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.

AN47310

PSoC 1 Power Savings Using Sleep Mode

Author: Grey Reynolds, Ashutosh Srivastava
Associated Project: Yes
Associated Part Family: CY8C21x23, CY8C21x34, CY8C21x45, CY8C22x45, CY8C23x33, CY8C24x33, CY8C24x23A, CY8C24x94, CY8C27x43, CY8C28xxx, CY8C29x66
Software Version: PSoC Designer™ 5.4 SP1
Related Application Notes: For a complete list of the application notes, click here.
To get the latest version of this application note, or the associated project file, click here

AN47310 introduces PSoC® 1 Sleep mode operation and power-saving techniques. The topics include Sleep mode basics, wakeup sources, power-saving techniques, and special Sleep mode considerations. An example project that demonstrates sleep and wakeup operation is also provided.

Contents

1 Introduction

Sleep mode is a simple and efficient way to reduce the average current consumption of a PSoC device. It places the device in a low-power state whenever the CPU and other internally clocked functions are not needed. Sleep mode is most useful for battery-powered systems but it is applicable to any design.

Sleep mode decreases the overall current drawn without limiting the functionality. You can achieve significant power savings by paying attention to proper entry, use, and exit of Sleep mode. Implemented in conjunction with many other power-saving features and techniques described in this document, Sleep mode can be extremely effective in reducing the overall power consumption in a PSoC based design.

This application note describes the fundamentals of Sleep mode and provides information on power-saving methods and other sleep-related considerations. It is assumed that the reader is familiar with the PSoC 1 architecture and PSoC Designer™ operation.

Note: In this document, 'PSoC' refers only to PSoC 1 devices.

2 PSoC Resources

Cypress provides a wealth of data at www.cypress.com to help you to select the right PSoC device for your design, and quickly and effectively integrate the device into your design. In this document, PSoC refers to the PSoC 1 family of devices. To learn more about PSoC 1, see the application note AN75320 - *Getting Started with PSoC® 1*.

The following is an abbreviated list for PSoC 1 resources:

- **Overview:** PSoC Portfolio, PSoC Roadmap
- **Product Selectors:** PSoC 1, PSoC 3, PSoC 4, PSoC 5LP, or PSoC 6. In addition, PSoC Designer includes a device selection tool.
- **Datasheets:** Describe and provide electrical specifications for the PSoC 1 device family.
- **Application Notes and Code Examples:** Cover a broad range of topics, from basic to advanced level. Many of the application notes include code examples.
- **Technical Reference Manuals (TRM):** Provide detailed descriptions of the internal architecture of the PSoC 1 devices.
- **Development Kits:**
  - CY3215A-DK In-Circuit Emulation Lite Development Kit includes an in-circuit emulator (ICE). While the ICE-Cube is primarily used to debug PSoC 1 devices, it can also program PSoC 1 devices using ISSP.
  - CY3210-PSOCEVAL1 Kit enables you to evaluate and experiment Cypress' PSoC 1 programmable system-on-chip design methodology and architecture.
  - CY8CKIT-001 is a common development platform for all PSoC family devices.
- The MiniProg1 and MiniProg3 devices provide an interface for flash programming.

2.1 PSoC Designer

PSoC Designer is a free Windows-based integrated design environment (IDE). Develop your applications using a library of pre-characterized analog and digital peripherals in a drag-and-drop design environment. Then, customize your design leveraging the dynamically generated API libraries of code. Figure 1 shows PSoC Designer windows.

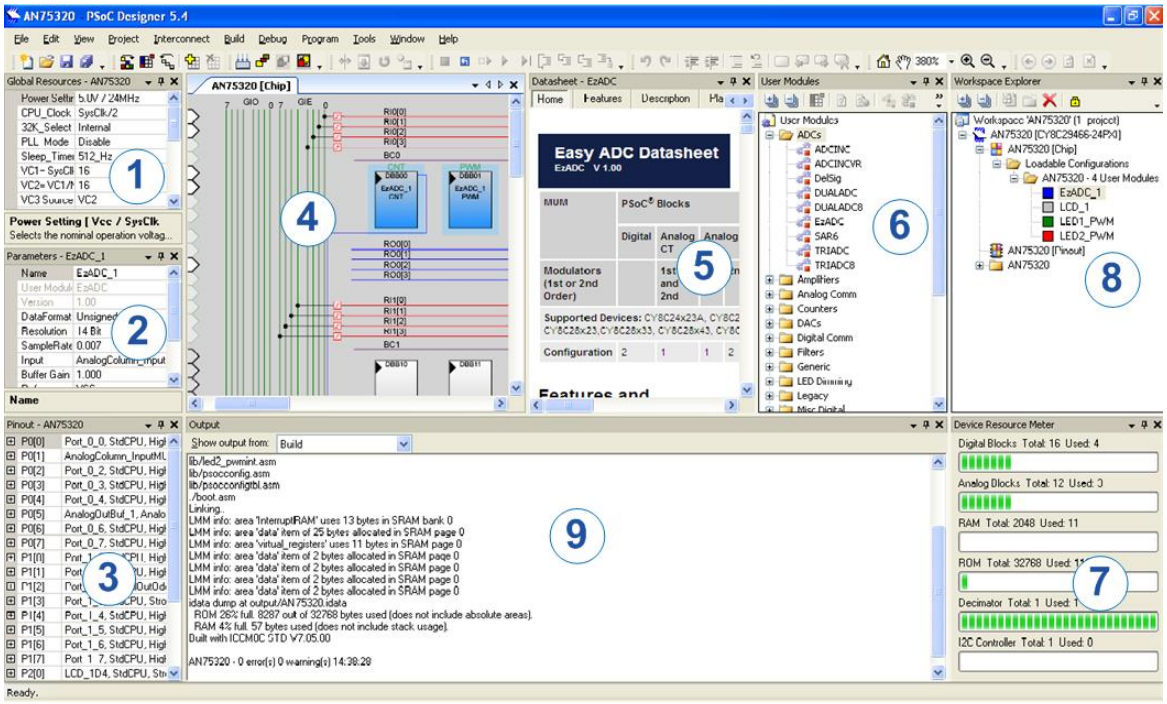
Note: This is not the default view.

1. **Global Resources –** All device hardware settings.
2. **Parameters –** The parameters of the currently selected User Modules.
3. **Pinout –** Information related to device pins.
4. **Chip-Level Editor –** A diagram of the resources available on the selected chip.
5. **Datasheet –** The datasheet for the currently selected UM
6. **User Modules –** All available User Modules for the selected device.
7. **Device Resource Meter –** Device resource usage for the current project configuration.

8. **Workspace –** A tree level diagram of files associated with the project.
9. **Output –** Output from project build and debug operations.

Note: For detailed information on PSoC Designer, go to **PSoC Designer** > **Help** > **Documentation** > **Designer Specific Documents** > **IDE User Guide**.

Figure 1. PSoC Designer Layout



2.2 Code Examples

The following webpage lists the PSoC Designer based code examples. These code examples can speed up your design process by starting you off with a complete design, instead of a blank page and show how PSoC Designer User Modules can be used for various applications.

http://www.cypress.com/documentation/code-examples/psoc-1-code-examples

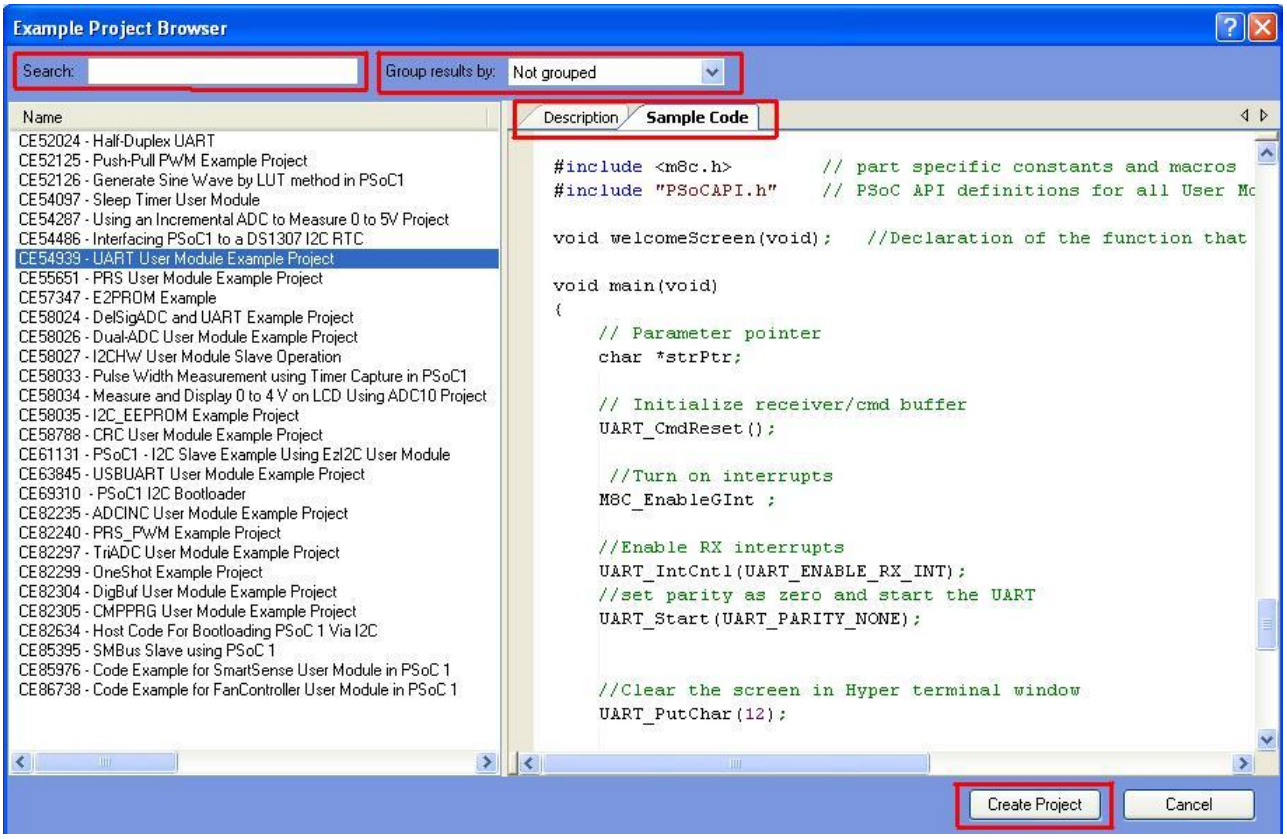
To access the code examples integrated with PSoC Designer, follow the path **Start Page > Design Catalog > Launch Example Browser** as shown in Figure 2.

Figure 2. Code Examples in PSoC Designer

In the Example Projects browser shown in Figure 3, you have the following options:

- Keyword search to filter the projects.
- Listing the projects based on Category.
- Review the datasheet for the selection (on the Description tab).
- Review the code example for the selection. You can copy and paste code from this window to your project, which can help speed up code development create a new project (and a new workspace if needed) based on the selection. This can speed up your design process by starting you off with a complete, basic design. You can then adapt that design to your application

Figure 3. Code Example Projects, with Sample Codes



2.3 Technical Support

If you have any questions, our technical support team is happy to assist you. You can create a support request on the Cypress Technical Support page.

You can also use the following support resources if you need quick assistance:

- Self-help
- Local Sales Office Locations

3 PSoC 1 Sleep Mode Basics

This section introduces the Sleep mode. This section also provides information on the behavior of the PSoC device during sleep and explains wakeup and reset sources and describes the firmware functions related to Sleep mode.

3.1 Entering Sleep Mode

The PSoC device enters sleep mode by writing a '1' to the SLEEP bit in the CPU_SCR0 register (bit 3). The M8C_Sleep macro is typically used to accomplish this, but a direct write to the register will have the same effect.

```
/* Enter Sleep Mode */
M8C_Sleep;
```

When the SLEEP bit is set, the PSoC sleep logic circuit asserts a device-wide power-down (PD) signal. When the PD signal is asserted, the PSoC device enters Sleep mode.

3.2 Behavior during Sleep Mode

The PD signal controls three major blocks: the internal main oscillator (IMO), flash memory, and the bandgap voltage reference. These blocks are powered down and remain in this state until a wakeup or reset event occurs. The CPU is also halted and placed in a low-power state.

Some core circuits remain active during Sleep mode. The internal low-speed oscillator (ILO), bandgap refresh circuit, supply voltage monitor, and switch mode pump (SMP) continue to operate to ensure that the PSoC device remains stable and can resume operation. The Sleep Timer and the bandgap refresh circuit use the ILO during Sleep mode. The ILO is also used to synchronize signals upon wakeup until the IMO and flash memory have had enough time to settle.

The switched capacitor (SC) blocks require a column clock. As the column clock is either VC1 or VC2, which is sourced by IMO, or by a digital block clocked by IMO, SC blocks are not functional during sleep. Only the continuous time (CT) blocks will be active during sleep. Additionally, GPIO pins remain in whatever state they were in when the PD signal was asserted. So, they can continue to source or sink current during Sleep mode operation.

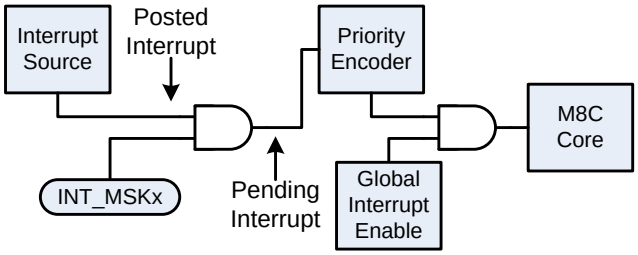
3.3 Wakeup from Sleep Mode

The only event that can wake the PSoC device from Sleep mode is a pending interrupt. Following a wakeup interrupt, the PD signal is deasserted. This causes the IMO, flash memory, and bandgap reference to be powered on. After a short delay (typically 15 µs) to allow for the clock settling, normal instruction execution continues. Any settings that were changed to reduce power during Sleep mode must be restored by the firmware when the CPU resumes code execution.

3.4 Posted Interrupts versus Pending Interrupts

An interrupt is posted when its interrupt condition occurs. A posted interrupt is not pending unless it is unmasked in the appropriate `INT_MSKx` register. So, to set up an interrupt to wake PSoC from sleep, the interrupt must be be unmasked using the `INT_MSKx` register. The CPU will not service any interrupts unless the Global Interrupt Enable bit is set in the CPU_F register. The block diagram in Figure 4 illustrates this logic.

Figure 4. Simplified Interrupt Controller Block Diagram



3.5 Interrupt Masks and Global Interrupt Enable

Global interrupts do not need to be enabled to wake the PSoC device from Sleep mode. The only requirement is to unmask the interrupt using the the INT_MSKx register. PSoC Designer provides an API for user modules to make the register writes more readable, as shown in the following example code.

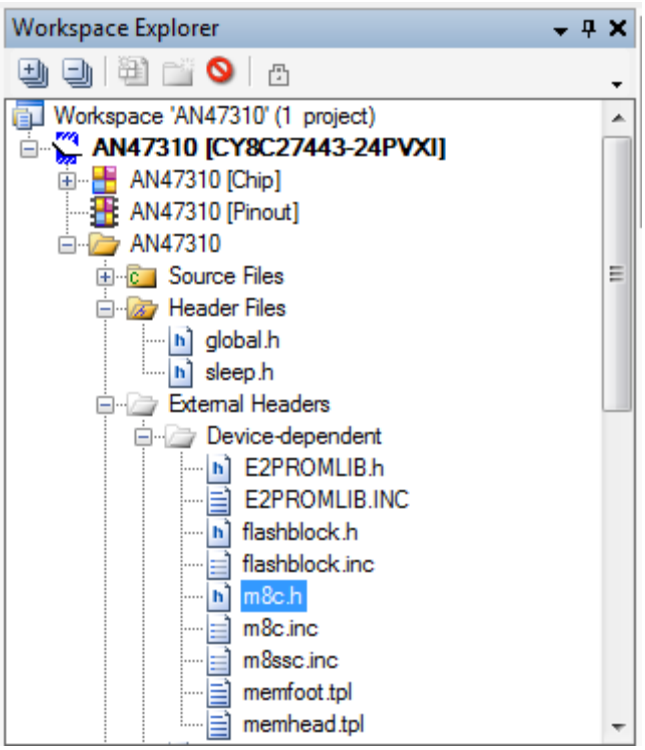
```
/* Set Mask for GPIO Interrupts */
M8C_EnableIntMask(INT_MSK0, INT_MSK0_GPIO);
```

If global interrupts are disabled, then the ISR that wakes the PSoC device is not executed, but the device still exits Sleep mode. The firmware must then either manually clear the pending interrupt or enable global interrupts to allow the ISR to be serviced. Interrupts are cleared in the `INT_CLRx` registers, as shown in the following example code:

```
/* Clear Posted Interrupt Flag */
M8C_ClearIntFlag (INT_CLR0, INT_MSK0_GPIO);
```

The macro M8C_ClearIntFlag is defined in file m8c.h. See the *Interrupt Controller* section of the PSoC 1 TRM for more detailed information on the function of the interrupt system. The #define values for the system registers and macros are in the *m8c.h* file of all PSoC Designer projects. The *m8c.h* file is in the **External Headers** > **Device-dependent** folder, as shown in Figure 5.

Figure 5. Location of m8c.h File



3.6 Interrupt Service Routines and Boot.tpl

The PSoC Designer Device Editor uses a template file called *boot.tpl*, located in the project's root directory, to create the configuration code that is executed at bootup. In PSoC devices, ISRs are mapped to interrupt vectors by adding a link to the ISR in the *boot.tpl* file.

The example project included with this document uses a flag set by an ISR to trigger Sleep mode. The ISR is serviced when a GPIO interrupt is pending. To properly define an ISR in PSoC devices, follow these three steps.

1. Using the “#pragma interrupt_handler directive, declare the interrupt handler function as an ISR. This ensures that the compiler terminates the function with a “reti” instruction to exit from the interrupt.

```
/* Declare the function as an ISR */
#pragma interrupt_handler GPIO_ISR
```

2. Write the function using the same name used in the previous step.

```
void GPIO_ISR(void);
{
/* GPIO ISR */
}
```

3. Open the *boot.tpl* file and add a jump to the interrupt handler function in the corresponding interrupt vector – in this example, the GPIO interrupt vector.

```
org    1Ch                  ;GPIO Interrupt Vector
ljmp _GPIO_ISR
reti
```

After the ISR has been defined and associated with a vector, the function will be executed when the pending interrupt is serviced. Note that any changes made to *boot.asm* are overwritten every time the project is generated. Therefore, changes should be made only to the *boot.tpl* file.

3.7 Wakeup Source

The interrupt sources available to wake up from sleep are the following:

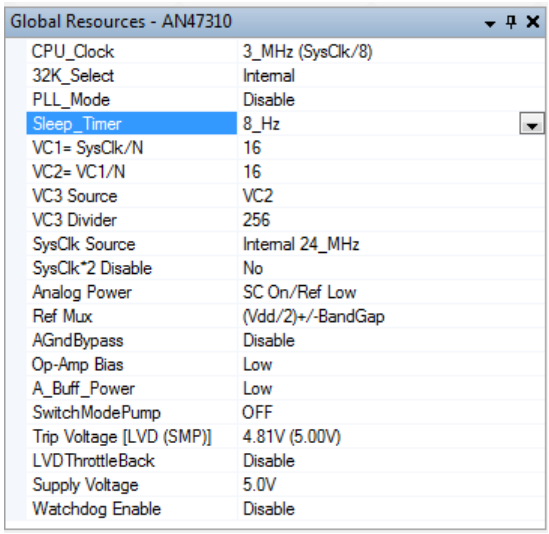
- Sleep Timer
- GPIO
- Low-voltage monitor
- Analog columns
- Digital blocks clocked by ILO

3.7.1 Sleep Timer Interrupt

The Sleep Timer is most often used to periodically wake the PSoC device from Sleep mode to do processing or check for activity. An example would be to wake it up at regular intervals to scan a sensor, process the data, and go back to sleep again.

The Sleep Timer consists of a 15-bit counter that is always enabled. It is clocked by the ILO or external crystal oscillator (ECO), so it can function in both Active and Sleep modes. The counter period can be set to roll over at 1 Hz, 8 Hz, 64 Hz, or 512 Hz in the Global **Resources** window of PSoC Designer, as shown in Figure 6.

Figure 6. Sleep Timer Period Selection



A write to the Sleep bits [4:3] of the OSC_CR0 register will also set the Sleep Timer interrupt period. Note that the ILO frequency may vary, so the Sleep Timer periods are approximate (see the ILO Variance section).

When the Sleep Timer rolls over, an interrupt is posted to the system. To allow an interrupt from the Sleep Timer to wake the PSoC device, it must be unmasked in the INT_MSK0 register like any other interrupt source.

```
/* Set Mask for Sleep Timer Interrupts */
M8C_EnableIntMask(INT_MSK0,INT_MSK0_SLEEP);
```

The SleepTimer User Module provides an API to control the interrupt period and mask or unmask the interrupt. It also uses the Sleep Timer to provide additional general-purpose timer functions, but those functions are available only when the PSoC device is in Active mode. They do not relate to Sleep mode operation.

The SleepTimer User Module datasheet contains detailed information about the features, parameters, and API associated with it. The datasheet is part of PSoC Designer and is also available on the Cypress website.

3.7.2 More than a 'Sleep Timer'

The Sleep Timer is a hardware counter that does not have to be used in conjunction with the PSoC sleep mechanism. When configured properly, the Sleep Timer generates a periodic interrupt and wakes the CPU from sleep. However, any interrupt source can wake the CPU, even from analog or digital blocks. So, the Sleep Timer may be used with PSoC sleep, but is also very useful as a standalone long-duration counter.

The Sleep Timer is tied to the watchdog timer (WDT) reset functionality in PSoC. When enabled, the WDT is designed to generate a hardware reset in PSoC after three cycles of the Sleep Timer. This is not intuitively obvious, so if you are using the Sleep Timer, and have enabled the watchdog timer, be aware of its implications on the watchdog reset.

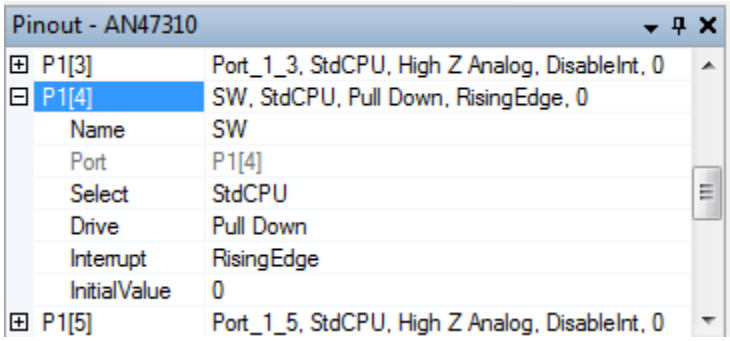
3.7.3 GPIO Interrupt

Each GPIO pin can be individually configured to trigger an interrupt on a rising edge, falling edge, or any change condition. Interrupts are enabled through a write to the PRTxIC1 and PRTxIC0 registers. Each register has one bit that applies to each GPIO pin in the bank. For example, to enable a rising-edge interrupt on P1[4] (a value of 01b in the registers), code similar to the following example can be used.

```
/* Enable P1[4] rising-edge interrupt */
PRT1IC1 |= 0x10; /* Force set bit 4 */
PRT1IC0 &= ~0x10; /* Force clear bit 4 */
```

Interrupts on GPIO pins are typically enabled in PSoC Designer so that the configuration is set at boot. The **Pinout** interface contains a field for every pin in the selected device. If the pin view is expanded, as shown in Figure 7, the drop-down menu that appears can be used to set the interrupt type.

Figure 7. Pinout View with GPIO Interrupt



Note that once a GPIO is configured, the GPIO block interrupt must still be unmasked in the INT_MSK0 register before it can wake the PSoC device.

3.7.4 Low-Voltage Monitor Interrupt

The bandgap and LVD blocks are periodically re-enabled during sleep to monitor for low-voltage conditions. This is accomplished by turning on the bandgap and refreshing the reference voltage. If a low supply voltage is detected during the refresh, an interrupt is posted to the system. If the interrupt is unmasked, the PSoC device wakes up. The voltage refresh rate is set with a write to the PSSDC bits [7:6] of the ECO_TR register, but modifying this value is not recommended. As with other interrupt sources, the low-voltage monitor interrupt must be unmasked in the INT_MSK0 register to wake the PSoC device.

3.7.5 Analog Column Interrupt

The analog columns each have associated interrupts that can be unmasked and used to wake the PSoC device from Sleep mode. Any PSoC function that interfaces to the analog columns can be used as an interrupt source, but only those that remain operational in Sleep mode can be used as wakeup sources.

The most common example is a comparator used to wake the PSoC device when an external signal crosses the trip voltage. The comparator control latch must be set to “transparent” mode. This can be done by clearing bit 6 (CLatch) of ACBxxCR2.

```
/* Set the comparator to transparent mode so that it can be used to wake up PSoC*/
ACBxxCR2 |= 0x10;
```

Register INT_MSK0 holds the four analog bits [4:1] that are used to mask the interrupts, one bit per column. The following example code shows how to unmask the interrupts for analog column 3.

```
/* Unmask analog column 3 interrupt */
M8C_EnableIntMask(INT_MSK0, INT_MSK0_ACOLUMN_3);
```

Some PSoC devices have less than four analog columns. In these cases, fewer bits are used in the register. See the individual PSoC datasheets for specific details.

Switched capacitor analog blocks cannot be used to wake the device from Sleep mode. These blocks do not operate under Sleep mode as they require the column clock to operate. The column clock is derived from the IMO, which is disabled during sleep.

3.7.6 Digital Blocks

Digital blocks can also be used as interrupt source to wake the PSoC from sleep. For example, a counter’s terminal count event can be used to wake the PSoC. However, there are two important points to remember. If the clock input of the digital block is derived from IMO, the digital block will not operate when the PSoC is put to sleep and, hence, will not be able to generate the wakeup interrupt. The ClockSync parameter of the digital block should be set to “Unsynchronized”.

3.8 Reset versus Wakeup

A reset will also wake the PSoC device from sleep, but the difference between a reset and an interrupt wakeup is what happens after sleep is exited. A reset takes the PSoC device out of Sleep mode and holds it in a reset state as long as the reset is asserted. Once the reset deasserts, the PSoC device begins executing code starting at the beginning of *Boot.asm*. There are three types of resets within PSoC: external reset (XRES), watchdog reset, and power-on reset (POR).

4 Additional Power-Saving Techniques

This section describes techniques to further reduce the power consumption in Sleep mode by disabling PSoC features that may remain active when the SLEEP bit is set.

4.1 Disable Analog Block References

PSoC analog blocks have individual power-down settings that are controlled by the firmware. The analog block references can be disabled through a write to the PWR bits [2:0] of the ARF_CR register, similar to the following code:

```
/* Turn off analog ref */
ARF_CR &= ~ARF_CR_REFPWR;
```

Note that this register is not available for the CY8C21x34, CY8C21x34B, and CY8C21x23 PSoC devices. Also, if the analog column was configured to generate interrupt to wake the PSoC, analog reference should not be turned off.

4.2 Disable Analog Buffers

The analog output buffers, which connect the analog column outputs to the output pins, can be disabled to save power. This is accomplished through a write to the ABUFxEN bits [5:2] in the ABF_CR0 register. The following example shows how to disable all four analog output buffers.

```
/* Turn off analog buffers */
ABF_CR0 &= ~(ABF_CR0_ABUF1EN | ABF_CR0_ABUF2EN | ABF_CR0_ABUF0EN | ABF_CR0_ABUF3EN);
```

Some PSoC devices have less than four analog output buffers or do not have analog output buffers at all. In these cases, fewer bits are used in the register. See the individual PSoC datasheets for information on the number of analog buffers.

4.3 Disable CT/SC Blocks

The continuous time (CT) blocks are powered down individually with a write to each `ACBxxCRy` or `ACExxCRy` register corresponding to the block's column. The switched capacitor (SC) blocks are similarly controlled by the `ASCxxCRy` or `ASDxxCRy` registers. The following example shows how to disable the CT and SC blocks for column 0:

```
#define ACB_CT_SC_ENABLE 0x03

ACB00CR2 &= ~ACB_CT_SC_ENABLE; /* Disable CT Block of row 0 and column 0 */
ASC10CR3 &= ~ACB_CT_SC_ENABLE; /* Disable typeC SC block of row 1 and column 0 */
ASD20CR3 &= ~ACB_CT_SC_ENABLE; /* Disable typeC SC block of row 2 and column 0 */
```

The CT blocks can remain in operation because they do not require a clock source. However, the SC blocks do not operate because there is no clock source for the switches.

4.4 Set GPIO Drive Modes to Analog HI-Z

The state of the GPIO drive mode can affect the power consumption in Sleep mode. GPIO pins retain their drive mode settings during sleep, so if a pin sources or sinks current during Active mode, then it will continue to do so in Sleep mode. Unless the signal must stay at that state during sleep, additional power savings can be realized by changing the drive mode of the GPIO to Analog HI-Z.

The drive modes can be set manually in firmware before Sleep mode is entered. The three registers that control the GPIO drive modes are PRTxDM0, PRTxDM1, and PRTxDM2. One bit per register is assigned to each pin, and the combination of the three bits determines the drive mode. So, changing the drive mode of a single pin or an entire port typically requires three register writes. Only one of the three PRTxDM registers can be written at a time, so it is important to consider the order in which it is done. Intermediate drive modes are enabled as the registers are written one by one. Controlling the order of the writes determines which of those states are enabled. It is recommended that resistive pull-up or pull-down modes be used as intermediate states to prevent glitches. For example, to transition P2[0] from a Strong drive mode to Analog HI-Z, the following code could be used:.

```
#define PORT_2_0    0x01

/* Original state: DM = 001 = Strong */
PRT2DM0 &= ~PORT_2_0; /* Pull-down */
PRT2DM1 |= PORT_2_0; /* HI-Z */
PRT2DM2 |= PORT_2_0; /* HI-Z Analog */
```

Current consumption can be reduced in many designs if consideration is given to the state of the GPIO pins during Sleep mode. For more information about GPIO functionality, see the PSoC 1 TRM and AN2094 – PSoC 1 – Getting Started with GPIO.

5 Other Sleep Mode Considerations

This section provides information on special considerations for using Sleep mode. These topics may not be applicable to all designs.

5.1 ILO Variance

By default, the Sleep Timer circuit uses the ILO as its clock source. The ILO frequency can deviate from –50 percent to +100 percent, however, and is not considered a precision wakeup source. If a precise wakeup time is needed, some PSoC devices have the capability to clock the Sleep Timer from an ECO. If the added cost of an external crystal is prohibitive, an alternative is an external R/C oscillator circuit implemented with two GPIO pins, a resistor, and a capacitor. For details on this method, see AN47215 – PSoC RC Oscillator to Accurately Time Sleep Cycles.

5.2 Interrupts Before Sleep Mode Entry

If an unmasked interrupt is pending when a write is made to the SLEEP bit, the SLEEP bit is not set and the system will not enter Sleep mode. To avoid this situation, the firmware should disable the global interrupts and clear any pending interrupts before preparing to enter Sleep mode.

```
M8C_DisableGInt; /*Disable interrupts */
INT_CLR0 = 0x00; /*Clear pending interrupts */
```

Global interrupts can be enabled again immediately before `M8C_Sleep` is called because the timing of the global interrupt logic makes it impossible for an interrupt to occur during the next instruction, which in this case is setting the SLEEP bit.

```
M8C_DisableGInt; /* Disable interrupts */
SleepPrep();/* Configure PSoC for sleep */
M8C_EnableGInt; /* Enable interrupts */
M8C_Sleep; /* Sleep the PSoC */
WakeupRestore();/*Reconfigure for active */
```

Note that even if global interrupts are disabled, an unmasked interrupt will still wake the PSoC device from sleep. Therefore, it is not required to enable global interrupts before Sleep mode is entered, especially if you do not want an ISR to be executed until after the PSoC device is reconfigured after a wakeup.

5.3 Prefetched Code

The instruction immediately following the sleep instruction is prefetched before the CPU is halted. If global interrupts are enabled when the PSoC device enters Sleep mode, then there is a chance that a jump into an ISR will be the action that follows the execution of the prefetched instruction. One way to ensure that the prefetched code will have no effect on the ISR is to place a “nop” after the sleep macro.

```
M8C_Sleep; /* Enter sleep mode */
asm("nop"); /* Use nop as pre-fetched code */
```

In this case, if there are no pending interrupts, the nop is executed at wakeup and nothing happens. If there is a pending interrupt, then the prefetched nop will not have any effect on the ISR after wakeup.

5.4 PLL Operation

If PLL_Mode is enabled in the **Global Resources** section of the PSoC Designer project, the PLL should be stopped and the CPU frequency should be reduced to 3 MHz or less before going to sleep. This is because the PLL may briefly overshoot as it attempts to relock after the PSoC device wakes up. The relock can take up to 10 ms.

The CPUSpeed bits [2:0] of the `OSC_CR0` register set the divider of SYSCLK, which is the CPU’s clock source, and the PLLMode bit [6] enables or disables the PLL. The frequency of SYSCLK is typically 24 MHz for PSoC devices, but it can be changed to 6 MHz on some devices. The following code is an example of using the `OSC_CR0` register to disable PLL mode and reduce the CPU clock speed.

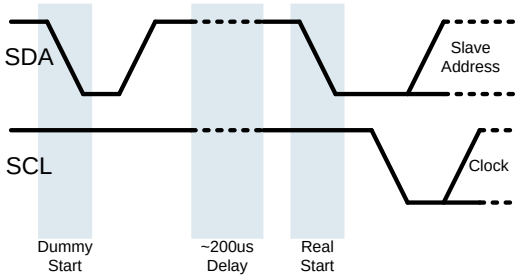
```
OSC_CR0 &= 0xb8; /* CPU = IMO/8 */
```

It is recommended that the firmware in projects that use PLL_Mode and sleep be able to execute at 3 MHz to ensure that there are no issues with functionality soon after wakeup. If the firmware cannot execute at 3 MHz, ensure that the PLL is locked before normal operation is resumed after wakeup.

5.5 I²C Slave Addressing and Sleep Mode

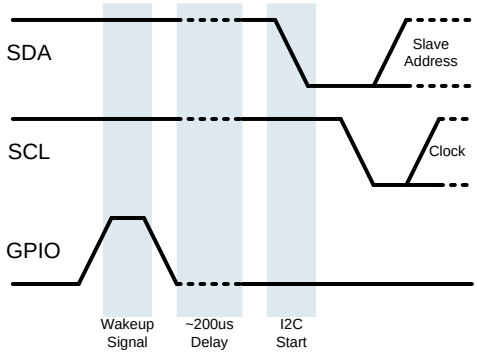
While in Sleep mode, a PSoC device cannot immediately process an I²C address and respond because the IMO and CPU are powered OFF. A workaround is to set up falling-edge interrupts on either the clock or the data lines of the I²C bus. The master must then send a dummy START condition to wake up the PSoC device or continue to transmit the address until an ACK is received. The master may need to delay up to 200 µs between the dummy START and the real one to allow the PSoC device to wake up and resume code execution, as shown in Figure 8.

Figure 8. Dummy START to Wake PSoC Device



A side effect of this workaround is that the PSoC device will wake up on any I²C falling-edge traffic, which means that it will result in more total active time and higher average current. To avoid the increased active time, a separate GPIO can be dedicated to wake up the PSoC device when the I²C master is ready to address it, as shown in Figure 9. The I²C master would need to toggle the GPIO pin and wait for the appropriate delay time before the initial transmission is sent.

Figure 9. Separate GPIO for Wakeup Signal



This technique does not require nonstandard I²C master behavior, but it does require the use of an additional GPIO pin.

Note: When using an external clock, follow the steps in the Implementation Sequence section to handle the sleep and wakeup process.

Immediately before the device goes to sleep, disable the external clock using the `EXTCLKEN` bit in the `OSC_CR2` register. Immediately after the device wakes from sleep, enable the external clock using the `EXTCLKEN` bit in the `OSC_CR2` register.

6 Implementation Sequence

To implement Sleep mode, follow this example sequence.

1. Unmask the interrupts that must wake the PSoC device; for example, unmask the GPIO or sleep interrupt.
2. Enable the global interrupts.
3. Enter the while loop.
4. Disable all peripherals such as analog reference, analog output buffer, CT blocks, and SC blocks to reduce power consumption.
5. Change the drive state of the GPIOs that are not used to HI-Z Analog.
6. Execute the `M8C_Sleep` macro. This puts PSoC to sleep. Any interrupt will wake PSoC.
7. Enable all resources that were disabled before sleep.
8. Configure the GPIOs to the original state.
9. If multiple sources of interrupts can wake PSoC, check which interrupt caused wakeup and process the event accordingly.
10. Go to step 4.

7 Example Project

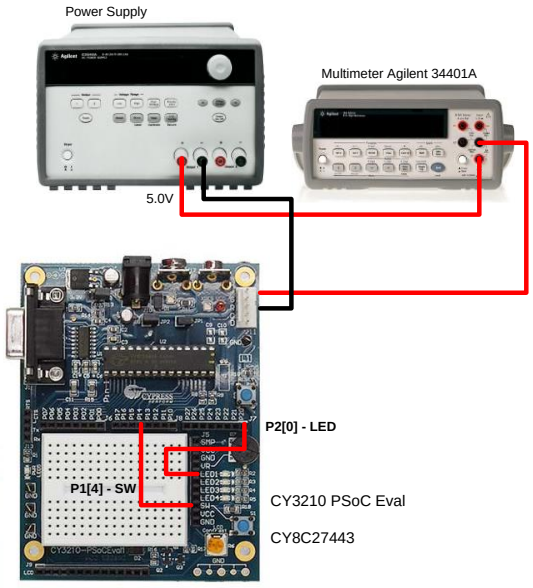
The example project included with this application note demonstrates Sleep mode operation with a CY8C27443 device. The project is applicable to any PSoC 1 device, but minor changes may be needed to accommodate the specific architecture of other devices. This section provides a general description of the project and the expected power consumption values.

7.1 Description

This project demonstrates how to put a PSoC 1 device to sleep and wake it again using the Sleep Timer and GPIO interrupts. It also demonstrates the use of different registers to disable portions of the PSoC device that may continue to draw power during sleep.

The example code was developed and tested using an Evaluation Kit CY3210-PSoCEval1 shown in Figure 10. Other hardware can easily be used to duplicate this setup if the PSoC device's power supply can be isolated for measurement.

Figure 10. Test Setup using CY3210-PSoCEval1 Kit



When the project is tested with the CY3210-PSoCEval kit, the measured current includes the device current and the current consumed by the other components on the kit. Follow these steps to measure the typical current consumption for only the PSoC device:

1. Place the PSoC 1 device on the kit and program the device.
2. Remove the device from the kit.
3. Turn ON the kit power supply and measure the current. This will be offset current I_{Offset}.
4. Turn OFF the kit power supply and place the PSoC 1 device on the Kit.
5. Turn ON the kit power supply and measure the current. This will be total current I_{Total}.
6. Subtract the I_{Offset} from I_{Total}. This will be the typical current consumption for the PSoC device.

To help demonstrate the power-saving techniques mentioned previously in this document, the CPU clock is set to 3 MHz and the supply voltage is set to 5 V. The following three pins are used in the project:

- P0[3]: Output from an inverting amplifier. This is used to demonstrate a way to disable the analog buffer and subsequent power saving.
- P1[4]: Digital input used to trigger a GPIO interrupt on a rising edge.
- P2[0]: Digital output for driving an LED. Used to demonstrate changing drive modes to Analog HI-Z.

With the unmodified example firmware, the PSoC device will enter sleep when a positive edge on P1[4] causes a GPIO ISR to be executed. On the CY3210-PSoCEval1 Kit, this pin is connected to a pushbutton switch for convenience. Comment the following set of `#defines` to modify the behavior of the PSoC device:

```
/* Defined Interrupt Sources */
#define USE_GPIO_INT      1             /* Use GPIO interrupt to wake PSoC */
#define USE_SLEEP_INT     1             /* Use sleep timer interrupt to wake PSoC */

/* Defined Sleep Mode Configuration Modifiers */
#define ANALOG_REF      1             /* Disable analog references in sleep */
#define ANALOG_OUTPUT_BUFFERS 1       /* Disable analog output buffers in sleep */
#define DRIVE_MODES    1              /* Set drive modes to Hi-Z in sleep */
#define CT_SC_BLOCKS     1            /* Disable CT and SC blocks in sleep */
```

Including or excluding these #defines when compiling the code will change the behavior of the example project. This will result in power consumption that is somewhere between normal device operation and the lowest possible power state for the particular PSoC device being used. Any combination of the previous definitions can be used with this example project.

Definitions used to enable the interrupt sources that trigger Sleep mode and wakeup are as follows:

- USE_GPIO_INT: Including this `#define` will enable the use of the GPIO interrupt to trigger the ISR to put the PSoC device into Sleep mode and wake it again. If it is not included, the PSoC device will not enter Sleep mode.
- USE_SLEEP_INT: Including this `#define` enables the Sleep Timer interrupt to periodically wake up the PSoC device from Sleep mode. This is done to reduce the average power consumption. If it is not included, then the Sleep Timer interrupt will not wake the PSoC device. The default Sleep Timer interval for this example project is 8 Hz.

Definitions used to disable features that can remain active in Sleep mode are as follows:

- ANALOG_REF: Including this `#define` will disable the analog references as part of the preparation for entering Sleep mode and enable them again as part of the wakeup recovery. If it is not included, the analog references will remain powered and active in Sleep mode.
- ANALOG_OUTPUT_BUFFERS: Including this `#define` will disable the analog output buffers as part of the preparation for entering Sleep mode and enable them again as part of the wakeup recovery. If it is not included, the analog output buffers will remain powered and active in Sleep mode.
- DRIVE_MODES: Including this `#define` will change the drive mode of the LED pin P2[0] to Analog HI-Z as part of the preparation for entering Sleep mode and change it back to Strong Drive as part of the wakeup recovery. If it is not included, the LED will continue to be driven by the PSoC device in Sleep mode.

Note: The current in this configuration depends on the external parameters such as external resistor, LED and applied VDD.

- CT_SC_BLOCKS: Including this #define will disable the CT and SC blocks as part of the preparation for entering Sleep mode and enable them again as part of the wakeup recovery. If it is not included, the CT and SC blocks will remain powered in Sleep mode. The blocks may not show activity, however, because their clock sources may be disabled.

After the hardware and firmware are configured as desired, the PSoC device can be programmed and its behavior can be observed with an ammeter.

Note: The MiniProg should be disconnected from the PSoC device after programming is complete to ensure that there is no current leakage through the program/debug pins.

7.2 Expected Power Consumption

The associated example project provides two methods to put the device into Sleep mode.

1. Using GPIO interrupt
2. Using Sleep Timer interrupt

Case 1: Table 1 gives the typical current consumption for the PSoC device when executing the example code on a CY8C27443 device enabling GPIO interrupt (default code):

```
#define USE_GPIO_INT      1        /* Use GPIO interrupt to wake up PSoC */
//#define USE_SLEEP_INT 1          /* Use sleep timer interrupt to wake up PSoC */
```

Table 2 gives the typical current consumption for the CY8C27443 device resources used in the example project under the above condition.

Table 1. Current Consumption by CY8C27443 Device using Example Project

Operating Mode	Current (mA)
Active Mode Current	20.0
Sleep Mode Current – With no Sleep Mode Modifiers	13.8
Sleep Mode Current – With only Analog References Off	3.4
Sleep Mode Current – With only Analog Buffers Off	13.4
Sleep Mode Current – With only CT/SC Blocks Off	9.8
Sleep Mode Current – With only Drive Modes Changed	10.8
Sleep Mode Current – Using all Sleep Mode Modifiers	0.004

Note: The values shown in Table 1 reflect typical numbers for the CY8C27x43 family of devices.

Table 2. Current Consumption by CY8C27443 Device Resources using Example Project

Device Resources	Current Consumed (mA)
Analog References	10.4
Analog Buffers	0.4
CT/SC Blocks	4.0
Drive Mode	3.0

Case 2: Table 3 shows the current consumed by the CY8C27443 device enabling sleep timer interrupt:

```
//#define USE_GPIO_INT    1        /* Use GPIO interrupt to wake up PSoC */
#define USE_SLEEP_INT     1        /* Use sleep timer interrupt to wake up PSoC */
```

In the Example Project the sleep timer interrupt frequency set to 8Hz and with a constant delay in the main loop for the device active time. The device active time is set to 12.8ms.

Table 3. Current Consumption by CY8C27443 Device using Sleep Timer Interrupt

Operating Condition	Current (mA)
Device Current – With no Sleep Mode Modifiers	14.4
Device Current – Using all Sleep Mode Modifiers	2.2

8 Summary

This application note described the PSoC 1 Sleep mode operation and power-saving techniques. It provided basic information on Sleep mode and power-saving methods, including the behavior of the PSoC device during sleep, wakeup, and reset, and discussed wakeup sources. The document also explained special considerations that may be relevant when using Sleep mode.

9 Related Application Notes

- AN2010 – PSoC® 1 Best Practices and Recommendations
- AN47215 – PSoC® RC Oscillator to Accurately Time Sleep Cycles
- AN2094 – PSoC® 1 – Getting Started with GPIO

Appendix A. Appendix A – Register Reference Table

Table 4 is a quick reference for some registers mentioned in this document that relate to Sleep mode. The PSoC 1 TRM and the device datasheets contain detailed descriptions of these registers, including register mapping tables and bit definitions.

Table 4. Register Reference Table

Name	Address	Relation to Sleep Mode
ABF_CR0	0x162h	The Analog Output Buffer Control Register 0 is used to enable or disable the analog output buffers.
ARF_CR	0x0F3h	The Analog Reference Control Register is used to enable or disable the analog reference power.
INT_MSK0	0x0E0h	The Interrupt Mask Register 0 is used to mask or unmask pending interrupts.
INT_CLR0	0x0DAh	The Interrupt Clear Register is used to clear pending interrupts.
RES_WDT	0x0E3h	The Reset Watchdog Timer Register is used to clear the sleep timer by writing a 0x38 to it. Note that any write to this register clears the WDT.
CPU_SCR0	0x0FFh	The System Status and Control Register 0 contain the SLEEP bit.
OSC_CR0	0x1E0h	The Oscillator Control Register 0 is used to configure the Sleep Timer interval.
PRTxDM2 PRTxDM1 PRTxDM0	See Datasheet	The Port Drive Mode Bit Registers are used to specify the drive mode for GPIO pins.
PRTxIC1 PRTxIC0	See Datasheet	The Port Interrupt Control Registers are used to specify the interrupt mode for GPIO pins.
ECO_TR	0x1EBh	The External Crystal Oscillator Trim Register is used to set the voltage refresh rate. The factory default values are strongly recommended.
ACBxxCRy	See Datasheet	These registers are used to enable or disable a type B CT PSoC block.
ACExxCRy	See Datasheet	These registers are used to enable or disable a type E CT PSoC block.
ASCxxCRy	See Datasheet	These registers are used to enable or disable a type C SC PSoC block.
ASDxxCRy	See Datasheet	These registers are used to enable or disable a type D SC PSoC block.

Appendix B. Appendix B – Example Project Description

This project demonstrates how to put PSoC to sleep and wake it again using the Sleep Timer and GPIO interrupts. It also demonstrates the use of different registers to disable portions of the PSoC device that may continue to draw power during sleep. Monitoring the power consumption of the PSoC device during normal operation and Sleep mode shows the difference in current.

The project includes three source files and two header files.

B.1 Source Files

- *main.c*: This file includes the main function. This function is the main loop for the example project. It sets the analog system to full power, unmasks any defined interrupt sources, starts the Sleep Timer (if defined), and turns ON the LED. It then enters an infinite while loop to either wait for a GPIO interrupt (if defined) or call Sleep mode every time through the while loop.
- *sleep.c*: This file includes three APIs.
  - SleepPrep: This function configures the PSoC device for sleep according to the Sleep mode configuration modifiers, such as ANALOG_REF, ANALOG_OUTPUT_BUFFERS, DRIVE_MODES, and CT_SC_BLOCKS, defined in *sleep.h*.
  - WakeUpRestore: This function restores the PSoC device to normal operation after sleep according to the Sleep mode configuration modifiers defined in *sleep.h*.
- *gpio_isr.c*: This file includes the GPIO_ISR function. This function is the ISR for the GPIO interrupt. It sets a flag that is checked at every pass through the main loop.

B.2 Header Files

- *sleep.h*: This header file includes the interrupt sources and sleep mode configuration modifier definitions. It also includes the prototypes for the functions that are used to configure the PSoC device for Sleep mode and wake it from Sleep mode.
- *global.h*: This file contains the global variables and functions declaration.

Document History

Document Title: AN47310 - PSoC 1 Power Savings Using Sleep Mode

Document Number: 001-47310

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	2551464	CFW	08/12/2008	New application note.
*A	3435093	GIR	11/10/2011	Complete rewrite and reformat of content for new template; new example project.
*B	3674258	GULA	07/12/2012	Code modularized. Replaced magic numbers with standard macros from m8c.h. Replaced the code in the document with the files and API description.
*C	3888965	GULA	29/01/2013	Updated the project to PSoC Designer 5.3 Added Figure 5 to show the location of m8c.h
*D	4652356	DIMA	02/05/2015	Updated the project to PSoC Designer 5.4 CP1 Added a note in page 7 on sleep and wakeup handling when using external clock Completing sunset review Updated template
*E	4792848	ASRI	09/02/2015	Changed section name “ Summary” to “Implementation Sequence” Added section “Summary” as application note summary Added wakeup source “Digital blocks clocked by ILO” in section “Wakeup Source” Removed the content related to Mini-Eval Programming Board Added Figure 11. Test Setup using CY3210-PSoCEval1 Programming Board Added steps to calculate the typical current consumption for the PSoC device. Added Table 2. - Current Consumption by Device Resources Used in the Example Project Added Table 3 - Current Consumption by Device using sleep timer interrupt Corrected the web links for TRMs and datasheets. Updated the project to PSoC Designer 5.4 SP1. Updated template.
*F	5703866	AESATMP9	04/20/2017	Updated logo and copyright.
*G	6350717	DIMA	10/15/2018	Sunset review. Updated template. Minor text edits.

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

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

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

Cypress Developer Community

Community | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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