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

CY7C63310/CY7C638xx

enCoRe™ II Low Speed USB Peripheral Controller

Features

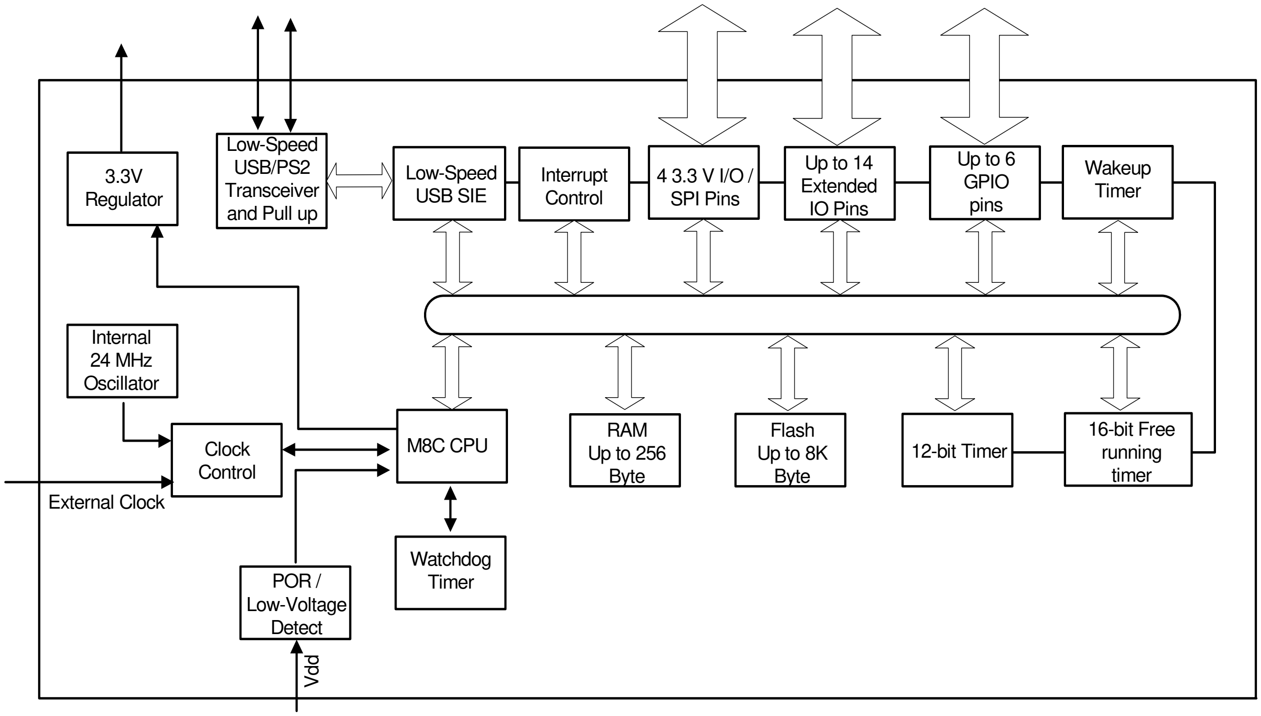
- USB 2.0-USB-IF certified (TID # 40000085)
- enCoRe™ II USB - ‘**en**hanced **Co**mponent **Re**duction’
  - Crystalless oscillator with support for an external clock. The internal oscillator eliminates the need for an external crystal or resonator.
  - Two internal 3.3 V regulators and an internal USB Pull-up resistor
  - Configurable I/O for real world interface without external components
- USB Specification compliance
  - Conforms to USB Specification, Version 2.0
  - Conforms to USB HID Specification, Version 1.1
  - Supports one low speed USB device address
  - Supports one control endpoint and two data endpoints
  - Integrated USB transceiver with dedicated 3.3 V regulator for USB signalling and D– pull-up.
- Enhanced 8-bit microcontroller
  - Harvard architecture
  - M8C CPU speed is up to 24 MHz or sourced by an external clock signal
- Internal memory
  - Up to 256 bytes of RAM
  - Up to eight Kbytes of flash including EEROM emulation
- Interface can auto configure to operate as PS/2 or USB
  - No external components for switching between PS/2 and USB modes
  - No General Purpose I/O (GPIO) pins required to manage dual mode capability
- Low power consumption
  - Typically 10 mA at 6 MHz
  - 10 µA sleep
- In system reprogrammability
  - Allows easy firmware update
- GPIO ports
  - Up to 20 GPIO pins
  - 2 mA source current on all GPIO pins. Configurable 8 or 50 mA/pin current sink on designated pins.
  - Each GPIO port supports high impedance inputs, configurable pull-up, open drain output, CMOS/TTL inputs, and CMOS output
  - Maskable interrupts on all I/O pins
- A dedicated 3.3 V regulator for the USB PHY. Aids in signalling and D– line pull-up
- 125 mA 3.3 V voltage regulator powers external 3.3 V devices
- 3.3 V I/O pins
  - 4 I/O pins with 3.3 V logic levels
  - Each 3.3 V pin supports high impedance input, internal pull-up, open drain output or traditional CMOS output
- SPI serial communication
  - Master or slave operation
  - Configurable up to 4 Mbps transfers in the master mode
  - Supports half duplex single data line mode for optical sensors
- 2-channel 8-bit or 1-channel 16-bit capture timer registers. Capture timer registers store both rising and falling edge times.
  - Two registers each for two input pins
  - Separate registers for rising and falling edge capture
  - Simplifies the interface to RF inputs for wireless applications
- Internal low power wakeup timer during suspend mode:
  - Periodic wakeup with no external components
- 12-bit Programmable Interval Timer with interrupts
- Advanced development tools based on Cypress PSoC® tools
- Watchdog timer (WDT)
- Low-voltage detection with user configurable threshold voltages
- Operating voltage from 4.0 V to 5.5 V DC
- Operating temperature from 0 °C–70 °C
- Available in 18-pin PDIP; 16, 18, and 24-pin SOIC; 24-pin QSOP, and 24-pin and 32-pin QFN Sawn packages
- Industry standard programmer support

Applications

The CY7C63310/CY7C638xx is targeted for the following applications:

- PC HID devices
  - Mice (optomechanical, optical, trackball)
- Gaming
  - Joysticks
  - Game pad
- General purpose
  - Barcode scanners
  - POS terminal
  - Consumer electronics
  - Toys
  - Remote controls
  - Security dongles

Logic Block Diagram



More Information

Cypress provides a wealth of data at www.cypress.com to help you to select the right enCoRe II device for your design, and to help you to quickly and effectively integrate the device into your design. For a comprehensive list of resources, see the product webpage http://www.cypress.com/?id=182.

- Overview: USB Portfolio, USB Roadmap
- USB Low Speed Product Selectors: enCoRe II, PRoC-LP, PRoC-LPstar
- Application notes: Cypress offers a large number of USB application notes covering a broad range of topics, from basic to advanced level. Recommended application notes for getting started with FX3 are:
  - AN6062 - enCoRe™ to enCoRe II Conversion
  - AN6075 - enCoRe™ II USB Bootloader
  - AN15482 - Using Capture Timers in enCoRe™ II and enCoRe II LV Devices
- Code Examples:
  - CE58786 - Implementing Pin Specific Interrupts in enCoRe™ II / enCoRe II LV
- User Module Datasheets:
  - User Module Datasheet: USB DEVICE DATASHEET, USB V 1.90 (CY7C639/638/633XX, CYRF69XX3)
  - User Module Datasheet: 12-Bit Programmable Interval Timer Datasheet, PITIMER12 V 1.1 (CY7C639/638/633/601/602XX, CYRF69XX3)
  - User Module Datasheet: 1 Millisecond Interval Timer Datasheet, MSTIMER V 1.2 (CY7C639/638/633/602/601XX, CYRF69XX3)
  - User Module Datasheet: SPI Master Datasheet SPIM V 1.30 (CY7C639/638/633/602/601xx, CYRF69xx3)
  - User Module Datasheet: EEPROM Datasheet E2PROM V 0.40 (CY7C633/638/639/601/602xx, CYRF69xx3)
  - User Module Datasheet: CyFi™ Star Network Protocol Stack Datasheet CYFISNP V 2.00 (CY7C601/602xx, CYRF69x13, CYRF89235, CYRF89435)
  - User Module Datasheet: SPI-based CyFi™ Transceiver Data Sheet CYFISPI (CY7C638x3, CY7C601/602xx, CYRF69103, CYRF69213)
- Development Kits:
  - CY3216 Modular Programmer Kit
  - CY3655 enCoRe™ II Development Kit
- Reference Designs:
  - CY4623 Mouse Reference Design
- Models: IBIS

PSoC Designer

PSoC Designer is the revolutionary Integrated Design Environment (IDE) that you can use to customize PSoC to meet your specific application requirements. PSoC Designer software accelerates system bring-up and time-to-market. Develop your applications using a library of pre-characterized analog and digital peripherals in a drag-and-drop design environment. Then, customize your design leveraging the dynamically generated API libraries of code. Finally, debug and test your designs with the integrated debug environment including in-circuit emulation and standard software debug features.

- Application Editor GUI for device and User Module configuration and dynamic reconfiguration
- Extensive User Module Catalog
- Integrated source code editor (C and Assembly)
- Free C compiler with no size restrictions or time limits
- Built-in Debugger
- Integrated Circuit Emulation (ICE)
- Built-in Support for Communication Interfaces:
  - Hardware and software I2C slaves and masters
  - Low/Full-speed USB 2.0
  - Up to 4 full-duplex UARTs, SPI master and slave, and Wireless

Contents

Introduction

Cypress has reinvented its leadership position in the low speed USB market with a new family of innovative microcontrollers. Introducing enCoRe II USB - '**en**hanced **Co**mponent **Re**duction.' Cypress has leveraged its design expertise in USB solutions to advance its family of low speed USB microcontrollers, which enable peripheral developers to design new products with a minimum number of components. The enCoRe II USB technology builds on the enCoRe family. The enCoRe family has an integrated oscillator that eliminates the external crystal or resonator, reducing overall cost. Also integrated into this chip are other external components commonly found in low speed USB applications, such as pull-up resistors, wakeup circuitry, and a 3.3 V regulator. Integrating these components reduces the overall system cost.

The enCoRe II is an 8-bit flash programmable microcontroller with an integrated low speed USB interface. The instruction set is optimized specifically for USB and PS/2 operations, although the microcontrollers may be used for a variety of other embedded applications.

The enCoRe II features up to 20 GPIO pins to support USB, PS/2, and other applications. The IO pins are grouped into four ports (Port 0 to 3). The pins on Port 0 and Port 1 may each be configured individually while the pins on Ports 2 and 3 are configured only as a group. Each GPIO port supports high impedance inputs, configurable pull-up, open drain output, CMOS/TTL inputs, and CMOS output with up to five pins that support a programmable drive strength of up to 50 mA sink current. GPIO Port 1 features four pins that interface at a voltage level of 3.3V. Additionally, each IO pin may be used to generate a GPIO interrupt to the microcontroller. Each GPIO port has its own GPIO interrupt vector; in addition, GPIO Port 0 has three dedicated pins that have independent interrupt vectors (P0.2–P0.4).

The enCoRe II features an internal oscillator. With the presence of USB traffic, the internal oscillator may be set to precisely tune to USB timing requirements (24 MHz ±1.5%). Optionally, an external 12 MHz or 24 MHz clock is used to provide a higher precision reference for USB operation. The clock generator provides the 12 MHz and 24 MHz clocks that remain internal to the microcontroller. The enCoRe II also has a 12-bit programmable interval timer and a 16-bit Free Running Timer with Capture Timer registers. In addition, the enCoRe II includes a Watchdog timer and a vectored interrupt controller.

The enCoRe II has up to eight Kbytes of flash for user code and up to 256 bytes of RAM for stack space and user variables.

The power on reset circuit detects logic when power is applied to the device, resets the logic to a known state, and begins executing instructions at flash address 0x0000. When power falls below a programmable trip voltage, it generates a reset or may be configured to generate an interrupt. There is a low voltage detect circuit that detects when V_{CC} drops below a programmable trip voltage. It is configurable to generate an LVD interrupt to inform the processor about the low voltage event. POR and LVD share the same interrupt. There is no separate interrupt for each. The Watchdog timer may be used to ensure the firmware never gets stalled in an infinite loop.

The microcontroller supports 22 maskable interrupts in the vectored interrupt controller. Interrupt sources include a USB bus reset, LVR/POR, a programmable interval timer, a 1.024 ms output from the free-running timer, three USB endpoints, two capture timers, four GPIO Ports, three Port 0 pins, two SPI, a 16-bit free running timer wrap, an internal sleep timer, and a bus active interrupt. The sleep timer causes periodic interrupts when enabled. The USB endpoints interrupt after a USB transaction complete is on the bus. The capture timers interrupt when a new timer value is saved because of a selected GPIO edge event. A total of seven GPIO interrupts support both TTL or CMOS thresholds. For additional flexibility on the edge sensitive GPIO pins, the interrupt polarity is programmed as rising or falling.

The free-running 16-bit timer provides two interrupt sources: the 1.024 ms outputs and the free running counter wrap interrupt. The programmable interval timer provides up to 1 μsec resolution and provides an interrupt every time it expires. These timers are used to measure the duration of an event under firmware control by reading the desired timer at the start and at the end of an event, then calculating the difference between the two values. The two 8-bit capture timer registers save a programmable 8-bit range of the free-running timer when a GPIO edge occurs on the two capture pins (P0.5, P0.6). The two 8-bit captures may be ganged into a single 16-bit capture.

The enCoRe II includes an integrated USB serial interface engine (SIE) that allows the chip to easily interface to a USB host. The hardware supports one USB device address with three endpoints.

The USB D+ and D– pins are optionally used as PS/2 SCLK and SDATA signals so that products are designed to respond to either USB or PS/2 modes of operation. The PS/2 operation is supported with internal 5 KΩ pull-up resistors on P1.0 (D+) and P1.1 (D–), and an interrupt to signal the start of PS/2 activity. In USB mode, the integrated 1.5 KΩ pull-up resistor on D– may be controlled under firmware. No external components are necessary for dual USB and PS/2 systems, and no GPIO pins need to be dedicated to switching between modes.

The enCoRe II supports in system programming by using the D+ and D– pins as the serial programming mode interface. The programming protocol is not USB.

Conventions

In this data sheet, bit positions in the registers are shaded to indicate which members of the enCoRe II family implement the bits.

	Available in all enCoRe II family members
(shaded)	CY7C638(1/2/3)3 only

Pinouts

Figure 1. Pin Diagrams

Top View

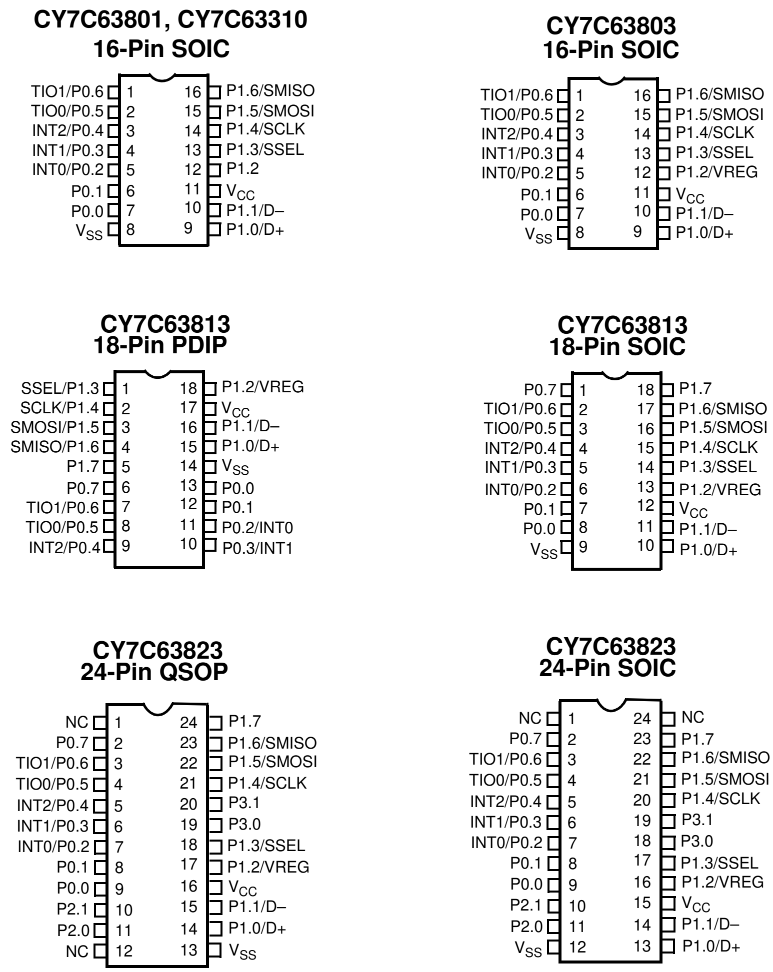


Figure 2. CY7C63803 24-Pin QFN

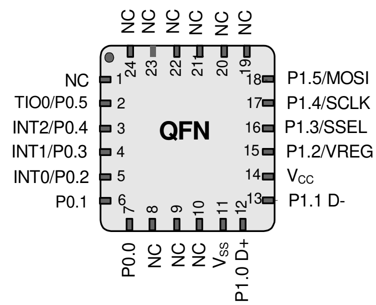
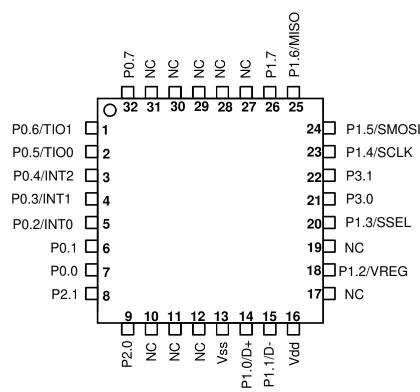


Figure 3. CY7C63833 32-Pin Sawn QFN



Pin Descriptions

32 QFN	24 QFN	24 QSOP	24 SOIC	18 SIOC	18 PDIP	16 SOIC	Name	Description
21	–	19	18	–	–	–	P3.0	**GPIO Port 3**. Configured as a group (byte).
22	–	20	19	–	–	–	P3.1	
9	–	11	11	–	–	–	P2.0	**GPIO Port 2**. Configured as a group (byte).
8	–	10	10	–	–	–	P2.1	
14	12	14	13	10	15	9	P1.0/D+	**GPIO Port 1 bit 0/USB D+** [1]**/ISSP-SCLK** If this pin is used as a General Purpose output, it draws current. This pin must be configured as an input to reduce current draw.
15	13	15	14	11	16	10	P1.1/D–	**GPIO Port 1 bit 1/USB D–** [1]**/ISSP-SDATA** If this pin is used as a General Purpose output, it draws current. This pin must be configured as an input to reduce current draw.
18	15	17	16	13	18	12	P1.2/VREG	**GPIO Port 1 bit 2**. Configured individually. 3.3V if regulator is enabled. (The 3.3 V regulator is not available in the CY7C63310 and CY7C63801.) A 1-µF min, 2-µF max capacitor is required on Vreg output.
20	16	18	17	14	1	13	P1.3/SSEL	**GPIO Port 1 bit 3**. Configured individually. Alternate function is SSEL signal of the SPI bus TTL voltage thresholds. Although Vreg is not available with the CY7C63310, 3.3 V I/O is still available.
23	17	21	20	15	2	14	P1.4/SCLK	**GPIO Port 1 bit 4**. Configured individually. Alternate function is SCLK signal of the SPI bus TTL voltage thresholds. Although Vreg is not available with the CY7C63310, 3.3 V I/O is still available.
24	18	22	21	16	3	15	P1.5/SMOSI	**GPIO Port 1 bit 5**. Configured individually. Alternate function is SMOSI signal of the SPI bus TTL voltage thresholds. Although Vreg is not available with the CY7C63310, 3.3 V I/O is still available.
25	–	23	22	17	4	16	P1.6/SMISO	**GPIO Port 1 bit 6**. Configured individually. Alternate function is SMISO signal of the SPI bus TTL voltage thresholds. Although Vreg is not available with the CY7C63310, 3.3 V I/O is still available.
26	–	24	23	18	5	–	P1.7	**GPIO Port 1 bit 7**. Configured individually. TTL voltage threshold.
7	7	9	9	8	13	7	P0.0	**GPIO Port 0 bit 0**. Configured individually. External clock input when configured as Clock In.
6	6	8	8	7	12	6	P0.1	**GPIO Port 0 bit 1**. Configured individually. Clock output when configured as Clock Out.
5	5	7	7	6	11	5	P0.2/INT0	**GPIO Port 0 bit 2**. Configured individually. Optional rising edge interrupt INT0.
4	4	6	6	5	10	4	P0.3/INT1	**GPIO Port 0 bit 3**. Configured individually. Optional rising edge interrupt INT1.
3	3	5	5	4	9	3	P0.4/INT2	**GPIO Port 0 bit 4**. Configured individually. Optional rising edge interrupt INT2.

Note
1. P1.0(D+) and P1.1(D–) pins must be in I/O mode when used as GPIO and in I_{sb} mode.

Pin Descriptions *(continued)*

32 QFN	24 QFN	24 QSOP	24 SOIC	18 SIOC	18 PDIP	16 SOIC	Name	Description
2	2	4	4	3	8	2	P0.5/TIO0	**GPIO Port 0 bit 5**. Configured individually Alternate function Timer capture inputs or Timer output TIO0
1	–	3	3	2	7	1	P0.6/TIO1	**GPIO Port 0 bit 6**. Configured individually Alternate function Timer capture inputs or Timer output TIO1
32	–	2	2	1	6	–	P0.7	**GPIO Port 0 bit 7**. Configured individually Not present in the 16 pin SOIC package
10	8	1	1	–	–	–	NC	**No connect**
11	9	12	24	–	–	–	NC	**No connect**
12	10	–	–	–	–	–	NC	**No connect**
17	20	–	–	–	–	–	NC	**No connect**
19	21	–	–	–	–	–	NC	**No connect**
27	22	–	–	–	–	–	NC	**No connect**
28	23	–	–	–	–	–	NC	**No connect**
29	24	–	–	–	–	–	NC	**No connect**
30	–	–	–	–	–	–	NC	**No connect**
31	–	–	–	–	–	–	NC	**No connect**
16	14	16	15	12	17	11	Vcc	**Supply**
13	11	13	12	9	14	8	V_{SS}	**Ground**

CPU Architecture

This family of microcontrollers is based on a high performance, 8-bit, Harvard architecture microprocessor. Five registers control the primary operation of the CPU core. These registers are affected by various instructions, but are not directly accessible through the register space by the user.

Table 1. CPU Registers and Register Names

CPU Register	Register Name
Flags	CPU_F
Program Counter	CPU_PC
Accumulator	CPU_A
Stack Pointer	CPU_SP
Index	CPU_X

The 16-bit Program Counter Register (CPU_PC) allows direct addressing of the full 8 Kbytes of program memory space.

The Accumulator Register (CPU_A) is the general purpose register, which holds the results of instructions that specify any of the source addressing modes.

The Index Register (CPU_X) holds an offset value that is used in the indexed addressing modes. Typically, this is used to address a block of data within the data memory space.

The Stack Pointer Register (CPU_SP) holds the address of the current top of the stack in the data memory space. It is affected by the PUSH, POP, LCALL, CALL, RETI, and RET instructions, which manage the software stack. It is also affected by the SWAP and ADD instructions.

The Flag Register (CPU_F) has three status bits: Zero Flag bit [1]; Carry Flag bit [2]; Supervisory State bit [3]. The Global Interrupt Enable bit [0] globally enables or disables interrupts. The user cannot manipulate the Supervisory State status bit [3]. The flags are affected by arithmetic, logic, and shift operations. The manner in which each flag is changed is dependent upon the instruction being executed, such as AND, OR, XOR, and others. See Table 18 on page 14.

CPU Registers

The CPU registers in enCoRe II devices are in two banks with 256 registers in each bank. Bit[4]/XI/O bit in the CPU Flags register must be set/cleared to select between the two register banks Table 2 on page 10.

Flags Register

The Flags Register is set or reset only with logical instruction.

Table 2. CPU Flags Register (CPU_F) [R/W]

Bit #	7	6	5	4	3	2	1	0
Field	Reserved			XIO	Super	Carry	Zero	Global IE
Read/Write	–	–	–	R/W	R	RW	RW	RW
Default	0	0	0	0	0	0	1	0

Bit [7:5]: Reserved

Bit 4: XIO

Set by the user to select between the register banks

0 = Bank 0

1 = Bank 1

Bit 3: Super

Indicates whether the CPU is executing user code or Supervisor Code. (This code cannot be accessed directly by the user.)

0 = User Code

1 = Supervisor Code

Bit 2: Carry

Set by the CPU to indicate whether there has been a carry in the previous logical/arithmetic operation.

0 = No Carry

1 = Carry

Bit 1: Zero

Set by the CPU to indicate whether there has been a zero result in the previous logical/arithmetic operation.

0 = Not Equal to Zero

1 = Equal to Zero

Bit 0: Global IE

Determines whether all interrupts are enabled or disabled

0 = Disabled

1 = Enabled

Note CPU_F register is only readable with the explicit register address 0xF7. The *OR F, expr and AND F, expr* instructions must be used to set and clear the CPU_F bits.

Table 3. CPU Accumulator Register (CPU_A)

Bit #	7	6	5	4	3	2	1	0
Field	CPU Accumulator [7:0]							
Read/Write	–	–	–	–	–	–	–	–
Default	0	0	0	0	0	0	0	0

Bit [7:0]: CPU Accumulator [7:0]

8-bit data value holds the result of any logical/arithmetic instruction that uses a source addressing mode.

Table 4. CPU X Register (CPU_X)

Bit #	7	6	5	4	3	2	1	0
Field	X [7:0]							
Read/Write	–	–	–	–	–	–	–	–
Default	0	0	0	0	0	0	0	0

Bit [7:0]: X [7:0]

8-bit data value holds an index for any instruction that uses an indexed addressing mode.

Table 5. CPU Stack Pointer Register (CPU_SP)

Bit #	7	6	5	4	3	2	1	0
Field	Stack Pointer [7:0]							
Read/Write	–	–	–	–	–	–	–	–
Default	0	0	0	0	0	0	0	0

Bit [7:0]: Stack Pointer [7:0]

8-bit data value holds a pointer to the current top of the stack.

Table 6. CPU Program Counter High Register (CPU_PCH)

Bit #	7	6	5	4	3	2	1	0
Field	Program Counter [15:8]							
Read/Write	–	–	–	–	–	–	–	–
Default	0	0	0	0	0	0	0	0

Bit [7:0]: Program Counter [15:8]

8-bit data value holds the higher byte of the program counter.

Table 7. CPU Program Counter Low Register (CPU_PCL)

Bit #	7	6	5	4	3	2	1	0
Field	Program Counter [7:0]							
Read/Write	–	–	–	–	–	–	–	–
Default	0	0	0	0	0	0	0	0

Bit [7:0]: Program Counter [7:0]

8-bit data value holds the lower byte of the program counter.

Addressing Modes

Source Immediate

The result of an instruction using this addressing mode is placed in the A register, the F register, the SP register, or the X register, which is specified as part of the instruction opcode. Operand 1 is an immediate value that serves as a source for the instruction. Arithmetic instructions require two sources; the second source is the A or the X register specified in the opcode. Instructions using this addressing mode are two bytes in length.

Table 8. Source Immediate

Opcode	Operand 1
Instruction	Immediate Value

Examples

ADD	A	7	The immediate value of 7 is added with the Accumulator and the result is placed in the Accumulator.
MOV	X	8	The immediate value of 8 is moved to the X register.
AND	F	9	The immediate value of 9 is logically ANDed with the F register and the result is placed in the F register.

Source Direct

The result of an instruction using this addressing mode is placed in either the A register or the X register, which is specified as part of the instruction opcode. Operand 1 is an address that points to a location in the RAM memory space or the register space that is the source of the instruction. Arithmetic instructions require two sources; the second source is the A register or X register specified in the opcode. Instructions using this addressing mode are two bytes in length.

Table 9. Source Direct

Opcode	Operand 1
Instruction	Source address

Examples

ADD	A	[7]	The value in the RAM memory location at address 7 is added with the Accumulator, and the result is placed in the Accumulator.
MOV	X	REG[8]	The value in the register space at address 8 is moved to the X register.

Source Indexed

The result of an instruction using this addressing mode is placed in either the A register or the X register, which is specified as part of the instruction opcode. Operand 1 is added to the X register forming an address that points to a location in the RAM memory space or the register space that is the source of the instruction. Arithmetic instructions require two sources; the second source is the A register or X register specified in the opcode. Instructions using this addressing mode are two bytes in length.

Table 10. Source Indexed

Opcode	Operand 1
Instruction	Source index

Examples

ADD	A	[X+7]	The value in the memory location at address X + 7 is added with the Accumulator, and the result is placed in the Accumulator.
MOV	X	REG[X+8]	The value in the register space at address X + 8 is moved to the X register.

Destination Direct

The result of an instruction using this addressing mode is placed within the RAM memory space or the register space. Operand 1 is an address that points to the location of the result. The source for the instruction is either the A register or the X register, which is specified as part of the instruction opcode. Arithmetic instructions require two sources; the second source is the location specified by Operand 1. Instructions using this addressing mode are two bytes in length.

Table 11. Destination Direct

Opcode	Operand 1
Instruction	Destination address

Examples

ADD	[7]	A	The value in the memory location at address 7 is added with the Accumulator, and the result is placed in the memory location at address 7. The Accumulator is unchanged.
MOV	REG[8]	A	The Accumulator is moved to the register space location at address 8. The Accumulator is unchanged.

Destination Indexed

The result of an instruction using this addressing mode is placed within the RAM memory space or the register space. Operand 1 is added to the X register forming the address that points to the location of the result. The source for the instruction is the A register. Arithmetic instructions require two sources; the second source is the location specified by Operand 1 added with the X register. Instructions using this addressing mode are two bytes in length.

Table 12. Destination Indexed

Opcode	Operand 1
Instruction	Destination index

Example

ADD	[X+7]	A	The value in the; memory location at address X+7 is added with the Accumulator, and the result is placed in the memory location at address x+7. The Accumulator is unchanged.

Destination Direct Source Immediate

The result of an instruction using this addressing mode is placed within the RAM memory space or the register space. Operand 1 is the address of the result. The source of the instruction is Operand 2, which is an immediate value. Arithmetic instructions require two sources; the second source is the location specified by Operand 1. Instructions using this addressing mode are three bytes in length.

Table 13. Destination Direct Source Immediate

Opcode	Operand 1	Operand 2
Instruction	Destination address	Immediate Value

Examples

ADD	[7]	5	The value in the memory location at address 7 is added to the immediate value of 5, and the result is placed in the memory location at address 7.
MOV	REG[8]	6	The immediate value of 6 is moved into the register space location at address 8.

Destination Indexed Source Immediate

The result of an instruction using this addressing mode is placed within the RAM memory space or the register space. Operand 1 is added to the X register to form the address of the result. The source of the instruction is Operand 2, which is an immediate value. Arithmetic instructions require two sources; the second source is the location specified by Operand 1 added with the X register. Instructions using this addressing mode are three bytes in length.

Table 14. Destination Indexed Source Immediate

Opcode	Operand 1	Operand 2
Instruction	Destination index	Immediate value

Examples

ADD	[X+7]	5	The value in the memory location at address X+7 is added with the immediate value of 5, and the result is placed in the memory location at address X+7.
MOV	REG[X+8]	6	The immediate value of 6 is moved into the location in the register space at address X+8.

Destination Direct Source Direct

The result of an instruction using this addressing mode is placed within the RAM memory. Operand 1 is the address of the result. Operand 2 is an address that points to a location in the RAM memory that is the source for the instruction. This addressing mode is only valid on the MOV instruction. The instruction using this addressing mode is three bytes in length.

Table 15. Destination Direct Source Direct

Opcode	Operand 1	Operand 2
Instruction	Destination address	Source address

Example

MOV	[7]	[8]	The value in the memory location at address 8 is moved to the memory location at address 7.

Source Indirect Post Increment

The result of an instruction using this addressing mode is placed in the Accumulator. Operand 1 is an address pointing to a location within the memory space, which contains an address (the indirect address) for the source of the instruction. The indirect address is incremented as part of the instruction execution. This addressing mode is only valid on the MVI instruction. The instruction using this addressing mode is two bytes in length. Refer to the *PSoC Designer: Assembly Language User Guide* for further details on MVI instruction.

Table 16. Source Indirect Post Increment

Opcode	Operand 1
Instruction	Source address address

Example

MVI	A	[8]	The value in the memory location at address 8 is an indirect address. The memory location pointed to by the indirect address is moved into the Accumulator. The indirect address is then incremented.

Destination Indirect Post Increment

The result of an instruction using this addressing mode is placed within the memory space. Operand 1 is an address pointing to a location within the memory space, which contains an address (the indirect address) for the destination of the instruction. The indirect address is incremented as part of the instruction execution. The source for the instruction is the Accumulator. This addressing mode is only valid on the MVI instruction. The instruction using this addressing mode is two bytes in length.

Table 17. Destination Indirect Post Increment

Opcode	Operand 1
Instruction	Destination address address

Example

MVI	[8]	A	The value in the memory location at address 8 is an indirect address. The Accumulator is moved into the memory location pointed to by the indirect address. The indirect address is then incremented.

Instruction Set Summary

The instruction set is summarized in Table 18 numerically and serves as a quick reference. If more information is needed, the Instruction Set Summary tables are described in detail in the *PSoC Designer Assembly Language User Guide* (available on the Cypress web site at http://www.cypress.com/?docID=15538).

Table 18. Instruction Set Summary Sorted Numerically by Opcode Order [2, 3]

Opcode Hex	Cycles	Bytes	Instruction Format	Flags	Opcode Hex	Cycles	Bytes	Instruction Format	Flags	Opcode Hex	Cycles	Bytes	Instruction Format	Flags
00	15	1	SSC	–	2D	8	2	OR [X+expr], A	Z	5A	5	2	MOV [expr], X	–
01	4	2	ADD A, expr	C, Z	2E	9	3	OR [expr], expr	Z	5B	4	1	MOV A, X	Z
02	6	2	ADD A, [expr]	C, Z	2F	10	3	OR [X+expr], expr	Z	5C	4	1	MOV X, A	–
03	7	2	ADD A, [X+expr]	C, Z	30	9	1	HALT	–	5D	6	2	MOV A, reg[expr]	Z
04	7	2	ADD [expr], A	C, Z	31	4	2	XOR A, expr	Z	5E	7	2	MOV A, reg[X+expr]	Z
05	8	2	ADD [X+expr], A	C, Z	32	6	2	XOR A, [expr]	Z	5F	10	3	MOV [expr], [expr]	–
06	9	3	ADD [expr], expr	C, Z	33	7	2	XOR A, [X+expr]	Z	60	5	2	MOV reg[expr], A	–
07	10	3	ADD [X+expr], expr	C, Z	34	7	2	XOR [expr], A	Z	61	6	2	MOV reg[X+expr], A	–
08	4	1	PUSH A	–	35	8	2	XOR [X+expr], A	Z	62	8	3	MOV reg[expr], expr	–
09	4	2	ADC A, expr	C, Z	36	9	3	XOR [expr], expr	Z	63	9	3	MOV reg[X+expr], expr	–
0A	6	2	ADC A, [expr]	C, Z	37	10	3	XOR [X+expr], expr	Z	64	4	1	ASL A	C, Z
0B	7	2	ADC A, [X+expr]	C, Z	38	5	2	ADD SP, expr	–	65	7	2	ASL [expr]	C, Z
0C	7	2	ADC [expr], A	C, Z	39	5	2	CMP A, expr	if (A=B) Z=1 if (A<B) C=1	66	8	2	ASL [X+expr]	C, Z
0D	8	2	ADC [X+expr], A	C, Z	3A	7	2	CMP A, [expr]		67	4	1	ASR A	C, Z
0E	9	3	ADC [expr], expr	C, Z	3B	8	2	CMP A, [X+expr]		68	7	2	ASR [expr]	C, Z
0F	10	3	ADC [X+expr], expr	C, Z	3C	8	3	CMP [expr], expr		69	8	2	ASR [X+expr]	C, Z
10	4	1	PUSH X	–	3D	9	3	CMP [X+expr], expr		6A	4	1	RLC A	C, Z
11	4	2	SUB A, expr	C, Z	3E	10	2	MVI A, [[expr]++]	Z	6B	7	2	RLC [expr]	C, Z
12	6	2	SUB A, [expr]	C, Z	3F	10	2	MVI [[expr]++], A	–	6C	8	2	RLC [X+expr]	C, Z
13	7	2	SUB A, [X+expr]	C, Z	40	4	1	NOP	–	6D	4	1	RRC A	C, Z
14	7	2	SUB [expr], A	C, Z	41	9	3	AND reg[expr], expr	Z	6E	7	2	RRC [expr]	C, Z
15	8	2	SUB [X+expr], A	C, Z	42	10	3	AND reg[X+expr], expr	Z	6F	8	2	RRC [X+expr]	C, Z
16	9	3	SUB [expr], expr	C, Z	43	9	3	OR reg[expr], expr	Z	70	4	2	AND F, expr	C, Z
17	10	3	SUB [X+expr], expr	C, Z	44	10	3	OR reg[X+expr], expr	Z	71	4	2	OR F, expr	C, Z
18	5	1	POP A	Z	45	9	3	XOR reg[expr], expr	Z	72	4	2	XOR F, expr	C, Z
19	4	2	SBB A, expr	C, Z	46	10	3	XOR reg[X+expr], expr	Z	73	4	1	CPL A	Z
1A	6	2	SBB A, [expr]	C, Z	47	8	3	TST [expr], expr	Z	74	4	1	INC A	C, Z
1B	7	2	SBB A, [X+expr]	C, Z	48	9	3	TST [X+expr], expr	Z	75	4	1	INC X	C, Z
1C	7	2	SBB [expr], A	C, Z	49	9	3	TST reg[expr], expr	Z	76	7	2	INC [expr]	C, Z
1D	8	2	SBB [X+expr], A	C, Z	4A	10	3	TST reg[X+expr], expr	Z	77	8	2	INC [X+expr]	C, Z
1E	9	3	SBB [expr], expr	C, Z	4B	5	1	SWAP A, X	Z	78	4	1	DEC A	C, Z
1F	10	3	SBB [X+expr], expr	C, Z	4C	7	2	SWAP A, [expr]	Z	79	4	1	DEC X	C, Z
20	5	1	POP X	–	4D	7	2	SWAP X, [expr]	–	7A	7	2	DEC [expr]	C, Z
21	4	2	AND A, expr	Z	4E	5	1	SWAP A, SP	Z	7B	8	2	DEC [X+expr]	C, Z
22	6	2	AND A, [expr]	Z	4F	4	1	MOV X, SP	–	7C	13	3	LCALL	–
23	7	2	AND A, [X+expr]	Z	50	4	2	MOV A, expr	Z	7D	7	3	LJMP	–
24	7	2	AND [expr], A	Z	51	5	2	MOV A, [expr]	Z	7E	10	1	RETI	C, Z
25	8	2	AND [X+expr], A	Z	52	6	2	MOV A, [X+expr]	Z	7F	8	1	RET	–
26	9	3	AND [expr], expr	Z	53	5	2	MOV [expr], A	–	8x	5	2	JMP	–
27	10	3	AND [X+expr], expr	Z	54	6	2	MOV [X+expr], A	–	9x	11	2	CALL	–
28	11	1	ROMX	Z	55	8	3	MOV [expr], expr	–	Ax	5	2	JZ	–
29	4	2	OR A, expr	Z	56	9	3	MOV [X+expr], expr	–	Bx	5	2	JNZ	–
2A	6	2	OR A, [expr]	Z	57	4	2	MOV X, expr	–	Cx	5	2	JC	–
2B	7	2	OR A, [X+expr]	Z	58	6	2	MOV X, [expr]	–	Dx	5	2	JNC	–
2C	7	2	OR [expr], A	Z	59	7	2	MOV X, [X+expr]	–	Ex	7	2	JACC	–
										Fx	13	2	INDEX	Z

Notes
2. Interrupt routines take 13 cycles before execution resumes at interrupt vector table.
3. The number of cycles required by an instruction is increased by one for instructions that span 256 byte boundaries in the flash memory space.

Memory Organization

Flash Program Memory Organization

Figure 4. Program Memory Space with Interrupt Vector Table

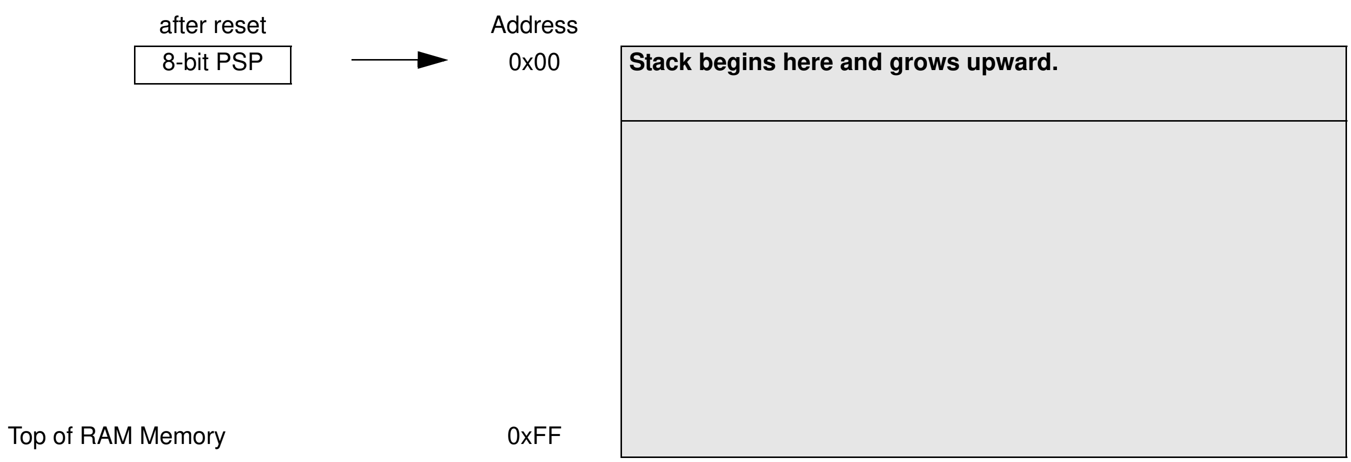
after reset: 16-bit PC → 0x0000

Address	
0x0000	Program execution begins here after a reset
0x0004	POR/LVD
0x0008	INT0
0x000C	SPI transmitter empty
0x0010	SPI receiver full
0x0014	GPIO port 0
0x0018	GPIO port 1
0x001C	INT1
0x0020	EP0
0x0024	EP1
0x0028	EP2
0x002C	USB reset
0x0030	USB active
0x0034	1 ms interval timer
0x0038	Programmable interval timer
0x003C	Timer capture 0
0x0040	Timer capture 1
0x0044	16-bit free running timer wrap
0x0048	INT2
0x004C	PS2 data low
0x0050	GPIO port 2
0x0054	GPIO port 3
0x0058	Reserved
0x005C	Reserved
0x0060	Reserved
0x0064	Sleep timer
0x0068	**Program Memory begins here (if below interrupts not used, program memory can start lower)**
0x0BFF	**3 KB ends here (CY7C63310)**
0x0FFF	**4 KB ends here (CY7C63801)**
0x1FFF	**8 KB ends here (CY7C638x3)**

Data Memory Organization

The CY7C63310/638xx microcontrollers provide up to 256 bytes of data RAM.

Figure 5. Data Memory Organization



Flash

This section describes the flash block of the enCoRe II. Much of the user visible flash functionality including programming and security are implemented in the M8C Supervisory Read Only Memory (SROM). The enCoRe II flash has an endurance of 1000 cycles and a 10 year data retention capability.

Flash Programming and Security

All flash programming is performed by code in the SROM. The registers that control the flash programming are only visible to the M8C CPU when it executes out of SROM. This makes it impossible to read, write or erase the flash by bypassing the security mechanisms implemented in the SROM.

Customer firmware can program the flash only through SROM calls. The data or code images are sourced through any interface with the appropriate support firmware. This type of programming requires a 'boot-loader', which is a piece of firmware resident on the flash. For safety reasons this boot-loader must not be overwritten during firmware rewrites.

The flash provides four extra auxiliary rows that are used to hold flash block protection flags, boot time calibration values, configuration tables, and any device values. The routines for accessing these auxiliary rows are documented in the section SROM on page 16 section. The auxiliary rows are not affected by the device erase function.

In System Programming

Most designs that include an enCoRe II part have a USB connector attached to the USB D+ and D– pins on the device. These designs require the ability to program or reprogram a part through the USB D+ and D– pins alone.

The enCoRe II devices enable this type of in system programming by using the D+ and D– pins as the serial programming mode interface. This allows an external controller to enable the enCoRe II part to enter the serial programming mode, and then use the test queue to issue flash access functions in the SROM. The programming protocol is not USB.

SROM

The SROM holds code that boots the part, calibrates circuitry, and performs flash operations (Table 19 on page 16 lists the SROM functions). The functions of the SROM are accessed in the normal user code or operating from flash. The SROM exists in a separate memory space from the user code. The SROM functions are accessed by executing the Supervisory System Call instruction (SSC), which has an opcode of 00h. Before executing the SSC the M8C's accumulator must be loaded with the desired SROM function code from Table 19 on page 16. Undefined functions cause a HALT if called from the user code. The SROM functions are executing code with calls; as a result, the functions require stack space. With the exception of Reset, all of the SROM functions have a *parameter block* in SRAM that must be configured before executing the SSC. Table 20 on page 17 lists all possible parameter block variables. The meaning of each parameter, with regards to a specific SROM function, is described later in this section.

Table 19. SROM Function Codes

Function Code	Function Name	Stack Space
00h	SWBootReset	0
01h	ReadBlock	7
02h	WriteBlock	10
03h	EraseBlock	9
05h	EraseAll	11
06h	TableRead	3
07h	CheckSum	3

Two important variables that are used for all functions are KEY1 and KEY2. These variables are used to help discriminate between valid SSCs and inadvertent SSCs. KEY1 must always have a value of 3Ah, while KEY2 must have the same value as the stack pointer when the SROM function begins execution. This would be the Stack Pointer value when the SSC opcode is executed, plus three. If either of the keys do not match the expected values, the M8C halts (with the exception of the SWBootReset function). The following code puts the correct value in KEY1 and KEY2. The code starts with a halt, to force the program to jump directly into the setup code and not run into it.

```
halt
SSCOP: mov [KEY1], 3ah
mov X, SP
mov A, X
add A, 3
mov [KEY2], A
```

Table 20. SROM Function Parameters

Variable Name	SRAM Address
Key1/Counter/Return Code	0,F8h
Key2/TMP	0,F9h
BlockID	0,FAh
Pointer	0,FBh
Clock	0,FCh
Mode	0,FDh
Delay	0,FEh
PCL	0,FFh

Return Codes

The SROM also features Return Codes and Lockouts.

Return codes aid in the determination of the success or failure of a particular function. The return code is stored in KEY1's position in the parameter block. The CheckSum and TableRead functions do not have return codes because KEY1's position in the parameter block is used to return other data.

Table 21. SROM Return Codes

Return Code	Description
00h	Success
01h	Function not allowed due to level of protection on block.
02h	Software reset without hardware reset.
03h	Fatal error, SROM halted.

Read, write, and erase operations may fail if the target block is read or write protected. Block protection levels are set during device programming.

The EraseAll function overwrites data in addition to leaving the entire user flash in the erase state. The EraseAll function loops through the number of flash macros in the product, executing the following sequence: erase, bulk program all zeros, erase. After all the user space in all the flash macros are erased, a second loop erases and then programs each protection block with zeros.

SROM Function Descriptions

SWBootReset Function

The SROM function, SWBootReset, is the function that is responsible for transitioning the device from a reset state to running user code. The SWBootReset function is executed whenever the SROM is entered with an M8C accumulator value of 00h: the SRAM parameter block is not used as an input to the function. This happens by design after a hardware reset, because the M8C's accumulator is reset to 00h or when the user code executes the SSC instruction with an accumulator value of 00h. The SWBootReset function is not executed when the SSC instruction is executed with a bad key value and a non-zero function code. An enCoRe II device executes the HALT instruction if a bad value is given for either KEY1 or KEY2.

The SWBootReset function verifies the integrity of the calibration data by way of a 16-bit checksum, before releasing the M8C to run user code.

ReadBlock Function

The ReadBlock function is used to read 64 contiguous bytes from flash: a block.

This function first checks the protection bits and determines if the desired BLOCKID is readable. If the read protection is turned on, the ReadBlock function exits setting the accumulator and KEY2 back to 00h. KEY1 has a value of 01h, indicating a read failure. If read protection is not enabled, the function reads 64 bytes from the flash using a ROMX instruction and stores the results in the SRAM using an MVI instruction. The first of the 64 bytes are stored in the SRAM at the address indicated by the value of the POINTER parameter. When the ReadBlock completes successfully, the accumulator, KEY1, and KEY2 all have a value of 00h.

Table 22. ReadBlock Parameters

Name	Address	Description
KEY1	0,F8h	3Ah
KEY2	0,F9h	Stack Pointer value, when SSC is executed.
BLOCKID	0,FAh	flash block number
POINTER	0,FBh	First of 64 addresses in SRAM where returned data must be stored.

WriteBlock Function

The WriteBlock function is used to store data in the flash. Data is moved 64 bytes at a time from SRAM to flash using this function. The WriteBlock function first checks the protection bits and determines if the desired BLOCKID is writable. If write protection is turned on, the WriteBlock function exits setting the accumulator and KEY2 back to 00h. KEY1 has a value of 01h, indicating a write failure. The configuration of the WriteBlock function is straightforward. The BLOCKID of the flash block, where the data is stored, must be determined and stored at SRAM address FAh.

The SRAM address of the first of the 64 bytes to be stored in flash must be indicated using the POINTER variable in the parameter block (SRAM address FBh). Finally, the CLOCK and DELAY value must be set correctly. The CLOCK value determines the length of the write pulse that is used to store the data in the flash. The CLOCK and DELAY values are dependent on the CPU speed and must be set correctly.

Table 23. WriteBlock Parameters

Name	Address	Description
KEY1	0,F8h	3Ah
KEY2	0,F9h	Stack Pointer value, when SSC is executed.
BLOCKID	0,FAh	8KB flash block number (00h–7Fh) 4KB flash block number (00h–3Fh) 3KB flash block number (00h–2Fh)
POINTER	0,FBh	First of 64 addresses in SRAM, where the data to be stored in flash is located before calling WriteBlock.
CLOCK	0,FCh	Clock divider used to set the write pulse width.
DELAY	0,FEh	For a CPU speed of 12 MHz set to 56h.

EraseBlock Function

The EraseBlock function is used to erase a block of 64 contiguous bytes in flash. The EraseBlock function first checks the protection bits and determines if the desired BLOCKID is writable. If write protection is turned on, the EraseBlock function exits setting the accumulator and KEY2 back to 00h. KEY1 has a value of 01h, indicating a write failure. The EraseBlock function is only useful as the first step in programming. When a block is erased, the data in the block is not one hundred percent unreadable. If the objective is to obliterate data in a block, the best method is to perform an EraseBlock followed by a WriteBlock of all zeros.

To set up the parameter block for the EraseBlock function, correct key values must be stored in KEY1 and KEY2. The block number to be erased must be stored in the BLOCKID variable and the CLOCK and DELAY values must be set based on the current CPU speed.

Table 24. EraseBlock Parameters

Name	Address	Description
KEY1	0,F8h	3Ah
KEY2	0,F9h	Stack Pointer value, when SSC is executed.
BLOCKID	0,FAh	8KB flash block number (00h–7Fh) 4KB flash block number (00h–3Fh) 3KB flash block number (00h–2Fh)
CLOCK	0,FCh	Clock divider used to set the erase pulse width.
DELAY	0,FEh	For a CPU speed of 12 MHz set to 56h

ProtectBlock Function

The enCoRe II devices offer flash protection on a block by block basis. Table 25 lists the protection modes available. In this table, ER and EW indicate the ability to perform external reads and writes. For internal writes, IW is used. Internal reading is permitted by way of the ROMX instruction. The ability to read by way of the SROM ReadBlock function is indicated by SR. The protection level is stored in two bits according to Table 25. These bits are bit packed into the 64 bytes of the protection block. As a result, each protection block byte stores the protection level for four flash blocks. The bits are packed into a byte, with the lowest numbered block's protection level stored in the lowest numbered bits Table 25.

The first address of the protection block contains the protection level for blocks 0 through 3; the second address is for blocks 4 through 7. The 64th byte stores the protection level for blocks 252 through 255.

Table 25. Protection Modes

Mode	Settings	Description	Marketing
00b	SR ER EW IW	Unprotected	Unprotected
01b	$\overline{SR}$ $\overline{ER}$ EW IW	Read protect	Factory upgrade
10b	$\overline{SR}$ $\overline{ER}$ $\overline{EW}$ $\overline{IW}$	Disable external write	Field upgrade
11b	$\overline{SR}$ $\overline{ER}$ $\overline{EW}$ $\overline{IW}$	Disable internal write	Full protection

7	6	5	4	3	2	1	0
Block n+3		Block n+2		Block n+1		Block n	

The level of protection is only decreased by an EraseAll, which places zeros in all locations of the protection block. To set the level of protection, the ProtectBlock function is used. This function takes data from SRAM, starting at address 80h, and ORs it with the current values in the protection block. The result of the OR operation is then stored in the protection block. The EraseBlock function does not change the protection level for a block. Because the SRAM location for the protection data is fixed and there is only one protection block per flash macro, the ProtectBlock function expects very few variables in the parameter block to be set before calling the function. The parameter block values that must be set, besides the keys, are the CLOCK and DELAY values.

Table 26. ProtectBlock Parameters

Name	Address	Description
KEY1	0,F8h	3Ah
KEY2	0,F9h	Stack Pointer value when SSC is executed.
CLOCK	0,FCh	Clock divider used to set the write pulse width.
DELAY	0,FEh	For a CPU speed of 12 MHz set to 56h.

EraseAll Function

The EraseAll function performs a series of steps that destroy the user data in the flash macros and resets the protection block in each flash macro to all zeros (the unprotected state). The EraseAll function does not affect the three hidden blocks above the protection block, in each flash macro. The first of these four hidden blocks is used to store the protection table for its eight Kbytes of user data.

The EraseAll function begins by erasing the user space of the flash macro with the highest address range. A bulk program of all zeros is then performed on the same flash macro, to destroy all traces of the previous contents. The bulk program is followed by a second erase that leaves the flash macro in a state ready for writing. The erase, program, erase sequence is then performed on the next lowest flash macro in the address space if it exists. After the erase of the user space, the protection block for the flash macro with the highest address range is erased. Following the erase of the protection block, zeros are written into every bit of the protection table. The next lowest flash macro in the address space then has its protection block erased and filled with zeros.

The end result of the EraseAll function is that all user data in the flash is destroyed and the flash is left in an unprogrammed state, ready to accept one of the various write commands. The protection bits for all user data are also reset to the zero state

The parameter block values that must be set, besides the keys, are the CLOCK and DELAY values.

Table 27. EraseAll Parameters

Name	Address	Description
KEY1	0,F8h	3Ah
KEY2	0,F9h	Stack Pointer value when SSC is executed.
CLOCK	0,FCh	Clock divider used to set the write pulse width.
DELAY	0,FEh	For a CPU speed of 12 MHz set to 56h

TableRead Function

The TableRead function gives the user access to part specific data stored in the flash during manufacturing. It also returns a Revision ID for the die (not to be confused with the Silicon ID).

Table 28. Table Read Parameters

Name	Address	Description
KEY1	0,F8h	3Ah
KEY2	0,F9h	Stack Pointer value when SSC is executed.
BLOCKID	0,FAh	Table number to read.

The table space for the enCoRe II is simply a 64 byte row broken up into eight tables of eight bytes. The tables are numbered zero through seven. All user and hidden blocks in the CY7C638xx parts consist of 64 bytes.

An internal table (Table 0) holds the Silicon ID and returns the Revision ID. The Silicon ID is returned in SRAM, while the Revision and Family IDs are returned in the CPU_A and CPU_X registers. The Silicon ID is a value placed in the table by programming the flash and is controlled by Cypress Semiconductor Product Engineering. The Revision ID is hard coded into the SROM and also redundantly placed in SROM Table 1. This is discussed in more detail later in this section.

SROM Table 1 holds Family/Die ID and Revision ID values for the device and returns a one-byte internal revision counter. The internal revision counter starts out with a value of zero and is incremented when one of the other revision numbers is not incremented. It is reset to zero when one of the other revision numbers is incremented. The internal revision count is returned in the CPU_A register. The CPU_X register is always set to FFh when Table 1 is read. The CPU_A and CPU_X registers always return a value of FFh when Tables 2-7 are read. The BLOCKID value, in the parameter block, indicates which table must be returned to the user. Only the three least significant bits of the BLOCKID parameter are used by TableRead function for enCoRe II devices. The upper five bits are ignored. When the function is called, it transfers bytes from the table to SRAM addresses F8h–FFh.

The M8C's A and X registers are used by the TableRead function to return the die's Revision ID. The Revision ID is a 16-bit value hard coded into the SROM that uniquely identifies the die's design.

The return values for corresponding Table calls are tabulated as shown in Table 29 on page 19.

Table 29. Return values for Table Read

Table Number	Return Value	
	A	X
0	Revision ID	Family ID
1	Internal revision counter	0xFF
2-7	0xFF	0xFF

Figure 6. SROM Table

	F8h	F9h	FAh	FBh	FCh	FDh	FEh	FFh
Table0	Silicon ID [15-8]	Silicon ID [7-0]						
Table1	Family/ Die ID	Revision ID						
Table2								
Table3								
Table4								
Table5								
Table6								
Table7								

The Silicon IDs for enCoRe II devices are stored in SROM tables in the part, as shown in Figure 6.

The Silicon ID can be read out from the part using SROM Table reads (Table 0). This is demonstrated in the following pseudo code. As mentioned in the section SROM on page 16, the SROM variables occupy address F8h through FFh in the SRAM. Each of the variables and their definition is given in the section SROM on page 16.

```
AREA SSCParmBlkA(RAM,ABS)

    org  F8h // Variables are defined starting at address F8h

SSC_KEY1:                 ; F8h  supervisory key
SSC_RETURNCODE:     blk 1 ; F8h  result code
SSC_KEY2 :          blk 1  ;F9h  supervisory stack ptr key
SSC_BLOCKID:        blk 1 ; FAh  block ID
SSC_POINTER:        blk 1 ; FBh  pointer to data buffer
SSC_CLOCK:          blk 1 ; FCh  Clock
SSC_MODE:           blk 1 ; FDh  ClockW ClockE multiplier
SSC_DELAY:          blk 1 ; FEh  flash macro sequence delay count
SSC_WRITE_ResultCode: blk 1 ; FFh  temporary result code

_main:
           mov   A, 0
           mov   [SSC_BLOCKID], A// To read from Table 0 - Silicon ID is stored in Table 0
//Call SROM operation to read the SROM table
          mov   X, SP      ; copy SP into X
          mov   A, X       ; A temp stored in X
          add   A, 3               ; create 3 byte stack frame (2 + pushed A)
          mov   [SSC_KEY2], A      ; save stack frame for supervisory code

   ; load the supervisory code for flash operations
          mov   [SSC_KEY1], 3Ah  ;flash_OPER_KEY - 3Ah

          mov   A,6        ; load A with specific operation. 06h is the code for Table read Table 19
          SSC              ; SSC call the supervisory ROM

// At the end of the SSC command the silicon ID is stored in F8 (MSB) and F9(LSB) of the SRAM

.terminate:
    jmp .terminate
```

Checksum Function

The Checksum function calculates a 16-bit checksum over a user specifiable number of blocks, within a single flash macro (Bank) starting from block zero. The BLOCKID parameter is used to pass in the number of blocks to calculate the checksum over. A BLOCKID value of 1 calculates the checksum of only block 0, while a BLOCKID value of 0 calculates the checksum of all 256 user blocks. The 16-bit checksum is returned in KEY1 and KEY2. The parameter KEY1 holds the lower eight bits of the checksum and the parameter KEY2 holds the upper eight bits of the checksum.

The checksum algorithm executes the following sequence of three instructions over the number of blocks times 64 to be checksummed.

```
romx
add [KEY1], A
adc [KEY2], 0
```

Table 30. Checksum Parameters

Name	Address	Description
KEY1	0,F8h	3Ah
KEY2	0,F9h	Stack Pointer value when SSC is executed.
BLOCKID	0,FAh	Number of flash blocks to calculate checksum on.

Clocking

The enCoRe II has two internal oscillators, the Internal 24 MHz Oscillator and the 32 kHz Low power Oscillator.

The Internal 24 MHz Oscillator is designed such that it may be trimmed to an output frequency of 24 MHz over temperature and voltage variation. With the presence of USB traffic, the Internal 24 MHz Oscillator may be set to precisely tune to the USB timing requirements (24 MHz ± 1.5%). Without USB traffic, the Internal 24 MHz Oscillator accuracy is 24 MHz ± 5% (between 0 °C–70 °C). No external components are required to achieve this level of accuracy.

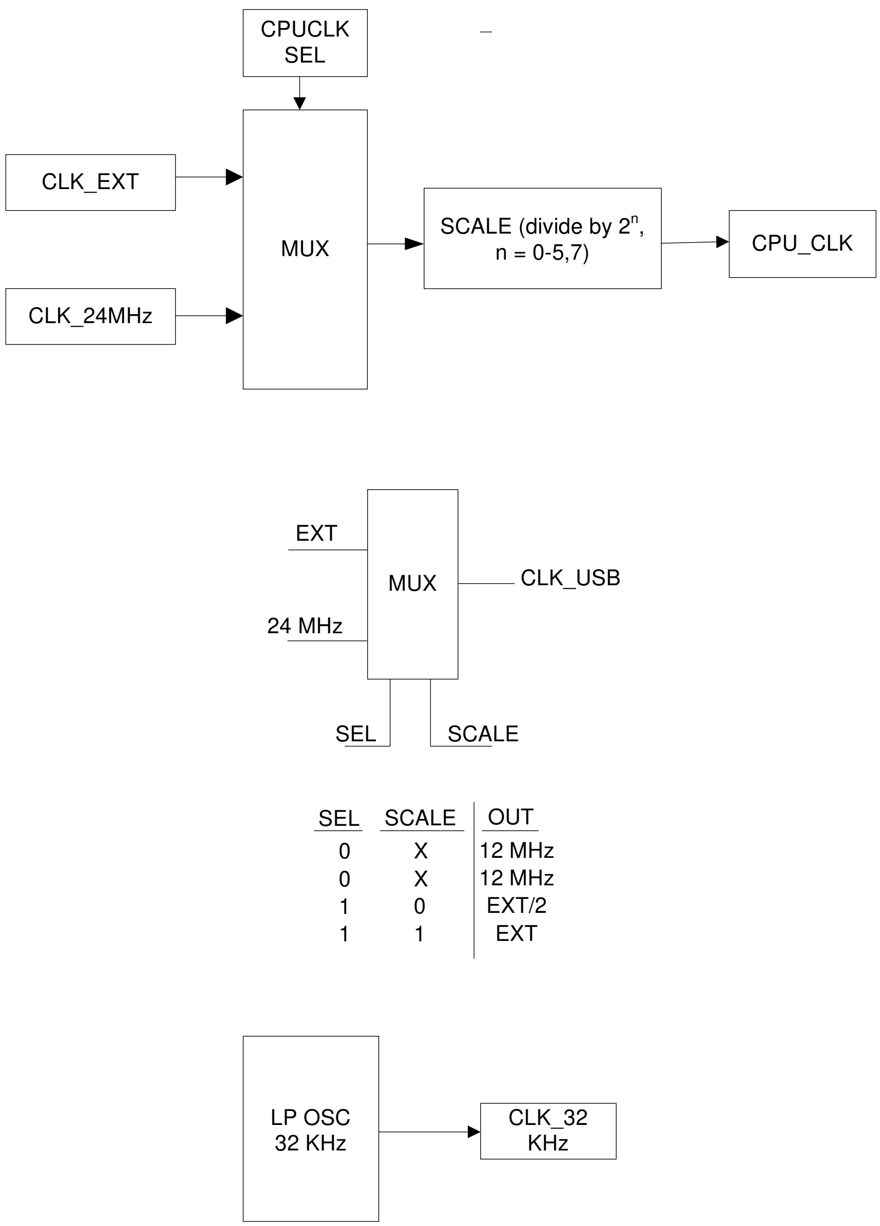
The internal low speed oscillator of nominally 32 kHz provides a slow clock source for the enCoRe II in suspend mode, particularly to generate a periodic wakeup interrupt and also to provide a clock to sequential logic during power up and power down events when the main clock is stopped. In addition, this oscillator can also be used as a clocking source for the Interval Timer clock (ITMRCLK) and Capture Timer clock (TCAPCLK). The 32 kHz Low power Oscillator can operate in low power mode or can provide a more accurate clock in normal mode. The Internal 32 kHz Low power Oscillator accuracy ranges (between 0 °C–70 °C) follow:

- 5 V Normal mode: –8% to + 16%
- 5 V LP mode: +12% to + 48%

When using the 32 kHz oscillator, the PITMRL/H registers must be read until 2 consecutive readings match before the result is considered valid. The following firmware example assumes the developer is interested in the lower byte of the PIT.

```
Read_PIT_counter:
mov A, reg[PITMRL]
mov [57h], A
mov A, reg[PITMRL]
mov [58h], A
mov [59h], A
mov A, reg[PITMRL]
mov [60h], A
;;;Start comparison
mov A, [60h]
mov X, [59h]
sub A, [59h]
jz done
mov A, [59h]
mov X, [58h]
sub A, [58h]
jz done
mov X, [57h]
;;;correct data is in memory location 57h
done:
mov [57h], X
ret
```

Figure 7. Clock Block Diagram



Clock Architecture Description

The enCoRe II clock selection circuitry allows the selection of independent clocks for the CPU, USB, Interval Timers and Capture Timers.

The CPU clock CPUCLK is sourced from an external clock or the Internal 24 MHz Oscillator. The selected clock source is optionally divided by 2^n, where *n* is 0-5,7 (see Table 34 on page 25).

USBCLK, which must be 12 MHz for the USB SIE to function properly, is sourced by the Internal 24 MHz Oscillator or an external 12 MHz/24 MHz clock. An optional divide by two allows the use of 24 MHz source.

The Interval Timer clock (ITMRCLK), is sourced from an external clock, the Internal 24 MHz Oscillator, the Internal 32 kHz low power oscillator, or from the timer capture clock (TCAPCLK). A programmable prescaler of 1, 2, 3, 4 then divides the selected source.

The Timer Capture clock (TCAPCLK) is sourced from an external clock, Internal 24 MHz Oscillator, or the Internal 32 kHz low power oscillator.

The CLKOUT pin (P0.1) is driven from one of many sources. This is used for test and is also used in some applications. The sources that drive the CLKOUT follow:

- CLKIN after the optional EFTB filter
- Internal 24 MHz Oscillator
- Internal 32 kHz low power oscillator
- CPUCLK after the programmable divider

Table 31. IOSC Trim (IOSCTR) [0x34] [R/W]

Bit #	7	6	5	4	3	2	1	0
Field	foffset[2:0]			Gain[4:0]				
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	D	D	D	D	D

The IOSC Calibrate register calibrates the internal oscillator. The reset value is undefined but during boot the SROM writes a calibration value that is determined during manufacturing test. This value does not require change during normal use. This is the meaning of ‘D’ in the Default field.

Bit [7:5]: foffset [2:0]

This value is used to trim the frequency of the internal oscillator. These bits are not used in factory calibration and are zero. Setting each of these bits causes the appropriate fine offset in oscillator frequency.

foffset bit 0 = 7.5 kHz

foffset bit 1 = 15 kHz

foffset bit 2 = 30 kHz

Bit [4:0]: Gain [4:0]

The effective frequency change of the offset input is controlled through the gain input. A lower value of the gain setting increases the gain of the offset input. This value sets the size of each offset step for the internal oscillator. Nominal gain change (kHz/offsetStep) at each bit, typical conditions (24 MHz operation):

Gain bit 0 = –1.5 kHz

Gain bit 1 = –3.0 kHz

Gain bit 2 = –6 kHz

Gain bit 3 = –12 kHz

Gain bit 4 = –24 kHz

Table 32. LPOSC Trim (LPOSCTR) [0x36] [R/W]

Bit #	7	6	5	4	3	2	1	0
Field	32 kHz Low Power	Reserved	32 kHz Bias Trim [1:0]		32 kHz Freq Trim [3:0]			
Read/Write	R/W	–	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	D	D	D	D	D	D	D

This register is used to calibrate the 32 kHz Low speed Oscillator. The reset value is undefined but during boot the SROM writes a calibration value that is determined during manufacturing tests. This value does not require change during normal use. This is the meaning of 'D' in the Default field. If the 32 kHz Low power bit is written, care must be taken to not disturb the 32 kHz Bias Trim and the 32 kHz Freq Trim fields from their factory calibrated values.

Bit 7: 32 kHz Low Power

0 = The 32 kHz Low speed Oscillator operates in normal mode

1 = The 32 kHz Low speed Oscillator operates in a low power mode. The oscillator continues to function normally, but with reduced accuracy.

Bit 6: Reserved

Bit [5:4]: 32 kHz Bias Trim [1:0]

These bits control the bias current of the low power oscillator.

0 0 = Mid bias

0 1 = High bias

1 0 = Reserved

1 1 = Reserved

Note Do not program the 32 kHz Bias Trim [1:0] field with the reserved 10b value because the oscillator does not oscillate at all corner conditions with this setting.

Bit [3:0]: 32 kHz Freq Trim [3:0]

These bits are used to trim the frequency of the low power oscillator.

Table 33. CPU/USB Clock Config (CPUCLKCR) [0x30] [R/W]

Bit #	7	6	5	4	3	2	1	0
Field	Reserved	USB CLK/2 Disable	USB CLK Select	Reserved				CPUCLK Select
Read/Write	–	R/W	R/W	–	–	–	–	R/W
Default	0	0	0	0	0	0	0	0

Bit 7: Reserved

Bit 6: USB CLK/2 Disable

This bit only affects the USBCLK when the source is the external clock. When the USBCLK source is the Internal 24 MHz Oscillator, the divide by two is always enabled

0 = USBCLK source is divided by two. This is the correct setting to use when the Internal 24 MHz Oscillator is used, or when the external source is used with a 24 MHz clock

1 = USBCLK is undivided. Use this setting only with a 12 MHz external clock

Bit 5: USB CLK Select

This bit controls the clock source for the USB SIE.

0 = Internal 24 MHz Oscillator. With the presence of USB traffic, the Internal 24 MHz Oscillator is trimmed to meet the USB requirement of 1.5% tolerance (see Table 35 on page 26)

1 = External clock—Internal Oscillator is not trimmed to USB traffic. **Proper USB SIE operation requires a 12 MHz or 24 MHz clock accurate to <1.5%.**

Bit [4:1]: Reserved

Bit 0: CPU CLK Select

0 = Internal 24 MHz Oscillator.

1 = External clock—External clock at CLKIN (P0.0) pin.

Note The CPU speed selection is configured using the OSC_CR0 Register (Table 34 on page 25).

Table 34. OSC Control 0 (OSC_CR0) [0x1E0] [R/W]

Bit #	7	6	5	4	3	2	1	0
Field	Reserved		No Buzz	Sleep Timer [1:0]		CPU Speed [2:0]		
Read/Write	–	–	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

Bit [7:6]: Reserved

Bit 5: No Buzz

During sleep (the Sleep bit is set in the CPU_SCR Register—Table 38 on page 30), the LVD and POR detection circuit is turned on periodically to detect any POR and LVD events on the V_{CC} pin (the Sleep Duty Cycle bits in the ECO_TR are used to control the duty cycle—Table 42 on page 35). To facilitate the detection of POR and LVD events, the No Buzz bit is used to force the LVD and POR detection circuit to be continuously enabled during sleep. This results in a faster response to an LVD or POR event during sleep at the expense of a slightly higher than average sleep current.

0 = The LVD and POR detection circuit is turned on periodically as configured in the Sleep Duty Cycle.

1 = The Sleep Duty Cycle value is overridden. The LVD and POR detection circuit is always enabled.

Note The periodic Sleep Duty Cycle enabling is independent with the sleep interval shown in the Sleep [1:0] bits below.

Bit [4:3]: Sleep Timer [1:0]

Note Sleep intervals are approximate.

Bit [2:0]: CPU Speed [2:0]

The enCoRe II may operate over a range of CPU clock speeds. The reset value for the CPU Speed bits is zero; as a result, the default CPU speed is one-eighth of the internal 24 MHz, or 3 MHz

Regardless of the CPU Speed bit's setting, if the actual CPU speed is greater than 12 MHz, the 24 MHz operating requirements apply. An example of this scenario is a device that is configured to use an external clock, which supplies a frequency of 20 MHz. If the CPU speed register's value is 0b011, the CPU clock is at 20 MHz. Therefore, the supply voltage requirements for the device are the same as if the part were operating at 24 MHz. The operating voltage requirements are not relaxed until the CPU speed is at 12 MHz or less.

CPU Speed [2:0]	CPU when Internal Oscillator is selected	External Clock
000	3 MHz (Default)	Clock In/8
001	6 MHz	Clock In/4
010	12 MHz	Clock In/2
011	24 MHz	Clock In/1
100	1.5 MHz	Clock In/16
101	750 kHz	Clock In/32
110	187 kHz	Clock In/128
111	Reserved	Reserved

Note Correct USB operations require the CPU clock speed be at least 1.5 MHz or not less than USB clock/8. If the two clocks have the same source, then the CPU clock divider must not be set to divide by more than 8. If the two clocks have different sources, the maximum ratio of USB Clock/CPU Clock must never exceed 8 across the full specification range of both clock sources.

Note This register exists in the second bank of I/O space. This requires setting the XIO bit in the CPU flags register.