

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

Author: Rodolfo Lossio

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

Associated Code Examples: Yes

Software Version: PSoC Creator™ 3.3 or higher

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 APIs and registers, power-saving techniques, and other low-power mode considerations. The associated PSoC Creator project demonstrates these principles.

Contents

1	Introduction.....	2	3.13 Sleep and Hibernate Regulators.....	17
1.1	Power Modes and Transitions.....	2	3.14 Power Consumption While Programming	17
1.2	Wakeup Sources.....	3	3.15 Is the Debug Interface Running?	18
2	Active Mode Power Reduction.....	4	4 Approximating Power Consumption	18
2.1	Power Off Unused Components	4	5 Summary	18
2.2	Use the ADC at a Slower Sample Rate	5	6 Related Application Notes	19
2.3	Use PSoC to Gate Current Paths	5	A Low-Power Example Projects.....	20
2.4	Dynamically Change Clock Speeds	6	A.1 Two Low-Power Demo Projects.....	20
2.5	Use DMA to Move Data	9	A.2 Voltage Alarm – No Optimization	21
2.6	Using Alternate Active mode.....	9	A.3 Voltage Alarm – With Optimizations.....	22
2.7	Use Power Mode Configuration Registers	9	B Power Measurements on DVKs	23
3	Other Power Mode Considerations.....	9	B.1 CY8CKIT-001 Modifications.....	23
3.1	Faster Clocks Can Mean Lower Power.....	9	B.2 CY8CKIT-030 and CY8CKIT-050 Modifications.....	24
3.2	32-kHz Crystal Low-Power Mode.....	11	C Power Management API and Registers	26
3.3	Low-Voltage Interrupts in PSoC Sleep Mode	12	C.1 CyPmSaveClocks().....	26
3.4	SegLCD in PSoC Sleep Mode	12	C.2 CyPmRestoreClocks()	26
3.5	Boost Converter in PSoC 3 Sleep Mode.....	12	C.3 CyPmAltAct()	26
3.6	Boost Converter in PSoC 5LP Sleep Mode.....	13	C.4 CyPmSleep()	26
3.7	Fast IMO Startup.....	13	C.5 CyPmHibernate() or CyPmHibernateEx()	26
3.8	Watchdog in PSoC Sleep Mode.....	14	C.6 Component Low-Power API.....	26
3.9	GPIOs in PSoC Low-Power Mode	14	C.7 Direct Register Writes	27
3.10	SIOs in PSoC Low-Power Mode	15	C.8 Power Management API Flow Charts	28
3.11	Digital Blocks in PSoC Low-Power Mode.....	15	C.9 Power Management API Register Reference.....	29
3.12	VREF Sources in PSoC Low-Power Mode	16		

1 Introduction

The PSoC 3 and PSoC 5LP low-power modes allow you to reduce overall current draw without limiting functionality, especially when they are 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 the PSoC 3 and PSoC 5LP device architecture and PSoC Creator.](#)

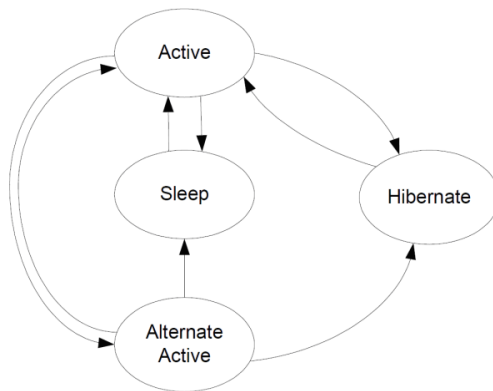
1.1 Power Modes and Transitions

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. It provides a method in which the device can switch to a subset of active analog/digital blocks quickly without having to turn them off individually then each one back on individually when returning to normal or Active mode. By default, the CPU is disabled.
- Sleep mode disables most subsystems to reduce average current consumption (~1 μ A for PSoC 3 and ~2 μ A for PSoC 5LP). Wakeup time is 15- μ s max for PSoC 3 and 25- μ s 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- μ s max for PSoC 3 and 125- μ s 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 these power mode transition processes.

1.1.1 Active Mode

Any valid wakeup or reset event returns the PSoC device 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 device to enter a low-power mode and update the register that controls the global power mode setting. See [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 that you do so.

1.1.2 AltAct Mode

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

PSoC automatically returns to Active mode when an unmasked wakeup source causes an interrupt. See [Table 1](#) for the list of wakeup sources available. You can also exit AltAct mode by transitioning to Sleep or Hibernate if the CPU has not been disabled in AltAct mode. Refer to the PM_STBY_CFG0 register in the [PSoC 3 Registers Technical Reference Manual \(TRM\)](#) to enable the CPU during AltAct mode.

1.1.3 Sleep Mode

The typical way to enter Sleep mode is to call the API function `CyPmSleep()`. If any interrupts are pending, the PSoC device 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 3 for the list of wakeup sources available.

1.1.4 Hibernate Mode

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

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

1.2 Wakeup Sources

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

- Periodic wakeup sources include the central time wheel (CTW), one pulse per second (OPPS), and LCD timers. The Real-Time Clock (RTC) and Sleep Timer components use these timers.
- Asynchronous wakeup sources include the Boost Converter, Comparator, I²C, LVI, and PICU.
- Reset wakeup sources include the external reset (XRES) pin and watchdog timer (WDT).

The low-power modes support some or all of these wakeup sources. [Table 1](#) lists the wakeup sources available for each power mode.

1.2.1 Multiple Wakeup Sources

PSoC applications can use multiple wakeup sources. For example, the PSoC device may need to wake periodically to check the battery status (OPPS), when a button is pressed (PICU), or if the external 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 [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	AltAct	Sleep	Hibernate	
Interrupt	✓			✓			Interrupt Component must be used.
CTW	✓	✓					Sleep time is configurable.
SleepTimer	✓	✓		✓	✓		Sleep Timer 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	✓	✓ ¹		✓	✓ ¹		Period depends on settings.
Boost Converter	✓	✓ ²	³	✓	✓	³	

Wakeup Source	PSoC 3			PSoC 5LP			Notes
	AltAct	Sleep	Hibernate	AltAct	Sleep	Hibernate	
WDT	✓	✓		✓	✓		WDT issues a reset if not fed.
LVI	✓	✓		✓	✓		
XRES	✓	✓	✓	✓	✓	✓	Device wakes and resets.

1. The LCD low-power feature cannot be used in conjunction with the `CyPmSaveClocks()` function.
2. In PSoC 3 Sleep mode, the Boost Converter can be used in its Active or Sleep mode. Sleep mode is recommended.
3. Hibernate is not recommended for applications requiring a boost. Use Sleep mode instead.

2 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.

2.1 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; therefore, 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 the Stop function. If code execution time is important, look at the generated source code to see if you can call the Stop function instead.

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

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

/* <Do something else here.> */

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

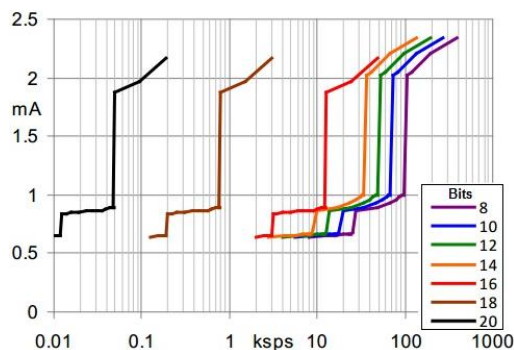
When a component is put to sleep, it should be awakened again by calling its Wakeup function. This restores the component to its pre-sleep state. The Start function also brings the component back into operation, but it is reinitialized to its default state.

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.

2.2 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 Versus 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 1 mA.

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.

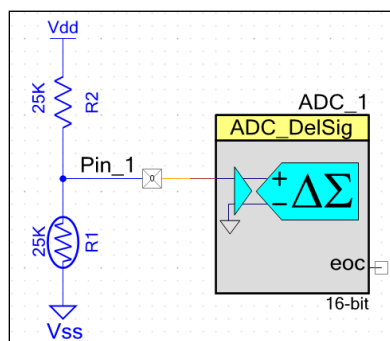
Refer to the [Delta-Sigma ADC component datasheet](#), [PSoC 3](#) or [PSoC 5LP](#) Architecture TRMs for more information on selecting the proper sample rate and resolution.

2.3 Use PSoC to Gate Current Paths

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

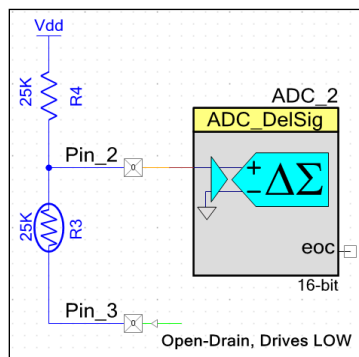
A good example of this scenario is a thermistor application, as [Figure 3](#) shows. In this case, the PSoC device measures temperature using the voltage on an analog pin, which changes as the thermistor resistance changes.

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



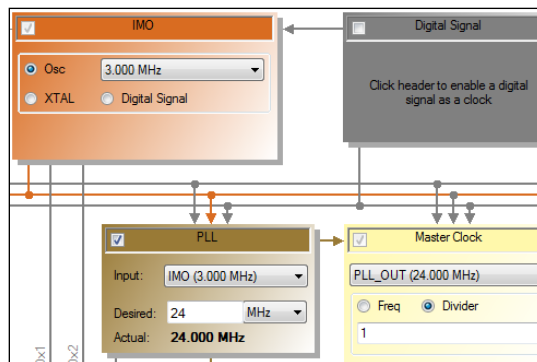
In this configuration, current flow can be stopped by writing a '1' to Pin_3. Writing a '0' resumes current flow. The added cost of this power-saving feature is only one pin and a few lines of code.

2.4 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 internal main oscillator (IMO) feeding a 24-MHz PLL, as Figure 5 shows.

Figure 5. 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 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.

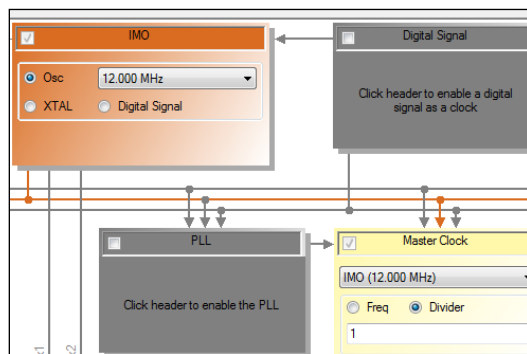
Figure 6. Warning– Clocks Too Slow for ADC Settings

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_DeSig_1_theACLK	2.590 MHz	3.000 MHz	Auto: MASTER_CLK
ADC_DeSig_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 simply 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](#) 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 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:

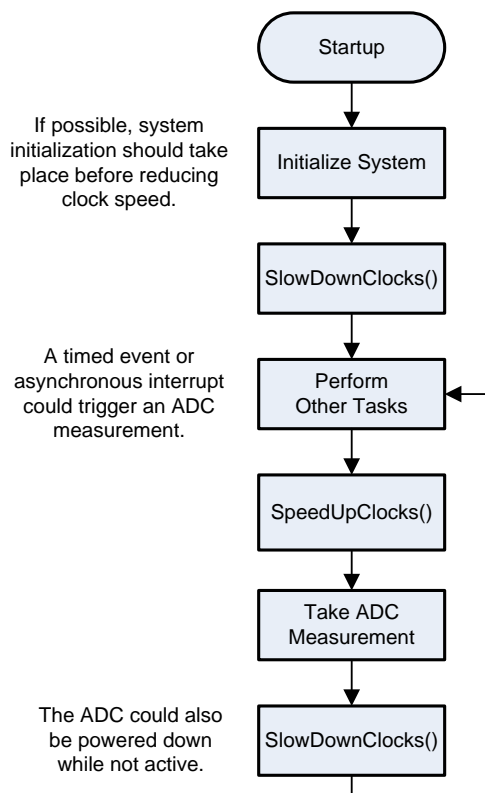
```

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 increased to 12 MHz only when the ADC needs to take a sample, as [Figure 8](#) shows.

Figure 8. Flow Chart for Clock Changing Example



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

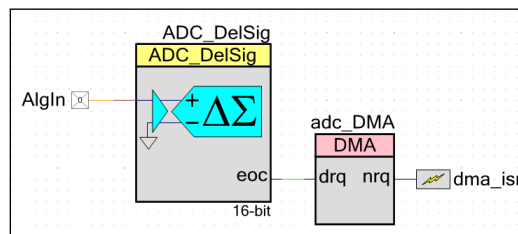
2.5 Use DMA to Move Data

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 mode 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.

DMA use is a complex topic that is too broad to cover in this document. You can find more information on www.cypress.com, including example projects and application notes, such as [AN52705 - PSoC® 3 and PSoC 5LP - Getting Started with DMA](#).

Figure 9. DMA Triggered by ADC Completion



2.6 Using Alternate Active mode

Alternative active mode is similar to active mode in most of its functionality. Alternative active mode also has its own additional set of subsystem template bits which determine whether a subsystem is enabled or disabled. This mode is made available for quick transitions between active and an alternate low-power mode with a subset of the devices operating including the CPU.

For example, you can write to the template bits to disable CPU and enable certain peripherals to operate in alternate active mode. While in alternate active mode, if any interrupt is generated, the device automatically transitions to active mode and begins executing the firmware in active mode. This is much more efficient and faster than powering individual blocks on and off.

2.7 Use Power Mode Configuration Registers

In Active or AltAct mode, you can control the power to most subsystems with the power mode configuration registers. The 14 PM_ACT_CFGx registers gate power and clocks in Active mode. The 14 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 in any of the CY_PM_ACT_CFGx registers.

The new settings take effect immediately but they do not persist 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](#) Registers TRMs.

3 Other Power Mode Considerations

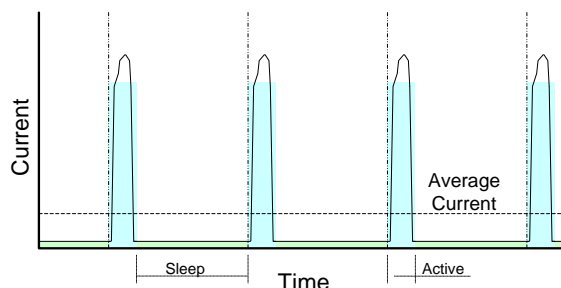
This section discusses a variety of tips, tricks, and recommendations related to the use of PSoC 3 and PSoC 5LP low-power modes.

3.1 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.

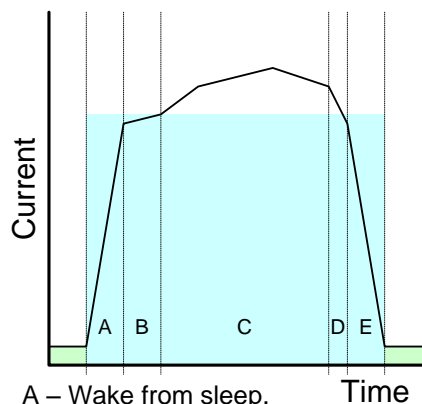
Sleep can be used to reduce power use when the PSoC device is idle, but the average current is higher because of the time spent in Active mode. Figure 10 is a representation of the current consumption of this example with the system clocks set at 3 MHz.

Figure 10. Example Current Profile with 3-MHz Clocks



If you look at the tasks that are being performed when the PSoC device 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 device is in Active mode for less time. Figure 11 is a representation of Active mode timing, broken up into tasks.

Figure 11. Analysis of Tasks in Active Mode at 3 MHz

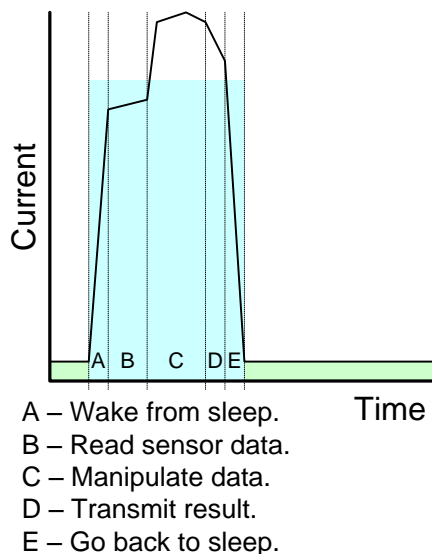


- A – Wake from sleep.
- B – Read sensor data.
- C – Manipulate data.
- D – Transmit result.
- E – Go back to sleep.

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.

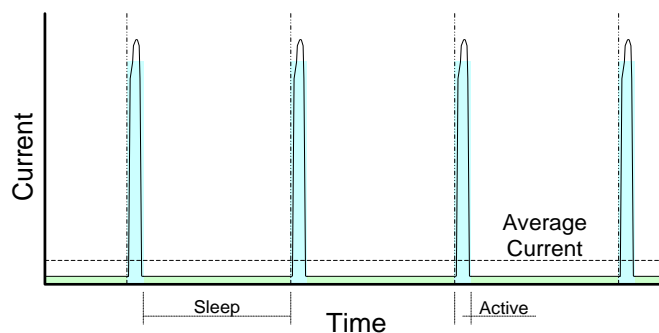
At some point, the benefit of a shorter Active time is overcome by the energy required to drive the clocks at a higher rate. Assume that the optimal speed is 12 MHz, as Figure 12 shows.

Figure 12. Analysis of Tasks in Active Mode at 12 MHz



The time spent in Active mode 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 consumption 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 included in this application note.

For more information on PSoC clocking, refer to application note [AN60631 - PSoC 3 and PSoC 5LP Clocking Resources](#).

3.2 32-kHz Crystal Low-Power Mode

The 32-kHz crystal can be configured to operate in a reduced power mode when the PSoC device 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 mode. The reduced power mode is ~1 μ A less than normal power mode.

The [System Reference Guide](#) offers more information about this function.

3.3 Low-Voltage Interrupts in PSoC Sleep Mode

The low-voltage interrupt (LVI) subsystem can be used to wake the PSoC device from Sleep mode, but it consumes ~1 μ A 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:

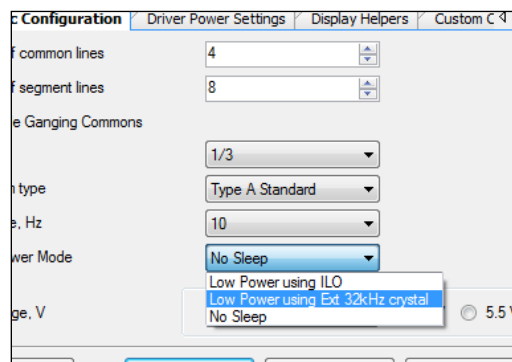
```
CyVdLvDigitEnable(<Reset>, <Threshold>);
CyVdLvAnalogEnable(<Reset>, <Threshold>);
```

The LVI subsystem is disabled by default. The [System Reference Guide](#) offers more information about these functions.

3.4 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 device is in Sleep mode, as [Figure 14](#) shows.

Figure 14. 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 Active and Sleep modes. 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.

3.5 Boost Converter in PSoC 3 Sleep Mode

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

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

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

The PSoC device cannot operate for more than a few microseconds (depending on the configuration) in Active or AltAct modes when the boost converter is in Boost Standby mode. Change the Boost mode to Standby immediately before putting the PSoC device to sleep. You should also set the Boost mode to Active 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 converter in Sleep mode. 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 Converter Component is available in the [Boost Converter component datasheet](#) and [System Reference Guide](#).

3.6 Boost Converter in PSoC 5LP Sleep Mode

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

- Boost Active mode is used in chip Active, AltAct, and Sleep modes. In this mode, the boost converter monitors the output voltage and can support full PSoC functionality.
- Boost Sleep mode is a low-power state that disables all boost converter functionality. You must wake the PSoC device and set the boost converter 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 mode are small because the boost converter draws more power than the rest of the chip.

The PSoC device cannot operate for more than a few microseconds (depending on the configuration) in Active or AltAct modes when the boost converter is in Sleep mode and is powering the PSoC device. Change the Boost mode to Sleep immediately before putting the PSoC device to sleep. You must periodically wake the PSoC device and set the boost converter 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 Converter Component is available in the [Boost Converter component datasheet](#) and [System Reference Guide](#).

3.7 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)	<input type="checkbox"/>
Store Configuration Data in ECC Memory	<input checked="" type="checkbox"/>
Instruction Cache Enabled	<input checked="" type="checkbox"/>
Enable Fast IMO During Startup	<input checked="" type="checkbox"/>
Clear SRAM During Startup	<input checked="" type="checkbox"/>
Unused Bonded IO	AllowButWarn
Force Reliable Analog Routes (on early silicon)	<input type="checkbox"/>
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](#) Architecture TRMs for more information about the FIMO feature and its impact on boot time.

3.8 Watchdog in PSoC Sleep Mode

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 device 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 (`CyWdtStart`) to ensure that the watchdog is configured as desired. The “Maximum Interval” option is set in the `PM_WDT_CFG` registers by default.

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

More information on the operation of the watchdog timer and the associated API is available in the [PSoC 3](#) and [PSoC 5LP](#) Architecture TRMs and the [System Reference Guide](#).

3.9 GPIOs in PSoC Low-Power Mode

The GPIOs can continue to drive when the PSoC device 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 My Pin 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 Pins Component can be set to Analog Hi-Z, then you can use the component’s `SetDriveMode` function.

```
/* Set My Pin 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 Pins Component, you can write to the port pin configuration registers. There is one register for each GPIO pin on the PSoC device named `CYREG_PRTx_PCy`, where “x” is the port number and “y” is the pin number.

The flexibility of 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.

3.10 SIOs in PSoC Low-Power Mode

Special input/output (SIO) pins should be put in single-ended mode before putting the PSoC device in Sleep mode to reduce PSoC sleep current. SIOs in differential mode consume a high current of 100 μ A. To put SIOs in single-ended mode, use the register PRT12_SIO_CFG and set the bits for the specific SIO pair to zero. After the PSoC device comes out of Sleep mode, the bits should be set back to their previous values. Table 2 shows the description of the SIO configuration register.

Table 2. PRT12_SIO_CFG Register

SIO[7:6]		SIO[5:4]		SIO[3:2]		SIO[1:0]	
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

See AN60580 – SIO Tips and Tricks in PSoC 3/PSoC 5LP for more information.

Also, if you are using SIO pins as outputs and you have the “Drive level” set to “Vref” instead of “Vddio” you may want to disable the Vref mode during low power modes to minimize the power. These are done in pairs so enable the Vref output mode in groups of two adjacent pins. See code example below that defines the mask for the pin pairs and how to enable or disable the Vref.

```
#define SIO_VREG_EN_1_0 0x01 // Mask for SIO pins[1:0]
#define SIO_VREG_EN_3_2 0x04 // Mask for SIO pins[3:2]
#define SIO_VREG_EN_5_4 0x10 // Mask for SIO pins[5:4]
#define SIO_VREG_EN_7_6 0x40 // Mask for SIO pins[7:6]

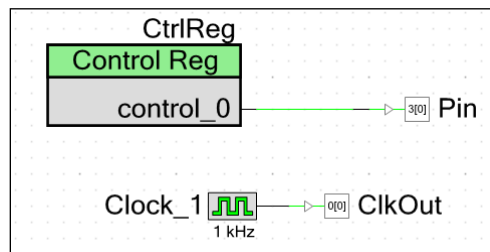
// Code to enable and disable Vref output mode (Instance name is "MySIO")
MySIO_SIO_CFG |= VREG_EN_3_2; // Set to use Vref reference
MySIO_SIO_CFG &= ~VREG_EN_3_2; // Clear Vref for Low Power mode
```

3.11 Digital Blocks in PSoC Low-Power Mode

Some subsystems are always powered off in Sleep and Hibernate modes. Connecting them to other subsystems that remain powered can lead to unwanted behavior.

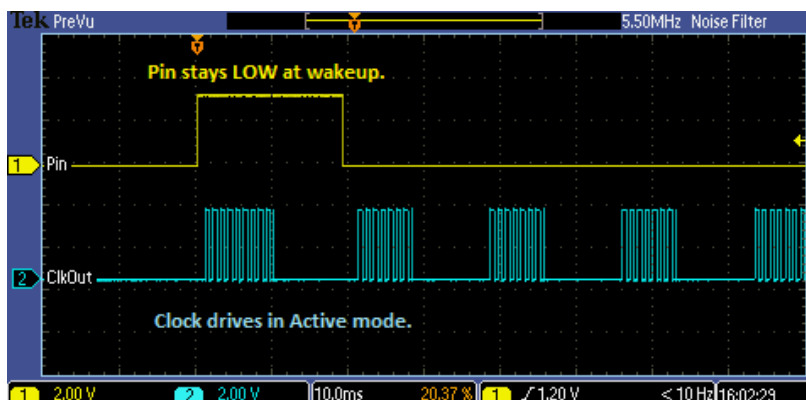
Figure 16 shows a Control Register that is used to set the output of a GPIO pin (the other pin and clock are used to indicate that the PSoC device is awake).

Figure 16. Control Register Used to Set Pin Output



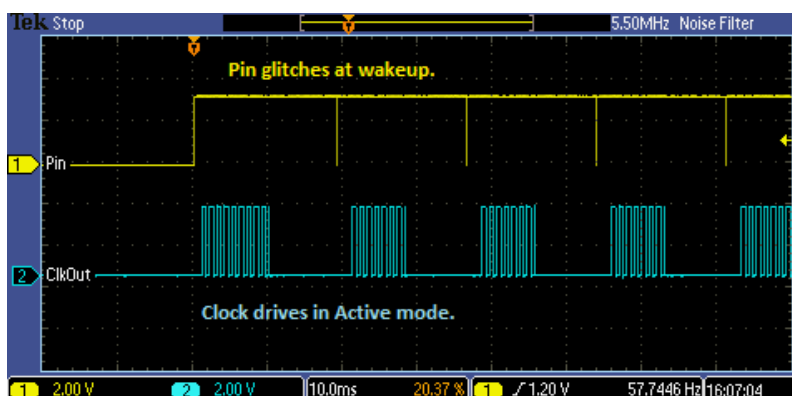
The pin continues to drive its last state (HIGH) while the chip is in low-power mode, but the physical block containing the Control Register is unpowered. When the PSoC device wakes and power to the digital block is restored, the Control Register bit is reset to '0'. The pin connected to the Control 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 the firmware can change it, as Figure 18 shows.

Figure 18. Control Register Glitch at Each Wakeup



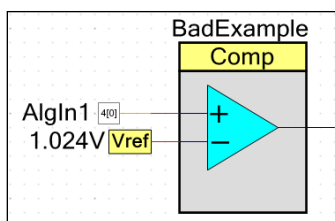
In this case, it is better to control the pin directly, instead of with a Control Register.

3.12 VREF Sources in PSoC Low-Power Mode

In PSoC, VDDA, VDDD, and VBAT remain connected during Sleep or Hibernate mode, 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 mode, you must use an external source.

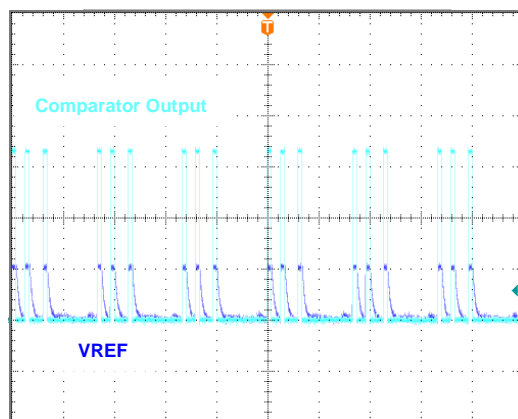
Figure 19 shows an example of this situation. The comparator is configured as a wakeup source and it has a VREF component connected to the negative terminal.

Figure 19. How Not to Use VREF with Low Power



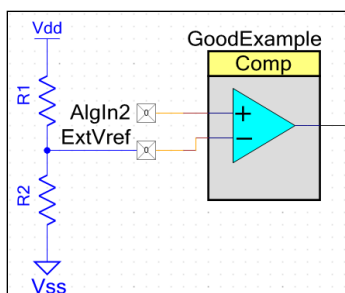
This is acceptable as long as the PSoC device is in Active or AltAct mode, but the VREF is disconnected in low-power mode. The result is intermittent wakeups when the negative terminal floats, as Figure 20 shows.

Figure 20. Intermittent Wakeup as Comparator Input Floats (Voltage scale = 1.0 V)



The solution to this issue is to supply an external reference voltage, as Figure 21 shows. This reference remains active while the PSoC device is in Sleep mode; therefore, the Comparator wakeup source functions as desired.

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.

3.13 Sleep and Hibernate Regulators

PSoC 3 and PSoC 5LP have two low-power regulators that are used to maintain logic states in Sleep and Hibernate modes.

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

The output of the sleep regulator can be seen on the VCCD and VCCA pins, but it 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.

3.14 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 device 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.

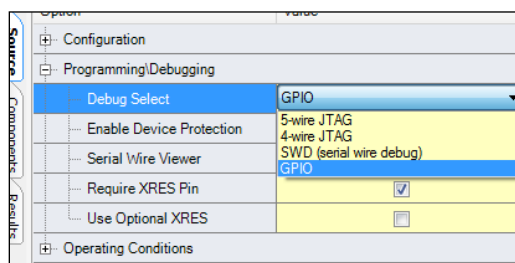
3.15 Is the Debug Interface Running?

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 device 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 be changed only during programming. 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



You can 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 device. 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. For more information about programming and debugging, refer to the device datasheets and TRMs.

4 Approximating Power Consumption

The device datasheet and the component datasheets provide sufficient information to estimate power consumption for a given project. To simplify this process, a spreadsheet is provided that includes typical power requirements for a wide range of internal components. This spreadsheet, *PSoC3_5_Power_Estimator.xls*, can be found on Cypress webpage that hosts this application note, [AN77900](#). Since every project is different, the power calculation provided by this spreadsheet is only an estimate, but it should be close enough to provide good feedback before your design is complete. There are several tabs in the spreadsheet; make sure that you read the **Instructions** tab before entering your data.

5 Summary

Proper low power consumption techniques can have a major impact on portable devices. They can decrease the size of the device battery which in turn reduces the size of your product, or increases your battery life for a specific battery. By taking advantage of the many power-saving features available in PSoC 3 and PSoC 5LP, you can optimize your design and ensure that it consumes the least amount of power.

6 Related Application Notes

These application notes give you more information about 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](#)
- [AN61290 – PSoC 3 and PSoC 5LP Hardware Design Considerations](#)
- [AN54460 – PSoC 3 and PSoC 5LP Interrupts](#)
- [AN60616 – PSoC 3 and PSoC 5LP Startup Procedure](#)
- [AN60631 – PSoC 3 and PSoC 5LP 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 and PSoC 5LP – Segment LCD Direct Drive](#)

About the Author

Name: Rodolfo Lossio

Title: Systems Engineer Sr. Staff

Background: Rodolfo Lossio has a BS in Control and Automation Engineering from Federal University of Santa Catarina, Brazil, and an MSEE from National Chiao Tung University, Taiwan.

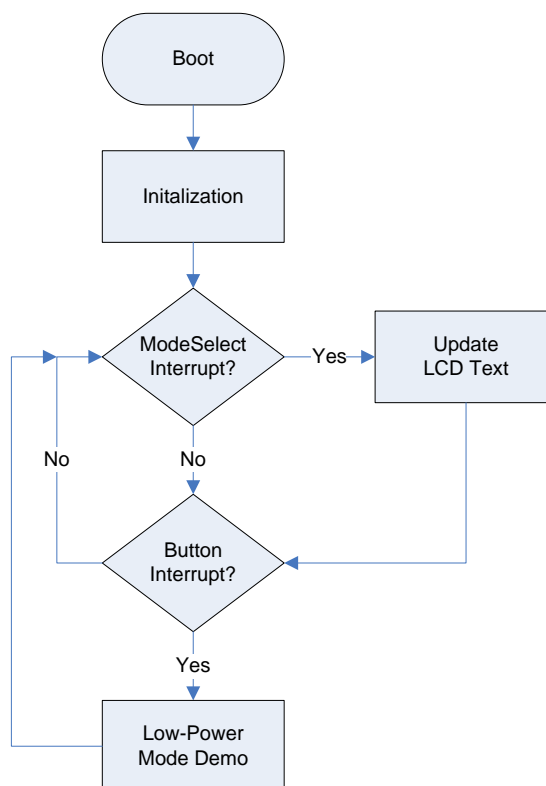
A Low-Power Example Projects

Using PSoC is one of the best ways to become familiar with it. Four example projects are 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.

A.1 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 device to execute the code to demonstrate the selected configuration, as [Figure 23](#) shows.

Figure 23. PSoC Low-Power Demo Flow Chart



The projects demonstrate all of the common Sleep and Hibernate wakeup sources. SegLCD has its own example projects in PSoC Creator, and boost regulation during Sleep mode 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 mode. [Table 3](#) lists the modes and wakeup sources that are demonstrated by these example projects.

Table 3. 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
	I ² C Address
	LCD (not implemented)
Hibernate	PICU
Custom	None (user defined)

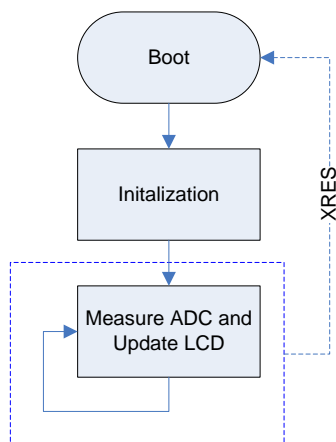
This code demonstrates the principles and techniques described in this application note. No effort has been made to optimize it for size or speed. Instructions for using the projects are located in the project's schematics and source files.

A.2 Voltage Alarm – No Optimization

This project demonstrates a simple voltage measurement and alarm system using a PSoC 3 device, 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 a LED begins to pulse and “Alarm” is displayed on the LCD.

Figure 24. Alarm System – No Optimizations

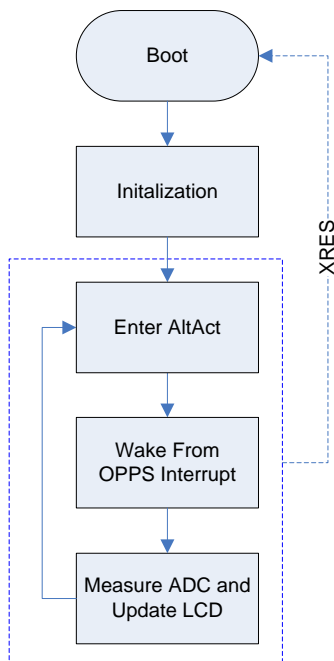


Instructions for using the project are located in the project's schematic and source files.

A.3 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 located in the project's schematic and source files.

B Power Measurements on DVKs

The [Cypress development kits](#) (DVKs) are designed to 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 web pages.

B.1 CY8CKIT-001 Modifications

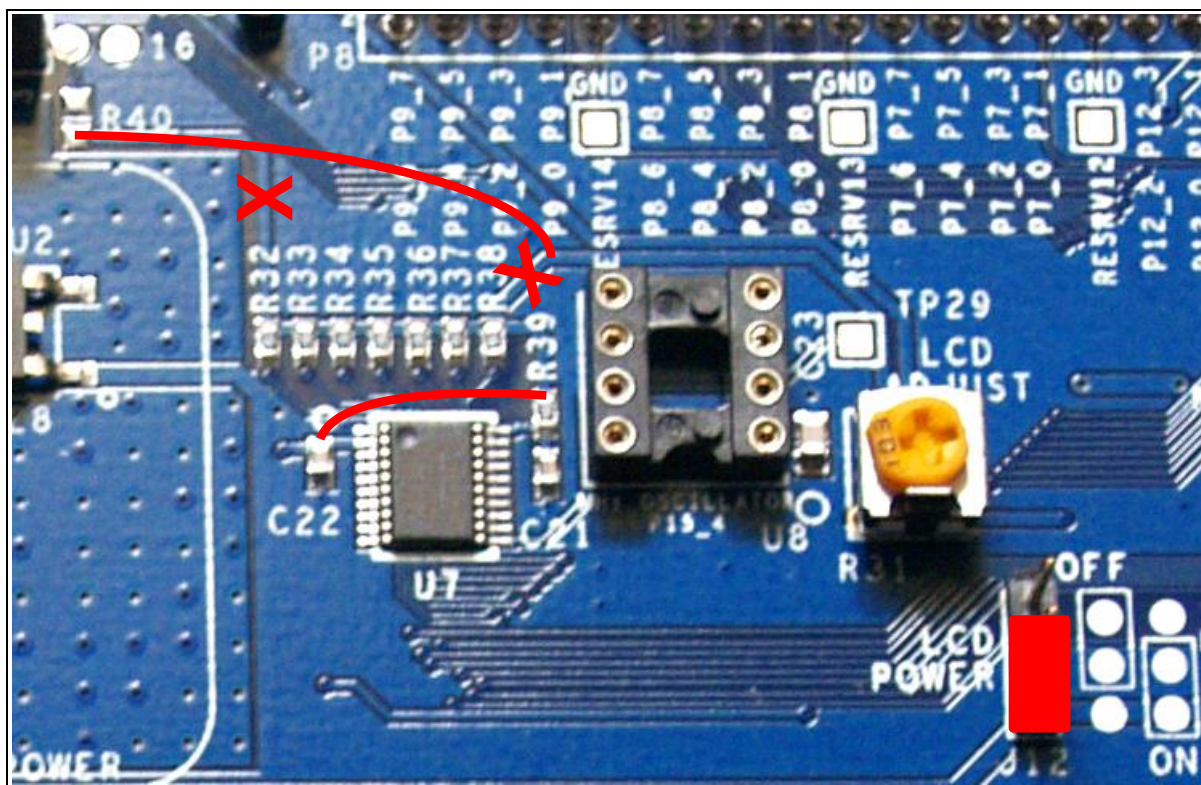
The following steps demonstrate how to modify the CY8CKIT-001 DVK board to ensure proper current measurements. The configuration described here uses a single power rail for VDDD, VDDA, and VDDIO. Power is measured by placing a meter in series with the PSoC VDDD pins.

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 I/O pins and skew current consumption numbers in Sleep or Hibernate mode. The following steps prevent this from happening and allow the character LCD to work whenever the PSoC device is operating at 3.3 V or 5 V. Be forewarned that this modification involves cutting traces on the PCB. [Figure 26](#) on [page 24](#) 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 solder mask 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 the solder mask 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 device 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 CY8CKIT-001 can be configured for 5-V or 3.3-V operation, without any further changes, by using SW3.

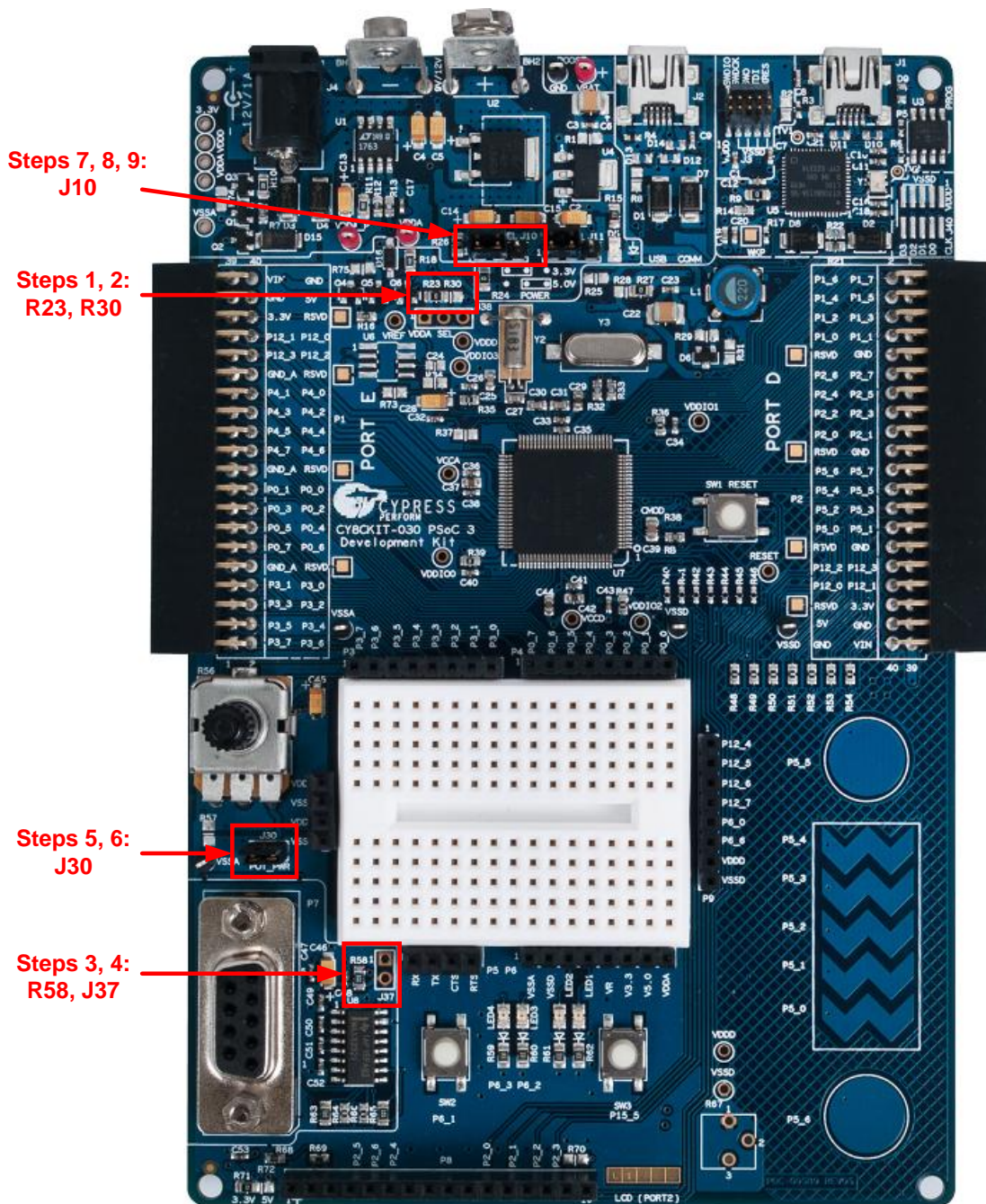
B.2 CY8CKIT-030 and CY8CKIT-050 Modifications

The following steps demonstrate how to modify the CY8CKIT-030 and CY8CKIT-050 DVK boards to ensure proper current measurements. The called-out hardware is identified in [Figure 27](#) on page 25. 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 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 device.
3. Remove R58. This separates the RS-232 driver U8 from VDDA. By default, U8 is powered by VDDA, so it appears 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 appear in your power measurements.
5. Remove the jumper from J30. By default, R56 is powered by VDDA, so it appears 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 device can be accurately measured. All the other components on the DVK board are powered separately from VDDD, so they do not appear 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. CY8CKIT-030/050 Low-Power Modifications



C Power Management API and Registers

Cypress provides API routines for handling the PSoC device'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 contain further information on the registers.

C.1 CyPmSaveClocks()

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

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

C.2 CyPmRestoreClocks()

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

C.3 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.

C.4 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 mode.

C.5 CyPmHibernate() or CyPmHibernateEx()

The *CyPmHibernate()* manages the transition to Hibernate mode. It has no parameters, enabling PICU and XRES as valid wakeup sources.

The *CyPmHibernateEx()* has one parameter used to change the wakeup sources from hibernate mode. They can be configured as: PICU interrupt, Comparator0, Comparator1, Comparator2, and Comparator3 output. This allows the user to specify the exact wakeup sources, and to change them easily if required by different modes of operation. It basically allows the design to mask off wakeup sources that are not needed or required.

You should call *CyPmSaveClocks()* before these functions to ensure that the clock states are saved properly before entering Hibernate mode. Also, the PSoC device should not re-enter a low-power mode until at least 20 µs after a wake from Hibernate mode to ensure that the sleep regulator is stable.

C.6 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, which 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.

C.7 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.

C.7.1 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 device, as [Table 4](#) on page 27 shows. Reading these bits tells you which mode the PSoC device is in currently.

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 device for low-power operation. More information on the global power settings and how they affect the PSoC 3 and PSoC 5LP subsystems is 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.

C.7.2 PM_TW_CFGx

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

C.7.3 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.

C.7.4 PICUx_INTTYPEy

These registers configure the PICU. There is one register for each pin to set the interrupt trigger condition.

C.7.5 PICUx_INTSTAT

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

C.7.6 FASTCLK

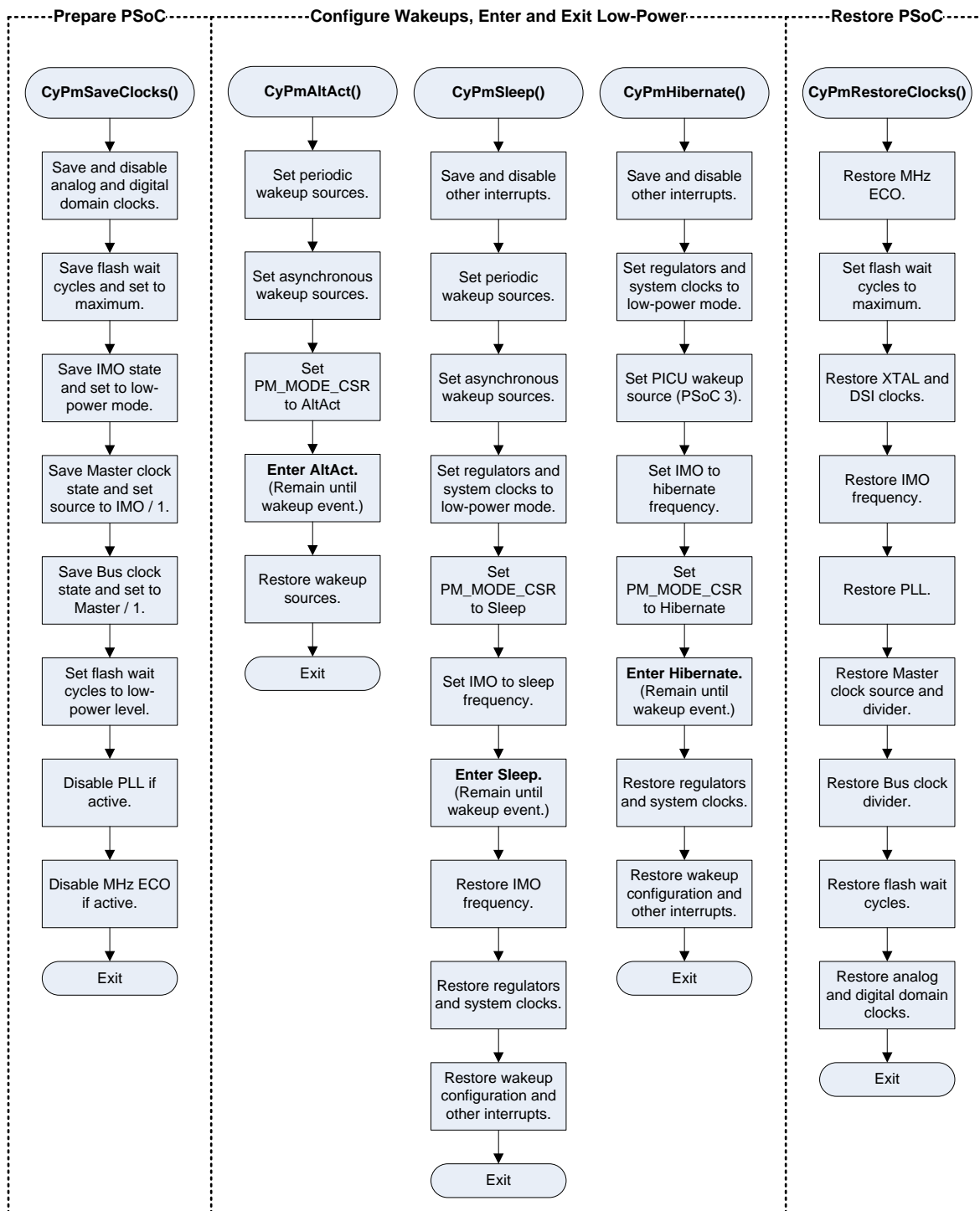
These registers configure the IMO, Master Clock, and PLL. Clock distribution is controlled by the CLKDIST registers.

C.7.7 SLOWCLK

These registers configure the internal low-speed oscillator (ILO) and 32-kHz crystal. Clock distribution is controlled by the CLKDIST registers.

C.8 Power Management API Flow Charts

Figure 28. Power Management Function Flow Charts



C.9 Power Management API Register Reference

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

Table 5. 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_SRC_PICU	PM_WAKEUP_CFG0 [0x04]
	I ² C Address	PM_ALT_ACT_SRC_I2C	PM_WAKEUP_CFG0 [0x08]
	Boost	PM_ALT_ACT_SRC_BOOSTCONVERTER	PM_WAKEUP_CFG0 [0x20]
	FTW	PM_ALT_ACT_SRC_FTW	PM_WAKEUP_CFG0 [0x40]
	CTW	PM_ALT_ACT_SRC_CTW	PM_WAKEUP_CFG0 [0x80]
	OPPS	PM_ALT_ACT_SRC_ONE_PPS	PM_WAKEUP_CFG0 [0x80]
	LCD	PM_ALT_ACT_SRC_LCD	PM_WAKEUP_CFG2 [0x01]
CyPmSleep()	Comparator0	PM_SLEEP_SRC_COMPARATOR0	PM_WAKEUP_CFG1 [0x01]
	Comparator1	PM_SLEEP_SRC_COMPARATOR1	PM_WAKEUP_CFG1 [0x02]
	Comparator2	PM_SLEEP_SRC_COMPARATOR2	PM_WAKEUP_CFG1 [0x04]
	Comparator3	PM_SLEEP_SRC_COMPARATOR3	PM_WAKEUP_CFG1 [0x08]
	PICU	PM_SLEEP_SRC_PICU	PM_WAKEUP_CFG0 [0x04]
	I ² C Address	PM_SLEEP_SRC_I2C	PM_WAKEUP_CFG0 [0x08]
	Boost Converter	PM_SLEEP_SRC_BOOSTCONVERTER	PM_WAKEUP_CFG0 [0x20]
	CTW	PM_SLEEP_SRC_CTW	PM_WAKEUP_CFG0 [0x80]
	OPPS	PM_SLEEP_SRC_ONE_PPS	PM_WAKEUP_CFG0 [0x80]
	LCD Glass Drive	PM_SLEEP_SRC_LCD	PM_WAKEUP_CFG2 [0x01]
CyPmHibernate()	PICU	N/A	PM_WAKEUP_CFG0 [0x04]

Table 6 lists the periodic wakeup sources available in PSoC 3 devices.

Table 6. 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_SLEEP_TIME_ONE_PPS	PM_TW_CFG2 [0x20] – Unmask PM_TW_CFG2 [0x10] – Enable
	CTW 2 ms	PM_ALT_ACT_TIME_CTW_2MS PM_SLEEP_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_SLEEP_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_SLEEP_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_SLEEP_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_SLEEP_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_SLEEP_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_SLEEP_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_SLEEP_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_SLEEP_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_SLEEP_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_SLEEP_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_SLEEP_TIME_CTW_4096MS	PM_TW_CFG2 [0x08] – Unmask PM_TW_CFG2 [0x04] – Enable PM_TW_CFG1 [0x0C] – 4096 ms Period
	FTW 10 μ s 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 supports only 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.
*C	4713744	RLOS	04/08/2015	Fixed constant names of the power management APIs. Added the following sections – “SIOs in PSoC Low-Power Mode” and “Approximating Power Consumption.”
*D	4817268	RLOS	07/07/2015	Updated projects to support PSoC Creator 3.2. Added the comparator as a wake-up source for the hibernate mode. Added explanation of CyPmHibernateEx() API in the Appendix C. Updated template Sunset Review
*E	5330253	MEH	07/25/2016	Added note about low power with SIO pins when using Vref output mode. Added clarification of AltAct mode. Made some grammatical corrections. Updated figures 17 and 18. Added clarification about CyPmHibernateEx() function. Updated template
*F	5701881	BENV	04/19/2017	Updated logo and copyright

Worldwide Sales and Design Support

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

Products

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

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

PSoC® Solutions

[PSoC 1](#) | [PSoC 3](#) | [PSoC 4](#) | [PSoC 5LP](#) | [PSoC 6](#)

Cypress Developer Community

[Forums](#) | [WICED IOT Forums](#) | [Projects](#) | [Videos](#) | [Blogs](#) | [Training](#) | [Components](#)

Technical Support

cypress.com/support



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

© Cypress Semiconductor Corporation, 2012-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spanion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spanion, the Spanion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.