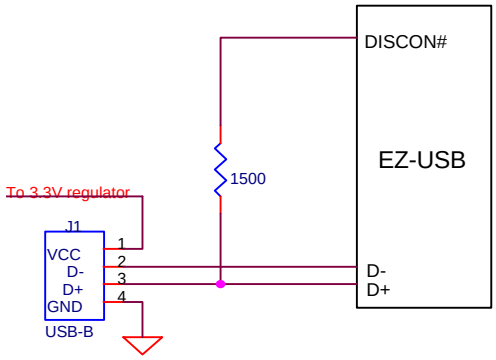


Figure 5-3. Typical Disconnect Circuit (DISCOE=1)

5.10 Multiple ReNumerations™

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.

5.11 Default Descriptor

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

Table 5-9. Anchor 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 spec version 1.00 (L)	00H
3	bcdUSB (H)	USB spec version 1.00 (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	Max packet size for EP0 = **64 bytes**	40H
8	idVendor (L)	Vendor id (L) Anchor Chips Inc. = 0547H	47H
9	idVendor (H)	Vendor id (H)	05H
10	idProduct (L)	Product id (L) EZ-USB = 2131H	31H
11	idProduct (H)	Product id (H)	21H
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 Anchor Chips Vendor, Product and Release Number ID's, and uses no string indices. Release Number ID's ("XX" and "YY") are found in individual Anchor Chips data sheets. The EZ-USB core returns this information in response to a "Get_Descriptor/Device" host request.

Table 5-10. Anchor 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 & 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 interface	01H
6	iConfiguration	Index of string describing this configuration = none	00H
7	bmAttributes	Attributes - bus powered, no wakeup	80H
8	MaxPower	Max 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 request specifying configuration #1 (offset 5).

Table 5-11. Anchor 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 EP0) = **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 is a “zero bandwidth” setting. The interface has no string index.

Table 5-12. Anchor 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 EP0) = **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. Anchor Default Interface 0, Alternate Setting 1, Interrupt 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)	max packet size = **16 bytes**	10H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds = **10 msec**	0AH

Interface 0, alternate setting 1 has one interrupt endpoint, IN1, which has a max packet size of 16 and a polling interval of 10 milliseconds.

Table 5-14. Anchor 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)	max packet size = **64 bytes**	40H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds (1 for iso)	00H
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 = **OUT2**	02H
3	bmAttributes	xfr type = **BULK**	02H
4	wMaxPacketSize (L)	max packet size = **64 bytes**	40H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds (1 for iso)	00H
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 = **IN4**	84H
3	bmAttributes	xfr type = **BULK**	02H
4	wMaxPacketSize (L)	max packet size = **64 bytes**	40H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds (1 for iso)	00H
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 = **OUT4**	04H
3	bmAttributes	xfr type = **BULK**	02H
4	wMaxPacketSize (L)	max packet size = **64 bytes**	40H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds (1 for iso)	00H
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 = **IN6**	86H
3	bmAttributes	xfr type = **BULK**	02H
4	wMaxPacketSize (L)	max packet size = **64 bytes**	40H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds (1 for iso)	00H
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 = **OUT6**	06H
3	bmAttributes	xfr type = **BULK**	02H
4	wMaxPacketSize (L)	max packet size = **64 bytes**	40H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds (1 for iso)	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, Bulk Transfers.)

Table 5-15. Anchor Default Interface 0, Alternate Setting 1, Isochronous 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)	max packet size = **16 bytes**	10H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds (1 for iso)	01H
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 = **OUT8**	08H
3	bmAttributes	xfr type = **ISO**	01H
4	wMaxPacketSize (L)	max packet size = **16 bytes**	10H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds (1 for iso)	01H
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 = **IN9**	89H
3	bmAttributes	xfr type = **ISO**	01H
4	wMaxPacketSize (L)	max packet size = **16 bytes**	10H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds (1 for iso)	01H
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 = **OUT9**	09H
3	bmAttributes	xfr type = **ISO**	01H
4	wMaxPacketSize (L)	max packet size = **16 bytes**	10H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds (1 for iso)	01H
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 = **IN10**	8AH
3	bmAttributes	xfr type = **ISO**	01H
4	wMaxPacketSize (L)	max packet size = **16 bytes**	10H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds (1 for iso)	01H
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 = **OUT10**	0AH
3	bmAttributes	xfr type = **ISO**	01H
4	wMaxPacketSize (L)	max packet size = **16 bytes**	10H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds (1 for iso)	01H

Interface 0, alternate setting 1 has six isochronous endpoints with max packet sizes of 16 bytes. This is a “low bandwidth” setting.

Table 5-16. Anchor Default Interface 0, Alternate Setting 2 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 = **2**	02H
4	bNumEndpoints	Number of endpoints (not counting EP0) = **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 2 has thirteen endpoints, whose individual descriptors follow the interface descriptor. Alternate setting 2 differs from alternate setting 1 in the max packet sizes of its interrupt endpoint and two of its isochronous endpoints (EP8IN and EP8OUT).

Table 5-17. Anchor Default Interface 0, Alternate Setting 2, Interrupt 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)	max packet size = **64 bytes**	40H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds = **10 msec**	0AH

Alternate setting 2 for the interrupt endpoint 1-IN increases the max packet size for the interrupt endpoint to 64.

Table 5-18. Anchor Default Interface 0, Alternate Setting 2, 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)	max packet size = **64 bytes**	40H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds (1 for iso)	00H
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 = **OUT2**	02H
3	bmAttributes	xfr type = **BULK**	02H
4	wMaxPacketSize (L)	max packet size = **64 bytes**	40H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds (1 for iso)	00H
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 = **IN4**	84H
3	bmAttributes	xfr type = **BULK**	02H
4	wMaxPacketSize (L)	max packet size = **64 bytes**	40H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds (1 for iso)	00H
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 = **OUT4**	04H
3	bmAttributes	xfr type = **BULK**	02H
4	wMaxPacketSize (L)	max packet size = **64 bytes**	40H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds (1 for iso)	00H
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 = **IN6**	86H
3	bmAttributes	xfr type = **BULK**	02H
4	wMaxPacketSize (L)	max packet size = **64 bytes**	40H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds (1 for iso)	00H
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 = **OUT6**	06H
3	bmAttributes	xfr type = **BULK**	02H
4	wMaxPacketSize (L)	max packet size = **64 bytes**	40H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds (1 for iso)	00H

The bulk endpoints for alternate setting 2 are identical to alternate setting 1.

Table 5-19. Anchor Default Interface 0, Alternate Setting 2, Isochronous 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)	max packet size = **256 bytes**	00H
5	WMaxPacketSize (H)	max packet size - high	01H
6	bInterval	polling interval in milliseconds (1 for iso)	01H
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 = **OUT8**	08H
3	bmAttributes	xfr type = **ISO**	01H
4	wMaxPacketSize (L)	max packet size = **256 bytes**	00H
5	WMaxPacketSize (H)	max packet size - high	10H
6	bInterval	polling interval in milliseconds (1 for iso)	01H
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 = **IN9**	89H
3	bmAttributes	xfr type = **ISO**	01H
4	wMaxPacketSize (L)	max packet size = **16 bytes**	10H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds (1 for iso)	01H
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 = **OUT9**	09H
3	bmAttributes	xfr type = **ISO**	01H
4	wMaxPacketSize (L)	max packet size = **16 bytes**	10H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds (1 for iso)	01H
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 = **IN10**	8AH
3	bmAttributes	xfr type = **ISO**	01H
4	wMaxPacketSize (L)	max packet size = **16 bytes**	10H
5	WMaxPacketSize (H)	max packet size - high	00H
6	bInterval	polling interval in milliseconds (1 for iso)	01H
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 = **OUT10**	0AH
3	bmAttributes	xfr type = **ISO**	01H
4	wMaxPacketSize (L)	max packet size = **16 bytes**	10H
5	WMaxPacketSize (H)	max 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 max packet sizes for EP8IN and EP8OUT. This is a “high bandwidth” setting using 256 bytes each.

6 EZ-USB Bulk Transfers

6.1 Introduction

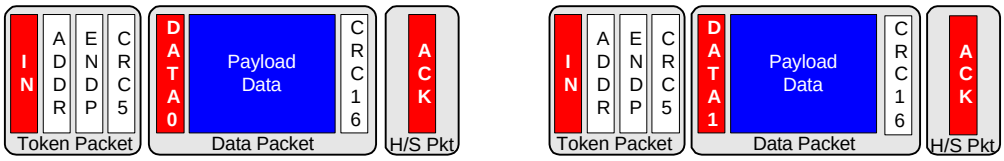


Figure 6-1. Two BULK transfers, IN and OUT

EZ-USB 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 CONTROL endpoint zero is described in Chapter 7.

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

Endpoint	Direc	Type	Size
0	Bidir	Control	64/64
1	IN	Bulk/Int	64
1	OUT	Bulk	64
2	IN	Bulk/Int	64
2	OUT	Bulk	64
3	IN	Bulk/Int	64
3	OUT	Bulk	64
4	IN	Bulk/Int	64
4	OUT	Bulk	64
5	IN	Bulk/Int	64
5	OUT	Bulk	64
6	IN	Bulk/Int	64
6	OUT	Bulk	64
7	IN	Bulk/Int	64
7	OUT	Bulk	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 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, which 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 EZ-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 byte count register, and a two-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 EZ-USB core sets an interrupt request bit. The EZ-USB 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 9, "Interrupts" fully describes this mechanism.

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

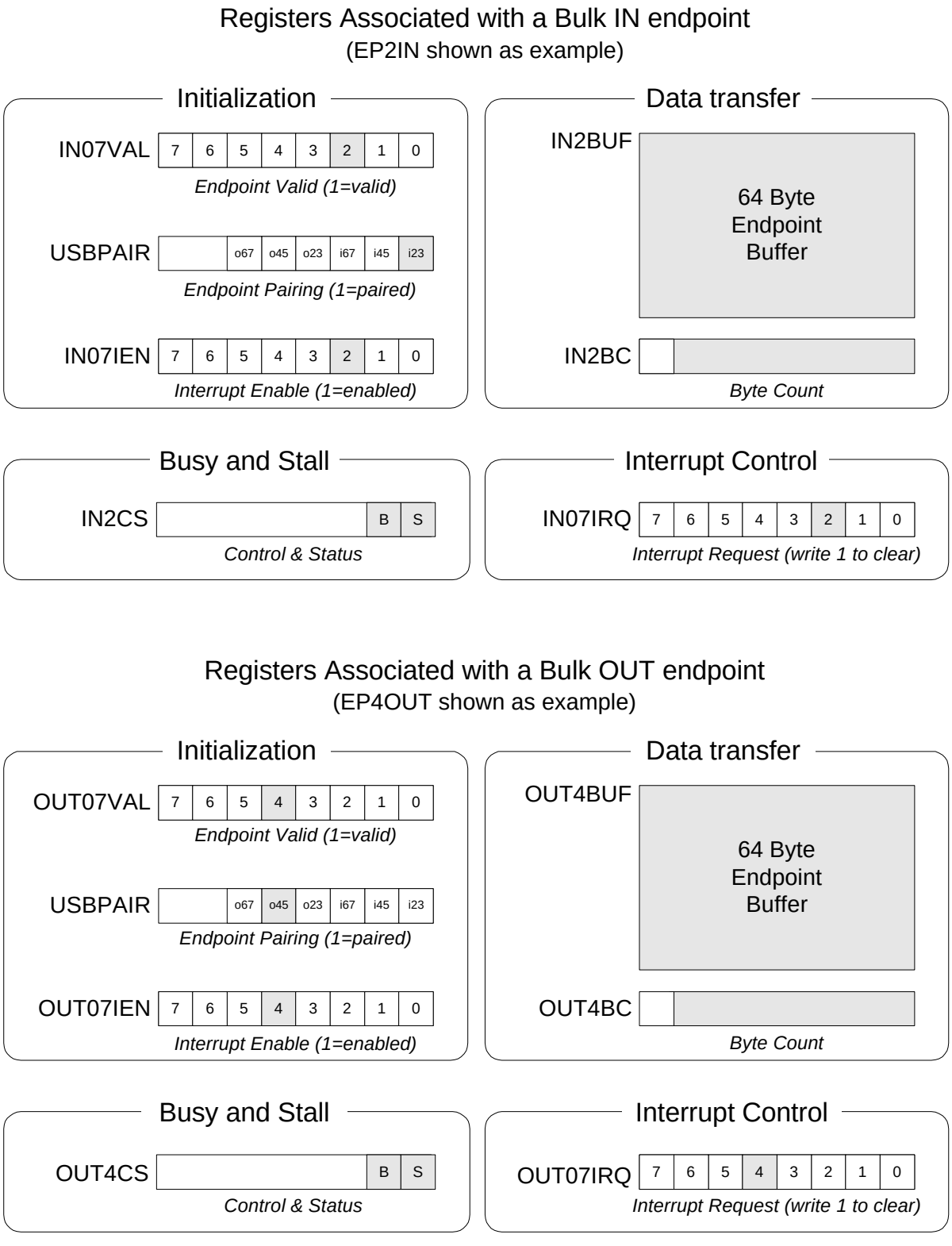


Figure 6-2. Registers Associated with Bulk Endpoints

6.2 Bulk IN Transfers

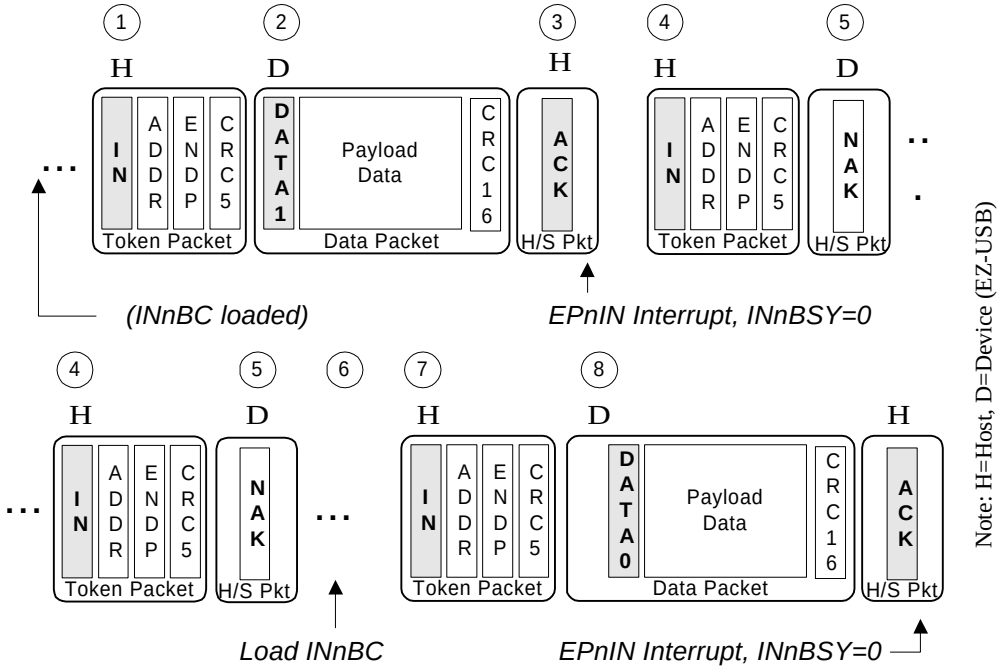


Figure 6-3. Anatomy of a Bulk IN Transfer

USB bulk "IN" data travels from device to host. The host requests an IN transfer by issuing an IN token to the EZ-USB core, which responds with data when it is ready. The 8051 indicates 'ready' by loading the endpoint's byte count register. If the EZ-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 EZ-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 EZ-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, 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 EZ-USB core transfers the next data packet ⑧.

6.3 Interrupt Transfers

Interrupt transfers are handled just like bulk IN 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 tokens to the interrupt endpoint.

The 8051 services an interrupt endpoint exactly the same way as a bulk IN endpoint. It loads data into the endpoint buffer, then loads a byte count. When the data has been successfully transferred, it gets an interrupt. The EZ-USB device does not know the difference between an interrupt transfer and a bulk IN transfer.

6.4 EZ-USB Bulk IN Example

Suppose 220 bytes are to be transferred to the host using endpoint 6-IN. Further assume that a 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 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 EZ-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 9, "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 Note:

When the EZ-USB chip comes out of RESET, or when the USB host issues a bus reset, the EZ-USB core "unarms" IN endpoints 1-7 by setting their busy bits to 1. Any IN transfer requests are NAK'd until the 8051 loads the appropriate INxBC register(s). The endpoint valid bits are not affected by an 8051reset or a USB reset. Chapter 10, "Reset and Power Management" 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, followed by a packet of data. The EZ-USB core then responds with an ACK, if it correctly received the data. If the endpoint buffer is not ready to accept data, the EZ-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 EZ-USB core responds with an ACK.

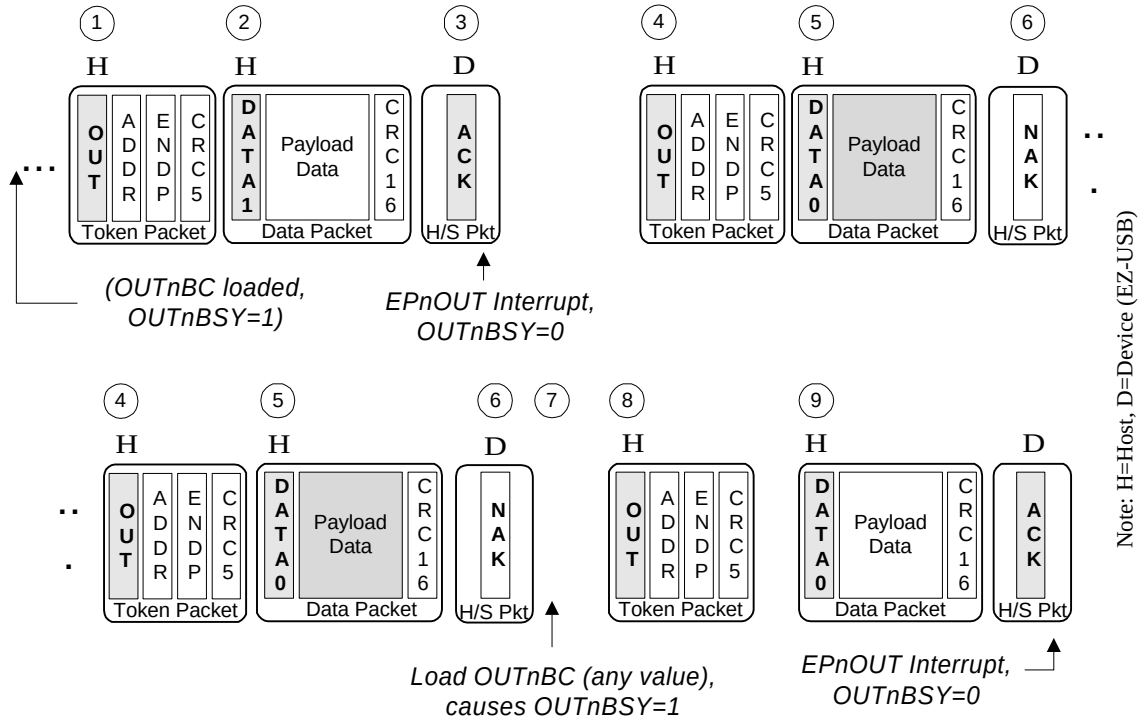


Figure 6-4. Anatomy of a Bulk OUT Transfer

Each EZ-USB 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 EZ-USB core that it 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 require 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 EZ-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 host issues a NAK, indicating 'busy' ⑥. The data at ⑤ is shaded to indicate that the EZ-USB core discards it, and does not over-write the data in the endpoint's OUT buffer.

The host continues to send OUT tokens ④,⑤,⑥ that are greeted by NAK's 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 EZ-USB core. Once armed, when the next OUT token arrives ⑧ the EZ-USB core accepts the next data packet ⑨.

Initializing OUT Endpoints

When the EZ-USB chip comes out of reset, or when the USB host issues a bus reset, the EZ-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 NAK'd 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 EZ-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

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

Figure 6-5. Endpoint Pairing Bits (in the USBPAIR register)

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.

Important Note:
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.

An 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 1-0; 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 Using Bulk Buffer Memory

Table 6-2. EZ-USB 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 6-2 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 register space so that any buffers not used for endpoints can be reclaimed as general purpose data RAM. The top of memory for the 8K EZ-USB 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 gives full details of the EZ-USB memory map.

Note:
Anchor uploads or downloads to unused bulk memory can be done only at the "Mirrored" (low) addresses shown in Table 6-2.

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", 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 IO 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.

Note:
At the present writing 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. Select the endpoint by writing the value 000d0eee to the TOGCTL register, where d is the direction and eee is the endpoint number.

2. Clear the toggle bit by writing the value 001d0eee to the TOGCTL register.

After step 1, the 8051 may read the state of the data toggle by reading the TOGCTL register and checking bit 7.

6.11 Polled Bulk Transfer Example

The following code illustrates the EZ-USB 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.

```
1   start:      mov     SP,#STACK-1     ; set stack
2               mov     dptr,#IN2BUF    ; fill EP2IN buffer with
3               mov     r7,#64          ; decrementing counter
4   fill:       mov     a,r7
5               movx    @dptr,a
6               inc     dptr
7               djnz    r7,fill
8   ;
9               mov     r1,#0           ; r1 is IN token counter
10              mov     r2,#0           ; r2 is OUT token counter
11              mov     dptr,#IN2BC     ; Point to EP2 Byte Count register
12              mov     a,#40h          ; 64-byte transfer
13              movx    @dptr,a         ; arm the IN2 transfer
14  ;
15  loop:       mov     dptr,#IN2CS     ; poll the EP2-IN Status
16  movx        a,@dptr
17              jnb     acc.1,serviceIN2       ; not busy--service it
18              mov     dptr,#OUT2CS
19              movx    a,@dptr
20             jb       acc.1,loop      ; EP2OUT is busy--keep looping
21  ;
22  serviceOUT2:
23              inc     r2              ; OUT packet counter
24              mov     dptr,#OUT2BC    ; load byte count register to re-arm
25              movx    @dptr,a         ; (any value)
26              sjmp    loop
27  ;
28  serviceIN2:
29              inc     r1              ; IN packet counter
30              mov     dptr,#IN2BUF    ; update the first data byte
31              mov     a,r1            ; in EP2IN buffer
32              movx    @dptr,a
33              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 endpont 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 since it tells the EZ-USB core how many bytes to transfer in response to the next IN token. The 8051 can load any byte count OUT transfers, since 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 to the polling loop.

6.12 Enumeration Note

The code in this example is complete, and runs on the EZ-USB 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 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 the chapter on Enumeration.

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 EZ-USB core inserts a special byte at location 0x45, which directs the jump instruction to a table of jump instructions which transfer control the endpoint-specific ISR.

Table 6-3. 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 6-4 shows all INT2 vectors, with the bulk endpoint vectors unshaded. The shaded interrupts apply to all the bulk endpoints.

Table 6-4. Byte inserted by EZ-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**
EP1-OUT	**0x24**
EP2-IN	**0x28**
EP2-OUT	**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 since 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.

Each bulk endpoint interrupt has an associated interrupt enable bit (in IN07IEN and OUT07IEN), and an interrupt request bit (in IN07IRQ and OUT07IRQ). The interrupt request bits are set by the EZ-USB hardware, and must be cleared by software in the interrupt service routine. **IRQ bits are cleared by writing a "1"**. Since all USB registers are accessed using 'movx @dptr' instructions, USB interrupt service routines must save and restore both data pointers, the DPS register, and the accumulator before clearing interrupt request bits.

NOTE: Any USB ISR should clear the 8051 INT2 interrupt request bit before clearing any of the EZ-USB endpoint IRQ bits, to avoid losing interrupts. Interrupts are discussed in more detail in Chapter 9.

Note:
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 '00000100' to IN07IRQ. This will not disturb the other interrupt request bits. ***Do not* read the contents of IN07IRQ, logical-OR the contents with 02, and write it back.** This will clear all other pending interrupts since you are writing 1's to them.

6.14 Interrupt Bulk Transfer Example

This 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    SPARE_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 (xx00).

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; the jump table in step 1 will automatically jump to it.

```
; ------------------------------------------------------------
; USB Interrupt Service Routine
; ------------------------------------------------------------
EP6OUT_ISR:    push    dps            ; save both dptrs, dps and acc
               push    dpl
               push    dph
               push    dpl1
               push    dph1
               push    acc
;
               mov     a,EXIF         ; clear USB IRQ (INT2)
               clr     acc.4
               mov     EXIF,a
;
               mov     dptr,#OUT07IRQ
               mov     a,#01000000b   ; a "1" clears the IRQ bit
               movx    @dptr,a        ; clear OUT6 int request
               setb    got_EP6_data   ; set my flag
;
               pop     acc            ; restore vital registers
               pop     dph1
               pop     dpl1
               pop     dph
               pop     dpl
               pop     dps
               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. Note that both data pointers and the DPS (Data Pointer Select) registers must be saved and restored in addition to the accumulator.

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 Anchor 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 the 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 6-9. 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.

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
;
                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 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 this example is complete, and runs on the EZ-USB 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 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 the chapter on Enumeration.

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 registers used for bulk transfers.

6.16 The Autopointer

Bulk endpoint data is available in 64-byte buffers in EZ-USB 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 which automatically increments when data is transferred. Using this Autopointer, the 8051 can access any contiguous block of internal EZ-USB RAM 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 any time. 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 penultimate line indicates, the Autopointer saves an "inc dptr" instruction which would be necessary if one of the 8051 data pointers were used to access the OUT4BUF RAM data. This improves the transfer time.

Fastest bulk transfer speed in and out of EZ-USB bulk buffers is achieved when the Autopointer is used in conjunction with the EZ-USB Fast Transfer mode.

As described in Chapter 8, "EZ-USB Isochronous Transfers", the EZ-USB core provides a method for transferring data directly between an internal FIFO and external memory in two 8051 cycles (333 ns). The fast transfer mode is active for bulk data when:

- The 8051 sets FBLK=1 in the FASTXFR register, enabling fast bulk transfers,
- The 8051 DPTR points to the AUTODATA register, and
- The 8051 executes a 'movx a,@dptr' or a 'movx @dptr,a' instruction.

The 8051 code example in Figure 6-14 shows a transfer loop for moving 64 bytes of external FIFO data into the endpoint 4-IN buffer. The FASTXFR register bits are explained in Chapter 8, "EZ-USB Isochronous Transfers".

Note:
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 8.9.1 on page 133 for a description of the ISODISAB bit.

```
        mov     dptr,#FASTXFR  ; set up the fast BULK transfer mode
        mov     a,#01000000b   ; FBLK=1, RPOL=0, RM1-0 = 00
        movx    @dptr,a        ; load the FASTXFR register
Init:   mov     dptr,#AUTOPTRH
        mov     a,HIGH(IN4BUF) ; High portion of IN4BUF
        movx    @dptr,a        ; Load AUTOPTRH
        mov     dptr,#AUTOPTRL
        mov     a,LOW(IN4BUF)  ; Low portion of IN4BUF buffer address
        movx    @dptr,a        ; Load AUTOPTRL
        mov     dptr,#AUTODATA ; point to the 'fifo' register
        mov     r7,#8          ; r7 is loop counter, 8 bytes per loop
;
loop:   movx    @dptr,a        ; (2) write IN 'fifo' using byte from external bus
        movx    @dptr,a        ; (2) again
        movx    @dptr,a        ; (2) again
        movx    @dptr,a        ; (2) again
        movx    @dptr,a        ; (2) again
        movx    @dptr,a        ; (2) again
        movx    @dptr,a        ; (2) again
        movx    @dptr,a        ; (2) again
        djnz    r7,loop        ; (3) do eight more, 'r7' times
```

Figure 6-14. 8051 code to transfer external data to a Bulk IN Buffer

This transfer loop takes 19 cycles per loop times 8 passes, or 22 microseconds (152 cycles). A USB bulk transfer of 64 bytes takes over 42 microseconds (64 * 8 * 83ns) of bus time to transfer the data bytes to or from the host. This calculation neglects USB overhead time.

From this simple example it is clear that by using the Autopointer and the EZ-USB Fast Transfer mode, the 8051 can transfer data in and out of EZ-USB endpoint buffers significantly faster than the USB can transfer it to and from the host. This means that the EZ-USB chip should never be a speed bottleneck in a USB system. 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.

7 EZ-USB Endpoint Zero

7.1 Introduction

Endpoint Zero has special significance in a USB system. It is a CONTROL endpoint, and 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 repertoire 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 chip handles endpoint zero requests.

Because the EZ-USB chip can enumerate without firmware (Chapter 5, "Enumeration/ReNumeration"), the EZ-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), and describes EZ-USB resources such as the Setup Data Pointer that simplify 8051 code that handles endpoint zero requests.

Endpoint zero is the only CONTROL endpoint in the EZ-USB chip. Although CONTROL endpoints are "bi-directional" the EZ-USB chip provides two 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 having to keep track of the three CONTROL transfer phases—SETUP, DATA and STATUS. The EZ-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, so the valid bits (LSB of IN07VAL and OUT07VAL registers) are permanently set to 1. Writing any value to these two bits has no effect, and 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 endpoints 0 or 1.

7.2 Control Endpoint EP0

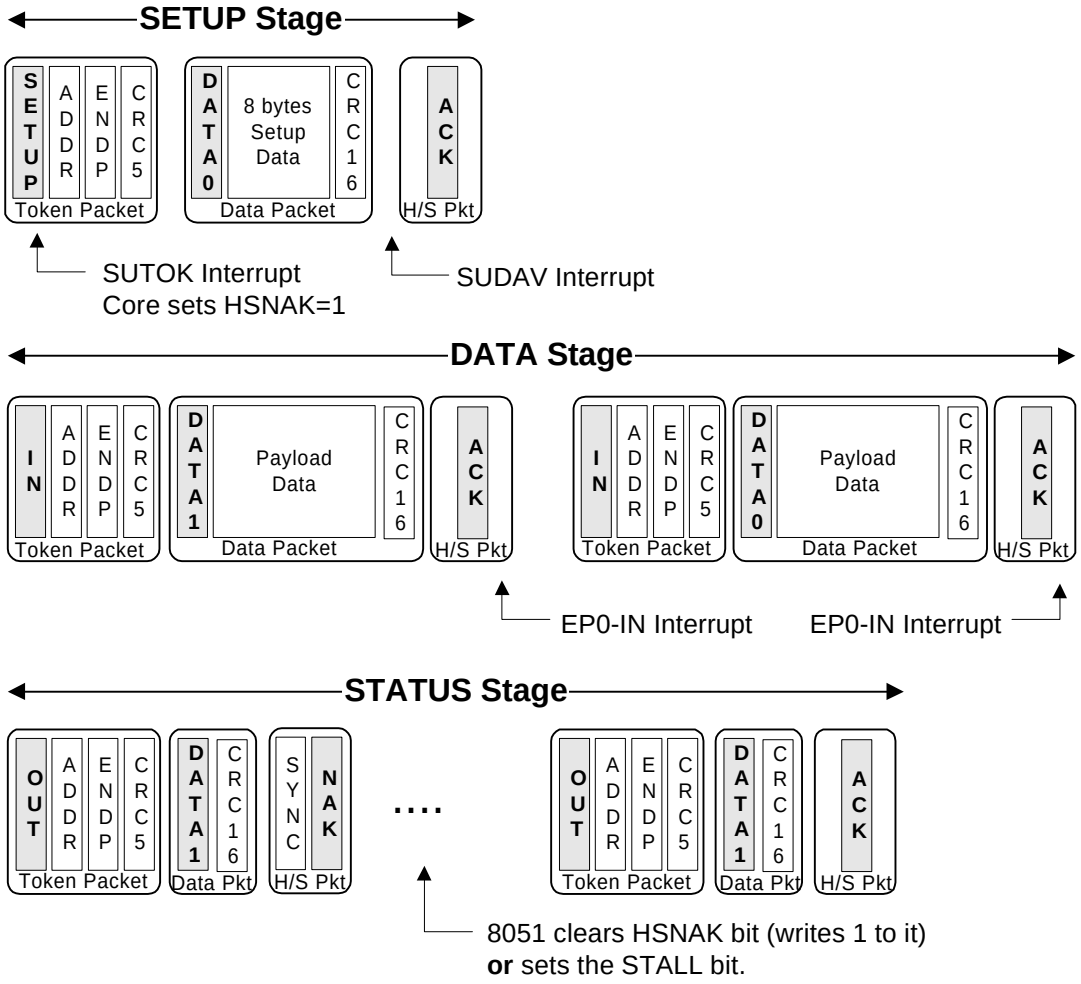


Figure 7-1. A USB Control Transfer. This one has a data stage.

Endpoint zero accepts a special SETUP packet, which contains an eight-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 eight-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 7-1 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 the 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

called HSNAK (Handshake NAK) to clear it and instruct the EZ-USB core to ACK the STATUS stage.

The HSNAK bit is used to hold 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 EZ-USB core responds with NAK's, 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 is non-zero.

Two 8051 interrupts provide notification that a SETUP packet has arrived, as shown in Figure 7-2.

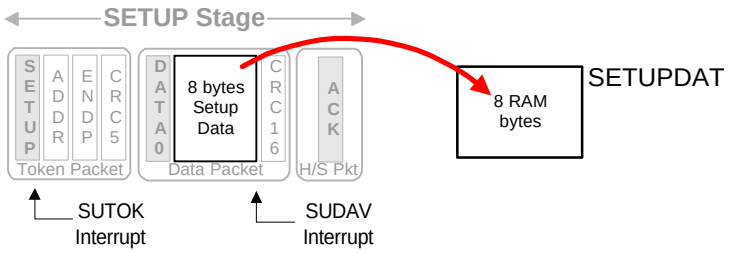


Figure 7-2. The two interrupts associated with EP0 CONTROL transfers

The EZ-USB core sets the SUTOKIR bit (SETUP Token Interrupt Request) when the EZ-USB core detects the SETUP token at the beginning of a CONTROL transfer. This interrupt is normally used only for debug.

The EZ-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 registers starting at SETUPDAT. The EZ-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 EZ-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 NAK'ed. 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 EZ-USB core automatically clears this stall bit when the next SETUP token arrives.

Like all EZ-USB interrupt requests, the SUTOKIR and SUDAVIR bits can be directly tested and reset by the CPU (they are reset by writing a "1"). Thus if the corresponding interrupt enable bits are zero, the interrupt request conditions can still be directly polled.

Figure 7-3 shows the EZ-USB registers that deal with CONTROL transactions over EP0.

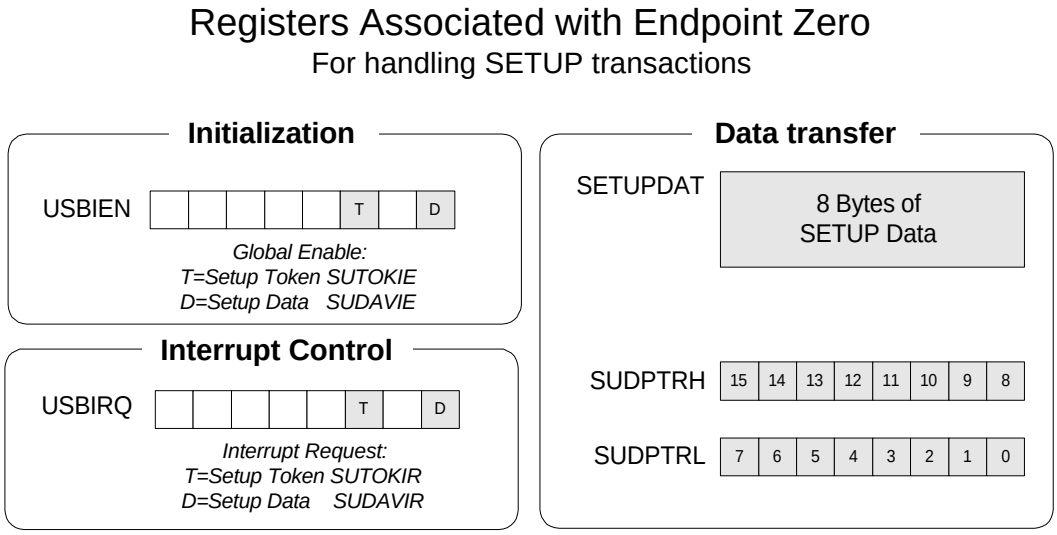


Figure 7-3. Registers associated with EP0 Control Transfers

These registers augment those associated with normal bulk transfers over endpoint zero, which are described in Chapter 6, “Bulk Transfers”.

Two bits in the USBIEN (USB Interrupt Enable) register enable the SETUP Token (SUTOKIE) and SETUP Data (SUDAVIE) 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 EZ-USB core transfers the eight SETUP bytes into eight bytes of RAM at SETUPDAT. A sixteen bit pointer, SUDPTRH/L gives hardware assistance for handling CONTROL IN transfers, in particular the USB “Get Descriptor” requests described later in this chapter.

7.3 USB Requests

The USB Specification defines a set of *Standard Device Requests* in Chapter 9, “USB Device Framework”. When the 8051 is in control (ReNum=1), the EZ-USB core handles one of these requests (Set Address) directly, and relies on the 8051 to support the others. The 8051 acts on device requests by decoding the eight bytes contained in the SETUP packet. Table 7-1 shows the meaning of these eight bytes.

Table 7-1. 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 7-2)
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 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 7-2 (page 101) shows the different values defined for bRequest, and how the 8051 responds to each request. The remainder of this chapter describes each of the Table 7-2 requests in detail.

Note: Table 7-2 applies when ReNum=1, which signifies that the 8051, and not the EZ-USB core, handles device requests. Table 5-2 on page 53 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 7-2. 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 (Anchor Load)		Up/Download RAM	---
0xA1 – 0xAF		SUDAV Interrupt	Reserved by Anchor Chips
All except 0xA0		SUDAV Interrupt	Depends on application

In the ReNumerated condition (ReNum=1), the EZ-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 EZ-USB core implements one vendor-specific request, namely “Anchor 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 Anchor Load feature, be sure to refrain from using the bRequest value 0xA0 for your custom requests. The Anchor Load feature is fully described in Chapter 5, “Enumeration and ReNumeration”.

Note:
To avoid future incompatibilities, vendor requests A0-AF (hex) are reserved by Anchor Chips.

7.3.1 Get Status

The USB Specification version 1.0 defines three USB status requests. A fourth request, to an interface, is indicated in the spec as "reserved". The four status requests are:

1. Remote Wakeup (Device request)
2. Self-Powered (Device request)
3. Stall (Endpoint request)
4. Interface request ("reserved")

The EZ-USB core activates the SUDAV interrupt request to tell the 8051 to decode the SETUP packet and supply the appropriate status information.

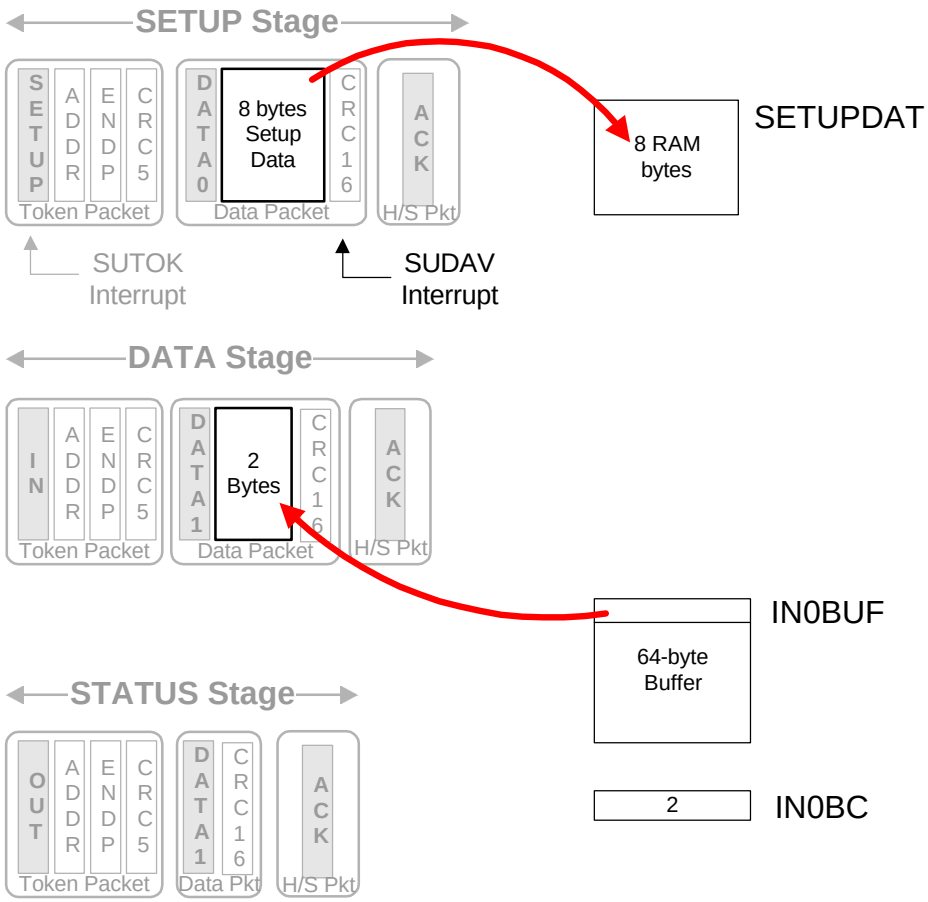


Figure 7-4. Data flow for a Get_Status Request

As Figure 7-4 illustrates, the 8051 responds to the SUDAV interrupt by decoding the eight bytes the EZ-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 EZ-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 EZ-USB core to ACK the status stage of the transfer.

The following tables show the eight SETUP bytes for Get_Status requests.

Table 7-3. Get Status-Device (Remote Wakekup and Self-Powered bits)

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**0x80**	IN, Device	*Load two bytes into IN0BUF* *Byte 0 : bit 0 = Self Powered bit* *: bit 1 = Remote Wakeup* *Byte 1 : zero*
1	bRequest	**0x00**	"Get Status"	
2	wValueL	0x00		
3	wValueH	0x00		
4	wIndexL	0x00		
5	wIndexH	0x00		
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 11, "Reset and 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 7-4. Get Status-Endpoint (Stall bits)

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**0x82**	IN, Endpoint	*Load two bytes into IN0BUF* *Byte 0 : bit 0 = Stall bit for EP(n)* *Byte 1 : zero* EP(n): 0x00-0x07: OUT0-OUT7 0x80-0x87: IN0-IN7
1	bRequest	**0x00**	"Get Status"	
2	wValueL	0x00		
3	wValueH	0x00		
4	wIndexL	**EP**	Endpoint Number	
5	wIndexH	0x00		
6	wLengthL	**0x02**	Two bytes requested	
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 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 would be in 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 7-5. Get Status-Interface

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**0x81**	IN, Interface	*Load two bytes into IN0BUF* *Byte 0 : zero* *Byte 1 : zero*
1	bRequest	**0x00**	"Get Status"	
2	wValueL	0x00		
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.

7.3.2 Set Feature

Set Feature is used to enable remote wakeup or stall an endpoint. No data stage is required.

Table 7-6. Set Feature-Device (Set Remote Wakeup Bit)

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**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 7-3). The host uses this bit to enable or disable remote wakeup by the device.

Table 7-7. Set Feature-Endpoint (Stall)

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**0x02**	OUT, Endpoint	*Set the STALL bit for the indicated endpoint:* EP: 0x00-0x07: OUT0-OUT7 0x80-0x87: IN0-IN7
1	bRequest	**0x03**	"Set Feature"	
2	wValueL	**0x00**	Feature Selector: STALL	
3	wValueH	0x00		
4	wIndexL	**EP**		
5	wIndexH	0x00		
6	wLengthL	0x00		
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 steps:

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 EZ-USB core automatically maintains the endpoint toggle bits to insure data integrity for USB transfers. The 8051 should directly manipulate these bits only for a very limited set of circumstances:

1. Set_Feature/Stall
2. Set_Configuration
3. Set_Interface

7.3.3 Clear Feature

Clear Feature is used to disable remote wakeup or to clear a stalled endpoint.

Table 7-8. Clear Feature-Device (Clear Remote Wakeup Bit)

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**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 7-9. Clear Feature-Endpoint (Clear Stall)

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**0x02**	OUT, Endpoint	*Clear the STALL bit for the indicated endpoint:* EP(n): 0x00-0x07: OUT0-OUT7 0x80-0x87: IN0-IN7
1	bRequest	**0x01**	"Clear Feature"	
2	wValueL	**0x00**	Feature Selector: STALL	
3	wValueH	0x00		
4	wIndexL	**EP**		
5	wIndexH	0x00		
6	wLengthL	0x00		
7	wLengthH	0x00		

If the USB device supports remote wakeup (as reported in its descriptor table when the device is enumerated), 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.

7.3.4 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 EZ-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 EZ-USB core does the rest.

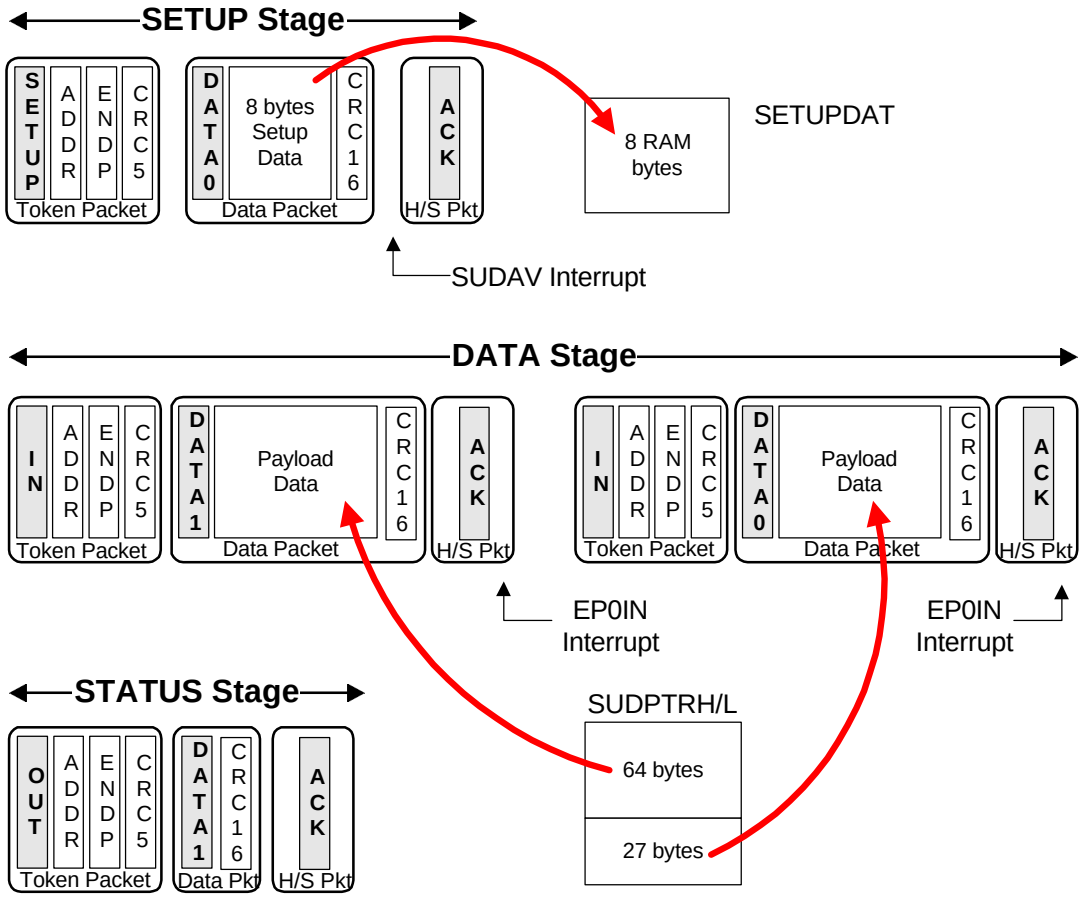


Figure 7-5. Using the Setup Data Pointer (SUDPTR) for Get_Descriptor requests

Figure 7-5 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 will fit into one packet. In the Figure 7-5 example, the descriptor data consists of 91 bytes.

The CONTROL transaction starts in the usual way, with the EZ-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 EZ-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 disablse these interrupts since the transfer requires no 8051 intervention.

Three types of descriptors are defined, Device, Configuration and String.

7.3.4.1 Get Descriptor-Device

*Table 7-10. Get Descriptor-**Device***

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**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 7-5, the 8051 loads the two-byte SUDPTR with the starting address of the Device Descriptor table. When SUDPTRL is loaded, the EZ-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 7-10).

2. Reads the requested string's descriptor to determine the actual string length.

3. Sends the smaller of (a) the requested number of bytes or (b) the actual number of bytes in the string, 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 (that you define), as long as bytes 6-7 contain the number of bytes in the transfer (for step 1, above).

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 would be a good USB training exercise, but not necessary due to the hardware support built into the EZ-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 since the Setup Data Pointer requires no byte transfers into the IN0BUF buffer.

7.3.4.2 Get Descriptor-Configuration

*Table 7-11. Get Descriptor-**Configuration***

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**0x80**	IN, Device	*Set SUDPTR H-L to start of 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**		

7.3.4.3 Get Descriptor-String

*Table 7-12. Get Descriptor-**String***

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**0x80**	IN, Device	*Set SUDPTR H-L to start of 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, loads the corresponding table pointer into SUDPTRH-L, and the EZ-USB core does the rest

7.3.5 Set Descriptor

*Table 7-13. Set Descriptor-**Device***

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**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 7-14. Set Descriptor-**Configuration***

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**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 7-15. Set Descriptor-**String***

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**0x00**	OUT, Device	*Read string descriptor data over OUT0BUF.*
1	bRequest	**0x07**	"Set_Descriptor"	
2	wValueL	0x00		
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 EZ-USB core keeps track of the number of bytes 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 EZ-USB core automatically responds to the status phase, which is the third and final stage of the CONTROL transfer.

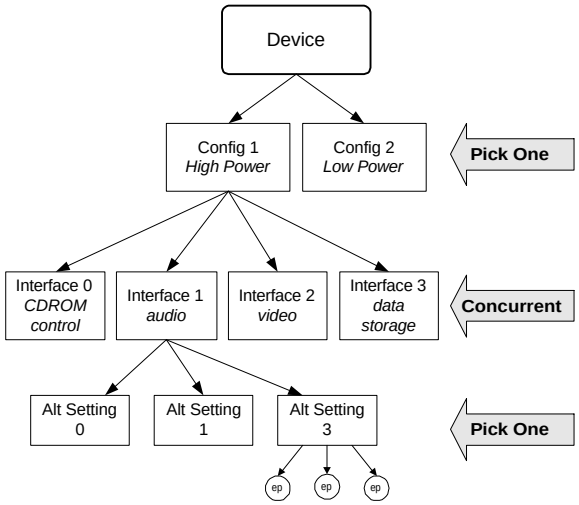
Note: the 8051 controls the flow of data in the Data Stage of a Control Transfer. After the 8051 processes each OUT packet it loads any value into the OUT endpoint's byte count register to re-arm the endpoint

Configurations, Interfaces and Alternate Settings

A USB device can have multiple configurations, only one of which is active at a time.

A configuration may have multiple interfaces, all of which are concurrently active. Multiple interfaces allow different device drivers to be associated with different portions of the USB device.

Each interface can have alternate settings. Each alternate setting has a collection of endpoints.



This structure is a software model; the EZ-USB core takes no action when these settings change. The 8051, however, must re-initialize endpoints when the host changes configurations or interface alternate settings.

As far as 8051 firmware is concerned, a "configuration" is simply a byte variable that indicates the current configuration. The host issues a Set_Configuration request to select a configuration, and a Get_Configuration request to determine the current configuration.

7.3.6 Set Configuration

Table 7-16. Set Configuration

Byte	Field	Value	Meaning	8051 Response
0	BmRequest	**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 in Table 7-16), 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.

Note:
After setting a configuration, the host issues "Set_Interface" commands to set up the various interfaces contained in the configuration.

7.3.7 Get Conguration

Table 7-17. Get Configuration

Byte	Field	Value	Meaning	8051 Response
0	BmRequest	**0x80**	IN, Device	*Send CFG over IN0BUF after reconfiguring.*
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 HSNAK bit (by writing 1 to it) to terminate the Set_Configuration CONTROL transfer.

7.3.8 Set Interface

This confusingly named USB command actually sets and reads back *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 8KHz, 22KHz and 44KHz sample rates, and 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 7-18. Set Interface (Actually, Set Alternate Setting AS for Interface IF)

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**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.

7.3.9 Get Interface

Table 7-19. Get Interface (actually, Get Alternate Setting AS for Interface IF)

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**0x80**	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.

7.3.10 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 EZ-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 EZ-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 , since the EZ-USB core automatically responds only to the host-assigned FNADDR value. The EZ-USB core makes it readable by the 8051 for debug/diagnostic purposes.

7.3.11 Sync Frame

Table 7-20. Sync Frame

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**0x82**	IN, Endpoint	*Send a frame number over IN0BUF to synchronize endpoint EP.* EP(n): 0x08-0x0F: OUT8-OUT15 0x88-0x8F: IN8-IN15
1	bRequest	**0x0C**	"Sync_Frame"	
2	wValueL	0x00		
3	wValueH	0x00		
4	wIndexL	**EP**	Endpoint number	
5	wIndexH	0x00		
6	wLengthL	**2**	LenL	
7	wLengthH	**0**	LenH	

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 1 at the same time (in the same frame).

To get in sync, the host issues the Sync_Frame request with EP=EP8-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 will initialize their sequence counters to 1. The 8051 reads the current frame count in the USBFRAMEL and USBFRAMEH registers.

Multiple isochronous endpoints can be synchronized in this manner. The 8051 keeps separate internal sequence counts for each endpoint.

About USB Frames

The USB host issues an 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 EZ-USB core detects a missing SOF packet, it uses an internal counter to generate the SOF interrupt.

7.3.12 Anchor Load

The USB endpoint zero protocol provides a mechanism for mixing vendor-specific requests with the previously described standard device requests. Bits 6:5 of the bmRequest field are set to 00 for a standard device request, and to 10 for a vendor request.

Table 7-21. Anchor Download

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**0x40**	Vendor Request, OUT	*None required*
1	bRequest	**0xA0**	"Anchor 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 7-22. Anchor Upload

Byte	Field	Value	Meaning	8051 Response
0	bmRequest	**0xC0**	Vendor Request, IN	*None required*
1	bRequest	**0xA0**	"Anchor 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 EZ-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).

Since 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 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 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 Anchor Download command.

8 EZ-USB Isochronous Transfers

8.1 Introduction

Isochronous endpoints typically handle time-critical, streamed data that is delivered or consumed in byte-sequential order. Examples might be 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 chip makes isochronous data available as a single byte that represents the head or tail of an endpoint FIFO.

The EZ-USB chip implements sixteen isochronous endpoints, IN8-IN15 and OUT8-OUT15. 1024 bytes of FIFO memory may be distributed over the 16 endpoint addresses. FIFO sizes for the isochronous endpoints are programmable.

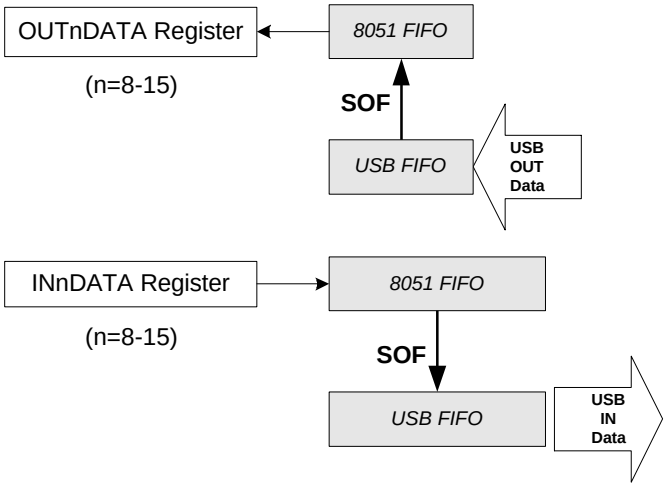


Figure 8-1. EZ-USB isochronous endpoints 8-15

The 8051 reads or writes isochronous data using sixteen FIFO data registers, one per endpoint. These FIFO registers are shown in Figure 8-1 as INnDATA (Endpoint n IN Data) and OUTnDATA (Endpoint n OUT Data).

The EZ-USB core provides a total of 2048 bytes of FIFO memory (1024 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 1024 FIFO bytes may be divided among the sixteen isochronous endpoints. The 8051 writes sixteen EZ-USB registers to allocate the FIFO buffer space to the isochronous endpoints. The 8051 also sets ‘endpoint valid’ bits to enable isochronous endpoints.

8.2 Isochronous IN Transfers

IN transfers travel from device to host. Figure 8-2 shows the EZ-USB registers and bits associated with isochronous IN transfers.

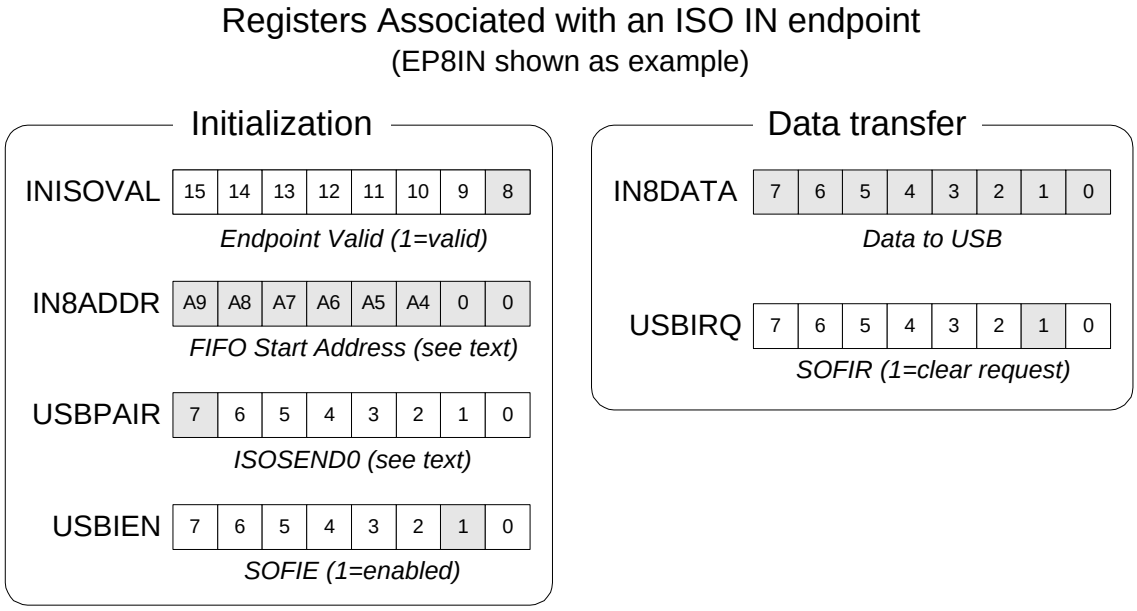


Figure 8-2. Isochronous IN endpoint registers

8.2.1 Initialization

To initialize an isochronous IN endpoint, the 8051 performs the following steps:

1. Sets the endpoint valid bit for the endpoint.
2. Sets the endpoint's FIFO size by loading a starting address (Section 8-4).
3. Sets the ISOSEND0 bit in the USBPAIR register for the desired response (see below).
4. Enables the SOF interrupt. All isochronous endpoints are serviced in response to the SOF interrupt.

The EZ-USB core uses the ISOSEND0 bit to determine what to do if:

1. The 8051 does not load any bytes to an INnDATA register during the previous frame, and
2. An IN token for that endpoint arrives from the host.

If ISOSEND0=0 (the default value), the EZ-USB core does not respond to the IN token. If ISOSEND0=1, the EZ-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, EP8IN through EP15IN.

8.2.2 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 one millisecond 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), and clears the SOFIR (Start Of Frame Interrupt Request) bit by writing a one to it. Then the 8051 loads data into the appropriate INnDATA FIFO register(s). Unlike bulk transfers, the 8051 does not load a byte count to "arm" an isochronous endpoint. The EZ-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 EZ-USB 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.

8.3 Isochronous OUT Transfers

OUT transfers travel from host to device. Figure 8-3 shows the EZ-USB registers and bits associated with isochronous OUT transfers.

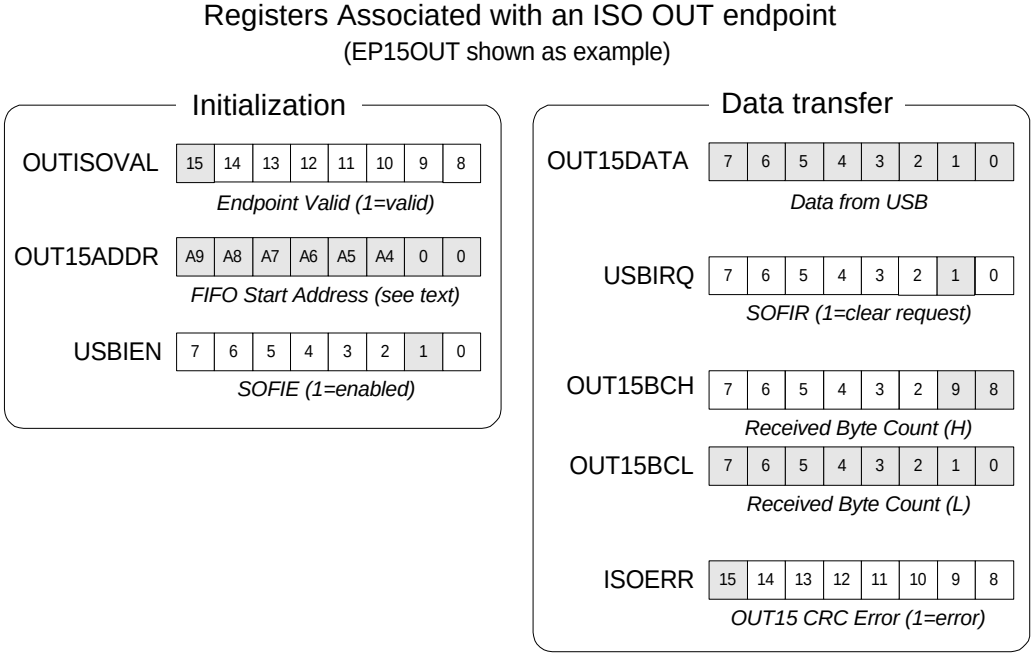


Figure 8-3. Isochronous OUT registers

8.3.1 Initialization

To initialize an isochronous OUT endpoint, the 8051:

1. Sets the endpoint valid bit for the endpoint.
2. Sets the endpoint's FIFO size by loading a starting address (Section 8.4).
3. Enables the SOF interrupt. All isochronous endpoints are serviced in response to this interrupt.

8.3.2 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 one millisecond 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 a one 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 resent, so the firmware must decide what to do with "bad-CRC" data.

8.4 Setting Isochronous FIFO Sizes

Up to sixteen EZ-USB isochronous endpoints share an EZ-USB 1024 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 1024 bytes, starting at address zero. The isochronous FIFOS can exist anywhere in the 1024 bytes, but the user must take care to insure that there is sufficient space between start addresses to accommodate the endpoint FIFO size.

Sixteen start address registers set the isochronous FIFO sizes (Table 8-1). The EZ-USB core constructs the address within the 1024 byte range from the register value as shown in Figure 8-4.

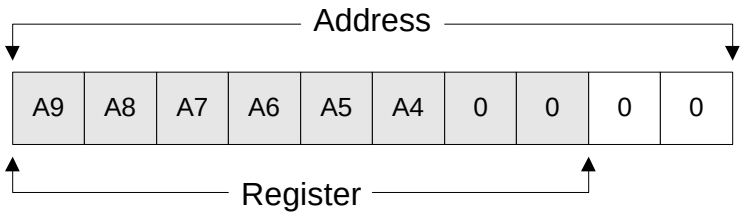


Figure 8-4. FIFO Start Address format

Table 8-1. Isochronous endpoint FIFO Starting Address Registers

Register	Function	b7	b6	b5	b4	b3	b2	b1	b0
OUT8ADDR	Endpt 8 OUT Start Addr	A9	A8	A7	A6	A5	A4	0	0
OUT9ADDR	Endpt 9 OUT Start Addr	A9	A8	A7	A6	A5	A4	0	0
OUT10ADDR	Endpt 10 OUT Start Addr	A9	A8	A7	A6	A5	A4	0	0
OUT11ADDR	Endpt 11 OUT Start Addr	A9	A8	A7	A6	A5	A4	0	0
OUT12ADDR	Endpt 12 OUT Start Addr	A9	A8	A7	A6	A5	A4	0	0
OUT13ADDR	Endpt 13 OUT Start Addr	A9	A8	A7	A6	A5	A4	0	0
OUT14ADDR	Endpt 14 OUT Start Addr	A9	A8	A7	A6	A5	A4	0	0
OUT15ADDR	Endpt 15 OUT Start Addr	A9	A8	A7	A6	A5	A4	0	0
IN8ADDR	Endpt 8 IN Start Addr	A9	A8	A7	A6	A5	A4	0	0
IN9ADDR	Endpt 9 IN Start Addr	A9	A8	A7	A6	A5	A4	0	0
IN10ADDR	Endpt 10 IN Start Addr	A9	A8	A7	A6	A5	A4	0	0
IN11ADDR	Endpt 11 IN Start Addr	A9	A8	A7	A6	A5	A4	0	0
IN12ADDR	Endpt 12 IN Start Addr	A9	A8	A7	A6	A5	A4	0	0
IN13ADDR	Endpt 13 IN Start Addr	A9	A8	A7	A6	A5	A4	0	0
IN14ADDR	Endpt 14 IN Start Addr	A9	A8	A7	A6	A5	A4	0	0
IN15ADDR	Endpt 15 IN Start Addr	A9	A8	A7	A6	A5	A4	0	0

The size of an isochronous endpoint FIFO is determined by subtracting consecutive addresses in Table 8-1, and multiplying by four. Values written to these registers should have the 2 LSB's set to zero. The last endpoint, EP15IN, has a size of 1024 minus IN15ADDR times four. Because the 10-bit effective address has the four LSB's set to zero (Figure 8-4), the FIFO sizes are allocated in increments of 16 bytes. For example, if OUT8ADDR=0x00 and OUT9ADDR=0x04, then EP8OUT has a FIFO size of the difference times 4, or 16 bytes.

An 8051 assembler or C compiler may be used to translate FIFO sizes into starting addresses. The assembler example in Figure 8-5 shows a block of equates for the16 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 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 8-5. Assembler translates FIFO sizes to addresses

The assembler computes starting addresses in Figure 8-5 by adding the previous endpoint's address to the desired size shifted right twice. This aligns A9 with bit 7 as shown in Table 8-1. The "LOW" operator takes the low byte of the resulting 16-bit expression.

The user of this code must insure 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.

8.5 Isochronous Transfer Speed

The amount of data USB can transfer during a one millisecond frame is slightly more than 1000 bytes per frame (1500 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 8-6 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 data bus and strobed by the RD signal, into an internal isochronous IN FIFO.

```
        mov     dptr,#8000H     ; pointer to any outside address
        inc     dps             ; switch to second data pointer
        mov     dptr,#IN8DATA   ; pointer to an IN endpoint FIFO (IN8 as example)
        inc     dps             ; back to first data pointer
        mov     r7,#nBytes      ; r7 is loop counter--transfer this many bytes
;
loop:   movx    a,@dptr         ; (2) read byte from external bus to acc
        inc     dps             ; (1) switch to second data pointer
        movx    @dptr,a         ; (2) write to ISO FIFO
        inc     dps             ; (1) switch back to first data pointer
        djnz    r7,loop         ; (3) loop 'nBytes' times
```

Figure 8-6. 8051 code to transfer data to an isochronous FIFO (IN8DATA)

The numbers in parentheses indicate 8051 cycles. One cycle is four clocks, and the EZ-USB 8051 is clocked at 24 MHz (42 nanoseconds). Thus an 8051 cycle takes 4*42=168 nanoseconds, and the loop takes 9 cycles or 1.5 microseconds. This loop can transfer about 660 bytes into an IN FIFO every millisecond (1ms/1.5us).

If more speed is required, the loop can be "unrolled" by in-line coding the first four instructions in the loop. Then a byte is transferred in 6 cycles (24 clocks) which equates to 1 microsecond per byte. Using this method, the 8051 could transfer 1000 bytes into an IN FIFO every millisecond. In practice a better solution is to in-line code only a portion of the loop code, which decreases full in-line performance only slightly and uses far fewer bytes of program code.

8.6 Fast Transfers

EZ-USB has a special “fast transfer” mode for applications that use external FIFOS connected to the EZ-USB data bus. These applications typically require very high transfer speeds in and out of EZ-USB endpoint buffers.

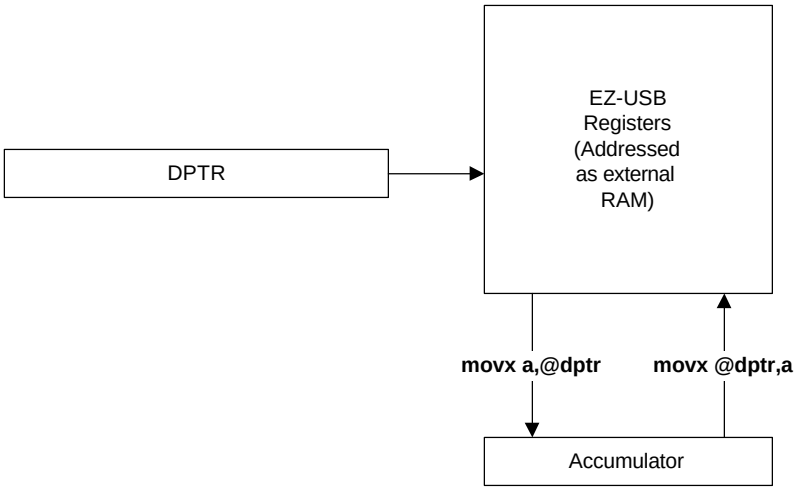


Figure 8-7. 8051 movx Instructions

The 8051 transfers data to and from EZ-USB registers and RAM using the ‘movx’ (move external) instruction (Figure 8-7). The 8051 loads one of its two 16-bit data pointers (DPTR) with an address in RAM, and then executes a ‘movx’ instruction to transfer data between the accumulator and the byte addressed by DPTR. The “@” symbol indicates that the address is supplied indirectly, by the DPTR.

The EZ-USB core monitors ‘movx’ transfers between the accumulator and *any of the sixteen isochronous FIFO registers*. If an enable bit is set (FISO=1 in the FASTXFR register), any read or write to an isochronous FIFO register causes the EZ-USB core to connect the data to the EZ-USB data bus D[7..0], and generate external read/write strobes. One ‘movx’ instruction thus transfers a byte of data in or out of an endpoint FIFO and generates timing strobes for an outside FIFO or memory. The 2-cycle ‘movx’ instruction takes 2 cycles or 333 nanoseconds. Figure 8-8 and Figure 8-9 show the data flow for fast writes and reads over the EZ-USB data bus.

Fast Bulk Transfers
The EZ-USB core provides a special auto-incrementing data pointer that makes the fast transfer mechanism available for bulk transfers. The 8051 loads a 16-bit RAM address into the AUTOPTRH/L registers, and then accesses RAM data as a FIFO using the AUTODATA register. Section 6.15 in Chapter 6, , “EZ-USB Bulk Transfers” describes this special pointer and register.

8.6.1 Fast Writes

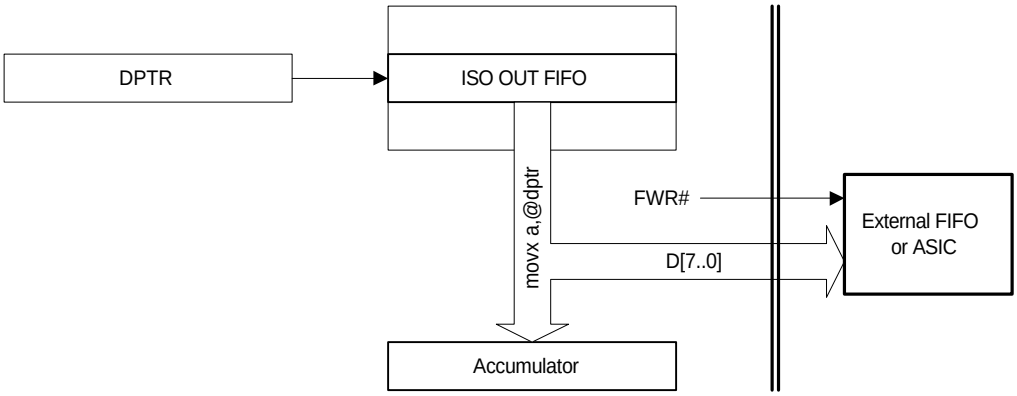


Figure 8-8. Fast Transfer, EZ-USB to outside memory

Fast writes are illustrated in Figure 8-8. When the fast mode is enabled, the DPTR points to an isochronous OUT FIFO register, and the 8051 executes the 'movx a,@dptr' instruction, the EZ-USB core broadcasts the data from the isochronous FIFO to the outside world via the data bus D[7..0], and generates a Write Strobe FWR# (Fast Write). A choice of eight waveforms is available for the write strobe, as shown in the next section.

8.6.2 Fast Reads

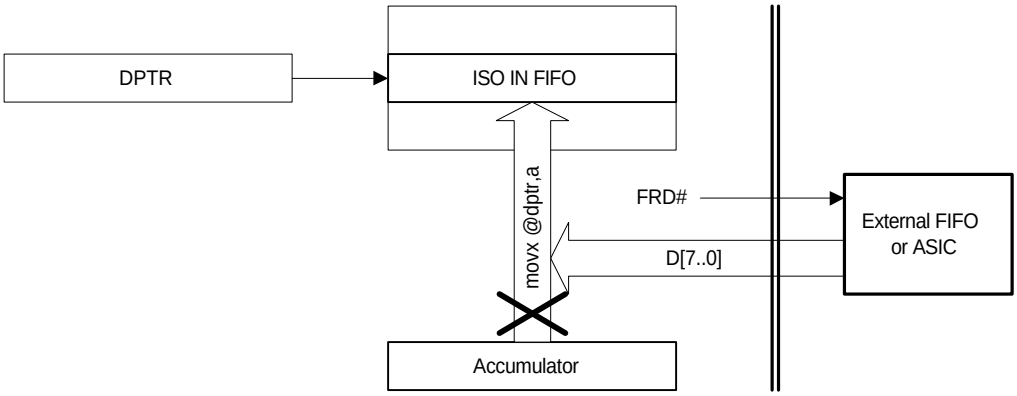


Figure 8-9. Fast Transfer, outside memory to EZ-USB

Fast reads are illustrated in Figure 8-9. When the fast mode is enabled, the DPTR points to an isochronous OUT FIFO register, and the 8051 executes the 'movx @dptr,a' instruction, the EZ-USB core breaks the data path from the accumulator to the IN FIFO register, and instead writes the IN FIFO using outside data from D[7..0]. The EZ-USB

core synchronizes this transfer by generating a FIFO Read Strobe FRD# (Fast Read). A choice of eight waveforms is available for the read strobe, as shown in the next section.

8.7 Fast Transfer Timing

The 8051 sets bits in the FASTXFR register to select the fast ISO and/or fast BULK mode and to adjust the timing and polarity of the read and write strobes FRD# and FWR#.

FASTXFR **Fast Transfer Control** **7FE2**

b7	b6	b5	b4	b3	b2	b1	b0
FISO	**FBLK**	**RPOL**	**RMOD1**	**RMOD0**	**WPOL**	**WMOD1**	**WMOD0**

Figure 8-10. The FASTXFR register controls FRD# and FWR# strobes

The 8051 sets FISO=1 to select the fast ISO mode and FBLK=1 to select the fast Bulk mode. The 8051 selects read and write strobe pulse polarities with the RPOL and WPOL bits, where 0=active low, 1=active high. Read and write strobe timings are set by RMOD1-RMOD0 for read strobes and WMOD1-WMOD0 for write strobes, as shown in Figure 8-11 (write) and Figure 8-12 (read).

Note:
When using the fast transfer feature, be sure to enable the FRD# and FWR# strobe signals in the PORTACFG register.

8.7.1 Fast Write Waveforms

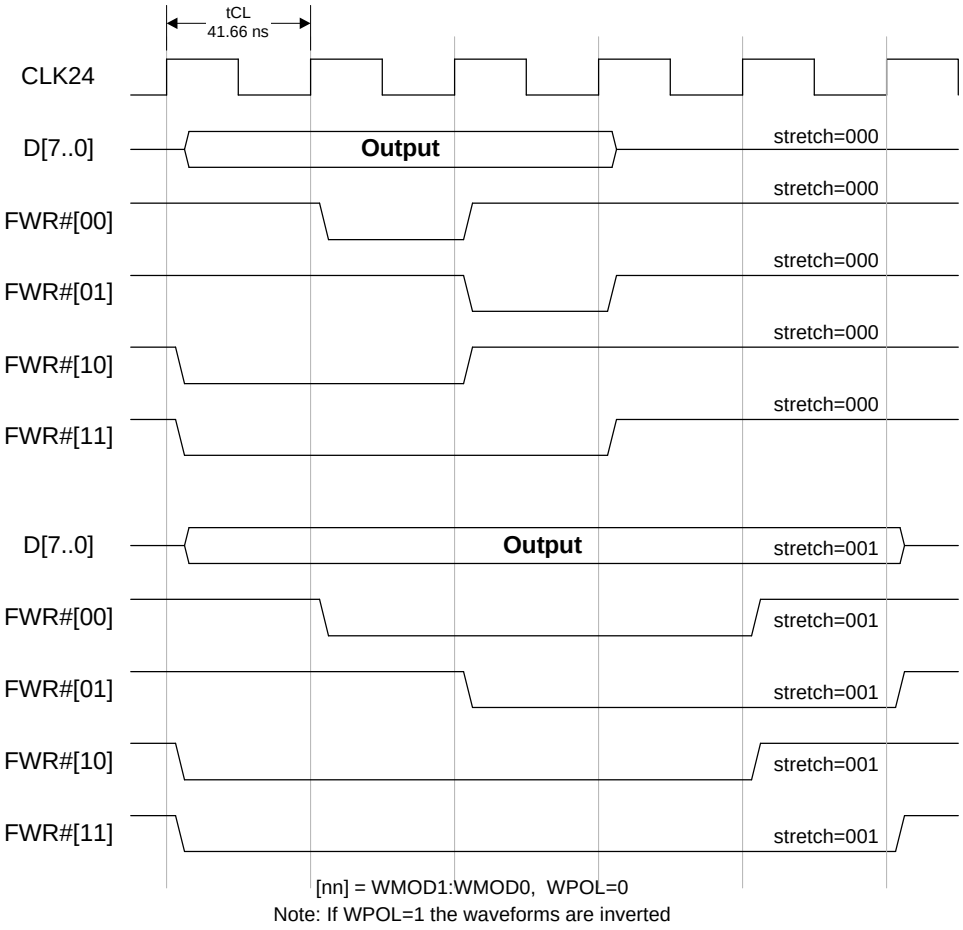


Figure 8-11. Fast Write timing

The timing choices for fast write pulses (FWR#) are shown in Figure 8-11. The 8051 can extend the output data and widths of these pulses by setting cycle stretch values greater than zero in the 8051 Clock Control Register CKCON (at SFR location 0x8E). The top five waveforms show the fastest write timings, with a stretch value of 000, which performs the write in eight 8051 clocks. The bottom five waveforms show the same waveforms with a stretch value of 001.

8.7.2 Fast Read Waveforms

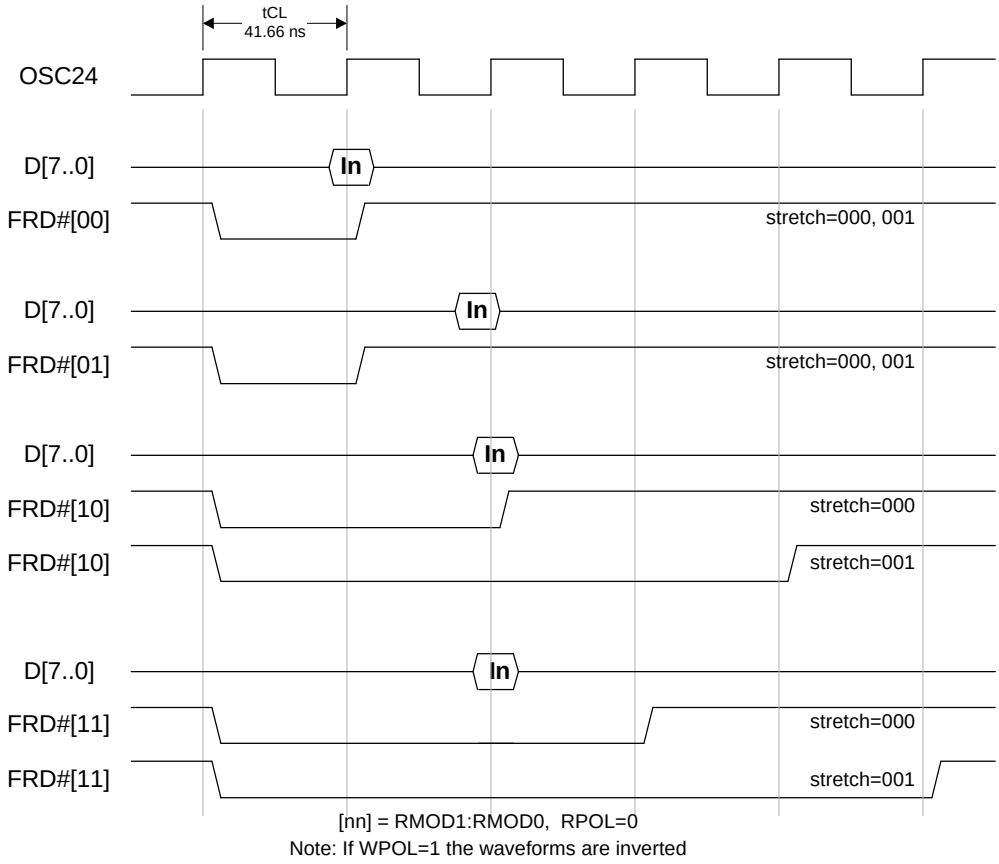


Figure 8-12. Fast Read timing

The timing choices for fast read pulses (FRD#) are shown in Figure 8-12. Read Strobe waveforms for stretch values of 000 and 001 are indicated. Although two of the the read strobe widths can be extended using stretch values greater than 000, the times that the input data is sampled by the EZ-USB core remains the same as shown.

FRD# strobes [00] and [01], along with the OSC24 clock signal are typically used to connect to an external synchronous FIFO. The one-clock-wide read strobe insures that the FIFO address advances only once per clock. The second strobe [01] is for FIFOS that put data on the bus one clock after the read strobe. Stretch values above 000 serve only to extend the 8051 cycle times, without affecting the width of the FRD# strobe.

FRD# strobes [10] and [11] are typically connected to an external *asynchronous* FIFO, where no clock is required. Strobe [10] samples the data at the same time as strobe [11], but provides a wider pulse width (for stretch=000), which is required by some audio CODECS. Timing values for these strobe signals are given in Chapter 13, “EZ-USB AC/DC Parameters”.

8.8 Fast Transfer Speed

The 8051 code example in Figure 8-13 shows a transfer loop for moving external FIFO data into the endpoint 8-IN FIFO . This code moves data from an external FIFO attached to the EZ-USB data bus and strobed by the FRD# signal, into the FIFO register IN8DATA.

```
(init)  mov     dptr,#FASTXFR  ; set up the fast ISO transfer mode
        mov     a,#10000000b   ; FISO=1, RPOL=0, RM1-0 = 00
        movx    @dptr,a        ; load the FASTXFR register
        mov     dptr,#IN8DATA  ; pointer to IN endpoint FIFO
        mov     r7,#80         ; r7 is loop counter, 8 bytes per loop
;
loop:   movx    @dptr,a        ; (2) write IN FIFO using byte from external bus
        movx    @dptr,a        ; (2) again
        movx    @dptr,a        ; (2) again
        movx    @dptr,a        ; (2) again
        movx    @dptr,a        ; (2) again
        movx    @dptr,a        ; (2) again
        movx    @dptr,a        ; (2) again
        movx    @dptr,a        ; (2) again
        djnz    r7,loop        ; (3) do eight more, 'r7' times
```

Figure 8-13. 8051 code to transfer 640 bytes of external data to an isochronous IN FIFO

This routine uses a combination of in-line and looped code to transfer 640 bytes into the EP8IN FIFO from an external FIFO. The loop transfers eight bytes in 19 cycles, and it takes 80 times through the loop to transfer 640 bytes. Therefore the total transfer time is 80 times 19 cycles, or 1520 cycles. The 640 byte transfer thus takes 1520*166ns or 252 microseconds, or approximately *one-fourth* of the 1 ms USB frame time.

Using this routine, the time to completely fill one isochronous FIFO with 1024 bytes (assuming all 1024 isochronous FIFO bytes are assigned to one endpoint) would be 128 times 19 cycles, or 2432 cycles. The 1024 byte transfer would take 403 microseconds, *less than half* of the 1 ms USB frame time.

If still faster time is required, the routine can be modified to put more of the 'movx' instructions in-line. For example, with sixteen in-line 'movx' instructions, the transfer time for 1024 bytes would be 35 cycles times 64 loops or 2240 cycles, or 371 microseconds, an 8% speed improvement over the eight instruction loop.

8.9 Other Isochronous Registers

Two additional registers, ISOCTL and ZBCOUT provide additional isochronous endpoint features.

8.9.1 Disable ISO

ISOCTL	**Isochronous Control**	**7FA1**

b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	**PPSTAT**	**MBZ**	**MBZ**	**ISODISAB**

Figure 8-14. ISOCTL register

Bit zero of the ISOCTL register is called ISODISAB. When the 8051 sets ISODISAB=1, all sixteen of EZ-USB 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 2048 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 2K of memory inside the EZ-USB chip. This is shown in Table 8-2.

Table 8-2. Addreses for RD# and WR# vs. ISODISAB bit

ISODISAB	**RD#, WR#**
0 (default)	2000-7B40, 8000-FFFF
1	**2800**-7B40, 8000-FFFF

The ISOCTL register bits shown as “MBZ” (must be zero) 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 0x00 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 will cause unpredictable operation.

Note:
The Autopointer is not usable from 0x2000-0x27FF (the reclaimed ISO buffer RAM) when ISODISAB=1.

8.9.2 Zero Byte Count Bits

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 8-15 shows this register.

ZBCOUT **Zero Byte Count Bits** **7FA2**

b7	b6	b5	b4	b3	b2	b1	b0
EP15	**EP14**	**EP13**	**EP12**	**EP11**	**EP10**	**EP9**	**EP8**

Figure 8-15. ZBCOUT register

The EZ-USB core updates these bits every SOF.

8.10 ISO IN Response with no data

The ISOSEND0 bit (bit 7 in the USBPAIR register) is used when the EZ-USB chip receives an isochronous IN token while the IN FIFO is empty. If ISOSEND0=0 (the default value) the EZ-USB core does not respond to the IN token. If ISOSEND0=1 the EZ-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, IN-8 through IN-15.

9 EZ-USB Interrupts

9.1 Introduction

The EZ-USB enhanced 8051 responds to the interrupts shown in Table 9-1. Interrupt sources that are not present in the standard 8051 are shown as checked in the "new" column. The three interrupts used by the EZ-USB core are shown in bold type.

Table 9-1. EZ-USB 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_O	UART0 Rx & Tx	**23**	5
✓	TF2	Timer 2 Overflow	**2B**	6
✓	**Resume (PFI)**	**WAKEUP# Pin or USB core**	**33**	0
✓	RI_1 & TI_1	UART1 Rx & Tx	**3B**	7
✓	**USB (INT2)**	**USB Core**	**43**	8
✓	**I^2C (INT3)**	**USB Core**	**4B**	9
✓	IE4	INT4 Pin	**53**	10
✓	IE5	INT5# Pin	**5B**	11
✓	IE6	INT6 Pin	**63**	12

The "Natural Priority" column in Table 9-1 shows the 8051 interrupt priorities. As explained in Appendix C, the 8051 can assign each interrupt to a high or low priority group. The 8051 resolves priorities within the groups using the natural priorities.

9.2 USB Core Interrupts

The EZ-USB core provides three interrupt request types, which are described in the following sections:

- **Wakeup**. After the EZ-USB chip detects USB suspend and the 8051 has entered its idle state, the EZ-USB core responds to an external signal on its WAKEUP# pin or resumption of USB bus activity by re-starting the EZ-USB oscillator and resuming 8051 operation.

- **USB signaling**. These include sixteen bulk endpoint interrupts, three interrupts not specific to a particular endpoint (SOF, Suspend, USB Reset), and two interrupts for CONTROL transfers (SUTOK, SUDAV). These twenty-one interrupts share the USB interrupt (INT2).

- **I^2C transfers** (INT3).

9.3 Wakeup Interrupt

Chapter 10, "EZ-USB Reset and Power Management", describes USB suspend-resume signaling in detail, along with a code example that uses the Wakeup interrupt.

Briefly, the USB host puts a device into SUSPEND by stopping bus activity to the device. When the EZ-USB core detects three milliseconds of no bus activity, it activates the USB suspend interrupt request. If enabled, the 8051 takes the suspend interrupt, does power management housekeeping (shutting down power to external logic), and finishes by setting SFR bit PCON.0. This signals the EZ-USB core to enter a very low power mode by turning off the 12 MHz oscillator.

When the 8051 sets PCON.0, it enters an idle state. 8051 execution is resumed by activation of any enabled interrupt. The EZ-USB chip uses a dedicated interrupt for USB Resume. 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 EZ-USB core re-starts the 12 MHz oscillator, allowing the 8051 to recognize the interrupt and continue executing instructions.

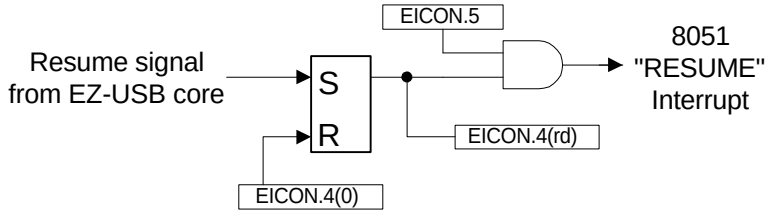


Figure 9-1. EZ-USB wakeup interrupt

Figure 9-1 shows the 8051 SFR bits associated with the RESUME interrupt. The EZ-USB core asserts the resume signal when the EZ-USB core senses a USB Global Resume, or when the EZ-USB WAKEUP# pin is pulled low. The 8051 enables the RESUME interrupt by setting EICON.5.

```
setb  EICON.5     ; enable Resume interrupt
```

The 8051 reads the RESUME interrupt request bit in EICON.4, and clears the interrupt request by writing a zero to EICON.4.

```
resume_isr:       clr   EICON.4     ; clear the 8051 W/U
                                    ; interrupt request
                  reti
```

9.4 USB Signaling Interrupts

Figure 9-2 shows the twenty-one 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 of the USB interrupt requests.

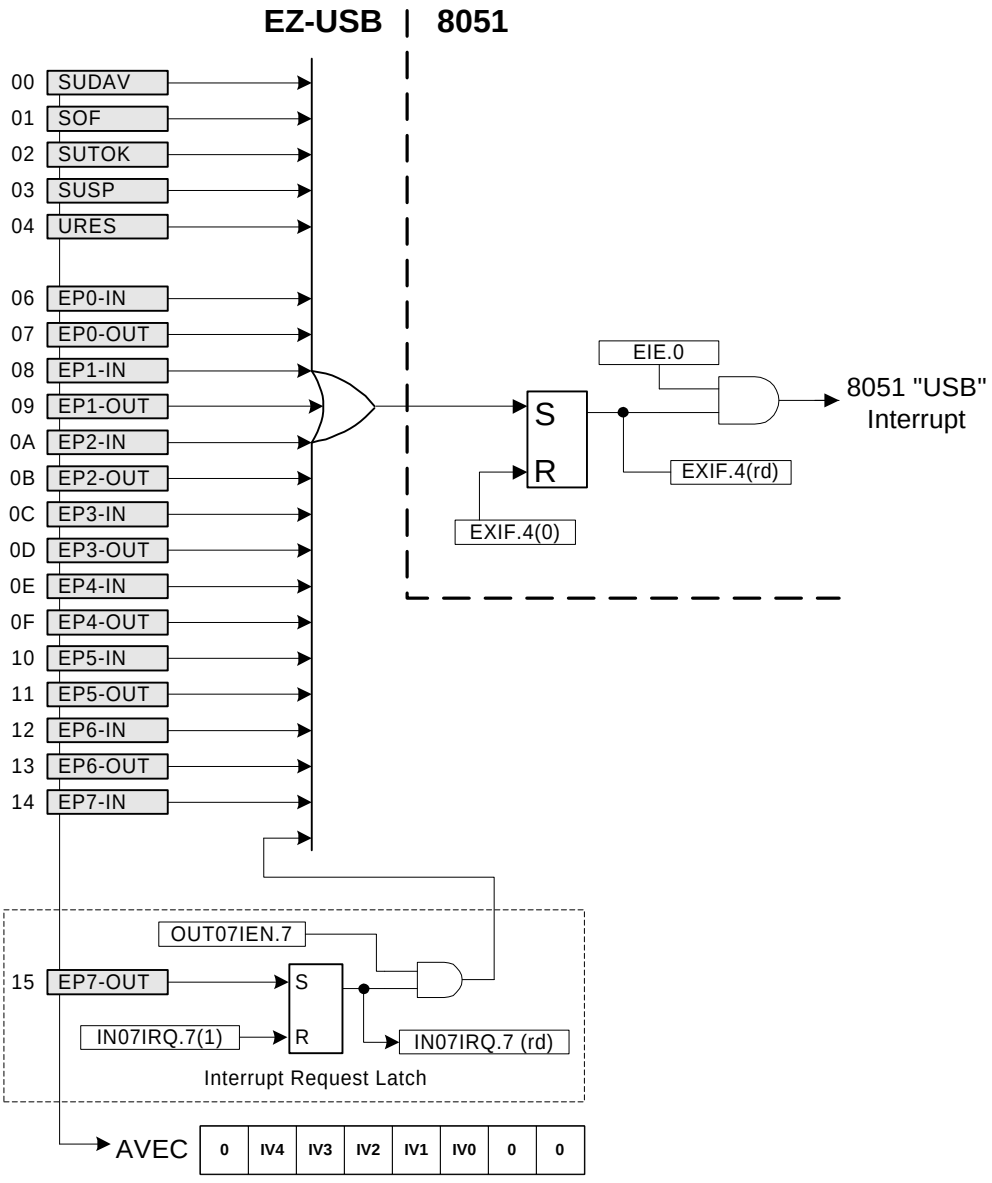


Figure 9-2. USB interrupts

Referring to the logic inside the dotted lines, each USB interrupt source has an interrupt request latch. The EZ-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 AND'ed with an IEN (Interrupt Enable) bit and then OR'ed with all the other USB interrupt request sources.

The EZ-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 automatically to vector to different addresses for each USB interrupt source. This mechanism is explained in detail in the "USB Autovectors" section of this chapter.

Due to the OR gate in Figure 9-2, 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.0 bit. Note that this is the opposite of clearing any of the individual USB interrupt sources, which the 8051 does by writing a one to the IRQ bit.

When a USB resource requires service (for example, an 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, OR'ed 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 EZ-USB core logic to generate another pulse for the highest-priority pending interrupt. If more than one is pending, they are serviced in the priority order shown in Figure 9-2, starting with SUDAV (priority 00) as the highest priority, and ending with EP7-OUT (priority 15) as the lowest.

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, and if the INT2 interrupt request latch has not been previously cleared the pending interrupt will be lost.

Figure 9-3 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 9-3. 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
-	-	-	**URESIR**	**SUSPIR**	**SUTOKIR**	**SOFIR**	**SUDAVIR**

IN07IEN **Endpoints 0-7 IN Interrupt Enables** **7FAC**

b7	b6	b5	b4	b3	b2	b1	b0
IN7IEN	**IN6EN**	**IN5IEN**	**IN4IEN**	**IN3IEN**	**IN2IEN**	**IN1IEN**	**IN0IEN**

OUT07IEN **Endpoints 0-7 OUT Interrupt Enable** **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
-	-	-	**URESIE**	**SUSPIE**	**SUTOKIE**	**SOFIE**	**SUDAVIE**

Figure 9-4. EZ-USB Interrupt Registers

Figure 9-4 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 9-2. The "IN07IEN, " and "OUT07" registers control the remaining sixteen USB interrupts, which correspond to the sixteen bulk endpoints IN0-IN7 and OUT0-OUT7.

The twenty-one USB interrupts are now described in detail.

9.5 SUTOK , SUDAV Interrupts

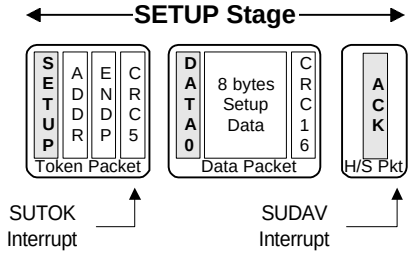


Figure 9-5. SUTOK and SUDAV interrupts

SUTOK and SUDAV are supplied to the 8051 by EZ-USB CONTROL endpoint zero. The first portion of a USB CONTROL transfer is the SETUP stage shown in Figure 9-5. (A full CONTROL transfer is shown in Figure 7-1 on page 96.) When the EZ-USB core decodes a SETUP packet, it asserts the SUTOK (SETUP Token) interrupt request. After the EZ-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 7).

The SUTOK interrupt is provided to give advance warning that the eight register bytes at SETUPDAT are about to be over-written. It is useful for debug and diagnostic purposes.

9.6 SOF Interrupt

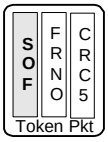


Figure 9-6. A Start Of Frame (SOF) packet

USB Start Of Frame interrupt requests occur every millisecond. When the EZ-USB core receives an SOF packet, it copies the eleven-bit frame number (FRNO in Figure 9-6) 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.

9.7 SUSPEND Interrupt

IF the EZ-USB detects three milliseconds of no bus activity, it activates the SUSP (Suspend) interrupt request. A full description of Suspend-Resume signaling appears in Chapter 11, EZ-USB Power Management.

9.8 USB RESET Interrupt

The USB signals a bus reset by driving both D+ and D- low for at least 10 milliseconds. When the EZ-USB core detects the onset of USB bus reset, it activates the URES interrupt request.

9.9 Bulk Endpoint Interrupts

The remaining sixteen USB interrupt requests are indexed to the sixteen EZ-USB bulk endpoints. The EZ-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 EZ-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 EZ-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.

9.10 USB Autovectors

The USB interrupt is shared by 21 interrupt sources. To save the code and processing time required to sort out which USB interrupt occurred, the EZ-USB core provides a second level of interrupt 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 an interrupt service routine. The 8051 jump instruction is encoded as follows:

Table 9-2. 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 EZ-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 9-3. 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	NOP
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

9.11 Autovector Coding

A detailed example of a program that uses Autovectoring is presented in Chapter 6 (Section 6-14, “Interrupt Bulk Transfer Example”). The coding steps are summarized here. To employ EZ-USB Autovectoring:

1. Insert a jump instruction at 0x43 to a table of jump instructions to the various USB interrupt service routines.

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.

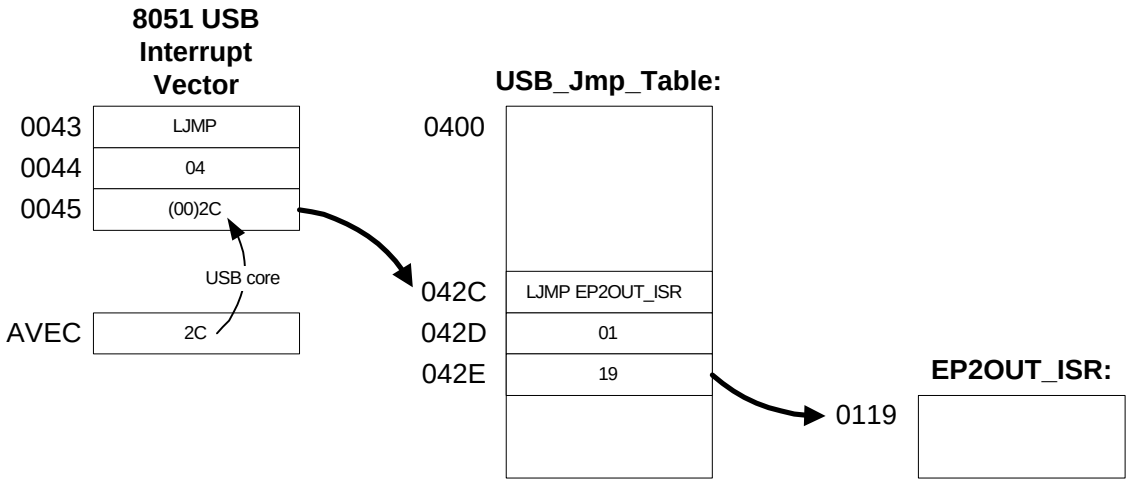


Figure 9-7. The Autovector mechanism in action

Figure 9-7 illustrates Autovectoring to an ISR that services endpoint 2-OUT. When endpoint 2-OUT requires service, the EZ-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 EZ-USB core substituting **2C** as the Autovector byte for Endpoint 2 OUT (Table 9-3, page 144). 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.

9.12 I²C Interrupt

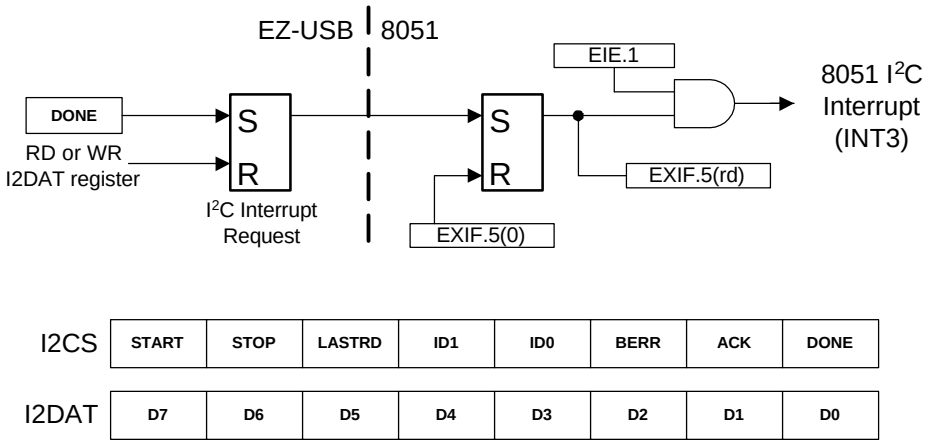


Figure 9-8. I²C interrupt enable bits and registers

Chapter 4, "EZ-USB Input/Output" describes the 8051 interface to the EZ-USB I²C controller. The 8051 uses two registers, I2CS (I²C Control and Status) and I2DAT (I²C Data) to transfer data over the I²C bus. The EZ-USB core signals completion of a byte transfer by setting the DONE bit (I2CS.0) high, which also sets an I²C interrupt request latch (Figure 9-8). This interrupt request is routed to the 8051 INT3 interrupt.

The 8051 enables the I²C 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 I²C interrupt request.

10 EZ-USB Resets

10.1 Introduction

The EZ-USB chip has three resets:

- A Power-On Reset (POR), which turns on the EZ-USB chip in a known state.
- An 8051 reset, controlled by the EZ-USB core.
- A USB bus reset, sent by the host to reset a device.

This chapter describes the effects of these three resets.

10.2 EZ-USB Power-On-Reset (POR)

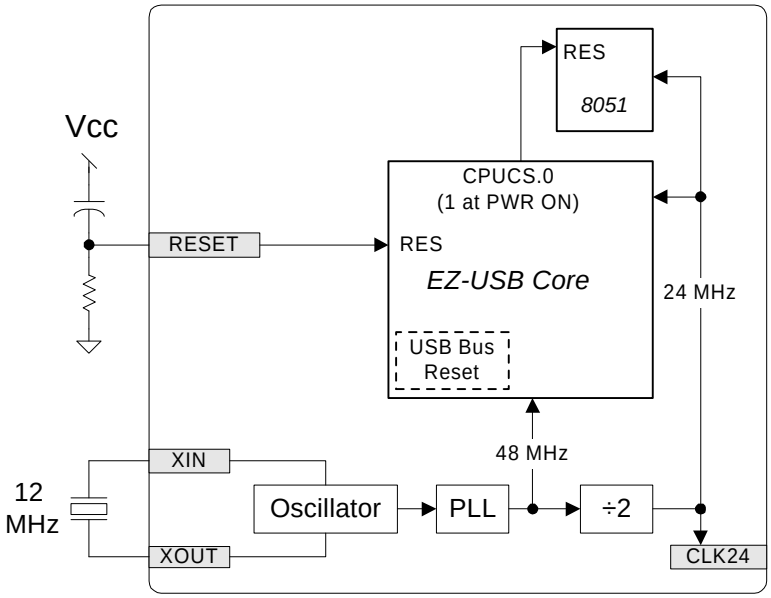


Figure 10-1. EZ-USB resets

When power is first applied to the EZ-USB chip, the external R-C circuit holds the EZ-USB core in reset until the on-chip PLL stabilizes. The CLK24 pin is active as soon as power is applied. The 8051 may clear an EZ-USB control bit, CLK24OE, to inhibit the CLK24 output pin for EMI-sensitive applications that do not need this signal. External logic can force a chip reset by pulling the RESET pin HI. The RESET pin is normally connected to VCC through a 1uF capacitor and to GND through a 10K resistor (Figure 10-1). The oscillator and PLL are unaffected by the state of the RESET pin.

The CLK24 signal is active while RESET = HI. When RESET returns LO, the activity on the CLK24 pin depends on whether or not the EZ-USB chip is in suspend state. If in

suspend CLK24 stops. Resumption of USB bus activity or asserting the WAKEUP# pin LO re-starts the CLK24 signal.

Power-on default values for all EZ-USB register bits are shown in Chapter 12, "EZ-USB Registers". Table 10-1 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 10-1. EZ-USB states after power-on reset

Item	Register	Default Value	Comment
1	Endpoint Data	xxxxxxxx	
2	Byte Counts	xxxxxxxx	
3	CPUCS	rrrr0011	rrrr=rev number, b1=CLK24OE, b0=8051RES
4	PORT Configs	00000000	IO, not alternate functions
5	PORT Registers	xxxxxxxx	
6	PORT OE's	00000000	Inputs
7	Interrupt Enables	00000000	Disabled
8	Interrupt Req's	00000000	Cleared
9	Bulk IN C/S	00000000	Bulk IN endpoints not busy (unarmed)
10	Bulk OUT C/S*	00000000	Bulk OUT endpoints not busy (unarmed)
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	EP8OUT, EP9OUT, EP10OUT valid
18	USBPAIR	0x000000	ISOsend0 (b7) = 0, no pairing
19	USBBAV	00000000	Break condition cleared, no Autovector
20	Configuration	0	Internal EZ-USB core value
21	Alternate Setting	0	Internal EZ-USB core value
*NOTE: When the 8051 is released from reset, the EZ-USB automatically arms the Bulk OUT endpoints by setting their CS registers to 00000010b.			

From Table 10-1, at power-on:

- Endpoint data buffers and byte counts are un-initialized (1,2).
- The 8051 is held in reset, and the CLK24 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 EZ-USB core will NAK IN or OUT tokens while the 8051 is reset. OUT endpoints are enabled when the 8051 is released from reset.
- Endpoint toggle bits are cleared (11).
- The RENUM bit is cleared. This means that the EZ-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 Anchor 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 EZ-USB core does not generate any response (18).
- The breakpoint condition is cleared, and autotovectoring is turned off (19)
- Configuration Zero, Alternate Setting Zero is in effect (20-21).

10.3 Releasing the 8051 Reset

The EZ-USB register bit CPUCS.0 resets the 8051. This bit is HI 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, as part of an EEPROM load.
- Automatically, when external ROM is used (EA=1)

10.3.1 RAM Download

Once enumerated, the host can download code into the EZ-USB RAM using the "Anchor Load" vendor request (Chapter 7, "EZ-USB Endpoint Zero"). The last packet loaded writes 0 to the CPUCS register, which clears the 8051 RESET bit.

NOTE: The other bit in the CPUCS register, CLK24OE, is writeable only by the 8051, so the host writing a zero byte to this register does not turn off the CLK24 signal.

10.3.2 EEPROM Load

Chapter 5 describes the EEPROM boot loads in detail. Briefly, at power-on the EZ-USB core checks for the presence of an EEPROM on its I^2C bus. If found, it reads the first EEPROM byte. If it reads 0xB2 as the first byte, the EZ-USB core downloads 8051 code from the EEPROM into internal RAM. The last byte of a "B2 load" writes 0x00 to the CPUCS register (at 0x7F92), which releases the 8051 from reset.

10.3.3 External ROM

EZ-USB systems can use external program memory containing 8051 code and USB device descriptors which include the VID/DID/PID bytes. Since these systems do not require an I^2C EEPROM to supply the VID/DID/PID, the EZ-USB core automatically releases 8051 reset when:

1. EA=1 (External code memory), *and*

2. No “B0/B2” EEPROM is detected on the I^2C bus

The EZ-USB core also sets the ReNUM bit to “1”, giving USB control to the 8051.

10.4 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 RAM. For such applications it is important to know the state of the EZ-USB 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 Power-On Reset described in the previous section. This discussion applies only to the condition where the EZ-USB 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 IO 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:

1. USB interrupts are disabled, but pending interrupt requests remain pending.
2. **During** the 8051 Reset, all bulk endpoints are unarmed, causing the EZ-USB core to NAK any IN or OUT tokens.
3. **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.
4. The breakpoint condition is cleared.

The ReNum bit is not affected by an 8051 reset.

When the 8051 comes out of reset, the pending interrupts are kept pending, but disabled (1). This gives the firmware writer the choice of acting on pre-8051-reset USB events, or ignoring them by clearing the pending interrupt(s).

During the 8051 reset time, the EZ-USB core holds off any USB traffic by NAK’ing IN and OUT tokens (2). The EZ-USB core automatically arms the OUT endpoints when the 8051 exits the reset state (3).

USBBAV.3, the breakpoint BREAK bit, is cleared (4). The other bits in the USBBAV register are unaffected.

10.5 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 milliseconds. The EZ-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 resources as shown in Table 10-2.

Table 10-2. EZ-USB 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 OE's	uuuuuuuu	
7	Interrupt Enables	uuuuuuuu	
8	Interrupt Req's	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	0	
20	Alternate Setting	0	

A USB bus reset leaves most EZ-USB resources unchanged. From Table 10-2, after a USB bus reset:

- The EZ-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 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.

10.6 EZ-USB Disconnect

Table 10-3. Effects of an EZ-USB 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 OE's	uuuuuuuu	
7	Interrupt Enables	uuuuuuuu	
8	Interrupt Req's	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	0	
20	Alternate Setting	0	

Although not strictly a 'reset', when the EZ-USB simulates a disconnect-reconnect in order to ReNumerate™, there are effects on the EZ-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).

10.7 RESET Summary

Table 10-4. Effects of various EZ-USB resets. "U" means "unaffected"

Resource	RESET pin	USB Bus Reset	Disconnect	8051 Reset
8051 reset	reset	U	U	N/A
EP0-7 IN EP's	unarm	unarm	unarm	unarm
EP0-7 OUT EP's	unarm	U	arm	unarm/arm
Breakpoint	reset	U	U	reset
Stall bits	reset	U	reset	U
Interrupt Enables	reset	U	U	reset
Interrupt Req's	reset	U	U	U
CLK24	run	U	U	U
Data Toggles	reset	reset	reset	U
Function Address	reset	reset	reset	U
Configuration	0	0	0	U
ReNum bit	0	U	U	U

Table 10-4 summarizes the effects of the four EZ-USB resets.

Note: The I^2C controller is not reset for any of the above conditions. It is reset only by the EZ-USB RESET pin.

11 EZ-USB Power Management

11.1 Introduction

The USB host can suspend a device to put it into a power-down mode. When the USB signals a SUSPEND operation, the EZ-USB 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. Once suspended, the EZ-USB chip is awakened either by resumption of USB bus activity, or by assertion of its WAKEUP# pin. This chapter describes the suspend-resume mechanism.

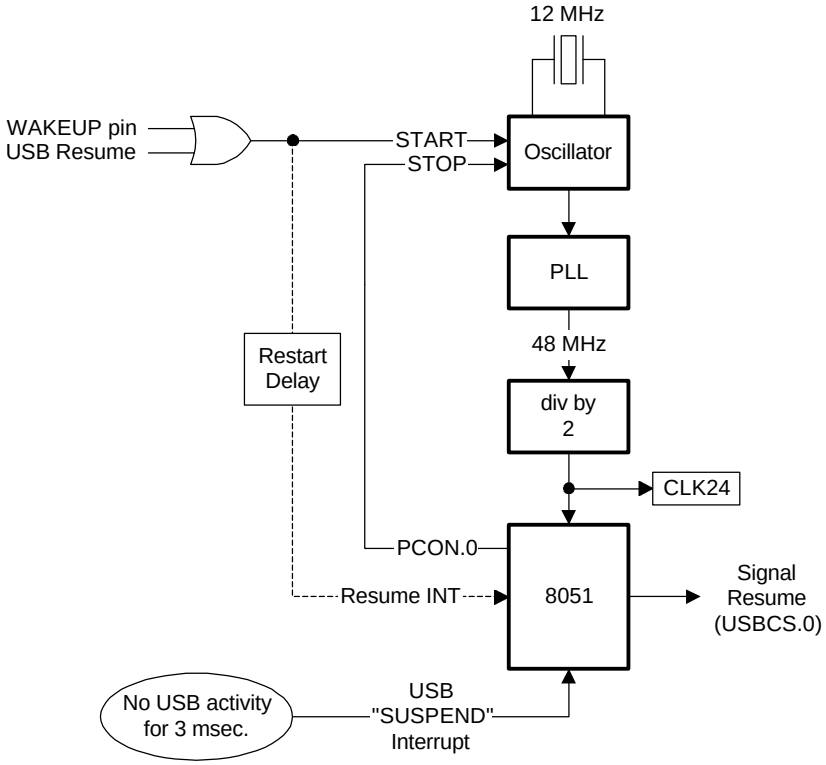


Figure 11-1. Suspend-Resume Control

Figure 11-1 illustrates the EZ-USB logic that implements USB suspend and resume. These operations are explained in the next sections.

11.2 Suspend

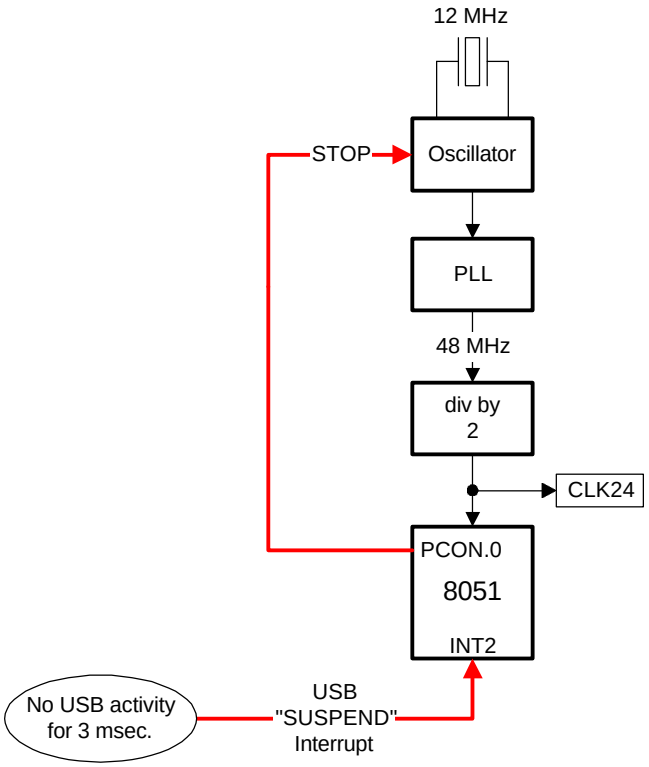


Figure 11-2. EZ-USB Suspend sequence

A USB devices recognizes SUSPEND as three milliseconds of a bus idle (“J”) state. The EZ-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.

The 8051 code responds to the SUSPEND interrupt by taking the following steps:

1. Performs any necessary housekeeping such as shutting off external power-consuming devices.

2. Sets bit 0 of the PCON SFR (Special Function Register). This has two effects:

   - The 8051 enters its “idle” mode, which is exited by any interrupt.
   - The 8051 sends an internal signal to the EZ-USB core which causes it to turn off the oscillator and PLL.

These actions put the EZ-USB chip into a low power mode, as required by the USB Specification.

11.3 Resume

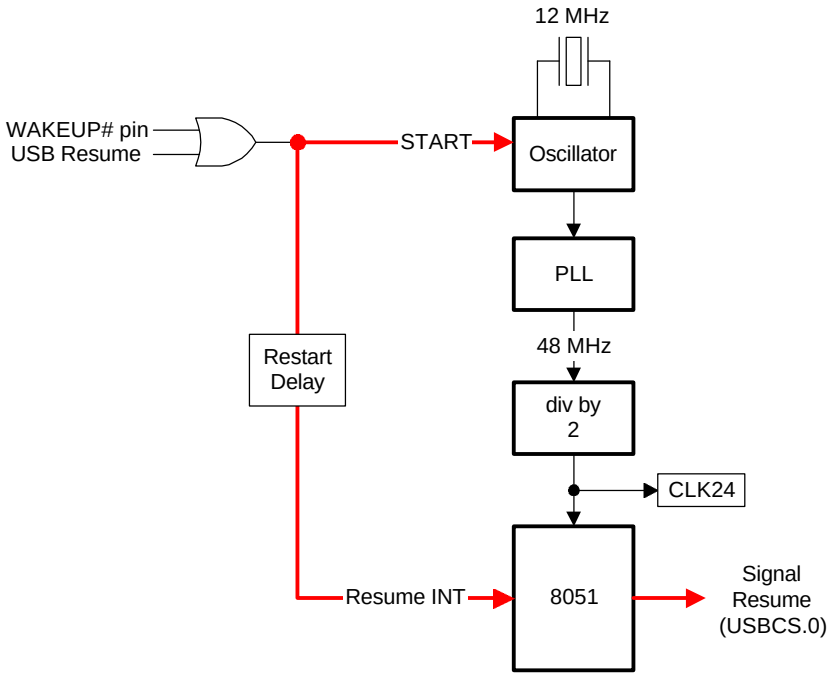


Figure 11-3. EZ-US Resume sequence

The EZ-USB oscillator re-starts when:

A. USB bus activity resumes (shown as "USB Resume" in Figure 11-3), or
B. External logic asserts the EZ-USB WAKEUP# pin low.

After an oscillator stabilization time, the EZ-USB core asserts the 8051 Resume interrupt (Figure 9-1, page 136). 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.*

The resume ISR clears the interrupt request flag, and executes an '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 logic supports only the RESUME interrupt for the USB resume operation. This is because the EZ-USB core asserts this particular interrupt after restarting the 8051 clock.

11.4 Remote Wakeup

USBCS	USB Control and Status	7FD6

b7	b6	b5	b4	b3	b2	b1	b0
WAKESRC	-	-	-	**DISCON**	**DISCOE**	**RENUM**	**SIGRSUME**

Figure 11-4. USB Control and Status register

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.

NOTE:
Holding the WAKEUP# pin low inhibits the EZ-USB chip from suspending.

When a USB device is suspended, the hub driver is tri-stated, and the bus pullup and pulldown resistors cause the bus to assume the "J", or idle state. A suspended device signals a remote wakeup by asserting the "K" state for 10-15 milliseconds. The 8051 controls this using the SIGRSUME bit in the USBCS register.

If the 8051 finds WAKESRC=1 after exiting the idle mode, it drives the "K" state for 10-15 milliseconds to signal the USB remote wakeup. It does this by setting SIGRSUME =1, waiting 10-15 milliseconds, and then setting SIGRSUME=0. When SIGRSUME=0, the EZ-USB bus buffer reverts to normal operation. 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-), and the "K" state means the opposite, (D+ < D-).

The Anchor 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 on page 64).

Remote Wakeup: The Big Picture

Additional factors besides the EZ-USB suspend-resume mechanism described in this section determine whether remote wakeup is possible. These are:

1. The device must report that it is capable of signaling a remote wakeup in the "bAttributes" field of its Configuration Descriptor. See Table 5-10 for an example of this descriptor.

2. The host must issue a "Set_Feature/Device" request with the feature selector field set to 0x01 to enable remote wakeup. See Table 7-6 on page 106 for the detailed request.

12 EZ-USB Registers

12.1 Introduction

This section describes the EZ-USB registers in the order they appear in the EZ-USB 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, 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

Examples:

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).

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 12-1. Register description format

Figure 12-1 illustrates the register description format used in this chapter.

- The top line shows the register name, functional description, and address in the EZ-USB 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.

12.2 Bulk Data Buffers

INnBUF,OUTnBUF Endpoint 0-7 IN/OUT Data Buffers 1B40-1F3F*

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 12-1 for individual endpoint buffer addresses.

Table 12-1. 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, and only at 0x7B40 in the 32K version of EZ-USB. 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.

12.3 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 12-2 for individual endpoint buffer addresses.

Table 12-2. 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 Receive FIFO byte (USB OUT data); writing a Data Register loads a Transmit FIFO byte (USB IN data).

12.4 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

OUTnBCL **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 12-3 for individual endpoint buffer addresses.

Table 12-3. Isochronous endpoint Byte Count Register addresses

Address	Isochronous Byte Counts	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 EZ-USB core uses the byte count registers to report isochronous data payload sizes for OUT data transferred from the host to the Anchor USB core. Ten bits of byte count data allow payload sizes up to 1023 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 Receive 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. At this moment the byte count for EP8 would read 50.

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 millisecond frame, and the host requests data from that endpoint during the next frame (IN token), the Anchor 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 insure that the number of bytes written to the IN FIFO does not exceed the maximum packet size as reported during enumeration.

12.5 CPU Registers

CPUCS **CPU Control and Status** **7F92**

b7	b6	b5	b4	b3	b2	b1	b0
RV3	**RV2**	**RV1**	**RV0**	**0**	**0**	**CLK24OE**	**8051RES**
R	R	R	R	R	R	R/W	R
RV3	RV2	RV1	RV0	0	0	1	1

This register enables the CLK24 output and permits the host to reset the 8051 using an Anchor download.

Bit 7-4 : **RV[3..0]** *Silicon Revision*

These register bits define the silicon revision. Consult individual Anchor Chips data sheets for values.

Bit 1: **CLK24OE** *Output enable – CLK24 pin*

When this bit is set to 1, the internal 24 MHz clock is connected to the EZ-USB CLK24 pin. When this bit is 0, the CLK24 pin drives HI. This bit can be written by the 8051 only.

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.

12.6 Port configuration

PORTACFG IO 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 IO 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 IO 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

These three registers control the three IO ports on the EZ-USB chip. They select between IO ports and various alternate functions. They are read/write by the 8051.

When PORTnCFG=0, the port pin functions as IO, using the OUT, PINS and OE control bits. Data written to an OUTn registers appears on an IO Port pin if the corresponding output enable bit (OEn) is HI.

When PORTnCFG=1, the pin assumes the alternate function shown in Table 12-4 (next page).

Table 12-4. IO Pin alternate functions

IO	Name	Alternate Function
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	UART0 synchronous (mode 0) data output
PA7	RXD1OUT	UART1 synchronous (mode 0) data output
PB0	T2	Timer/Counter 2 clock input
PB1	T2EX	Timer 2 capture/reload input
PB2	RxD1	Serial Port 1 input
PB3	TxD1	Serial Port 1 data (modes 1-3) or clock (mode 0) output
PB4	INT4	INT4 Interrupt Request
PB5	INT5#	INT5 Interrupt Request
PB6	INT6	INT4 interrupt Request
PB7	T2OUT	Timer 2 overflow indication
PC0	RxD0	Serial Port 0 Input
PC1	TxD0	Serial Port 0 data (modes 1-3) or clock (mode 0) 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

12.7 Input-Output Port Registers

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

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

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

The OUTn registers provide the data that drives the port pin when OE=1 **and** the PORTnCFG pin is 0. 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.

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

The PINSn registers contain the current value of the port pins, whether they are selected as IO ports or 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	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

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

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

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

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

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’.

12.8 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

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	x	0	0	0

Bit 3: **PPSTAT** *Ping-Pong Status*

This bit indicates the isochronous buffer currently in use by the EZ-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 sixteen isochronous endpoints.

ISODISAB=1 disables ***all sixteen*** isochronous endpoints, making the 2048 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

Bit 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.

12.9 I²C Registers

I2CS **I²C 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 **I²C 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

The 8051 uses these registers to transfer data over the EZ-USB I²C bus.

Bit 7: **START** *Signal START condition*

The 8051 sets the START bit to 1 to prepare an I²C 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 I²C controller clears the START bit during the ACK interval.

Bit 6: **STOP** *Signal STOP condition*

The 8051 sets STOP=1 to terminate an I²C bus transfer. The I²C 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 I²C bus, an I²C 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 I²C controller clears the LastRD bit at the end of the transfer (at ACK time).

Note: 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. They are normally used only for debug purposes.

Bit 2: BERR *Bus Error*

This bit indicates an I^2C 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 IDATA register.

Bit 1: ACK *Acknowledge bit*

Every ninth SCL of a write transfer the slave indicates reception of the byte by asserting ACK. The EZ-USB 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 EZ-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^2C Transfer DONE*

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

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	x	0	0	0

IVEC indicates the source of an interrupt from the Anchor USB Core. When the USB core generates an INT2 (USB) interrupt request, it updates IVEC to indicate the source of the interrupt. The twenty-one interrupt sources are encoded on IV[4..0] as shown in Figure 9-2 on page 138.

IN07IRQ **Endpoint 0-7 IN Interrupt Requests** **7FA9**

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**

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

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 (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 note).

Note:
Do not clear an IRQ bit by reading an IRQ register, OR'ing 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 the IRQ you want to clear) directly to the IRQ register.

USBIRQ **USB Interrupt Request** **7FAB**

b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	**URESIR**	**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

USBIRQ indicates the interrupt request status of the USB reset, suspend, setup token, start of frame, and setup data available interrupts.

Bit 4: **URESIR** *USB Reset Interrupt Request*

The EZ-USB core sets this bit to 1 when it detects a USB bus reset.

Since 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 10, EZ-USB Resets" for more information about this bit.

Bit 3: **SUSPIR** *USB Suspend Interrupt Request*

The EZ-USB core sets this bit to 1 when it detects USB SUSPEND signaling (no bus activity for 3 milliseconds). Write a 1 to this bit to clear the interrupt request.

Since 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 11, "EZ-USB Power Management" for more information about this bit.

Bit 2: **SUTOKIR** *SETUP Token Interrupt Request*

The EZ-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 7, EZ-USB Endpoint Zero" for more information on the handling of SETUP tokens.

Since 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 EZ-USB core sets this bit to 1 when it receives an SOF packet. Write a 1 to this bit to clear the interrupt condition.

Since 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 EZ-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 condition.

Since 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.

IN07IEN **Endpoint 0-7 IN Interrupt Enables** **7FAC**

b7	b6	b5	b4	b3	b2	b1	b0
IN7IEN	**IN6EN**	**IN5IEN**	**IN4IEN**	**IN3IEN**	**IN2IEN**	**IN0IEN**	**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**

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
0	0	0	0	0	0	0	0

The Endpoint Interrupt Enable registers define which endpoints have active 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.

Note: 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 Enables** **7FAE**

b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	**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

USBIEN bits gate the interrupt requests to the 8051 for USB reset, suspend, SETUP token, start of frame, and SETUP data available.

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 SOFIR 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 SUDAVIR 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
-	-	-	-	**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

Bit 3: **BREAK** *Breakpoint Enable*

The BREAK bit is set when the 8051 address bus matches the address held in the breakpoint address registers (next page). 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 CLK24 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 Auto-vector feature is enabled. If it is 0, the auto-vector feature is disabled. See Chapter 9 for more information on the auto-vector feature.

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
0	0	0	0	0	0	0	0

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 CLK24 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.

12.11 Endpoint 0 Control and Status Registers

EP0CS **Endpoint Zero Control and Status** **7FB4**

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**

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**

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

These registers control EZ-USB 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 EZ-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 EZ-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 one*** to the register bit.

While HSNAK=1, the EZ-USB core NAK's the handshake (status) phase of the CONTROL transfer. When HSNAK=0, it ACK's 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 one to the register bit.

While EP0STALL=1, the EZ-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.

Note:
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 since the host keeps the control transfer pending.

12.12 Endpoint 1-7 Control and Status Registers

Endpoints 1-7 IN and 1-7 OUT are used for bulk or interrupt data. Table 12-5 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 the previous section.

Table 12-5. Control and Status register addressess 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	OUT3CS
7FCB	Byte Count – Endpoint OUT3	OUT3BC
7FCC	Control and Status – Endpoint OUT4	OUT4CS
7FCD	Byte Count – Endpoint OUT4	OU4TBC
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***

b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	-	**INnBSY**	**INnSTL**
R	R	R	R	R	R	R	R/W
0	0	0	0	0	0	0	0

* See Table 12-5 on page 186 for individual control/status register addresses.

Bit 1: **INnBSY** IN Endpoint (1-7) Busy

The "BSY" bit indicates the status of the endpoint's IN Buffer INnBUF. The EZ-USB core sets BSY=0 when the endpoint's IN buffer is empty and ready for loading by the 8051. The 8051 sets 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 NAK'ed 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 dis-arm a previously armed endpoint. 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.

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 EZ-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 chip ReNumerates™ by pulsing the DISCON bit HI.

INnBC **Endpoint (1-7) IN Byte Count** **7FB7-7FC3***

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 12-5 on page 186 for individual byte count register addresses.

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 BulkTransfers") 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/Status** **7FC6-7FD2***

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 12-5 on page 186 for individual control/status register addresses.

Bit 1: **OUTnBSY** *OUT Endpoint (1-7) Busy*

The "BSY" bit indicates the status of the endpoint's OUT Buffer OUTnBUF. The EZ-USB core sets BSY=0 when 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 OUTnBUF is available for the 8051 to read. USB OUT tokens for the endpoint are NAK'ed 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 EZ-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 chip ReNumerates™.

OUTnBC **Endpoint (1-7) OUT Byte Count** **7FC7-7FD3***

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 12-5 on page 186 for individual byte count register addresses.

The 8051 reads this register to determine the number of bytes sent to an OUT endpoint. Legal sizes are 0 to 64 bytes.

Each EZ-USB 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 EZ-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 EZ-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 NAK'ed until the 8051 is released from RESET, whereupon the ACK/NAK behavior is based on pairing.

12.13 Global USB Registers

SUDPTRH **Setup Data Pointer High** **7FD4**

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**

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

When the EZ-USB chip receives a "Get_Descriptor" request on endpoint zero, it can instruct the EZ-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.

NOTE:
Any host request that uses the EZ-USB Setup Data Pointer to transfer IN data must indicate the number of bytes to transfer in bytes 6 (wLengthL) 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 EZ-USB core how many bytes to transfer using the Setup Data Pointer.

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

Bit 7: **WAKESRC** *Wakeup Source*

This bit indicates that a high to low transition 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 DISCON# pin reflects the complement of this bit. This bit is normally set to 0 so that the action of the DISCOE bit (below) either floats the DISCON# pin or drives it HI.

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).

Bit 1: **RENUM** ReNumerate™

This bit controls which entity, the USB core or the 8051, handles USB device requests. When RENUM=0, the EZ-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 EZ-USB core automatically sets RENUM=1 under two conditions:

1. Completion of a “B2” boot load (Chapter 5).
2. When external memory is used (EA=1) and no boot I^2C EEPROM is used (Section 10.3.3).

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 milliseconds, then sets SIGRSUME=0.

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

Bit 7: **Q** *Data Toggle Value*

Q=0 indicates DATA0 and Q=1 indicates DATA1, for the endpoint selected by the IO 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.

NOTE:
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 7, “EZ-USB 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 bit 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

Every millisecond the host sends a SOF token indicating “Start Of Frame”, along with an eleven-bit incrementing frame count. The EZ-USB 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 7, “EZ-USB Endpoint Zero).

If the Anchor USB core detects a missing or garbled SOF, it generates an internal SOF and increments USBFRAMEL-USBFRAMEH.

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

During the USB enumeration process, the host sends a device a unique 7-bit address, which the EZ-USB core copies into this register. There is normally no reason for the CPU to know its USB device address since the Anchor Core automatically responds only to its assigned address.

Note:
During ReNumeration™ the Anchor USB core sets register to 0 to allow the EZ-USB chip to respond to the default address 0.

USBPAIR **USB Endpoint Pairing** **7FDD**

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

Bit 7: ISOSEND0 *Isochronous Send Zero Length Data Packet*

The ISOSEND0 bit is used when the EZ-USB chip receives an isochronous IN token while the IN FIFO is empty. If ISOSEND0=0 (the default value), the EZ-USB core does not respond to the IN token. If ISOSEND0=1, the EZ-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**

b7	b6	b5	b4	b3	b2	b1	b0
IN7VAL	**IN6VAL**	**IN5VAL**	**IN4VAL**	**IN3VAL**	**IN2VAL**	**IN1VAL**	**1**
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**

b7	b6	b5	b4	b3	b2	b1	b0
OUT7VAL	**OUT6VAL**	**OUT5VAL**	**OUT4VAL**	**OUT3VAL**	**OUT2VAL**	**OUT1VAL**	**1**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	1	0	1	0	1	0	1

The 8051 sets VAL=1 for any active endpoints, and VAL=0 for inactive endpoints. These bits instruct the EZ-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 Anchor USB device (see Table 5-1 on page 52).

INISOVAL Isochronous IN Endpoint Valid Bits 7FE0

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

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

The 8051 sets VAL=1 for active endpoints, and VAL=0 for inactive endpoints. These bits instruct the EZ-USB core to return “no response” if an invalid endpoint is accessed.

The default values of these registers are set to support all endpoints that exist in the default Anchor USB device (see Table 5-1 on page 52).

12.14 Fast Transfers

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

The EZ-USB core provides a fast transfer mode that improves the 8051 transfer speed between external logic and 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 all sixteen isochronous endpoint FIFOS. When FISO=0 fast transfers are disabled for all sixteen isochronous endpoints.

Bit 6: FBLK Enable Fast BULK Transfers

The 8051 sets FBLK=1 to enable fast bulk transfers using the Autopointer (next section) 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. See Figure 8-11 on page 130.

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. See Figure 8-12 on page 131.

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

These registers implement the EZ-USB *Autopointer*.

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 RAM. The 8051 can read these registers to determine the address of the next byte to be accessed via the AUTODATA register.

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 Bulk Transfers” and Chapter 8, “EZ-USB Isochronous Transfers” explain how to use the Autopointer for fast transfers to and from the EZ-USB endpoint buffers.

12.15 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/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

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. The 8051 responds to the SUDAV interrupt by reading the SETUP bytes from this buffer.

12.16 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 12-6 for individual start address register addresses

Table 12-6. 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 Isochronous endpoints use a pool of 1024 double-buffered FIFO bytes. The 1024 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.

See Chapter 8, “EZ-USB Isochronous Endpoints”, Section 8-8 for details about how to set these registers.

13 EZ-USB AC/DC Parameters

13.1 ELECTRICAL CHARACTERISTICS

13.1.1 ABSOLUTE MAXIMUM RATINGS

Storage Temperature . . . . . .-65 °C to +150 °C
Ambient Temperature Under Bias . . . . . -40 °C to +85 °C
Supply Voltage to Ground Potential . . . . . –0.5V to +4.0 V
DC Input Voltage to Any Pin . . . . . .–0.5V to +5.8 V

13.1.2 OPERATING CONDITIONS

Ta (Ambient Temperature Under Bias) . . . . . 0 °C to +70 °C
Supply Voltage . . . . . +3.0V to +3.6 V
Ground Voltage . . . . . 0 V
F_{OSC} (Oscillator or Crystal Frequency). . . . . .. .12 MHz +/- 0.25%

13.1.3 DC CHARACTERISTICS

Symbol	Parameter	Condition	Min	Max	Unit	Notes
V_{CC}	Supply Voltage		3.0	3.6	V	
V_{IH}	Input High Voltage		2	5.25	V	
V_{IL}	Input Low Voltage		-.5	.8	V	
I_I	Input Leakage Current	$0< V_{IN} < V_{CC}$		$\pm$ 10	uA	
V_{OH}	Output Voltage High	I_{OUT} = 1.6 ma	2.4		V	
V_{OL}	Output Low Voltage	I_{OUT} = -1.6 ma		.8	V	
C_{IN}	Input Pin Capacitance			10	pF	
I_{SUSP}	Suspend Current			275	uA	
I_{CC}	Supply Current	8051 running, connected to USB		50	mA	

13.1.4 AC ELECTRICAL CHARACTERISTICS

Specified Conditions: Capacitive load on all pins = 30 pF

13.1.5 GENERAL MEMORY TIMING

Symbol	Parameter	Min	Typ	Max	Unit	Notes
tCL	1/CLK24 Frequency		41.66		ns	
tAV	Delay from Clock to Valid Address	0		10	ns	
tCD	Delay from CLK24 to CS#	2		15	ns	
tOED	Delay from CLK24 to OE#	2		15	ns	
tWD	Delay from CLK24 to WR#	2		15	ns	
tRD	Delay from CLK24 to RD#	2		15	ns	
tPD	Delay from CLK24 to PSEN#	2		15	ns	

13.1.6 PROGRAM MEMORY READ

Symbol	Parameter	Formula	Min	Max	Unit	Notes
tAA1	Address Access Time	3tCL-tAV-TDSU1	103		ns	
tAH1	Address Hold from CLK24	tCL+1	42		ns	
tDSU1	Data Setup to CLK24		12		ns	
tDH1	Data Hold from CLK24		0		ns	

13.1.7 DATA MEMORY READ

Symbol	Parameter	Formula	Min	Max	Unit	Notes
tAA2	Address Access Time	3tCL-tAV-TDSU1	103		ns	
tAH2	Address Hold from CLK24	tCL+1	42		ns	
tDSU2	Data Setup to CLK24		12		ns	
tDH2	Data Hold from CLK24		0		ns	

13.1.8 DATA MEMORY WRITE

Symbol	Parameter	Formula	Min	Max	Unit	Notes
tAH3	Address Hold from CLK24	tCL+2	43		ns	
tDZV	CLK24 to Data Valid			15	ns	
tDVZ	CLK24 to High Impedance	tCL+16	57		ns	

13.1.9 FAST DATA WRITE

Symbol	Parameter	Conditions	Min	Max	Unit	Notes
tCDO	Clock to Data Output Delay		3	15	ns	
tCWO	Clock to FIFO Write Output Delay		2	10	ns	
tPFWD	Propagation Delay Difference from FIFO Write to Data Out		1		ns	

13.1.10 FAST DATA READ

Symbol	Parameter	Conditions	Min	Max	Unit	Notes
tCRO	Clock to FIFO Read Output Delay		2	10	ns	
tDSU4	Data Setup to Rising CLK24		12		ns	
tDH4	Data Hold to Rising CLK24		0		ns	

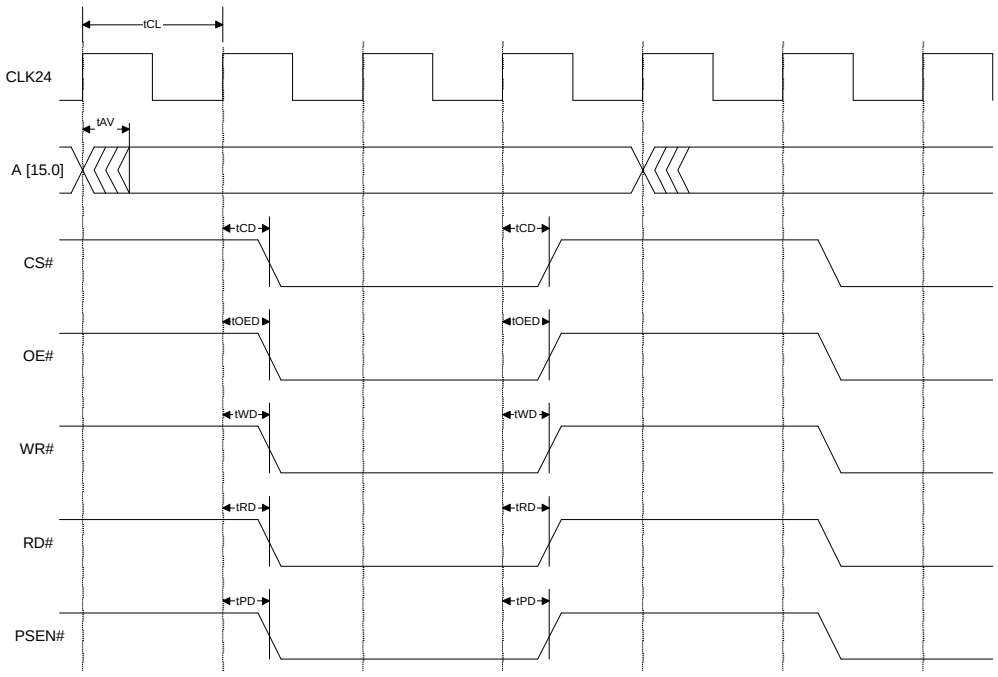


Figure 13-1. External memory timing

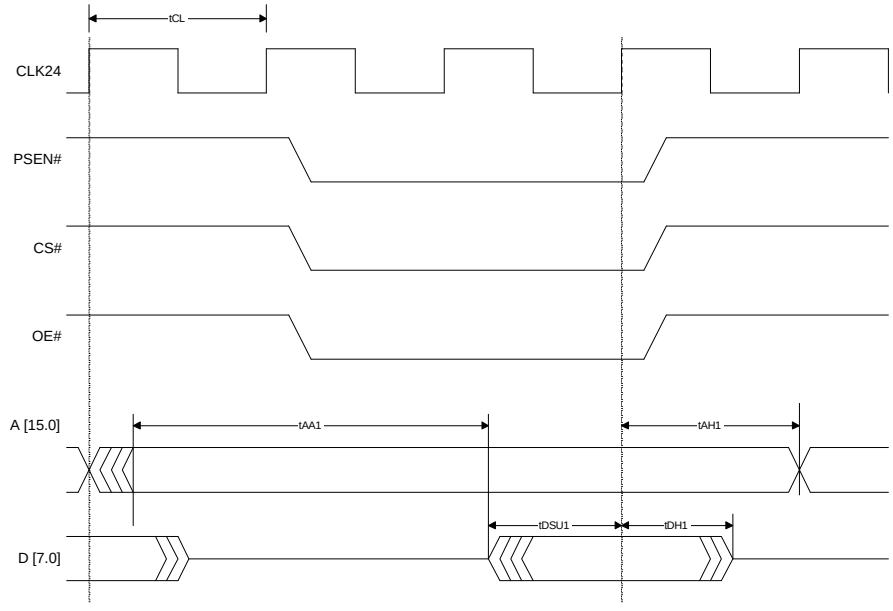


Figure 13-2. Program memory read timing

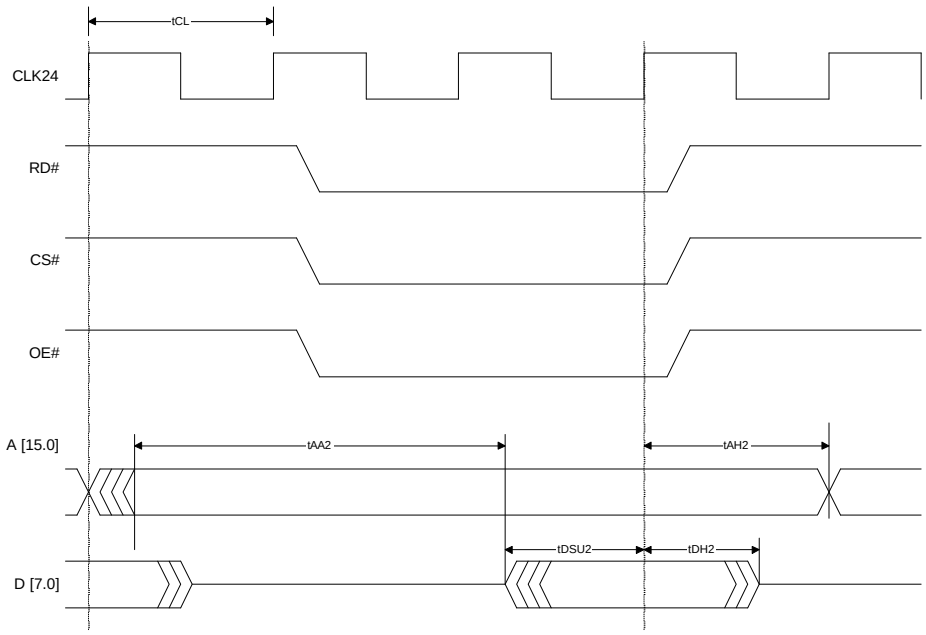


Figure 13-3. Data memory read timing

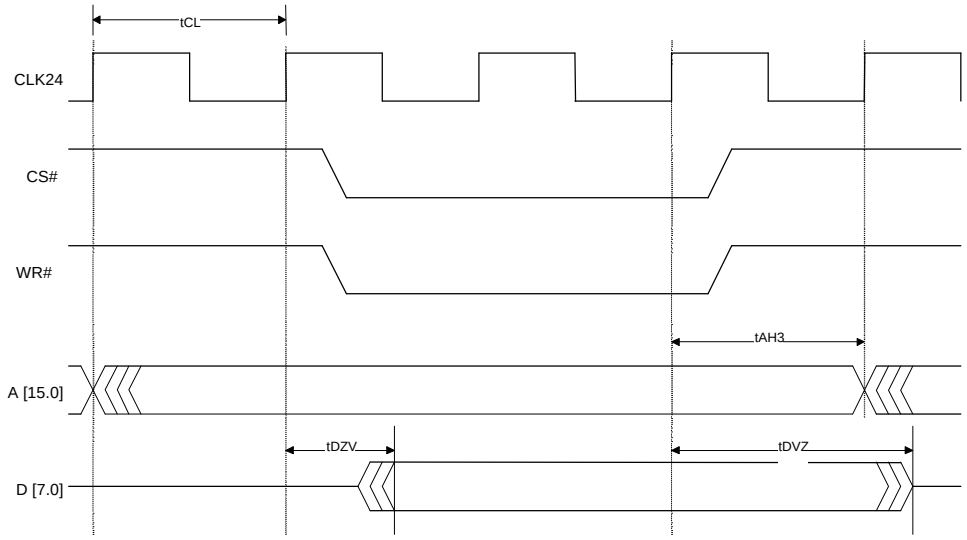


Figure 13-4. Data memory write timing

EZ-USB
Fast Transfer Block Diagram

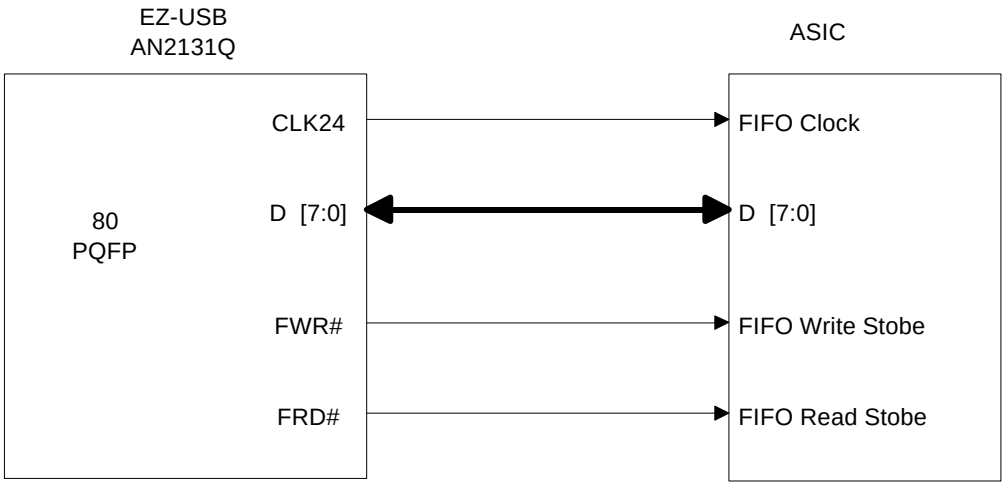


Figure 13-5. Fast transfer mode block diagram

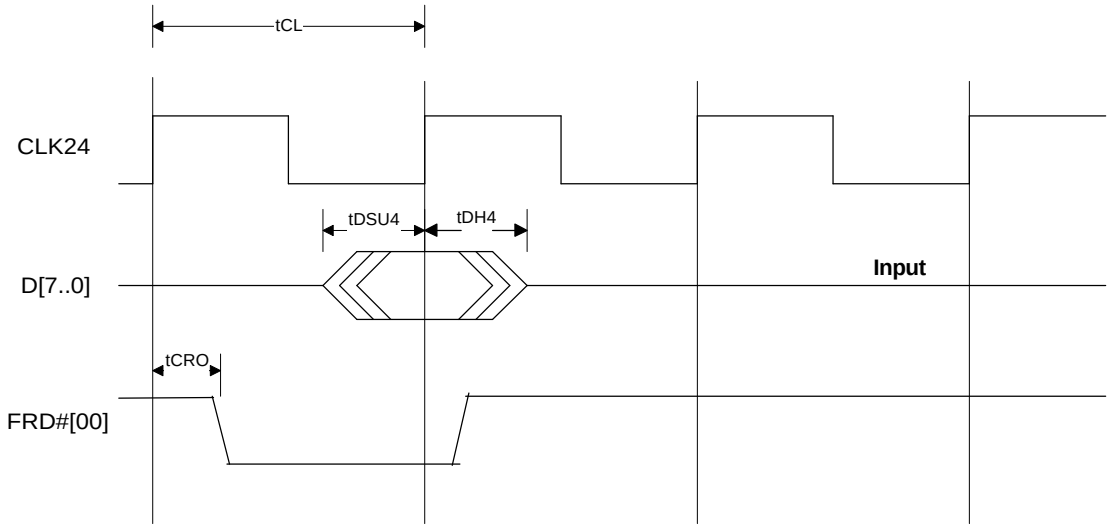


Figure 13-6. Fast transfer read timing [mode 00]

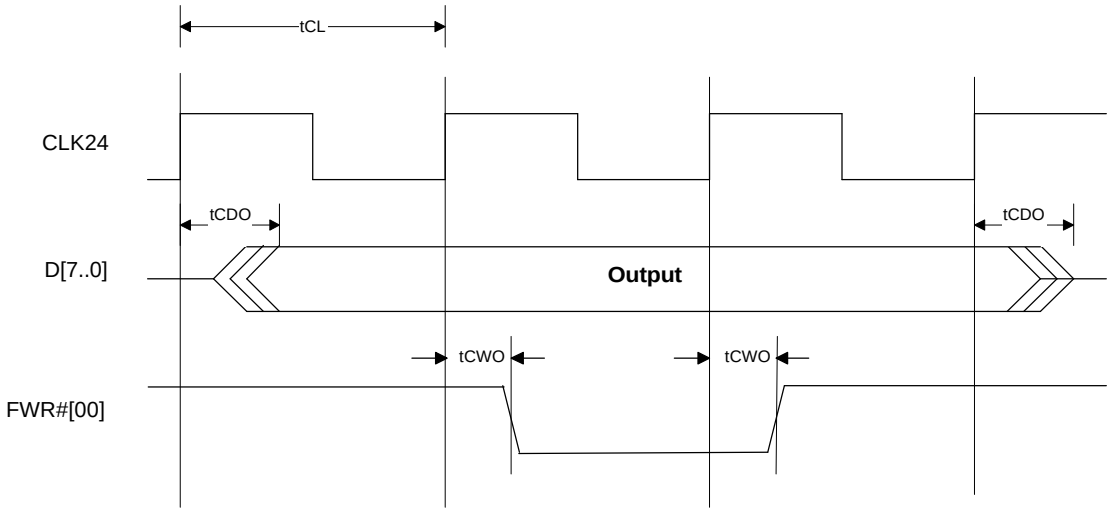


Figure 13-7. Fast transfer write timing [mode 00]

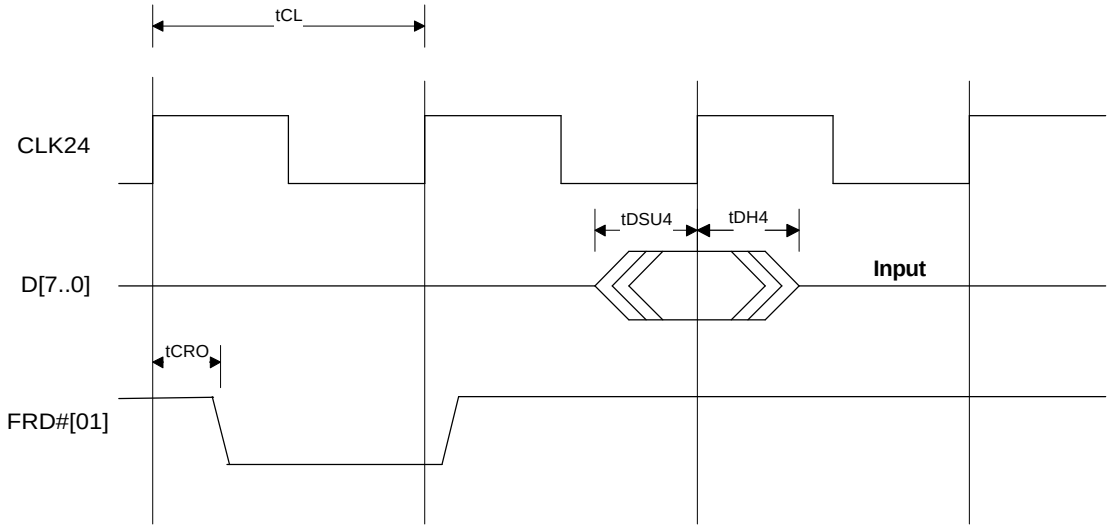


Figure 13-8. Fast transfer read timing [mode 01]

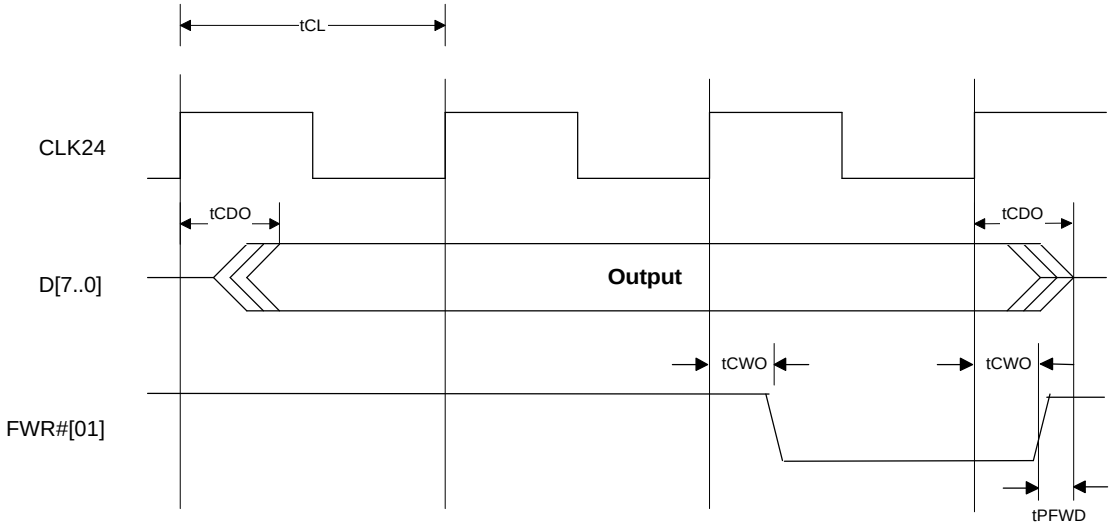


Figure 13-9. Fast transfer write timing [mode 01]

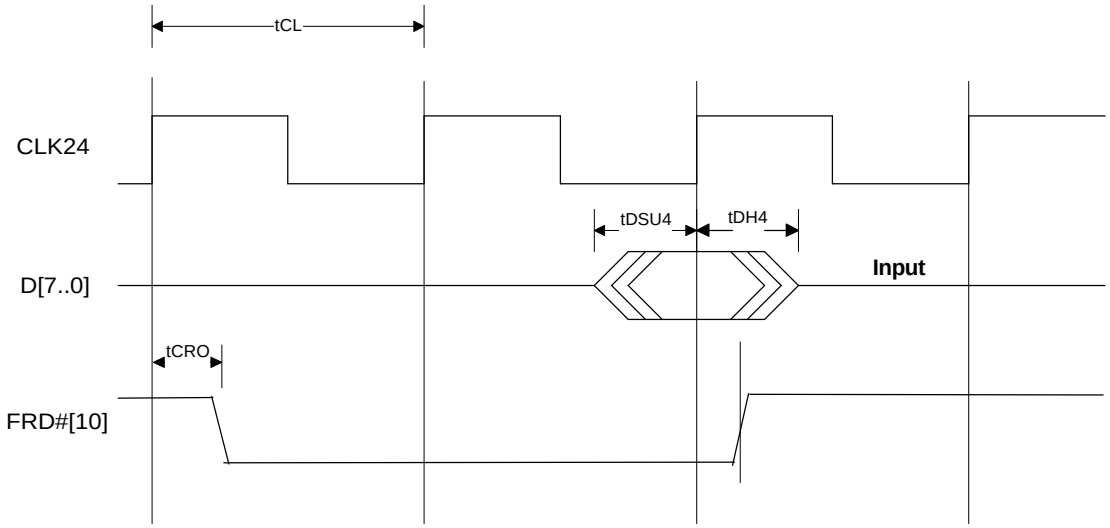


Figure 13-10. Fast transfer read timing [mode 10]

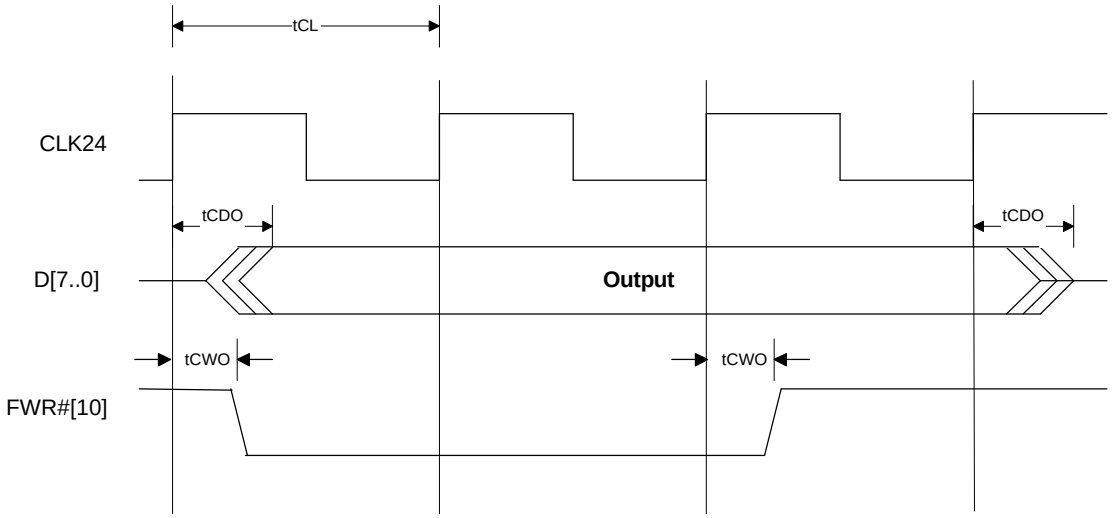


Figure 13-11. Fast transfer write timing [mode 10]

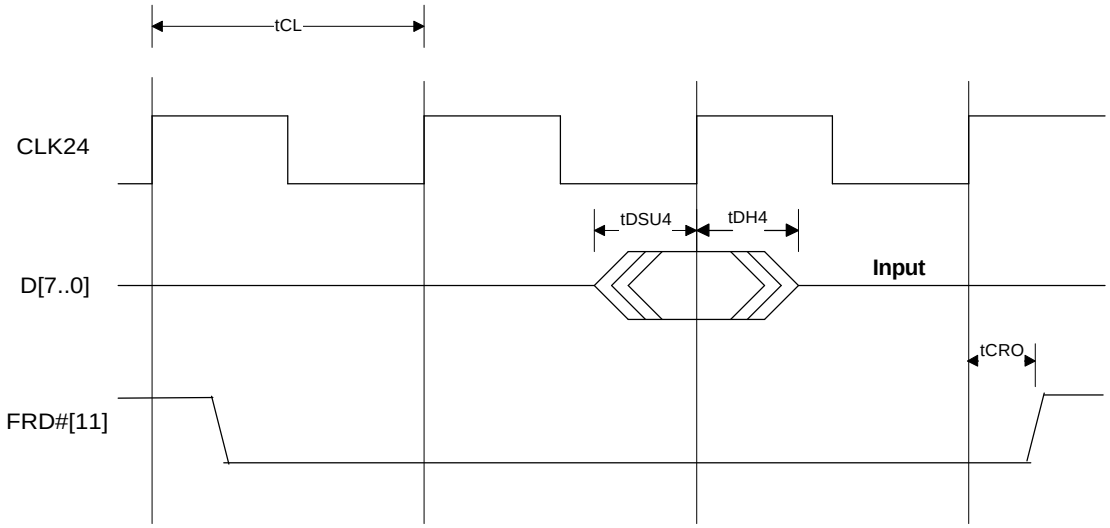


Figure 13-11. Fast transfer read timing [mode 11]

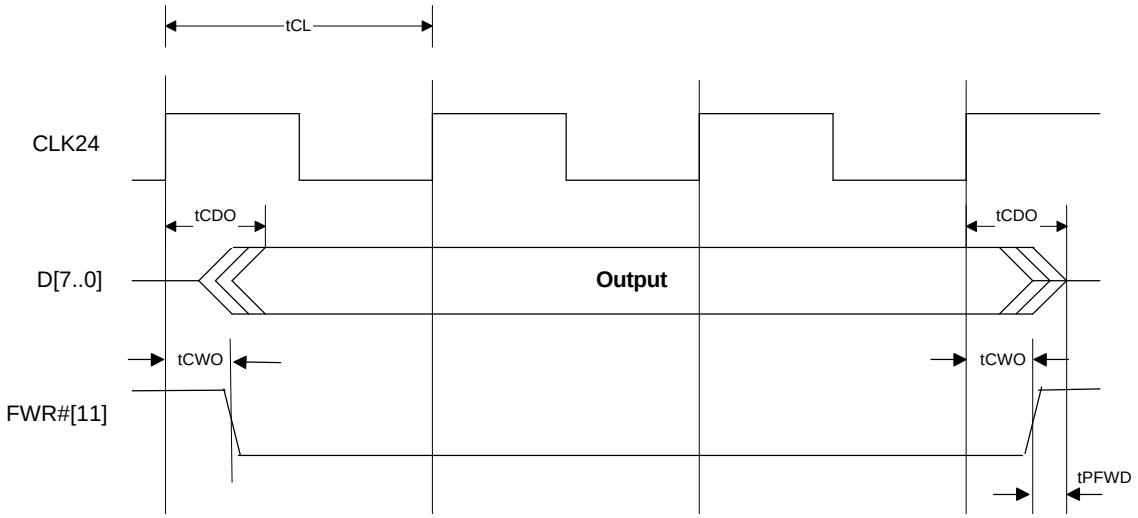


Figure 13-12. Fast transfer write timing [mode 11]

14 EZ-USB Packaging

14.1 44-Pin PQFP Package

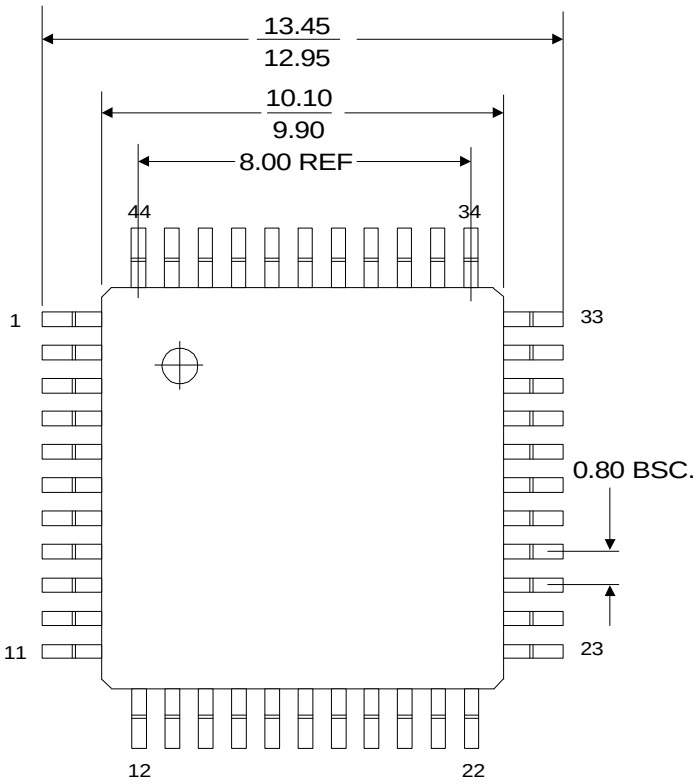


Figure 14-1. 44-Pin PQFP Package (top view)

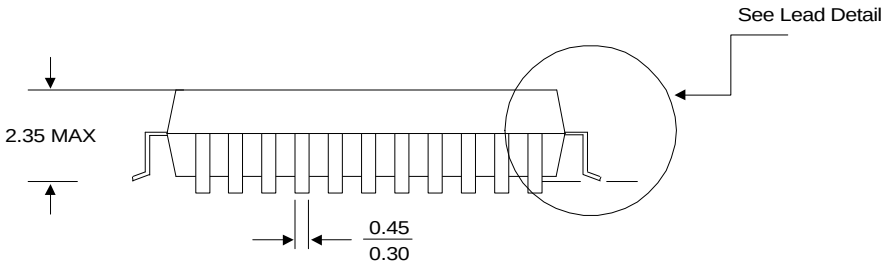
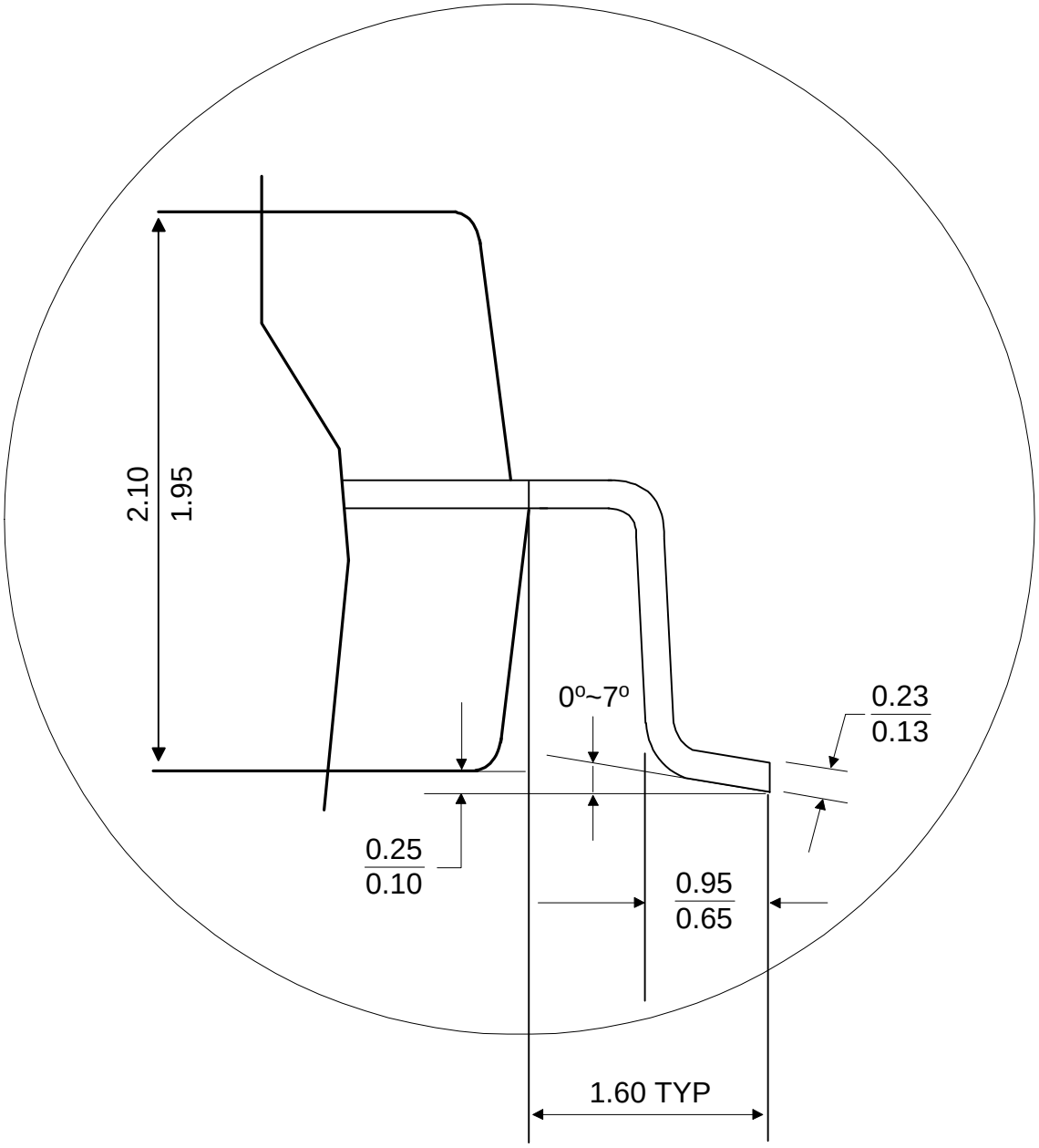


Figure 14-2. 44-Pin PQFP Package (side view)



Lead Detail: A(S=N/S)

Figure 14-3. 44-Pin PQFP Package (detail view)

14.2 80-Pin PQFP Package

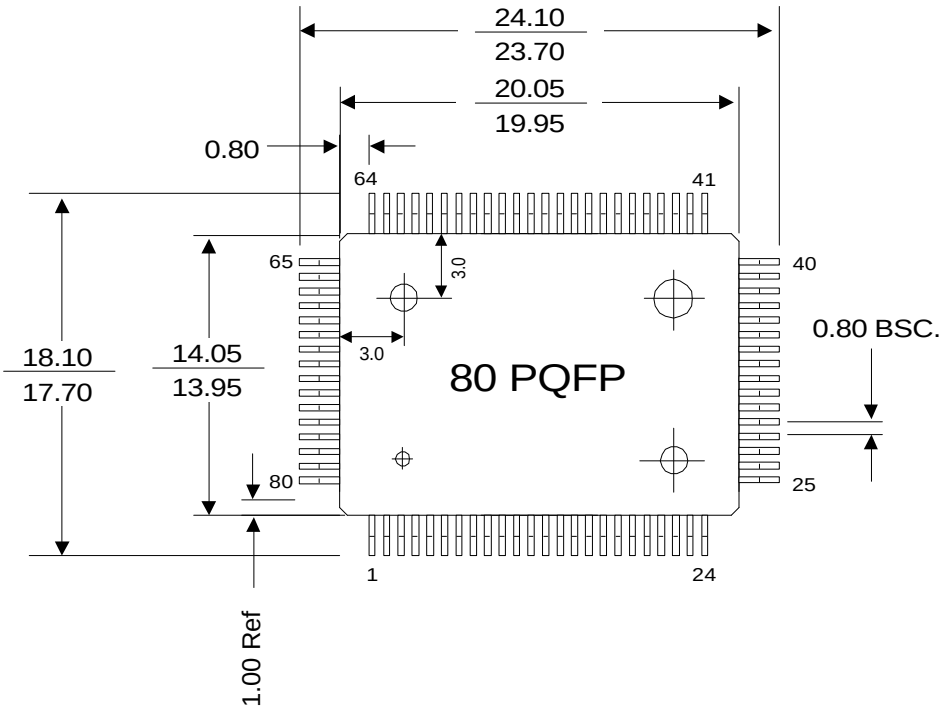


Figure 14-4. 80-Pin PQFP Package (top view)

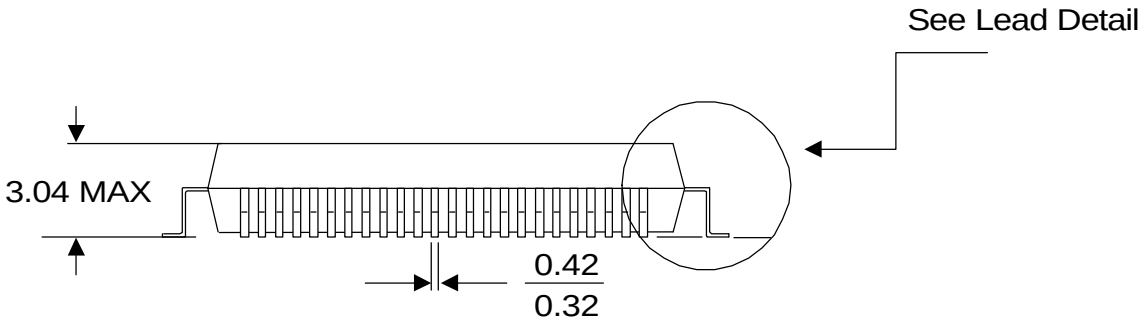


Figure 14-5. 80-Pin PQFP Package (side view)

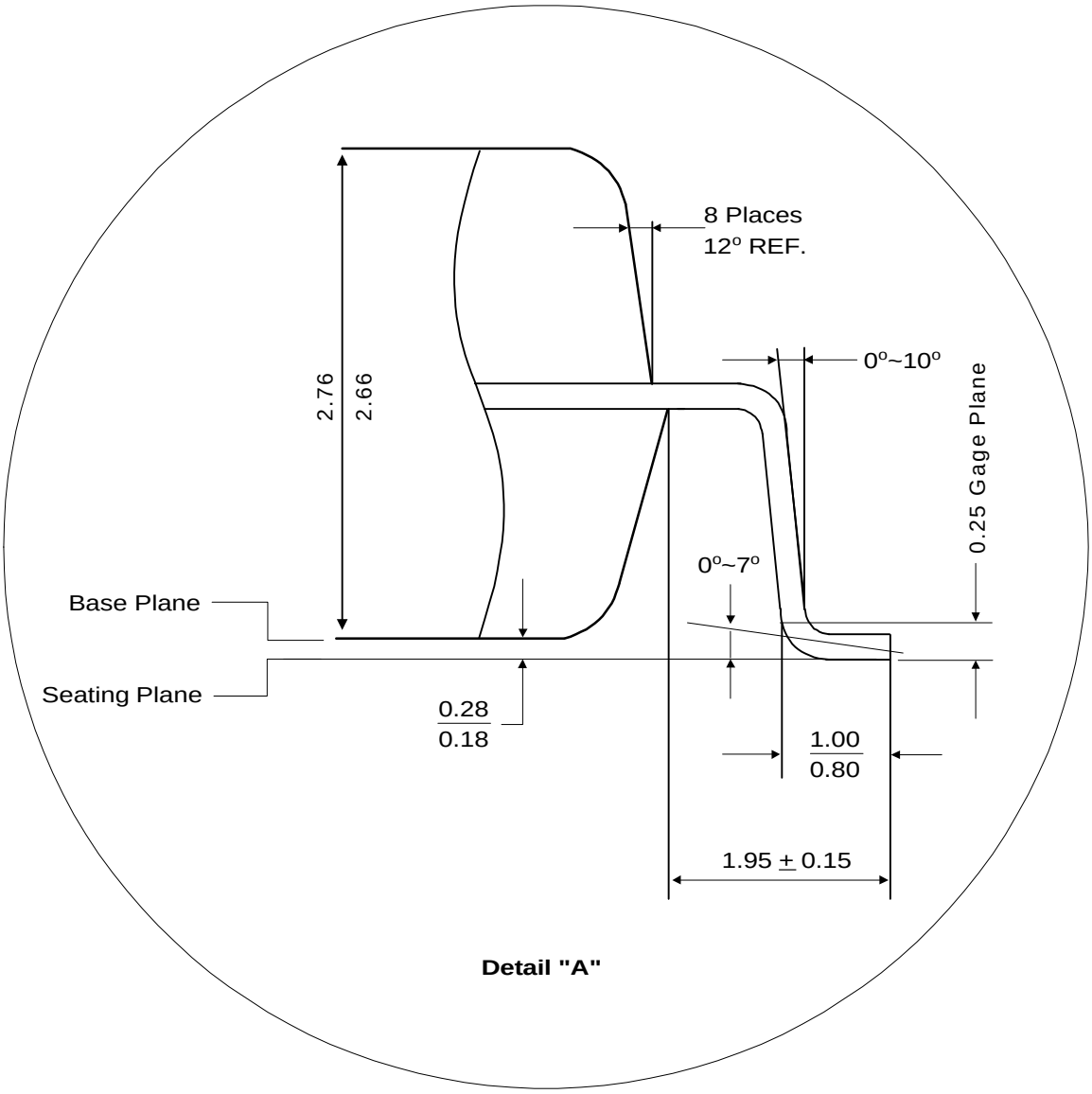


Figure 14-6. 80-Pin PQFP Package (side view)