



Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!



Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832

Email & Skype: info@chipsmall.com Web: www.chipsmall.com

Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China



USB Hub with Microcontroller

Features

- Full Speed USB hub with an integrated microcontroller
- 8-bit USB optimized microcontroller
 - Harvard architecture
 - 6-MHz external clock source
 - 12-MHz internal CPU clock
 - 48-MHz internal hub clock
- Internal memory
 - 256 bytes of RAM
 - 8 KB of PROM
- Integrated Master/Slave I²C-compatible Controller (100 kHz) enabled through General-purpose I/O (GPIO) pins
- I/O ports
 - Two GPIO ports (Port 0 to 2) capable of sinking 7 mA per pin (typical)
 - 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
 - Maskable interrupts on all I/O pins
- 12-bit free-running timer with one microsecond clock ticks
- Watchdog 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 one or two device addresses with up to 5 user-configured endpoints
 - Up to two 8-byte control endpoints
 - Up to four 8-byte data endpoints
 - Up to two 32-byte data endpoints
 - Integrated USB transceivers
 - Supports four downstream USB ports
 - GPIO pins can provide individual power control outputs for each downstream USB port
 - GPIO pins can provide individual port over current inputs for each downstream USB port
- Improved output drivers to reduce electromagnetic interference (EMI)
- Operating voltage from 4.0V to 5.5V DC
- Operating temperature from 0° to 70° C
- Available in 28-pin SOIC (-SXC) package
- Industry-standard programmer support

Functional Overview

The CY7C65113C device is a one-time programmable 8-bit microcontroller with a built-in 12-Mbps USB hub that supports up to four downstream ports. The microcontroller instruction set has been optimized specifically for USB operations, although the microcontrollers can be used for a variety of non-USB embedded applications.

GPIO

The CY7C65113C has 11 GPIO pins (P0[7:0], P1[2:0]), both rated at 7 mA per pin (typical) sink current. Multiple GPIO pins can be connected together to drive a single output for more drive current capacity.

Clock

The microcontroller uses an external 6-MHz crystal and an internal oscillator to provide a reference to an internal phase-locked loop (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.

Memory

The CY7C65113C is offered with 8 KB of PROM.

Power-on Reset, Watchdog, and Free-running Timer

These parts include power-on reset logic, a Watchdog timer, and a 12-bit free-running timer. The 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 Watchdog 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.

I²C

The microcontroller can communicate with external electronics through the GPIO pins. An I²C-compatible interface accommodates a 100-kHz serial link with an external device.

Timer

The free-running 12-bit timer clocked at 1 MHz provides two interrupt sources, 128- μ s 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.

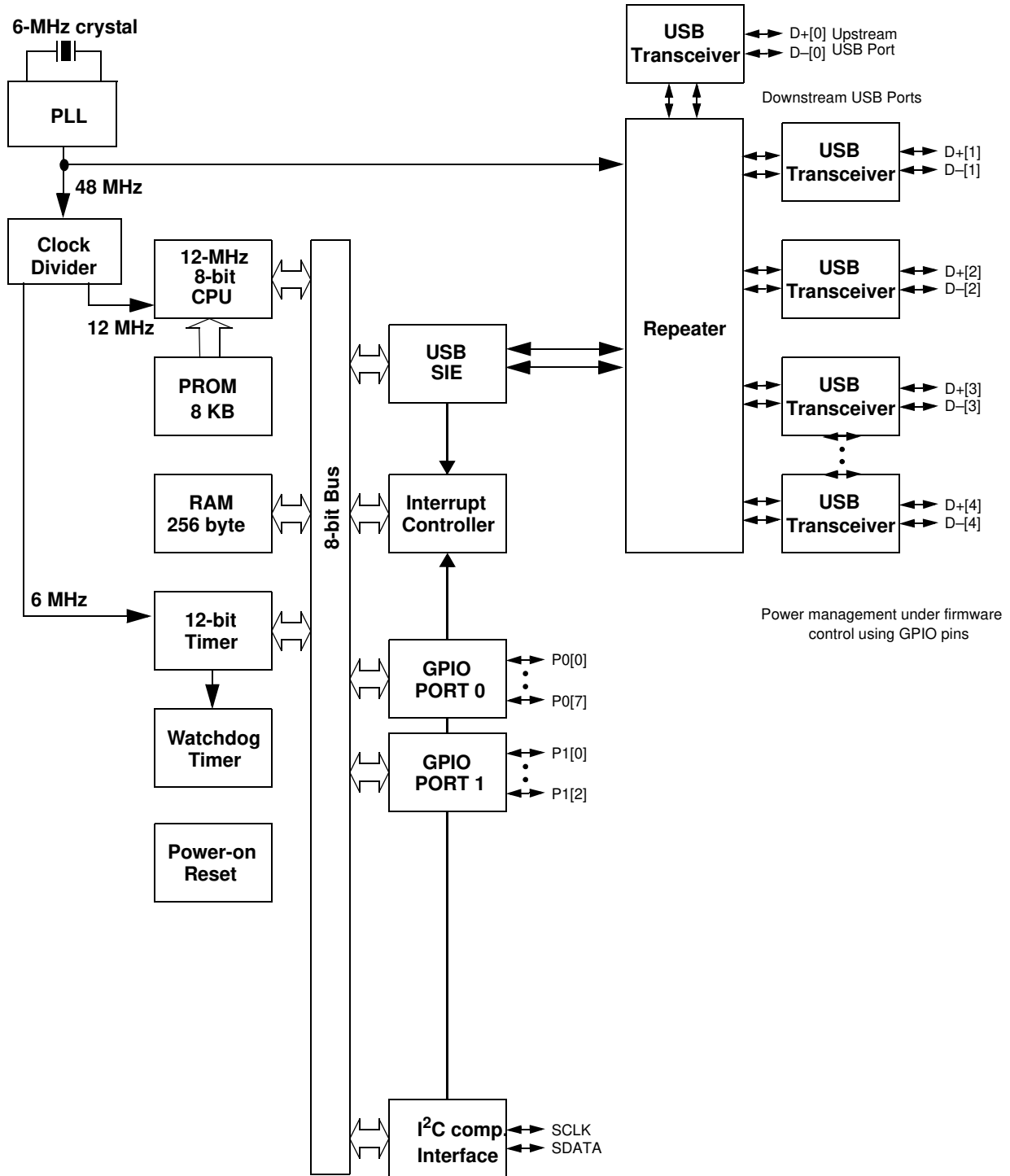
Interrupts

The microcontroller supports ten maskable interrupts in the vectored interrupt controller. Interrupt sources include the USB Bus Reset interrupt, the 128- μ s (bit 6) and 1.024-ms (bit 9) outputs from the free-running timer, five USB endpoints, the USB hub, the GPIO ports, and the I²C-compatible 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 GPIO ports also have a level of masking to select which GPIO inputs can cause a GPIO interrupt. 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').

USB

The CY7C65113C includes an integrated USB Serial Interface Engine (SIE) that supports the integrated peripherals and the hub controller function. The hardware supports up to two USB device addresses with one device address for the hub (two endpoints) and a device address for a compound device (three endpoints). The SIE allows the USB host to communicate with the hub and functions integrated into the microcontroller. The CY7C65113C part includes a 1:4 hub repeater with one upstream port and four downstream ports. The USB Hub allows power management control of the downstream ports by using GPIO pins assigned by the user firmware. The user has the option of ganging the downstream ports together with a single pair of power management pins, or providing power management for each port with four pairs of power management pins.

Logic Block Diagram



*I²C-compatible interface enabled by firmware through P1[1:0]

Power management under firmware control using GPIO pins

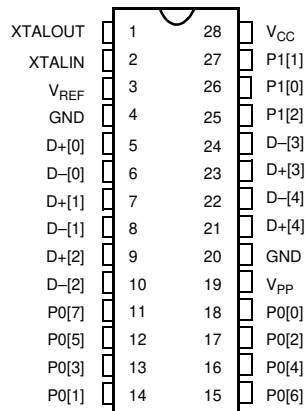
Contents

Pin Configurations	5	Sample Schematic	41
Product Summary Tables	5	Absolute Maximum Ratings	41
Programming Model	8	Electrical Characteristics	42
Clocking	11	Switching Characteristics	43
Reset	12	Ordering Information	44
Suspend Mode	13	Ordering Code Definitions	44
General-purpose I/O Ports	14	Package Diagram	45
12-bit Free-Running Timer	17	Acronyms	46
I²C Configuration Register	18	Document Conventions	46
I2C-compatible Controller	18	Units of Measure	46
Processor Status and Control Register	20	Document History Page	47
Interrupts	21	Sales, Solutions, and Legal Information	48
USB Overview	26	Worldwide Sales and Design Support	48
USB Hub	26	Products	48
USB Mode Tables	35	PSoC Solutions	48
Register Summary	39		

Pin Configurations

Figure 1. CY7C65113C 28-Pin SOIC

Top View



Product Summary Tables

Pin Assignments

Table 1. Pin Assignments

Name	I/O	28-pin	Description
D+[0], D-[0]	I/O	5, 6	Upstream port, USB differential data.
D+[1], D-[1]	I/O	7, 8	Downstream Port 1, USB differential data.
D+[2], D-[2]	I/O	9, 10	Downstream Port 2, USB differential data.
D+[3], D-[3]	I/O	23, 24	Downstream Port 3, USB differential data.
D+[4], D-[4]	I/O	21, 22	Downstream Port 4, USB differential data.
P0	I/O	P1[7:0] 11, 15, 12, 16, 13, 17, 14, 18	GPIO Port 0 capable of sinking 7 mA (typical).
P1	I/O	P1[2:0] 25, 27, 26	GPIO Port 1 capable of sinking 7 mA (typical).
XTAL _{IN}	IN	2	6-MHz crystal or external clock input.
XTAL _{OUT}	OUT	1	6-MHz crystal out.
V _{PP}		19	Programming voltage supply, tie to ground during normal operation.
V _{CC}		28	Voltage supply.
GND		4, 20	Ground.
V _{REF}	IN	3	External 3.3V supply voltage for the downstream differential data output buffers and the D+ pull-up.

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. Do not 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 2. I/O Register Summary

Register Name	I/O Address	Read/Write	Function	Page
Port 0 Data	0x00	R/W	GPIO Port 0 Data	14
Port 1 Data	0x01	R/W	GPIO Port 1 Data	17
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
GPIO Configuration	0x08	R/W	GPIO Port Configurations	18
I ² C Configuration	0x09	R/W	I ² C Position Configuration	20
USB Device Address A	0x10	R/W	USB Device Address A	31
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	21
Endpoint Interrupt Enable	0x21	R/W	USB Endpoint Interrupt Enables	21
Interrupt Vector	0x23	R	Pending Interrupt Vector Read/Clear	23
Timer (LSB)	0x24	R	Lower Eight Bits of Free-running Timer (1 MHz)	20
Timer (MSB)	0x25	R	Upper Four Bits of Free-running Timer	20
WDR Clear	0x26	W	Watchdog Reset Clear	13
I ² C Control & Status	0x28	R/W	I ² C Status and Control	21
I ² C Data	0x29	R/W	I ² C Data	18
Reserved	0x30		Reserved	
Reserved	0x31		Reserved	
Reserved	0x32		Reserved	
Reserved	0x38-0x3F		Reserved	
USB Device Address B	0x40	R/W	USB Device Address B (not used in 5-endpoint mode)	31
EP B0 Counter Register	0x41	R/W	USB Address B, Endpoint 0 Counter	33
EP B0 Mode Register	0x42	R/W	USB Address B, Endpoint 0 Configuration, or USB Address A, Endpoint 3 in 5-endpoint mode	32
EP B1 Counter Register	0x43	R/W	USB Address B, Endpoint 1 Counter	33
EP B1 Mode Register	0x44	R/W	USB Address B, Endpoint 1 Configuration, or USB Address A, Endpoint 4 in 5-endpoint mode	33
Hub Port Connect Status	0x48	R/W	Hub Downstream Port Connect Status	27
Hub Port Enable	0x49	R/W	Hub Downstream Ports Enable	27

Table 2. I/O Register Summary (continued)

Register Name	I/O Address	Read/Write	Function	Page
Hub Port Speed	0x4A	R/W	Hub Downstream Ports Speed	27
Hub Port Control (Ports [4:1])	0x4B	R/W	Hub Downstream Ports Control (Ports [4:1])	28
Hub Port Suspend	0x4D	R/W	Hub Downstream Port Suspend Control	30
Hub Port Resume Status	0x4E	R	Hub Downstream Ports Resume Status	30
Hub Ports SE0 Status	0x4F	R	Hub Downstream Ports SE0 Status	29
Hub Ports Data	0x50	R	Hub Downstream Ports Differential Data	29
Hub Downstream Force Low	0x51	R/W	Hub Downstream Ports Force LOW (Ports [1:4])	28
Processor Status & Control	0xFF	R/W	Microprocessor Status and Control Register	20

Instruction Set Summary

Refer to the *CYASM Assembler User's Guide* for more details. Note that conditional jump instructions (i.e., JC, JNC, JZ, JNZ) take five cycles if jump is taken, four cycles if no jump.

Table 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

Table 3. Instruction Set Summary (continued)

MNEMONIC	operand	opcode	cycles		MNEMONIC	operand	opcode	cycles
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 (or 4)
JMP	addr	80-8F	5		JNC	addr	D0-DF	5 (or 4)
CALL	addr	90-9F	10		JACC	addr	E0-EF	7
JZ	addr	A0-AF	5 (or 4)		INDEX	addr	F0-FF	14
JNZ	addr	B0-BF	5 (or 4)					

Programming Model

14-bit Program Counter

The 14-bit Program Counter (PC) allows access to up to 8 KB of PROM available with the CY7C65113C 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 23](#)).

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.

Program Memory Organization

Figure 2. Program Memory Space with Interrupt Vector Table

Address	Description
0x0000	Program execution begins here after a reset
0x0002	USB Bus Reset interrupt vector
0x0004	128- μ s timer interrupt vector
0x0006	1.024-ms timer interrupt vector
0x0008	USB address A endpoint 0 interrupt vector
0x000A	USB address A endpoint 1 interrupt vector
0x000C	USB address A endpoint 2 interrupt vector
0x000E	USB address B endpoint 0 interrupt vector
0x0010	USB address B endpoint 1 interrupt vector
0x0012	Hub interrupt vector
0x0014	Reserved
0x0016	GPIO interrupt vector
0x0018	I ² C interrupt vector
0x001A	Program Memory begins here
0x1FDF	(8 KB -32) PROM ends here (CY7C65113C)

Note that the upper 32 bytes of the 8K PROM are reserved. Therefore, user's program must not overwrite this space.

8-bit Accumulator (A)

The accumulator is the general-purpose register for the micro-controller.

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 for additional information.

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.

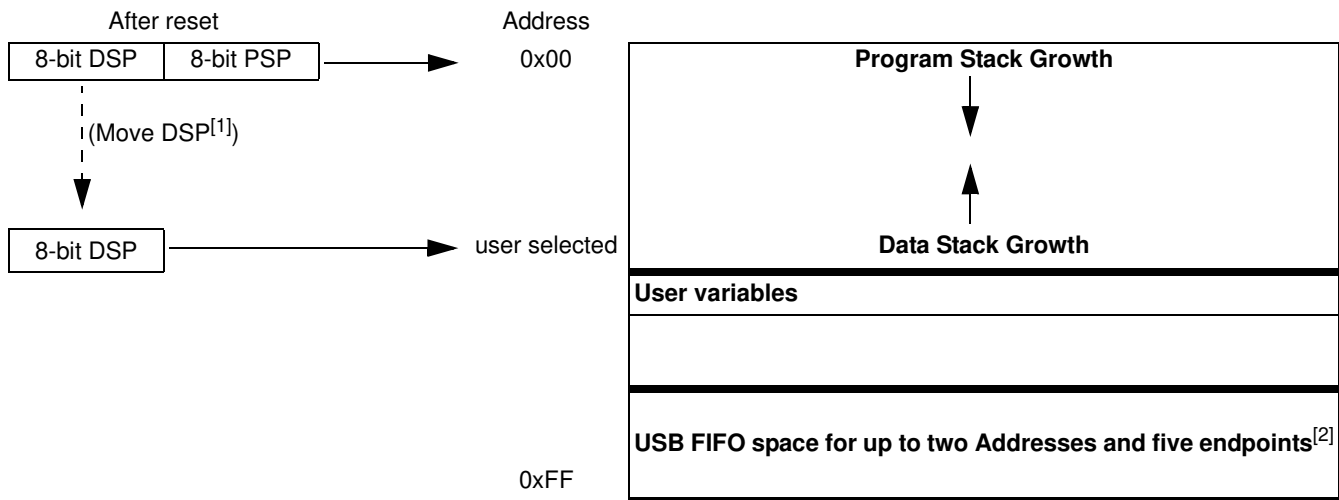
The Return From Interrupt (RETI) instruction decrements the PSP, then restores the second byte from memory addressed by the PSP. The PSP is decremented again and the first byte is restored from memory addressed by the PSP. After the program counter and flags have been restored from stack, the interrupts 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.

Data Memory Organization

The CY7C65113C 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 for a description of DSP.
2. Endpoint sizes are fixed by the Endpoint Size Bit (I/O register 0x1F, Bit 7). See Table 10.

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 17.2. 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.
```

Address Modes

The CY7C65113 microcontrollers support three addressing modes for instructions that require data operands: data, direct, and indexed.

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.

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].

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.

Clocking

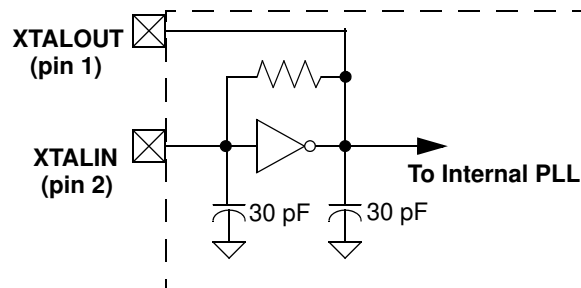


Figure 3. Clock Oscillator On-Chip Circuit

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 frequency parallel resonant crystal can be connected to these pins to provide a reference frequency for the internal PLL. The two internal 30-pF load caps appear in series to the external crystal and would be equivalent to a 15-pF load. Therefore, the crystal must have a required load capacitance of about 15–18 pF. A ceramic resonator does not allow the microcontroller to meet the timing specifications of 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.

Reset

The CY7C65113C supports two resets: POR and WDR. Each of these resets causes:

- all registers to be restored to their default states
- the USB device addresses to be set to 0
- all interrupts to be disabled
- the PSP and 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. 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.

Power-on Reset

When V_{CC} is first applied to the chip, the POR signal is asserted and the CY7C65113C 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.

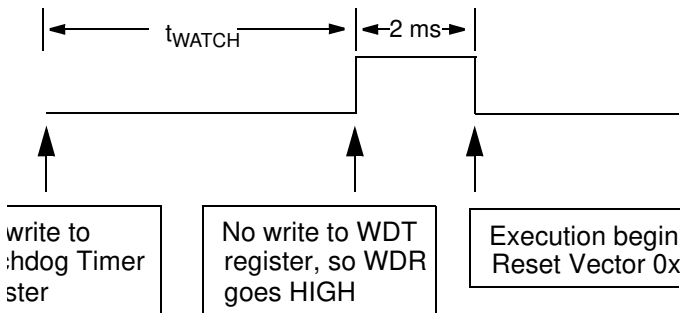
If a USB Bus Reset occurs on the upstream port during the 95 ms semi-suspend time, the semi-suspend state is aborted and program execution begins immediately from address 0x0000. In this case, the Bus Reset interrupt is pending but not serviced until firmware sets the USB Bus Reset Interrupt Enable bit (Bit 0, *Figure 18*) 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.

Watchdog Reset

The WDR occurs when the internal Watchdog Timer rolls over. Writing any value to the write-only Watchdog Reset Clear Register (Figure) clears the timer. The timer rolls over and WDR occurs if it is not cleared within t_{WATCH} of the last clear (see Section for the value of t_{WATCH}). Bit 6 of the Processor Status and Control Register (Figure 17) is set to record this event (the register contents are set to 010X0001 by the WDR). A Watchdog Timer Reset lasts for 2 ms, after which the microcontroller begins execution at ROM address 0x0000.

Figure 4. Watchdog Reset (Address 0x26)



The USB transmitter is disabled by a Watchdog Reset because the USB Device Address Registers are cleared (see Section). 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 (Figure 17) to be set following a POR event. If a firmware interrogates the Processor Status and Control Register for a set condition on the WDR bit, the WDR bit should be ignored if the POR bit is set (Bit 3 of the Processor Status and Control Register).

Suspend Mode

The CY7C65113C 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 Watchdog 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. **Note:** This also applies to internal port pins that may not be bonded in a particular package.

Typical code for entering suspend is shown below:

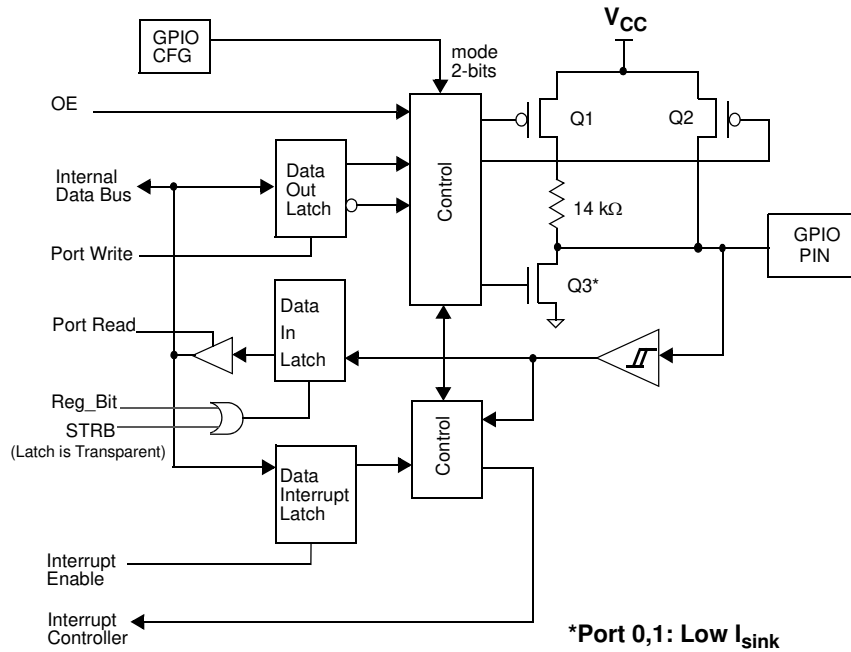
```

...           ; 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.

```

General-purpose I/O Ports

Figure 5. Block Diagram of a GPIO Pin



There are 11 GPIO pins (P0[7:0] and P1[2:0]) for the hardware interface. Each port can be configured as inputs with internal pull-ups, open drain outputs, or traditional CMOS outputs. The data for each GPIO port is accessible through the data registers. Port data registers are shown in Figure 6 through Figure 7, and are set to 1 on reset.

Figure 6. Port 0 Data.

Port 0 Data								Address 0x00
Bit #	7	6	5	4	3	2	1	0
Bit Name	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	1	1	1	1	1	1	1

Figure 7. Port1 Data

Port 1 Data						Address 0x01		
Bit #	-	-	-	-	-	2	1	0
Bit Name	-	-	-	-	-	P1.2	P1.1	P1.0
Read/Write	-	-	-	-	-	R/W	R/W	R/W
Reset	-	-	-	-	-	1	1	1

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.'

A read from a GPIO port always returns the present state of the voltage at the pin, independent of the settings in the Port Data Registers. During reset, all of the GPIO pins are set to a high-impedance input state. 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.

GPIO Configuration Port

Every GPIO port can be programmed as inputs with internal pull-ups, outputs LOW or HIGH, or Hi-Z (floating, the pin is not driven internally). In addition, the interrupt polarity for each port can be programmed. The Port Configuration bits (Figure) and the Interrupt Enable bit (Figure 10 through Figure 10) determine the interrupt polarity of the port pins

Figure 8. GPIO Configuration Register.

GPIO Configuration								Address 0x08	
Bit #	7	6	5	4	3	2	1	0	
Bit Name	Reserved	Reserved	Reserved	Reserved	Port 1 Config Bit 1	Port 1 Config Bit 0	Port 0 Config Bit 1	Port 0 Config Bit 0	
Read/Write	-	-	-	-	R/W	R/W	R/W	R/W	
Reset	-	-	-	-	0	0	0	0	

As shown in Table 4 below, a positive polarity on an input pin represents a rising edge interrupt (LOW to HIGH), and a negative polarity on an input pin represents a falling edge interrupt (HIGH to LOW).

The GPIO interrupt is generated when all of the following conditions are met: the Interrupt Enable bit of the associated Port Interrupt Enable Register is enabled, the GPIO Interrupt Enable bit of the Global Interrupt Enable Register (Figure 18) is enabled, the Interrupt Enable Sense (bit 2, Figure 17) is set, and the GPIO pin of the port sees an event matching the interrupt polarity.

The driving state of each GPIO pin is determined by the value written to the pin's Data Register (Figure 6 through Figure) and by its associated Port Configuration bits as shown in the GPIO Configuration Register (Figure). These ports are configured on a per-port basis, so all pins in a given port are configured together. The possible port configurations are detailed in Table 4. As shown in this table below, when a GPIO port is configured with CMOS outputs, interrupts from that port are disabled.

During reset, all of the bits in the GPIO Configuration Register are written with '0' to select Hi-Z mode for all GPIO ports as the default configuration.

Table 4. GPIO Port Output Control Truth Table and Interrupt Polarity

Port Config Bit 1	Port Config Bit 0	Data Register	Output Drive Strength	Interrupt Enable Bit	Interrupt Polarity
1	1	0	Output LOW	0	Disabled
		1	Resistive	1	- (Falling Edge)
1	0	0	Output LOW	0	Disabled
		1	Output HIGH	1	Disabled
0	1	0	Output LOW	0	Disabled
		1	Hi-Z	1	- (Falling Edge)
0	0	0	Output LOW	0	Disabled
		1	Hi-Z	1	+ (Rising Edge)

Q1, Q2, and Q3 discussed below are the transistors referenced in Figure . The available GPIO drive strength are:

- Output LOW Mode:** The pin's Data Register is set to '0.'
 Writing '0' to the pin's Data Register puts the pin in output LOW mode, regardless of the contents of the Port Configuration Bits[1:0]. In this mode, Q1 and Q2 are OFF. Q3 is ON. The GPIO pin is driven LOW through Q3.
- Output HIGH Mode:** The pin's Data Register is set to 1 and the Port Configuration Bits[1:0] is set to '10.'
 In this mode, Q1 and Q3 are OFF. Q2 is ON. The GPIO is pulled up through Q2. The GPIO pin is capable of sourcing... of current.
- Resistive Mode:** The pin's Data Register is set to 1 and the Port Configuration Bits[1:0] is set to '11.'
 Q2 and Q3 are OFF. Q1 is ON. The GPIO pin is pulled up with an internal 14kΩ resistor. In resistive mode, the pin may serve

as an input. Reading the pin's Data Register returns a logic HIGH if the pin is not driven LOW by an external source.

- Hi-Z Mode:** The pin's Data Register is set to 1 and Port Configuration Bits[1:0] is set either '00' or '01.'
 Q1, Q2, and Q3 are all OFF. The GPIO pin is not driven internally. In this mode, the pin may serve as an input. Reading the Port Data Register returns the actual logic value on the port pins.

GPIO Interrupt Enable Ports

Each GPIO pin can be individually enabled or disabled as an interrupt source. The Port 0–1 Interrupt Enable Registers provide this feature with an Interrupt Enable bit for each GPIO pin.

During a reset, GPIO interrupts are disabled by clearing all of the GPIO Interrupt Enable bits. 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 .

Figure 9. Port 0 Interrupt Enable

Port 0 Interrupt Enable								Address 0x04
Bit #	7	6	5	4	3	2	1	0
Bit Name	P0.7 Intr Enable	P0.6 Intr Enable	P0.5 Intr Enable	P0.4 Intr Enable	P0.3 Intr Enable	P0.2 Intr Enable	P0.1 Intr Enable	P0.0 Intr Enable
Read/Write	W	W	W	W	W	W	W	W
Reset	0	0	0	0	0	0	0	0

Figure 10. Port 1 Interrupt Enable

Port 1 Interrupt Enable								Address 0x05
Bit #	7	6	5	4	3	2	1	0
Bit Name	Reserved	Reserved	Reserved	Reserved	Reserved	P0.2 Intr Enable	P1.1 Intr Enable	P1.0 Intr Enable
Read/Write	-	-	-	-	-	W	W	W
Reset	-	-	-	-	-	0	0	0

12-bit Free-Running Timer

The 12-bit timer operates with a 1- μ s tick, provides two interrupts (128 μ s and 1.024 ms) and allows the firmware to directly time events that are up to 4 ms in duration. The lower eight bits of the timer can be read directly by the firmware. Reading the lower eight bits latches the upper four bits into a temporary register. When the firmware reads the upper four bits of the timer, it is actually reading the count stored in the temporary register. The effect of this is to ensure a stable 12-bit timer value can be read, even when the two reads are separated in time.

Figure 11. Timer LSB Register

Timer LSB								Address 0x24
Bit #	7	6	5	4	3	2	1	0
Bit Name	Timer Bit 7	TimerBit 6	Timer Bit 5	Timer Bit 4	Timer Bit 3	Timer Bit 2	Timer Bit 1	Timer Bit 0
Read/Write	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bit [7:0]: Timer lower eight bits

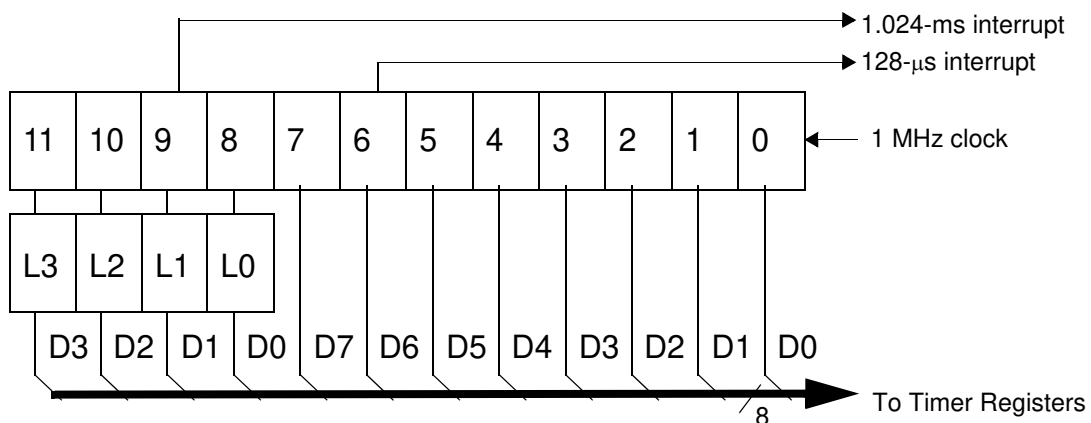
Figure 12. Timer MSB Register.

Timer MSB								Address 0x25
Bit #	7	6	5	4	3	2	1	0
Bit Name	Reserved	Reserved	Reserved	Reserved	Timer Bit 11	Timer Bit 10	Timer Bit 9	Timer Bit 8
Read/Write	–	–	–	–	R	R	R	R
Reset	0	0	0	0	0	0	0	0

Bit [3:0]: Timer higher nibble

Bit [7:4]: Reserved.

Figure 13. Timer Block Diagram



I²C Configuration Register

Internal hardware supports communication with external devices through an I²C-compatible interface. I²C-compatible function is discussed in detail in Section .^[3] The I²C Position bit (Bit 7, *Figure 14*) and I²C Port Width bit (Bit 1, *Figure 14*) select the locations of the SCL (clock) and SDA (data) pins on Port 1 as shown in *Table 5*. These bits are cleared on reset. When the GPIO is configured for I²C function, the internal pull ups on the pins are disabled. Addition of an external weak pull-up resistors on SCL and SDA is recommended.

Figure 14. I²C Configuration Register

I ² C Configuration								Address 0x09	
Bit #	7	6	5	4	3	2	1	0	
Bit Name	I ² C Position	Reserved	Reserved	Reserved	Reserved	Reserved	I ² C Port Width	Reserved	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Reset	0	0	0	0	0	0	0	0	

Table 5. I²C Port Configuration

I ² C Position (Bit7, <i>Figure 14</i>)	I ² C Port Width (Bit1, <i>Figure 14</i>)	I ² C Position
0	0	I ² C on P1[1:0], 0:SCL, 1:SDA

I²C-compatible Controller

The I²C-compatible block provides a versatile two-wire communication with external devices, supporting master, slave, and multi-master modes of operation. The I²C-compatible 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-compatible bus idle if necessary.

The I²C-compatible block generates an interrupt to the micro-controller 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 .

The I²C-compatible interface consists of two registers, an I²C Data Register (*Figure 15*) and an I²C Status and Control Register (*Figure 16*). The I²C 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 clock (SCL) is connected to bit 0 of GPIO port 1, and the I²C SDA data is connected to bit 1 GPIO port 1. The port selection is determined by settings in the I²C Port Configuration Register (Section). Once the I²C-compatible functionality is enabled by setting the I²C Enable bit of the I²C Status and Control Register (bit 0, *Figure 16*), the two LSB ([1:0]) of the corresponding GPIO port is placed in Open Drain mode, regardless of the settings of the GPIO Configuration Register. In Open Drain mode, the GPIO pin outputs LOW if the pin's Data Register is '0', and the pin is in Hi-Z mode if the pin's Data Register is '1'. The electrical characteristics of the I²C-compatible interface is the same as that of GPIO port 1. Note that the I_{OL} (max) is 2 mA @ V_{OL} = 2.0V for port 1.

All control of the I²C clock (SCL) and data (SDA) lines is performed by the I²C-compatible block.

Figure 15. I²C Data Register

I ² C Data								Address 0x29	
Bit #	7	6	5	4	3	2	1	0	
Bit Name	I ² C Data 7	I ² C Data 6	I ² C Data 5	I ² C Data 4	I ² C Data 3	I ² C Data 2	I ² C Data 1	I ² C Data 0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Reset	X	X	X	X	X	X	X	X	

Note

- I²C-compatible function must be separately enabled, as described in Section .

Bits [7..0]: I²C Data

Contains the 8-bit data on the I²C Bus

Figure 16. I²C Status and Control Register.

I ² C Status and Control								Address 0x28
Bit #	7	6	5	4	3	2	1	0
Bit Name	MSTR Mode	Continue/Busy	Xmit Mode	ACK	Addr	ARB Lost/Restart	Received Stop	I ² C Enable
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

The I²C Status and Control register bits are defined in Table 6, with a more detailed description following.

Table 6. I²C Status and Control Register Bit Definitions

Bit	Name	Description
0	I ² C Enable	When set to '1', the I ² C-compatible function is enabled. When cleared, I ² C GPIO pins operate normally.
1	Received Stop	Reads 1 only in slave receive mode, when I ² C 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 I ² C-compatible 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.

Bit 7: MSTR Mode

Setting this bit to 1 causes the I²C-compatible block 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.

Clearing this bit (set to 0) causes the GPIO pins to operate normally.

In master mode, the I²C-compatible block generates the clock (SCK), and drives the data line as required depending on transmit or receive state. The I²C-compatible block performs any required arbitration and clock synchronization. IN the event of a loss of arbitration, this MSTR bit is cleared, the ARB Lost bit is set, and an interrupt is generated by 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.

Bit 6: 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-compatible 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.

Bit 5: Xmit Mode

This bit is set by firmware to enter transmit mode and perform a data transmit in master or slave mode. Clearing this bit sets the part in 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 Re-

start bits, as these cases always cause transmit mode for the first byte.

Bit 4: 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-compatible bus. Writing a 1 to this bit generates an ACK (SDA LOW) on the I²C-compatible bus at the ACK bit time. During transmits (Xmit Mode = 1), this bit should be cleared.

Bit 3: Addr

This bit is set by the I²C-compatible 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.

Bit 2: 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.

Bit 1: 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-compatible bus, e.g., in receive mode if firmware sets the Continue bit and clears the ACK bit.

Bit 0: I²C Enable

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

Processor Status and Control Register

Figure 17. Processor Status and Control Register

Processor Status and Control							Address 0xFF	
Bit #	7	6	5	4	3	2	1	0
Bit Name	IRQ Pending	Watchdog Reset	USB Bus Reset Interrupt	Power-on Reset	Suspend	Interrupt Enable Sense	Reserved	Run
Read/Write	R	R/W	R/W	R/W	R/W	R	R/W	R/W
Reset	0	0	0	1	0	0	0	1

Bit 0: Run

This bit 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 Watchdog). This bit should normally be written as a '1.'

Bit 1: Reserved

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

Bit 2: Interrupt Enable Sense

This bit indicates 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 (Figure 18) and USB End Point Interrupt Enable Register (Figure 19). Instructions DI, EI, and RETI manipulate the state of this bit.

Bit 3: Suspend

Writing a '1' to the Suspend bit halts the processor and cause 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 USB bus activity is present. See Section for more details on suspend mode operation.

Bit 4: Power-on Reset

The Power-on Reset 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 Watchdog timeout. A POR event may be followed by a Watchdog reset before firmware begins executing, as explained below.

Bit 5: USB Bus Reset Interrupt

The USB Bus Reset Interrupt bit is set when the USB Bus Reset is detected on receiving a USB Bus Reset signal on the upstream port. The USB Bus Reset signal 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.

Bit 6: Watchdog Reset

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

Bit 7: IRQ Pending

The IRQ pending, 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 (Figure 18, Figure 19) 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), a Watchdog 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 Watchdog Reset, the Processor Status and Control Register is set to 01XX0001, which indicates a Watchdog Reset (bit 6 set) has occurred and no interrupts are pending (bit 7 clear). The Watchdog Reset does not effect the state of the POR and the Bus Reset Interrupt bits.

Interrupts

Interrupts are generated by GPIO pins, internal timers, I²C-compatible operation, internal USB hub and 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.

Figure 18. Global Interrupt Enable Register

Global Interrupt Enable Register

Address 0X20

Bit #	7	6	5	4	3	2	1	0
Bit Name	Reserved	I ² C Interrupt Enable	GPIO Interrupt Enable	Reserved	USB Hub Interrupt Enable	1.024-ms Interrupt Enable	128-μs Interrupt Enable	USB Bus RST Interrupt Enable
Read/Write	–	R/W	R/W	-	R/W	R/W	R/W	R/W
Reset	–	0	0	X	0	0	0	0

Bit 0: USB Bus RST Interrupt Enable

1 = Enable Interrupt on a USB Bus Reset; 0 = Disable interrupt on a USB Bus Reset (Refer to section).

Bit 1:128-μs Interrupt Enable

1 = Enable Timer interrupt every 128 μs; 0 = Disable Timer Interrupt for every 128 μs.

Bit 2: 1.024-ms Interrupt Enable

1 = Enable Timer interrupt every 1.024 ms; 0 = Disable Timer Interrupt every 1.024 ms.

Bit 3: USB Hub Interrupt Enable

1 = Enable Interrupt on a Hub status change; 0 = Disable interrupt due to hub status change. (Refer to section .)

Bit 4: Reserved.

Bit 5: GPIO Interrupt Enable

1 = Enable Interrupt on falling/rising edge on any GPIO; 0 = Disable Interrupt on falling/rising edge on any GPIO (Refer to section , and).

Bit 6: I²C Interrupt Enable

1 = Enable Interrupt on I2C related activity; 0 = Disable I2C related activity interrupt. (Refer to section .)

Bit 7: Reserved

Figure 19. USB Endpoint Interrupt Enable Register.

USB Endpoint Interrupt Enable

Address 0X21

Bit #	7	6	5	4	3	2	1	0
Bit Name	Reserved	Reserved	Reserved	EPB1 Interrupt Enable	EPB0 Interrupt Enable	EPA2 Interrupt Enable	EPA1 Interrupt Enable	EPA0 Interrupt Enable
Read/Write	–	–	–	R/W	R/W	R/W	R/W	R/W
Reset	–	–	–	0	0	0	0	0

Bit 0: EPA0 Interrupt Enable

- 1 = Enable Interrupt on data activity through endpoint A0;
- 0 = Disable Interrupt on data activity through endpoint A0

Bit 1: EPA1 Interrupt Enable

- 1 = Enable Interrupt on data activity through endpoint A1;
- 0 = Disable Interrupt on data activity through endpoint A1

Bit 2: EPA2 Interrupt Enable

- 1 = Enable Interrupt on data activity through endpoint A2;
- 0 = Disable Interrupt on data activity through endpoint A2.

Bit 3: EPB0 Interrupt Enable

- 1 = Enable Interrupt on data activity through endpoint B0;
- 0 = Disable Interrupt on data activity through endpoint B0

Bit 4: EPB1 Interrupt Enable

- 1 = Enable Interrupt on data activity through endpoint B1;
- 0 = Disable Interrupt on data activity through endpoint B1

Bit [7..5]: Reserved

During a reset, the contents of the Global Interrupt Enable Register and USB End Point Interrupt Enable Register are cleared, effectively disabling all interrupts,

The interrupt controller contains a separate flip-flop for each interrupt. See [Figure 20](#) 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 does the following:

1. 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, *Figure 17*).
2. Clears the flip-flop of the current interrupt.
3. Generates an automatic CALL instruction to the ROM address associated with the interrupt being serviced (i.e., the Interrupt Vector, see Section).

The instruction in the interrupt table is typically a JMP instruction to the address of the Interrupt Service Routine (ISR). The user can reenables 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.

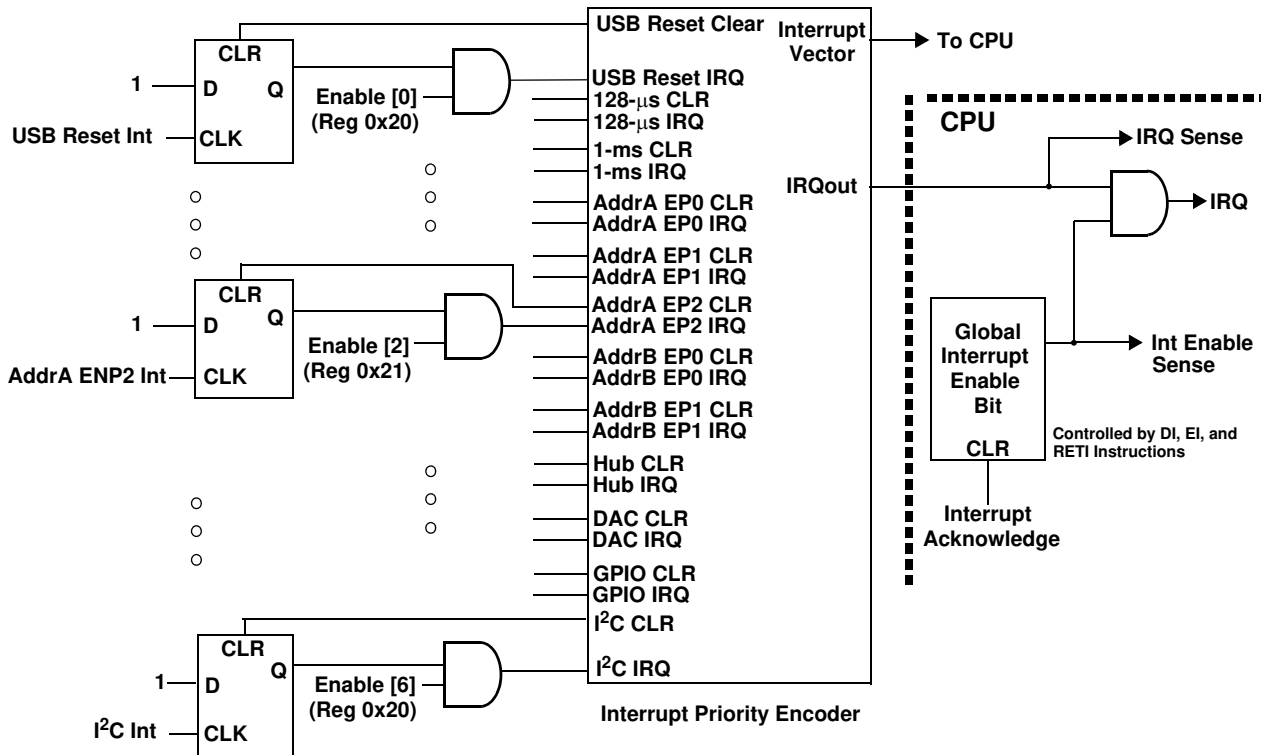
The Program Counter value as well as the Carry and Zero flags (CF, ZF) are stored onto the Program Stack by the automatic CALL instruction generated as part of the interrupt acknowledge process. The user firmware is responsible for ensuring that the processor state is preserved and restored during an 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 counters CF and ZF are restored and interrupts are enabled when the RETI instruction is executed.

The IDI and EI instruction can be used to disable and enable interrupts, respectively. These instruction 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 exits 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).

Interrupt Vectors

The Interrupt Vectors supported by the USB Controller are listed in Table 7. The lowest-numbered interrupt (USB Bus Reset interrupt) has the highest priority, and the highest-numbered interrupt (I²C interrupt) has the lowest priority.

Figure 20. Interrupt Controller Function Diagram



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 two bytes long, the interrupt vectors occupy two bytes.

Table 7. 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 B Endpoint 0 interrupt
8	0x0010	USB Address B Endpoint 1 interrupt
9	0x0012	USB Hub interrupt
10	0x0014	DAC interrupt
11	0x0016	GPIO interrupt
12	0x0018	I ² C interrupt

Interrupt Latency

Interrupt latency can be calculated from the following equation:

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

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.

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. SE0 is defined as the condition in which both the D+ line and the D– line are LOW. A USB Bus Reset may be recognized for an SE0 as short as 12 μs, but is always recognized for an SE0 longer than 16 μs. When a USB Bus Reset is detected, bit 5 of the Processor Status and Control Register (*Figure 17*) is set to record this event. In addition, the controller clears the following registers:

- SIE Section:.....USB Device Address Registers (0x10, 0x40)
- Hub Section: Hub Ports Connect Status (0x48)
- Hub Ports Enable (0x49)
- Hub Ports Speed (0x4A)
- Hub Ports Suspend (0x4D)
- Hub Ports Resume Status (0x4E)
- Hub Ports SE0 Status (0x4F)
- Hub Ports Data (0x50)
- Hub Downstream Force (0x51).

A USB Bus Reset Interrupt is generated at the end of the USB Bus Reset condition when the SE0 state is deasserted. If the USB reset occurs during the start-up delay following a POR, the delay is aborted as described in Section .

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.

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 on an IN transfer, or on the device ACK on an OUT transfer). If no ACK is received during an IN transaction, no interrupt is generated.

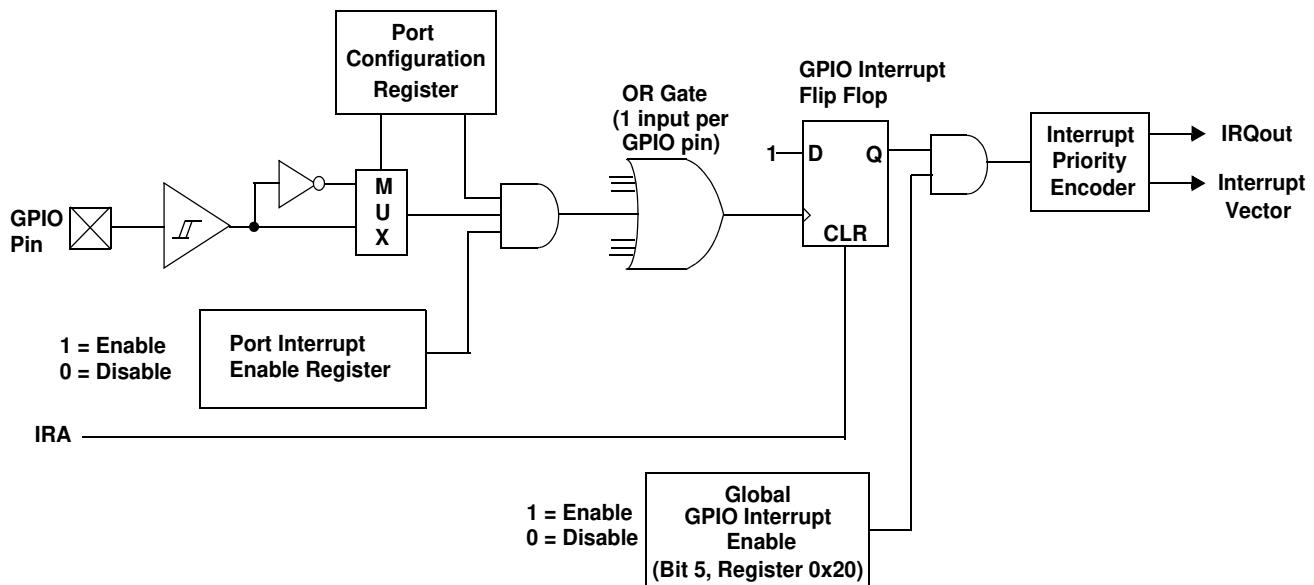
USB Hub Interrupt

A USB hub interrupt is generated by the hardware after a connect/disconnect change, babble, or a resume event is detected by the USB repeater hardware. The babble and resume events are additionally gated by the corresponding bits of the Hub Port Enable Register (*Figure 24*). The connect/disconnect event on a port does not generate an interrupt if the SIE does not drive the port (i.e., the port is being forced).

GPIO 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](#) .

Figure 21. GPIO Interrupt Structure



Refer to Sections and 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.

I²C Interrupt

The I²C interrupt occurs after various events on the I²C-compatible bus to signal the need for firmware interaction. This generally involves reading the I²C Status and Control Register (Figure 16) to determine the cause of the interrupt, loading/reading the I²C Data Register as appropriate, and finally writing the Processor Status and Control Register (Figure 17) 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 for details on the I²C registers.

When enabled, the I²C-compatible 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/Busy* 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/Busy* 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-compatible 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 *MSTR MODE* bit at the same time causes the master state machine to issue a stop signal to the I²C-compatible bus and leave the I²C-compatible hardware in the idle state.
6. When the master loses arbitration: This condition clears the *MSTR MODE* bit and sets the *ARB Lost/Restart* bit immediately and then waits for a stop signal on the I²C-compatible 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/Busy* 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/Busy* bit, without checking the value of the *Continue/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.