AN77900

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

Author: Max Kingsbury

Associated Projects: Yes

Associated Part Family: All PSoC® 3 and PSoC 5LP parts

Software Version: PSoC Creator 2.2 SP1

For a complete list of related application notes, click here

AN77900 is an introduction to the PSoC 3 and PSoC 5LP 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. The associated PSoC Creator project demonstrates these principles.

Contents

Introduction

The PSoC 3 and PSoC 5LP low-power modes allow you to reduce 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 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 5LP 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.

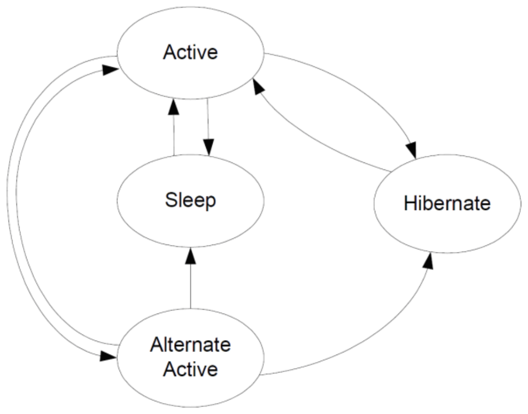
Power Modes and Transitions

PSoC 3 and PSoC 5LP 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. By default, the CPU is disabled.
- Sleep mode disables most subsystems to reduce average current consumption (~1 µA for PSoC 3 and ~2 µA for PSoC 5LP). Wakeup time is 15 µs max for PSoC 3 and 25 µs max for PSoC 5LP.
- Hibernate mode disables all but the absolute minimum resources to provide the greatest power savings (~200 nA for PSoC 3 and ~300 nA for PSoC 5LP). Wakeup time is 100 µs max for PSoC 3 and 125 µs max for PSoC 5LP.

Active and AltAct modes can transition to any other mode. Sleep and Hibernate modes always wake up and transition back to Active mode, as Figure 1 shows.

Figure 1. PSoC 3 and PSoC 5LP Power Mode Transitions



Making a transition from one power mode to another affects the functionality of all subsystems throughout PSoC. 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 to Active mode and enables the CPU. A return to Active mode is usually automatic, so there is no API function for this transition.

The typical way to exit Active mode is to call a low-power mode API function. These functions prepare the PSoC to enter 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 immediately returns to Active mode.

PSoC automatically returns to Active mode when an unmasked wakeup source causes an interrupt. See Table 1 on page 2 for the list of wakeup sources available. You can also exit AltAct mode by transitioning to Sleep or Hibernate.

Sleep Mode

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

The only way to exit Sleep mode is through a reset or wakeup event, because the CPU and most subsystems are halted. See Table 1 on page 2 for the list of wakeup sources available.

Hibernate Mode

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

The only way to exit Hibernate mode is with an enabled PICU interrupt or a hardware reset.

Wakeup Sources

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

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

The low-power modes support some or all of these wakeup sources. Table 1 shows the wakeup sources available for each power mode.

Multiple Wakeup Sources

PSoC applications can use multiple wakeup sources. For example, the PSoC 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 5LP			Notes
	AltAct	Sleep	Hibernate	AltAlct	Sleep	Hibernate	
Interrupt	✓			✓			Interrupt component must be used
CTW	✓	✓					Sleep time is configurable
SleepTimer	✓	✓		✓	✓		SleepTimer uses the CTW
OPPS	✓	✓					Requires 32-kHz crystal
RTC	✓	✓		✓	✓		RTC uses the OPPS
FTW	✓			✓			
PICU	✓	✓	✓	✓	✓	✓	Any unmasked pin interrupt
Comparator	✓	✓		✓	✓		
I^{2}C Address Match	✓	✓		✓	✓		Fixed block slave address match
Segment LCD Refresh	✓	✓[3]		✓	✓[3]		Period depends on settings
Boost Converter	✓	✓[1]	[2]	✓	✓	[2]	
WDT	✓	✓		✓	✓		WDT issues a reset if not fed
LVI	✓	✓		✓	✓		
XRES	✓	✓	✓	✓	✓	✓	Device wakes and resets

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

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

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

Active Mode Power Reduction

Your application 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 Off 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();
```

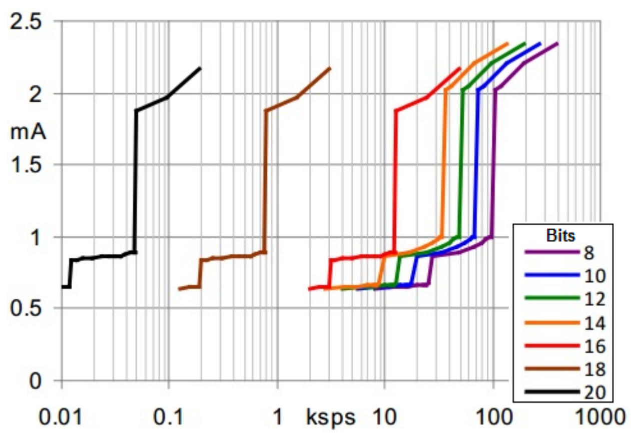
When a component is put to sleep, it should be awakened again by calling its Wakeup function. This restores the component to its pre-sleep state. The Start function also brings the component back into operation, but it is reinitialized to its default state.

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 the ADC at a Slower Sample Rate

The ADCs in PSoC have several different power profiles. The profiles are enabled automatically depending on the resolution and sample rate. For example, the power consumption of the DelSig ADC in PSoC 3 changes greatly when you switch power modes, as Figure 2 shows.

Figure 2. PSoC 3 DelSig Idd vs Sample Rate (Buffered)



A slightly slower sample rate can result in noticeable power savings. In this case, if the ADC was configured for 16-bit resolution, the difference between 10 ksps and 12 ksps is almost 1mA.

If Active mode power consumption is a concern, then it is worthwhile to check the component datasheets to see if a slower sample rate will result in significant power savings.

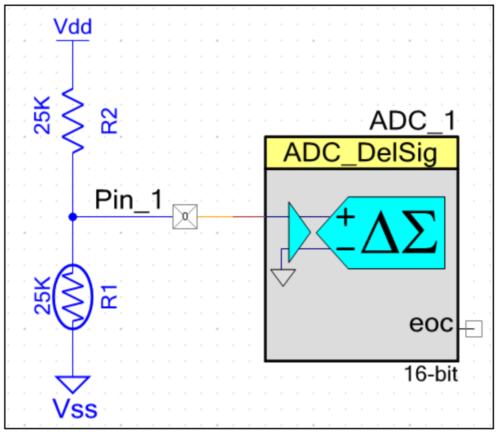
The Delta-Sigma ADC component datasheet and *PSoC 3 and PSoC 5LP TRMs* have more information on selecting the proper sample rate and resolution.

Use PSoC to Gate Current Paths

Your PCB may have other components that draw power, and the PSoC 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.

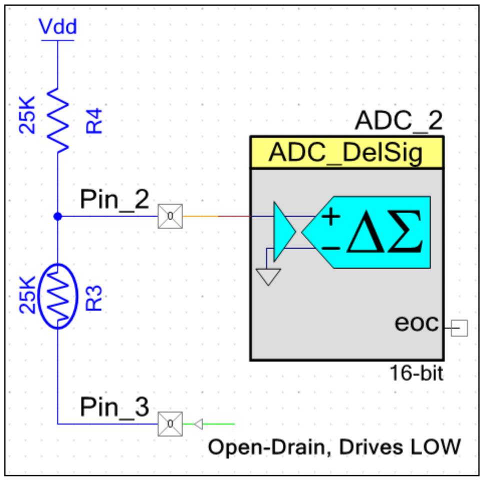
A good example of this scenario is a thermistor application, as Figure 3 on page 5 shows. In this case, the PSoC measures temperature using the voltage on an analog pin, which changes as the thermistor resistance changes.

Figure 3. 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 is to use a second pin as a switch to ground, as Figure 4 shows.

Figure 4. Using a GPIO as a Ground Switch



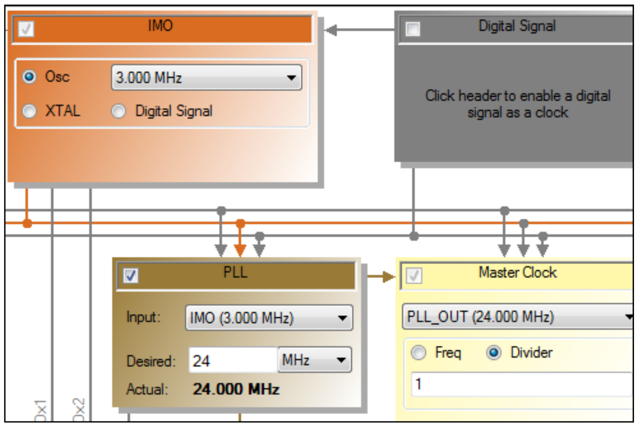
In this configuration, current flow can be stopped by writing a ‘1’ to Pin_3. Writing a ‘0’ resumes current flow. The added cost of this power-saving feature is just one pin and a few lines of code.

Dynamically Change Clock Speeds

PSoC 3 and PSoC 5LP 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 Figure 5 shows.

Figure 5. Default IMO and PLL Settings for a New Project



IMO
[✓ Osc] 3.000 MHz
XTAL
Digital Signal

Digital Signal
Click header to enable a digital signal as a clock

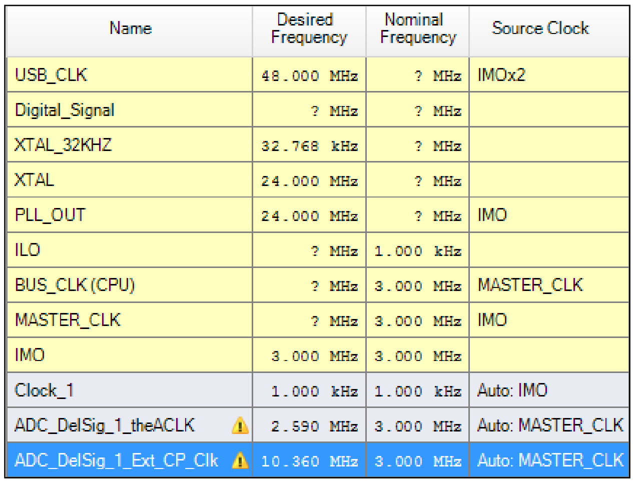
PLL
Input: IMO (3.000 MHz)
Desired: 24 MHz
Actual: 24.000 MHz

Master Clock
PLL_OUT (24.000 MHz)
Freq [✓ Divider]
1

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 10k samples per second requires the Master Clock to run at a minimum of 12 MHz, as Figure 6 shows. To do this, you should either run the IMO at 12 MHz or the PLL at its minimum of 24 MHz.

Figure 6. Warning - Clocks Too Slow for ADC Settings

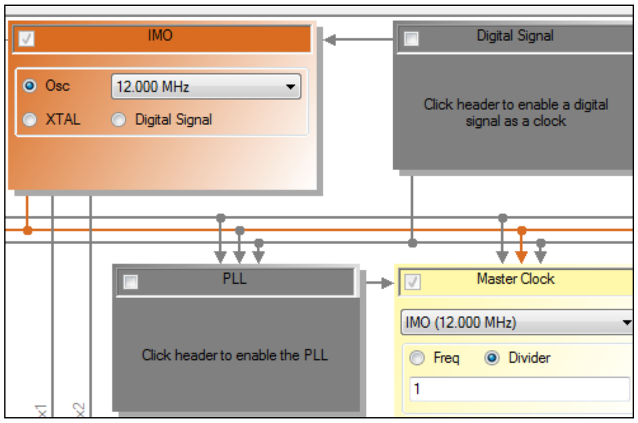


Name	Desired Frequency	Nominal Frequency	Source Clock
USB_CLK	48.000 MHz	? MHz	IMOx2
Digital_Signal	? MHz	? MHz	
XTAL_32KHZ	32.768 kHz	? MHz	
XTAL	24.000 MHz	? MHz	
PLL_OUT	24.000 MHz	? MHz	IMO
ILO	? MHz	1.000 kHz	
BUS_CLK (CPU)	? MHz	3.000 MHz	MASTER_CLK
MASTER_CLK	? MHz	3.000 MHz	IMO
IMO	3.000 MHz	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 Figure 7 on page 6 shows.

Figure 7. 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.

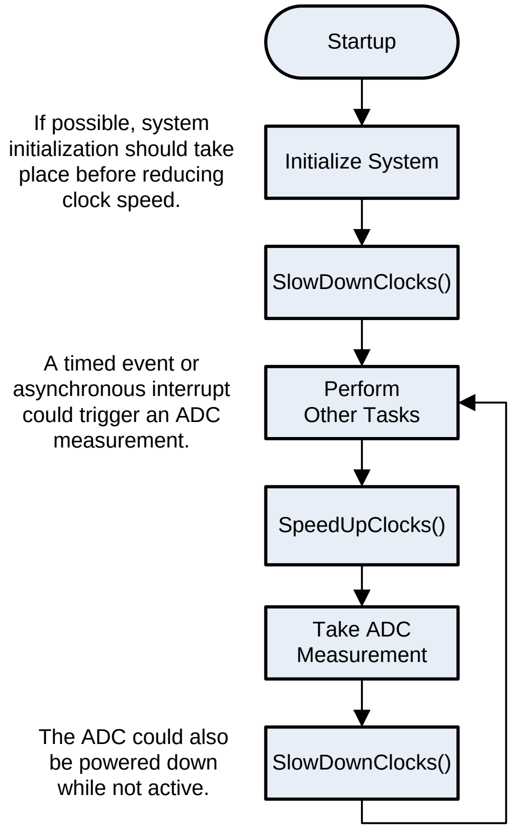
You can create two functions to change the IMO speed:

```
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 Figure 8 shows.

Figure 8. Flowchart for Clock Changing Example


Startup
If possible, system initialization should take place before reducing clock speed.
Initialize System
SlowDownClocks()
A timed event or asynchronous interrupt could trigger an ADC measurement.
Perform Other Tasks
SpeedUpClocks()
Take ADC Measurement
The ADC could also be powered down while not active.
SlowDownClocks()

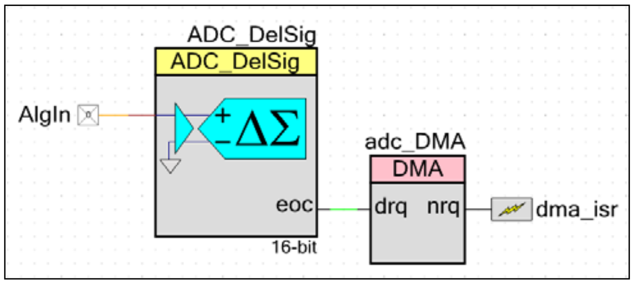
Clock speed adjustments can be used in conjunction with other power saving techniques to reduce the average current consumption without using the low-power modes.

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 5LP 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 9, 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 9. 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, at 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 is always mapped to the correct physical subsystem. The Power Mode Configuration register bitmaps are described in the *PSoC 3 and PSoC 5LP Register Technical Reference Manuals*.

Other Power Mode Considerations

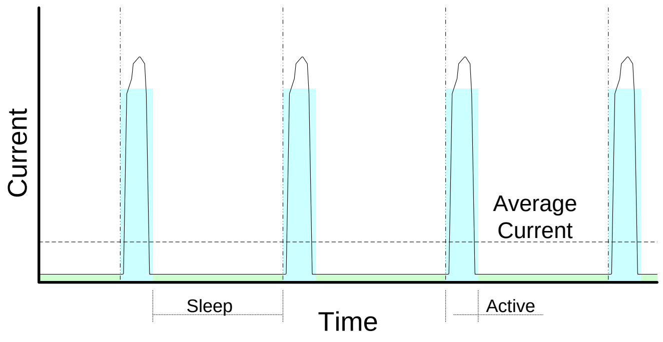
This section discusses a variety of tips, tricks, and recommendations related to the use of PSoC 3 and PSoC 5LP low-power modes.

Faster Clocks Can Mean Lower Power

In some cases, running the clocks faster can actually result in a lower average current consumption. 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.

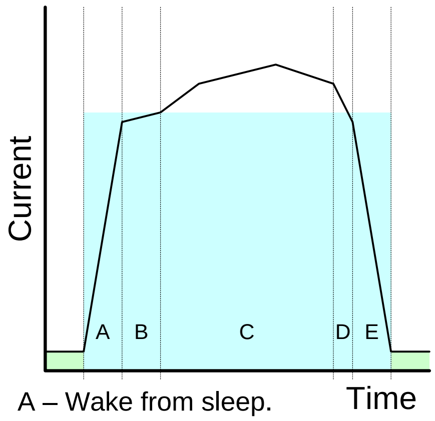
Sleep can be used to reduce power use when the PSoC Is idle, but the average current is higher because of the time spent in Active. Figure 10 is a representation of the current consumption of this example with the system clocks set at 3 MHz.

Figure 10. 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 sooner by running the system clocks faster. This can reduce the average current because the PSoC is in Active mode for less time. Figure 11 on page 8 is a representation of Active mode timing, broken up into tasks.

Figure 11. Analysis of Tasks in Active at 3 MHz

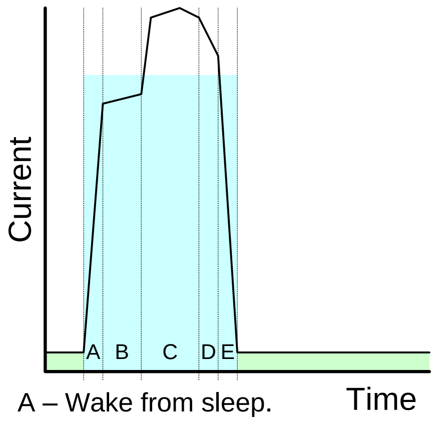


A – Wake from sleep.
B – Read sensor data.
C – Manipulate data.
D – Transmit result.
E – Go back to sleep.

The time required for some tasks does not change even if the system clock frequency increases – sensor reading and data transmitting fall into this category. The other tasks, however, 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 Figure 12. Shows.

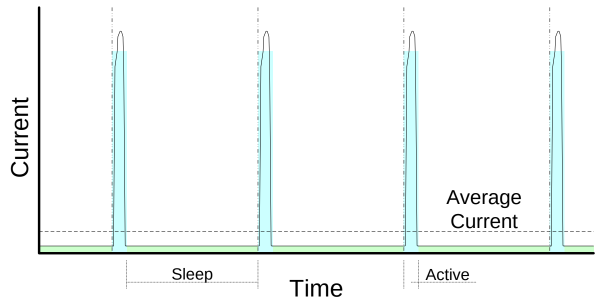
Figure 12. Analysis of Tasks in Active at 12 MHz



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 13 shows that the peak current consumption is greater when the clocks are faster, but the overall average is lower.

Figure 13. 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.

32 kHz Crystal Low-Power Mode

The 32 kHz crystal can be configured to operate in a reduced power mode when the PSoC is in Sleep mode. It is configured to operate at normal power by default. Use the following function near the top of main() to enable this feature:

```
CyXTAL_32KHZ_SetPowerMode(1);
```

The crystal continues to operate at normal power during Active and AltAct modes; it is disabled during Hibernate. The reduced power mode is ~1 uA less than normal power mode.

The *System Reference Guide* has more information about this function.

Low-voltage Interrupts in PSoC Sleep Mode

The low-voltage interrupt (LVI) subsystem can be used to wake the PSoC from Sleep, but it consumes ~1 uA of power when enabled. If your application uses the LVI during Active mode, but does not need it during Sleep mode, it can be disabled to save power. Use the following API functions to disable this feature:

```
CyVdLvDigitDisable(); /* Digital LVI */
CyVdLvAnalogDisable(); /* Analog LVI */
```

To enable it again after wakeup, use:

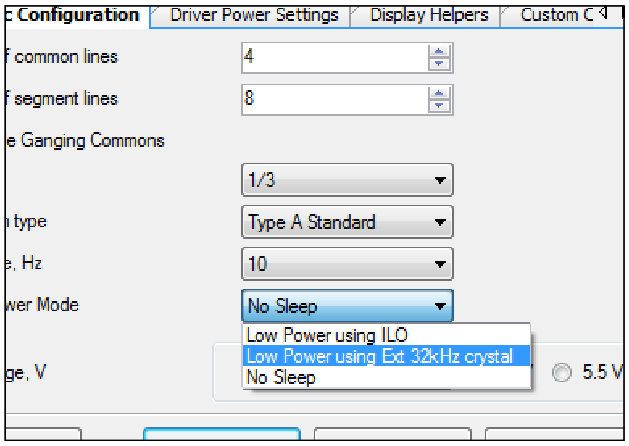
```
CyVdLvDigitEnable(<Reset>,<Threshold>);
CyVdLvAnalogEnable(<Reset>,<Threshold>);
```

The LVI subsystem is disabled by default. The *System Reference Guide* has more information about these functions.

SegLCD in PSoC 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 is in Sleep, as Figure 14 shows.

Figure 14. SegLCD Configuration Wizard



f common lines	4
f segment lines	8
e Ganging Commons	
	1/3
ı type	Type A Standard
ε, Hz	10
wer Mode	No Sleep
ge, V	

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 and PSoC 5LP Segment LCD Direct Drive for more information*.

Boost in PSoC 3 Sleep Mode

The PSoC 3 boost regulator can be used with all PSoC 3 power modes. The boost has two modes of operation:

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

It is not recommended to use chip Hibernate with the boost. Any power savings from using Hibernate, instead of Sleep, are small because the boost draws more power than the rest of the chip.

The PSoC cannot operate for more than a few microseconds (depending on the configuration) in Active or AltAct modes when Boost is in Standby mode. Change the boost mode to Standby immediately before putting the PSoC 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 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 subsystem are available in the PSoC 3 datasheets and TRM. More information on the boost component is available in the Boost component datasheet and *System Reference Guide*.

Boost in PSoC 5LP Sleep Mode

The PSoC 5LP boost regulator can be used with all PSoC 5LP power modes. The boost has two modes of operation:

- Boost Active mode is used in chip Active, AltAct, and Sleep. In this mode, the Boost monitors the output voltage and can support full PSoC functionality.
- Boost Sleep mode is a low-power state that disables all boost functionality. You must wake the PSoC and set boost to Active mode to refresh the regulator power supply.

It is not recommended to use chip Hibernate with the boost subsystem. Any power savings from using Hibernate, instead of Sleep, are small because the boost draws more power than the rest of the chip.

The PSoC cannot operate for more than a few microseconds (depending on the configuration) in Active or AltAct modes when the boost is in Sleep mode and is powering the PSoC. Change the boost mode to Sleep immediately before putting the PSoC to sleep. You must periodically wake the PSoC and set the boost to Active mode to ensure that the PSoC 5LP internal regulators remain charged.

```
/* Set system clocks for low-power. */
CyPmSaveClocks();
/* Set boost to Standby mode. */
BoostConv_SetMode
   (BoostConv_BOOSTMODE_SLEEP);
```

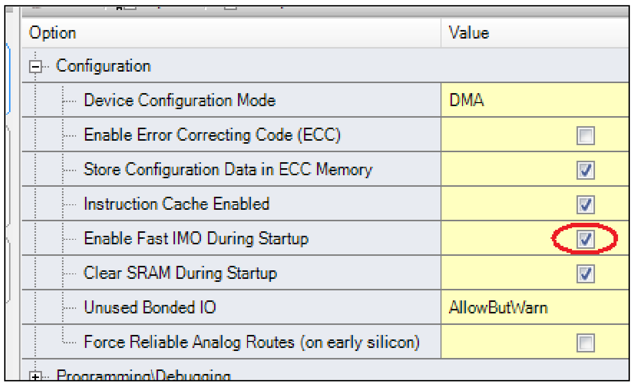
```
/* Sleep until boost refresh is needed. */
CyPmSleep
   (PM_SLEEP_TIME_NONE, PM_SLEEP_SRC_CTW);
/* Restore boost to Active mode. */
BoostConv_SetMode
   (BoostConv_BOOSTMODE_ACTIVE);
/* Restore system clocks. */
CyPmRestoreClocks();
```

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

Fast IMO Startup

PSoC 3 and PSoC 5LP have 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 Figure 15 shows.

Figure 15. Fast IMO Selection in the .cydwr Tab



Option	Value
Configuration	
Device Configuration Mode	DMA
Enable Error Correcting Code (ECC)	☐
Store Configuration Data in ECC Memory	✓
Instruction Cache Enabled	✓
Enable Fast IMO During Startup	✓
Clear SRAM During Startup	✓
Unused Bonded IO	AllowButWarn
Force Reliable Analog Routes (on early silicon)	☐
Programming\Debugging	

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 5LP TRMs* 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 modes. 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 5LP 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);
```

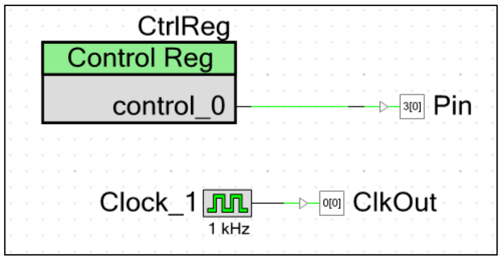
The flexibility of the PSoC 3 and PSoC 5LP makes it easy to manage GPIO drive modes to prevent unwanted current leakage. See *AN72382 - Using PSoC 3 and PSoC 5LP GPIO Pins* for more information.

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.

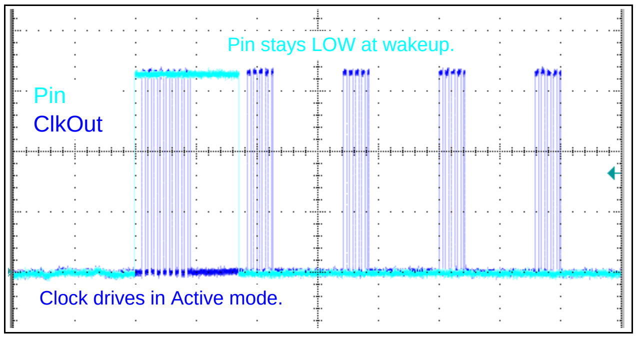
Figure 16 shows a Control Register that is used to set the output of a GPIO pin (the other pin and clock are used to indicate that the PSoC is awake).

Figure 16. Control Register used to Set Pin Output



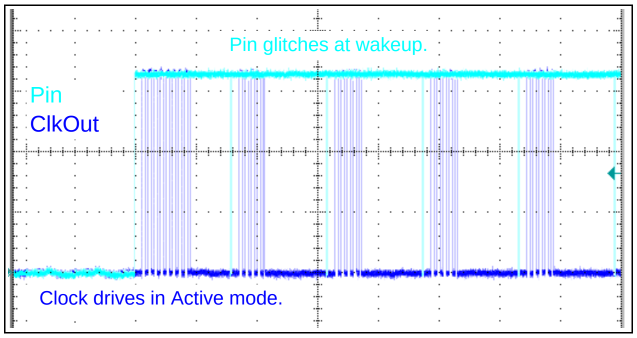
The pin continues to drive its last state (HIGH) while the chip is in low-power mode, but the physical block containing the Control Register is unpowered. When the PSoC wakes and power to the digital block is restored, the Control Register bit is reset to ‘0’. The pin connected to the Control Register then switches from a HIGH to LOW output, as Figure 17 shows.

Figure 17. 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 Figure 18 shows.

Figure 18. Control Register Glitch at Each Wakeup



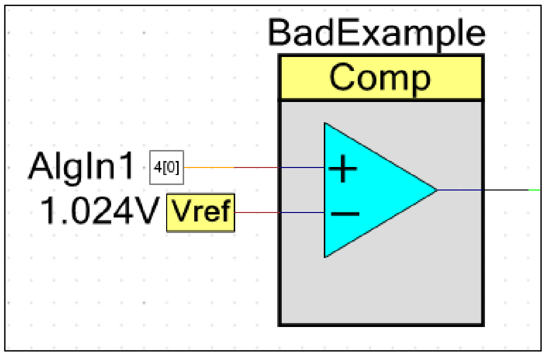
In this case, it is better to control the pin directly, instead of with a Control Register.

Vref Sources in Low Power

In PSoC, 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.

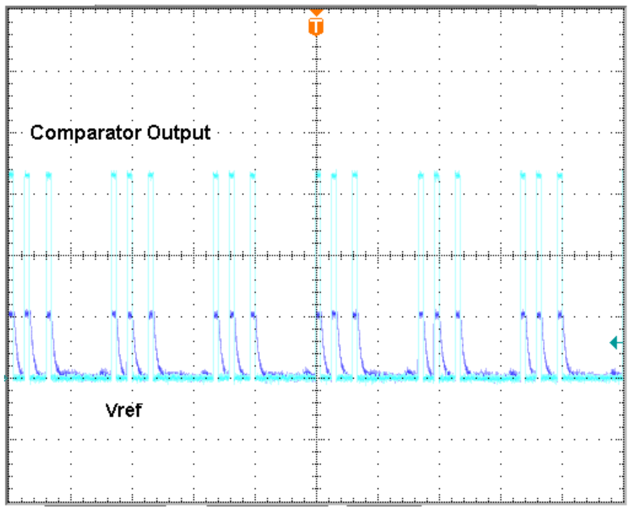
Figure 19 shows an example of this situation. The comparator is configured as a wakeup source, and it has a Vref component connected to the negative terminal.

Figure 19. How Not to Use Vref with Low Power



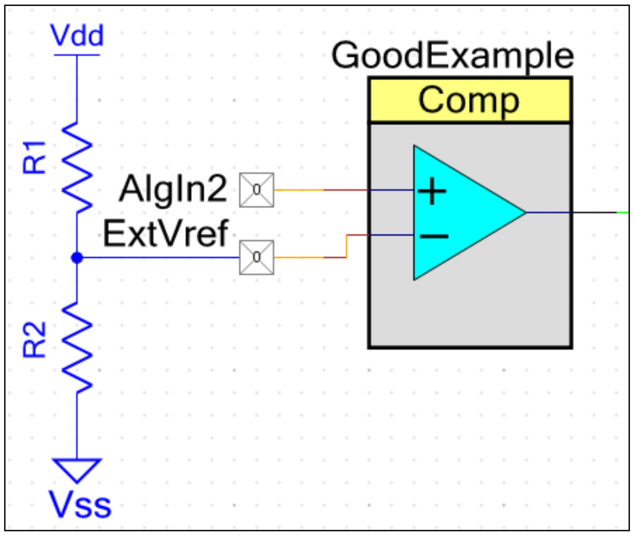
This is fine as long as the PSoC is in Active or AltAct, but the Vref is disconnected in low-power mode. The result is intermittent wakeups when the negative terminal floats, as Figure 20 shows.

Figure 20. Intermittent Wakeup as Comparator Input Floats



The solution to this issue is to supply an external reference voltage, as Figure 21 on page 12 shows. This reference remains active while the PSoC is in Sleep mode, so the Comparator wakeup source functions as wanted.

Figure 21. 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

PSoC 3 and PSoC 5LP 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 the sleep regulator can be seen on the Vccd and Vcca pins, but does not provide enough current to power anything but the internal PSoC resources. You should not attempt to use the sleep regulators for anything else.

The hibernate regulator does not power the Vccd and Vcca nets, so its output cannot be observed.

Power Consumption While Programming

PSoC 3 and PSoC 5LP 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.

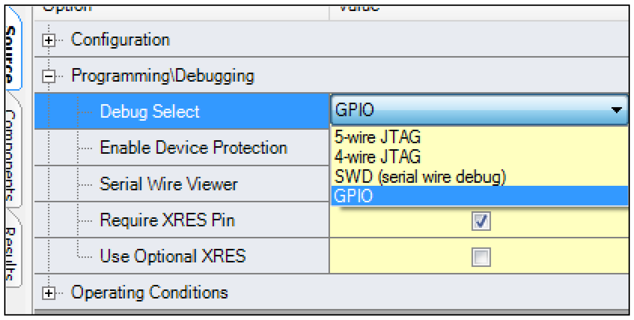
Is Your Debug Interface Running?

PSoC 3 and PSoC 5LP support 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.

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 Figure 22 shows.

Figure 22. Disable Debug Interface to Reduce Power



Option	Value
Configuration	
Programming\Debugging	
Debug Select	GPIO
Enable Device Protection	5-wire JTAG
Serial Wire Viewer	4-wire JTAG / SWD (serial wire debug) / GPIO
Require XRES Pin	✓
Use Optional XRES	☐
Operating Conditions	

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 5LP, you can optimize your design and ensure that it consumes the least amount of power.

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

Related Application Notes

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

- AN86233 - PSoC® 4 Low-Power Modes and Power Reduction Techniques
- AN54181 - Getting Started with PSoC 3
- AN77759 - Getting Started with PSoC 5LP
- AN77835 – PSoC 3 to PSoC 5LP Migration Guide
- AN84741 – PSoC 5 to PSoC 5LP Migration Guide
- AN61290 - PSoC 3 and PSoC 5LP Hardware Design Considerations
- AN54460 - PSoC 3 and PSoC 5LP Interrupts
- AN60616 - PSoC 3 Startup Procedure
- AN60631 - PSoC 3 Clocking Resources
- AN72382 - Using PSoC 3 and PSoC 5LP GPIO Pins
- AN60580 - SIO Tips and Tricks in PSoC 3 / PSoC 5LP
- AN52705 - PSoC 3 and PSoC 5LP - Getting Started with DMA
- AN52927 - PSoC 3: Segment LCD Direct Drive

About the Author

Name:	Max Kingsbury
Title:	Applications Engineer Senior
Background:	Max earned a bachelor's degree in electrical engineering from Washington State University. He enjoys crafting high quality technical documentation and debugging embedded designs.
Contact:	Maxk@Cypress.com

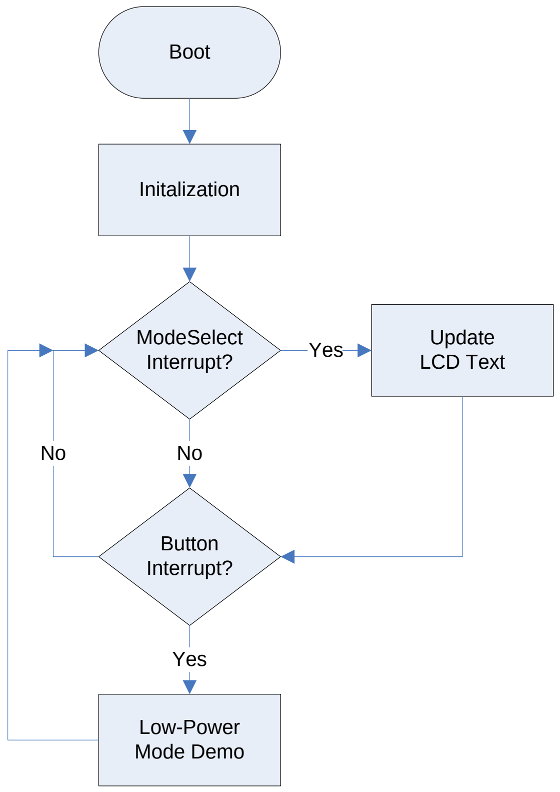
Appendix A – Low-Power Example Projects

Using PSoC is one of the best ways to become familiar with it. 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.

Two Low-Power Demo Projects

These projects demonstrate PSoC 3 and PSoC 5LP power mode transitions and wakeup sources. Select the power mode and wakeup source 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, as Figure 23 shows.

Figure 23. PSoC Low-power Demo Flowchart



The projects demonstrate all of the common Sleep and Hibernate wakeup sources. SegLCD has its own example projects in PSoC Creator, and Boost during 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 these example projects.

Table 2. PSoC Demo Wakeup Sources

Mode	Wakeup Source
Active	None
AltAct	PICU
	RTC
Sleep	PICU
	RTC (Uses the OPPS)
	SleepTimer (Uses the CTW)
	CTW (PSoC 3 Only)
	Comparator
	I2C Address
	LCD (Not implemented)
Hibernate	PICU
Custom	None (User Defined)

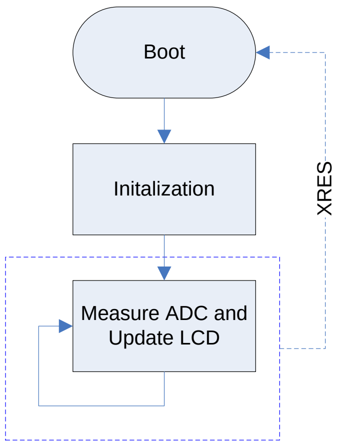
This code demonstrates the principles and techniques described in this application note. No effort has been made to optimize it for size or speed. Instructions for using the projects are found in the project's schematics and source files.

Voltage Alarm – No Optimization

This project demonstrates a simple voltage measurement and alarm system using a PSoC 3, as Figure 24 shows. The project 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 24. Alarm System – No Optimizations

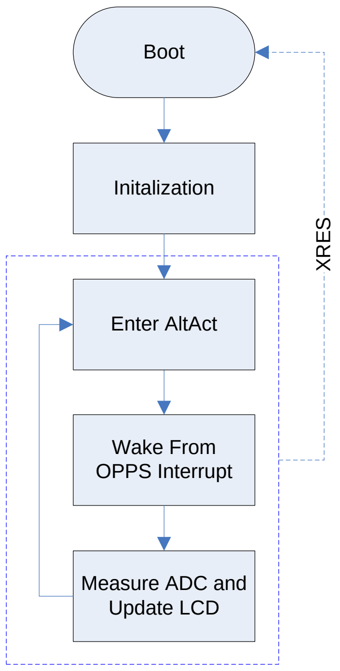


Instructions for using the project are found in the project's schematic and source files.

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, as Figure 25 shows.

Figure 25. 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 only by the PSoC device. 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. Schematics are available on the individual kit webpages.

CY8CKIT-001 Modifications

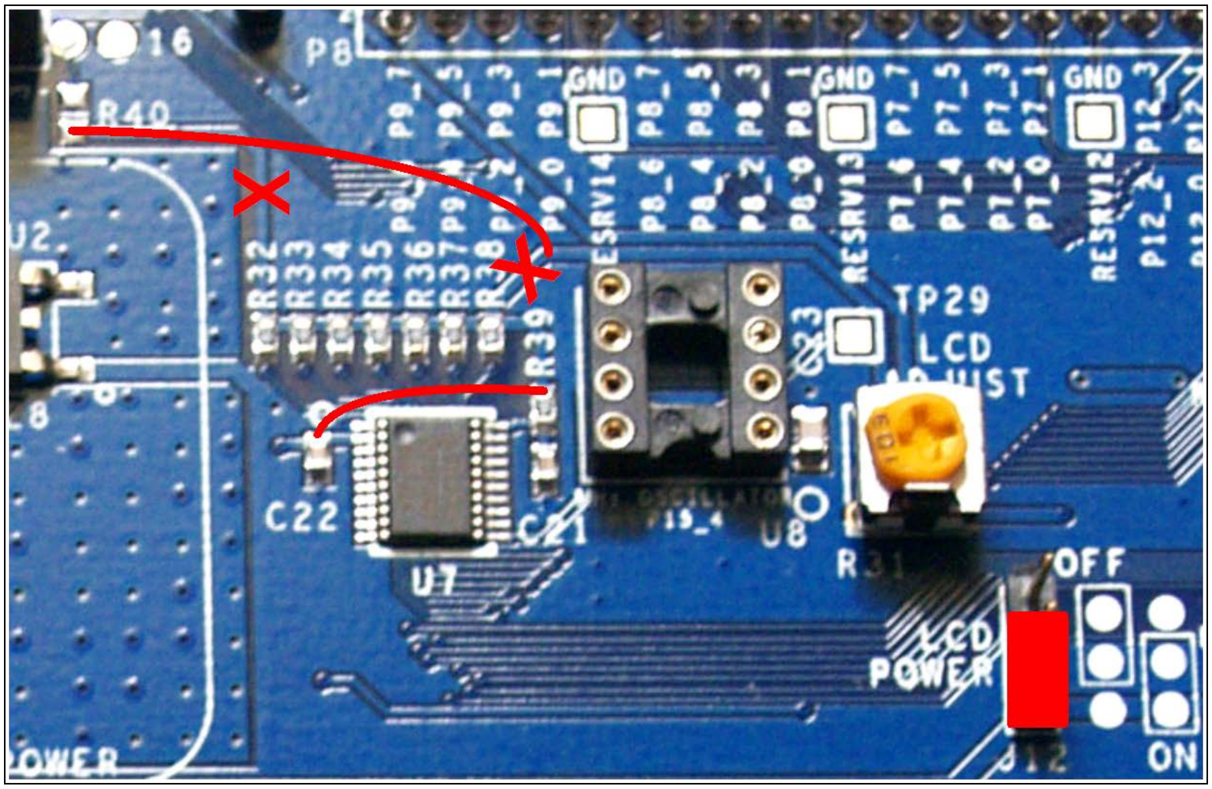
The following steps demonstrate how to modify the CY8CKIT-001 DVK board to ensure proper current measurements. The configuration described here uses a single power rail for Vddd, Vdda, and Vddio. Power is measured by placing a meter in series with the PSoC's Vddd pins.

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 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 26 on page 17 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 26. 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 KIT-001 board are powered separately from Vddd, so they are not seen in your power measurements. The KIT-001 can be configured for 5-V or 3.3-V operation, without any further changes, by using SW3.

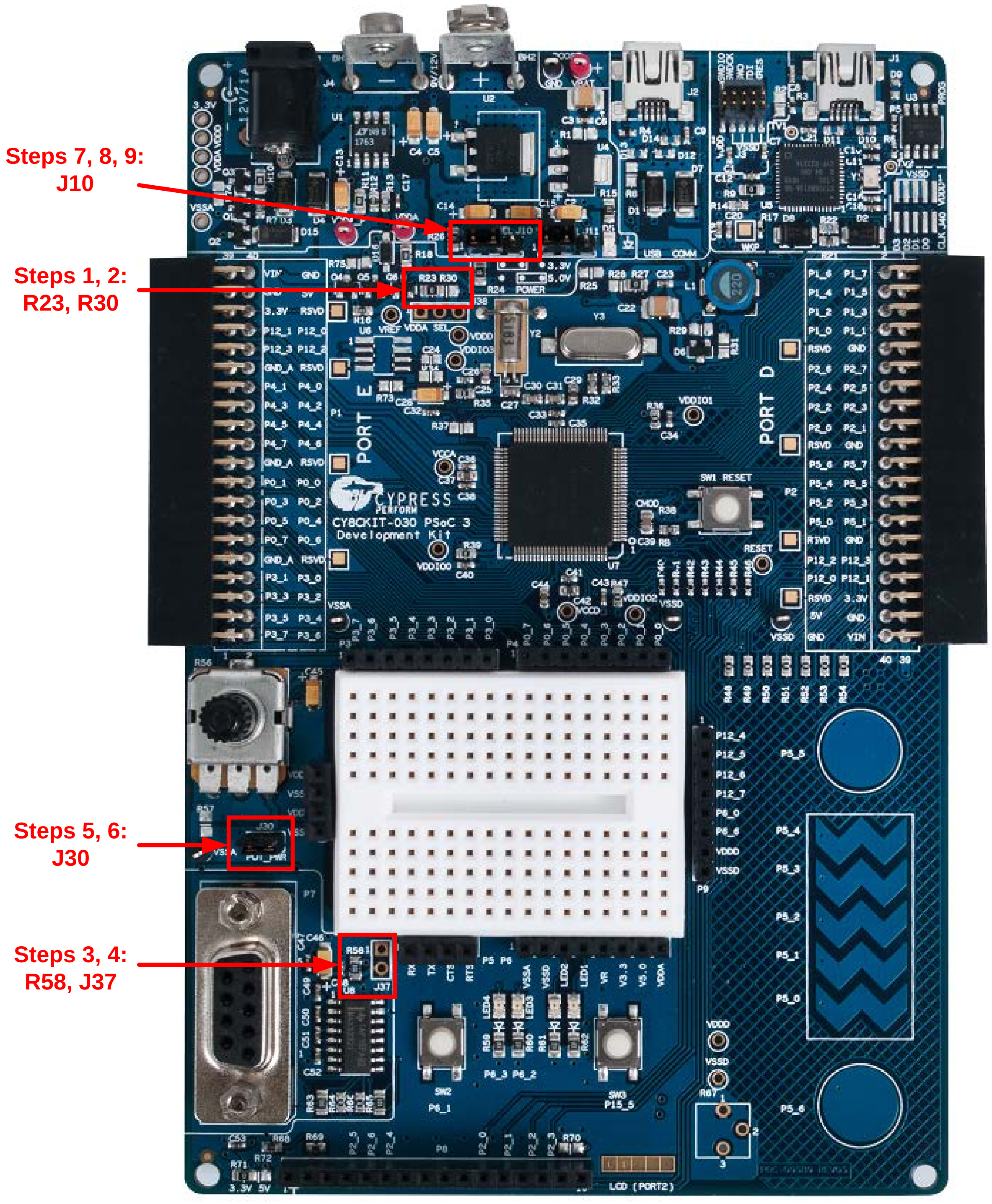
CY8CKIT-030 and CY8CKIT-050 Modifications

The following steps demonstrate how to modify the CY8CKIT-030 and CY8CKIT-050 DVK boards to ensure proper current measurements. Called out hardware is identified in Figure 26. 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 shows 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 does not show up in your power measurements.
5. Remove the jumper from J30. By default, R56 is powered by VDDA, so it shows up in your power measurement.
6. If potentiometer R56 is needed, short pin 2 of J30 (below the “J30” text) to the VDDA_P test point. This powers the potentiometer 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 are not 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.

Figure 27. KIT-030/050 Low Power Modifications



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 5LP 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 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 do not check for component idle or busy status. You should add a check in your code to ensure that active components are not busy. 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 power modes. Bits [2:0] in this register control the global power mode for the entire PSoC, as Table 3 on page 22 shows. Reading these bits tells you what mode the PSoC is currently in.

Table 3. 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 5LP subsystems are available in the *PSoC 3 and PSoC 5LP 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 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

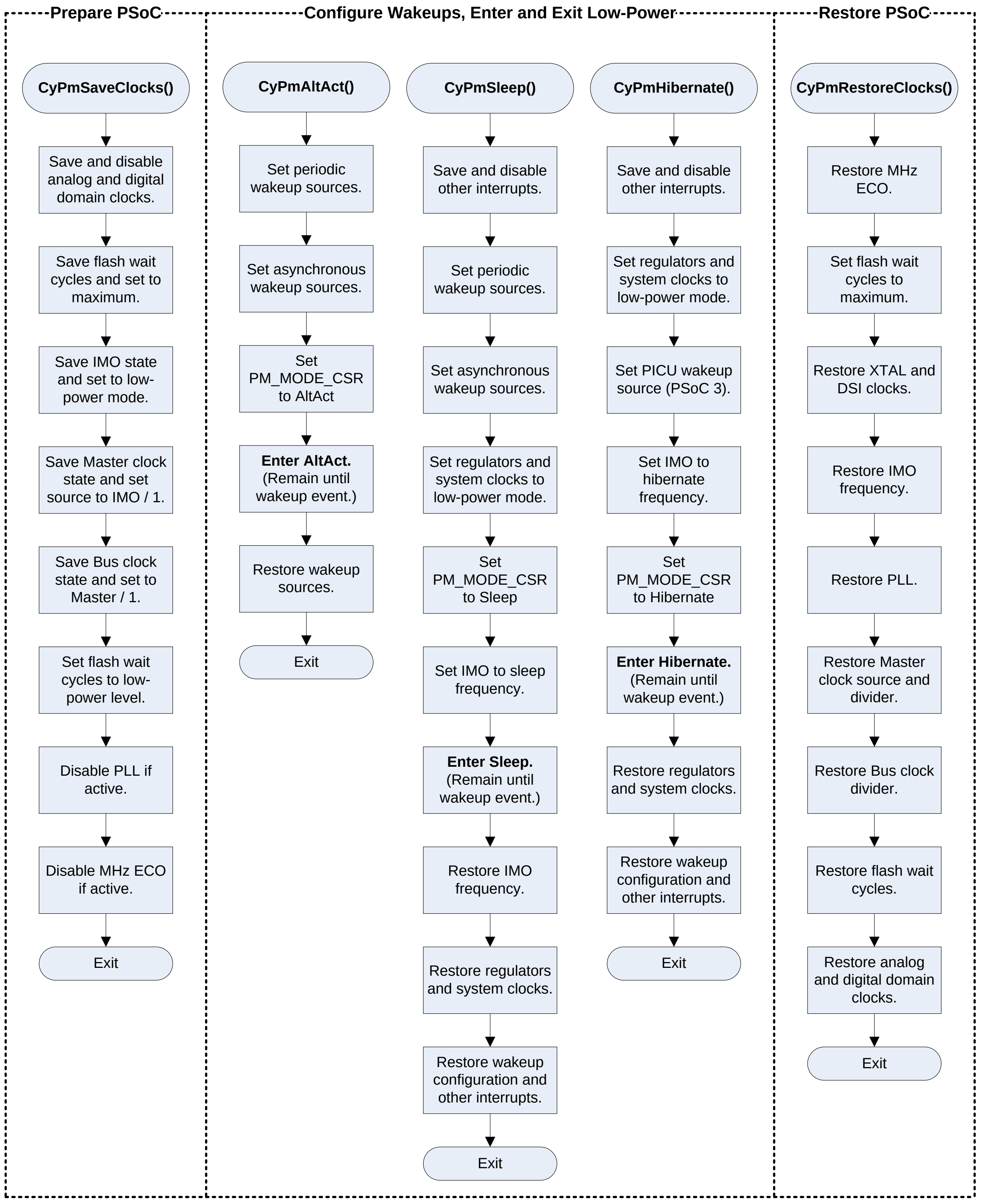
These registers configure the IMO, MHz crystal, and PLL. Clock distribution is controlled by the *CLKDIST* registers.

SLOWCLK

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

Power Management API Flowcharts

Figure 28. Power Management Function Flowcharts


Prepare PSoC
Configure Wakeups, Enter and Exit Low-Power
Restore PSoC
CyPmSaveClocks()
Save and disable analog and digital domain clocks.
Save flash wait cycles and set to maximum.
Save IMO state and set to low-power mode.
Save Master clock state and set source to IMO / 1.
Save Bus clock state and set to Master / 1.
Set flash wait cycles to low-power level.
Disable PLL if active.
Disable MHz ECO if active.
Exit
CyPmAltAct()
Set periodic wakeup sources.
Set asynchronous wakeup sources.
Set PM_MODE_CSR to AltAct
Enter AltAct. (Remain until wakeup event.)
Restore wakeup sources.
Exit
CyPmSleep()
Save and disable other interrupts.
Set periodic wakeup sources.
Set asynchronous wakeup sources.
Set regulators and system clocks to low-power mode.
Set PM_MODE_CSR to Sleep
Set IMO to sleep frequency.
Enter Sleep. (Remain until wakeup event.)
Restore IMO frequency.
Restore regulators and system clocks.
Restore wakeup configuration and other interrupts.
Exit
CyPmHibernate()
Save and disable other interrupts.
Set regulators and system clocks to low-power mode.
Set PICU wakeup source (PSoC 3).
Set IMO to hibernate frequency.
Set PM_MODE_CSR to Hibernate
Enter Hibernate. (Remain until wakeup event.)
Restore regulators and system clocks.
Restore wakeup configuration and other interrupts.
Exit
CyPmRestoreClocks()
Restore MHz ECO.
Set flash wait cycles to maximum.
Restore XTAL and DSI clocks.
Restore IMO frequency.
Restore PLL.
Restore Master clock source and divider.
Restore Bus clock divider.
Restore flash wait cycles.
Restore analog and digital domain clocks.
Exit

Power Management API Register Reference

Table 4 lists the asynchronous wakeup sources available in the PSoC 3 and PSoC 5LP devices.

Table 4. CyPm Wakeup Source Configuration

API	Wakeup Source	Defined wakeupSource Mask	Affected Register [bits]
CyPmAltAct()	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()	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()	PICU	N/A	PM_WAKEUP_CFG0 [0x04]

Table 5 lists the periodic wakeup sources available in PSoC 3 devices.

Table 5. CyPm Wakeup Period Configuration

API	Wakeup Source	Defined wakeupTime Mask	Affected Register [bits]
CyPmAltAct() and CyPmSleep()[1]	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 5LP only supports the CTW and OPPS wakeup sources as part of a SleepTimer or RTC component.			

Document History

Document Title: AN77900 - PSoC® 3 and PSoC 5LP 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.
*A	3845784	GIR	12/18/2012	Updated to support PSoC 5LP and PSoC Creator 2.1 SP1. Corrected links to the segment LCD application note. Other minor changes.
*B	4034418	MAXK	06/19/2013	Updated author info. Added picture for kit modifications. Project updated. Minor typo fixes.

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 5LP

Cypress Developer Community

Community | Forums | Blogs | Video | Training

Technical Support

cypress.com/go/support

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

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