AN77900

PSoC® 3 and PSoC 5 Low-power Modes and Power Reduction Techniques

Author: Greg Reynolds

Associated Projects: Yes

Associated Part Family: CY8C3xxxx, CY8C5xxxx

Software Version: PSoC Creator 2.0

For a complete list of related application notes, click here

If you have a question about this application note, contact the author at gir@cypress.com

AN77900 is an introduction to the PSoC 3 and PSoC 5 low-power modes and features. Major topics include PSoC power modes, power management API and registers, additional power-saving techniques, and other low-power mode considerations.

Contents

Introduction 2
- Power Mode Transitions 2
- Wakeup Sources 3

Active Mode Power Reduction 4
- Power Down Unused Components 4
- Use PSoC to Gate Current Paths 4
- Dynamically Change Clock Speeds 5
- Use DMA to Move Data 6
- Use Power Mode Configuration Registers 6

Other Power Mode Considerations 7
- Faster Clocks Can Mean Lower Power 7
- SegLCD in PSoC 3 Sleep Mode 8
- Boost in PSoC 3 Sleep Mode 8
- PSoC 3 Fast IMO Startup 9
- Watchdog in Sleep 9
- GPIOs in Low Power 9
- Digital Blocks in Low Power 9
- Vref Sources in Low Power 10
- Sleep and Hibernate Regulators 11
- Power Consumption While Programming 11
- Is Your Debug Interface Running? 11

Summary 11

Related Application Notes 12

Appendix A – Low-Power Example Projects 13
- PSoC 3 Low-Power Demo 13
- PSoC 5 Low-Power Demo 13
- Voltage Alarm – No Optimization 14
- Voltage Alarm – With Optimizations 14

Appendix B – Power Measurements on DVKs 15
- CY8CKIT-030 and CY8CKIT-050 Modifications 15
- CY8CKIT-001 Modifications 15

Appendix C – Power Management API and Registers 17
- CyPmSaveClocks() 17
- CyPmRestoreClocks() 17
- CyPmAltAct() 17
- CyPmSleep() 17
- CyPmHibernate() 17
- Component Low-power API 17
- Direct Register Writes 17
- Power Management API Flowcharts 19
- Power Management API Register Reference 20

Document History 22

Worldwide Sales and Design Support 23

Common Acronyms

CTW – central timewheel

FTW – fast timewheel

IMO – internal main oscillator

ILO – internal low-speed oscillator

LCD – liquid crystal display

OPPS – one pulse per second timer

PICU – pin interrupt control unit

RTC – real-time clock

SPC – system performance controller

TRM – technical reference manual

USB – universal serial bus

WDT – watchdog timer

XRES – external reset pin

Introduction

The proliferation of portable electronics and the constant focus on “green” technology have increased the importance of reducing a design’s power consumption. The PSoC 3 and PSoC 5 low-power modes allow you to lower overall current draw without limiting functionality, especially when implemented with other power-saving features and techniques.

This application note describes the fundamentals of the PSoC 3 and PSoC 5 low-power modes, provides information on Active mode power-saving methods, and discusses other low-power considerations. It is assumed that the reader is familiar with PSoC 3 and PSoC 5 device architecture and PSoC Creator operation. A list of related documents that expand on some complex topics mentioned here is available at the end of this application note.

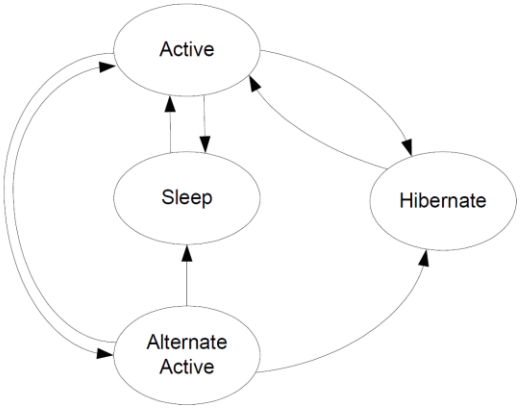
Power Mode Transitions

PSoC 3 and PSoC 5 devices feature four modes of operation – Active, Alternate Active (AltAct), Sleep, and Hibernate.

- Active mode is the primary operating mode of the device and is the default power mode at boot. Active mode typically consumes the most power.
- AltAct mode is similar to Active mode – it is an alternate power configuration for Active mode.
- Sleep mode disables most subsystems to reduce average current consumption (~1 µA in PSoC 3 and ~2 µA in PSoC 5). Wakeup time is 15 µs max in PSoC 3 and 125 µs max in PSoC 5.
- Hibernate mode disables all but the absolute minimum resources to provide the greatest power savings (~200 nA in PSoC 3 and ~300 nA in PSoC 5). Wakeup time in PSoC 3 is 100 µs max.

Figure 1 shows the power mode transitions.

Figure 1. PSoC 3 and PSoC 5 Power Mode Transitions



Active and AltAct modes can transition to any other mode. Sleep and Hibernate modes always wake up and transition back to Active mode.

Making a transition from one power mode to another affects the functionality of all subsystems throughout PSoC 3 and PSoC 5. The APIs provided by PSoC Creator help simplify and manage the power mode transition processes.

Active Mode

Any valid wakeup or reset event returns the PSoC 3 and PSoC 5 to Active and enables the CPU. A return to Active is usually automatic, so there is no API function for this transition.

The typical way to exit Active is to call a power mode API function. The power mode API functions prepare the PSoC 3 and PSoC 5 for entering a low-power mode, and update the register that controls the global power mode setting. See Appendix C – Power Management API and Registers for more information. You do not need to call API functions to exit Active mode, but it is strongly recommended.

Alternate Active Mode

The typical way to enter AltAct is to call the API function `CyPmAltAct()`. If any interrupts are pending, the PSoC 3 and PSoC 5 immediately return to Active mode.

The PSoC 3 and PSoC 5 automatically return to Active when an unmasked wakeup source causes an interrupt. See Table 1 for the list of wakeup sources available in the PSoC 3 and PSoC 5. You can also exit AltAct in the same manner as Active mode.

Sleep Mode

The typical way to enter Sleep is to call the API function `CyPmSleep()`. If any interrupts are pending, the PSoC 3 and PSoC 5 immediately return to Active mode.

The only way to exit Sleep is through a reset or wakeup event, because the CPU and most subsystems are halted. See Table 1 for the list of wakeup sources available in the PSoC 3 and PSoC 5.

Hibernate Mode

The typical way to enter Hibernate is to call the API function `CyPmHibernate()`. If any PICU interrupts are pending, the PSoC 3 and PSoC 5 immediately return to Active mode.

In PSoC 3, a PICU or XRES event returns the PSoC to Active. In PSoC 5, only the XRES reset is available.

Wakeup Sources

Wakeup sources are grouped into three types – periodic, asynchronous, and reset.

- Periodic wakeup sources include the CTW, OPPS, and LCD timers. The PSoC Creator components RTC and SleepTimer set the period of these timers.
- Asynchronous wakeup sources include the boost converter, comparator, I2C, LVI, and PICU.
- Reset wakeup sources include XRES and WDT.

Each low-power mode supports some of these wakeup sources. Table 1 shows the wakeup sources available for the PSoC 3 and PSoC 5 family and power mode.

Multiple Wakeup Sources

PSoC 3 applications can use multiple wakeup sources. For example, it may need to wake periodically to check battery status (OPPS), or when a button is pressed (PICU), or if the internal temperature is too high (Comparator).

To configure multiple wakeup sources, call the power mode API function with multiple parameters OR'ed together. You must read an interrupt status register after wakeup to determine the wakeup source. See Appendix C – Power Management API and Registers for details.

Table 1. Low-power Modes and Wakeup Sources

Wakeup Source	PSoC 3			PSoC 5			Notes
	AltAct	Sleep	Hibernate	AltAlct	Sleep	Hibernate	
Interrupt	✓			✓			Interrupt component must be used
CTW	✓	✓		≈[5]	✓[3]		Sleep time is configurable
SleepTimer	✓	✓		≈[5]	✓[3]		SleepTimer uses the CTW
OPPS	✓	✓					Requires 32-kHz crystal
RTC	✓	✓					RTC uses the OPPS interrupt
FTW	✓						
PICU	✓	✓	✓	≈[5]			Any unmasked pin interrupt
Comparator	✓	✓		≈[5]			
I^2C Address Match	✓	✓		≈[5]			Fixed block slave address match
Segment LCD Refresh	✓	✓[4]					Period depends on settings
Boost Converter	✓	✓[1]	[2]				
WDT	✓	✓		✓	✓		WDT issues a reset upon wakeup
LVI	✓	✓		✓[6]			
XRES	✓	✓	✓	✓	✓	✓	Device wakes and resets

1 In PSoC 3 Sleep mode, the Boost can be used in its Active or Standby mode. Standby mode is recommended.

2 Hibernate is not recommended for applications requiring Boost. Use Sleep mode instead.

3 In PSoC 5, the SleepTimer (CTW) wakeup period is limited to a maximum of 128 ms.

4 The LCD low-power feature cannot be used in conjunction with the CyPmSaveClocks() function.

5 In PSoC 5, these wakeup sources continue to operate in AltAct mode and can be used in conjunction with an interrupt component.

6 In PSoC 5, only the reset function is available in AltAct mode.

Active Mode Power Reduction

Your design may not be able to use the low-power modes, or it may need to spend most of its time in Active mode. You can still reduce average power consumption in Active mode without going into low-power modes.

Power Down Unused Components

One of the easiest ways to reduce power in Active mode is to turn off unused components.

Any component that can be disabled in Active has a Stop function in its API. This function immediately halts all operation of the component and sets it to its lowest power state. The component may be actively performing a task, so check its status before stopping it.

```
/* <Check task status.> */

/* Stop the component. */
MyComponent_Stop();
```

After a component is stopped, it can be restarted by calling its Start function.

```
/* Start the component. */
MyComponent_Start();
```

Any component that must preserve its configuration data before powering down has a Sleep function in its API. The Sleep function saves all necessary component settings and then calls the Stop function. In a few cases, the Sleep function does nothing but call Stop. If code execution time is important, look in the generated source code to see if you can just call Stop instead.

```
/* < Check task status.> */

/* Sleep the component. */
MyComponent_Sleep();

/* <Do something else here.> */

/* Wake the component. */
MyComponent_Wakeup();
```

When a component is put to sleep, it should be awakened again by calling its Wakeup function. This restores the component to its pre-sleep state. The Start function also brings the component back into operation, but it is reinitialized to its default state.

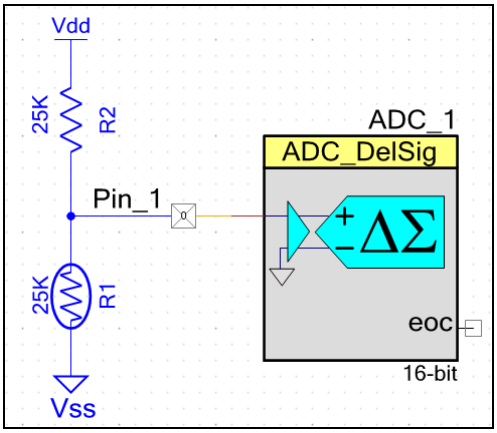
Sleep and Stop both result in the same amount of power savings. The difference is whether the component needs to resume from exactly where it left off. The example projects associated with this application note show how to use Stop/Start and Sleep/Wakeup.

Use PSoC to Gate Current Paths

The PCB may have other components that draw power, and the PSoC 3 and PSoC 5 can be used to control the current through them. Note that the maximum pin source and sink capabilities, listed in the datasheet, must not be exceeded.

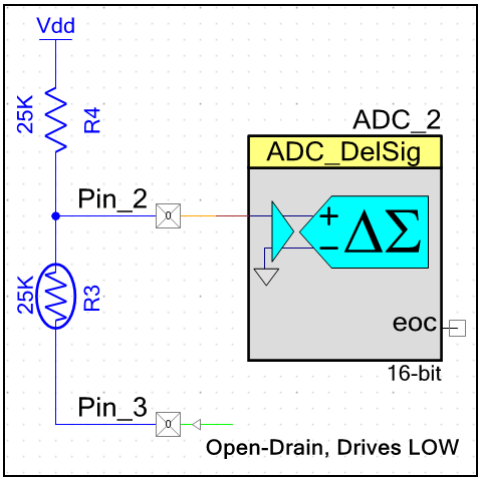
A good example of this scenario is a thermistor application, shown in Figure 2. In this case, the PSoC measures temperature using the voltage on an analog pin, which changes as the thermistor resistance changes.

Figure 2. Typical Thermistor Application



The ADC can be turned off when not in use, but the external components still consume power because nothing stops the flow of current through the resistor and thermistor. An easy solution with PSoC 3 and PSoC 5 is to use a second pin as a switch to ground, as shown in Figure 3.

Figure 3. Using a GPIO as a Ground Switch



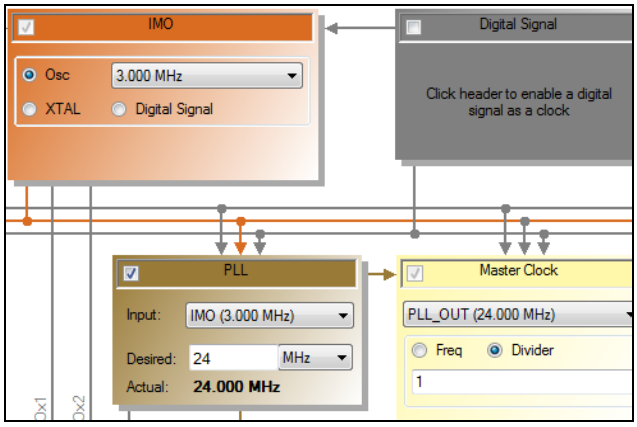
In this configuration, current flow can be stopped by writing a '1' to Pin_3. Writing a '0' resumes current flow. The added cost of this power-saving feature is just one pin and a few lines of code.

Dynamically Change Clock Speeds

PSoC 3 and PSoC 5 can change clock speeds during runtime. This allows you to set the clocks slower most of the time, and then increase their speed when it is needed to perform complex operations.

In a new project, the default clock settings are a 3-MHz IMO feeding a 24-MHz PLL, as shown in Figure 4.

Figure 4. Default IMO and PLL Settings for a New Project



Current consumption can be reduced if the PLL is disabled and the IMO is set to 3 MHz. This is great for an empty project, but your application may require a faster clock.

For example, a 16-bit ADC set to multi-sample mode at 10 k samples per second requires the Master Clock to run at a minimum of 12 MHz. To do this, you should either run the IMO at 12 MHz or the PLL at its minimum of 24 MHz.

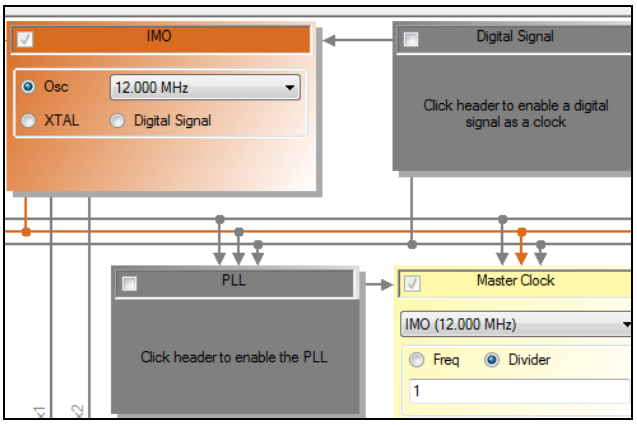
Figure 5. Warning - Clocks Too Slow for ADC Settings

Name	Desired Frequency	Nominal Frequency	Source Clock
USB_CLK	48.000 MHz	? MHz	IMOx2
Digital_Signal	? MHz	? MHz	
XTAL_32KHZ	32.768 kHz	? MHz	
XTAL	24.000 MHz	? MHz	
PLL_OUT	24.000 MHz	? MHz	IMO
ILO	? MHz	1.000 kHz	
BUS_CLK (CPU)	? MHz	3.000 MHz	MASTER_CLK
MASTER_CLK	? MHz	3.000 MHz	IMO
IMO	3.000 MHz	3.000 MHz	
Clock_1	1.000 kHz	1.000 kHz	Auto: IMO
ADC_DelSig_1_theACLK ⚠	2.590 MHz	3.000 MHz	Auto: MASTER_CLK
ADC_DelSig_1_Ext_CP_Clk ⚠	10.360 MHz	3.000 MHz	Auto: MASTER_CLK

When the ADC is not sampling, this project can operate with the IMO set to 3 MHz. This means you can just disable the PLL and switch the IMO between 3 MHz and 12 MHz.

PSoC Creator requires that you configure the clocks to support the speed of all components in your project. In this example, you must make the default configuration run at 12 MHz, as shown in Figure 6.

Figure 6. Clock Settings for Dynamic Changes



After the settings are finalized in the design, you can write firmware to change the clock speeds. This example uses three API functions to configure the PSoC properly:

- `CyIMO_SetFreq()` – this function sets the frequency of the IMO clock.
- `CyFlash_SetWaitCycles()` – this function calculates and sets the number of wait cycles needed for proper flash read and write operations.
- `CyDelayFreq()` – this function calculates and sets the number of cycles needed to accurately time a CyDelay operation.

Other clocks are not automatically changed. You must add code to adjust their settings. Refer to the component datasheets and the *System Reference Guide* for information about the API and registers used to make these adjustments.

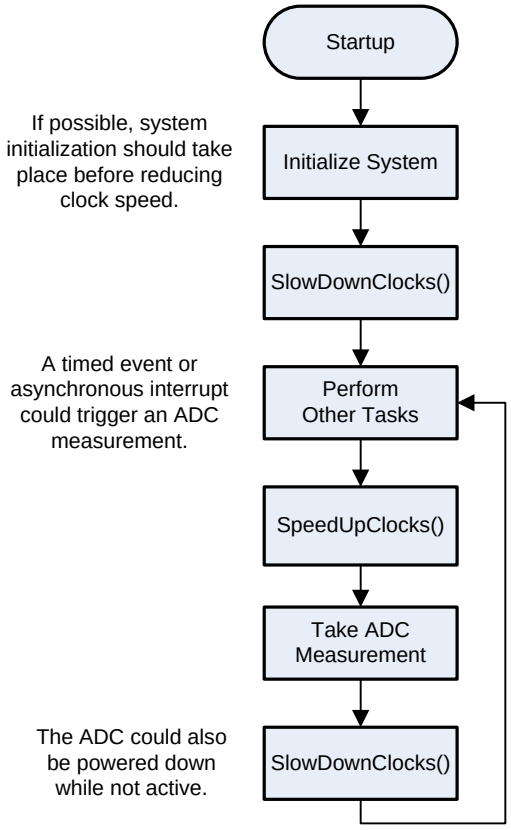
You can create two functions to change the IMO speed:

```
void SlowDownClocks(void)
{
    /* Set IMO frequency to 3MHz. */
    CyIMO_SetFreq(CY_IMO_FREQ_3MHZ);
    /* Set Flash wait to 3MHz. */
    CyFlash_SetWaitCycles(3);
    /* Set CyDelay frequency to 3MHz. */
    CyDelayFreq(3000000);
    /* Change any other active clocks. */
    OtherClock_SetDivider(0); /* 3MHz/1 */
}
```

```
void SpeedUpClocks(void)
{
    /* Set IMO frequency to 12MHz. */
    CyIMO_SetFreq(CY_IMO_FREQ_12MHZ);
    /* Set Flash wait to 12MHz. */
    CyFlash_SetWaitCycles(12);
    /* Set CyDelay frequency to 12MHz. */
    CyDelayFreq(12000000);
    /* Change any other active clocks. */
    OtherClock_SetDivider(3); /* 12MHz/4 */
}
```

In the example, the clock can be slowed to 3 MHz immediately after system initialization. It is only increased to 12 MHz while the ADC is actively taking a sample, as shown in Figure 7.

Figure 7. Flowchart for Clock Changing Example



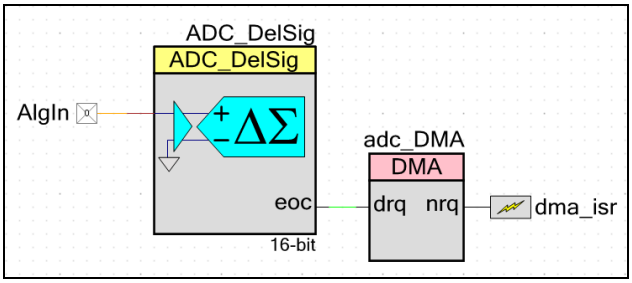
Clock speed adjustments can be used in conjunction with other power saving techniques to reduce the average current consumption without using the low-power modes.

Use DMA to Move Data

You can save power any time you offload a task from the CPU and either halt it or let it do something else in parallel. PSoC 3 and PSoC 5 have a DMA engine that can be used in Active or AltAct to transfer data with no CPU intervention.

In the example shown in Figure 8, the ADC triggers a DMA transfer when a conversion is complete. The DMA engine moves the results of the ADC to another location (without using the CPU), and then it triggers an interrupt to indicate that the transfer is complete.

Figure 8. DMA Triggered by ADC Completion



DMA use is a complex topic in itself and is too large to cover in this document. You can find more information, including example projects and application notes, on http://www.cypress.com.

Use Power Mode Configuration Registers

In Active or AltAct, you can control the power to most subsystems with the Power Mode Configuration registers. The fourteen PM_ACT_CFGx registers gate power and clocks in Active mode. The fourteen PM_STBY_CFGx registers apply to AltAct mode.

Based on the project settings, PSoC Creator generates a set of default values for the Power Mode Configuration registers. These values are stored in the *cyfitter_cfg.c* file and are loaded into the registers at boot. You can change the configuration during runtime with a register write.

```
/* Disable Opamp1 */
CY_SET_REG8(CYDEV_PM_ACT_CFG4 ,
   CY_GET_REG8(CYDEV_PM_ACT_CFG4) && 0xFD);
```

The new settings take effect immediately, but they do not remain beyond a reset. Also, some bits in these registers are automatically set to '1' by interrupts from the subsystems.

It is better to call the Stop function in the component API instead, because it will always be mapped to the correct physical subsystem. The Power Mode Configuration register bitmaps are described in the *PSoC 3 and PSoC 5 Register Technical Reference Manuals*.

Other Power Mode Considerations

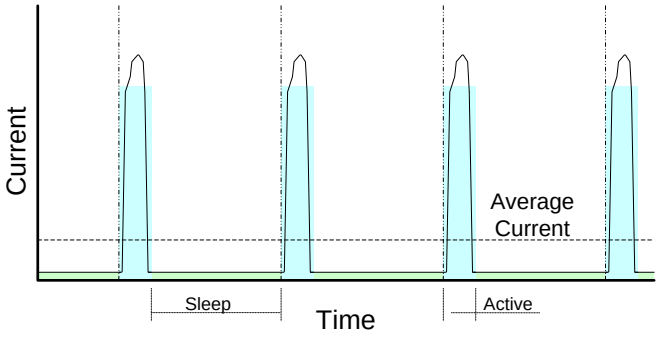
This section discusses a variety of tips, tricks, and recommendations related to the use of PSoC 3 and PSoC 5 low-power modes.

Faster Clocks Can Mean Lower Power

Running the clocks faster can mean lower average current consumption in some cases. For example, consider a PSoC design that takes a reading from a sensor once every second, performs several parsing and calculation operations, and then transmits the results to another device.

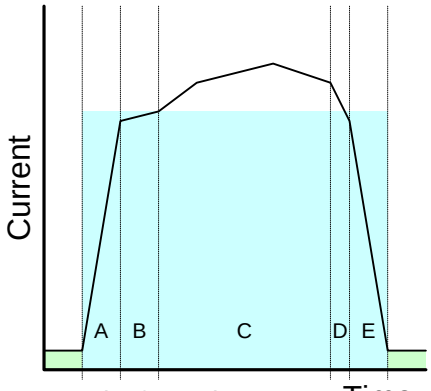
Sleep can be used to reduce power use when the PSoC Is idle, but the average current is higher because of the time spent in Active. Figure 9 is a representation of the current consumption of this example with the system clocks set at 3 MHz.

Figure 9. Example Current Profile with 3-MHz Clocks



If you look at the tasks that are being performed when the PSoC is awake, it may be possible to complete them earlier by running the system clocks faster. This can reduce the average current because the PSoC 3 and PSoC 5 are in Active for less time. Figure 10 is a representation of Active mode timing, broken up into tasks.

Figure 10. Analysis of Tasks in Active at 3 MHz

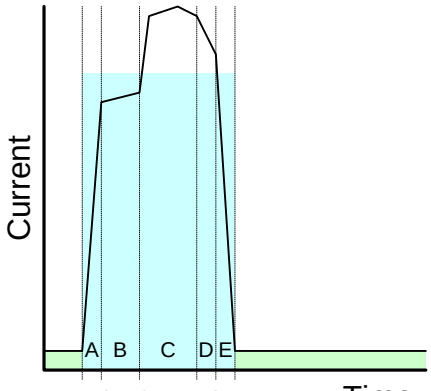


A – Wake from sleep.
B – Read sensor data.
C – Manipulate data.
D – Transmit result.
E – Go back to sleep.

The time required for some tasks will not change even if the system clock frequency increases – sensor reading and data transmitting fall into this category. The other tasks, however, will require less time if the CPU operates at a faster frequency.

At some point, the benefit of a shorter Active time is overcome by the energy required to drive the clocks at a higher rate. Let us assume that the optimal speed is 12 MHz, as shown in Figure 11.

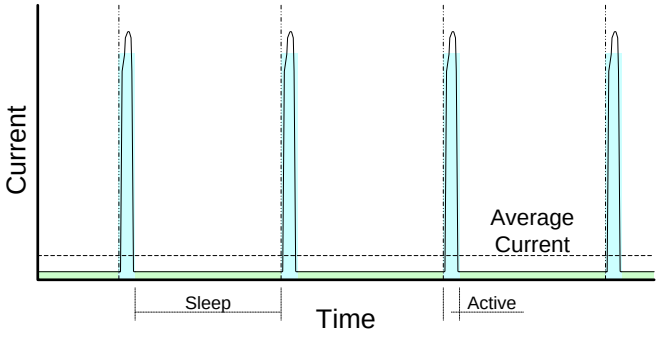
Figure 11. Analysis of Tasks in Active at 12 MHz



A – Wake from sleep.
B – Read sensor data.
C – Manipulate data.
D – Transmit result.
E – Go back to sleep.

The time spent in Active is about half as long as with the slower clocks. Figure 12 shows that the peak current consumption is greater when the clocks are faster, but the overall average is lower.

Figure 12. Example Current Profile with 12-MHz Clocks



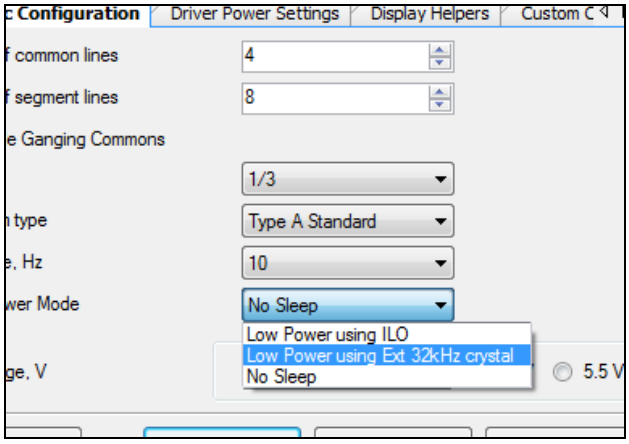
You may even be able to reduce the peak Active current by applying some of the other suggestions found in this application note.

More information on PSoC clocking is available in the *PSoC 3 Clocking Resources* application note.

SegLCD in PSoC 3 Sleep Mode

The Direct Drive Segment LCD (SegLCD) component has a low-power configuration to ensure that the LCD segments are refreshed when the PSoC 3 is in Sleep.

Figure 13. SegLCD Configuration Wizard



The `CyPmSaveClocks()` function cannot be used and the project must be configured to run at 12 MHz for the SegLCD component to operate properly in Sleep mode. This is because the `CyPmSaveClocks()` function turns off the digital clocks that the component uses, and the `CyPmSleep()` function sets the system clocks to run from the IMO at 12 MHz.

PSoC Creator includes two SegLCD example projects to demonstrate operation in both Active and Sleep. Refer to these projects for details on implementing the SegLCD component in a system that uses Sleep mode. You can also refer to *AN52927 PSoC 3 Segment LCD Direct Drive* for more information.

Boost in PSoC 3 Sleep Mode

The PSoC 3 boost regulator circuit can be used in Active, AltAct, and Sleep mode. It has two modes of operation:

- Boost Active is used in Active or AltAct. In this mode, the Boost circuit monitors the output voltage and can support full PSoC 3 functionality.
- Boost Standby is a low-power state that provides enough power for the PSoC 3 to operate while in Sleep.

The PSoC 3 cannot operate for more than a few microseconds (depending on the configuration) in Active or AltAct modes when the boost circuit is in Standby mode. Change the boost mode to Standby immediately before putting the PSoC 3 to sleep. You should also set the boost to Active mode as soon as possible after wakeup.

```
/* Set system clocks for low-power. */
CyPmSaveClocks();
/* Set boost to Standby mode. */
BoostConv_SetMode
   (BoostConv_BOOSTMODE_STANDBY);
/* Sleep PSoC 3 until wakeup event. */
CyPmSleep
   (PM_SLEEP_TIME_NONE,
PM_SLEEP_SRC_BOOSTCONVERTER);
/* Restore boost to Active mode. */
BoostConv_SetMode
   (BoostConv_BOOSTMODE_ACTIVE);
/* Restore system clocks. */
CyPmRestoreClocks();
```

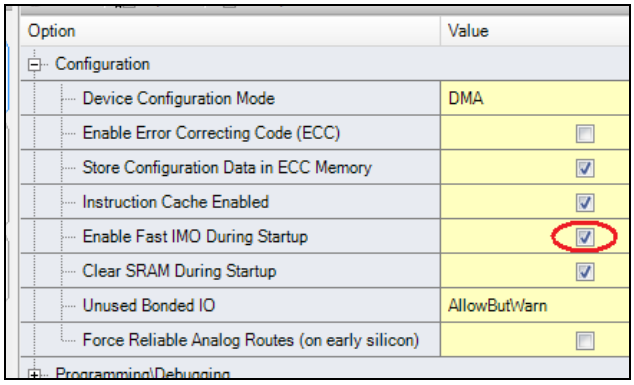
The external 32-kHz crystal is used to time the automatic refresh, or “thump”, of the boost circuit in Sleep. Thump is enabled by default if the API function is used to put the boost converter in Standby mode.

Additional details about the boost circuit are available in the PSoC 3 and PSoC 5 datasheets and TRM. More information on the boost component is available in the component datasheet and *System Reference Guide*.

PSoC 3 Fast IMO Startup

The PSoC 3 has a Fast IMO (FIMO) feature that starts the IMO at 48 MHz to speed up boot. This means that current consumption may be higher during boot than regular operation. This feature is enabled by default in the *Design Wide Resources* file of a PSoC Creator project, as shown in Figure 14.

Figure 14. Fast IMO Selection in the .cydwr Tab



Disabling this feature can reduce current at startup, but it results in a slower boot and initialization period. Refer to the *Clocking System* section of the *PSoC 3 and PSoC 5 TRM* for more information about the FIMO feature and its impact on boot time.

Watchdog in Sleep

The watchdog timer can operate in Active, AltAct, and Sleep. Three options are available for low-power watchdog behavior:

- No change – the watchdog continues to run with the specified interval and must be cleared before the interval is reached.
- Maximum interval – the watchdog continues to run and must be cleared, but the interval is set to the maximum (1024 ticks). It returns to the original interval after being cleared for the first time after wakeup.
- Disabled – The watchdog is disabled while the PSoC is in a low-power mode. It is enabled again upon wakeup and runs at the specified interval.

It is recommended that you use the API to ensure that the watchdog is configured as wanted. The "Maximum Interval" option is set in the PM_WDT_CFG registers by default.

The watchdog is not active in Hibernate. It is reset after a wake from Hibernate, but the behavior may not match the configuration defined in user firmware.

More information on the operation of the watchdog timer and the associated API is available in the PSoC 3 and PSoC 5 *TRM* and the *System Reference Guide*.

GPIOs in Low Power

The GPIOs can continue to drive when the PSoC is in a low-power mode. This is helpful when you need to hold other external logic at a fixed level, but it can lead to wasted power if the pins needlessly source or sink current.

You should analyze your design and determine the best state for your GPIOs during low-power operation. If holding a digital output pin at logic 1 or 0 is best, then use the component's Write function to set it.

```
/* Set MyPin to '0' for low power. */
MyPin_Write(0);
```

Any unused GPIO should be configured as Analog Hi-Z unless there is a specific reason to use a different drive mode. If all of the physical pins associated with a Pin component can be set to Analog Hi-Z, then you can use the component's SetDriveMode function.

```
/* Set MyPin to Alg Hi-Z for low power. */
MyPin_SetDriveMode(MyPin_DM_ALG_HIZ);
```

If you need to change only some of the pins that are associated with a Pin component, you can write to the Port Pin Configuration Registers. There is one register for each GPIO pin on the PSoC.

```
/* Set P6[3] to Alg HiZ for low-power. */
    CY_SET_REG8(CYREG_PRT6_PC3,
        CY_GET_REG8(CYREG_PRT6_PC3) & 0xF1);
```

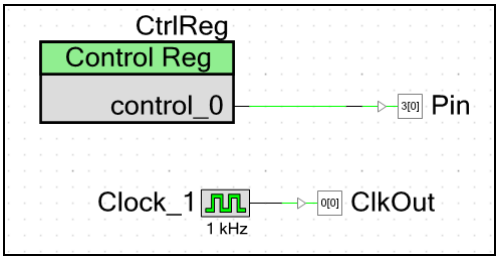
The flexibility of the PSoC 3 and PSoC 5 makes it easy to manage GPIO drive modes to prevent unwanted current leakage. See *AN72382 - Using PSoC 3 and PSoC 5 GPIO Pins* for more information.

Digital Blocks in Low Power

Some subsystems are always powered off in Sleep and Hibernate. If they are connected to other subsystems that remain powered, it can lead to unwanted behavior.

Figure 15 shows a Control Register that is used to set the output of a GPIO pin (the other pin and clock show when the PSoC is awake).

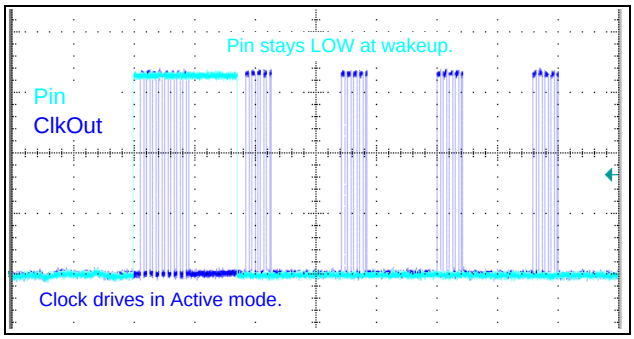
Figure 15. Control Register used to Set Pin Output



The pin continues to drive its last state (HIGH) while the chip is in low-power mode, but the physical block containing the Control Register is unpowered. When the

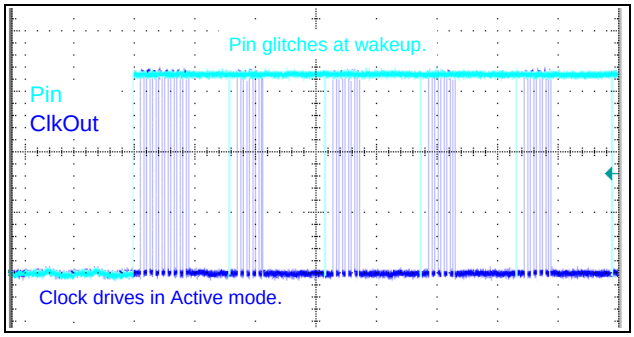
PSoC wakes and power to the digital block is restored, the Control Register bit is reset to '0'. The pin connected to the Control Register then switches from a HIGH to LOW output, as shown in Figure 16.

Figure 16. Control Register not Persistent through Sleep



Even if firmware is added to set the Control Register back to '1' at wakeup, a glitch occurs because the pin logic briefly sees a '0' at wakeup before firmware can change it, as shown in Figure 17.

Figure 17. Control Register Glitch at Each Wakeup



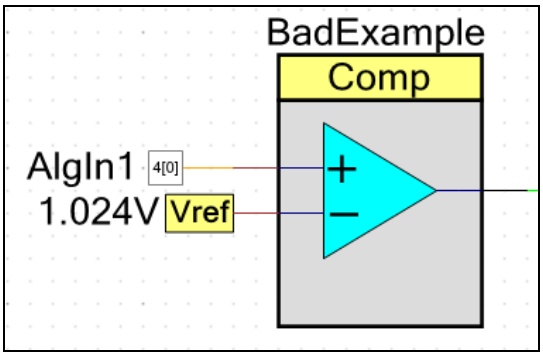
In this case, it is better to control the pin directly, instead of with a Control Register.

Vref Sources in Low Power

Vref sources are always disconnected in PSoC 5 because the Ignore Sleep settings are not allowed. In PSoC 3, Vdda, Vddd, and Vbat remain connected during Sleep or Hibernate, but the other Vref sources do not. Additionally, the Vdda/2 reference remains active, but the voltage divider is disconnected. If a reference voltage is required during Sleep or Hibernate, you must use an external source.

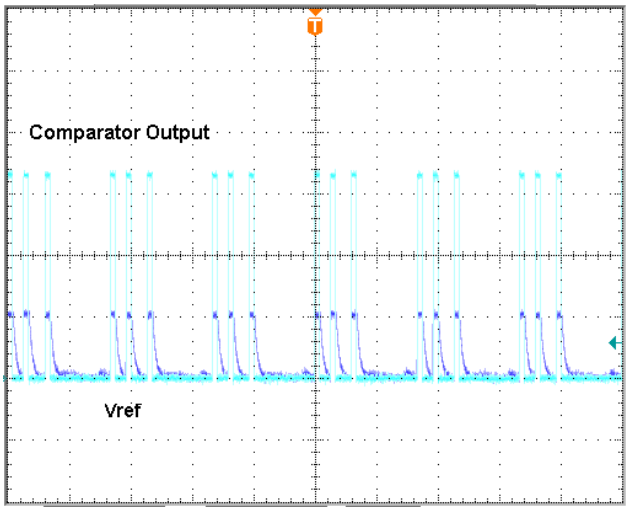
Figure 18 shows an example of this situation. The comparator is configured as a wakeup source, and it has a Vref component connected to the negative terminal.

Figure 18. How Not to Use Vref with Low Power



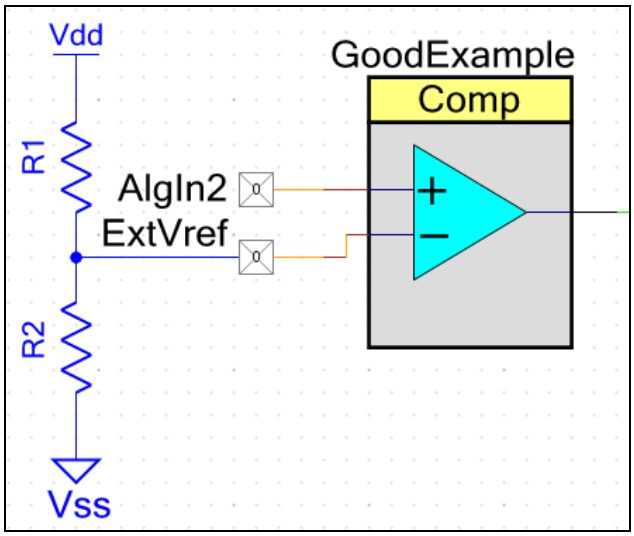
This is fine as long as the PSoC is in Active or AltAct, but the Vref is disconnected in low-power mode. The result is intermittent wakeups when the negative terminal floats, as shown in Figure 19.

Figure 19. Intermittent Wakeup as Comparator Input Floats



The solution to this issue is to supply an external reference voltage, as shown in Figure 20. This reference remains active while the PSoC 3 is in Sleep, so the Comparator wakeup source functions as wanted.

Figure 20. Using an External Vref with Low Power



Very little current is needed on the input terminals, so large resistances can be used to limit the power lost through the divider.

Sleep and Hibernate Regulators

The PSoC 3 and PSoC 5 have two low-power regulators used to maintain logic states in Sleep and Hibernate.

- The sleep regulator provides enough power for a fast wakeup, and to meet the needs of subsystems that remain active in Sleep.
- The hibernate regulator provides only enough power to maintain logic states in essential registers, memories, and latches during Hibernate.

The output of these regulators can be seen on the Vccd and Vcca pins, but neither provides enough current to power anything but the internal PSoC 3 and PSoC 5 resources. You should not attempt to use these regulators for anything else.

Power Consumption While Programming

The PSoC 3 and PSoC 5 consume approximately 14 mA during programming. This is because the programming and debug logic is enabled along with the clocks and CPU.

If your design cannot supply this level of current, then you must power the PSoC from an external source, such as the MiniProg3 during programming or debug. Refer to the PSoC Creator help files for information on how to configure the MiniProg3 to supply power to the PSoC.

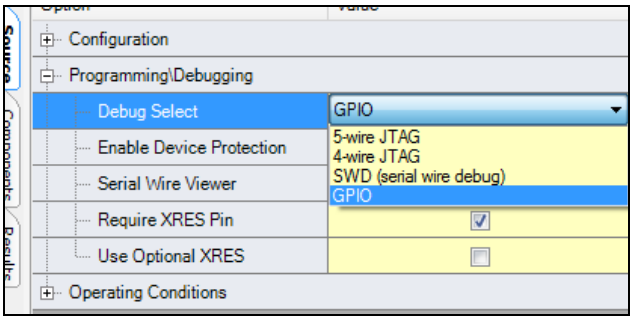
Is Your Debug Interface Running?

Every PSoC 3 and PSoC 5 supports on-chip debug. You may observe higher current consumption than you expect while in debug mode. This is normal, because the programming and debug interface remains active in low-power modes in PSoC 3, and the PSoC 5 does not enter low-power modes while the debugger is running.

Power measurements may also be skewed if the debug pins are set to SWD or JTAG mode and a MiniProg3 is attached – even if the PSoC is not in debug mode.

The debug interface pins are set to GPIO mode on all chips from the factory, but a new PSoC Creator project sets them to SWD mode by default. The registers that control the debug interface can only be changed at programming time. Use the *System* tab in the *cydwr* file of the PSoC Creator project to set the pins to GPIO mode, as shown in Figure 21.

Figure 21. Disable Debug Interface to Reduce Power



You can still program and debug even if the pins are set to GPIO mode. If the debug interface is disabled, a reset must be done to access the debug controller inside the PSoC. This means that the only thing you cannot do is attach the debugger to a running project.

It is recommended that the debug interface pins be set to GPIO mode for any released product. More information about programming and debugging can be found in the device datasheets and *TRMs*.

Summary

Power consumption can mean the difference between a good idea and a successful design. By taking advantage of the many power-saving features available in the PSoC 3 and PSoC 5, you can optimize your design and ensure that it consumes the least amount of power.

Related Application Notes

These application notes give you more information relating to topics that are not fully discussed here:

- AN54181 - Getting Started with PSoC 3
- AN77759 - Getting Started with PSoC 5
- AN62083 - PSoC 3 to PSoC 5 Migration Guide
- AN61290 - PSoC 3 and PSoC 5 Hardware Design Considerations
- AN54460 - PSoC 3 and PSoC 5 Interrupts
- AN60616 - PSoC 3 Startup Procedure
- AN60631 - PSoC 3 Clocking Resources
- AN72382 - Using PSoC 3 and PSoC 5 GPIO Pins
- AN60580 - SIO Tips and Tricks in PSoC 3 / PSoC 5
- AN52705 - PSoC 3 and PSoC 5 - Getting Started with DMA
- AN52927 - PSoC 3: Segment LCD Direct Drive

About the Author

Greg Reynolds has been with Cypress in several different roles for more than a decade, but is sometimes convinced that it has been much, much longer. He always wanted to be a Chemical Engineer, but it was not until the second year of college that he realized how much he hated chemistry and loved electronics. He thinks it is for the best that he changed his major.

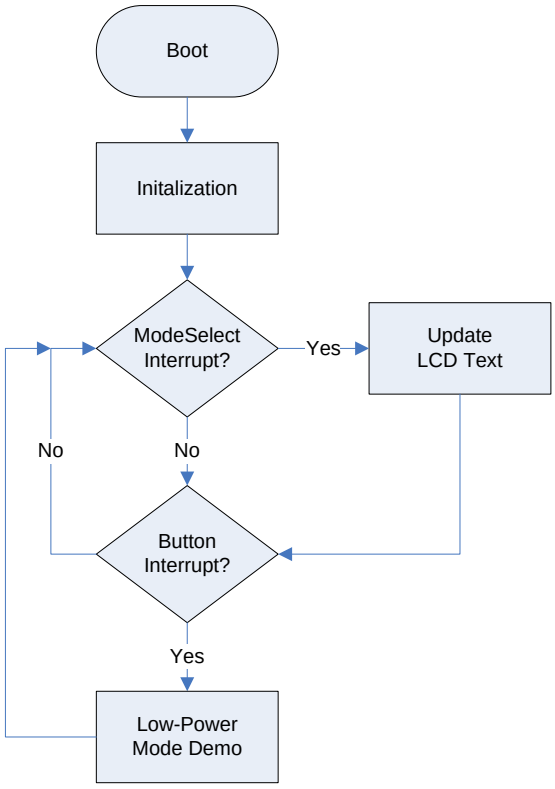
Appendix A – Low-Power Example Projects

There are four example projects associated with this application note. They are all part of the same PSoC Creator workspace and can run on the same development kit (DVK) hardware setup.

PSoC 3 Low-Power Demo

This project demonstrates PSoC 3 power mode transitions and wakeup sources. Select the power mode and wakeup source to be demonstrated by setting the Port 0 pins to a specified value. A falling edge on P2[7] causes the PSoC to execute the code to demonstrate the selected configuration. Figure 22 illustrates the basic flow.

Figure 22. PSoC 3 Low-power Demo Flowchart



The project demonstrates all of the Sleep and Hibernate wakeup sources except Segment LCD and Boost. SegLCD has its own example projects in PSoC Creator; using Boost in sleep is explained in the main body of this application note. Only two of the AltAct sources are demonstrated because they are generally the same as in Sleep. Table 2 lists the modes and wakeup sources that are demonstrated by this example project.

Table 2. PSoC 3 Demo Wakeup Sources

Mode	Wakeup Source
Active	None
AltAct	PICU
	RTC
Sleep	PICU
	RTC
	SleepTimer
	CTW
	Comparator
	I2C Address
	LCD (Not implemented)
Hibernate	PICU
Custom	None (User Defined)

This code demonstrates the principles and techniques described in this application note. No effort has been made to optimize it for size or speed. Instructions for using the projects are found in the project's schematics and source files.

PSoC 5 Low-Power Demo

This project demonstrates PSoC 5 power mode transitions and wakeup sources. The overall flow of the project is the same as the PSoC 3 low-power demo, and it uses the same chip pinout.

The project demonstrates all of the Sleep and Hibernate wakeup sources. Two AltAct sources are demonstrated to illustrate a periodic and asynchronous approach. Table 3 lists the modes and wakeup sources that are demonstrated by this example project.

Table 3. PSoC 5 Demo Wakeup Sources

Mode	Wakeup Source
Active	None
AltAct	Pin (Interrupt)
	Sleep Timer (Interrupt)
Sleep	SleepTimer
Hibernate	None (XRES)
Custom	None (User Defined)

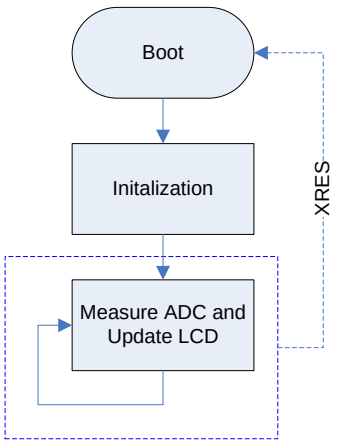
This code demonstrates the principles and techniques described in this application note. No effort has been made to optimize it for size or speed. Instructions for using the projects are found in the project's schematics and source files.

Voltage Alarm – No Optimization

This project demonstrates a simple voltage measurement and alarm system. It is not optimized for low power consumption. The default clocking and global power settings are unchanged, and no low-power modes are used.

A DelSig ADC reads the value of an analog input and the reading is displayed on the LCD. The RTC component is also operating and a time value is displayed on the LCD. If the voltage on the input exceeds a defined level, then an LED begins to pulse and "Alarm" is displayed on the LCD.

Figure 23. Alarm System – No Optimizations

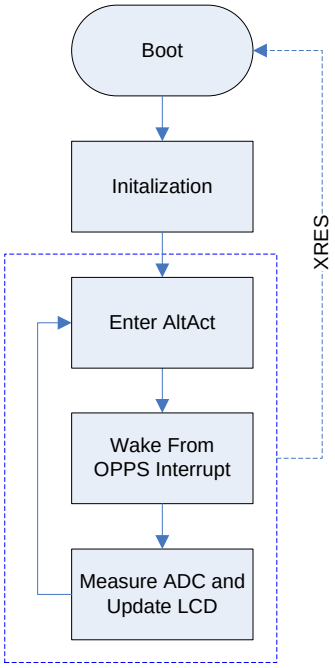


Instructions for using the project are found in the project's schematic and source files.

Voltage Alarm – With Optimizations

This project demonstrates the same basic functionality as the previous voltage measurement and alarm. Several low-power optimizations have been implemented to reduce power consumption.

Figure 24. Alarm System – With Optimizations



Instructions for using the project are found in the project's schematic and source files.

Appendix B – Power Measurements on DVKs

The Cypress development kits are designed to best demonstrate the analog and digital features available in the PSoC devices. They are not optimized for easy measurement of the current drawn by the PSoC device alone. This appendix describes how to modify the CY8CKIT-001, CY8CKIT-030, and CY8CKIT-050 boards for accurate PSoC power measurement. It is recommended that you become familiar with the schematics for these boards before making any modifications. The schematics are available on http://www.cypress.com; search for the DVK part number (CY8CKIT-001/030/050).

CY8CKIT-030 and CY8CKIT-050 Modifications

The following steps demonstrate how to modify the CY8CKIT-030 and CY8CKIT-050 DVK boards to ensure proper current measurements. The configuration described here uses a single power rail for Vddd, Vdda, and Vddio. Power is measured by placing a meter in series with the PSoC's Vddd pins.

1. Remove R23. This separates the VDDA net from the VDDA_P net.
2. Add a zero-ohm resistor to the pads for R30. This ties VDDD to VDDA so that you can measure the total current consumed by the PSoC.
3. Remove R58. This separates the RS-232 driver U8 from VDDA. By default, U8 is powered by VDDA, so it will show up in your power measurement.
4. If RS-232 is needed, short pin 2 of J37 (above the "J37" text) to the VDDA_P test point. This powers the RS-232 driver from VDDA_P instead of VDDA so that it will not show up in your power measurements.
5. Remove the jumper from J30. By default, R56 is powered by VDDA, so it will show up in your power measurement.
6. Short pin 2 of J30 (below the "J30" text) to the VDDA_P test point. This powers the potentiometer R56 from VDDA_P.
7. Remove the jumper from J10.
8. Connect the LOW side lead from the multimeter to the center pin of J10.
9. Connect the HIGH side lead from the multimeter to one of the outside pins (for 5-V or 3.3-V operation) of J10.

After these modifications are made, the total current drawn by the PSoC can be accurately measured. All the other components on the DVK board are powered separately from Vddd, so they will not be seen in your power measurements. The DVK can be configured for 5-V or 3.3-V operation without any further changes by using J10 and J11.

CY8CKIT-001 Modifications

The following steps demonstrate how to modify the CY8CKIT-001 DVK board to ensure proper current measurements. The configuration described here uses a single power rail for Vddd, Vdda, and Vddio. Power is measured by placing a meter in series with the PSoC's Vddd pins.

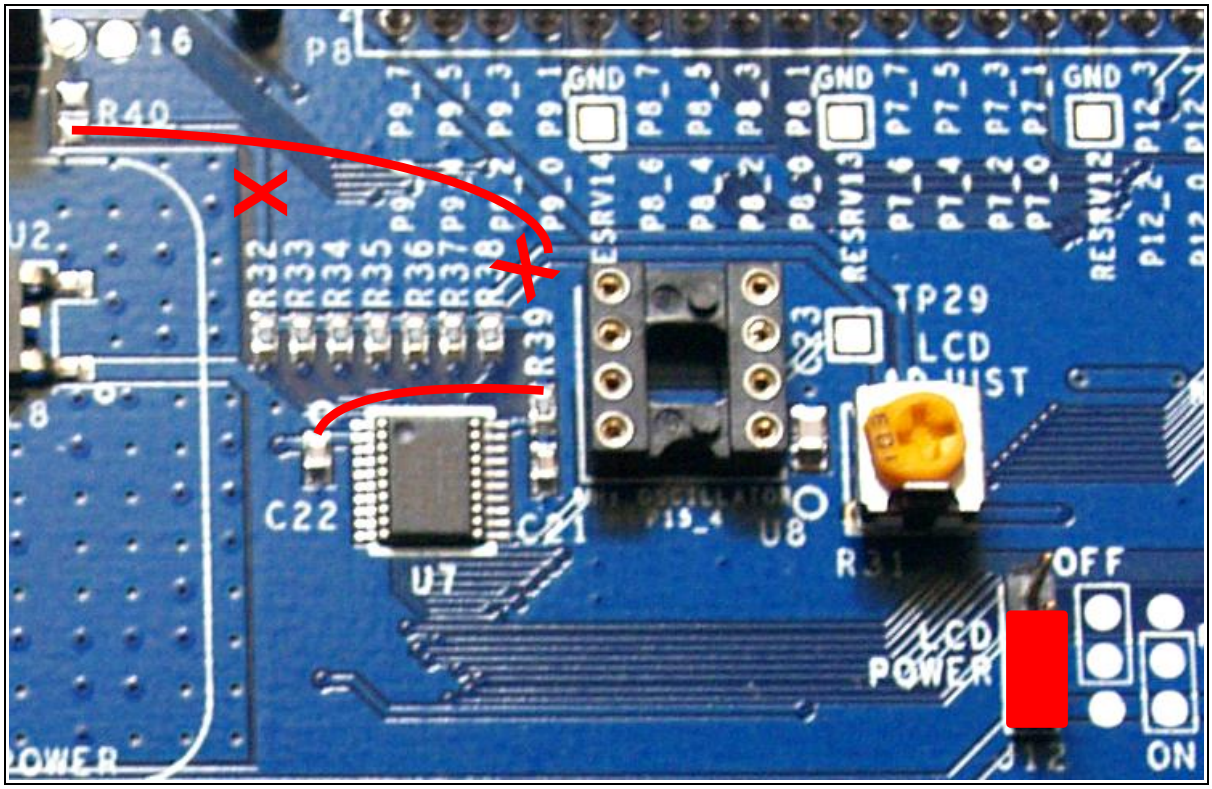
1. Remove jumpers from J2, J3, J4, J5, J6, J7, J10, and J11.
2. Short the center pin of J6 to the center pin of J7. This ties the VDD_DIG and VDD_ANLG nets together.
3. Short the center pins of J2, J3, J4, and J5 to the center pin of J6. This ties the VDDIO nets to VDD_DIG.
4. Short pin 1 of J11 (above the "VR" text) to pin 1 of J10 (next to the "RS232" text). This sources the potentiometer R20 from the VDD net instead of VDD_ANLG.
5. Connect the LOW side lead from the multimeter to the center pin of J7.
6. Connect the HIGH side lead from the multimeter to the VDD test point.

Note If you are operating at 5 V, then no further modification is needed. If you are operating below 5 V, then the LCD level shifter circuit can source current to the PSoC's I/O pins and skew current consumption numbers in Sleep or Hibernate. The following steps will prevent this and allow the character LCD to work whenever the PSoC is operating at 3.3 V or 5 V. Be warned that this modification involves cutting traces on the PCB. Figure 25 shows the cuts and jumps described in the following steps.

7. Cut the trace that runs through the "R32" text on the top side of the DVK board. It is recommended that you cut it just past the text, before it turns a corner toward R40, where there are no vias or components nearby.
8. Cut the same trace again between where it crosses the "R38" text and the "RESRV14" text. These two cuts separate the pull-up resistors on the high-side shifter signals from the VCC_LCD net.
9. Scrape the soldermask from the trace that was just cut, on the side closest to the "RESRV14" text, to expose copper for soldering.

10. Solder a jumper wire from the bottom pad of R40 (below the “R40” text) to the area where soldermask was scraped away in the previous step. This bypasses the section that was isolated by the two previous cuts and ties the LCD module’s power supply back to the center pin of J12.
11. Solder a jumper wire from the top pad of R39 (below the “R39” text) to the bottom pad of C22 (next to the circle silkscreen of U7). This ties the high side shifter’s pull-ups and reference pins to 3.3V.
12. Set the jumper on J12 to the “ON” position to enable the LCD module.

Figure 25. CY8CKIT-001 LCD Level Shift Bypass for Low-Power Measurements at 3.3 V



After these modifications are made, the total current drawn by the PSoC can be accurately measured. All other components on the DVK board are powered separately from Vddd, so they will not be seen in your power measurements. The DVK can be configured for 5-V or 3.3-V operation, without any further changes, by using SW3.

Appendix C – Power Management API and Registers

Cypress provides API routines for handling the PSoC's transitions between low-power modes. The functions discussed here are the most typically used ones. They are available in *CyPm.c*, which is part of every PSoC Creator project.

The *System Reference Guide* has a detailed section on the power management API, and the PSoC 3 and PSoC 5 *TRMs* have further information on the registers.

CyPmSaveClocks()

This function prepares the PSoC clocks for low-power operation. It should be called immediately before entering Sleep or Hibernate. It is not typically called before entering AltAct because the clocks are expected to remain in operation.

Failure to call this function before entering Sleep or Hibernate may result in undefined behavior.

CyPmRestoreClocks()

This function is used to restore the PSoC to the settings that were saved when `CyPmSaveClocks()` was called. It is typically the first function called after exiting the low-power mode.

CyPmAltAct()

This function manages the transition to AltAct mode. It has two parameters, wakeupTime and wakeupSource. The wakeupTime parameter is used to set the frequency of a timer and unmask it as a wakeup source. The wakeupSource parameter is used to unmask the asynchronous and component-based wakeup sources.

Note If "Interrupt" is configured as an AltAct wakeup source, you cannot mask individual interrupt components. Additionally, the power manager sees all raw interrupts before they are filtered by the interrupt controller. This means that any edge detection or enable settings, which are normally applied to the interrupt signal, are ignored.

CyPmSleep()

This function manages the transition to Sleep mode. It has two parameters, wakeupTime and wakeupSource. The wakeupTime parameter is used to set the frequency of a timer and unmask it as a wakeup source. The wakeupSource parameter is used to unmask the asynchronous and component-based wakeup sources.

You should call `CyPmSaveClocks()` before this function to ensure that the clock states have been properly saved before entering Sleep.

CyPmHibernate()

This function manages the transition to Hibernate mode. It has no parameters because the only valid wakeup sources are PICU (in PSoC 3 only) and XRES.

You should call `CyPmSaveClocks()` before this function to ensure that the clock states have been properly saved before entering Hibernate. Also, the PSoC should not reenter a low-power mode until at least 20 µs after a wake from Hibernate to ensure that the sleep regulator is stable.

Component Low-power API

Most PSoC Creator components have API functions to put the component into a low-power state. The sleep function saves component settings and calls the stop function. There is no power savings difference between calling stop or sleep.

Unless the component is being used as a wakeup source, its sleep function should be called prior to calling `CyPmSleep()` or `CyPmHibernate()`. The sleep function is formatted as:

```
MyComponentName_Sleep();
```

Any component that has a sleep function also has a wakeup function. The wakeup function restores the component to the previous state. It is formatted as:

```
MyComponentName_Wakeup();
```

The sleep and stop functions typically do not check for the component to be idle, so check for the completion of any ongoing tasks. Refer to the individual component datasheets for details on how to manage low-power operation.

Direct Register Writes

The API functions should be used to control power mode transitions whenever possible, but register writes can be used instead. The registers mentioned here are the most commonly used ones related to power mode transitions.

PM_MODE_CSR

The Power Mode Control and Status Register is perhaps the most important register relating to PSoC 3/5 power modes. Bits [2:0] in this register control the global power mode for the entire PSoC, as shown in Table 4. Reading these bits tells you what mode the PSoC is currently in.

Table 4. PM_MODE_CSR[2:0] Values

[2:0] Value	Mode
000	Active mode
001	AltAct mode
010	(Unsupported)
011	Sleep mode
100	Hibernate mode
101	(Unsupported)
110	(Unsupported)
111	(Unsupported)

Setting these bits does not automatically configure the entire PSoC for low-power operation. More information on the global power settings and how they affect the PSoC 3 and PSoC 5 subsystems are available in the PSoC 3 and PSoC 5 *TRMs*.

Note Using an unsupported low-power mode or wakeup source may result in unreliable behavior. Use the standard API to enter low-power modes, whenever possible.

PM_TW_CFGx

These three registers enable and configure the timers used to wake the PSoC 3/5 from low-power modes.

PM_WAKEUP_CFGx

These three registers mask the power mode wakeup sources. Individual sources can be dynamically masked or unmasked to allow for different wakeup sources under different conditions.

PICUx_INTTYPEy

These registers configure the Pin Interrupt Control Unit. There is one register for each pin, to set the interrupt trigger condition.

PICUx_STAT

These registers hold the status of pin interrupts. There is one status register for each port, with one bit per pin.

FASTCLK

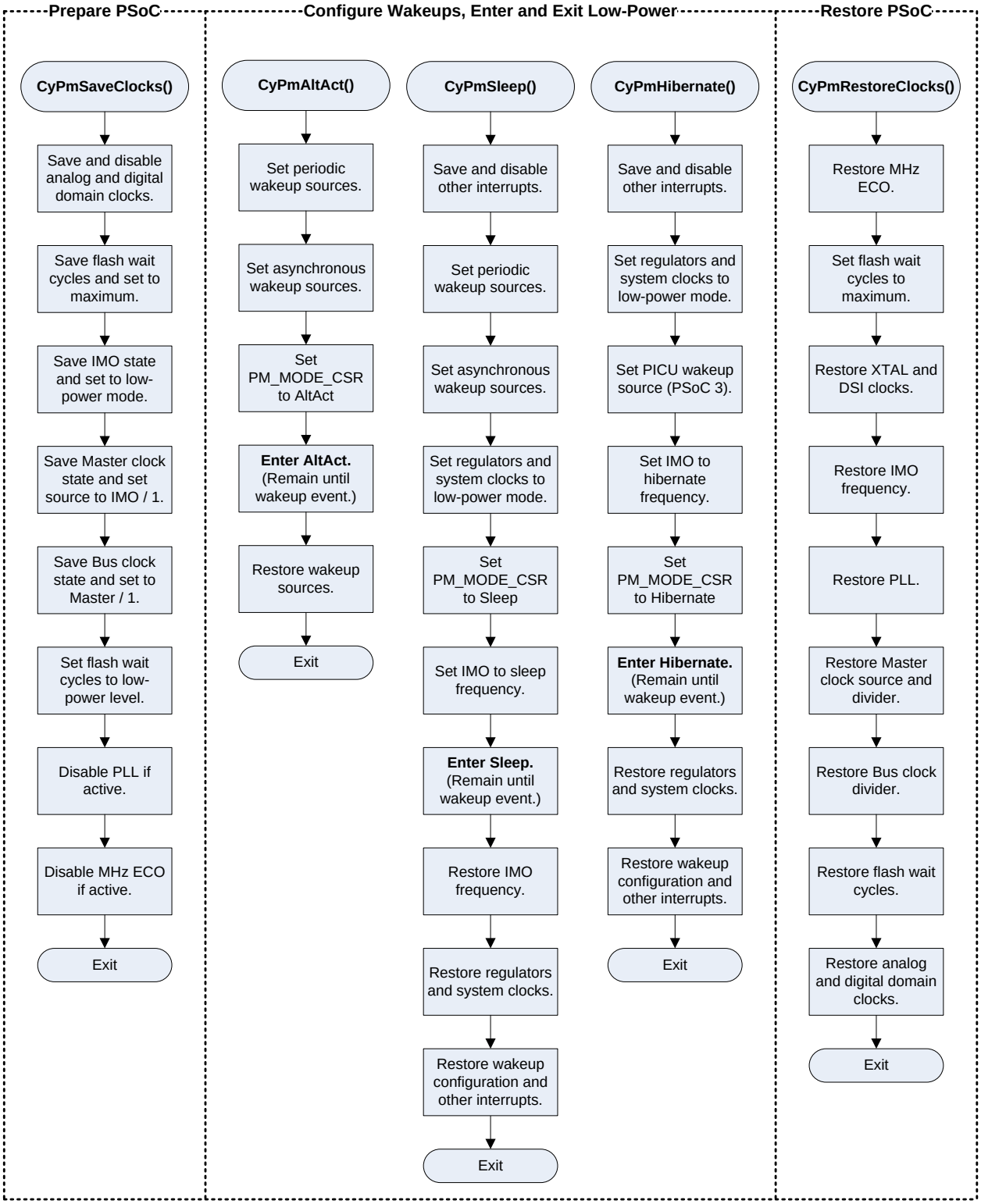
These registers configure the IMO, MHz crystal, and PLL. Clock distribution is controlled by the *CLKDIST* registers.

SLOWCLK

These registers configure the ILO and 32-kHz crystal. Clock distribution is controlled by the *CLKDIST* registers.

Power Management API Flowcharts

Figure 26. Power Management Function Flowcharts



Power Management API Register Reference

The following table lists the asynchronous wakeup sources available in the PSoC 3/5 devices.

Table 5. CyPm Wakeup Source Configuration

API	Wakeup Source	Defined wakeupSource Mask	Affected Register [bits]
CyPmAltAct() [1]	Comparator0	PM_ALT_ACT_SRC_COMPARATOR0	PM_WAKEUP_CFG1 [0x01]
	Comparator0	PM_ALT_ACT_SRC_COMPARATOR1	PM_WAKEUP_CFG1 [0x02]
	Comparator0	PM_ALT_ACT_SRC_COMPARATOR2	PM_WAKEUP_CFG1 [0x04]
	Comparator0	PM_ALT_ACT_SRC_COMPARATOR3	PM_WAKEUP_CFG1 [0x08]
	Any Interrupt	PM_ALT_ACT_SRC_INTERRUPT	PM_WAKEUP_CFG0 [0x01]
	PICU	PM_ALT_ACT_TIME_PICU	PM_WAKEUP_CFG0 [0x04]
	I2C Address	PM_ALT_ACT_TIME_I2C	PM_WAKEUP_CFG0 [0x08]
	Boost	PM_ALT_ACT_TIME_BOOSTCONVERTER	PM_WAKEUP_CFG0 [0x20]
	FTW	PM_ALT_ACT_TIME_FTW	PM_WAKEUP_CFG0 [0x40]
	CTW	PM_ALT_ACT_TIME_CTW	PM_WAKEUP_CFG0 [0x80]
	OPPS	PM_ALT_ACT_TIME_ONE_PPS	PM_WAKEUP_CFG0 [0x80]
	LCD	PM_ALT_ACT_TIME_LCD	PM_WAKEUP_CFG2 [0x01]
CyPmSleep()[2]	Comparator0	PM_ALT_ACT_SRC_COMPARATOR0	PM_WAKEUP_CFG1 [0x01]
	Comparator1	PM_ALT_ACT_SRC_COMPARATOR1	PM_WAKEUP_CFG1 [0x02]
	Comparator2	PM_ALT_ACT_SRC_COMPARATOR2	PM_WAKEUP_CFG1 [0x04]
	Comparator3	PM_ALT_ACT_SRC_COMPARATOR3	PM_WAKEUP_CFG1 [0x08]
	PICU	PM_ALT_ACT_TIME_PICU	PM_WAKEUP_CFG0 [0x04]
	I2C Address	PM_ALT_ACT_TIME_I2C	PM_WAKEUP_CFG0 [0x08]
	Boost	PM_ALT_ACT_TIME_BOOSTCONVERTER	PM_WAKEUP_CFG0 [0x20]
	CTW	PM_ALT_ACT_TIME_CTW	PM_WAKEUP_CFG0 [0x80]
	OPPS	PM_ALT_ACT_TIME_ONE_PPS	PM_WAKEUP_CFG0 [0x80]
	LCD Glass Drive	PM_ALT_ACT_TIME_LCD	PM_WAKEUP_CFG2 [0x01]
CyPmHibernate()[3]	PICU	N/A	PM_WAKEUP_CFG0 [0x04]

[1] PSoC 5 only supports an interrupt component wakeup source in AltAct mode. It is configured by the API automatically when calling CyPmAltAct().

[2] PSoC 5 only supports the CTW wakeup source as part of a SleepTimer component in Sleep mode. It is unmasked by the API automatically when calling CyPmSleep(). SleepTimer intervals in PSoC 5 are limited to 2-8 ms.

[3] PSoC 5 does not support any wakeup sources in Hibernate mode. An XRES reset must be used to wake the PSoC 5.

The following table lists the periodic wakeup sources available in PSoC 3 and PSoC 5 devices.

Table 6. CyPm Wakeup Period Configuration

API	Wakeup Source	Defined wakeupTime Mask	Affected Register [bits]
CyPmAltAct()[1] and CyPmSleep()[2]	OPPS	PM_ALT_ACT_TIME_ONE_PPS	PM_TW_CFG2 [0x20] – Unmask PM_TW_CFG2 [0x10] – Enable
	CTW 2 ms	PM_ALT_ACT_TIME_CTW_2MS	PM_TW_CFG2 [0x08] – Unmask PM_TW_CFG2 [0x04] – Enable PM_TW_CFG1 [0x01] – 2 ms Period
	CTW 4 ms	PM_ALT_ACT_TIME_CTW_4MS	PM_TW_CFG2 [0x08] – Unmask PM_TW_CFG2 [0x04] – Enable PM_TW_CFG1 [0x02] – 4 ms Period
	CTW 8 ms	PM_ALT_ACT_TIME_CTW_8MS	PM_TW_CFG2 [0x08] – Unmask PM_TW_CFG2 [0x04] – Enable PM_TW_CFG1 [0x03] – 8 ms Period
	CTW 16 ms	PM_ALT_ACT_TIME_CTW_16MS	PM_TW_CFG2 [0x08] – Unmask PM_TW_CFG2 [0x04] – Enable PM_TW_CFG1 [0x04] – 16 ms Period
	CTW 32 ms	PM_ALT_ACT_TIME_CTW_32MS	PM_TW_CFG2 [0x08] – Unmask PM_TW_CFG2 [0x04] – Enable PM_TW_CFG1 [0x05] – 32 ms Period
	CTW 64 ms	PM_ALT_ACT_TIME_CTW_64MS	PM_TW_CFG2 [0x08] – Unmask PM_TW_CFG2 [0x04] – Enable PM_TW_CFG1 [0x06] – 64 ms Period
	CTW 128 ms	PM_ALT_ACT_TIME_CTW_128MS	PM_TW_CFG2 [0x08] – Unmask PM_TW_CFG2 [0x04] – Enable PM_TW_CFG1 [0x07] – 128 ms Period
	CTW 256 ms	PM_ALT_ACT_TIME_CTW_256MS	PM_TW_CFG2 [0x08] – Unmask PM_TW_CFG2 [0x04] – Enable PM_TW_CFG1 [0x08] – 256 ms Period
	CTW 512 ms	PM_ALT_ACT_TIME_CTW_512MS	PM_TW_CFG2 [0x08] – Unmask PM_TW_CFG2 [0x04] – Enable PM_TW_CFG1 [0x09] – 512 ms Period
	CTW 1024 ms	PM_ALT_ACT_TIME_CTW_1024MS	PM_TW_CFG2 [0x08] – Unmask PM_TW_CFG2 [0x04] – Enable PM_TW_CFG1 [0x0A] – 1024 ms Period
	CTW 2048 ms	PM_ALT_ACT_TIME_CTW_2048MS	PM_TW_CFG2 [0x08] – Unmask PM_TW_CFG2 [0x04] – Enable PM_TW_CFG1 [0x0B] – 2048 ms Period
	CTW 4096 ms	PM_ALT_ACT_TIME_CTW_4096MS	PM_TW_CFG2 [0x08] – Unmask PM_TW_CFG2 [0x04] – Enable PM_TW_CFG1 [0x0C] – 4096 ms Period
	FTW 10 µs to 2.56 ms (AltAct Only)	PM_ALT_ACT_TIME_FTW(1-256)	PM_TW_CFG2 [0x02] – Unmask PM_TW_CFG2 [0x01] – Enable PM_TW_CFG0 [0x00 to 0xFF] Period
CyPmHibernate()	N/A	N/A	N/A

[1] PSoC 5 only supports an interrupt component wakeup source in AltAct mode. It is configured by the API automatically when calling CyPmAltAct().

[2] PSoC 5 only supports the CTW wakeup source as part of a SleepTimer component in Sleep mode. It is unmasked by the API automatically when calling CyPmSleep(). SleepTimer intervals in PSoC 5 are limited to 2-8 ms.

Document History

Document Title: PSoC® 3 and PSoC 5 Low-power Modes and Power Reduction Techniques

Document Number: 001-77900

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3653707	GIR	06/22/2012	New application note.

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 cypress.com/go/plc
Memory	cypress.com/go/memory
Optical Navigation Sensors	cypress.com/go/ons
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.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 5

Cypress Developer Community

Community | Forums | Blogs | Video | Training

Technical Support

cypress.com/go/support

PSoC is a registered trademark and PSoC Creator is a 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

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com

© Cypress Semiconductor Corporation, 2012. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.
This Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.
Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.
Use may be limited by and subject to the applicable Cypress software license agreement.