Welcome to E-XFL.COM

Embedded - Microcontrollers - Application Specific: Tailored Solutions for Precision and Performance

Embedded - Microcontrollers - Application Specific represents a category of microcontrollers designed with unique features and capabilities tailored to specific application needs. Unlike general-purpose microcontrollers, application-specific microcontrollers are optimized for particular tasks, offering enhanced performance, efficiency, and functionality to meet the demands of specialized applications.

What Are Embedded - Microcontrollers - Application Specific?

Application-specific microcontrollers are engineered to

Details	
Product Status	Obsolete
Applications	USB Microcontroller
Core Processor	8051
Program Memory Type	ROMless
Controller Series	AN213x
RAM Size	8K x 8
Interface	I²C, USB
Number of I/O	24
Voltage - Supply	3V ~ 3.6V
Operating Temperature	0°C ~ 70°C
Mounting Type	Surface Mount
Package / Case	80-QFP
Supplier Device Package	80-PQFP
Purchase URL	https://www.e-xfl.com/product-detail/infineon-technologies/an2131qc

Tables

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 (eight clocks) per byte.

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 USB device (enumerated by the USB 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."

For purposes of downloading 8051 code, the Default USB Device requires only CONTROL endpoint zero. Nevertheless, the USB default machine is enhanced to support other endpoints as shown in Figure 5-1 (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 pre-configured endpoints.

When the EZ-USB core establishes the Default USB 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 USB Device, so the EZ-USB core sets the corresponding EPVAL bits. Chapter 6, “EZ-Bulk Transfers” contains a detailed explanation of the EPVAL bits.

Tables 5-9 through 5-13 show the various descriptors returned to the host by the 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.

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

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

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

4. Write the endpoint 2 transfer program.

```
1  loop:      jnb    got_EP2_data,loop
2             clr    got_EP2_data          ; clear my flag
3  ;
4  ; The user sent bytes to OUT2 endpoint using the USB Control Panel.
5  ; Find out how many bytes were sent.
6  ;
7             mov    dptr,#OUT2BC          ; point to OUT2 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 OUT2 endpoint to the IN2 endpoint
13 ; buffer.  Number of bytes in r6 and r7.
14 ;
15            mov    dptr,#OUT2BUF         ; first data pointer points to EP2OUT buffer
16            inc    dps                   ; select the second data pointer
17            mov    dptr,#IN2BUF          ; second data pointer points to EP2IN buffer
18            inc    dps                   ; back to first data pointer
19 transfer:  movx   movx                  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 IN2BC.  This arms in IN transfer
28 ;
29            mov    dptr,#IN2BC
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 OUT2BC.  This arms the next OUT transfer.
34 ;
35            mov    dptr,#OUT2BC
36            movx   @dptr,a               ; use whatever is in acc
37            sjmp   loop                  ; start checking for another OUT2 packet
```

Figure 6-10. Background Program Transfers Endpoint 2-OUT Data to Endpoint 2-IN

The main program loop tests the “got_EP2_data” flag, waiting until it is set by the endpoint 2 OUT interrupt service routine in Figure 6-10. This indicates that a new data packet has arrived in OUT2BUF. Then the service routine is entered, where the flag is cleared in line 2. The number of bytes received in OUT2BUF is retrieved from the OUT2BC register (Endpoint 2 Byte Count) and saved in registers R6 and R7 in lines 7-10.

The dual data pointers are initialized to the source (OUT2BUF) and destination (IN2BUF) buffers for the data transfer in lines 15-18. These labels represent the start of the 64-byte buffers for endpoint 2-OUT and endpoint 2-IN, respectively. Each byte is read from the OUT2BUF buffer and written to the IN2BUF buffer in lines 19-25. The saved value of

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 NAKd. It is therefore possible that a CONTROL transfer could arrive while the 8051 is still servicing a previous one. In this case the previous CONTROL transfer service should be aborted and the new one serviced. The SUTOK interrupt gives advance warning that a new CONTROL transfer is about to over-write the eight SETUPDAT bytes.

If the 8051 stalls endpoint zero (by setting the EP0STALL and HSNAK bits to 1), the 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.

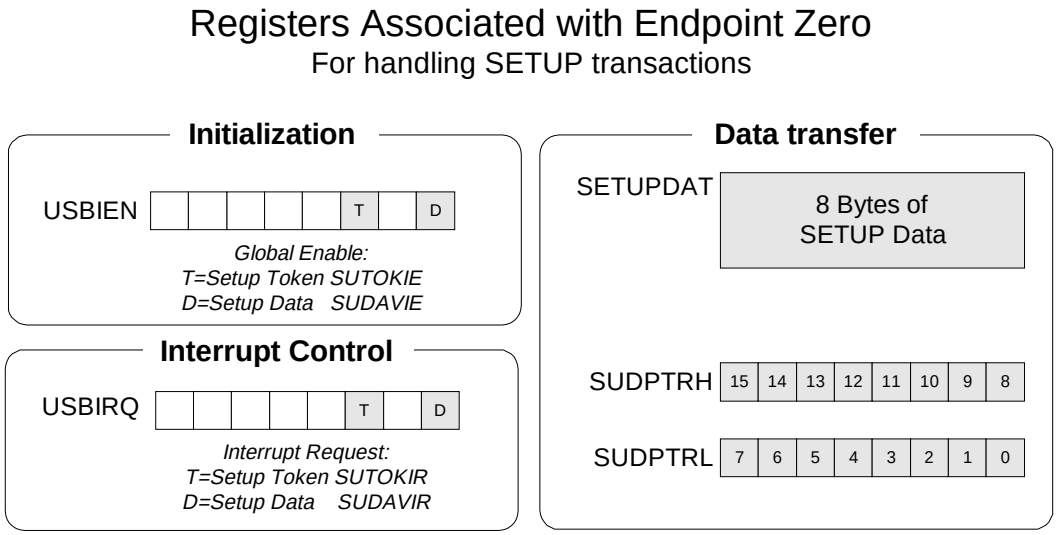

Figure 7-3. Registers Associated with EP0 Control Transfers

These registers augment those associated with normal bulk transfers over endpoint zero, which are described in Chapter 6, "EZ-USB Bulk Transfers."

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 disables these interrupts because 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	bmRequestType	**0x80**	IN, Device	*Set SUDPTR H-L to start of*
1	bRequest	**0x06**	"Get_Descriptor"	*Device Descriptor table in RAM*
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 2-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 7-11).

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

8.5 Isochronous Transfer Speed

The amount of data USB can transfer during a 1-ms frame is slightly more than 1,000 bytes per frame (1,500 bytes theoretical, without accounting for USB overhead and bus utilization). A device's actual isochronous transfer bandwidth is usually determined by how fast the CPU can move data in and out of its isochronous endpoint FIFOs.

The 8051 code example in Figure 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 ns). Thus, an 8051 cycle takes 4*42=168 ns, and the loop takes 9 cycles or 1.5 μs. This loop can transfer about 660 bytes into an IN FIFO every millisecond (1 ms/1.5 μs).

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 μs per byte. Using this method, the 8051 could transfer 1,000 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.

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 0x01 to the ISOCTL register.

Caution!

If you use this option, be absolutely certain that the host never sends isochronous data to your device. Isochronous data directed to a disabled isochronous endpoint system 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.

The USBIEN and USBIRQ registers control the first five interrupts shown in Figure 9-2. The IN07IEN and OUT07 registers control the remaining 16 USB interrupts, which correspond to the 16 bulk endpoints IN0-IN7 and OUT0-OUT7.

The 21 USB interrupts are now described in detail.

9.5 SUTOK, SUDAV Interrupts

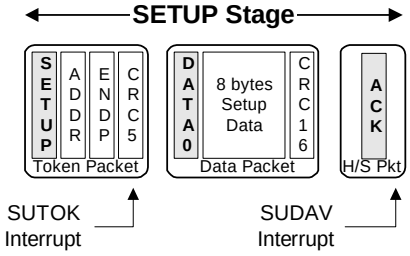

Figure 9-5. SUTOK and SUDAV Interrupts

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 the SETUP stage shown in Figure 7-1.) 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, "EZ-USB Endpoint Zero").

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

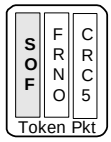

Figure 9-6. A Start Of Frame (SOF) Packet

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 3 ms 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 ms. 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 16 USB interrupt requests are indexed to the 16 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.

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

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

* AN2122/AN2126 only.

Figure 12-18. USB Interrupt Request (IRQ) Registers

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

Bit 5: IBNIR *IN Bulk NAK Interrupt Request*

This bit is in the AN2122 and AN2126 versions only. The EZ-USB core sets this bit when any of the IN bulk endpoints responds to an IN token with a NAK. This interrupt occurs when the host sends an IN token to a bulk IN endpoint which has not been *armed* by the 8051 writing its byte count register. Individual enables and requests (per endpoint) are controlled by the IBNIRQ and IBNIEN registers (7FB0, 7FB1).

Bit 4: URESIR *USB Reset Interrupt Request*

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

Because this bit can change state while the 8051 is in reset, it may be active when the 8051 comes out of reset, although it is reset to “0” by a power-on reset. Write a “1” to this bit to clear the interrupt request. See Chapter 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 ms). Write a “1” to this bit to clear the interrupt request.

Because this bit can change state while the 8051 is in reset, it may be active when the 8051 comes out of reset, although it is reset to “0” by a power-on reset. See Chapter 11, "EZ-USB Power Management" for more information about this bit.

IBNIRQ IN Bulk NAK Interrupt Requests 7FB0

b7	b6	b5	b4	b3	b2	b1	b0
-	**EP6IN**	**EP5IN**	**EP4IN**	**EP3IN**	**EP2IN**	**EP1IN**	**EP0IN**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

* AN2122/AN2126 only.

Figure 12-22. IN Bulk NAK Interrupt Request Register

These bits are set when a bulk IN endpoint (0-6) received an IN token while the endpoint was not *armed* for data transfer. In this case the SIE automatically sends a NAK response, and sets the corresponding IBNIRQ bit. If the IBN interrupt is enabled (USBIEN.5=1), and the endpoint interrupt is enabled in the IBNIEN register, an interrupt is request generated. The 8051 can test the IBNIRQ register to determine which of the endpoints caused the interrupt. The 8051 clears an IBNIRQ bit by writing a "1" to it.

IBNIEN IN Bulk NAK Interrupt Enables 7FB1

b7	b6	b5	b4	b3	b2	b1	b0
-	**EP6IN**	**EP5IN**	**EP4IN**	**EP3IN**	**EP2IN**	**EP1IN**	**EP0IN**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	0	0	0	0	0	0

Figure 12-23. IN Bulk NAK Interrupt Enable Register

Each of the individual IN endpoints may be enabled for an IBN interrupt using the IBNEN register. The 8051 sets an interrupt enable bit to 1 to enable the corresponding interrupt.

FNADDR **Function Address** **7FDB**

b7	b6	b5	b4	b3	b2	b1	b0
0	**FA6**	**FA5**	**FA4**	**FA3**	**FA2**	**FA1**	**FA0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

Figure 12-34. Function Address Register

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 because the USB Core automatically responds only to its assigned address.

Note

During ReNumeration™ the USB Core sets register to 0 to allow the EZ-USB chip to respond to the default address 0.

13.1.4 AC Electrical Characteristics

Specified Conditions: Capacitive load on all pins = 30 pF

13.1.5 General Memory Timing

Table 13-2. 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

Table 13-3. 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

Table 13-4. 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	

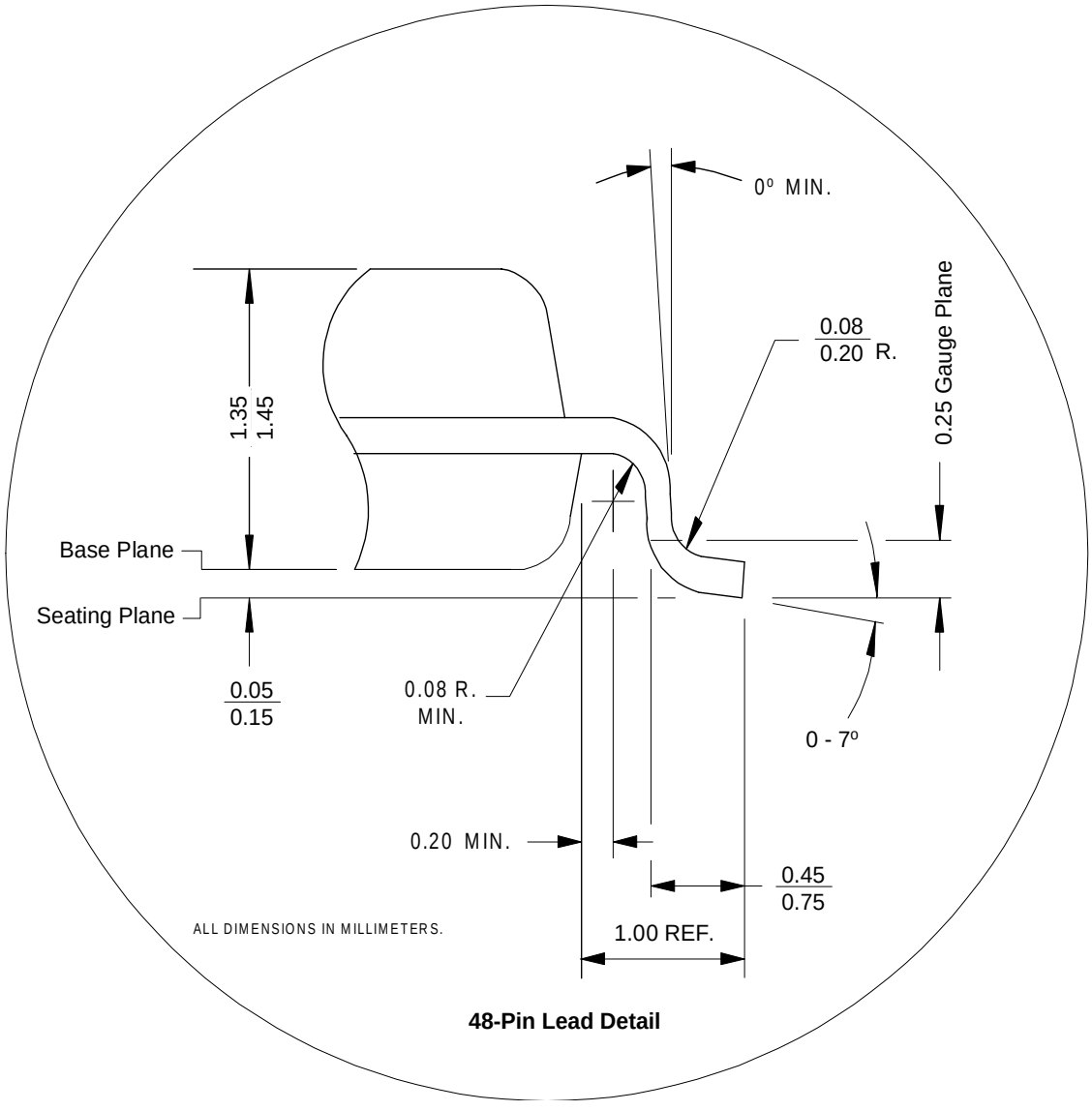

Figure 14-9. 48-Pin TQFP Package (Detail View)

Table B-2. 8051 Instruction Set

Mnemonic	Description	Byte	Instr. Cycles	Hex Code
XCH A, @Ri	Exchange A and data memory	1	1	C6-C7
XCHD A, @Ri	Exchange A and data memory nibble	1	1	D6-D7
* *Number of cycles is user-selectable. See* "Stretch Memory Cycles (Wait States)" on page B-10.				
Boolean				
CLR C	Clear carry	1	1	C3
CLR bit	Clear direct bit	2	2	C2
SETB C	Set carry	1	1	D3
SETB bit	Set direct bit	2	2	D2
CPL C	Complement carry	1	1	B3
CPL bit	Complement direct bit	2	2	B2
ANL C, bit	AND direct bit to carry	2	2	82
ANL C, /bit	AND direct bit inverse to carry	2	2	B0
ORL C, bit	OR direct bit to carry	2	2	72
ORL C, /bit	OR direct bit inverse to carry	2	2	A0
MOV C, bit	Move direct bit to carry	2	2	A2
MOV bit, C	Move carry to direct bit	2	2	92
Branching				
ACALL addr 11	Absolute call to subroutine	2	3	11-F1
LCALL addr 16	Long call to subroutine	3	4	12
RET	Return from subroutine	1	4	22
RETI	Return from interrupt	1	4	32
AJMP addr 11	Absolute jump unconditional	2	3	01-E1
LJMP addr 16	Long jump unconditional	3	4	02
SJMP rel	Short jump (relative address)	2	3	80
JC rel	Jump on carry = 1	2	3	40
JNC rel	Jump on carry = 0	2	3	50
JB bit, rel	Jump on direct bit = 1	3	4	20

C.3.3 Mode 1

Mode 1 provides standard asynchronous, full-duplex communication, using a total of 10 bits: 1 start bit, 8 data bits, and 1 stop bit. For receive operations, the stop bit is stored in RB8_0 (or RB8_1). Data bits are received and transmitted LSB first.

C.3.3.1 Mode 1 Baud Rate

The mode 1 baud rate is a function of timer overflow. Serial Port 0 can use either Timer 1 or Timer 2 to generate baud rates. Serial Port 1 can only use Timer 1. The two serial ports can run at the same baud rate if they both use Timer 1, or different baud rates if Serial Port 0 uses Timer 2 and Serial Port 1 uses Timer 1.

Each time the timer increments from its maximum count (FFh for Timer 1 or FFFFh for Timer 2), a clock is sent to the baud rate circuit. The clock is then divided by 16 to generate the baud rate.

When using Timer 1, the SMOD0 (or SMOD1) bit selects whether or not to divide the Timer 1 rollover rate by 2. Therefore, when using Timer 1, the baud rate is determined by the equation:

$$\text{Baud Rate} = \frac{2^{\text{SMODx}}}{32} \times \text{Timer 1 Overflow}$$

SMOD0 is SFR bit PCON.7; SMOD1 is SFR bit EICON.7.

When using Timer 2, the baud rate is determined by the equation:

$$\text{Baud Rate} = \frac{\text{Timer 2 Overflow}}{16}$$

To use Timer 1 as the baud rate generator, it is best to use Timer 1 mode 2 (8-bit counter with auto-reload), although any counter mode can be used. The Timer 1 reload is stored in the TH1 register, which makes the complete formula for Timer 1:

$$\text{Baud Rate} = \frac{2^{\text{SMODx}}}{32} \times \frac{\text{CLK24}}{12 \times (256 - \text{TH1})}$$

$$\text{Baud Rate} = \frac{\text{CLK24}}{32 \times (65536 - \text{RCAP2H,RCAP2L})}$$

where RCAP2H,RCAP2L is the content of RCAP2H and RCAP2L taken as a 16-bit unsigned number.

The 32 in the denominator is the result of CLK24 being divided by 2 and the Timer 2 overflow being divided by 16. Setting TCLK or RCLK to 1 automatically causes CLK24 to be divided by 2, as shown in Figure C-6., instead of the 4 or 12 determined by the T2M bit in the CKCON SFR.

To derive the required RCAP2H and RCAP2L values from a known baud rate, use the equation:

$$\text{RCAP2H,RCAP2L} = 65536 - \frac{\text{CLK24}}{32 \times \text{Baud Rate}}$$

When either RCLK or TCLK is set, the TF2 flag will not be set on a Timer 2 roll over, and the T2EX reload trigger is disabled.

Table C-11. Timer 2 Reload Values for Common Serial port Mode 1 Baud Rates

Nominal Rate	C/T2	Divisor	Reload Val	Actual Rate	Error
57600	0	13	F3	57692.31	0.16%
38400	0	20	EC	37500	-2.34%
28800	0	26	E6	28846.15	0.16%
19200	0	39	D9	19230.77	0.16%
9600	0	78	B2	9615.385	0.16%
4800	0	156	64	4807.692	0.16%
2400	0	312	FEC8	2403.846	0.16%
Note: using rates that are off by 2.3% or more will not work in all systems.					