CY7C64013
CY7C64113
Full-Speed USB (12 Mbps) Function

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF TABLES

1.0 Features

- Full-speed USB Microcontroller
- 8-bit USB Optimized Microcontroller
  - Harvard architecture
  - 6-MHz external clock source
  - 12-MHz internal CPU clock
  - 48-MHz internal clock
- Internal memory
  - 256 bytes of RAM
  - 8 KB of PROM (CY7C64013, CY7C64113)
- Integrated Master/Slave I^2C Controller (100 kHz) enabled through General-Purpose I/O (GPIO) pins
- Hardware Assisted Parallel Interface (HAPI) for data transfer to external devices
- I/O ports
  - Three GPIO ports (Port 0 to 2) capable of sinking 7 mA per pin (typical)
  - An additional GPIO port (Port 3) capable of sinking 12 mA per pin (typical) for high current requirements: LEDs
  - Higher current drive achievable by connecting multiple GPIO pins together to drive a common output
  - Each GPIO port can be configured as inputs with internal pull-ups or open drain outputs or traditional CMOS outputs
  - A Digital to Analog Conversion (DAC) port with programmable current sink outputs is available on the CY7C64113 devices
  - Maskable interrupts on all I/O pins
- 12-bit free-running timer with one microsecond clock ticks
- Watch dog timer (WDT)
- Internal power-on reset (POR)
- USB Specification Compliance
  - Conforms to USB Specification, Version 1.1
  - Conforms to USB HID Specification, Version 1.1
  - Supports up to five user configured endpoints
    - Up to four 8-byte data endpoints
    - Up to two 32-byte data endpoints
  - Integrated USB transceivers
- Improved output drivers to reduce EMI
- Operating voltage from 4.0V to 5.5V DC
- Operating temperature from 0 to 70 degrees Celsius
  - CY7C64013 available in 28-pin SOIC and 28-pin PDIP packages
  - CY7C64113 available in 48-pin SSOP packages
- Industry standard programmer support

2.0 Functional Overview

The CY7C64013 and CY7C64113 are 8-bit One Time Programmable microcontrollers that are designed for full-speed USB applications. The instruction set has been optimized specifically for USB operations, although the microcontrollers can be used for a variety of non-USB embedded applications.

The CY7C64013 features 19 GPIO pins to support USB and other applications. The I/O pins are grouped into three ports (P0[7:0], P1[7:0], P3[7,2,0]) where each port can be configured as inputs with internal pull-ups, open drain outputs, or traditional CMOS outputs. There are 16 GPIO pins (Ports 0 and 1) which are rated at 7 mA typical sink current. Port 3 pins are rated at 12 mA typical sink current, a current sufficient to drive LEDs. Multiple GPIO pins can be connected together to drive a single output for more drive current capacity. Additionally, each GPIO can be used to generate a GPIO interrupt to the microcontroller. All of the GPIO interrupts all share the same “GPIO” interrupt vector.

Thirty-two GPIO pins (P0[7:0], P1[7:0], P2[7:0], P3[7:0]) and four Digital to Analog Conversion (DAC) pins (P4[7,2:0]) are available on the CY7C64113. Every DAC pin includes an integrated 14-kΩ pull-up resistor. When a ‘1’ is written to a DAC I/O pin, the output current sink is disabled and the output pin is driven HIGH by the internal pull-up resistor. When a ‘0’ is written to a DAC I/O pin, the internal pull-up resistor is disabled and the output pin provides the programmed amount of sink current. A DAC I/O pin can be used as an input with an internal pull-up by writing a ‘1’ to the pin.

The sink current for each DAC I/O pin can be individually programmed to one of 16 values using dedicated Isink registers. DAC bits P4[1:0] can be used as high-current outputs with a programmable sink current range of 3.2 to 16 mA (typical). DAC bits P4[7,2] have a programmable current sink range of 0.2 to 1.0 mA (typical). Multiple DAC pins can be connected together to drive a single output that requires more sink current capacity. Each I/O pin can be used to generate a DAC interrupt to the microcontroller. Also, the interrupt polarity for each DAC I/O pin is individually programmable.

The microcontroller uses an external 6-MHz crystal and an internal oscillator to provide a reference to an internal PLL-based clock generator. This technology allows the customer application to use an inexpensive 6-MHz fundamental crystal that reduces the clock-related noise emissions (EMI). A PLL clock generator provides the 6-, 12-, and 48-MHz clock signals for distribution within the microcontroller.

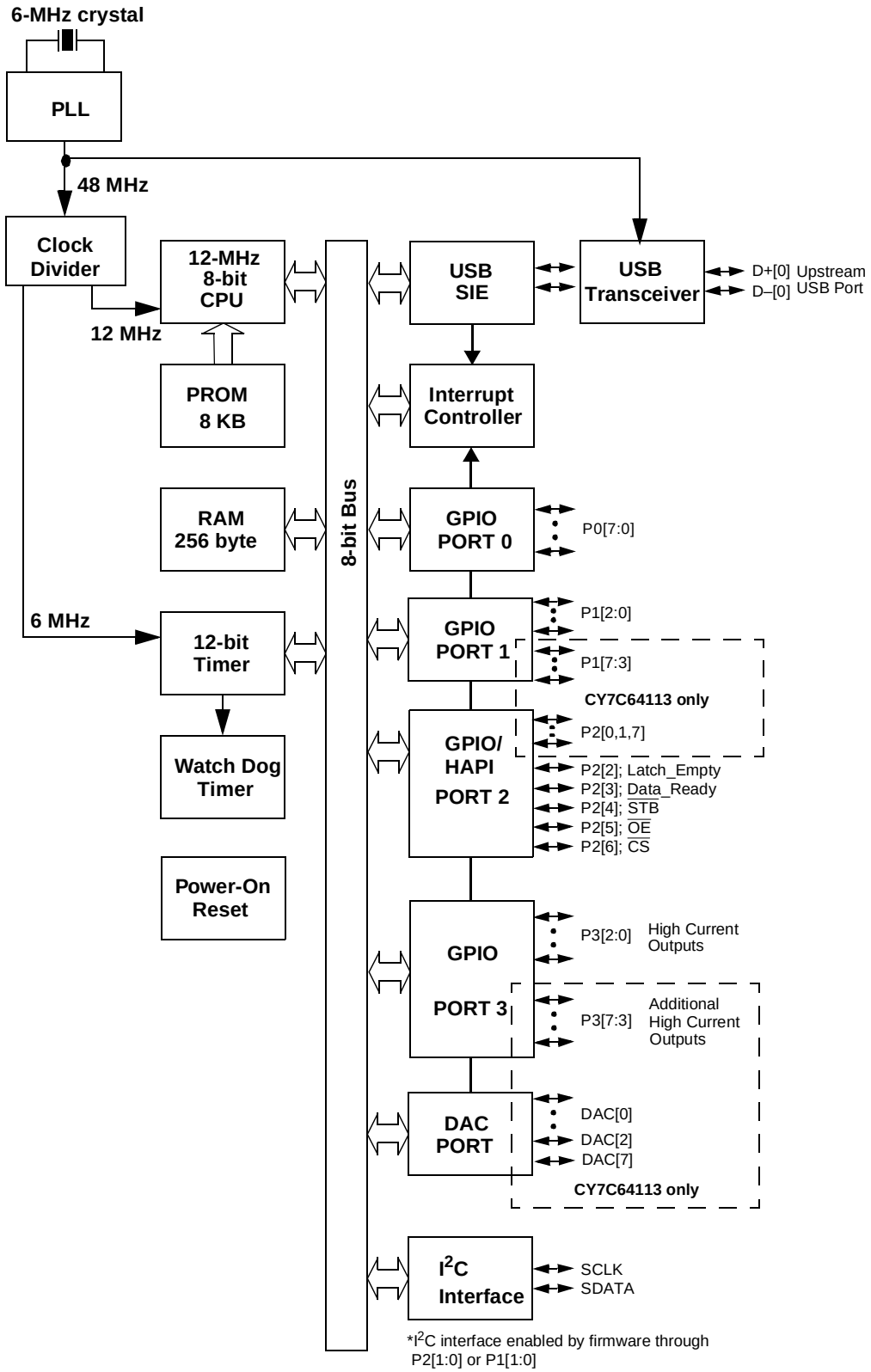
The CY7C64013 and CY7C64113 have 8 KB of PROM. These parts include power-on reset logic, a watch dog timer, and a 12-bit free-running timer. The power-on reset (POR) logic detects when power is applied to the device, resets the logic to a known state, and begins executing instructions at PROM address 0x0000. The watch dog timer is used to ensure the microcontroller recovers after a period of inactivity. The firmware may become inactive for a variety of reasons, including errors in the code or a hardware failure such as waiting for an interrupt that never occurs.

The microcontroller can communicate with external electronics through the GPIO pins. An I^2C interface accommodates a 100-kHz serial link with an external device. There is also a Hardware Assisted Parallel Interface (HAPI) which can be used to transfer data to an external device.

The free-running 12-bit timer clocked at 1 MHz provides two interrupt sources, 128-µs and 1.024-ms. The timer can be used to measure the duration of an event under firmware control by reading the timer at the start of the event and after the event is complete. The difference between the two readings indicates the duration of the event in microseconds. The upper four bits of the timer are latched into an internal register when the firmware reads the lower eight bits. A read from the upper four bits actually reads data from the internal register, instead of the timer. This feature eliminates the need for firmware to try to compensate if the upper four bits increment immediately after the lower eight bits are read.

The microcontroller supports 11 maskable interrupts in the vectored interrupt controller. Interrupt sources include the USB Bus Reset interrupt, the 128-µs (bit 6) and 1.024-ms (bit 9) outputs from the free-running timer, five USB endpoints, the DAC port, the GPIO ports, and the I^2C master mode interface. The timer bits cause an interrupt (if enabled) when the bit toggles from LOW ‘0’ to HIGH ‘1’. The USB endpoints interrupt after the USB host has written data to the endpoint FIFO or after the USB controller sends a packet to the USB host. The DAC ports have an additional level of masking that allows the user to select which DAC inputs can cause a DAC interrupt. The GPIO ports also have a level of masking to select which GPIO inputs can cause a GPIO interrupt. For additional flexibility, the input transition polarity that causes an interrupt is programmable for each pin of the DAC port. Input transition polarity can be programmed for each GPIO port as part of the port configuration. The interrupt polarity can be rising edge (‘0’ to ‘1’) or falling edge (‘1’ to ‘0’).

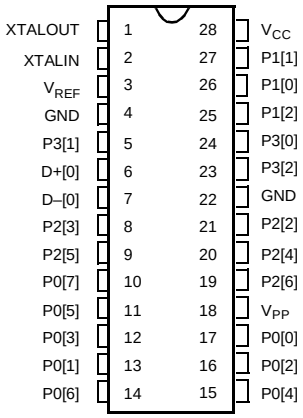
Logic Block Diagram



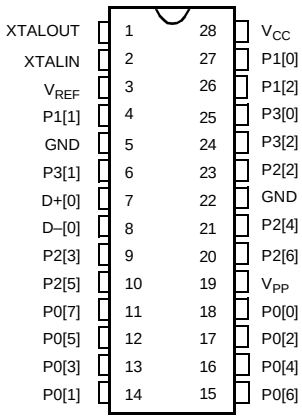
3.0 Pin Configurations

TOP VIEW

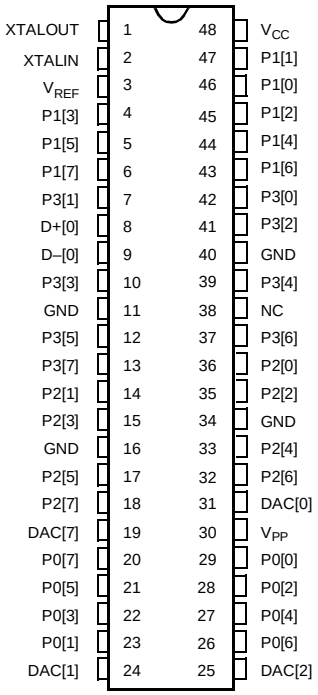
CY7C64013
28-pin SOIC



CY7C64013
28-pin PDIP



CY7C64113
48-pin SSOP



4.0 Product Summary Tables

4.1 Pin Assignments

Table 4-1. Pin Assignments

Name	I/O	28-Pin SOIC	28-Pin PDIP	48-Pin SSOP	Description
D+[0], D–[0]	I/O	6, 7	7, 8	7, 8	Upstream port, USB differential data.
P0	I/O	P0[7:0] 10, 14, 11, 15, 12, 16, 13, 17	P0[7:0] 11, 15, 12, 16, 13, 17, 14, 18	P0[7:0] 20, 26, 21, 27, 22, 28, 23, 29	GPIO Port 0 capable of sinking 7 mA (typical).
P1	I/O	P1[2:0] 25, 27, 26	P1[2:0] 26, 4, 27	P1[7:0] 6, 43, 5, 44, 4, 45, 47, 46	GPIO Port 1 capable of sinking 7 mA (typical).
P2	I/O	P2[6:2] 19, 9, 20, 8, 21	P2[6:2] 20, 10, 21, 9, 23	P2[7:0] 18, 32, 17, 33, 15, 35, 14, 36	GPIO Port 2 capable of sinking 7 mA (typical). HAPI is also supported through P2[6:2].
P3	I/O	P3[2:0] 23, 5, 24	P3[2:0] 24, 6, 25	P3[7:0] 13, 37, 12, 39, 10, 41, 7, 42	GPIO Port 3, capable of sinking 12 mA (typical).
DAC	I/O			DAC[7,2:0] 19, 25, 24, 31	DAC Port with programmable current sink outputs. DAC[1:0] offer a programmable range of 3.2 to 16 mA typical. DAC[7,2] have a programmable sink current range of 0.2 to 1.0 mA typical.
$XTAL_{IN}$	IN	2	2	2	6-MHz crystal or external clock input.
$XTAL_{OUT}$	OUT	1	1	1	6-MHz crystal out.
V_{PP}	IN	18	19	30	Programming voltage supply, tie to ground during normal operation.
V_{CC}	IN	28	28	48	Voltage supply.
GND	IN	4, 22	5, 22	11, 16, 34, 40	Ground.
V_{REF}	IN	3	3	3	External 3.3V supply voltage for the differential data output buffers and the D+ pull-up.
NC				38	No Connect

4.2 I/O Register Summary

I/O registers are accessed via the I/O Read (IORD) and I/O Write (IOWR, IOWX) instructions. IORD reads data from the selected port into the accumulator. IOWR performs the reverse; it writes data from the accumulator to the selected port. Indexed I/O Write (IOWX) adds the contents of X to the address in the instruction to form the port address and writes data from the accumulator to the specified port. Specifying address 0 (e.g., IOWX 0h) means the I/O register is selected solely by the contents of X.

All undefined registers are reserved. It is important not to write to reserved registers as this may cause an undefined operation or increased current consumption during operation. When writing to registers with reserved bits, the reserved bits must be written with ‘0.’

Table 4-2. I/O Register Summary

Register Name	I/O Address	Read/Write	Function	Page
Port 0 Data	0x00	R/W	GPIO Port 0 Data	18
Port 1 Data	0x01	R/W	GPIO Port 1 Data	18
Port 2 Data	0x02	R/W	GPIO Port 2 Data	18
Port 3 Data	0x03	R/W	GPIO Port 3 Data	18
Port 0 Interrupt Enable	0x04	W	Interrupt Enable for Pins in Port 0	19
Port 1 Interrupt Enable	0x05	W	Interrupt Enable for Pins in Port 1	19
Port 2 Interrupt Enable	0x06	W	Interrupt Enable for Pins in Port 2	19
Port 3 Interrupt Enable	0x07	W	Interrupt Enable for Pins in Port 3	19

Table 4-2. I/O Register Summary (continued)

Register Name	I/O Address	Read/Write	Function	Page
GPIO Configuration	0x08	R/W	GPIO Port Configurations	19
HAPI and I^2C Configuration	0x09	R/W	HAPI Width and I^2C Position Configuration	22
USB Device Address A	0x10	R/W	USB Device Address A	32
EP A0 Counter Register	0x11	R/W	USB Address A, Endpoint 0 Counter	33
EP A0 Mode Register	0x12	R/W	USB Address A, Endpoint 0 Configuration	32
EP A1 Counter Register	0x13	R/W	USB Address A, Endpoint 1 Counter	33
EP A1 Mode Register	0x14	R/W	USB Address A, Endpoint 1 Configuration	33
EP A2 Counter Register	0x15	R/W	USB Address A, Endpoint 2 Counter	33
EP A2 Mode Register	0x16	R/W	USB Address A, Endpoint 2 Configuration	33
USB Status & Control	0x1F	R/W	USB Upstream Port Traffic Status and Control	31
Global Interrupt Enable	0x20	R/W	Global Interrupt Enable	26
Endpoint Interrupt Enable	0x21	R/W	USB Endpoint Interrupt Enables	26
Interrupt Vector	0x23	R	Pending Interrupt Vector Read / Clear	28
Timer (LSB)	0x24	R	Lower 8 Bits of Free-running Timer (1 MHz)	21
Timer (MSB)	0x25	R	Upper 4 Bits of Free-running Timer	21
WDT Clear	0x26	W	Watch Dog Timer Clear	16
I^2C Control & Status	0x28	R/W	I^2C Status and Control	23
I^2C Data	0x29	R/W	I^2C Data	23
DAC Data	0x30	R/W	DAC Data	20
DAC Interrupt Enable	0x31	W	Interrupt Enable for each DAC Pin	21
DAC Interrupt Polarity	0x32	W	Interrupt Polarity for each DAC Pin	21
DAC Isink	0x38-0x3F	W	Input Sink Current Control for each DAC Pin	20
Reserved	0x40		Reserved	
EP A3 Counter Register	0x41	R/W	USB Address A, Endpoint 3 Counter	33
EP A3 Mode Register	0x42	R/W	USB Address A, Endpoint 3 Configuration	32
EP A4 Counter Register	0x43	R/W	USB Address A, Endpoint 4 Counter	33
EP A4 Mode Register	0x44	R/W	USB Address A, Endpoint 4 Configuration	33
Reserved	0x48		Reserved	
Reserved	0x49		Reserved	
Reserved	0x4A		Reserved	
Reserved	0x4B		Reserved	
Reserved	0x4C		Reserved	
Reserved	0x4D		Reserved	
Reserved	0x4E		Reserved	
Reserved	0x4F		Reserved	
Reserved	0x50		Reserved	
Reserved	0x51		Reserved	
Processor Status & Control	0xFF	R/W	Microprocessor Status and Control Register	25

4.3 Instruction Set Summary

Refer to the *CYASM Assembler User's Guide* for more details.

Table 4-3. Instruction Set Summary

MNEMONIC	operand	opcode	cycles		MNEMONIC	operand	opcode	cycles
HALT		00	7		NOP		20	4
ADD A,expr	data	01	4		INC A	acc	21	4
ADD A,[expr]	direct	02	6		INC X	x	22	4
ADD A,[X+expr]	index	03	7		INC [expr]	direct	23	7
ADC A,expr	data	04	4		INC [X+expr]	index	24	8
ADC A,[expr]	direct	05	6		DEC A	acc	25	4
ADC A,[X+expr]	index	06	7		DEC X	x	26	4
SUB A,expr	data	07	4		DEC [expr]	direct	27	7
SUB A,[expr]	direct	08	6		DEC [X+expr]	index	28	8
SUB A,[X+expr]	index	09	7		IORD expr	address	29	5
SBB A,expr	data	0A	4		IOWR expr	address	2A	5
SBB A,[expr]	direct	0B	6		POP A		2B	4
SBB A,[X+expr]	index	0C	7		POP X		2C	4
OR A,expr	data	0D	4		PUSH A		2D	5
OR A,[expr]	direct	0E	6		PUSH X		2E	5
OR A,[X+expr]	index	0F	7		SWAP A,X		2F	5
AND A,expr	data	10	4		SWAP A,DSP		30	5
AND A,[expr]	direct	11	6		MOV [expr],A	direct	31	5
AND A,[X+expr]	index	12	7		MOV [X+expr],A	index	32	6
XOR A,expr	data	13	4		OR [expr],A	direct	33	7
XOR A,[expr]	direct	14	6		OR [X+expr],A	index	34	8
XOR A,[X+expr]	index	15	7		AND [expr],A	direct	35	7
CMP A,expr	data	16	5		AND [X+expr],A	index	36	8
CMP A,[expr]	direct	17	7		XOR [expr],A	direct	37	7
CMP A,[X+expr]	index	18	8		XOR [X+expr],A	index	38	8
MOV A,expr	data	19	4		IOWX [X+expr]	index	39	6
MOV A,[expr]	direct	1A	5		CPL		3A	4
MOV A,[X+expr]	index	1B	6		ASL		3B	4
MOV X,expr	data	1C	4		ASR		3C	4
MOV X,[expr]	direct	1D	5		RLC		3D	4
reserved		1E			RRC		3E	4
XPAGE		1F	4		RET		3F	8
MOV A,X		40	4		DI		70	4
MOV X,A		41	4		EI		72	4
MOV PSP,A		60	4		RETI		73	8
CALL	addr	50 - 5F	10		JC	addr	C0-CF	5
JMP	addr	80-8F	5		JNC	addr	D0-DF	5
CALL	addr	90-9F	10		JACC	addr	E0-EF	7
JZ	addr	A0-AF	5		INDEX	addr	F0-FF	14
JNZ	addr	B0-BF	5					

5.0 Programming Model

5.1 14-Bit Program Counter (PC)

The 14-bit program counter (PC) allows access to up to 8 KB of PROM available with the CY7C64x13 architecture. The top 32 bytes of the ROM in the 8K part are reserved for testing purposes. The program counter is cleared during reset, such that the first instruction executed after a reset is at address 0x0000h. Typically, this is a jump instruction to a reset handler that initializes the application (see Interrupt Vectors on page 27).

The lower eight bits of the program counter are incremented as instructions are loaded and executed. The upper six bits of the program counter are incremented by executing an XPAGE instruction. As a result, the last instruction executed within a 256-byte “page” of sequential code should be an XPAGE instruction. The assembler directive “XPAGEON” causes the assembler to insert XPAGE instructions automatically. Because instructions can be either one or two bytes long, the assembler may occasionally need to insert a NOP followed by an XPAGE to execute correctly.

The address of the next instruction to be executed, the carry flag, and the zero flag are saved as two bytes on the program stack during an interrupt acknowledge or a CALL instruction. The program counter, carry flag, and zero flag are restored from the program stack during a RETI instruction. Only the program counter is restored during a RET instruction.

The program counter cannot be accessed directly by the firmware. The program stack can be examined by reading SRAM from location 0x00 and up.

5.1.1 Program Memory Organization

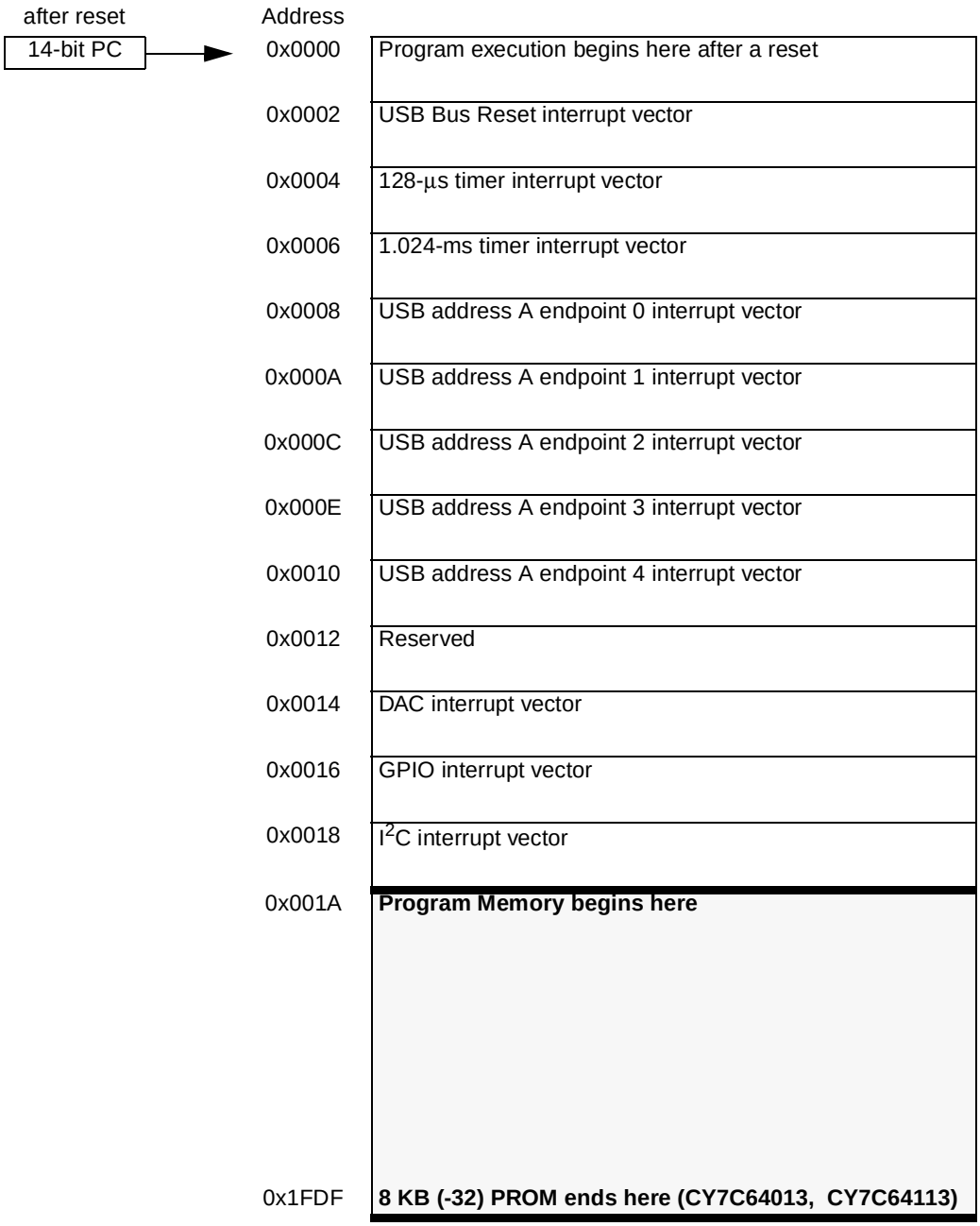


Figure 5-1. Program Memory Space with Interrupt Vector Table

5.2 8-Bit Accumulator (A)

The accumulator is the general-purpose register for the microcontroller.

5.3 8-Bit Temporary Register (X)

The “X” register is available to the firmware for temporary storage of intermediate results. The microcontroller can perform indexed operations based on the value in X. Refer to Section 5.6.3 for additional information.

5.4 8-Bit Program Stack Pointer (PSP)

During a reset, the program stack pointer (PSP) is set to 0x00 and “grows” upward from this address. The PSP may be set by firmware, using the MOV PSP,A instruction. The PSP supports interrupt service under hardware control and CALL, RET, and RETI instructions under firmware control. The PSP is not readable by the firmware.

During an interrupt acknowledge, interrupts are disabled and the 14-bit program counter, carry flag, and zero flag are written as two bytes of data memory. The first byte is stored in the memory addressed by the PSP, then the PSP is incremented. The second byte is stored in memory addressed by the PSP, and the PSP is incremented again. The overall effect is to store the program counter and flags on the program “stack” and increment the PSP by two.

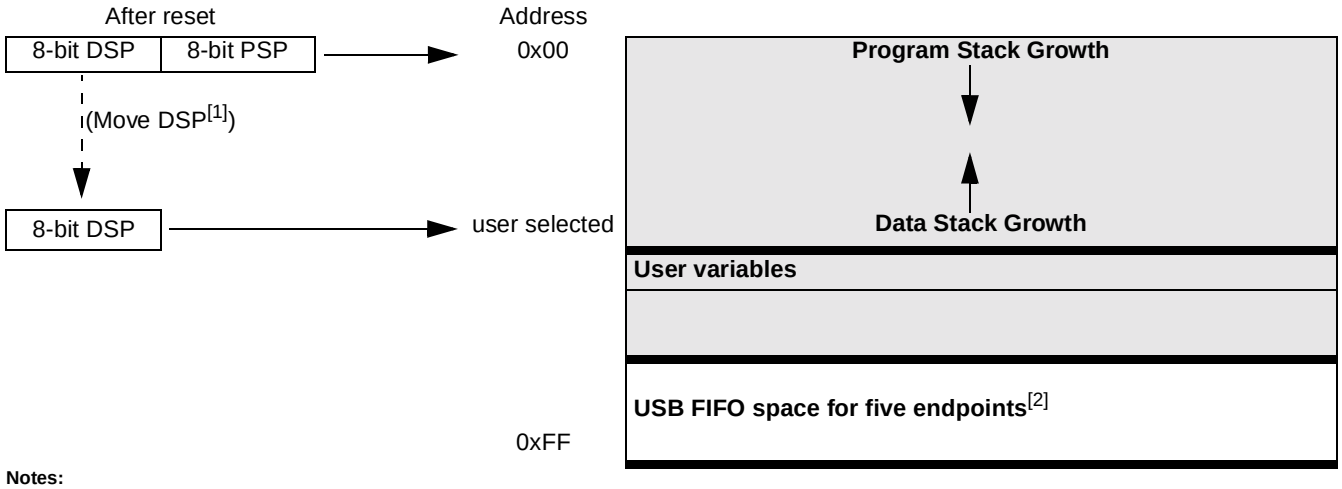
The return from interrupt (RETI) instruction decrements the PSP, then restores the second byte from memory addressed by the PSP. The PSP is decremented again and the first byte is restored from memory addressed by the PSP. After the program counter and flags have been restored from stack, the interrupts are enabled. The overall effect is to restore the program counter and flags from the program stack, decrement the PSP by two, and re-enable interrupts.

The call subroutine (CALL) instruction stores the program counter and flags on the program stack and increments the PSP by two.

The return from subroutine (RET) instruction restores the program counter but not the flags from the program stack and decrements the PSP by two.

5.4.1 Data Memory Organization

The CY7C64x13 microcontrollers provide 256 bytes of data RAM. Normally, the SRAM is partitioned into four areas: program stack, user variables, data stack, and USB endpoint FIFOs. The following is one example of where the program stack, data stack, and user variables areas could be located.



Notes:
1. Refer to Section 5.5 for a description of DSP.
2. Endpoint sizes are fixed by the Endpoint Size Bit (I/O register 0x1F, Bit 7), see *Table 18-1*.

5.5 8-Bit Data Stack Pointer (DSP)

The data stack pointer (DSP) supports PUSH and POP instructions that use the data stack for temporary storage. A PUSH instruction pre-decrements the DSP, then writes data to the memory location addressed by the DSP. A POP instruction reads data from the memory location addressed by the DSP, then post-increments the DSP.

During a reset, the DSP is reset to 0x00. A PUSH instruction when DSP equals 0x00 writes data at the top of the data RAM (address 0xFF). This writes data to the memory area reserved for USB endpoint FIFOs. Therefore, the DSP should be indexed at an appropriate memory location that does not compromise the Program Stack, user-defined memory (variables), or the USB endpoint FIFOs.

For USB applications, the firmware should set the DSP to an appropriate location to avoid a memory conflict with RAM dedicated to USB FIFOs. The memory requirements for the USB endpoints are described in Section 18.2. Example assembly instructions to do this with two device addresses (FIFOs begin at 0xD8) are shown below:

```
MOV A,20h     ; Move 20 hex into Accumulator (must be D8h or less)
SWAP A,DSP    ; swap accumulator value into DSP register
```

5.6 Address Modes

The CY7C64013 and CY7C64113 microcontrollers support three addressing modes for instructions that require data operands: data, direct, and indexed.

5.6.1 Data (Immediate)

“Data” address mode refers to a data operand that is actually a constant encoded in the instruction. As an example, consider the instruction that loads A with the constant 0xD8:

- MOV A,0D8h

This instruction requires two bytes of code where the first byte identifies the “MOV A” instruction with a data operand as the second byte. The second byte of the instruction is the constant “0xD8”. A constant may be referred to by name if a prior “EQU” statement assigns the constant value to the name. For example, the following code is equivalent to the example shown above:

- DSPINIT: EQU 0D8h
- MOV A,DSPINIT

5.6.2 Direct

“Direct” address mode is used when the data operand is a variable stored in SRAM. In that case, the one byte address of the variable is encoded in the instruction. As an example, consider an instruction that loads A with the contents of memory address location 0x10:

- MOV A,[10h]

Normally, variable names are assigned to variable addresses using “EQU” statements to improve the readability of the assembler source code. As an example, the following code is equivalent to the example shown above:

- buttons: EQU 10h
- MOV A,[buttons]

5.6.3 Indexed

“Indexed” address mode allows the firmware to manipulate arrays of data stored in SRAM. The address of the data operand is the sum of a constant encoded in the instruction and the contents of the “X” register. Normally, the constant is the “base” address of an array of data and the X register contains an index that indicates which element of the array is actually addressed:

- array: EQU 10h
- MOV X,3
- MOV A,[X+array]

This would have the effect of loading A with the fourth element of the SRAM “array” that begins at address 0x10. The fourth element would be at address 0x13.

6.0 Clocking

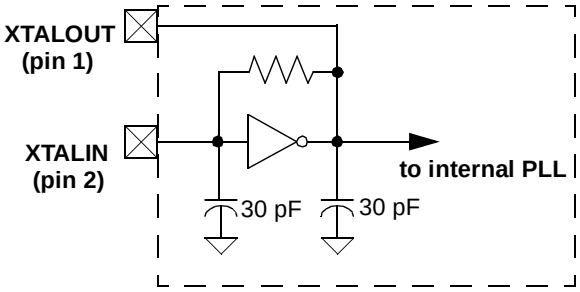


Figure 6-1. Clock Oscillator On-Chip Circuit

The XTALIN and XTALOUT are the clock pins to the microcontroller. The user can connect an external oscillator or a crystal to these pins. When using an external crystal, keep PCB traces between the chip leads and crystal as short as possible (less than 2 cm). A 6-MHz fundamental crystal can be connected to these pins to provide a reference frequency for the internal PLL. A ceramic resonator does not allow the microcontroller to meet the timing specifications of a full speed USB and therefore a ceramic resonator is not recommended with these parts.

An external 6-MHz clock can be applied to the XTALIN pin if the XTALOUT pin is left open. Grounding the XTALOUT pin when driving XTALIN with an oscillator does not work because the internal clock is effectively shorted to ground.

7.0 Reset

The CY7C64x13 supports two resets: Power-On Reset (POR) and a Watch Dog Reset (WDR). Each of these resets causes:

- all registers to be restored to their default states,
- the USB Device Address to be set to 0,
- all interrupts to be disabled,
- the PSP and Data Stack Pointer (DSP) to be set to memory address 0x00.

The occurrence of a reset is recorded in the Processor Status and Control Register, as described in Section 15.0. Bits 4 and 6 are used to record the occurrence of POR and WDR, respectively. Firmware can interrogate these bits to determine the cause of a reset.

Program execution starts at ROM address 0x0000 after a reset. Although this looks like interrupt vector 0, there is an important difference. Reset processing does NOT push the program counter, carry flag, and zero flag onto program stack. The firmware reset handler should configure the hardware before the “main” loop of code. Attempting to execute a RET or RETI in the firmware reset handler causes unpredictable execution results.

7.1 Power-On Reset (POR)

When V_{CC} is first applied to the chip, the Power-On Reset (POR) signal is asserted and the CY7C64x13 enters a “semi-suspend” state. During the semi-suspend state, which is different from the suspend state defined in the USB specification, the oscillator and all other blocks of the part are functional, except for the CPU. This semi-suspend time ensures that both a valid V_{CC} level is reached and that the internal PLL has time to stabilize before full operation begins. When the V_{CC} has risen above approximately 2.5V, and the oscillator is stable, the POR is deasserted and the on-chip timer starts counting. The first 1 ms of suspend time is not interruptible, and the semi-suspend state continues for an additional 95 ms unless the count is bypassed by a USB Bus Reset on the upstream port. The 95 ms provides time for V_{CC} to stabilize at a valid operating voltage before the chip executes code.

If a USB Bus Reset occurs on the upstream port during the 95-ms semi-suspend time, the semi-suspend state is aborted and program execution begins immediately from address 0x0000. In this case, the Bus Reset interrupt is pending but not serviced until firmware sets the USB Bus Reset Interrupt Enable bit (bit 0 of register 0x20) and enables interrupts with the EI command.

The POR signal is asserted whenever V_{CC} drops below approximately 2.5V, and remains asserted until V_{CC} rises above this level again. Behavior is the same as described above.

7.2 Watch Dog Reset (WDR)

The Watch Dog Timer Reset (WDR) occurs when the internal Watch Dog timer rolls over. Writing any value to the write-only Watch Dog Restart Register at address 0x26 clears the timer. The timer rolls over and WDR occurs if it is not cleared within t_{WATCH} (8 ms minimum) of the last clear. Bit 6 of the Processor Status and Control Register is set to record this event (the register contents are set to 010X0001 by the WDR). A Watch Dog Timer Reset lasts for 2 ms, after which the microcontroller begins execution at ROM address 0x0000.

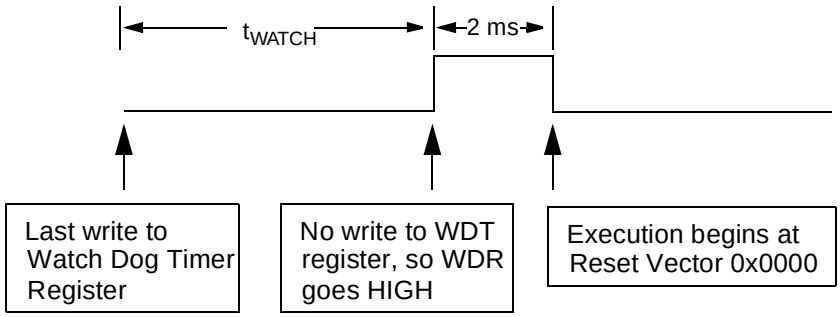


Figure 7-1. Watch Dog Reset (WDR)

The USB transmitter is disabled by a Watch Dog Reset because the USB Device Address Register is cleared (see Section 18.1). Otherwise, the USB Controller would respond to all address 0 transactions.

It is possible for the WDR bit of the Processor Status and Control Register (0xFF) to be set following a POR event. The WDR bit should be ignored If the firmware interrogates the Processor Status and Control Register for a Set condition on the WDR bit and if the POR (bit 3 of register 0xFF) bit is set.

8.0 Suspend Mode

The CY7C64x13 can be placed into a low-power state by setting the Suspend bit of the Processor Status and Control register. All logic blocks in the device are turned off except the GPIO interrupt logic and the USB receiver. The clock oscillator and PLL, as well as the free-running and Watch Dog timers, are shut down. Only the occurrence of an enabled GPIO interrupt or non-idle bus activity at a USB upstream or downstream port wakes the part out of suspend. The Run bit in the Processor Status and Control Register must be set to resume a part out of suspend.

The clock oscillator restarts immediately after exiting suspend mode. The microcontroller returns to a fully functional state 1 ms after the oscillator is stable. The microcontroller executes the instruction following the I/O write that placed the device into suspend mode before servicing any interrupt requests.

The GPIO interrupt allows the controller to wake-up periodically and poll system components while maintaining a very low average power consumption. To achieve the lowest possible current during suspend mode, all I/O should be held at V_{CC} or Gnd. This also applies to internal port pins that may not be bonded in a particular package.

Typical code for entering suspend is shown below:

```
...             ; All GPIO set to low-power state (no floating pins)
...             ; Enable GPIO interrupts if desired for wake-up
mov a, 09h      ; Set suspend and run bits
iowr FFh        ; Write to Status and Control Register - Enter suspend, wait for USB activity (or GPIO Interrupt)
nop             ; This executes before any ISR
...             ; Remaining code for exiting suspend routine
```

9.0 General-Purpose I/O (GPIO) Ports

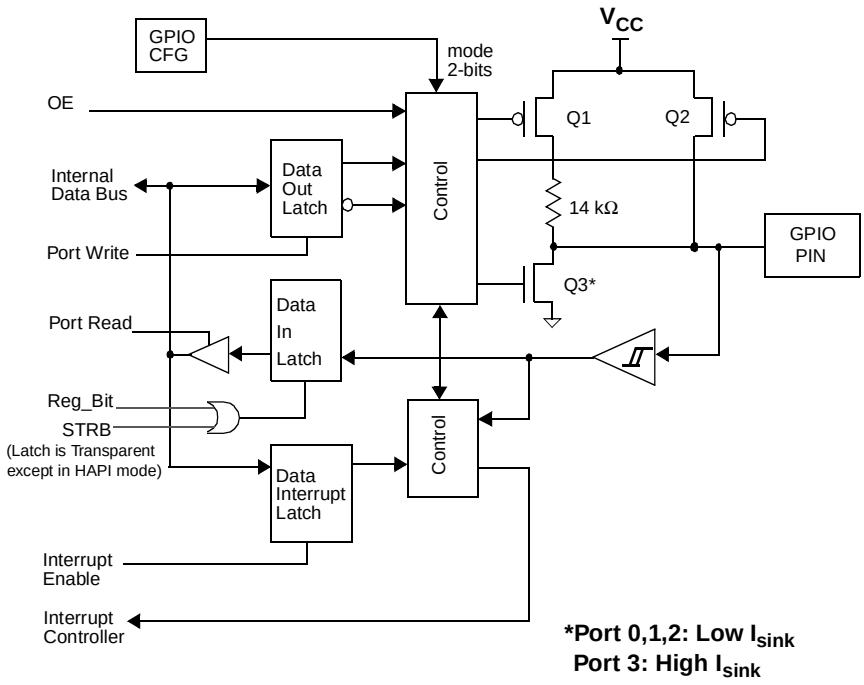


Figure 9-1. Block Diagram of a GPIO Pin

There are up to 32 GPIO pins (P0[7:0], P1[7:0], P2[7:0], and P3[7:0]) for the hardware interface. The number of GPIO pins changes based on the package type of the chip. Each port can be configured as inputs with internal pull-ups, open drain outputs, or traditional CMOS outputs. Port 3 offers a higher current drive, with typical current sink capability of 12 mA. The data for each GPIO port is accessible through the data registers. Port data registers are shown in *Figure 9-2* through *Figure 9-5*, and are set to 1 on reset.

7	6	5	4	3	2	1	0
P0[7]	P0[6]	P0[5]	P0[4]	P0[3]	P0[2]	P0[1]	P0[0]

Figure 9-2. Port 0 Data 0x00 (read/write)

7	6	5	4	3	2	1	0
P1[7]	P1[6]	P1[5]	P1[4]	P1[3]	P1[2]	P1[1]	P1[0]

Figure 9-3. Port 1 Data 0x01 (read/write)

7	6	5	4	3	2	1	0
P2[7]	P2[6]	P2[5]	P2[4]	P2[3]	P2[2]	P2[1]	P2[0]

Figure 9-4. Port 2 Data 0x02 (read/write)

7	6	5	4	3	2	1	0
P3[7] (see text)	P3[6]	P3[5]	P3[4]	P3[3]	P3[2]	P3[1]	P3[0]

Figure 9-5. Port 3 Data 0x03 (read/write)

Special care should be taken with any unused GPIO data bits. An unused GPIO data bit, either a pin on the chip or a port bit that is not bonded on a particular package, must not be left floating when the device enters the suspend state. If a GPIO data bit is left floating, the leakage current caused by the floating bit may violate the suspend current limitation specified by the USB Specifications. If a '1' is written to the unused data bit and the port is configured with open drain outputs, the unused data bit remains in an indeterminate state. Therefore, if an unused port bit is programmed in open-drain mode, it must be written with a '0.' Notice that the CY7C64013 part always requires that the data bits P1[7:3], P2[7,1,0], and P3[7:3] be written with a '0.'

In normal non-HAPI mode, reads from a GPIO port always return the present state of the voltage at the pin, independent of the settings in the Port Data Registers. If HAPI mode is activated for a port, reads of that port return latched data as controlled by the HAPI signals (see Section 14.0). During reset, all of the GPIO pins are set to a high-impedance input state ('1' in open drain mode). Writing a '0' to a GPIO pin drives the pin LOW. In this state, a '0' is always read on that GPIO pin unless an external source overdrives the internal pull-down device.

9.1 GPIO Configuration Port

Every GPIO port can be programmed as inputs with internal pull-ups, open drain outputs, and traditional CMOS outputs. In addition, the interrupt polarity for each port can be programmed. With positive interrupt polarity, a rising edge ('0' to '1') on an input pin causes an interrupt. With negative polarity, a falling edge ('1' to '0') on an input pin causes an interrupt. As shown in the table below, when a GPIO port is configured with CMOS outputs, interrupts from that port are disabled. The GPIO Configuration Port register provides two bits per port to program these features. The possible port configurations are detailed in *Table 9-1*:

Table 9-1. Port Configurations

Port Configuration bits	Pin Interrupt Bit	Driver Mode	Interrupt Polarity
11	0	Resistive	Disabled
	1	Resistive	–
10	0	CMOS Output	Disabled
	1	Open Drain	Disabled
01	0	Open Drain	Disabled
	1	Open Drain	–
00 (Reset State)	0	Open Drain	Disabled (Default Condition)
	1	Open Drain	+

In "Resistive" mode, a 14-kΩ pull-up resistor is conditionally enabled for all pins of a GPIO port. An I/O pin is driven HIGH through a 14-kΩ pull-up resistor when a '1' has been written to the pin. The output pin is driven LOW with the pull-up disabled when a '0' has been written to the pin. An I/O pin that has been written as a '1' can be used as an input pin with the integrated 14-kΩ pull-up resistor. Resistive mode selects a negative (falling edge) interrupt polarity on all pins that have the GPIO interrupt enabled.

In “CMOS” mode, all pins of the GPIO port are outputs that are actively driven. A CMOS port is not a possible source for interrupts.

In “Open Drain” mode, the internal pull-up resistor and CMOS driver (HIGH) are both disabled. An open drain I/O pin that has been written as a ‘1’ can be used as an input or an open drain output. An I/O pin that has been written as a ‘0’ drives the output low. The interrupt polarity for an open drain GPIO port can be selected as positive (rising edge) or negative (falling edge).

During reset, all of the bits in the GPIO Configuration Register are written with ‘0’ to select Open Drain output for all GPIO ports as the default configuration.

7	6	5	4	3	2	1	0
Port 3 Config Bit 1	Port 3 Config Bit 0	Port 2 Config Bit 1	Port 2 Config Bit 0	Port 1 Config Bit 1	Port 1 Config Bit 0	Port 0 Config Bit 1	Port 0 Config Bit 0

Figure 9-6. GPIO Configuration Register 0x08 (read/write)

9.2 GPIO Interrupt Enable Ports

Each GPIO pin can be individually enabled or disabled as an interrupt source. The Port 0–3 Interrupt Enable registers provide this feature with an interrupt enable bit for each GPIO pin. When HAPI mode (discussed in Section 14.0) is enabled the GPIO interrupts are blocked, including ports not used by HAPI, so GPIO pins cannot be used as interrupt sources.

During a reset, GPIO interrupts are disabled by clearing all of the GPIO interrupt enable ports. Writing a ‘1’ to a GPIO Interrupt Enable bit enables GPIO interrupts from the corresponding input pin. All GPIO pins share a common interrupt, as discussed in Section 16.7.

7	6	5	4	3	2	1	0
P0[7]	P0[6]	P0[5]	P0[4]	P0[3]	P0[2]	P0[1]	P0[0]

Figure 9-7. Port 0 Interrupt Enable 0x04 (read/write)

7	6	5	4	3	2	1	0
P1[7]	P1[6]	P1[5]	P1[4]	P1[3]	P1[2]	P1[1]	P1[0]

Figure 9-8. Port 1 Interrupt Enable 0x05 (read/write)

7	6	5	4	3	2	1	0
P2[7]	P2[6]	P2[5]	P2[4]	P2[3]	P2[2]	P2[1]	P2[0]

Figure 9-9. Port 2 Interrupt Enable 0x06 (read/write)

7	6	5	4	3	2	1	0
reserved - set to zero	P3[6]	P3[5]	P3[4]	P3[3]	P3[2]	P3[1]	P3[0]

Figure 9-10. Port 3 Interrupt Enable 0x07 (read/write)

10.0 DAC Port

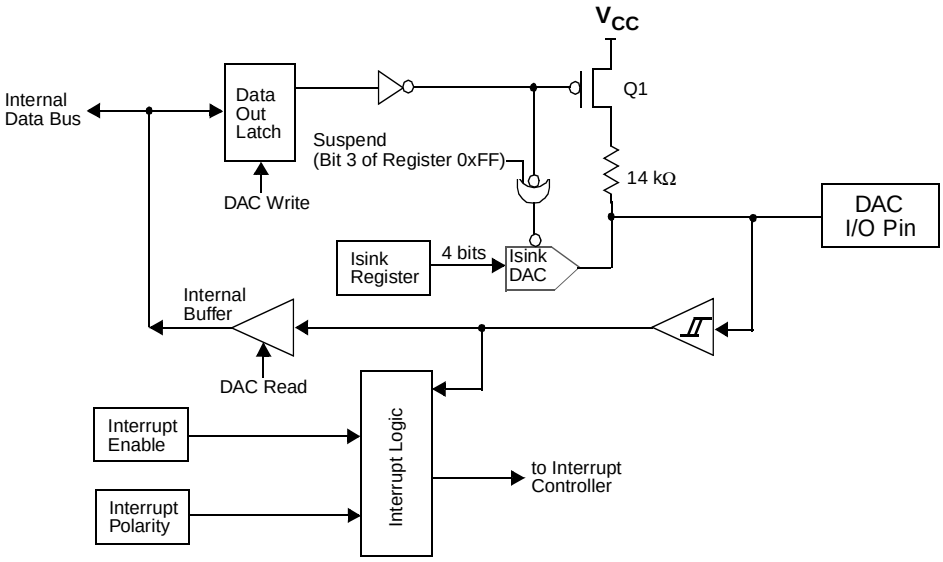


Figure 10-1. Block Diagram of a DAC Pin

The CY7C64113 features a Digital to Analog Conversion (DAC) port which has programmable current sink on each I/O pin. Writing a '1' to a DAC I/O pin disables the output current sink (Isink DAC) and drives the I/O pin HIGH through an integrated 14-kΩ resistor. When a '0' is written to a DAC I/O pin, the Isink DAC is enabled and the pull-up resistor is disabled. This causes the Isink DAC to sink current to drive the output LOW. The amount of sink current for the DAC I/O pin is programmable over 16 values based on the contents of the DAC Isink Register for that output pin. DAC[1:0] are high-current outputs that are programmable from 3.2 mA to 16 mA (typical). DAC[7:2] are low-current outputs, programmable from 0.2 mA to 1.0 mA (typical).

When the suspend bit in Processor Status and Control Register (0xFF) is set, the Isink DAC block of the DAC circuitry is disabled. Special care should be taken when the CY7C64x13 device is placed in the suspend mode. The DAC Port Data Register(0x30) should normally be loaded with all '1's (0xFF) before setting the suspend bit. If any of the DAC bits are set to '0' when the device is suspended, that DAC input will float. The floating pin could result in excessive current consumption by the device, unless an external load places the pin in a deterministic state.

When a DAC I/O bit is written as a '1', the I/O pin is an output pulled HIGH through the 14-kΩ resistor or an input with an internal 14-kΩ pull-up resistor. All DAC port data bits are set to '1' during reset.

Low current outputs 0.2 mA to 1.0 mA typical						High current outputs 3.2 mA to 16 mA typical	
7	6	5	4	3	2	1	0
DAC[7]	DAC[6]	DAC[5]	DAC[4]	DAC[3]	DAC[2]	DAC[1]	DAC[0]

Figure 10-2. DAC Port Data 0x30 (read/write)

10.1 DAC Isink Registers

Each DAC I/O pin has an associated DAC Isink register to program the output sink current when the output is driven LOW. The first Isink register (0x38) controls the current for DAC[0], the second (0x39) for DAC[1], and so on until the Isink register at 0x3F controls the current to DAC[7]. Writing all '0's to the Isink register causes 1/5 of the max. current to flow through the DAC I/O pin. Writing all '1's to the Isink register provides the maximum current flow through the pin. The other 14 states of the DAC sink current are evenly spaced between these two values.

				Isink Value			
7	6	5	4	3	2	1	0
reserved	reserved	reserved	reserved	Isink[3]	Isink[2]	Isink[1]	Isink[0]

Figure 10-3. DAC Port Isink 0x38 to 0x3F (write only)

10.2 DAC Port Interrupts

A DAC port interrupt can be enabled/disabled for each pin individually. The DAC Port Interrupt Enable register provides this feature with an interrupt enable bit for each DAC I/O pin. Writing a '1' to a bit in this register enables interrupts from the corresponding bit position. Writing a '0' to a bit in the DAC Port Interrupt Enable register disables interrupts from the corresponding bit position. All of the DAC Port Interrupt Enable register bits are cleared to '0' during a reset. All DAC pins share a common interrupt, as explained in Section 16.6.

7	6	5	4	3	2	1	0
DAC[7]	DAC[6]	DAC[5]	DAC[4]	DAC[3]	DAC[2]	DAC[1]	DAC[0]

Figure 10-4. DAC Port Interrupt Enable 0x31 (write only)

As an additional benefit, the interrupt polarity for each DAC pin is programmable with the DAC Port Interrupt Polarity register. Writing a '0' to a bit selects negative polarity (falling edge) that causes an interrupt (if enabled) if a falling edge transition occurs on the corresponding input pin. Writing a '1' to a bit in this register selects positive polarity (rising edge) that causes an interrupt (if enabled) if a rising edge transition occurs on the corresponding input pin. All of the DAC Port Interrupt Polarity register bits are cleared during a reset.

7	6	5	4	3	2	1	0
DAC[7]	DAC[6]	DAC[5]	DAC[4]	DAC[3]	DAC[2]	DAC[1]	DAC[0]

Figure 10-5. DAC Port Interrupt Polarity 0x32 (write only)

11.0 12-Bit Free-Running Timer

The 12-bit timer provides two interrupts (128-µs and 1.024-ms) and allows the firmware to directly time events that are up to 4 ms in duration. The lower 8 bits of the timer can be read directly by the firmware. Reading the lower 8 bits latches the upper 4 bits into a temporary register. When the firmware reads the upper 4 bits of the timer, it is accessing the count stored in the temporary register. The effect of this logic is to ensure a stable 12-bit timer value can be read, even when the two reads are separated in time.

11.1 Timer (LSB)

7	6	5	4	3	2	1	0
Timer Bit 7	Timer Bit 6	Timer Bit 5	Timer Bit 4	Timer Bit 3	Timer Bit 2	Timer Bit 1	Timer Bit 0

Figure 11-1. Timer Register 0x24 (read only)

11.2 Timer (MSB)

7	6	5	4	3	2	1	0
Reserved	Reserved	Reserved	Reserved	Timer Bit 11	Timer Bit 10	Timer Bit 9	Timer Bit 8

Figure 11-2. Timer Register 0x25 (read only)

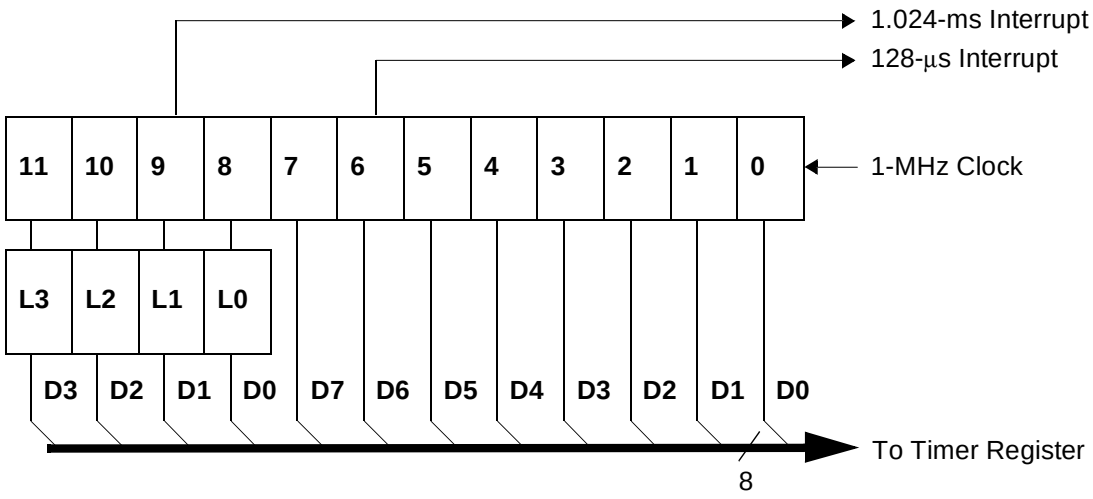


Figure 11-3. Timer Block Diagram

12.0 I²C and HAPI Configuration Register

Internal hardware supports communication with external devices through two interfaces: a two-wire I²C, and a HAPI for 1, 2, or 3 byte transfers. The I²C and HAPI functions, discussed in detail in Sections 13.0 and 14.0, share a common configuration register (see *Figure 12-1*). All bits of this register are cleared on reset.

7	6	5	4	3	2	1	0
R/W		R/W	R/W	R	R	R/W	R/W
I²C Position	Reserved	LEMPTY Polarity	DRDY Polarity	Latch Empty	Data Ready	HAPI Port Width Bit 1	HAPI Port Width Bit 0

Figure 12-1. HAPI/I²C Configuration Register 0x09 (read/write)

Bits [7,1:0] of the HAPI/I²C Configuration Register control the pin out configuration of the HAPI and I²C interfaces. Bits [5:2] are used in HAPI mode only, and are described in Section 14.0. *Table 12-1* shows the HAPI port configurations, and *Table 12-2* shows I²C pin location configuration options. These I²C options exist due to pin limitations in certain packages, and to allow simultaneous HAPI and I²C operation.

HAPI operation is enabled whenever either HAPI Port Width Bit (Bit 1 or 0) is non-zero. This affects GPIO operation as described in Section 14.0. I²C must be separately enabled as described in Section 13.0.

Table 12-1. HAPI Port Configuration

Port Width Bits[1:0]	HAPI Port Width
11	24 Bits: P3[7:0], P1[7:0], P0[7:0]
10	16 Bits: P1[7:0], P0[7:0]
01	8 Bits: P0[7:0]
00	No HAPI Interface

Table 12-2. I²C Port Configuration

I²C Position Bit[7]	Port Width Bit[1]	I²C Position
X	1	I²C on P2[1:0], 0:SCL, 1:SDA
0	0	I²C on P1[1:0], 0:SCL, 1:SDA
1	0	I²C on P2[1:0], 0:SCL, 1:SDA

13.0 I²C Controller

The I²C block provides a versatile two-wire communication with external devices, supporting master, slave, and multi-master modes of operation. The I²C block functions by handling the low-level signaling in hardware, and issuing interrupts as needed to allow firmware to take appropriate action during transactions. While waiting for firmware response, the hardware keeps the I²C bus idle if necessary.

The I²C generates an interrupt to the microcontroller at the end of each received or transmitted byte, when a stop bit is detected by the slave when in receive mode, or when arbitration is lost. Details of the interrupt responses are given in Section 16.8.

The I²C interface consists of two registers, an I²C Data Register (*Figure 13-1*) and an I²C Status and Control Register (*Figure 13-2*). The Data Register is implemented as separate read and write registers. Generally, the I²C Status and Control Register should only be monitored after the I²C interrupt, as all bits are valid at that time. Polling this register at other times could read misleading bit status if a transaction is underway.

The I²C SCL clock is connected to bit 0 of GPIO port 1 or GPIO port 2, and the I²C SDA data is connected to bit 1 of GPIO port 1 or GPIO port 2. Refer to Section 12.0 for the bit definitions and functionality of the HAPI/I²C Configuration Register, which is used to set the locations of the configurable I²C pins. Once the I²C functionality is enabled by setting bit 0 of the I²C Status & Control Register, the two LSB bits ([1:0]) of the corresponding GPIO port are placed in Open Drain mode, regardless of the settings of the GPIO Configuration Register.

All control of the I²C clock and data lines is performed by the I²C block.

7	6	5	4	3	2	1	0
I²C Data 7	I²C Data 6	I²C Data 5	I²C Data 4	I²C Data 3	I²C Data 2	I²C Data 1	I²C Data 0

Figure 13-1. I²C Data Register 0x29 (separate read/write registers)

7	6	5	4	3	2	1	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
MSTR Mode	Continue/ Busy	Xmit Mode	ACK	Addr	ARB Lost/ Restart	Received Stop	I²C Enable

Figure 13-2. I²C Status and Control Register 0x28 (read/write)

The I²C Status and Control register bits are defined in *Table 13-1*, with a more detailed description following.

Table 13-1. I²C Status and Control Register Bit Definitions

Bit	Name	Description
0	I²C Enable	Write to 1 to enable I²C function. When cleared, I²C GPIO pins operate normally.
1	Received Stop	Reads 1 only in slave receive mode, when I²C Stop bit detected (unless firmware did not ACK the last transaction).
2	ARB Lost/Restart	Reads 1 to indicate master has lost arbitration. Reads 0 otherwise. Write to 1 in master mode to perform a restart sequence (also set Continue bit).
3	Addr	Reads 1 during first byte after start/restart in slave mode, or if master loses arbitration. Reads 0 otherwise. This bit should always be written as 0.
4	ACK	In receive mode, write 1 to generate ACK, 0 for no ACK. In transmit mode, reads 1 if ACK was received, 0 if no ACK received.
5	Xmit Mode	Write to 1 for transmit mode, 0 for receive mode.
6	Continue / Busy	Write 1 to indicate ready for next transaction. Reads 1 when I²C block is busy with a transaction, 0 when transaction is complete.
7	MSTR Mode	Write to 1 for master mode, 0 for slave mode. This bit is cleared if master loses arbitration. Clearing from 1 to 0 generates Stop bit.

MSTR Mode: Setting this bit causes the I²C to initiate a master mode transaction by sending a start bit and transmitting the first data byte from the data register (this typically holds the target address and R/W bit). Subsequent bytes are initiated by setting the Continue bit, as described below.

In master mode, the I²C block generates the clock (SCK), and drives the data line as required depending on transmit or receive state. The I²C block performs any required arbitration and clock synchronization. The loss of arbitration results in the clearing of this bit, the setting of the ARB Lost bit, and the generation of an interrupt to the microcontroller. If the chip is the target of an external master that wins arbitration, then the interrupt is held off until the transaction from the external master is completed.

When MSTR Mode is cleared from 1 to 0 by a firmware write, an I²C Stop bit is generated.

Continue / Busy: This bit is written by the firmware to indicate that the firmware is ready for the next byte transaction to begin. In other words, the bit has responded to an interrupt request and has completed the required update or read of the data register. During a read this bit indicates if the hardware is busy and is locking out additional writes to the I²C Status and Control register. This locking allows the hardware to complete certain operations that may require an extended period of time. Following an I²C interrupt, the I²C block does not return to the Busy state until firmware sets the Continue bit. This allows the firmware to make one control register write without the need to check the Busy bit.

Xmit Mode: This bit is set by firmware to enter transmit mode and perform a data transmit in master or slave mode. Clear this bit for receive mode. Firmware generally determines the value of this bit from the R/W bit associated with the I²C address packet. The Xmit Mode bit state is ignored when initially writing the MSTR Mode or the Restart bits, as these cases always cause transmit mode for the first byte.

ACK: This bit is set or cleared by firmware during receive operation to indicate if the hardware should generate an ACK signal on the I²C bus. Writing a 1 to this bit generates an ACK (SDA LOW) on the I²C bus at the ACK bit time. During transmits (Xmit Mode=1), this bit should be cleared.

Addr: This bit is set by the I²C block during the first byte of a slave receive transaction, after an I²C start or restart. The Addr bit is cleared when the firmware sets the Continue bit. This bit allows the firmware to recognize when the master has lost arbitration, and in slave mode it allows the firmware to recognize that a start or restart has occurred.

ARB Lost/Restart: This bit is valid as a status bit (ARB Lost) after master mode transactions. In master mode, set this bit (along with the Continue and MSTR Mode bits) to perform an I²C restart sequence. The I²C target address for the restart must be written to the data register before setting the Continue bit. To prevent false ARB Lost signals, the Restart bit is cleared by hardware during the restart sequence.

Receive Stop: This bit is set when the slave is in receive mode and detects a stop bit on the bus. The Receive Stop bit is not set if the firmware terminates the I²C transaction by not acknowledging the previous byte transmitted on the I²C bus, e.g., in receive mode if firmware sets the Continue bit and clears the ACK bit.

I²C Enable: Set this bit to override GPIO definition with I²C function on the two I²C pins. When this bit is cleared, these pins are free to function as GPIOs. In I²C mode, the two pins operate in open drain mode, independent of the GPIO configuration setting.

14.0 Hardware Assisted Parallel Interface (HAPI)

The CY7C64x13 processor provides a hardware assisted parallel interface for bus widths of 8, 16, or 24 bits, to accommodate data transfer with an external microcontroller or similar device. Control bits for selecting the byte width are in the HAPI/I²C Configuration Register (*Figure 12-1*), bits 1 and 0.

Signals are provided on Port 2 to control the HAPI interface. *Table 14-1* describes these signals and the HAPI control bits in the HAPI/I²C Configuration Register. Enabling HAPI causes the GPIO setting in the GPIO Configuration Register (0x08) to be overridden. The Port 2 output pins are in CMOS output mode and Port 2 input pins are in input mode (open drain mode with Q3 OFF in *Figure 9-1*).

Table 14-1. Port 2 Pin and HAPI Configuration Bit Definitions

Pin	Name	Direction	Description (Port 2 Pin)
P2[2]	LatEmptyPin	Out	Ready for more input data from external interface
P2[3]	DReadyPin	Out	Output data ready for external interface
P2[4]	$\overline{STB}$	In	Strobe signal for latching incoming data
P2[5]	$\overline{OE}$	In	Output Enable, causes chip to output data
P2[6]	$\overline{CS}$	In	Chip Select (Gates $\overline{STB}$ and $\overline{OE}$)
Bit	**Name**	**R/W**	**Description (HAPI/I^{2}C Configuration Register)**
2	Data Ready	R	Asserted after firmware writes data to Port 0, until $\overline{OE}$ driven LOW.
3	Latch Empty	R	Asserted after firmware reads data from Port 0, until $\overline{STB}$ driven LOW.
4	DRDY Polarity	R/W	Determines polarity of Data Ready bit and DReadyPin: If 0, Data Ready is active LOW, DReadyPin is active HIGH. If 1, Data Ready is active HIGH, DReadyPin is active LOW.
5	LEMPTY Polarity	R/W	Determines polarity of Latch Empty bit and LatEmptyPin: If 0, Latch Empty is active LOW, LatEmptyPin is active HIGH. If 1, Latch Empty is active HIGH, LatEmptyPin is active LOW.

HAPI Read by External Device from CY7C64x13: In this case (see *Figure 22-3*), firmware writes data to the GPIO ports. If 16-bit or 24-bit transfers are being made, Port 0 should be written last, since writes to Port 0 asserts the Data Ready bit and the DReadyPin to signal the external device that data is available.

The external device then drives the $\overline{OE}$ and $\overline{CS}$ pins active (LOW), which causes the HAPI data to be output on the port pins. When $\overline{OE}$ is returned HIGH (inactive), the HAPI/GPIO interrupt is generated. At that point, firmware can reload the HAPI latches for the next output, again writing Port 0 last.

The Data Ready bit reads the opposite state from the external DReadyPin on pin P2[3]. If the DRDY Polarity bit is 0, DReadyPin is active HIGH, and the Data Ready bit is active LOW.

HAPI Write by External Device to CY7C64x13: In this case (see *Figure 22-4*), the external device drives the $\overline{STB}$ and $\overline{CS}$ pins active (LOW) when it drives new data onto the port pins. When this happens, the internal latches become full which causes the Latch Empty bit to be deasserted. When $\overline{STB}$ is returned HIGH (inactive), the HAPI/GPIO interrupt is generated. Firmware then reads the parallel ports to empty the HAPI latches. If 16-bit or 24-bit transfers are being made, Port 0 should be read last because reads from Port 0 assert the Latch Empty bit and the LatEmptyPin to signal the external device for more data.

The Latch Empty bit reads the opposite state from the external LatEmptyPin on pin P2[2]. If the LEMPTY Polarity bit is 0, LatEmptyPin is active HIGH, and the Latch Empty bit is active LOW.

15.0 Processor Status and Control Register

7	6	5	4	3	2	1	0
R	R/W	R/W	R/W	R/W	R		R/W
IRQ Pending	Watch Dog Reset	USB Bus Reset Interrupt	Power-On Reset	Suspend	Interrupt Enable Sense	reserved	Run

Figure 15-1. Processor Status and Control Register 0xFF

The Run bit, bit 0, is manipulated by the HALT instruction. When Halt is executed, all the bits of the Processor Status and Control Register are cleared to 0. Since the run bit is cleared, the processor stops at the end of the current instruction. The processor remains halted until an appropriate reset occurs (power-on or watch dog). This bit should normally be written as a '1.'

Bit 1 is reserved and must be written as a zero.

The Interrupt Enable Sense (bit 2) shows whether interrupts are enabled or disabled. Firmware has no direct control over this bit as writing a zero or one to this bit position has no effect on interrupts. A '0' indicates that interrupts are masked off and a '1' indicates that the interrupts are enabled. This bit is further gated with the bit settings of the Global Interrupt Enable Register (0x20) and USB End Point Interrupt Enable Register (0x21). Instructions DI, EI, and RETI manipulate the state of this bit.

Writing a '1' to the Suspend bit (bit 3) halts the processor and causes the microcontroller to enter the suspend mode that significantly reduces power consumption. A pending, enabled interrupt or USB bus activity causes the device to come out of suspend. After coming out of suspend, the device resumes firmware execution at the instruction following the IOWR which put the part into suspend. An IOWR attempting to put the part into suspend is ignored if non-idle USB bus activity is present. See Section 8.0 for more details on suspend mode operation.

The Power-On Reset (bit 4) is set to '1' during a power-on reset. The firmware can check bits 4 and 6 in the reset handler to determine whether a reset was caused by a power-on condition or a watch dog timeout. Note that a POR event may be followed by a watch dog reset before firmware begins executing, as explained below.

The USB Bus Reset Interrupt (bit 5) occurs when a USB Bus Reset is received on the upstream port. The USB Bus Reset is a single-ended zero (SE0) that lasts from 12 to 16 µs. An SE0 is defined as the condition in which both the D+ line and the D– line are LOW at the same time. When the SIE detects that this SE0 condition is removed, the USB Bus Reset interrupt bit is set in the Processor Status and Control Register and a USB Bus Reset interrupt is generated.

The Watch Dog Reset (bit 6) is set during a reset initiated by the Watch Dog Timer. This indicates the Watch Dog Timer went for more than t_{WATCH} (8 ms minimum) between Watch Dog clears. This can occur with a POR event, as noted below.

The IRQ pending (bit 7), when set, indicates that one or more of the interrupts has been recognized as active. An interrupt remains pending until its interrupt enable bit is set (registers 0x20 or 0x21) and interrupts are globally enabled. At that point, the internal interrupt handling sequence clears this bit until another interrupt is detected as pending.

During power-up, the Processor Status and Control Register is set to 00010001, which indicates a POR (bit 4 set) has occurred and no interrupts are pending (bit 7 clear). During the 96 ms suspend at start-up (explained in Section 7.1), a Watch Dog Reset also occurs unless this suspend is aborted by an upstream SE0 before 8 ms. If a WDR occurs during the power-up suspend interval, firmware reads 01010001 from the Status and Control Register after power-up. Normally, the POR bit should be cleared so a subsequent WDR can be clearly identified. If an upstream bus reset is received before firmware examines this register, the Bus Reset bit may also be set.

During a Watch Dog Reset, the Processor Status and Control Register is set to 01XX0001, which indicates a Watch Dog Reset (bit 6 set) has occurred and no interrupts are pending (bit 7 clear). The Watch Dog Reset does not effect the state of the POR and the Bus Reset Interrupt bits.

16.0 Interrupts

Interrupts are generated by the GPIO/DAC pins, the internal timers, I^2C or HAPI operation, or on various USB traffic conditions. All interrupts are maskable by the Global Interrupt Enable Register and the USB End Point Interrupt Enable Register. Writing a '1' to a bit position enables the interrupt associated with that bit position. During a reset, the contents the Global Interrupt Enable Register and USB End Point Interrupt Enable Register are cleared, effectively disabling all interrupts.

7	6	5	4	3	2	1	0
	R/W	R/W	R/W		R/W	R/W	R/W
Reserved	I^2C Interrupt Enable	GPIO/HAPI Interrupt Enable	DAC Interrupt Enable	Reserved	1.024-ms Interrupt Enable	128-µs Interrupt Enable	USB Bus RST Interrupt Enable

Figure 16-1. Global Interrupt Enable Register 0x20 (read/write)

7	6	5	4	3	2	1	0
			R/W	R/W	R/W	R/W	R/W
Reserved	Reserved	Reserved	EPB1 Interrupt Enable	EPB0 Interrupt Enable	EPA2 Interrupt Enable	EPA1 Interrupt Enable	EPA0 Interrupt Enable

Figure 16-2. USB Endpoint Interrupt Enable Register 0x21 (read/write)

The interrupt controller contains a separate flip-flop for each interrupt. See *Figure 16-3* for the logic block diagram of the interrupt controller. When an interrupt is generated, it is first registered as a pending interrupt. It stays pending until it is serviced or a reset occurs. A pending interrupt only generates an interrupt request if it is enabled by the corresponding bit in the interrupt enable registers. The highest priority interrupt request is serviced following the completion of the currently executing instruction.

When servicing an interrupt, the hardware first disables all interrupts by clearing the Global Interrupt Enable bit in the CPU (the state of this bit can be read at Bit 2 of the Processor Status and Control Register). Second, the flip-flop of the current interrupt is cleared. This is followed by an automatic CALL instruction to the ROM address associated with the interrupt being serviced (i.e., the Interrupt Vector, see Section 16.1). The instruction in the interrupt table is typically a JMP instruction to the address of the Interrupt Service Routine (ISR). The user can re-enable interrupts in the interrupt service routine by executing an EI instruction. Interrupts can be nested to a level limited only by the available stack space.

The Program Counter value, as well as the Carry and Zero flags (CF, ZF), are stored onto the Program Stack by the automatic CALL instruction generated as part of the interrupt acknowledge process. The user firmware is responsible for ensuring that the processor state is preserved and restored during an interrupt. The PUSH A instruction should typically be used as the first command in the ISR to save the accumulator value and the POP A instruction should be used to restore the accumulator value

just before the RETI instruction. The program counter CF and ZF are restored and interrupts are enabled when the RETI instruction is executed.

The DI and EI instructions can be used to disable and enable interrupts, respectively. These instructions affect only the Global Interrupt Enable bit of the CPU. If desired, EI can be used to re-enable interrupts while inside an ISR, instead of waiting for the RETI that exists the ISR. While the global interrupt enable bit is cleared, the presence of a pending interrupt can be detected by examining the IRQ Sense bit (Bit 7 in the Processor Status and Control Register).

16.1 Interrupt Vectors

The Interrupt Vectors supported by the USB Controller are listed in *Table 16-1*. The lowest-numbered interrupt (USB Bus Reset interrupt) has the highest priority, and the highest-numbered interrupt (I^2C interrupt) has the lowest priority. Although Reset is not an interrupt, the first instruction executed after a reset is at PROM address 0x0000h—which corresponds to the first entry in the Interrupt Vector Table. Because the JMP instruction is 2 bytes long, the interrupt vectors occupy 2 bytes.

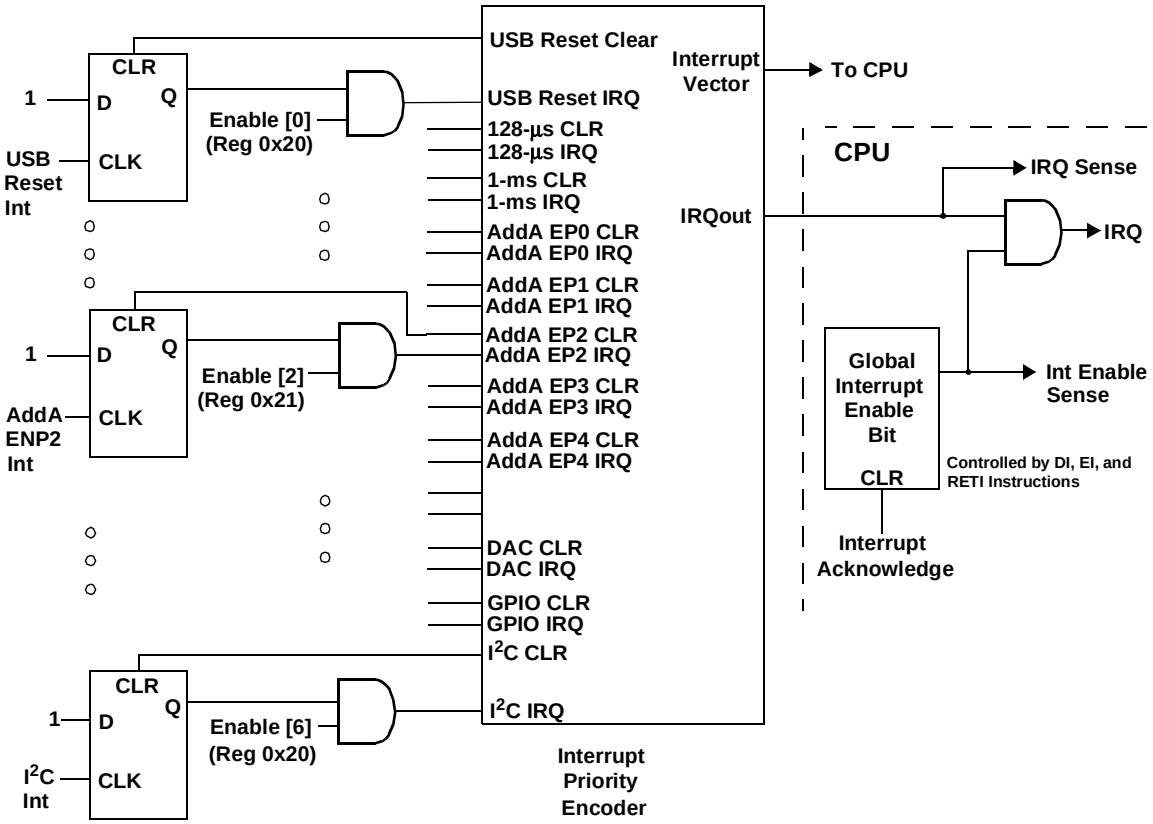


Figure 16-3. Interrupt Controller Functional Diagram

Table 16-1. Interrupt Vector Assignments

Interrupt Vector Number	ROM Address	Function
Not Applicable	0x0000	Execution after Reset begins here
1	0x0002	USB Bus Reset interrupt
2	0x0004	128-µs timer interrupt
3	0x0006	1.024-ms timer interrupt
4	0x0008	USB Address A Endpoint 0 interrupt
5	0x000A	USB Address A Endpoint 1 interrupt
6	0x000C	USB Address A Endpoint 2 interrupt
7	0x000E	USB Address A Endpoint 3 interrupt
8	0x0010	USB Address A Endpoint 4 interrupt
9	0x0012	Reserved
10	0x0014	DAC interrupt
11	0x0016	GPIO / HAPI interrupt
12	0x0018	I^2C interrupt

A pending address can be read from the Interrupt Vector Register (*Figure 16-4*). The value read from this register is only valid if the Global Interrupt bit has been disabled, by executing the DI instruction or in an Interrupt Service Routine before interrupts have been re-enabled. The value read from this register is the interrupt vector address; for example, a 0x06 indicates the 1 ms timer interrupt is the highest priority pending interrupt.

7	6	5	4	3	2	1	0
			R	R	R	R	R
Reserved	Reserved	Reserved	Interrupt Vector Bit 4	Interrupt Vector Bit 3	Interrupt Vector Bit 2	Interrupt Vector Bit 1	Reads ‘0’

Figure 16-4. Interrupt Vector Register 0x23 (read only)

16.2 Interrupt Latency

Interrupt latency can be calculated from the following equation:

Interrupt latency = (Number of clock cycles remaining in the current instruction) + (10 clock cycles for the CALL instruction) + (5 clock cycles for the JMP instruction)

For example, if a 5 clock cycle instruction such as JC is being executed when an interrupt occurs, the first instruction of the Interrupt Service Routine executes a minimum of 16 clocks (1+10+5) or a maximum of 20 clocks (5+10+5) after the interrupt is issued. For a 12-MHz internal clock (6-MHz crystal), 20 clock periods is 20 / 12 MHz = 1.667 µs.

16.3 USB Bus Reset Interrupt

The USB Controller recognizes a USB Reset when a Single Ended Zero (SE0) condition persists on the upstream USB port for 12–16 µs (the Reset may be recognized for an SE0 as short as 12 µs, but is always recognized for an SE0 longer than 16 µs). SE0 is defined as the condition in which both the D+ line and the D– line are LOW. Bit 5 of the Status and Control Register is set to record this event. The interrupt is asserted at the end of the Bus Reset. If the USB reset occurs during the start-up delay following a POR, the delay is aborted as described in Section 7.1. The USB Bus Reset Interrupt is generated when the SE0 state is deasserted.

A USB Bus Reset clears the following registers:

SIE Section:USB Device Address Registers (0x10, 0x40)

16.4 Timer Interrupt

There are two periodic timer interrupts: the 128-µs interrupt and the 1.024-ms interrupt. The user should disable both timer interrupts before going into the suspend mode to avoid possible conflicts between servicing the timer interrupts first or the suspend request first.

16.5 USB Endpoint Interrupts

There are five USB endpoint interrupts, one per endpoint. A USB endpoint interrupt is generated after the USB host writes to a USB endpoint FIFO or after the USB controller sends a packet to the USB host. The interrupt is generated on the last packet of the transaction (e.g., on the host's ACK during an IN, or on the device ACK during on OUT). If no ACK is received during an IN transaction, no interrupt is generated.

16.6 DAC Interrupt

Each DAC I/O pin can generate an interrupt, if enabled. The interrupt polarity for each DAC I/O pin is programmable. A positive polarity is a rising edge input while a negative polarity is a falling edge input. All of the DAC pins share a single interrupt vector, which means the firmware needs to read the DAC port to determine which pin or pins caused an interrupt.

If one DAC pin has triggered an interrupt, no other DAC pins can cause a DAC interrupt until that pin has returned to its inactive (non-trigger) state or the corresponding interrupt enable bit is cleared. The USB Controller does not assign interrupt priority to different DAC pins and the DAC Interrupt Enable Register is not cleared during the interrupt acknowledge process.

16.7 GPIO/HAPI Interrupt

Each of the GPIO pins can generate an interrupt, if enabled. The interrupt polarity can be programmed for each GPIO port as part of the GPIO configuration. All of the GPIO pins share a single interrupt vector, which means the firmware needs to read the GPIO ports with enabled interrupts to determine which pin or pins caused an interrupt. A block diagram of the GPIO interrupt logic is shown in *Figure 16-5*. Refer to Sections 9.1 and 9.2 for more information of setting GPIO interrupt polarity and enabling individual GPIO interrupts.

If one port pin has triggered an interrupt, no other port pins can cause a GPIO interrupt until that port pin has returned to its inactive (non-trigger) state or its corresponding port interrupt enable bit is cleared. The USB Controller does not assign interrupt priority to different port pins and the Port Interrupt Enable Registers are not cleared during the interrupt acknowledge process.

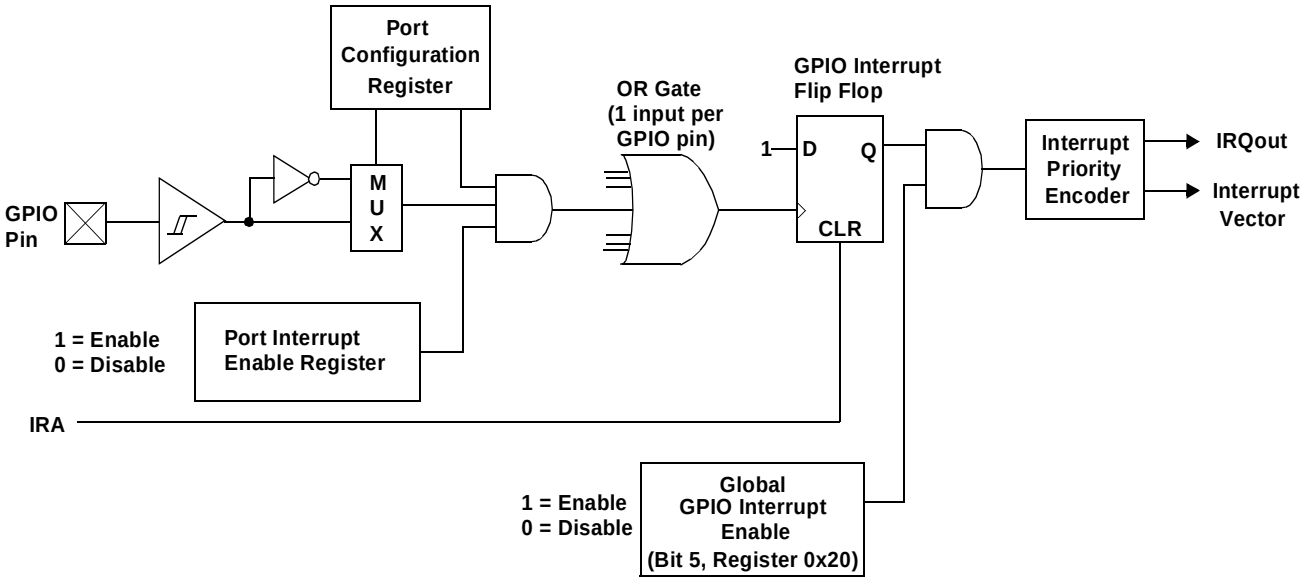


Figure 16-5. GPIO Interrupt Structure

When HAPI is enabled, the HAPI logic takes over the interrupt vector and blocks any interrupt from the GPIO bits, including ports/bits not being used by HAPI. Operation of the HAPI interrupt is independent of the GPIO specific bit interrupt enables, and is enabled or disabled only by bit 5 of the Global Interrupt Enable Register (0x20) when HAPI is enabled. The settings of the GPIO bit interrupt enables on ports/bits not used by HAPI still effect the CMOS mode operation of those ports/bits. The effect of

modifying the interrupt bits while the Port Config bits are set to “10” is shown in *Table 9-1*. The events that generate HAPI interrupts are described in Section 14.0.

16.8 I²C Interrupt

The I²C interrupt occurs after various events on the I²C bus to signal the need for firmware interaction. This generally involves reading the I²C Status and Control Register (*Figure 13-2*) to determine the cause of the interrupt, loading/reading the I²C Data Register as appropriate, and finally writing the Status and Control Register to initiate the subsequent transaction. The interrupt indicates that status bits are stable and it is safe to read and write the I²C registers. Refer to Section 13.0 for details on the I²C registers.

When enabled, the I²C state machines generate interrupts on completion of the following conditions. The referenced bits are in the I²C Status and Control Register.

1. In slave receive mode, after the slave receives a byte of data. The Addr bit is set if this is the first byte since a start or restart signal was sent by the external master. Firmware must read or write the data register as necessary, then set the ACK, Xmit Mode, and Continue bits appropriately for the next byte.
2. In slave receive mode, after a stop bit is detected. The Received Stop bit is set. If the stop bit follows a slave receive transaction where the ACK bit was cleared to 0, no stop bit detection occurs.
3. In slave transmit mode, after the slave transmits a byte of data. The ACK bit indicates if the master that requested the byte acknowledged the byte. If more bytes are to be sent, firmware writes the next byte into the Data Register and then sets the Xmit Mode and Continue bits as required.
4. In master transmit mode, after the master sends a byte of data. Firmware should load the Data Register if necessary, and set the Xmit Mode, MSTR Mode, and Continue/Busy bits appropriately. Clearing the MSTR Mode bit issues a stop signal to the I²C bus and return to the idle state.
5. In master receive mode, after the master receives a byte of data. Firmware should read the data and set the Ack and Continue/Busy bits appropriately for the next byte. Clearing the Master bit at the same time causes the master state machine to issue a stop signal to the I²C bus and leave the I²C hardware in the idle state.
6. When the master loses arbitration. This condition clears the Master bit and sets the Arbitration Lost bit immediately and then waits for a stop signal on the I²C bus to generate the interrupt.

The Continue/Busy bit is cleared by hardware prior to interrupt conditions 1 to 4. Once the Data Register has been read or written, firmware should configure the other control bits and set the Continue bit for subsequent transactions.

Following an interrupt from master mode, firmware should perform only one write to the Status and Control Register that sets the Continue bit, without checking the value of the Busy bit. The Busy bit may otherwise be active and I²C register contents may be changed by the hardware during the transaction, until the I²C interrupt occurs.

17.0 USB Overview

The USB hardware consists of the logic for a full-speed USB Port. The full-speed serial interface engine (SIE) interfaces the microcontroller to the USB bus. An external series resistor (R_{ext}) must be placed in series with the D+ and D– lines, as close to the corresponding pins as possible, to meet the USB driver requirements of the USB specifications.

17.1 USB Serial Interface Engine (SIE)

The SIE allows the CY7C64x13 microcontroller to communicate with the USB host. The SIE simplifies the interface between the microcontroller and USB by incorporating hardware that handles the following USB bus activity independently of the microcontroller:

- Bit stuffing/unstuffing
- Checksum generation/checking
- ACK/NAK/STALL
- Token type identification
- Address checking

Firmware is required to handle the following USB interface tasks:

- Coordinate enumeration by responding to SETUP packets
- Fill and empty the FIFOs
- Suspend/Resume coordination
- Verify and select DATA toggle values

17.2 USB Enumeration

The USB device is enumerated under firmware control. The following is a brief summary of the typical enumeration process of the CY7C64x13 by the USB host. For a detailed description of the enumeration process, refer to the USB specification.

In this description, ‘Firmware’ refers to embedded firmware in the CY7C64x13 controller.

1. The host computer sends a SETUP packet followed by a DATA packet to USB address 0 requesting the Device descriptor.
2. Firmware decodes the request and retrieves its Device descriptor from the program memory tables.
3. The host computer performs a control read sequence and Firmware responds by sending the Device descriptor over the USB bus, via the on-chip FIFOs.
4. After receiving the descriptor, the host sends a SETUP packet followed by a DATA packet to address 0 assigning a new USB address to the device.
5. Firmware stores the new address in its USB Device Address Register after the no-data control sequence completes.
6. The host sends a request for the Device descriptor using the new USB address.
7. Firmware decodes the request and retrieves the Device descriptor from program memory tables.
8. The host performs a control read sequence and Firmware responds by sending its Device descriptor over the USB bus.
9. The host generates control reads from the device to request the Configuration and Report descriptors.
10. Once the device receives a Set Configuration request, its functions may now be used.

17.3 USB Upstream Port Status and Control

USB status and control is regulated by the USB Status and Control Register, as shown in *Figure 17-1*. All bits in the register are cleared during reset.

7	6	5	4	3	2	1	0
R/W	R/W	R	R	R/C	R/W	R/W	R/W
Endpoint Size	Endpoint Mode	D+ Upstream	D– Upstream	Bus Activity	Control Bit 2	Control Bit 1	Control Bit 0

Figure 17-1. USB Status and Control Register 0x1F (read/write)

The three control bits allow the upstream port to be driven manually by firmware. For normal USB operation, all of these bits must be cleared. *Table 17-1* shows how the control bits affect the upstream port.

Table 17-1. Control Bit Definition for Upstream Port

Control Bits	Control Action
000	Not Forcing (SIE Controls Driver)
001	Force D+[0] HIGH, D–[0] LOW
010	Force D+[0] LOW, D–[0] HIGH
011	Force SE0; D+[0] LOW, D–[0] LOW
100	Force D+[0] LOW, D–[0] LOW
101	Force D+[0] HiZ, D–[0] LOW
110	Force D+[0] LOW, D–[0] HiZ
111	Force D+[0] HiZ, D–[0] HiZ

Bus Activity (bit 3) is a “sticky” bit that indicates if any non-idle USB event has occurred on the upstream USB port. Firmware should check and clear this bit periodically to detect any loss of bus activity. Writing a ‘0’ to the Bus Activity bit clears it, while writing a ‘1’ preserves the current value. In other words, the firmware can clear the Bus Activity bit, but only the SIE can set it.

The Upstream D– and D+ (bits 4 and 5) are read only. These give the state of each upstream port pin individually: 1=HIGH, 0=LOW.

Endpoint Mode (bit 6) and Endpoint Size (bit 7) are used to configure the number and size of USB endpoints. See Section 18.2 for a detailed description of these bits.

18.0 USB Serial Interface Engine Operation

USB Device Address A includes up to five endpoints: EPA0, EPA1, EPA2, EPA3, and EPA4. Endpoint (EPA0) allows the USB host to recognize, set-up, and control the device. In particular, EPA0 is used to receive and transmit control (including set-up) packets.

18.1 USB Device Address

The USB Controller provides one USB Device Address with five endpoints. The USB Device Address Register contents are cleared during a reset, setting the USB device address to zero and marking this address as disabled. *Figure 18-1* shows the format of the USB Address Registers.

7	6	5	4	3	2	1	0
Device Address Enable	Device Address Bit 6	Device Address Bit 5	Device Address Bit 4	Device Address Bit 3	Device Address Bit 2	Device Address Bit 1	Device Address Bit 0

Figure 18-1. USB Device Address Register 0x10 (read/write)

Bit 7 (Device Address Enable) in the USB Device Address Register must be set by firmware before the SIE can respond to USB traffic to this address. The Device Addresses in bits [6:0] are set by firmware during the USB enumeration process to the non-zero address assigned by the USB host.

18.2 USB Device Endpoints

The CY7C64x13 controller supports one USB device address and five endpoints for communication with the host. The configuration of these endpoints, and associated FIFOs, is controlled by bits [7,6] of the USB Status and Control Register (0x1F). Bit 7 controls the size of the endpoints and bit 6 controls the number of endpoints. These configuration options are detailed in *Table 18-1*. The “unused” FIFO areas in the following table can be used by the firmware as additional user RAM space.

Table 18-1. Memory Allocation for Endpoints

I/O status [7,6]	[0,0]			[1,0]			[0,1]			[1,1]		
	Label	Start Address	Size	Label	Start Address	Size	Label	Start Address	Size	Label	Start Address	Size
	unused	0xD8	8	unused	0xA8	8	EPA4	0xD8	8	EPA4	0xA8	8
	unused	0xE0	8	unused	0xB0	8	EPA3	0xE0	8	EPA3	0xB0	8
	EPA2	0xE8	8	EPA0	0xB8	8	EPA2	0xE8	8	EPA0	0xB8	8
	EPA1	0xF0	8	EPA1	0xC0	32	EPA1	0xF0	8	EPA1	0xC0	32
	EPA0	0xF8	8	EPA2	0xE0	32	EPA0	0xF8	8	EPA2	0xE0	32

When the SIE writes data to a FIFO, the internal data bus is driven by the SIE; not the CPU. This causes a short delay in the CPU operation. The delay is three clock cycles per byte. For example, an 8-byte data write by the SIE to the FIFO generates a delay of 2 µs (3 cycles/byte * 83.33 ns/cycle * 8 bytes).

18.3 USB Control Endpoint Mode Register

All USB devices are required to have a control endpoint 0 (EPA0) that is used to initialize and control each USB address. Endpoint 0 provides access to the device configuration information and allows generic USB status and control accesses. Endpoint 0 is bidirectional to both receive and transmit data. The other endpoints are unidirectional, but selectable by the user as IN or OUT endpoints.

The endpoint mode register is cleared during reset. The endpoint zero EPA0 mode register uses the format shown in *Figure 18-2*.

7	6	5	4	3	2	1	0
Endpoint 0 SETUP Received	Endpoint 0 IN Received	Endpoint 0 OUT Received	ACK	Mode Bit 3	Mode Bit 2	Mode Bit 1	Mode Bit 0

Figure 18-2. USB Device Endpoint Zero Mode Register 0x12 (read/write)

Bits[7:5] in the endpoint 0 mode registers are status bits that are set by the SIE to report the type of token that was most recently received by the corresponding device address. These bits must be cleared by firmware as part of the USB processing.

The ACK bit (bit 4) is set whenever the SIE engages in a transaction to the register's endpoint that completes with an ACK packet.

The SETUP PID status (bit 7) is forced HIGH from the start of the data packet phase of the SETUP transaction until the start of the ACK packet returned by the SIE. The CPU is prevented from clearing this bit during this interval, and subsequently, until the CPU first does an IORD to this endpoint 0 mode register.

Bits[6:0] of the endpoint 0 mode register are locked from CPU write operations whenever the SIE has updated one of these bits, which the SIE does only at the end of the token phase of a transaction (SETUP... Data... ACK, OUT... Data... ACK, or IN... Data... ACK). The CPU can unlock these bits by doing a subsequent read of this register. Only endpoint 0 mode registers are locked when updated. The locking mechanism does not apply to the mode registers of other endpoints.

Because of these hardware locking features, firmware must perform an IORD after an IOWR to an endpoint 0 register. This verifies that the contents have changed as desired, and that the SIE has not updated these values.

While the SETUP bit is set, the CPU cannot write to the endpoint zero FIFOs. This prevents firmware from overwriting an incoming SETUP transaction before firmware has a chance to read the SETUP data. Refer to *Table 18-1* for the appropriate endpoint zero memory locations.

The Mode bits (bits [3:0]) control how the endpoint responds to USB bus traffic. The mode bit encoding is shown in*Table 19-1*. Additional information on the mode bits can be found in*Table 19-2* and *Table 19-3*.

18.4 USB Non-Control Endpoint Mode Registers

The format of the non-control endpoint mode register is shown in *Figure 18-3*.

7	6	5	4	3	2	1	0
STALL	Reserved	Reserved	ACK	Mode Bit 3	Mode Bit 2	Mode Bit 1	Mode Bit 0

Figure 18-3. USB Non-Control Device Endpoint Mode Registers 0x14, 0x16, 0x42, 0x44, (read/write)

The mode bits (bits [3:0]) of the Endpoint Mode Register control how the endpoint responds to USB bus traffic. The mode bit encoding is shown in *Table 19-1*.

The ACK bit (bit 4) is set whenever the SIE engages in a transaction to the register's endpoint that completes with an ACK packet.

If STALL (bit 7) is set, the SIE stalls an OUT packet if the mode bits are set to ACK-IN, and the SIE stalls an IN packet if the mode bits are set to ACK-OUT. For all other modes, the STALL bit must be a LOW.

Bits 5 and 6 are reserved and must be written to zero during register writes.

18.5 USB Endpoint Counter Registers

There are five Endpoint Counter registers, with identical formats for both control and non-control endpoints. These registers contain byte count information for USB transactions, as well as bits for data packet status. The format of these registers is shown in *Figure 18-4*:

7	6	5	4	3	2	1	0
Data 0/1 Toggle	Data Valid	Byte Count Bit 5	Byte Count Bit 4	Byte Count Bit 3	Byte Count Bit 2	Byte Count Bit 1	Byte Count Bit 0

Figure 18-4. USB Endpoint Counter Registers 0x11, 0x13, 0x15, 0x41, 0x43 (read/write)

The counter bits (bits [5:0]) indicate the number of data bytes in a transaction. For IN transactions, firmware loads the count with the number of bytes to be transmitted to the host from the endpoint FIFO. Valid values are 0 to 32, inclusive. For OUT or SETUP transactions, the count is updated by hardware to the number of data bytes received, plus 2 for the CRC bytes. Valid values are 2 to 34, inclusive.

Data Valid bit 6 is used for OUT and SETUP tokens only. Data is loaded into the FIFOs during the transaction, and then the Data Valid bit is set if a proper CRC is received. If the CRC is not correct, the endpoint interrupt occurs, but Data Valid is cleared to a zero.

Data 0/1 Toggle bit 7 selects the DATA packet's toggle state: 0 for DATA0, 1 for DATA1. For IN transactions, firmware must set this bit to the desired state. For OUT or SETUP transactions, the hardware sets this bit to the state of the received Data Toggle bit.

Whenever the count updates from a SETUP or OUT transaction on endpoint 0, the counter register locks and cannot be written by the CPU. Reading the register unlocks it. This prevents firmware from overwriting a status update on incoming SETUP or OUT

transactions before firmware has a chance to read the data. Only endpoint 0 counter register is locked when updated. The locking mechanism does not apply to the count registers of other endpoints.

18.6 Endpoint Mode/Count Registers update and Locking Mechanism

The contents of the endpoint mode and counter registers are updated, based on the packet flow diagram in *Figure 18-5*. Two time points, UPDATE and SETUP, are shown in the same figure. The following activities occur at each time point:

UPDATE:

1. Endpoint Mode Register - All the bits are updated (except the SETUP bit of the endpoint 0 mode register).
2. Counter Registers - All bits are updated.
3. Interrupt - If an interrupt is to be generated as a result of the transaction, the interrupt flag for the corresponding endpoint is set at this time. For details on what conditions are required to generate an endpoint interrupt, refer to *Table 19-2*.
4. The contents of the updated endpoint 0 mode and counter registers are locked, except the SETUP bit of the endpoint 0 mode register which was locked earlier.

SETUP:

The SETUP bit of the endpoint 0 mode register is forced HIGH at this time. This bit is forced HIGH by the SIE until the end of the data phase of a control write transfer. The SETUP bit can not be cleared by firmware during this time.

The affected mode and counter registers of endpoint 0 are locked from any CPU writes once they are updated. These registers can be unlocked by a CPU read, only if the read operation occurs after the UPDATE. The firmware needs to perform a register read as a part of the endpoint ISR processing to unlock the effected registers. The locking mechanism on mode and counter registers ensures that the firmware recognizes the changes that the SIE might have made since the previous IO read of that register.

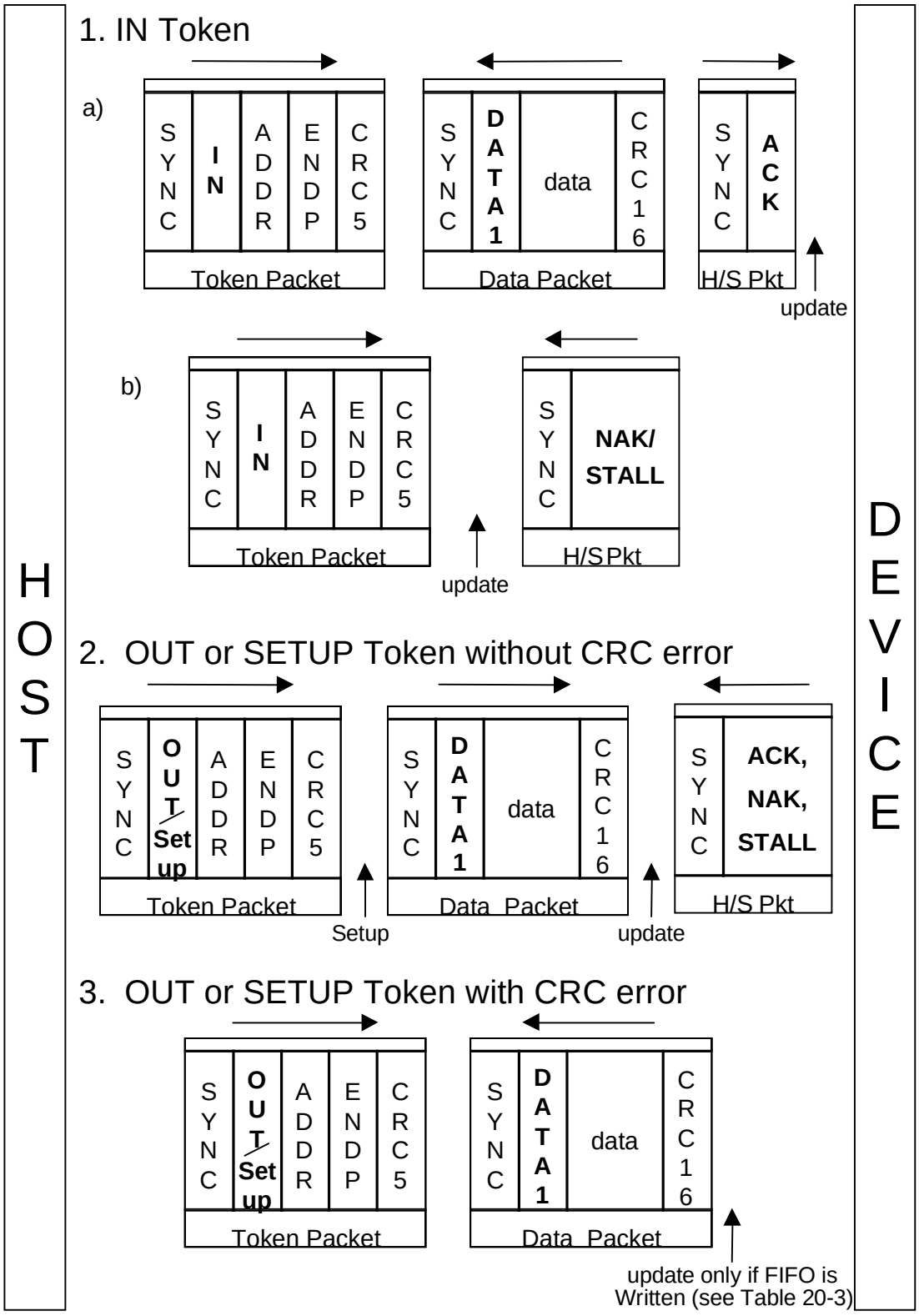


Figure 18-5. Token/Data Packet Flow Diagram

19.0 USB Mode Tables

Table 19-1. USB Register Mode Encoding

Mode	Encoding	Setup	In	Out	Comments
Disable	**0000**	ignore	ignore	ignore	Ignore all USB traffic to this endpoint
Nak In/Out	**0001**	accept	NAK	NAK	Forced from Set-up on Control endpoint, from modes other than 0000
Status Out Only	**0010**	accept	stall	check	For Control endpoints
Stall In/Out	**0011**	accept	stall	stall	For Control endpoints
Ignore In/Out	**0100**	accept	ignore	ignore	For Control endpoints
Isochronous Out	**0101**	ignore	ignore	always	For Isochronous endpoints
Status In Only	**0110**	accept	TX 0	stall	For Control Endpoints
Isochronous In	**0111**	ignore	TX cnt	ignore	For Isochronous endpoints
Nak Out	**1000**	ignore	ignore	NAK	An ACK from mode 1001 --> 1000
Ack Out(STALL[3]=0) Ack Out(STALL[3]=1)	**1001** **1001**	ignore ignore	ignore ignore	ACK stall	This mode is changed by SIE on issuance of ACK --> 1000
Nak Out - Status In	**1010**	accept	TX 0	NAK	An ACK from mode 1011 --> 1010
Ack Out - Status In	**1011**	accept	TX 0	ACK	This mode is changed by SIE on issuance of ACK --> 1010
Nak In	**1100**	ignore	NAK	ignore	An ACK from mode 1101 --> 1100
Ack IN(STALL[3]=0) Ack IN(STALL[3]=1)	**1101** **1101**	ignore ignore	TX cnt stall	ignore ignore	This mode is changed by SIE on issuance of ACK --> 1100
Nak In - Status Out	**1110**	accept	NAK	check	An ACK from mode 1111 --> 111 Ack In - Status Out
Ack In - Status Out	**1111**	accept	TX cnt	check	This mode is changed by SIE on issuance of ACK -->1110

Note:
3. STALL bit is bit 7 of the USB Non-Control Device Endpoint Mode registers. For more information, refer to Section 18.4.

The 'In' column represents the SIE's response to the token type.

A disabled endpoint remains disabled until it is changed by firmware, and all endpoints reset to the disabled state.

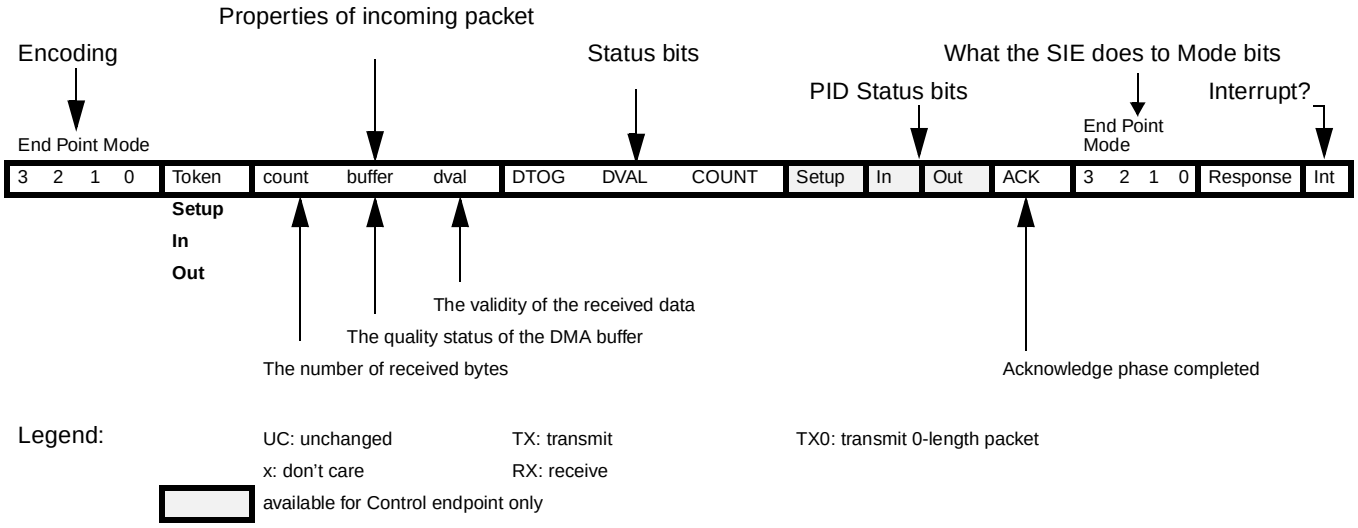
Any SETUP packet to an enabled endpoint with mode set to accept SETUPs is changed by the SIE to 0001 (NAKing). Any mode set to accept a SETUP, ACKs a valid SETUP transaction.

Most modes that control transactions involving an ending ACK, are changed by the SIE to a corresponding mode which NAKs subsequent packets following the ACK. Exceptions are modes 1010 and 1110.

A Control endpoint has three extra status bits for PID (Setup, In and Out), but must be placed in the correct mode to function as such. Non-Control endpoints should not be placed into modes that accept SETUPs.

A 'check' on an Out token during a Status transaction checks to see that the Out is of zero length and has a Data Toggle (DTOG) of '1'. If the DTOG bit is set and the received Out Packet has zero length, the Out is ACKed to complete the transaction. Otherwise, the Out is STALLed.

Table 19-2. Decode table for *Table 19-3*: "Details of Modes for Differing Traffic Conditions"



The response of the SIE can be summarized as follows:

1. The SIE only responds to valid transactions and ignores non-valid ones.
2. The SIE generates an interrupt when a valid transaction is completed or when the FIFO is corrupted. FIFO corruption occurs during an OUT or SETUP transaction to a valid internal address that ends with a non-valid CRC.
3. An incoming Data packet is valid if the count is ≤ Endpoint Size + 2 (includes CRC) and passes all error checking.
4. An IN is ignored by an OUT configured endpoint and vice versa.
5. The IN and OUT PID status is updated at the end of a transaction.
6. The SETUP PID status is updated at the beginning of the Data packet phase.
7. The entire Endpoint 0 mode register and the count register are locked from CPU writes at the end of any transaction to that endpoint in which either an ACK is transferred or the mode bits have changed. These registers are only unlocked by a CPU read of these registers, and only if that read happens after the transaction completes. This represents about a 1-µs window in which the CPU is locked from register writes to these USB registers. Normally, the firmware should perform a register read at the beginning of the Endpoint ISRs to unlock and get the mode register information. The interlock on the Mode and Count registers ensures that the firmware recognizes the changes that the SIE might have made during the previous transaction.

Table 19-3. Details of Modes for Differing Traffic Conditions (see *Table 19-2* for the decode legend)

End Point Mode											PID				Set End Point Mode					
3	**2**	**1**	**0**	**token**	**count**	**buffer**	**dval**	**DTOG**	**DVAL**	**COUNT**	**Setup**	**In**	**Out**	**ACK**	**3**	**2**	**1**	**0**	**response**	**int**
Setup Packet (if accepting)																				
See *Table 19-1*				Setup	<= 10	data	valid	updates	1	updates	1	UC	UC	1	0	0	0	1	ACK	yes
See *Table 19-1*				Setup	> 10	junk	x	updates	updates	updates	1	UC	UC	UC	NoChange				ignore	yes
See *Table 19-1*				Setup	x	junk	invalid	updates	0	updates	1	UC	UC	UC	NoChange				ignore	yes
Disabled																				
0	0	0	0	x	x	UC	x	UC	UC	UC	UC	UC	UC	UC	NoChange				ignore	no
Nak In/Out																				
0	0	0	1	Out	x	UC	x	UC	UC	UC	UC	UC	1	UC	NoChange				NAK	yes
0	0	0	1	In	x	UC	x	UC	UC	UC	UC	1	UC	UC	NoChange				NAK	yes
Ignore In/Out																				
0	1	0	0	Out	x	UC	x	UC	UC	UC	UC	UC	UC	UC	NoChange				ignore	no
0	1	0	0	In	x	UC	x	UC	UC	UC	UC	UC	UC	UC	NoChange				ignore	no
Stall In/Out																				
0	0	1	1	Out	x	UC	x	UC	UC	UC	UC	UC	1	UC	NoChange				Stall	yes
0	0	1	1	In	x	UC	x	UC	UC	UC	UC	1	UC	UC	NoChange				Stall	yes
Control Write																				
Normal Out/premature status In																				
1	0	1	1	Out	<= 10	data	valid	updates	1	updates	UC	UC	1	1	1	0	1	0	ACK	yes
1	0	1	1	Out	> 10	junk	x	updates	updates	updates	UC	UC	1	UC	NoChange				ignore	yes
1	0	1	1	Out	x	junk	invalid	updates	0	updates	UC	UC	1	UC	NoChange				ignore	yes
1	0	1	1	In	x	UC	x	UC	UC	UC	UC	1	UC	1	NoChange				TX 0	yes
NAK Out/premature status In																				
1	0	1	0	Out	<= 10	UC	valid	UC	UC	UC	UC	UC	1	UC	NoChange				NAK	yes
1	0	1	0	Out	> 10	UC	x	UC	UC	UC	UC	UC	UC	UC	NoChange				ignore	no
1	0	1	0	Out	x	UC	invalid	UC	UC	UC	UC	UC	UC	UC	NoChange				ignore	no
1	0	1	0	In	x	UC	x	UC	UC	UC	UC	1	UC	1	NoChange				TX 0	yes
Status In/extra Out																				
0	1	1	0	Out	<= 10	UC	valid	UC	UC	UC	UC	UC	1	UC	0	0	1	1	Stall	yes
0	1	1	0	Out	> 10	UC	x	UC	UC	UC	UC	UC	UC	UC	NoChange				ignore	no
0	1	1	0	Out	x	UC	invalid	UC	UC	UC	UC	UC	UC	UC	NoChange				ignore	no
0	1	1	0	In	x	UC	x	UC	UC	UC	UC	1	UC	1	NoChange				TX 0	yes
Control Read																				
Normal In/premature status Out																				
1	1	1	1	Out	2	UC	valid	1	1	updates	UC	UC	1	1	NoChange				ACK	yes
1	1	1	1	Out	2	UC	valid	0	1	updates	UC	UC	1	UC	0	0	1	1	Stall	yes
1	1	1	1	Out	!=2	UC	valid	updates	1	updates	UC	UC	1	UC	0	0	1	1	Stall	yes
1	1	1	1	Out	> 10	UC	x	UC	UC	UC	UC	UC	UC	UC	NoChange				ignore	no
1	1	1	1	Out	x	UC	invalid	UC	UC	UC	UC	UC	UC	UC	NoChange				ignore	no
1	1	1	1	In	x	UC	x	UC	UC	UC	UC	1	UC	1	1	1	1	0	ACK (back)	yes
Nak In/premature status Out																				
1	1	1	0	Out	2	UC	valid	1	1	updates	UC	UC	1	1	NoChange				ACK	yes
1	1	1	0	Out	2	UC	valid	0	1	updates	UC	UC	1	UC	0	0	1	1	Stall	yes
1	1	1	0	Out	!=2	UC	valid	updates	1	updates	UC	UC	1	UC	0	0	1	1	Stall	yes
1	1	1	0	Out	> 10	UC	x	UC	UC	UC	UC	UC	UC	UC	NoChange				ignore	no
1	1	1	0	Out	x	UC	invalid	UC	UC	UC	UC	UC	UC	UC	NoChange				ignore	no
1	1	1	0	In	x	UC	x	UC	UC	UC	UC	1	UC	UC	NoChange				NAK	yes
Status Out/extra In																				
0	0	1	0	Out	2	UC	valid	1	1	updates	UC	UC	1	1	NoChange				ACK	yes
0	0	1	0	Out	2	UC	valid	0	1	updates	UC	UC	1	UC	0	0	1	1	Stall	yes
0	0	1	0	Out	!=2	UC	valid	updates	1	updates	UC	UC	1	UC	0	0	1	1	Stall	yes

Table 19-3. Details of Modes for Differing Traffic Conditions (see *Table 19-2* for the decode legend) (continued)

End Point Mode											PID				Set End Point Mode					
3	**2**	**1**	**0**	**token**	**count**	**buffer**	**dval**	**DTOG**	**DVAL**	**COUNT**	**Setup**	**In**	**Out**	**ACK**	**3**	**2**	**1**	**0**	**response**	**int**
0	0	1	0	Out	> 10	UC	x	UC	UC	UC	UC	UC	UC	UC	NoChange				ignore	no
0	0	1	0	Out	x	UC	invalid	UC	UC	UC	UC	1	UC	UC	NoChange				ignore	no
0	0	1	0	In	x	UC	x	UC	UC	UC	UC	1	UC	UC	0	0	1	1	Stall	yes
Out endpoint																				
Normal Out/erroneous In																				
1	0	0	1	Out	<= 10	data	valid	updates	1	updates	UC	UC	1	1	1	0	0	0	ACK	yes
1	0	0	1	Out	> 10	junk	x	updates	updates	updates	UC	UC	1	UC	NoChange				ignore	yes
1	0	0	1	Out	x	junk	invalid	updates	0	updates	UC	UC	1	UC	NoChange				ignore	yes
1	0	0	1	In	x	UC	x	UC	UC	UC	UC	UC	UC	UC	NoChange				ignore (STALL[3] = 0)	no
1	0	0	1	In	x	UC	x	UC	UC	UC	UC	UC	UC	UC	NoChange				Stall (STALL[3] = 1)	no
NAK Out/erroneous In																				
1	0	0	0	Out	<= 10	UC	valid	UC	UC	UC	UC	UC	1	UC	NoChange				NAK	yes
1	0	0	0	Out	> 10	UC	x	UC	UC	UC	UC	UC	UC	UC	NoChange				ignore	no
1	0	0	0	Out	x	UC	invalid	UC	UC	UC	UC	UC	UC	UC	NoChange				ignore	no
1	0	0	0	In	x	UC	x	UC	UC	UC	UC	UC	UC	UC	NoChange				ignore	no
Isochronous endpoint (Out)																				
0	1	0	1	Out	x	updates	updates	updates	updates	updates	UC	UC	1	1	NoChange				RX	yes
0	1	0	1	In	x	UC	x	UC	UC	UC	UC	UC	UC	UC	NoChange				ignore	no
In endpoint																				
Normal In/erroneous Out																				
1	1	0	1	Out	x	UC	x	UC	UC	UC	UC	UC	UC	UC	NoChange				ignore (STALL[3] = 0)	no
1	1	0	1	Out	x	UC	x	UC	UC	UC	UC	UC	UC	UC	NoChange				stall (STALL[3] = 1)	no
1	1	0	1	In	x	UC	x	UC	UC	UC	UC	1	UC	1	1	1	0	0	ACK (back)	yes
NAK In/erroneous Out																				
1	1	0	0	Out	x	UC	x	UC	UC	UC	UC	UC	UC	UC	NoChange				ignore	no
1	1	0	0	In	x	UC	x	UC	UC	UC	UC	1	UC	UC	NoChange				NAK	yes
Isochronous endpoint (In)																				
0	1	1	1	Out	x	UC	x	UC	UC	UC	UC	UC	UC	UC	NoChange				ignore	no
0	1	1	1	In	x	UC	x	UC	UC	UC	UC	1	UC	UC	NoChange				TX	yes

20.0 Absolute Maximum Ratings

Storage Temperature .....–65°C to +150°C
Ambient Temperature with Power Applied .....0°C to +70°C
Supply voltage on V_{CC} relative to V_{SS} .....–0.5V to +7.0V
DC Input Voltage..... –0.5V to +V_{CC}+0.5V
DC Voltage Applied to Outputs in High Z State ..... –0.5V to +V_{CC}+0.5V
Power Dissipation .....500 mW
Static Discharge Voltage ..... >2000V
Latch-up Current ..... >200 mA
Max Output Sink Current into Port 0, 1, 2, 3, and DAC[1:0] Pins ..... 60 mA
Max Output Sink Current into DAC[7:2] Pins ..... 10 mA

21.0 Electrical Characteristics

f_{OSC} = 6 MHz; Operating Temperature = 0 to 70°C, V_{CC} = 4.0 to 5.25 Volts

	Parameter	Min.	Max.	Unit	Conditions
	General				
V_{REF}	Reference Voltage	3.15	3.45	V	3.3V ±5%
V_{pp}	Programming Voltage (disabled)	–0.4	0.4	V	
I_{CC}	V_{CC} Operating Current		50	mA	No GPIO source current
I_{SB1}	Supply Current—Suspend Mode		50	µA	
I_{ref}	V_{REF} Operating Current		30	mA	Note 5
I_{il}	Input Leakage Current		1	µA	any pin
	USB Interface				
V_{di}	Differential Input Sensitivity	0.2		V	| (D+)–(D–) |
V_{cm}	Differential Input Common Mode Range	0.8	2.5	V	
V_{se}	Single Ended Receiver Threshold	0.8	2.0	V	
C_{in}	Transceiver Capacitance		20	pF	
I_{lo}	Hi-Z State Data Line Leakage	–10	10	µA	0 V < V_{in} < 3.3 V
R_{ext}	External USB Series Resistor	19	21	Ω	In series with each USB pin
R_{UUP}	External Upstream USB Pull-up Resistor	1.425	1.575	kΩ	1.5 kΩ ±5%, D+ to V_{REG}
	Power On Reset				
t_{vccs}	V_{CC} Ramp Rate	0	100	ms	linear ramp 0V to V_{CC}[4]
	USB Upstream				
V_{UOH}	Static Output High	2.8	3.6	V	15 kΩ ±5% to Gnd
V_{UOL}	Static Output Low		0.3	V	1.5 kΩ ±5% to V_{REF}
Z_O	USB Driver Output Impedance	28	44	Ω	Including R_{ext} Resistor
	General Purpose I/O (GPIO)				
R_{up}	Pull-up Resistance (typical 14 kΩ)	8.0	24.0	kΩ	
V_{ITH}	Input Threshold Voltage	20%	40%	V_{CC}	All ports, LOW to HIGH edge
V_H	Input Hysteresis Voltage	2%	8%	V_{CC}	All ports, HIGH to LOW edge
V_{OL0}	Port 0,1,2 Output Low Voltage		0.4 2.0	V V	I_{OL} = 3 mA I_{OL} = 5 mA
V_{OL3}	Port 3 Output Low Voltage		0.4 2.0	V V	I_{OL} = 3 mA I_{OL} = 8 mA
V_{OH}	Output High Voltage	2.4		mA	I_{OH} = 1.9 mA (all ports 0,1,2,3)

	Parameter	Min.	Max.	Unit	Conditions
	DAC Interface				
R_{up}	DAC Pull-up Resistance (typical 14 kΩ)	8.0	24.0	kΩ	
$I_{sink0(0)}$	DAC[7:2] Sink current (0)	0.1	0.3	mA	V_{out} = 2.0V DC
$I_{sink0(F)}$	DAC[7:2] Sink current (F)	0.5	1.5	mA	V_{out} = 2.0V DC
$I_{sink1(0)}$	DAC[1:0] Sink current (0)	1.6	4.8	mA	V_{out} = 2.0V DC
$I_{sink1(F)}$	DAC[1:0] Sink current (F)	8	24	mA	V_{out} = 2.0V DC
I_{range}	Programmed Isink Ratio: max/min	4	6		V_{out} = 2.0V DC[6]
T_{ratio}	Tracking Ratio DAC[1:0] to DAC[7:2]	14	22		V_{out} = 2.0V[7]
$I_{sinkDAC}$	DAC Sink Current	1.6	4.8	mA	V_{out} = 2.0V DC
I_{lin}	Differential Nonlinearity		0.6	LSB	DAC Port[8]

Notes:

4. Power-on Reset occurs whenever the voltage on V_{CC} is below approximately 2.5V.
5. This is based on transitions every 2 full-speed bit times on average.
6. Irange: Isinkn(15)/ Isinkn(0) for the same pin.
7. T_{ratio} = Isink1[1:0](n)/Isink0[7:2](n) for the same n, programmed.
8. I_{lin} measured as largest step size vs. nominal according to measured full scale and zero programmed values.

22.0 Switching Characteristics (f_{OSC} = 6.0 MHz)

Parameter	Description	Min.	Max.	Unit
	Clock Source			
f_{OSC}	Clock Rate	6 ±0.25%		MHz
t_{cyc}	Clock Period	166.25	167.08	ns
t_{CH}	Clock HIGH time	0.45 t_{CYC}		ns
t_{CL}	Clock LOW time	0.45 t_{CYC}		ns
	USB Full Speed Signaling[9]			
t_{rfs}	Transition Rise Time	4	20	ns
t_{ffs}	Transition Fall Time	4	20	ns
t_{rfmfs}	Rise / Fall Time Matching; (t_r/t_f)	90	111	%
$t_{dratefs}$	Full Speed Date Rate	12 ±0.25%		Mb/s
	DAC Interface			
t_{sink}	Current Sink Response Time		0.8	µs
	HAPI Read Cycle Timing			
t_{RD}	Read Pulse Width	15		ns
t_{OED}	OE LOW to Data Valid[10, 11]		40	ns
t_{OEZ}	OE HIGH to Data High-Z[11]		20	ns
t_{OEDR}	OE LOW to Data_Ready Deasserted[10, 11]	0	60	ns
	HAPI Write Cycle Timing			
t_{WR}	Write Strobe Width	15		ns
t_{DSTB}	Data Valid to STB HIGH (Data Set-up Time)[11]	5		ns
t_{STBZ}	STB HIGH to Data High-Z (Data Hold Time)[11]	15		ns
t_{STBLE}	STB LOW to Latch_Empty Deasserted[10, 11]	0	50	ns
	Timer Signals			
t_{watch}	WatchDog Timer Period	8.192	14.336	ms

Notes:
9. Per Table 7-6 of revision 1.1 of USB specification.
10. For 25-pF load.
11. Assumes chip select $\overline{CS}$ is asserted (LOW).

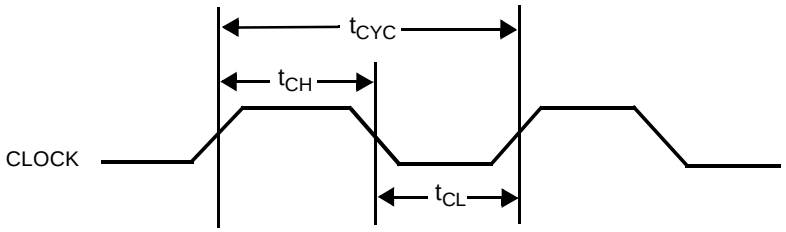


Figure 22-1. Clock Timing

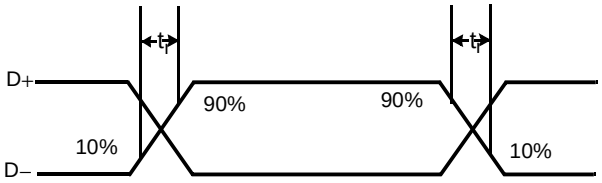


Figure 22-2. USB Data Signal Timing

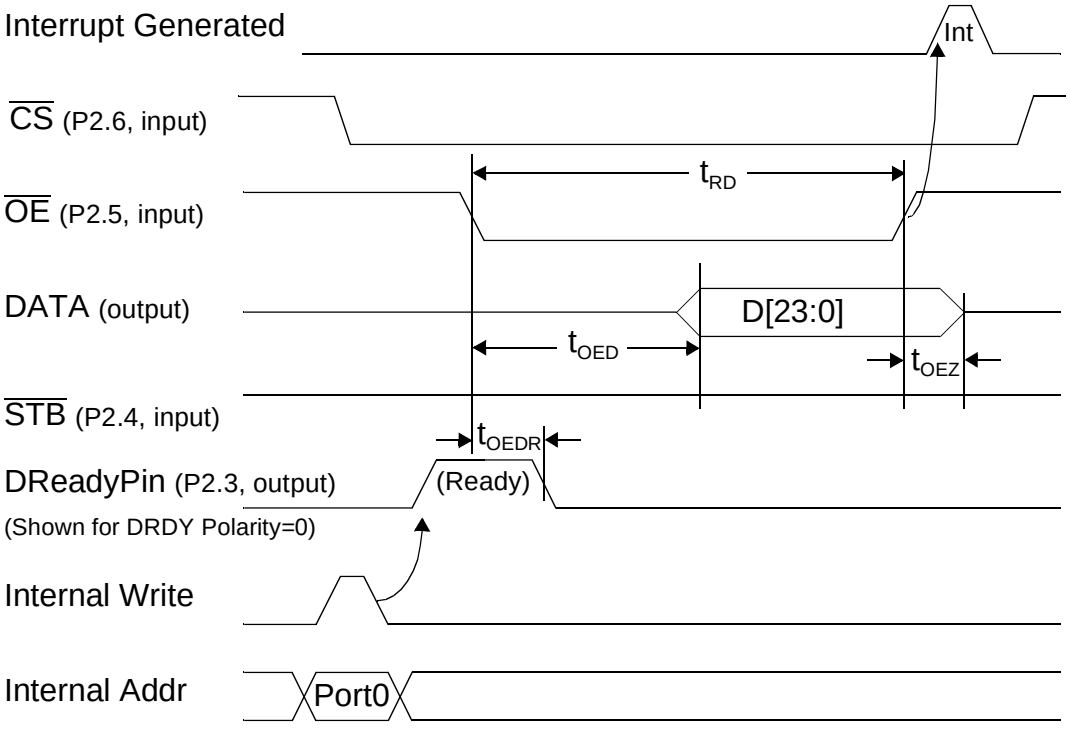


Figure 22-3. HAPI Read by External Interface from USB Microcontroller

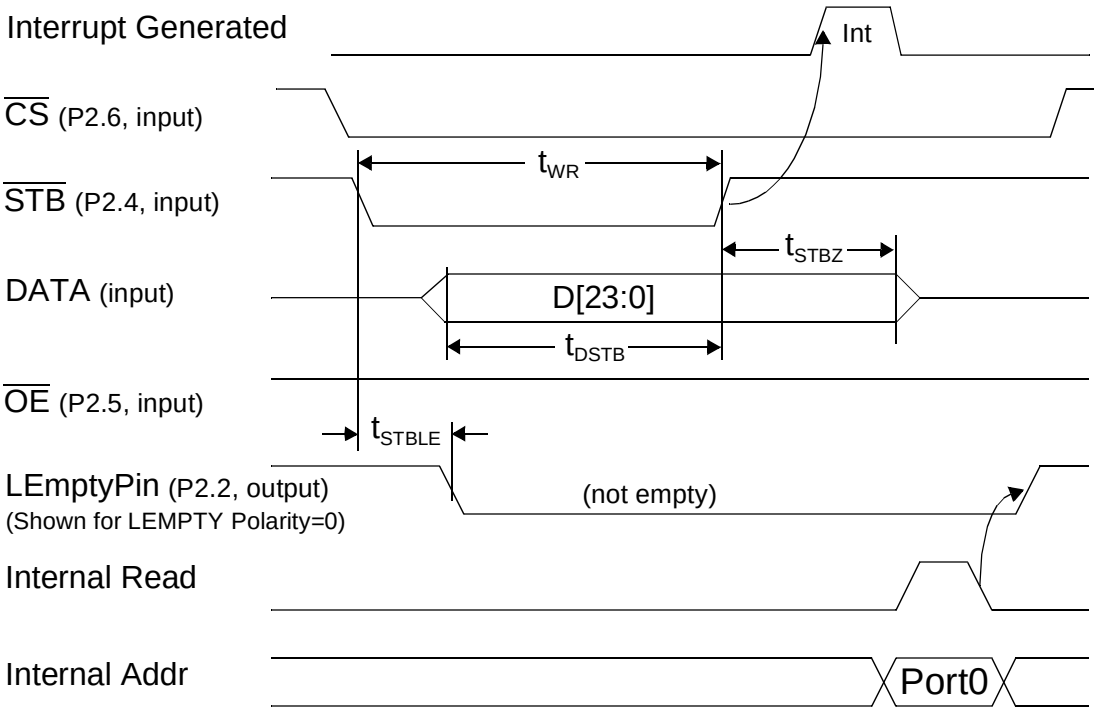


Figure 22-4. HAPI Write by External Device to USB Microcontroller

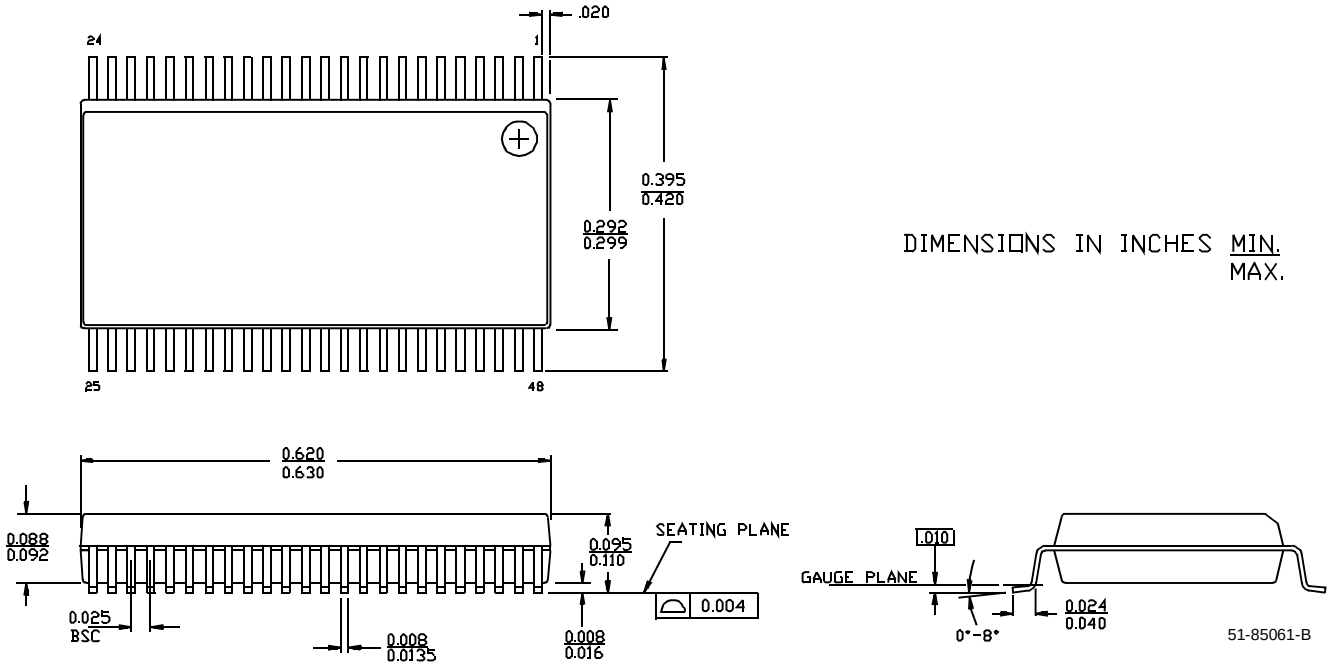
23.0 Ordering Information

Ordering Code	PROM Size	Package Name	Package Type	Operating Range
CY7C64013-SC	8 KB	S21	28-Pin (300-Mil) SOIC	Commercial
CY7C64013-PC	8 KB	P21	28-Pin (300-Mil) PDIP	Commercial
CY7C64113-PVC	8 KB	O48	48-Pin (300-Mil) SSOP	Commercial

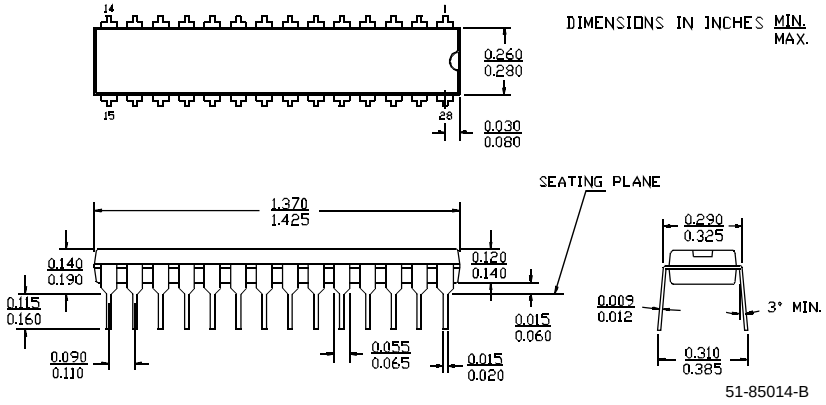
Document #: 38-00626-D

24.0 Package Diagrams

48-Lead Shrunk Small Outline Package O48



28-Lead (300-Mil) Molded DIP P21



24.0 Package Diagrams (continued)

28-Lead (300-Mil) Molded SOIC S21