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
CY7C639xx

enCoRe™ II Low-Speed USB Peripheral Controller

1.0 Features

- **enCoRe™ II USB—"enhanced Component Reduction"**
  - **Crystalless oscillator with support for an external crystal or resonator. The internal oscillator eliminates the need for an external crystal or resonator**
  - **Internal 3.3V regulator and internal USB pull-up resistor**
  - **Configurable IO 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**
- **Enhanced 8-bit microcontroller**
  - **Harvard architecture**
  - **M8C CPU speed can be up to 24 MHz or sourced by an external crystal, resonator, or 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 GPIO pins needed to manage dual-mode capability**
- **Low power consumption**
  - **Typically 10 mA at 6 MHz**
  - **10-µA sleep**
- **In-system re-programmability**
  - **Allows easy firmware update**
- **General-purpose I/O ports**
  - **Up to 36 General Purpose I/O (GPIO) pins**
  - **High current drive on 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**
- **125-mA 3.3V voltage regulator can power external 3.3V devices**
- **3.3V I/O pins**
  - **4 I/O pins with 3.3V logic levels**
  - **Each 3.3V 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 2-Mbit/second transfers**
  - **Supports half duplex single data line mode for optical sensors**
- **2-channel 8-bit or 1-channel 16-bit capture timer. Capture timers registers store both rising and falling edge times**
  - **Two registers each for two input pins**
  - **Separate registers for rising and falling edge capture**
  - **Simplifies interface to RF inputs for wireless applications**
- **Internal low-power wake-up timer during suspend mode**
  - **Periodic wake-up with no external components**
- **Programmable Interval Timer interrupts**
- **Reduced RF emissions at 27 MHz and 96 MHz**
- **Advanced development tools based on Cypress MicroSystems PSoC™ tools**
- **Watchdog timer (WDT)**
- **Low-voltage detection with user-configurable threshold voltages**
- **Improved output drivers to reduce EMI**
- **Operating voltage from 4.0V to 5.25VDC**
- **Operating temperature from 0–70°C**
- **Available in 16/18/24/40-pin PDIP, 16/18/24-pin SOIC, 24-pin QSOP, 28/48-pin SSOP, and DIE form**
- **Industry standard programmer support**

1.1 Applications

The CY7C633xx/CY7C638xx/CY7C639xx is targeted for the following applications:

- PC HID devices
  - Mice (optomechanical, optical, trackball)
  - Keyboards
- Gaming
  - Joysticks
  - Game pads
  - Console keyboards
- General-purpose
  - Barcode scanners
  - POS terminal
  - Consumer electronics
  - Toys
  - Remote controls

2.0 Introduction

Cypress has reinvented its leadership position in the low-speed USB market with a new family of innovative microcontrollers. Introducing enCoRe II USB — "enhanced Component Reduction." 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 to 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, wake-up circuitry, and a 3.3V regulator.

All of this adds up to a lower system cost.

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

The enCoRe II features up to 36 general-purpose I/O (GPIO) pins to support USB, PS/2 and other applications. The I/O pins are grouped into five ports (Port 0 to 4). The pins on Port 0 and Port 1 may each be configured individually while the pins on Ports 2, 3, and 4 may only be configured 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 programmable drive strength of up to 50-mA sink current. GPIO Port 1 features four pins that interface at a voltage level of 3.3 volts. Additionally, each I/O pin can be used to generate a GPIO interrupt to the microcontroller. Each GPIO port has its own GPIO interrupt vector with the exception of GPIO Port 0. 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 can be set to precisely tune to USB timing requirements (24 MHz ±1.5%). Optionally, an external 12-MHz or 24-MHz crystal can be 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 has up to eight Kbytes of Flash for user's code and up to 256 bytes of RAM for stack space and user variables.

In addition, the enCoRe II includes a Watchdog timer, a vectored interrupt controller, a 16-bit Free-Running Timer, and Capture Timers. The Power-on reset circuit detects logic when power is applied to the device, generates resets the logic to a known state, and begins executing instructions at Flash address 0x0000. When power falls below a programmable trip voltage generates reset or may be configured to generate interrupt. There is a Low-voltage detect circuit that detects when V_{CC} drops below a programmable trip voltage. It may be 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 can be used to ensure the firmware never gets stalled in an infinite loop.

The microcontroller supports 23 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, five GPIO Ports, three GPIO pins, two SPI, a 16-bit free running timer wrap, an internal wake-up timer, and a bus active interrupt. The wake-up timer causes periodic interrupts when enabled. The USB endpoints interrupt after a USB transaction complete is on the bus. The capture timers interrupt whenever a new timer value is saved due to a selected GPIO edge event. A total of eight GPIO interrupts support both TTL or CMOS thresholds. For additional flexibility, on the edge sensitive GPIO pins, the interrupt polarity is programmable to be either rising or falling.

The free-running 16-bit timer provides two interrupt sources: the programmable interval timer with 1-µs resolution and the 1.024-ms outputs. The timer can be used to measure the duration of an event under firmware control by reading the timer at the start and at the end of an event, then calculating the difference between the two values. The two 8-bit capture timers 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 can be ganged into a single 16-bit capture.

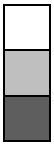
The enCoRe II includes an integrated USB serial interface engine (SIE) that allows the chip to easily interface to a USB host. The hardware supports one USB device address with three endpoints.

The USB D+ and D– pins can optionally be used as PS/2 SCLK and SDATA signals so that products can be designed to respond to either USB or PS/2 modes of operation. 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– can 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. Slow edge rates operate in both modes to reduce EMI.

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.

3.0 Conventions

In this document, 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

CY7C639xx and CY7C638xx only

CY7C639xx only

4.0 Logic Block Diagram

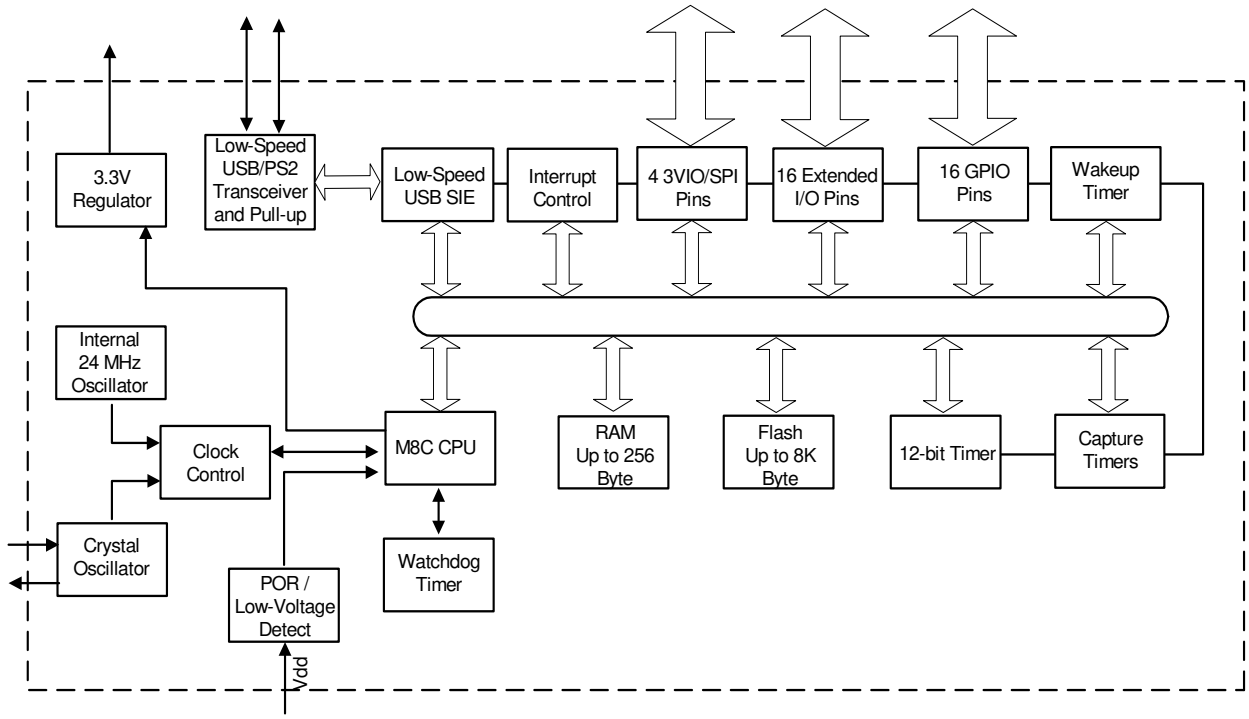
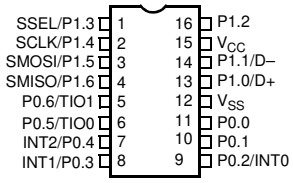


Figure 4-1. CY7C633xx/CY7C638xx/CY7C639xx Block Diagram

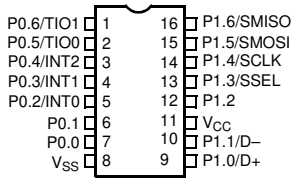
5.0 Packages/Pinouts

Top View

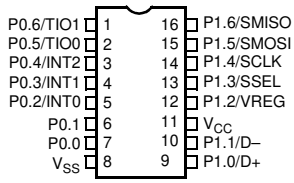
CY7C63801
16-pin PDIP
CY7C63310
16-pin PDIP



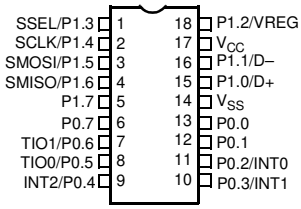
CY7C63801
16-pin SOIC
CY7C63310
16-pin SOIC



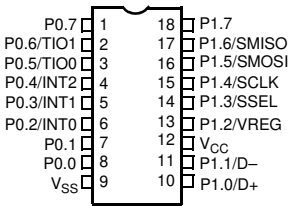
CY7C63803
16-pin SOIC



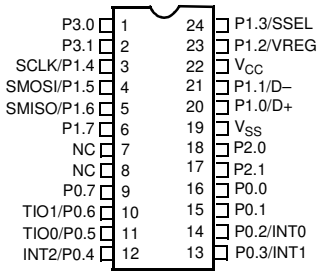
CY7C63813
18-pin PDIP



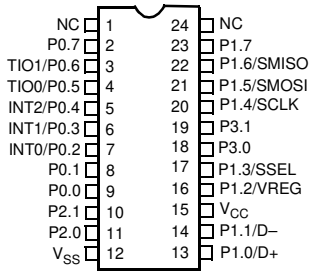
CY7C63813
18-pin SOIC



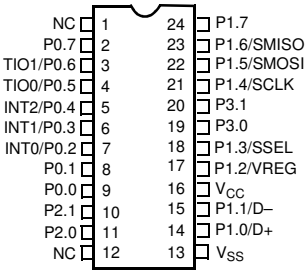
CY7C63823
24-pin PDIP



CY7C63823
24-pin SOIC



CY7C63823
24-pin QSOP



CY7C63903
28-pin SSOP

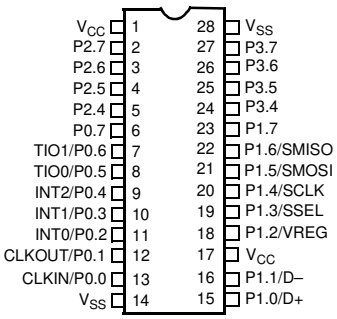
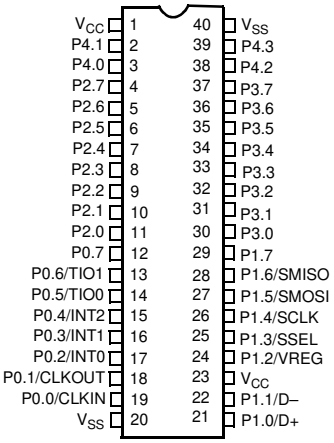


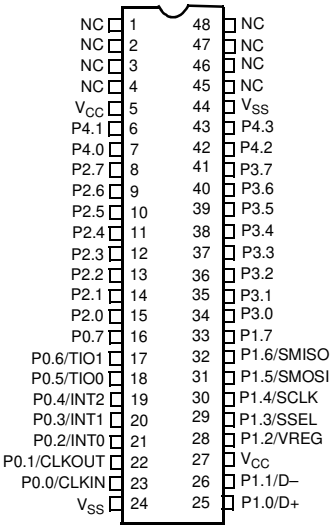
Figure 5-1. Package Configurations

Top View

CY7C63913
40-pin PDIP



CY7C63923
48-pin SSOP



CY7C63923-XC
DIE

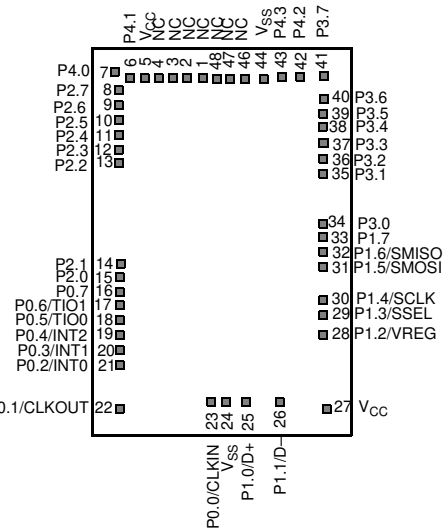


Figure 5-1 Package Configurations (continued)

5.1 Pinouts Assignments

Table 5-1. Pin Assignments

48 SSOP	40 PDIP	28 SSOP	24 QSOP	24 SOIC	24 PDIP	18 SIOC	18 PDIP	16 SOIC	16 PDIP	Die Pad	Name	Description
7	3									7	P4.0	GPIO Port 4 – configured as a group (nibble)
6	2									6	P4.1	
42	38									42	P4.2	
43	39									43	P4.3	
34	30			18	1					34	P3.0	GPIO Port 3 – configured as a group (byte)
35	31		20	19	2					35	P3.1	
36	32		19							36	P3.2	
37	33									37	P3.3	
38	34	24								38	P3.4	
39	35	25								39	P3.5	
40	36	26								40	P3.6	
41	37	27								41	P3.7	
15	11		11	11	18					15	P2.0	GPIO Port 2 – configured as a group (byte)
14	10		10	10	17					14	P2.1	
13	9									13	P2.2	
12	8									12	P2.3	
11	7	5								11	P2.4	
10	6	4								10	P2.5	
9	5	3								9	P2.6	
8	4	2								8	P2.7	

Table 5-1. Pin Assignments (continued)

48 SSOP	40 PDIP	28 SSOP	24 QSOP	24 SOIC	24 PDIP	18 SIOC	18 PDIP	16 SOIC	16 PDIP	Die Pad	Name	Description
25	21	15	14	13	20	10	15	9	13	25	P1.0/D+	GPIO Port 1 bit 0 / USB D+[1]
26	22	16	15	14	21	11	16	10	14	26	P1.1/D–	GPIO Port 1 bit 1 / USB D–[1]
28	24	18	17	16	23	13	18	12	16	28	P1.2/VREG	GPIO Port 1 bit 2—Configured individually. 3.3V if regulator is enabled. (The 3.3V regulator is not available in the CY7C63310 and CY7C63801.)
29	25	19	18	17	24	14	1	13	1	29	P1.3/SSEL	GPIO Port 1 bit 3—Configured individually. Alternate function is SSEL signal of the SPI bus TTL voltage thresholds
30	26	20	21	20	3	15	2	14	2	30	P1.4/SCLK	GPIO Port 1 bit 4—Configured individually. Alternate function is SCLK signal of the SPI bus TTL voltage thresholds
31	27	21	22	21	4	16	3	15	3	31	P1.5/SMOSI	GPIO Port 1 bit 5—Configured individually. Alternate function is SMOSI signal of the SPI bus TTL voltage thresholds
32	28	22	23	22	5	17	4	16	4	32	P1.6/SMISO	GPIO Port 1 bit 6—Configured individually. Alternate function is SMISO signal of the SPI bus TTL voltage thresholds
33	29	23	24	23	6	18	5			33	P1.7	GPIO Port 1 bit 7—Configured individually. TTL voltage threshold.
23	19	13	9	9	16	8	13	7	11	23	P0.0/CLKIN	GPIO Port 0 bit 0—Configured individually. On CY7C639xx, optional Clock In when external crystal oscillator is disabled or crystal input when external crystal oscillator is enabled. On CY7C638xx and CY7C63310, oscillator input when configured as Clock In
22	18	12	8	8	15	7	12	6	10	22	P0.1 / CLKOUT	GPIO Port 0 bit 1—Configured individually On CY7C639xx, optional clock out when external crystal oscillator is disabled or crystal output drive when external crystal oscillator is enabled. On CY7C638xx and CY7C63310, oscillator output when configured as Clock out.
21	17	11	7	7	14	6	11	5	9	21	P0.2/INT0	GPIO port 0 bit 2—Configured individually Optional rising edge interrupt INT0
20	16	10	6	6	13	5	10	4	8	20	P0.3/INT1	GPIO port 0 bit 3—Configured individually Optional rising edge interrupt INT1
19	15	9	5	5	12	4	9	3	7	19	P0.4/INT2	GPIO port 0 bit 4—Configured individually Optional rising edge interrupt INT2
18	14	8	4	4	11	3	8	2	6	18	P0.5/TIO0	GPIO port 0 bit 5—Configured individually Alternate function Timer capture inputs or Timer output TIO0
17	13	7	3	3	10	2	7	1	5	17	P0.6/TIO1	GPIO port 0 bit 6—Configured individually Alternate function Timer capture inputs or Timer output TIO1
16	12	6	2	2	9	1	6			16	P0.7	GPIO port 0 bit 7—Configured individually Not in 16 pin PDIP or SOIC package
1,2,3, 4			1	1	7					1,2, 3,4	NC	No connect

Note:
1. P1.0(D+) and P1.1(D-) pins should be in I/O mode when used as GPIO and in I_{SB} mode.

Table 5-1. Pin Assignments (continued)

48 SSOP	40 PDIP	28 SSOP	24 QSOP	24 SOIC	24 PDIP	18 SIOC	18 PDIP	16 SOIC	16 PDIP	Die Pad	Name	Description
45,46, 47,48			12	24	8					45, 46, 47, 48	NC	No connect
5	1									5	V_{CC}	Power
27	23	1	16	15	22	12	17	11	15	27		
44	40		–	–	–	–				44	V_{SS}	Ground
24	20	28	13	12	19	9	14	8	12	24		

6.0 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 6-1. CPU Registers and Register Names

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 for direct addressing of the full eight Kbytes of program memory space.

The Accumulator Register (CPU_A) is the general-purpose register that 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-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 can also be 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] is used to globally enable or disable 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 (i.e., AND, OR, XOR). See *Table 8-1*.

7.0 CPU Registers

7.1 Flags Register

The Flags Register can only be set or reset with logical instruction.

Table 7-1. 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 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 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 explicit register address 0xF7. The *OR F, expr and AND F, expr* instructions must be used to set and clear the CPU_F bits

7.1.1 Accumulator Register

Table 7-2. 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

7.1.2 Index Register

Table 7-3. 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

7.1.3 Stack Pointer Register

Table 7-4. 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-stack

7.1.4 CPU Program Counter High Register

Table 7-5. 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

7.1.5 CPU Program Counter Low Register

Table 7-6. 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

7.2 Addressing Modes

7.2.1 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. Instructions using this addressing mode are two bytes in length.

Table 7-7. Source Immediate

Opcode	Operand 1
Instruction	Immediate Value

Examples

```
ADD   A,  7   ;In this case, the immediate value
              ;of 7 is added with the Accumulator,
              ;and the result is placed in the
              ;Accumulator.
MOV   X,  8   ;In this case, the immediate value
              ;of 8 is moved to the X register.
AND   F,  9   ;In this case, the immediate value
              ;of 9 is logically ANDed with the F
              ;register and the result is placed
              ;in the F register.
```

7.2.2 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 either the RAM memory space or the register space that is the source for 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 7-8. Source Direct

Opcode	Operand 1
Instruction	Source Address

Examples

```
ADD    A,    [7]      ;In this case, 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]   ;In this case, the value in
                      ;the register space at address
                      ;8 is moved to the X register.
```

7.2.3 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 either the RAM memory space or the register space that is the source for 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 7-9. Source Indexed

Opcode	Operand 1
Instruction	Source Index

Examples

```
ADD    A,    [X+7]      ;In this case, 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]   ;In this case, the value in
                        ;the register space at
                        ;address X + 8 is moved to
                        ;the X register.
```

7.2.4 Destination Direct

The result of an instruction using this addressing mode is placed within either 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 7-10. Destination Direct

Opcode	Operand 1
Instruction	Destination Address

Examples

```
ADD    [7],    A      ;In this case, 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      ;In this case, the Accumula-
                      ;tor is moved to the regis-
                      ;ter space location at
                      ;address 8. The Accumulator
                      ;is unchanged.
```

7.2.5 Destination Indexed

The result of an instruction using this addressing mode is placed within either 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 7-11. Destination Indexed

Opcode	Operand 1
Instruction	Destination Index

Example

```
ADD    [X+7],    A    ;In this case, 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.
```

7.2.6 Destination Direct Immediate

The result of an instruction using this addressing mode is placed within either the RAM memory space or the register space. Operand 1 is the address of the result. The source for 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 7-12. Destination Direct Immediate

Opcode	Operand 1	Operand 2
Instruction	Destination Address	Immediate Value

Examples

```
ADD   [7],     5    ;In this case, value in the mem-
                    ;ory 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    ;In this case, the immediate
                    ;value of 6 is moved into the
                    ;register space location at
                    ;address 8.
```

7.2.7 Destination Indexed Immediate

The result of an instruction using this addressing mode is placed within either 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 for 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 7-13. Destination Indexed Immediate

Opcode	Operand 1	Operand 2
Instruction	Destination Index	Immediate Value

Examples

```
ADD    [X+7],     5     ;In this case, 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     ;In this case, the immedi-
                        ;ate value of 6 is moved
                        ;into the location in the
                        ;register space at
                        ;address X+8.
```

7.2.8 Destination 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 7-14. Destination Direct

Opcode	Operand 1	Operand 2
Instruction	Destination Address	Source Address

Example

```
MOV   [7],  [8]  ;In this case, the value in the
                 ;memory location at address 8 is
                 ;moved to the memory location at
                 ;address 7.
```

7.2.9 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 7-15. Source Indirect Post Increment

Opcode	Operand 1
Instruction	Source Address Address

Example

```
MVI   A,   [8]   ;In this case, the value in the
                 ;memory location at address 8 is
                 ;an indirect address. The memory
                 ;location pointed to by the indi-
                 ;rect address is moved into the
                 ;Accumulator. The indirect
                 ;address is then incremented.
```

7.2.10 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 7-16. Destination Indirect Post Increment

Opcode	Operand 1
Instruction	Destination Address Address

Example

```
MVI    [8],  A     ;In this case, the value in
                   ;the memory location at
                   ;address 8 is an indirect
                   ;address. The Accumulator is
                   ;moved into the memory loca-
                   ;tion pointed to by the indi-
                   ;rect address. The indirect
                   ;address is then incremented.
```

8.0 Instruction Set Summary

The instruction set is summarized in *Table 8-1* by 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 www.cypress.com web site).

Table 8-1. 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.

9.0 Memory Organization

9.1 Flash Program Memory Organization

after reset	Address	
16-bit PC →	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	GPIO Port 4
	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 (CY7C639xx and CY7C638x3)**

Figure 9-1. Program Memory Space with Interrupt Vector Table

9.2 Data Memory Organization

The CY7C633xx/638xx/639xx microcontrollers provide up to 256 bytes of data RAM. In normal usage, the SRAM is partitioned into two areas: stack, and user variables:

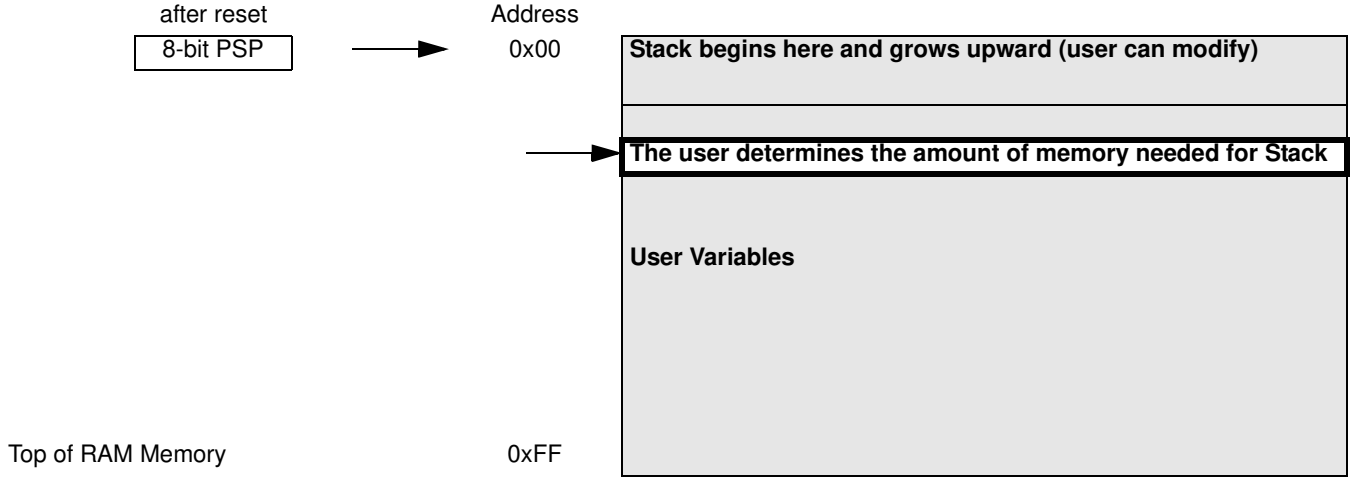


Figure 9-2. Data Memory Organization

9.3 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).

9.3.1 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 is executing 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 only program the Flash via SROM calls. The data or code images can be sourced via any interface with the appropriate support firmware. This type of programming requires a 'boot-loader'—a piece of firmware resident on the Flash. For safety reasons this boot-loader should 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 SROM section. The auxiliary rows are not affected by the device erase function.

9.3.2 In-System Programming

Most designs that include an enCoRe II part will have a USB connector attached to the USB D+/D– pins on the device. These designs require the ability to program or reprogram a part through these two pins alone. The programming protocol is not USB.

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 cause the enCoRe II part to enter serial programming mode and then to use the test queue to issue Flash access functions in the SROM.

9.4 SROM

The SROM holds code that is used to boot the part, calibrate circuitry, and perform Flash operations. (*Table 9-1* lists the SROM functions.) The functions of the SROM may be accessed in normal user code or operating from Flash. The SROM exists in a separate memory space from user code. The SROM functions are accessed by executing the Supervisory System Call instruction (SSC), which has an opcode of 00h. Prior to executing the SSC the M8C's accumulator needs to be loaded with the desired SROM function code from *Table 9-1*. Undefined functions will cause a HALT if called from user code. The SROM functions are executing code with calls; therefore, 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 9-2* lists all possible parameter block variables. The meaning of each parameter, with regards to a specific SROM function, is described later in this chapter.

Table 9-1. 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 will halt (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 9-2. 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

The SROM also features Return Codes and Lockouts.

9.4.1 Return Codes

Return codes aid in the determination of 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 9-3. 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.

9.5 SROM Function Descriptions

9.5.1 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 will happen, by design, after a hardware reset, because the M8C's accumulator is reset to 00h or when user code executes the SSC instruction with an accumulator value of 00h. The SWBootReset function will not execute when the SSC instruction is executed with a bad key value and a non-zero function code. An enCoRe II device will execute 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.

9.5.2 ReadBlock Function

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

The first thing this function does is to check the protection bits and determine if the desired BLOCKID is readable. If read protection is turned on, the ReadBlock function will exit setting the accumulator and KEY2 back to 00h. KEY1 will have a value of 01h, indicating a read failure. If read protection is not enabled, the function will read 64 bytes from the Flash using a ROMX instruction and store the results in SRAM using an MVI instruction. The first of the 64 bytes will be stored in SRAM at the address indicated by the value of the POINTER parameter. When the ReadBlock completes successfully the accumulator, KEY1 and KEY2 will all have a value of 00h.

Table 9-4. 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 should be stored

9.5.3 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 first thing the WriteBlock function does is to check the protection bits and determine if the desired BLOCKID is writable. If write protection is turned on, the WriteBlock function will exit setting the accumulator and KEY2 back to 00h. KEY1 will have 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 will be used to store the data in the Flash. The CLOCK and DELAY values are dependent on the CPU speed and must be set correctly. Refer to “Clocking” Section for additional information.

Table 9-5. 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 prior to 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

9.5.4 EraseBlock Function

The EraseBlock function is used to erase a block of 64 contiguous bytes in Flash. The first thing the EraseBlock function does is to check the protection bits and determine if the desired BLOCKID is writable. If write protection is turned on, the EraseBlock function will exit setting the accumulator and KEY2 back to 00h. KEY1 will have a value of 01h, indicating a write failure. The EraseBlock function is only useful as the first step in programming. Erasing a block will not cause data in a block to be 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 9-6. EraseBlock Parameters

Name	Address	Description
KEY1	0,F8h	3Ah
KEY2	0,F9h	Stack Pointer value, when SSC is executed.
BLOCKID	0,FAh	Flash block number (00h–7Fh)
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

9.5.5 ProtectBlock Function

The enCoRe II devices offer Flash protection on a block-by-block basis. *Table 9-7* lists the protection modes available. In the table, ER and EW are used to indicate the ability to perform external reads and writes. For internal writes, IW is used. Internal reading is always 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 9-7*. These bits are bit packed into the 64 bytes of the protection block. Therefore, 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 9-7*.

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 will store the protection level for blocks 252 through 255.

Table 9-7. 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 prior to calling the function. The parameter block values that must be set, besides the keys, are the CLOCK and DELAY values.

Table 9-8. 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

9.5.6 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. Following 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 9-9. 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

9.5.7 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 9-10. 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 and CY7C639xx parts consist of 64 bytes.

An internal table holds the Silicon ID and returns the Revision ID. The Silicon ID is returned in SRAM, while the Revision ID is 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. The Revision ID is discussed in more detail later in this section.

An internal table holds alternate trim 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 each time one of the other revision numbers is not incremented. It is reset to zero each time one of the other revision numbers is incremented. The internal revision count is returned in the CPU_A register. The CPU_X register will always be set to FFh when trim values are read. The BLOCKID value, in the parameter block, is used to indicate which table should be returned to the user. Only the three least significant bits of the BLOCKID parameter are used by TableRead function for the CY7C638xx and CY7C639xx. 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.

9.5.8 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 will calculate the checksum of only block 0, while a BLOCKID value of 0 will calculate 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 9-11. 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.

10.0 Clocking

The enCoRe II internal oscillator outputs two frequencies, 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 can be set to precisely tune to USB timing requirements (24 MHz ± 1.5%). Without USB traffic, the Internal 24-MHz Oscillator accuracy is 24 MHz ± 5% (between 0°–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 wake-up 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 from –85% to +120% (between 0°–70° C).

For applications that require a higher clock accuracy, the CY7C639xx part can optionally be sourced from an external crystal oscillator. When operating in USB mode, the supplied crystal oscillator must be either 12 MHz or 24 MHz in order for the USB blocks to function properly. In non-USB mode, the external oscillator can be up to 24 MHz.

10.1 Clock Architecture Description

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

On the CY7C639xx, the external oscillator can be sourced by the crystal oscillator or when the crystal oscillator is disabled it is sourced directly from the CLKIN pin. The external crystal oscillator is fed through the EFTB block, which can optionally be bypassed.

The CPU clock, CPUCLK, can be sourced from the external crystal oscillator or the Internal 24-MHz Oscillator. The selected clock source can optionally be divided by 2^n where *n* is 0-5,7 (see *Table 10-5*).

USBCLK, which must be 12 MHz for the USB SIE to function properly, can be sourced by the Internal 24-MHz Oscillator or the external crystal oscillator. An optional divide by two allows the use of 24-MHz source.

The Interval Timer clock (ITMRCLK), can be sourced from the external crystal oscillator, 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) can be sourced from the external crystal oscillator, Internal 24-MHz Oscillator, or the Internal 32-KHz Low-power Oscillator.

When it is not being used by the external crystal oscillator, the CLKOUT pin can be driven from one of many sources. This is used for test and can also be used in some applications. The sources that can drive the CLKOUT are:

- CLKIN after the optional EFTB filter
- Internal 24-MHz Oscillator
- Internal 32-KHz Low-power Oscillator
- CPUCLK after the programmable divider

10.1.1 Clock Control Registers

10.1.2 Internal Clock Trim

Table 10-1. 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 is used to calibrate 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 should 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 will be 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

10.1.3 External Clock Trim

Table 10-2. XOSC Trim (XOSCTR) [0x35] [R/W]

Bit #	7	6	5	4	3	2	1	0
Field	Reserved			XOSC XGM [2:0]			Reserved	Mode
Read/Write	–	–	–	R/W	R/W	R/W	–	R/W
Default	0	0	0	D	D	D	0	D

This register is used to calibrate the external crystal oscillator. The reset value is undefined but during boot the SROM writes a calibration value that is determined during manufacturing test. This is the meaning of 'D' in the Default field
Bit [7:5]: Reserved
Bit [4:2]: XOSC XGM [2:0]
Amplifier transconductance setting. The Xgm settings are recommended for resonators with frequencies of interest for the enCoRe II as below

Resonator	XGM Setting	Worst Case R (Ohms)
6-MHz Crystal	001	403
12-MHz Crystal	011	201
24-MHz Crystal	111	101
6-MHz Ceramic	001	70.4
12-MHz Ceramic	011	41

Bit 1: Reserved
Bit 0: Mode
0 = Oscillator Mode
1 = Fixed Maximum Bias test Mode

10.1.4 LPOSC Trim

Table 10-3. 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	D	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 test. This value should not require change during normal use. This is the meaning of 'D' in the Default field. If the 32-KHz Low-power bit needs to be written, care should be taken not to 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 = Disable (off)
Important Note: Do not program the 32-KHz Bias Trim [1:0] field with the reserved 10b value as 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

10.1.5 CPU/USB Clock Configuration

Table 10-4. 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 crystal oscillator. 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 can be trimmed to meet the USB requirement of 1.5% tolerance (see *Table 10-6*)
1 = External clock—external oscillator on CLKIN and CLKOUT if the external oscillator is enabled (the XOSC Enable bit set in the CLKIOCR Register—*Table 10-8*), or the CLKIN input if the external oscillator is disabled. 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 crystal oscillator—External crystal oscillator on CLKIN and CLKOUT if the external crystal oscillator is enabled, CLKIN input if the external crystal oscillator is disabled
Note: the CPU speed selection is configured using the OSC_CR0 Register (*Table 10-5*)

10.1.6 OSC_CR0 Clock Configuration

Table 10-5. 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 11-1*), 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 13-3*). 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]

Sleep Timer [1:0]	Sleep Timer Clock Frequency (Nominal)	Sleep Period (Nominal)	Watchdog Period (Nominal)
00	512 Hz	1.95 ms	6 ms
01	64 Hz	15.6 ms	47 ms
10	8 Hz	125 ms	375 ms
11	1 Hz	1 sec	3 sec

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; therefore, 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 is supplying a frequency of 20 MHz. If the CPU speed register's value is 0b011, the CPU clock will be 20 MHz. Therefore the supply voltage requirements for the device are the same as if the part was 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

Important Note: Correct USB operations require the CPU clock speed to be at least eight times greater than the USB clock. If the two clocks have the same source then the CPU clock divider should not be set to divide by more than 8. If the two clocks have different sources, care must be taken to ensure that the maximum ratio of USB Clock/CPU Clock can never exceed 8 across the full specification range of both clock sources

10.1.7 USB Oscillator Lock Configuration

Table 10-6. USB Osclock Clock Configuration (OSCLCKCR) [0x39] [R/W]

Bit #	7	6	5	4	3	2	1	0
Field	Reserved						Fine Tune Only	USB Osclock Disable
Read/Write	–	–	–	–	–	–	R/W	R/W
Default	0	0	0	0	0	0	0	0

This register is used to trim the Internal 24-MHz Oscillator using received low-speed USB packets as a timing reference. The USB Osclock circuit is active when the Internal 24-MHz Oscillator provides the USB clock
Bit [7:2]: Reserved
Bit 1: Fine Tune Only
0 = Enable
1 = Disable the oscillator lock from performing the course-tune portion of its retuning. The oscillator lock must be allowed to perform a course tuning in order to tune the oscillator for correct USB SIE operation. After the oscillator is properly tuned this bit can be set to reduce variance in the internal oscillator frequency that would be caused course tuning
Bit 0: USB Osclock Disable
0 = Enable. With the presence of USB traffic, the Internal 24-MHz Oscillator precisely tunes to 24 MHz ± 1.5%
1 = Disable. The Internal 24-MHz Oscillator is not trimmed based on USB packets. This setting is useful when the internal oscillator is not sourcing the USBSIE clock

10.1.8 Timer Clock Configuration

Table 10-7. Timer Clock Config (TMRCLKCR) [0x31] [R/W]

Bit #	7	6	5	4	3	2	1	0
Field	TCAPCLK Divider		TCAPCLK Select		ITMRCLK Divider		ITMRCLK Select	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	1	0	0	0	1	1	1	1

Bit [7:6]: TCAPCLK Divider [1:0]
TCAPCLK Divider controls the TCAPCLK divisor
0 0 = Divider Value 2
0 1 = Divider Value 4
1 0 = Divider Value 6
1 1 = Divider Value 8
Bit [5:4]: TCAPCLK Select
The TCAPCLK Select field controls the source of the TCAPCLK
0 0 = Internal 24-MHz Oscillator
0 1 = External crystal oscillator—external crystal oscillator on CLKIN and CLKOUT if the external crystal oscillator is enabled, CLKIN input if the external crystal oscillator is disabled (the XOSC Enable bit of the CLKIOCR Register is cleared—*Table 10-8*)
1 0 = Internal 32-KHz Low-power Oscillator
1 1 = TCAPCLK Disabled
Note: The 1024-µs interval timer is based on the assumption that TCAPCLK is running at 4 MHz. Changes in TCAPCLK frequency will cause a corresponding change in the 1024-µs interval timer frequency
Bit [3:2]: ITMRCLK Divider
ITMRCLK Divider controls the ITMRCLK divisor.
0 0 = Divider value of 1
0 1 = Divider value of 2
1 0 = Divider value of 3
1 1 = Divider value of 4
Bit [1:0]: ITMRCLK Select
0 0 = Internal 24-MHz Oscillator
0 1 = External crystal oscillator – external crystal oscillator on CLKIN and CLKOUT if the external crystal oscillator is enabled, CLKIN input if the external crystal oscillator is disabled
1 0 = Internal 32-KHz Low-power Oscillator
1 1 = TCAPCLK

10.1.9 Clock In / Clock Out Configuration

Table 10-8. Clock I/O Config (CLKIOCR) [0x32] [R/W]

Bit #	7	6	5	4	3	2	1	0
Field	Reserved			XOSC Select	XOSC Enable	EFTB Disabled	CLKOUT Select	
Read/Write	–	–	–	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

Bit [7:5]: Reserved
Bit 4: XOSC Select
This bit when set, selects the external crystal oscillator clock as clock source of external clock. Care needs to be taken while selecting the crystal oscillator clock. First enable the crystal oscillator and wait for few cycles, which is oscillator stabilization period. Then select the crystal clock as clock source. Similarly, while deselect crystal clock, first deselect crystal clock as clock source then disable the crystal oscillator.
0 = Not select external crystal oscillator clock
1 = Select the external crystal oscillator clock
Bit 3: XOSC Enable
This bit when set enables the external crystal oscillator. The external crystal oscillator shares pads CLKIN and CLKOUT with two GPIOs—P0.0 and P0.1, respectively. When the external crystal oscillator is enabled, the CLKIN signal comes from the external crystal oscillator block and the output enables on the GPIOs for P0.0 and P0.1 are disabled, eliminating the possibility of contention. When the external crystal oscillator is disabled the source for CLKIN signal comes from the P0.0 GPIO input.
0 = Disable the external oscillator
1 = Enable the external oscillator
Note: The external crystal oscillator startup time takes up to 2 ms.
Bit 2: EFTB Disabled
This bit is only available on the CY7C639xx
0 = Enable the EFTB filter
1 = Disable the EFTB filter, causing CLKIN to bypass the EFTB filter
Bit [1:0]: CLKOUT Select
0 0 = Internal 24-MHz Oscillator
0 1 = External crystal oscillator – external crystal oscillator on CLKIN and CLKOUT if the external crystal oscillator is enabled, CLKIN input if the external oscillator is disabled
1 0 = Internal 32-KHz Low-power Oscillator
1 1 = CPUCLK

10.2 CPU Clock During Sleep Mode

When the CPU enters sleep mode the CPUCLK Select (Bit 1, *Table 10-4*) is forced to the Internal Oscillator, and the oscillator is stopped. When the CPU comes out of sleep mode it is running on the internal oscillator. The internal oscillator recovery time is three clock cycles of the Internal 32-KHz Low-power Oscillator.

If the system requires the CPU to run off the external clock after awaking from sleep mode, firmware will need to switch the clock source for the CPU. If the external clock source is the external oscillator and the oscillator is disabled, firmware will need to enable the external oscillator, wait for it to stabilize, and then change the clock source.

11.0 Reset

The microcontroller supports two types of resets: Power-on Reset (POR) and Watchdog Reset (WDR). When reset is initiated, all registers are restored to their default states and all interrupts are disabled.

The occurrence of a reset is recorded in the System Status and Control Register (CPU_SCR). Bits within this register record the occurrence of POR and WDR Reset respectively. The firmware can interrogate these bits to determine the cause of a reset.

The microcontroller resumes execution from Flash address 0x0000 after a reset. The internal clocking mode is active after a reset, until changed by user firmware.

Note: The CPU clock defaults to 3 MHz (Internal 24-MHz Oscillator divide-by-8 mode) at POR to guarantee operation at the low V_{CC} that might be present during the supply ramp.

Table 11-1. System Status and Control Register (CPU_SCR) [0xFF] [R/W]

Bit #	7	6	5	4	3	2	1	0
Field	GIES	Reserved	WDRS	PORS	Sleep	Reserved		Stop
Read/Write	R	–	R/C[4]	R/C[4]	R/W	–	–	R/W
Default	0	0	0	1	0	0	0	0

The bits of the CPU_SCR register are used to convey status and control of events for various functions of an enCoRe II device
Bit 7: GIES
The Global Interrupt Enable Status bit is a read only status bit and its use is discouraged. The GIES bit is a legacy bit, which was used to provide the ability to read the GIE bit of the CPU_F register. However, the CPU_F register is now readable. When this bit is set, it indicates that the GIE bit in the CPU_F register is also set which, in turn, indicates that the microprocessor will service interrupts
0 = Global interrupts disabled
1 = Global interrupt enabled
Bit 6: Reserved
Bit 5: WDRS
The WDRS bit is set by the CPU to indicate that a WDR event has occurred. The user can read this bit to determine the type of reset that has occurred. The user can clear but not set this bit
0 = No WDR
1 = A WDR event has occurred
Bit 4: PORS
The PORS bit is set by the CPU to indicate that a POR event has occurred. The user can read this bit to determine the type of reset that has occurred. The user can clear but not set this bit
0 = No POR
1 = A POR event has occurred. (Note that WDR events will not occur until this bit is cleared)
Bit 3: SLEEP
Set by the user to enable CPU sleep state. CPU will remain in sleep mode until any interrupt is pending. The Sleep bit is covered in more detail in the Sleep Mode section
0 = Normal operation
1 = Sleep
Bit [2:1]: Reserved
Bit 0: STOP
This bit is set by the user to halt the CPU. The CPU will remain halted until a reset (WDR, POR, or external reset) has taken place. If an application wants to stop code execution until a reset, the preferred method would be to use the HALT instruction rather than writing to this bit
0 = Normal CPU operation
1 = CPU is halted (not recommended)

11.1 Power-on Reset

POR occurs every time the power to the device is switched on. POR is released when the supply is typically 2.6V for the upward supply transition, with typically 50 mV of hysteresis during the power-on transient. Bit 4 of the System Status and Control Register (CPU_SCR) is set to record this event (the register contents are set to 00010000 by the POR). After a POR, the microprocessor is held off for approximately 20 ms for the V_{CC} supply to stabilize before executing the first instruction at address 0x00 in the Flash. If the V_{CC} voltage drops below the POR downward supply trip point, POR is reasserted. The V_{CC} supply needs to ramp linearly from 0 to 4V in 0 to 200 ms.

Important: The PORS status bit is set at POR and can only be cleared by the user. It cannot be set by firmware.

11.2 Watchdog Timer Reset

The user has the option to enable the WDT. The WDT is enabled by clearing the PORS bit. Once the PORS bit is cleared, the WDT cannot be disabled. The only exception to this is if a POR event takes place, which will disable the WDT.

The sleep timer is used to generate the sleep time period and the Watchdog time period. The sleep timer uses the Internal 32-KHz Low-power Oscillator system clock to produce the sleep time period. The user can program the sleep time period using the Sleep Timer bits of the OSC_CR0 Register (*Table 10-5*). When the sleep time elapses (sleep timer overflows), an interrupt to the Sleep Timer Interrupt Vector will be generated.

The Watchdog Timer period is automatically set to be three counts of the Sleep Timer overflows. This represents between two and three sleep intervals depending on the count in the Sleep Timer at the previous WDT clear. When this timer reaches three, a WDR is generated.

The user can either clear the WDT, or the WDT and the Sleep Timer. Whenever the user writes to the Reset WDT Register (RES_WDT), the WDT will be cleared. If the data that is written is the hex value 0x38, the Sleep Timer will also be cleared at the same time.

Note:
4. C = Clear. This bit can only be cleared by the user and cannot be set by firmware.

Table 11-2. Reset Watchdog Timer (RESWDT) [0xE3] [W]

Bit #	7	6	5	4	3	2	1	0
Field	Reset Watchdog Timer [7:0]							
Read/Write	W	W	W	W	W	W	W	W
Default	0	0	0	0	0	0	0	0
Any write to this register will clear Watchdog Timer, a write of 0x38 will also clear the Sleep Timer **Bit [7:0]:** Reset Watchdog Timer [7:0]								

12.0 Sleep Mode

The CPU can only be put to sleep by the firmware. This is accomplished by setting the Sleep bit in the System Status and Control Register (CPU_SCR). This stops the CPU from executing instructions, and the CPU will remain asleep until an interrupt comes pending, or there is a reset event (either a Power-on Reset, or a Watchdog Timer Reset).

The Low-voltage Detection circuit (LVD) drops into fully functional power-reduced states, and the latency for the LVD is increased. The actual latency can be traded against power consumption by changing Sleep Duty Cycle field of the ECO_TR Register.

The Internal 32-KHz Low-speed Oscillator remains running. Prior to entering suspend mode, firmware can optionally configure the 32-KHz Low-speed Oscillator to operate in a low-power mode to help reduce the over all power consumption (Using Bit 7, *Table 10-3*). This will help save approximately 5 µA; however, the trade off is that the 32-KHz Low-speed Oscillator will be less accurate (–85% to +120% deviation).

All interrupts remain active. Only the occurrence of an interrupt will wake the part from sleep. The Stop bit in the System Status and Control Register (CPU_SCR) must be cleared for a part to resume out of sleep. The Global Interrupt Enable bit of the CPU Flags Register (CPU_F) does not have any effect. Any unmasked interrupt will wake the system up. As a result, any interrupts not intended for waking should be disabled through the Interrupt Mask Registers.

When the CPU enters sleep mode the CPUCLK Select (Bit 1, *Table 10-4*) is forced to the Internal Oscillator. The internal oscillator recovery time is three clock cycles of the Internal 32-KHz Low-power Oscillator. The Internal 24-MHz Oscillator restarts immediately on exiting Sleep mode. If the external crystal oscillator is used, firmware will need to switch the clock source for the CPU.

Unlike the Internal 24-MHz Oscillator, the external oscillator is not automatically shut down during sleep. Systems that need the external oscillator disabled in sleep mode will need to disable the external oscillator prior to entering sleep mode. In systems where the CPU runs off the external oscillator, firmware will need to switch the CPU to the internal oscillator prior to disabling the external oscillator.

On exiting sleep mode, once the clock is stable and the delay time has expired, the instruction immediately following the sleep instruction is executed before the interrupt service routine (if enabled).

The Sleep interrupt allows the microcontroller to wake up periodically and poll system components while maintaining very low average power consumption. The Sleep interrupt may also be used to provide periodic interrupts during non-sleep modes.