Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

THIS SPEC IS OBSOLETE

Spec No: 001-15343

Spec Title: ENCORE(TM) TO ENCORE II CONVERSION - AN6062

Replaced by: NONE

AN6062

Encore™ to Encore II Conversion

Author: Kevin Hung
Associated Project: No
Associated Part Family: enCoRe™, enCoRe II
Software Version: NA
Related Application Notes: None

To get the latest version of this application note, or the associated project file, please visit http://www.cypress.com/go/AN6062.

Having sold hundreds of millions of units, the Cypress enCoRe™ low-speed USB microcontroller family is the most successful USB device in the industry. However, it has not seen an update since its introduction in 2000. To keep up with the demands for increasing product functionality and decreasing system cost, Cypress has released the enCoRe II. AN6062 is targeted at developers who are familiar with the enCoRe devices, and who want to migrate to the next generation enCoRe II. It highlights the differences between the products that require attention during the migration, and also provides guidance on how to use some of the enCoRe II features. Although this application note discusses some hardware issues, its emphasis is on firmware.

Contents

Overview 2
enCoRe II Core Components and Architecture 2
- Memory 2
- Interrupts 3
- Clocking 3
- Reset 4
- USB Transceiver 4
- Sleep Timer Interrupt / Wakeup Timer 5
- GPIO 6
- Serial Peripheral Interface 7
- Timers 7
- CPU Differences 7
- Assembler Instruction Differences Between enCoRe and enCoRe II 9
- New Instructions in M8C 12

Development Environment 13
- Development Hardware 13
- Development Software 14

Using enCoRe II Features and Functional Blocks 20
- Using the 1-ms Timer Interrupt 20
- Capture Timer 21
- 12-Bit Programmable Interval Timer 21
- The Sleep Timer Interrupt 22
- Serial Peripheral Interface 23
- GPIO Interrupts 23
- E2PROM User Module 24
- USB 25
- USB and PS/2 Macros 27

EMC Suggestions 27
- GND PAD Assignment 27
- Avoiding Antenna Effects 27
- WDT Timer Usage 27
- Clock Control 28

Summary 28
Worldwide Sales and Design Support 30

Overview

enCoRe II is a new low-speed microcontroller family that is an extension of the enCoRe/M8 families. Its technical base comes from enCoRe and adds new and more flexible functions. Cypress recommends that customers move their designs from enCoRe to enCoRe II for the following reasons:

- enCoRe II is the newest family — legacy products will eventually become obsolete
- enCoRe II is a lower system cost solution than enCoRe
- enCoRe II has the following advantageous new features:
  - Flash with EEPROM emulation. Eliminates external EEPROM in some applications
  - SPI interface on all devices
  - 16-bit capture timer for better resolution
  - Internal USB 1.5 kΩ pull-up
  - Internal PS2 pull-up
  - In-system reprogrammable through the USB connector
  - Lower power consumption for battery-powered applications
  - More efficient M8C instructions
  - Tools for future expansion
  - Smaller packages for space-constrained designs

enCoRe II Core Components and Architecture

enCoRe II is a next-generation low-speed USB controller; it is the followup to the current enCoRe/M8 families. However, some architectural changes from enCoRe are required for increased capabilities, as summarized in the following sections.

Memory

This section gives information on memory, memory usage, and memory maps.

Program Memory: ROM vs. Flash (OTP vs. In-System Reprogrammable)

enCoRe is a one-time programmable (OTP) product. Customers cannot reprogram it if they find something wrong after programming. Customers never run into this situation with enCoRe II, because it is reprogrammable. New changes can easily be reloaded. Multiple flash protection modes are included to ensure customer code security.

Programming

enCoRe devices could be factory programmed, or could be programmed out of system by the developer. A dedicated programming voltage pin was required on all devices, eliminating space for one IO pin. On enCoRe II, the programming pins are overlaid with USB D+ and D– and, of course, power and ground. Therefore, enCoRe II devices can be factory programmed, programmed out of system by the user, or programmed in-system, without infringing on the IO pins available for the application.

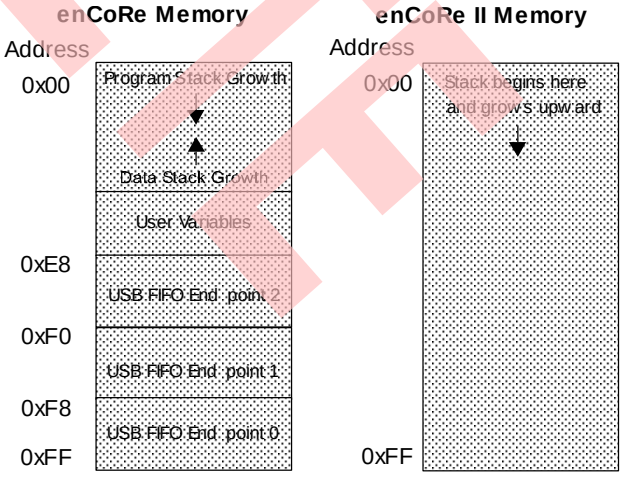
There are also options for in-system programming. The parts can be programmed through a physical USB connector, but not using the USB protocol. This is extremely powerful for programming or reprogramming on the manufacturing floor, because devices can be in their enclosures in their final commercial form. enCoRe II can also be reprogrammed through firmware. This allows boot-loader' code to be written that can take a data stream from USB, SPI, or any other interface and reprogram the flash real time. Subsets of the flash can also be reprogrammed to provide EEPROM emulation.

Data Memory: Single Stack vs. Separate DSP and PSP Stack

In enCoRe, the Program Stack Pointer (PSP) grows upward from zero and the Data Stack Pointer (DSP) grows downward from the top of the user variable space, as shown in Figure 1. You must be careful to avoid memory conflict issues during DSP and PSP use.

In enCoRe II, a single stack grows from zero. You can determine the amount of memory needed for the stack. (You must pay special attention when arranging the user variable area and stack space to prevent an overflow condition.)

Figure 1. Data Memory Comparison



Endpoint FIFOs

In enCoRe, the data memory area is reserved for a FIFO for USB endpoint (as shown in Figure 1). In enCoRe II, endpoint FIFOs are located in the registers.

Table 1. enCoRe II Endpoint FIFOs

Location	Description	Contents
50–57	EP0DATA Endpoint 0	Data Buffer [7:0]
58–5F	EP0DATA Endpoint 1	Data Buffer [7:0]
60–67	EP0DATA Endpoint 2	Data Buffer [7:0]

Interrupts

This section discusses interrupt similarities and differences.

Interrupt Vectors

There are 12 vectors on enCoRe and 26 vectors in enCoRe II. Table 2 shows some differences between enCoRe and enCoRe II. In general enCoRe II has an equivalent interrupt for every enCoRe interrupt.

1. There are 2 bytes for each interrupt vector in enCoRe compared to 4 bytes in enCoRe II. This may provide slightly more flexibility in cases where special handling is required.
2. In enCoRe II, there are five GPIO interrupt vectors. Each GPIO port (P0-P4) has its own interrupt compared to a combined interrupt for all GPIO ports (P0-P2) in enCoRe.
3. enCoRe II has a 12-bit Programmable Interval Timer and a 16-bit free running counter. A 1-ms (nominal) interrupt is configurable off the programmable interval timer. enCoRe has a 1-ms interrupt and an available 128-µs timer interrupt.
4. enCoRe II has a PS2 Data Low interrupt vector that can be used for PS/2 mode specific purposes.

Table 2. Interrupts

enCoRe Interrupt		Equivalent enCoRe II Interrupt	
Location	**Description**	**Location**	**Description**
0x0000	Start of program execution	0x000	Start of program execution
0x0002	USB Bus Reset	0x002C	USB Bus Reset
0x0004	128-µs timer	0x0038	Programmable Interval Timer
0x0006	1.024-ms timer	0x0034	1-ms Interval Timer
0x0008	USB endpoint 0	0x0020	EP0
0x000A	USB endpoint 1	0x0024	EP1
0x000C	USB endpoint 2	0x0028	EP2
0x000E	SPI	0x000C	SPI Transmitter Empty
		0x0010	SPI Receiver Full
0x0010	Capture timer A	0x003C	Timer Capture 0
0x0012	Capture timer B	0x0040	Timer Capture 1
0x0014	GPIO	0x0014	GPIO Port 0
		0x0018	GPIO Port 1
		0x0050	GPIO Port 2
		0x0054	GPIO Port 3
		0x0058	GPIO Port 4
0x0016	Wake up	0x0064	Sleep Timer
New enCoRe II Interrupts:		0x0004	POR/LVD
		0x0008	INT0
		0x001C	INT1
		0x0030	USB Active
		0x0044	16-bit Free Running Timer Wrap
		0x0048	INT2
		0x004C	PS2 Data Low
		0x005C	Reserved
		0x0060	Reserved

Interrupt Latency

There are slight differences in calculating interrupt latency between the two devices.

enCoRe:

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

enCoRe II:

Latency = Time for current instruction to finish + Time for internal interrupt routine to execute (13 cycles) + Time for LJMP instruction in interrupt table to execute (7 cycles).

Clocking

The similarities and differences in the internal clock, external clock, and clock output are described in this section.

Internal Clock

The enCoRe clocks are fixed: 6 MHz for the USBCLK and 12 MHz for the CPUCLK. On the enCoRe II, the CPU, USB Timer, and Capture Timer clocks are individually configured and can be sourced from the internal 32 kHz or 24 MHz clocks. A divisor is used to divide the internal 24 MHz to support multiple CPU clock configurations. enCoRe II supports the same and higher clocks.

External Clock

For some applications that require a higher clock accuracy, both enCoRe and enCoRe II can select external clocks to meet the higher accuracy requirements. In enCoRe, an external 6 MHz ceramic resonator can be used to provide a higher precision reference for USB operation. An external 6 MHz clock can also be supplied if the XTALOUT pin is left open.

The enCoRe II parts in the CY7C638xx and CY7C633xx families (those most closely related to enCoRe) can accept a clock source of up to 24 MHz on Pin 0.0, but have no support for an external crystal. Devices in the CY7C639xx family also can drive an external oscillator from 1 to 24 MHz. When operating in USB mode, the supplied clock or crystal oscillator must be either 12 MHz or 24 MHz for the USB blocks to function properly.

Clock Output

In enCoRe, the 6 MHz clock is driven out on the XTALOUT pin (by default) when the internal oscillator is in use. By contrast, enCoRe II allows any of the internal clock sources to be driven on the XOUT pin (P0.1) through the CLKOUT select bits of the CLKIOCR register 0x31.

Reset

enCoRe has a Low Voltage Reset (LVR), Brown Out Reset (BOR), and Watchdog Reset (WDR). enCoRe II has a Power On Reset (POR) and Watchdog Reset.

LVR/BOR vs. POR

The enCoRe II POR has equivalent functionality to the enCoRe LVR and BOR. For enCoRe, the LVR holds the part in reset until voltage rises above 3.5 V. BOR places the part in reset when voltage drops below 2.5 V. The LVR is then re-enabled to keep the part in reset until it is above the LVR threshold again. For enCoRe II, the POR handles both the rising and falling voltage conditions. It has a hysteresis of approximately 50 mV and has four configurable settings, with the default value set at 2.7 V. Note that enCoRe II also has a Low Voltage Detection circuit configurable between 2.7 V and 4.8 V.

After LVR, the enCoRe CPU is held off for 24 to 60 ms and starts at 6 MHz. For enCoRe II, the CPU is held for 20 ms after POR and the CPU defaults back to 3 MHz.

Watchdog Reset

Watchdog Reset is always enabled on enCoRe, but it can be enabled on enCoRe II by clearing the PORS bit in the CPU_SCR register. After it is enabled, it cannot be disabled. The Watchdog Reset duty cycles are also different. The enCoRe duty cycle is 10.1~14 ms. The enCoRe II duty cycle defaults to 3 counts of the 32 kHz sleep timer interrupt. The number of cycles before a sleep timer interrupt occurs can be selected by the user. Note that enCoRe II provides an option to allow RAM contents to be preserved through the reset event.

USB Transceiver

This section discusses the voltage regulator, pull up resistor, bus reset, and the manual control of D+/D–.

3.3 V Regulator

Both devices provide a dedicated 3.3 V regulator to support the USB PHY. In enCoRe II, this regulator is in addition to the second 3.3 V regulator provided for the 3.3 V IO pins and the 125 mA output on P1.2. Both devices disable the 3.3 V regulator when the part is placed in sleep mode. However, enCoRe II provides a means to override this with the No Buzz bit of the OSC_CR0 register.

D– Pull Up Resistor

enCoRe provides a dedicated pin for the 1.5K ohm D– pull up resistor, but the resistor itself is external. Because the IO pin accounts for 200 ohms, a 1.3K ohm resistor is used. On enCoRe II, the pull up is integrated, thus saving component cost and also freeing up an IO pin.

For both devices, when the part is in suspend and the 3.3 V regulator is disabled, a sleep pull up resistor is enabled. For enCoRe, the 3.3 V IO pin is pulled up with an additional 6.2 K ohm to V_{CC} if the enable bit is set. In enCoRe II, the sleep pull up of ~7K ohm to V_{CC} replaces the 1.5 K ohm pull up when the pull up enable bit is set.

Bus Reset

Before determining that a bus reset has occurred, enCoRe looks for 100 µs of SE0. enCoRe II uses two cycles of the 32-kHz oscillator plus 2 µs of SE0. Therefore, in enCoRe II, the actual time depends upon where in the 32-kHz clock cycle the bus reset event started. In either case, this is well beyond the minimum 2.5 µs required by the USB specification, but helps to protect against some poorly behaved hosts that may drive an erroneously long SE0 at the end of some packets.

Manual Control of D+/D–

There are different methods of forcing the D± pins on enCoRe and enCoRe II. In enCoRe, the USB signal can be controlled by the Reg 0x1F bit [0:2]. In enCoRe II, the D+/D– (SCK/SDATA) pin can also be configured as a normal GPIO pin when no USB function is needed. Therefore, before controlling the USB signal, firmware must enable the USB Force State bit in USB_XCR register 0x74. Bits [0:1] in the P1Data register 0x01 then control the D+/D– value.

Table 3. enCoRe Manual Control of D+/DP State

D+/D– Forcing Bit [2:0]	Control Action	Application
000	Not Forcing (SIE controls driver)	Any Mode
001	Force K (D+ High, D– Low)	USB Mode
010	Force J (D+ Low, D– High)	
011	Force SE0 (D– Low, D+ Low)	
100	Force D– Low, D+ Low	PS/2 Mode
101	Force D– Low, HiZ	
110	Force D– HiZ, D+ Low	
111	Force D– HiZ, D+ HiZ	

Table 4. enCoRe II Manual Control of D+/D– State

Bit #	7	6	5	4	3	2	1	0
	USB_XCR 0x74							
Field	USB pull up Enable	Reserved						USB Force State
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0
	P1DATA 0x01							
Field	P1.7	P1.6 MISO	P1.5 MOSI	P1.4 SCLK	P1.3 SSEL	P1.2 VREG	P1.1 D–	P1.0 D+
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

Sleep Timer Interrupt / Wakeup Timer

When the microprocessor is put to sleep, enCoRe and enCoRe II have similar timer interrupt control. In enCoRe, the internal wakeup timer is normally used to wake the part from sleep, but it can also provide an interrupt when the part is awake. The wakeup timer is cleared whenever the wakeup interrupt enable bit is written with a '0'; it runs whenever that bit is written with a '1'. When the interrupt is enabled, the wakeup timer provides periodic interrupts at multiples of the period. The period of the wakeup timer can be adjusted by setting the wakeup timer adjust bits in the Clock Configuration register 0xF8.

In enCoRe II, 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. You can set the sleep time period using the sleep timer bits of the OSC_CR0 register 0x1E0. When the sleep time elapses (sleep timer overflows), an interrupt to the sleep timer interrupt vector is generated.

External Clock During Sleep

enCoRe and enCoRe II have the same behavior when using an external clock. The oscillator is stopped during sleep mode. When the CPU comes out of sleep mode, it initially runs on the internal oscillator. The system disables the internal clock and switches back to external clock.

Internal Clock Recovery Time

In enCoRe, the internal oscillator recovery time is 8 µs. In enCoRe II the internal oscillator recovery time is three clock cycles of the internal 32 kHz low-power oscillator.

GPIO

enCoRe and enCoRe II have similar conflagration capabilities, with slightly more flexibility on enCoRe II. On both devices, two complete ports are individually configurable on a per-pin basis. For enCoRe II, the other three ports are configured on a by-port basis.

Pin Flexibility

On enCoRe, certain pins have limited IO capability. These include:

- USB D+/D– pins (alternately usable as PS/2 clock/data)
- Vreg output pin (alternately usable as input in PS/2 mode)
- XTALOUT pin (alternately usable as input with internal oscillator)
- XTALIN pin (alternately usable as input with internal oscillator on CY7C632xx devices only)

enCoRe II has all of these functions, but the pins are fully configurable as GPIOs when those functions are not in use. For enCoRe, only the XTALOUT (and XTALIN for CY7C632xx) are likely to be usable by the application. For enCoRe II, Vreg (P1.2), CLKIN (P0.0), and CLKOUT (P0.1) are all likely to be usable by the application. Excluding the USB pins, that designates the IO capabilities shown in Table 5.

Table 5. IO Capabilities by Package Pin Count

Package	enCoRe IOs	enCoRe II IOs
16 pin	10 (2 are input only)	12
18 pin	11 - CY7C637xx (1 input only) 12 - CY7C632xx (2 are input only)	14
24 pin	17 (1 input only)	18

enCoRe II has an added capability in that port pins P1.3 to P1.6, which are also the SPI pins, can be referenced to either Vcc (5V) or Vreg (3.3 V).

Sink Capability on GPIO Pins

Both the enCoRe and enCoRe II have good sink ability. enCoRe users can choose low, medium, or high sink. Low sink is 2 mA, medium sink is 8 mA, and high sink is 50 mA.

enCoRe II supports only low sink of 8 mA and high sink of 50 mA. All pins can sink 8 mA, but only specific pins support 50-mA sink capability. On the CY7C638xx and CY7C633xx families, only pins P1.7 to P1.3 support 50 mA. The CY7C639xx devices use different pins for high sink.

IO Configuration

For enCoRe, each GPIO pin is configured by a combination of one bit in each of two registers (mode0 and mode1). Interrupt enables and polarities are handled by two additional registers. For enCoRe II, each bit-configurable pin has its own configuration register. Port-configurable registers have one configuration register for the entire port. Other than the sink current differences mentioned previously, the configuration options for the two parts are similar:

- High current drive capability (2 mA)
- High sink capability
- High-impedance input
- Resistive pull up
- Open drain output
- CMOS/TTL input
- CMOS output

Code Example for GPIO Accessing

Accessing port0.0 of enCoRe using IORD/IOWR instructions:

```
Port0_Data:equ 0h
IORD Port0_Data; Read Port0 data value
OR A, 01h; Set p0.0 to 1
IOWR Port0_Data; Write data back to Port0's
data register
```

Accessing port0.0 of enCoRe II using MOV instruction:

```
P0DATA:equ 00h; Defined in m8c.inc
MOV A, REG[P0DATA]; Read P0DATA data value
OR A, 01h; Set p0.0 to 1
MOV REG[P0DATA], A; Write data back to
P0DATA register
```

Serial Peripheral Interface

In enCoRe, only the CY7C637xx family has SPI, whereas all enCoRe II devices have SPI. Both support master and slave modes of operation.

SPI Master Clocking

enCoRe has four selectable SCLK frequencies based on the SCK select bits of the SPI Control register 0x61: 2 Mbit/s, 1 Mbit/s, 0.5 Mbit/s, and 0.0625 Mbit/s.

For enCoRe II, the frequency is setup based on CPUCLK and a divider. Table 6 shows SCLK values for 12 and 24 MHz CPU speeds.

Table 6. SCLK Frequency Selection for enCoRe II

SCLK Select	CPUCLK Divisor	SCLK Frequency when CPUCLK =	
		12 MHz	24 MHz
00	6	2 MHz	4 MHz
01	12	1 MHz	2 MHz
10	48	250 kHz	500 kHz
11	96	125 kHz	250 kHz

Configuration

For enCoRe, the default is MSB sent first; for enCoRe II, a configurable MSB or LSB is sent first.

enCoRe supports only the standard 4-wire configuration: SS, SCLK, MOSI, and MISO. For enCoRe II, in addition to the 4-wire support, you have the option to support a 3-wire mode (SS, SCLK, SDATA) when a half-duplex single-data line is required. MISO and MOSI pin direction can be swapped by firmware with the swap bit of the SPICR register 0x3D.

enCoRe SPI applications normally use the SPI flags for managing the interface. enCoRe II uses interrupt status flags instead.

As mentioned in the GPIO section, enCoRe II SPI pin voltage reference is selectable between external Vcc or the internal 3.3 V source.

Timers

enCoRe II has additional timer capabilities beyond what is supported in enCoRe.

enCoRe has a single 12-bit free-running timer. It is fixed at 1 µs ticks. Reading the lower 8 bits also causes the upper bits to be latched. This register is read-only.

enCoRe II has a 16-bit free running timer. Its source is configurable. It can be referenced to the external clock, internal 24 MHz oscillator, or internal 32 kHz oscillator with a 2, 4, 6, or 8 divider. It can also be disabled. Reading the lower 8 bits also causes the upper bits to be latched. This register can be read or written.

Additionally, enCoRe II has a 12-bit Programmable Interval Timer. This is a down counter with a user-specified reload value. It has a selectable source: external clock, internal 24 MHz oscillator, internal 32 kHz oscillator, or 16-bit free running timer (TACPCLK). There is also a selectable clock divider of 1, 2, 3, or 4. Reading the low-order 8 bits latches the upper bits.

A simplified block diagram of the clocking options for the timers is shown in Figure 2.

Capture Registers

Both enCoRe and enCoRe II have capture registers associated with the free-running timers (12-bit timer on enCoRe, 16-bit timer on enCoRe II). They both have two 8-bit capture registers for both rising and falling edges. Both support interrupts on these events. enCoRe II has an added capability; it is able to cascade the capture registers into 16-bit rising and falling edge capture registers. Both support a pre-scaler; however, there are five pre-scaler choices for enCoRe and eight for enCoRe II.

Interrupts

In addition to the timer capture interrupts, both devices provide additional timer interrupts. enCoRe has two fixed interrupt sources, one at 1.024 ms and one at 128 µs.

With enCoRe II, the interrupt sources are somewhat more flexible. An interrupt is provided for the 16-bit free-running timer wraparound, the 12-bit programmable interval timer reload, and a nominal 1.024-ms interrupt. The 1.024-ms interrupt is based on the assumption that the 16-bit free running timer (TCAPCLK) is running at 4 MHz. Changes in TCAPCLK frequency cause a corresponding change in the 1024-ms interrupt frequency.

Output pulses from the 1.024-ms interrupt and programmable interval timer interrupt can also be generated on port pins P0.5 and P0.6, respectively. When configured for outputs, these signals are not gated by the actual interrupt enables.

CPU Differences

The M8B microprocessor is used in enCoRe, as well as in the previous CY7C634/5/6xx and CY7C64/5/6xxx families. It is a second generation to the M8A that is used in the CY7C63xxx devices. Cypress Microsystems created an additional evolution of the core for the PSoC product line, and it has now been ported back to the USB microcontroller line. The main features differences are summarized in Table 7.

Figure 2. Timer Clocking Block Diagram

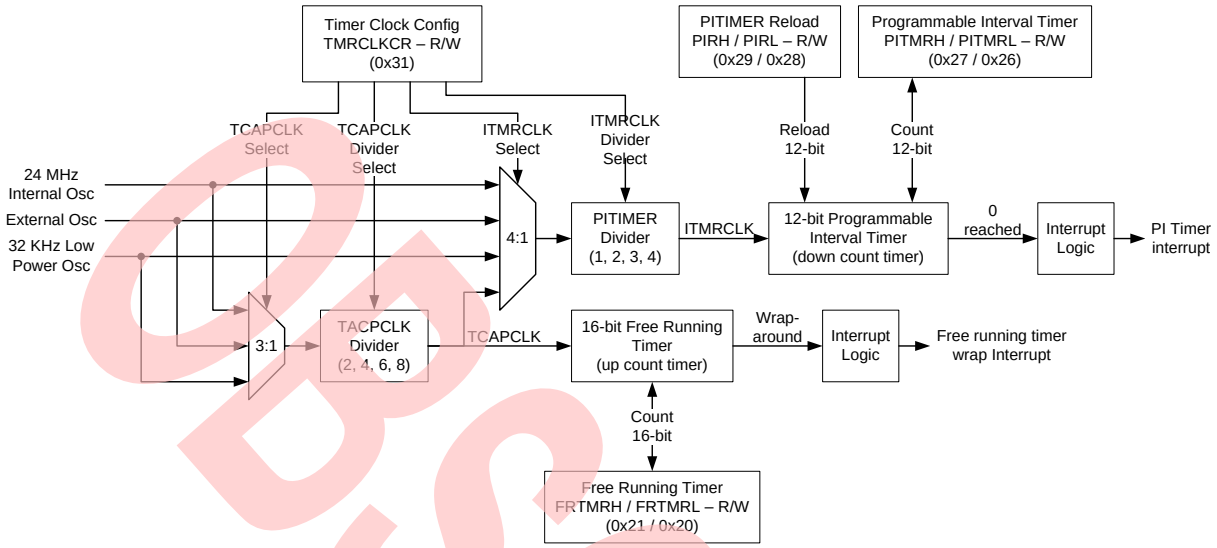


Table 7. General Comparison Between M8B and M8C

Item	M8B	M8C
Accessing across 4 K boundary of 8 K ROM	Limited; can only 'call' from lower to upper 4 K	No limitations (LJMP, LCALL instructions)
Instruction Set	Irregular/non-orthogonal	Many more instructions, nearly orthogonal access
Program/Data Stack	Two separate stacks, grow toward each other	Single stack
Bit Test/Set/Clear	No direct support	Supported with expanded instruction set
The items below are also differences, although the current enCoRe II does not take advantage of them. Future enCoRe II devices are expected to use these features.		
ROM size access	Limited to 8 K	Access up to 64 K
RAM size access	Limited to 256 bytes	Access up to 4 K (extended address bits, can be divided between RAM and IO registers)
Crossing 256-byte page boundaries	Assembler inserts XPAGE automatically	No XPAGE; microprocessor adds one clock cycle on page crossings

Other important M8C changes are:

- The Carry and Zero flags have moved to a new FLAG register.
- Because of the separate flag register, interrupts (and reti) store (re-store) are three bytes instead of two.
- The Interrupt Enable signal, previously handled with EI/DI instructions, is now a bit in the flag register. Logical operations on the flag register perform the equivalent EI/DI function.
- Generally, most instructions have four new addressing modes for a total of seven for each function.

Assembler Instruction Differences Between enCoRe and enCoRe II

There are differences in compatibility and instruction timing between the M8B and M8C instruction set. Table 8 summarizes these differences.

Table 8. M8B and M8C Instruction Comparison

M8B			M8C			Δ Cycles	Notes
Opcode	Instruction	Cycles	Equivalent Instruction	Cycles	Opcode		
00	HAL	7	HALT	9	30	2	[1]
01	ADD A, expr	4	ADD A, k	4	01	0	
02	ADD A,[expr]	6	ADD A, M[k]	6	02	0	
03	ADD A,[X+expr]	7	ADD A, M[X+k]	7	03	0	
04	ADC A, expr	4	ADC A, k	4	09	0	
05	ADC A,[expr]	6	ADC A, M[k]	6	0A	0	
06	ADC A,[X+expr]	7	ADC A, M[X+k]	7	0B	0	
07	SUB A, expr	4	SUB A, k	4	11	0	
08	SUB A,[expr]	6	SUB A, M[k]	6	12	0	
09	SUB A,[X+expr]	7	SUB A, M[X+k]	7	13	0	
0A	SBB A, expr	4	SBB A, k	4	19	0	
0B	SBB A,[expr]	6	SBB A, M[k]	6	1A	0	
0C	SBB A,[X+expr]	7	SBB A, M[X+k]	7	1B	0	
0D	OR A, expr	4	OR A, k	4	29	0	
0E	OR A,[expr]	6	OR A, M[k]	6	2A	0	
0F	OR A,[X+expr]	7	OR A, M[X+k]	7	2B	0	
10	AND A, expr	4	AND A, k	4	21	0	
11	AND A,[expr]	6	AND A, M[k]	6	22	0	
12	AND A,[X+expr]	7	AND A, M[X+k]	7	23	0	
13	XOR A, expr	4	XOR A, k	4	31	0	
14	XOR A,[expr]	6	XOR A, M[k]	6	32	0	
15	XOR A,[X+expr]	7	XOR A, M[X+k]	7	33	0	
16	CMP A, expr	5	CMP A, k	5	39	0	
17	CMP A,[expr]	7	CMP A, M[k]	7	3A	0	
18	CMP A,[X+expr]	8	CMP A, M[X+k]	8	3B	0	
19	MOV A,expr	4	MOV A, k	4	50	0	[2]
1A	MOV A,[expr]	5	MOV A, M[k]	5	51	0	[3]

[1] The M8B Halt instruction operates by clearing the FFh register; M8C's halt adds one to this register.

[2] In M8B, moves (or IORD) to the accumulator do not affect the zero flag; in M8C, they do.

[3] In M8B, moves (or IORD) to the accumulator do not affect the zero flag; in M8C, they do.

M8B			M8C			Δ Cycles	Notes
Opcode	Instruction	Cycles	Equivalent Instruction	Cycles	Opcode		
1B	MOV A,[X+expr]	6	MOV A, M[X+k]	6	52	0	[4]
1C	MOV X,expr	4	MOV X, k	4	57	0	
1D	MOV X,[expr]	5	MOV X, M[k]	6	58	1	
1E	reserved	4					
1F	XPAGE	4					[5]
20	NOP	4	NOP	4	40	0	
21	INC A	4	INC A	4	74	0	
22	INC X	4	INC X	4	75	0	
23	INC [expr]	7	INC M[k]	7	76	0	
24	INC [X+expr]	8	INC M[X+k]	8	77	0	
25	DEC A	4	DEC A	4	78	0	
26	DEC X	4	DEC X	4	79	0	
27	DEC [expr]	7	DEC M[k]	7	7A	0	
28	DEC [X+expr]	8	DEC M[X+k]	8	7B	0	
29	IORD expr	5	MOV A, IO[k]	6	5D	1	[6]
2A	IOWR expr	5	MOV IO[k], A	5	60	0	
2B	POP A	4	POP A	5	18	1	
2C	POP X	4	POP X	5	20	1	
2D	PUSH A	5	PUSH A	4	08	-1	
2E	PUSH X	5	PUSH X	4	10	-1	
2F	SWAP A,X	5	SWAP A, X	5	4B	0	
30	SWAP A,DSP	5					
31	MOV [expr],A	5	MOV M[k], A	5	53	0	
32	MOV [X+expr],A	6	MOV M[X+k], A	6	54	0	
33	OR [expr],A	7	OR M[k], A	7	2C	0	
34	OR [X+expr],A	8	OR M[X+k], A	8	2D	0	
35	AND [expr],A	7	AND M[k], A	7	24	0	
36	AND [X+expr],A	8	AND M[X+k], A	8	25	0	
37	XOR [expr],A	7	XOR M[k], A	7	34	0	
38	XOR [X+expr],A	8	XOR M[X+k], A	8	35	0	
39	IOWX [X+expr]	6	MOV IO[X+k], A	6	61	0	

[4] In M8B, moves (or IORD) to the accumulator do not affect the zero flag; in M8C, they do.

[5] XPAGE is replaced by automatic increment of the high program-counter byte during 256-byte page crossings, and this adds a single cycle to the instruction crossing the page.

[6] In M8B, moves (or IORD) to the accumulator do not affect the zero flag; in M8C, they do.

M8B			M8C			Δ Cycles	Notes
Opcode	Instruction	Cycles	Equivalent Instruction	Cycles	Opcode		
3A	CPL	4	CPL A	4	73	0	
3B	ASL	4	ASL A	4	64	0	
3C	ASR	4	ASR A	4	67	0	
3D	RLC	4	RLC A	4	6A	0	
3E	RRC	4	RRC A	4	6D	0	
3F	RET	8	RET	8	7F	0	
40	MOV A,X	4	MOV A, X	4	5B	0	
41	MOV X,A	4	MOV X, A	4	5C	0	
50	CALL	10	LCALL k, i	13	7C	3	[7]
60	MOV PSP,A	4	SWAP A, SP	5	4E	1	[8]
70	DI	4	AND F, k	4	70	0	[9]
72	EI	4	OR F, k	4	71	0	[10]
73	RETI	8	RETI	10	7E	2	
80	JMP	5	JMP k	5	80	0	[11]
90	CALL	10	CALL k	11	90	1	[12]
A0	JZ (false)	4	JZ k	4	A0	0	[13]
A1	JZ (true)	5	JZ k	5	A1	0	[14]
B0	JNZ (true)	5	JNZ k	5	B0	0	[15]
B1	JNZ (false)	4	JNZ k	4	B1	0	[16]
C0	JC (false)	4	JC k	4	C0	0	[17]
C1	JC (true)	5	JC k	5	C1	0	[18]
D0	JNC (true)	5	JNC k	5	D0	0	[19]

[7] The long call instruction, LCALL, is a 3-byte instruction (compared to 2 bytes for M8B calls).

[8] There is no direct equivalent for this seldom used instruction; if the accumulator must be preserved in M8C, this sequence could be used: push a; swap a,sp; pop a.

[9] For DI, the equivalent is AND F,FEh (takes 2 bytes vs. 1 byte for DI).

[10] For EI, the equivalent is OR F, 1 (takes 2 byes vs 1 byte for EI).

[11] In M8C, jumps, calls, and index instructions are +2k from the preset location, while in M8B these cover the current 4k page.

[12] In M8C, jumps, calls, and index instructions are +2k from the preset location, while in M8B these cover the current 4k page.

[13] In M8C, jumps, calls, and index instructions are +2k from the preset location, while in M8B these cover the current 4k page.

[14] In M8C, jumps, calls, and index instructions are +2k from the preset location, while in M8B these cover the current 4k page.

[15] In M8C, jumps, calls, and index instructions are +2k from the preset location, while in M8B these cover the current 4k page.

[16] In M8C, jumps, calls, and index instructions are +2k from the preset location, while in M8B these cover the current 4k page.

[17] In M8C, jumps, calls, and index instructions are +2k from the preset location, while in M8B these cover the current 4k page.

[18] In M8C, jumps, calls, and index instructions are +2k from the preset location, while in M8B these cover the current 4k page.

[19] In M8C, jumps, calls, and index instructions are +2k from the preset location, while in M8B these cover the current 4k page.

M8B			M8C			Δ Cycles	Notes
Opcode	Instruction	Cycles	Equivalent Instruction	Cycles	Opcode		
D1	JNC (false)	4	JNC k	4	D1	0	[20]
E0	JACC	7	JACC k	7	E0	0	[21]
F0	INDEX	14	INDEX k	13	F0	-1	[22, 23]

New Instructions in M8C

The larger ROM size in M8C allows for many more instructions. The additional instructions (new ones not covered in the above M8B-equivalent set) are listed in Table 9. These can be categorized as offering the following capabilities, listed roughly in descending order of typical usefulness:

1. Bit test/set/clear/toggle operations on IO registers.
2. New addressing modes, mainly for direct memory operations (for example, AND M[k],a) — including operating with immediate values (3-byte instructions).
3. Expanded set of move/swap choices.
4. The LJMP, with the LCALL shown previously, are 3-byte automatically assembled jump/call instructions that allow movement around the full program memory (up to 64 k) without restrictions.
5. Indirect addressing into RAM with an auto-incrementing pointer (MVI instructions).
6. Logical operations on the new flag register (including the AND F,k and OR F,k shown above as EI/DI substitutes).
7. SSC, supervisory system call. This allows access to a supervisory ROM for user functions, such as programming the flash memory.
8. ROMX, which indexes a byte of program memory by concatenating the A and X registers.
9. Ability to directly add an immediate value to the (single) stack pointer.

Table 9. New M8C Instructions

Opcode	Name	Cycles		Opcode	Name	Cycles
00	SWI	15		43	OR IO[k],i	9
04	ADD M[k],A	7		44	OR IO[X+k],i	10
05	ADD M[x+k],A	8		45	TST M[k],i	9
06	ADD M[k],i	9		46	XOR IO[X+k],i	10
07	ADD M[X+k],i	10		47	TST M[k],i	8
0C	ADC M[k],A	7		48	TST M[X+k],i	9
0D	ADC M[X+k],A	8		49	TST IO[k]	8
0E	SUB M[k],A	9		4A	TST IO[X+k],i	9
0F	SUB M[X+k],A	10		4C	SWAP A,M[k]	7
14	SUB M[k],i	7		4D	SWAP X,M[k]	7

[20] In M8C, jumps, calls, and index instructions are +2k from the preset location, while in M8B these cover the current 4k page.

[21] For DI, the equivalent is AND F,FEh (takes 2 bytes vs. 1 byte for DI).

[22] In M8C, jumps, calls, and index instructions are +2k from the preset location, while in M8B these cover the current 4k page.

[23] In M8B, INDEX temporarily uses one byte of stack space; the M8C INDEX does not use the stack.

Opcode	Name	Cycles		Opcode	Name	Cycles
15	SUB M[X+k],i	8		4E	SWAP A,SP	5
16	SUB M[k],i	9		4F	MOV X,SP	4
17	SUB M[X+k],i	10		55	MOV M[k],i	8
1C	SBB M[k],A	7		56	MOV M[X+k],i	9
1D	SBB M[X+k],A	8		59	MOV X,M[X+k]	7
1E	SBB M[k],i	9		5A	MOV M[k],X	5
1F	SBB M[X+k],i	10		5E	MOV A,IO[X+k]	7
26	AND M[k],i	9		5F	MOV M[i],M[k]	10
27	AND M[X+k],i	10		62	MOV IO[k],i	8
28	ROMX	11		63	MOV IO[X+k],i	9
2E	OR M[k],i	9		65	ASL M[k]	7
2F	OR M[X+k],i	10		66	ASL M[X+k]	8
36	XOR M[k],i	9		68	ASR M[k]	7
37	XOR M[X+k],i	10		69	ASR M[X+k]	8
38	ADD SP,i	5		6B	RLC M[k]	7
3C	CMP M[k],i	8		6C	RLC M[X+k]	8
3D	CMP M[X+k],i	9		6E	RRC M[k]	7
3E	MVI A[M[k]++]	10		6F	RRC M[X+k]	8
3F	MVI M[M[k]++],A	10		72	XOR F,K	4
41	AND IO[k],i	9		7D	LIMP k,i	7
42	AND IO[X+k],i	10				

Development Environment

enCoRe and enCoRe II use completely different development environments. The new environment, built around the PSoC Designer software, is now being used for many Cypress product families including PSoC (Programmable System on Chip), PRoC (Programmable Radio on Chip), enCoRe II, enCoRe III, and Wireless enCoRe II. This provides a substantial advantage to customers who are designing a range of products using different Cypress chips by reducing the cost of tools as well as minimizing the learning curve for future products.

Development Hardware

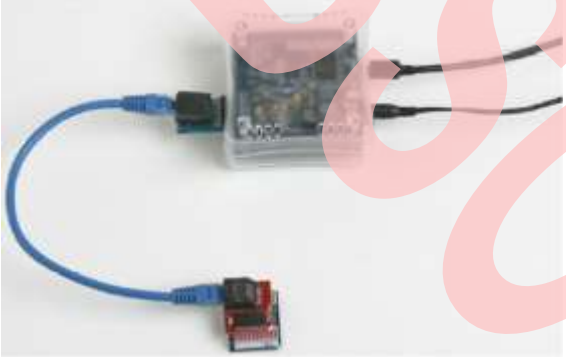
The development kit for enCoRe is an FPGA-based emulation system. The base platform for the emulator is the CY3654 Platform Board. This base board supports three different personality boards for various Cypress USB families. In particular, the CY3654-P05 is the personality board for the enCoRe family (shown in Figure 3). There are certain features of the part that cannot be reliably emulated using this environment.

Figure 3. CY3554 Platform Board

enCoRe II uses the ICE-Cube In Circuit Emulator. The ICE-Cube has a USB interface to the PC and is tightly integrated with the PSoC Designer software tool. The ICE-Cube connects to the target hardware using a pod or flex-pod (different architectures have been used for different device families, but all have the same essential function).

One thing to note about this architecture is that the pod uses actual silicon from the target device family that contains the addition of an On-chip Debugger (OCD) interface. The use of real silicon greatly increases the reliability of development and debug compared to an FPGA-based system. The separate device with the OCD interface means that production devices are spared from having to pay the added cost burden of incorporating this interface.

Figure 4. ICE-Cube In-Circuit Emulator



Development Software

With the enCoRe devices, two separate environments were provided for firmware development. Assembly developers (by far the most common) used the Cypress CYASM assembler. The ByteCraft C compiler was sold separately for C developers. Both were command-line based tools.

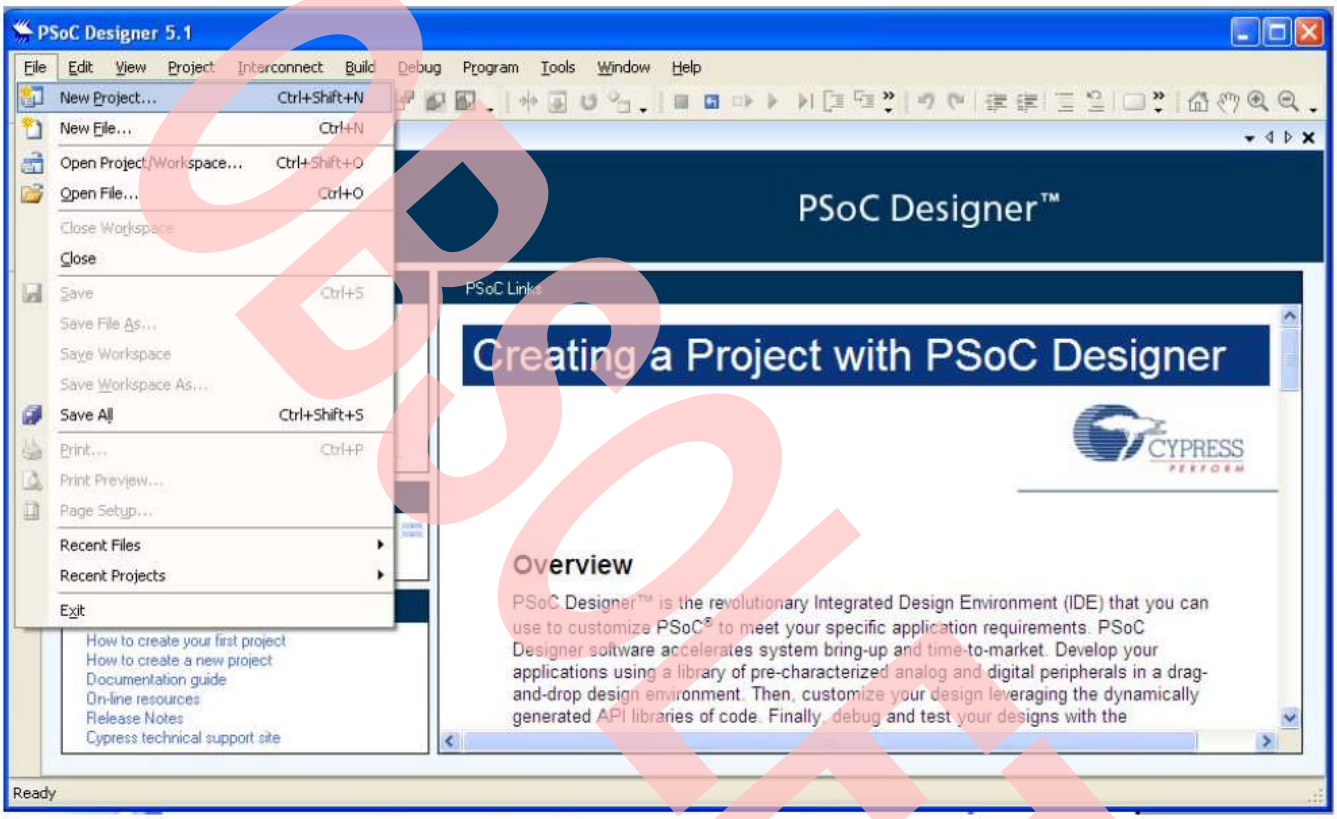
For enCoRe II, the PSoC Designer Integrated Development Environment provides a complete solution. It incorporates a graphical device configuration interface, code editor, assembler, C compiler, linker and debugger — and is available free for download from http://www.cypress.com.

It is not the intent of this application note to give detailed guidance on the use of this tool; however, a brief overview is worthwhile. Cypress provides ample information to assist customers with the details. User Guides are provided as part of the download, and Cypress also offers multiple Tele-Training courses to provide further assistance. These courses are offered periodically in a live web-based forum, but the presentation materials are also available for download from the Cypress website at any time. Tele-Training Module 1 provides a basic overview on the use of the PSoc Designer software.

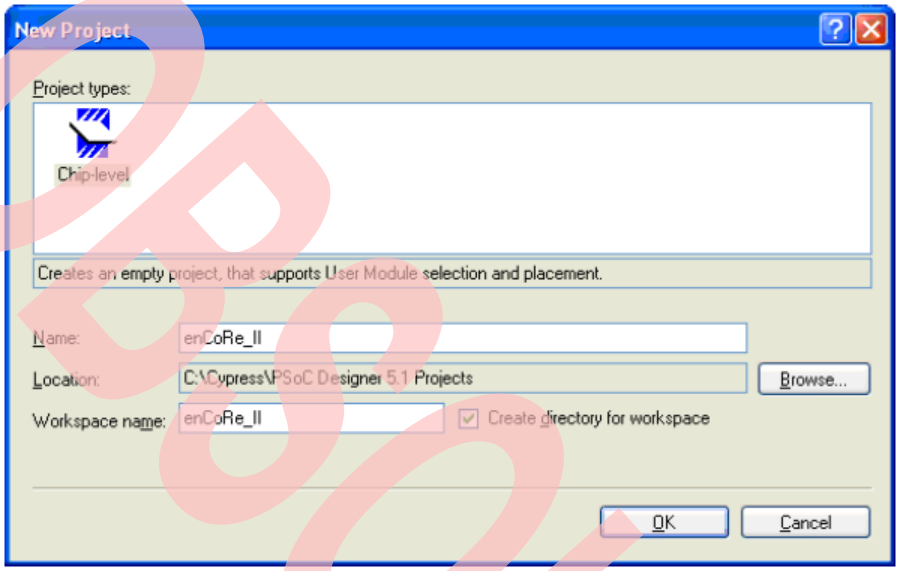
Getting Started on a PSoC Designer Project

When you start PSoC Designer by double-clicking on a project file, the tool opens and loads that project. When PSoC Designer is launched independently, you must take a few steps to create a new project:

1. Choose **Start New Project.** In the Start Page, you can start new project by click **File** > **New Project**. Note that you also have option to open an existing project under File menu.



2. Create the new project by doing the following:
   - Choose Chip-level as project type
   - Type the name of your project (there are restrictions on the file names allowed: only letters, numbers, and underscore "_").
   - Browse to the appropriate location for your project.
   - Check "**Create directory for workspace**" if you wish to.
   - Select **OK**. You are prompted to create the new directory.



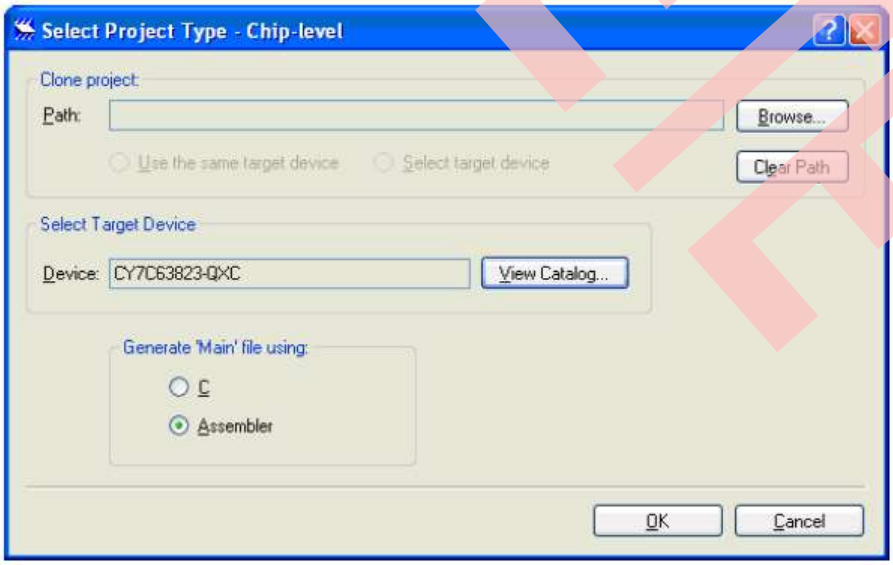
3. Perform the final steps to create the project:

a. You can clone the Device type by specifying the existing project path

b. You can also manually select the part number. Click the **View Catalog** button for a complete list, filterable by features.

c. Choose **C** or **Assembly** for you 'Main' file.

d. Click **OK**.

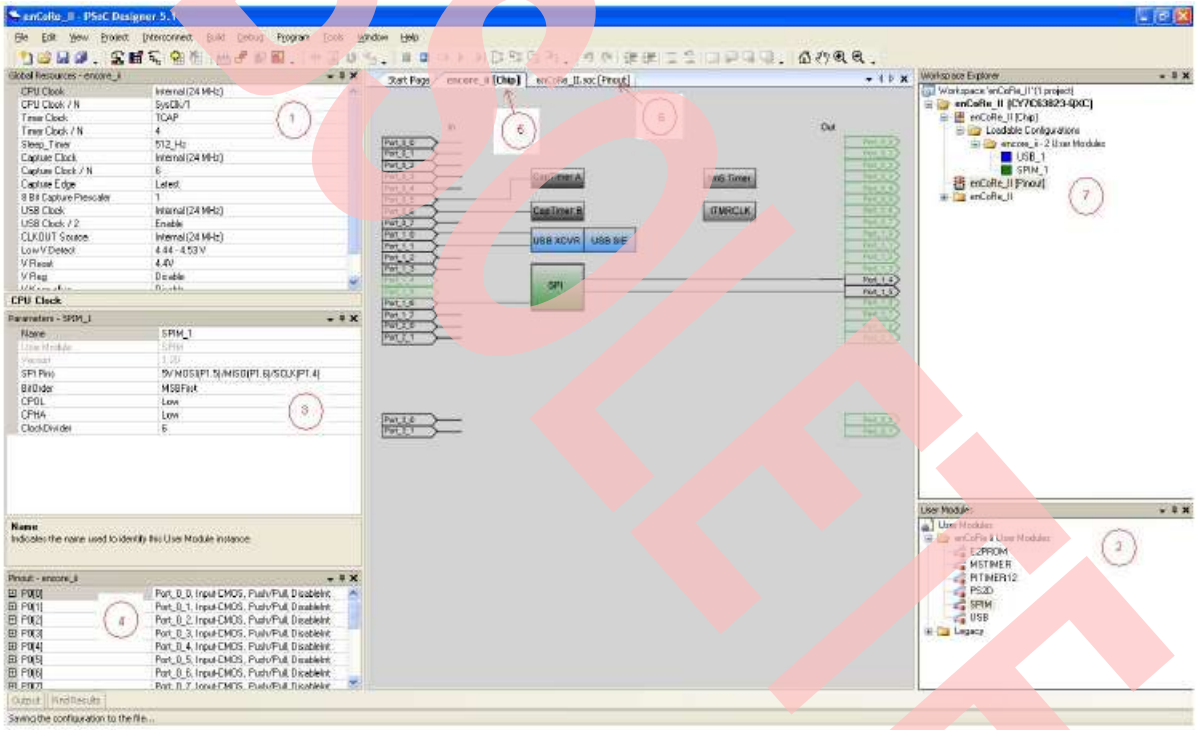


Using the Device Editor

Refer to Figure 5 to clarify the following information:

- Global Resource (1): Here you can setup some basic design parameters, such as CPU clock, LVD, WDT, and so on.
- User Module Tray(2): Allows placement of User Modules. If you need to reference information for a user module(such as APIs), right click on the user module and choose Datasheet.
- User Module Parameter Selection (3): Here you can setup User Module parameters.
- GPIO Configuration (4): Here you can setup the initial IO configuration.
- Floorplan Window(5): Here you can see the status of all user modules and port connections.
- Pin Out Window (6): After you finish all interconnection, the pin assignment chart is displayed. All pin names are printed on the IC pin package.
- Workspace Explorer (7): All source files are displayed in this window. Separate folders are provided for application and library files, source and headers.

Figure 5. Device Editor Floorplan View

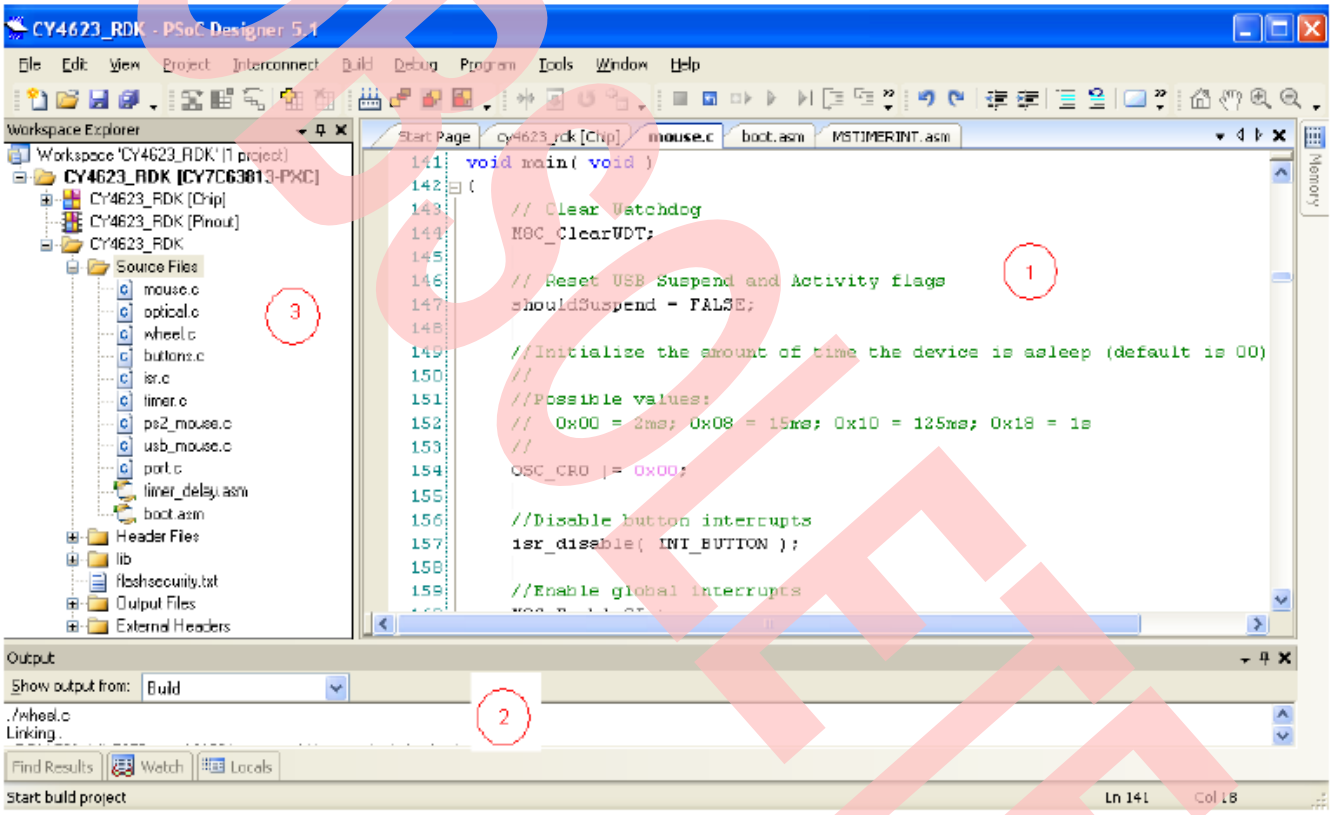


Using the Application Editor

Refer to Figure 6 to clarify the following information:

- Firmware Code Window (1): You can modify or write code in this window.
- Build Status (2): This information is displayed when you presse the **Build** button. The complier status is also displayed.
- Files List Window (3): All source files are displayed in this window. Separate folders are provided for application and library files, source and headers.

Figure 6. Application Editor

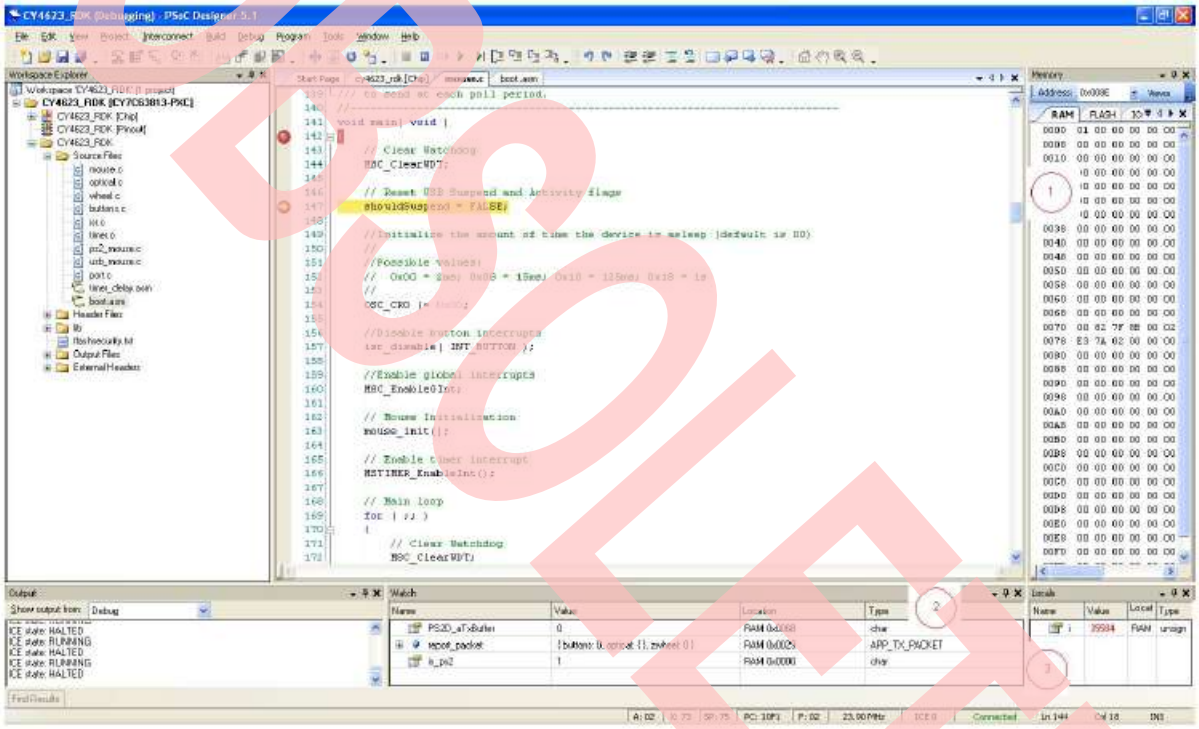


Using the Debugger

Refer to Figure 7 to clarify the following information:

- You can see the register status from this window (1); it is a very useful window for step-by-step debugging.
- The global variable window (2) displays the value of the global variables.
- If you need to check the status of a local variable, insert breakpoints and bookmarks in the routine. After running the program, the value of the local variables is displayed.

Figure 7. Using the Debugger



Using enCoRe II Features and Functional Blocks

This section gives additional guidance on how to use of some of the enCoRe II functions. Because of the new PSoC Designer environment, this approach may be very different from what traditional enCoRe customers would expect.

Using the 1-ms Timer Interrupt

Although the 1-ms timer interrupt is actually configurable to intervals other than 1 ms, most customers will probably want to specify a 1-ms periodicity. To set this up, make sure that the 'MSTIMER' User Module is selected and placed. To achieve a 1-ms interval, the most common selections would be to use the 24 MHz internal oscillator as a reference with a divider of 6, as shown in Figure 8. Recall that the 1-ms interrupt is based on a 4 MHz TCAPCLK. If the clock source, clock source frequency, or capture divider change, the period of the interrupt changes.

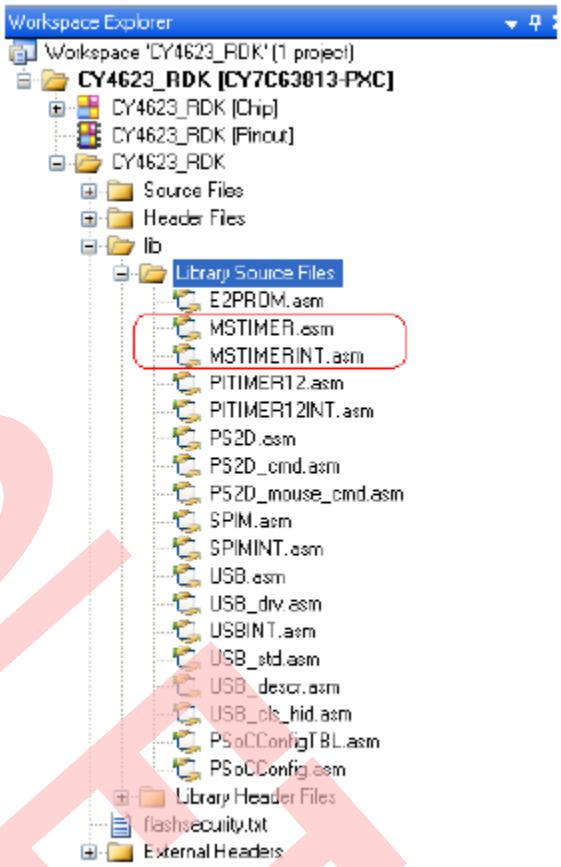
Figure 8. 1 ms Timer Resource Configuration

Global Resources - cy4623_rdk

CPU Clock	Internal (24 MHz)
CPU Clock / N	SysClk/1
Timer Clock	Internal (24 MHz)
Timer Clock / N	2
Sleep_Timer	512_Hz
Capture Clock	Internal (24 MHz)
Capture Clock / N	6
Capture Edge	Latest
8 Bit Capture Prescaler	1
USB Clock	Internal (24 MHz)
USB Clock / 2	Enable
CLKOUT Source	Internal (24 MHz)
Low V Detect	4.44 - 4.53 V
V Reset	4.4V
V Reg	Disable
V Keep-alive	Disable
Watchdog Enable	Disable

When the User Module is selected, various firmware files are added to the project to provide the APIs, as shown in Figure 9.

Figure 9. Library Code Files for MSTIMER User Module



You can build firmware code based on the sample code included. Note the specific location in the code segment below for the insertion of custom application code.

Code 1. Adding Custom Code to the 1-ms ISR Routine

```
_MSTIMER_ISR:
   ;@PSoC_UserCode_BODY_1@ (Do not change this line
   ;---------------------------------------------
   ; Insert your custom code below this banner
   ;---------------------------------------------
   ;   NOTE: interrupt service routines must preser
   ;   the values of the A and X CPU registers.
   Add custom code here
   ;---------------------------------------------
   ; Insert your custom code above this banner
   ;---------------------------------------------
   ;@PSoC_UserCode_END@ (Do not change this line.)
   RETI
; end of file MSTIMERINT.asm
```

The following is an example:

Code 2. Subroutine for 1-ms Timer Interrupt

```
_MSTIMER_ISR:
   ;@PSoC_UserCode_BODY_1@ (Do not change this lin
   ;---------------------------------------------
   ; Insert your custom code below this banner
   ;---------------------------------------------
   ;   NOTE: interrupt service routines must prese
   ;   the values of the A and X CPU registers.
   PUSH A; Save context
   PUSH X
   ; Call a subroutine to count the time
   LCALL ms_timer_isr
   POP X; Restore context
   POP A
   ;---------------------------------------------
   ; Insert your custom code above this banner
   ;---------------------------------------------
   ;@PSoC_UserCode_END@ (Do not change this line.)
   RETI
; end of file MSTIMERINT.asm
```

In the new subroutine you can, for example, manage various flags to handle timed based events.

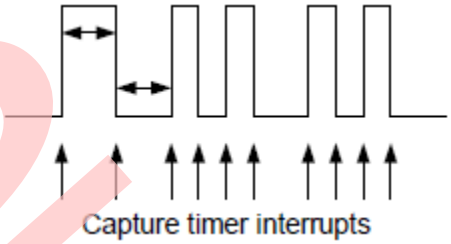
Capture Timer

There is no User Module for the capture timers. 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. Detailed information can be found in the enCoRe II Data Sheet, registers 0x40 - 0x45).

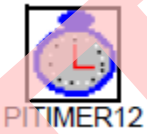
The capture timers interrupt whenever a new timer value is saved due to a selected GPIO edge event. A common use for the capture timers is on wireless receivers where it is necessary to decode an incoming RF signal. The capture timer capability is used to measure the received pulse widths.

To use the capture timer, you must configure the 16-bit free-running timer (TCAPCLK) as discussed in the previous section. To use the interrupts, you must enable the TCAP0 and/or TACP1 interrupts (Interrupt Mask 1 register 0xE1 and Interrupt Mask 2 register 0xDF respectively) as well as the applicable rising and/or falling edge interrupts in the Capture Interrupt Enable register 0x2B. Timing a pulse width involves enabling both rising and falling edge interrupts and then calculating the difference between the two captured values.

Figure 10. Capture Timer Usage

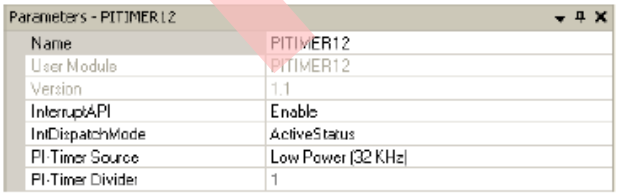


12-Bit Programmable Interval Timer



Two parameters affect the programmable timer period: one is the PI-Timer Source and the other is the PI-Timer Divider. These can be configured within the User Module resources sections of PSoC Designer, as shown in Figure 11).

Figure 11. 12-Bit PIT Configuration Options



The User Module also adds additional library source files to the application. Figure 9 shows the 'pitimer12...' files. Within an application, firmware can adjust the PI-Timer Source and PI-Timer Divider dynamically to change the resolution if needed.

At startup, the timer begins to count down and the interrupt occurs on the count of '0'. At that moment, you can reset the period of the next timer interrupt. Custom code can be added to the default interrupt handler as follows.

Code 3. Add Custom Code to the12-bit PIT ISR

```
;-----------------------------------------------
; FUNCTION NAME: _PITIMER12_ISR
;
; DESCRIPTION: Calls the UM specified Callback
; function if it was enabled
;
;-----------------------------------------------
_PITIMER12_ISR:
   ;@PSoC_UserCode_BODY_1@ (Do not change this lin
   ;-----------------------------------------------
   ; Insert your custom code below this banner
   ;-----------------------------------------------
   ;   NOTE: interrupt service routines must prese
   ;   the values of the A and X CPU registers.
        Add custom code here
   ;-----------------------------------------------
   ; Insert your custom code above this banner
   ;-----------------------------------------------
   ;@PSoC_UserCode_END@ (Do not change this line.)
   RETI
; end of file PITIMER12INT.asm
```

The Sleep Timer Interrupt

The sleep timer is a very important function that can be used in low-power mode (microcontroller sleep) to trigger a periodic wakeup event. The following four steps must be followed to setup the sleep timer interrupt:

1. Make sure the Sleep Timer Interrupt is included on the interrupt vector table.

```
org 64h ; Sleep Timer Interrupt Vector
ljmp_SleepTimer_ISR
reti
```

2. Edit the content for the SleepTimer_ISR.

```
void _SleepTimer_ISR(void)
{
    // Put your code here
    return;
}
```

3. Setup the Sleep Timer period using the Sleep Timer bits in the OSC_CR0 register 0x1E0:

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	1s	3s

4. Enable the Sleep Timer interrupt

   Firmware handles any housekeeping events, such as placing USB into suspend, placing IOs into low-power modes, and so on. An example follows.

Code 4. Sleep Example

```
//Enable the sleep timer interrupt
//When the sleep timer interrupt occurs,
//it will simply blink LED
//Set the amount of sleep timer interrupt period
1s

    OSC_CR0 |= 0x11;
    M8C_EnableIntMask(INT_MSK0, INT_MSK0_SLEEP);
    M8C_EnableGInt;

    while(1)
    {
      //Clear Watchdog and Sleep
       M8C_ClearWDTAndSleep;

       //The next instruction after the M8C_Sleep
       //is prefetched/ We insert two NOPs so the
       //compare for the loop isn't prefetched.

       //sleep
       M8C_Sleep;
       asm("nop");
       asm("nop");
    }

void SleepTimer_ISR(void)
{
    char pt0;
    pt0 = PRT0DR;
    if (pt0 & (1<<4))
        pt0 &= ~(1<<4);
    else
        pt0 |= (1<<4);
    PRT0DR = pt0;
}
```

Serial Peripheral Interface

There are multiple configuration options provided within PSoC Designer to select all of the SPI options that were discussed earlier in this application note. These are shown in Figure 12 and include:

- IntDispatchMode: ActiveStatus or OffsetPreCalc
- SPI pin voltage potential (3.3 V versus 5 V)
- MSB first versus LSB first
- SPI clock idle polarity (high or low)
- Clock phase for sampling data
- Clock divider for SPI clock

Figure 12. SPI

Parameters - SPIM

Name	SPIM
User Module	SPIM
Version	1.20
IntDispatchMode	ActiveStatus
SPI Pins	5V SDIO(P1.5)/SCK(P1.4)
BitOrder	MSBFirst
CPOL	High
CPHA	High
ClockDivider	96

The option to support a 3-wire (SS, SCLK, SDATA) must be managed in firmware due to the need to toggle the Swap bit to change direction.

APIs for the SPI User Module can be found by referring to the User Module Data Sheet in PSoC Designer.

Sample firmware source code could also be found in the SPIM module datasheet.

GPIO Interrupts

Multiple GPIO interrupts support both TTL or CMOS thresholds. For additional flexibility, the interrupt polarity is programmable to be either the rising or falling edge. Using a GPIO interrupt requires the following steps:

- Set the Interrupt mode in the GPIO pin block.
- Enable the bit interrupt in the GPIO block.
- Set the mask bit for the (global) GPIO interrupt.
- Assert the overall Global Interrupt Enable.

The multiple GPIO interrupt vectors can be seen in the *boot.asm* code generated by PSoC Designer.

Code 5. Subset of Interrupt Vectors in boot.asm

```
org 08h ;INT0 Interrupt Vector
ljmpINT0_ISR
reti

org 0Ch ;SPI TX Empty Interrupt Vector
ljmp_SPIM_1_TX_ISR
reti

org 10h ;SPI RX Full Interrupt Vector
ljmp_SPIM_1_RX_ISR
reti

org 14h ;GPIO Port 0 Interrupt Vector
ljmpPORT0_ISR
reti

org 18h ;GPIO Port 1 Interrupt Vector
ljmpPORT1_ISR
reti

org 1Ch ;INT1 Interrupt Vector
ljmpINT1_ISR
reti
```

When the interrupt occurs, the firmware will jump to the routine below. Custom code can be added in the appropriate section to handle the event.

Code 6. GPIO Interrupt Handler

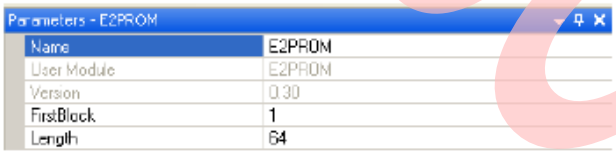
```
INT1_ISR:
   ;@PSoC_UserCode_BODY_2@ (Do not change this lin
   ;-------------------------------------------
   ; Insert your custom code below this banner
   ;-------------------------------------------
   ;   NOTE: interrupt service routines must prese
   ;   the values of the A and X CPU registers.
   Add custom code here
   ;-------------------------------------------
   ; Insert your custom code above this banner
   ;-------------------------------------------
   ;@PSoC_UserCode_END@ (Do not change this line.)
   RETI
```

E2PROM User Module

The E2PROM User Module is a software algorithm that uses no enCoRe II hardware resources. One or more instances of these E2PROM virtual devices can be created.

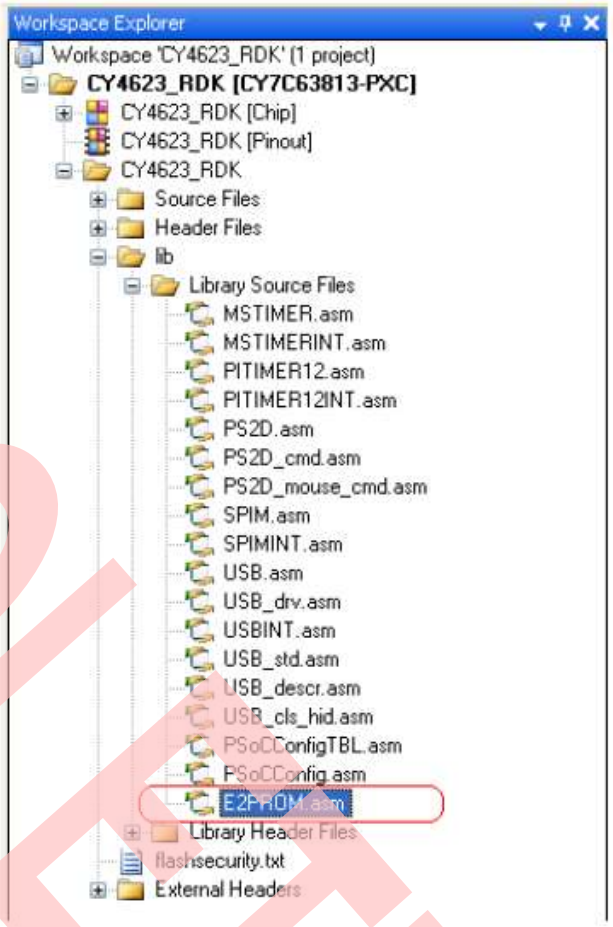
The flash is organized in 64 byte blocks in all enCoRe II devices. The architecture allows the flash data to be read on a byte-by-byte basis, but requires the data to be written on a block-by-block basis — 64 bytes at a time. The intent of this user module is to emulate an EEPROM device (a byte-read, byte-write oriented device) on a Flash-based memory device (a byte-read, block-write oriented device). To use the E2PROM User Module, you must still configure it on block boundaries, as shown in Figure 13.

Figure 13. E2PROM Configuration Options



When using the user module, library files are added as shown in Figure 14.

Figure 14. Library Code Files for E2PROM User Module



Sample firmware source code could also be found in the E2PROM module datasheet.

Note that the E2PROM user module has a fairly large code footprint (approximately 800 bytes) to provide its flexibility. Cypress also has reference code for simplified, smaller flash read/write routines for cases where 64 bytes or less of EEPROM is needed (for example, a single flash block). These are currently included in our WirelessUSB keyboard/mouse reference designs, which can be downloaded from the Cypress website. Refer to the *flash.asm* routines in the CY4636.

USB

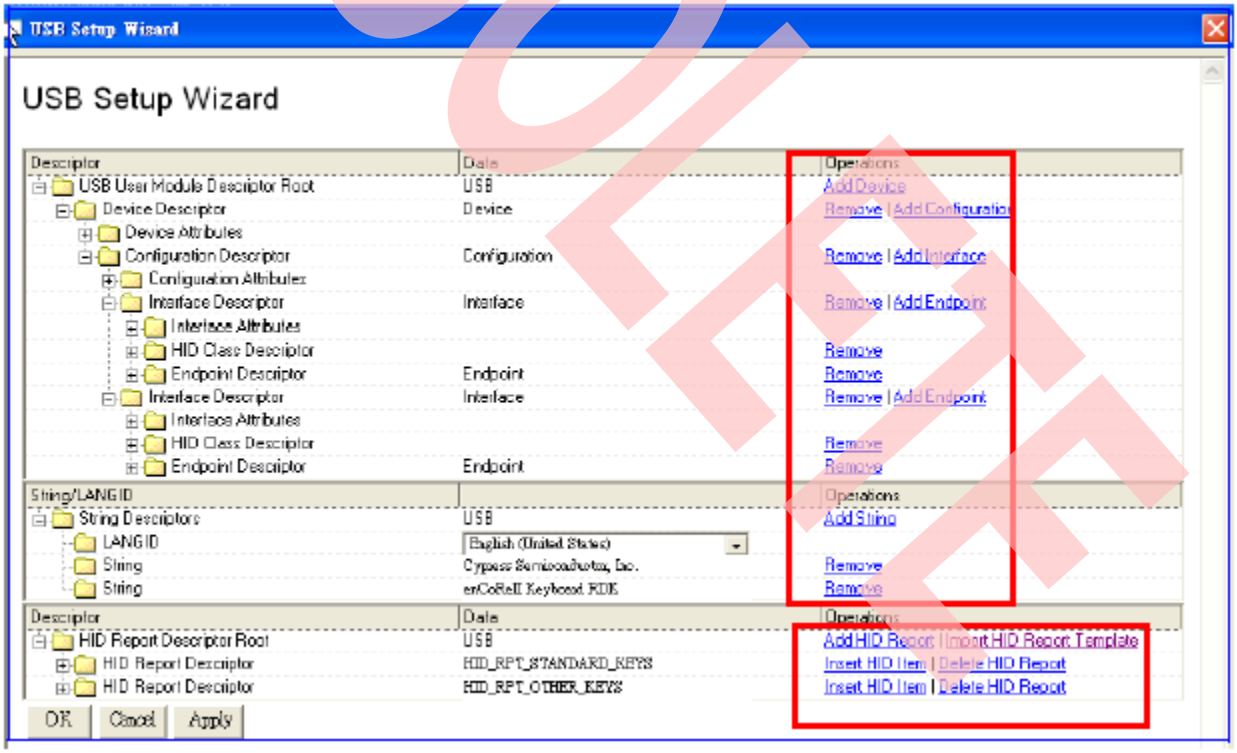
This section presents information on the USB Wizard and its uses.

USB Wizard

PSOC Designer includes a wizard to help you configure the USB descriptors. It cannot handle all of the options allowed within the USB or HID specifications, but it should be suitable for most HID customers. Users who cannot fit their definitions within the capabilities of the wizard (or users who prefers to manage the process themselves) can create descriptors in a more traditional way by adding their own descriptor file to the project.

1. Click the right mouse button on USB module in the floor plan view of the Device Editor, and then select **USB Wizard**.
2. The USB Setup Wizard is displayed. Customers can edit this table according to their design's requirement. To help you design quickly, Cypress also provides two reference templates of the HID report. For customer who just want to design a keyboard or mouse, select **Import HID Report Template** or load the 'RDK FW' then modify the template. This is a good way to start a design.

Figure 15. USB Setup Wizard



Get/Set Report Commands

PSOC Designer and the USB User Module handle most of the USB functions. You need only to be aware of some key buffers. Three buffers are important for Get/Set Report:

- USB_Interface_0_Feature for Get/Set report feature
- USB_Interface_0_Input for Get/Set report input
- USB_Interface_0_Output for Get/Set report output

Note: For EP1 data, USB_LoadEP can be a buffer for the user to send data to the PC side.

Using Vendor Commands

Many customers need to build their own vendor commands to receive some special instructions from the host driver. Cypress provides a framework for adding vendor support. To enable this, use this procedure:

1. Set '1' in the custom code block in USB.inc to enable vendor functions (one or more can be set to '1' as required by the application).

   Code 7: Enabling Vendor Commands

```
USB_CB_h2d_vnd_dev: equ 1
USB_CB_h2d_vnd_ifc: equ 1
USB_CB_h2d_vnd_ep: equ 0
USB_CB_h2d_vnd_oth: equ 0
```

2. Add custom code to the routines below. These are Vendor Specific Request Dispatch Routine templates.

   Code 8: Handlers for Vendor Commands

```
IF USB_CR_h2d_vnd_dev
export USB_DT_h2d_vnd_dev_Dispatch
USB_DT_h2d_vnd_dev_Dispatch:
; add your code here
;
LJMP USB_InitNoDataStageControlTransfer
ENDIF

;-----------------------------------------------
IF USB_CR_h2d_vnd_ifc
export USB_DT_h2d_vnd_ifc_Dispatch
USB_DT_h2d_vnd_ifc_Dispatch:
; add your code here
;
; initialize USB_CurrentTD data for transfer data
; USB_DataSource:
; USB_TransferSize:
; USB_DataPtr:
LJMP USB_InitControlWrite
ENDIF

;-----------------------------------------------
IF USB_CR_h2d_vnd_ep
export USB_DT_h2d_vnd_ep_Dispatch
USB_DT_h2d_vnd_ep_Dispatch:
LJMP USB_NotSupported
ENDIF

;-----------------------------------------------
IF USB_CR_h2d_vnd_oth
export USB_DT_h2d_vnd_oth_Dispatch
USB_DT_h2d_vnd_oth_Dispatch:
LJMP USB_NotSupported
ENDIF
```

USB and PS/2 Macros

enCoRe II has many internal APIs that can help you build much of the firmware code automatically. These APIs are especially helpful for managing the protocol-related portions of the code. Both USB and PS/2 support has been provided, so you need only to be aware of the appropriate functions and use them at the proper time.

USB APIs

USB_start()

This function enables the USB user module to support USB functions. A call to it should generally be inserted in an appropriate portion of the application initialization routine.

USB_bGetConfiguration

This function can get the configuration status of device. Normally, this is used to check whether it is time to send the report to the host.

USB_LoadEP

This can be used to load data in preparation for sending an IN transfer.

USB_Force

This is a very useful function that can be used to manually control the state of the D–/D+ lines. Four options are provided.

Force State	D+ Status	D- Status
USB_FORCE_J	0	1
USB_FORCE_K	1	0
USB_FORCE_SE0	0	0
USB_FORCE_NONE	Control of D+/D– is released	

USB_bCheckActivity

This function can be used to check for USB activity. It is normally used for suspend mode detection. The developer can place it in a timer interrupt routine and make a determination of when the system is required to enter suspend mode.

USB_Suspend

This function puts the USB Transceiver into power-down mode, while maintaining the USB address assigned by the USB host. To restore the USB transceiver to normal operation, the USB_Resume function must be called.

USB_Resume

This function puts the USB Transceiver into normal operation following a call to USB_Suspend. It retains the USB address that had been assigned by the USB host.

PS/2 APIs

PS2D_Start

This function enables the user module to support PS/2.

PS2D_DoCommand

This function receives and processes a PS/2 Host command. It must be called at least once every 1-2 milliseconds.

PS2D_TransferInProgress

This function returns ‘1’ if a transfer to the PS/2 Host is in progress, otherwise returns ‘0’.

EMC Suggestions

Many customer designs will be required to pass certain standards for EMI, ESD and/or EFTB. Although it is not the purpose of this application note to provide exhaustive guidance on this subject, the low-cost, minimalist design approaches that are typical of low-speed USB products tend to make this a significant issue. Therefore, some brief design guidance may be beneficial for minimizing the chance of encountering problems

GND PAD Assignment

GND PAD assignment is very important on the PCB design. Proper GND PAD assignment can reduce ESD or EMI issues. Generally speaking, the more GND the better the design. But in some special cases, noise may be coupled to signal IO pins through GND.

If there is any analog signalling in the design, digital and analog GND must be separated into two distinct zones, and then connected on the signal-out interface of the PCB.

Avoiding Antenna Effects

When laying out the PCB, it is important to be conscious of trace lengths. Long traces have a tendency to produce antenna effects, causing undesired radiation of noise. This kind of phenomenon is common on keyboards and similar types of products where traces are used to produce some type of input-scanning matrix. Additional noise suppression may be required on these IOs if trace lengths cannot be reduced.

WDT Timer Usage

When doing EFTB testing, the electrostatic burst may impact the microcontroller firmware execution. All designs must make use of the watchdog timer. Watchdog resets are generally an effective means to manage these types of events.

Clock Control

If EMI issues are encountered, remember that enCoRe II has a great deal of control over its clocks. Make sure that clocks are run at the lowest frequency required by the design. Unused clocks must be disabled or selected to the lowest frequency source. Although the low-power 32-kHz oscillator is provided as a sleep timer, it can also make a useful clock source for system timing functions in some applications. This can be extremely helpful not only because it is a much lower frequency, but also because it is not derived from the single 24-MHz source that is typically used for all other clocks.

Summary

This Application note documents the architectural enhancements made in enCoRe II in comparison with enCoRe series MCU.

It also serves guide for customers to migrate from enCoRe to enCoRe II.

Document History

Document Title: enCoRe™ to enCoRe II Conversion - AN6062

Document Number: 001-15343

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	1778266	KUH	11/27/2007	Recatalogued application note.
*A	3206553	NXZ	03/26/2011	Added assembly code example to demonstrate how to migrate GPIO accessing from enCoRe to enCoRe II. Changed title to include application note number as per guidelines.
*B	3276299	WQWU	08/15/2011	Update figures and module description in accordance with PSoC Designer 5.1
*C	4492975	ANKC	09/05/2014	Updated in new template. Completing Sunset Review.
*D	5889003	HPPC	09/19/2017	Please Obsolete the document.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc
Memory	cypress.com/go/memory
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/go/support

enCoRe is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709