

PSoC[®] 4 Low-Power Modes and Power Reduction Techniques

Authors: Kannan Sadasivam, Max Kingsbury

Associated Project: Yes

Associated Part Family: PSoC 4100/4200/4100M/4200M/4200L

Software Version: PSoC Creator™ 3.2 and higher

Related Application Notes: For a complete list, [click here](#).

To get the latest version of this application note, or the associated project file, please visit <http://www.cypress.com/go/AN86233>.

AN86233 discusses how to use the PSoC[®] 4 low-power modes and features to operate at very low power levels while retaining essential functionality. Major topics include the five power modes, PSoC Creator™ power management functions, and other power-saving techniques and considerations. Three PSoC Creator example projects are included to demonstrate different aspects of low-power programming.

Contents

1	Introduction.....	1	6.5	Debug Interface	11
2	Power Mode Summary	2	7	Example Projects.....	12
3	Low-Power Mode Details.....	3	7.1	AN86233_PowerModes.....	13
3.1	Sleep Mode.....	3	7.2	AN86233_LPComp.....	14
3.2	Deep-Sleep Mode.....	3	7.3	AN86233_DeepSleepADC	14
3.3	Hibernate Mode	4	8	Hardware Configuration.....	17
3.4	Stop Mode	5	8.1	CY8CKIT-042 Kit (PSoC 4200).....	18
4	Power Mode Wakeup Summary	6	8.2	CY8CKIT-049-42xx Kit (PSoC 4200).....	18
5	Power Reduction Techniques.....	6	8.3	CY8CKIT-043 Kit (PSoC 4200M).....	19
5.1	Turn Off Unused Components	6	8.4	Measuring Current with a DMM	19
5.2	Run Components at a Lower Speed	7	8.5	Approximating Power Consumption.....	19
5.3	Reduce Supply Voltage	7	9	Summary	20
5.4	Use PSoC Device to Gate Current Paths	7	10	Related Application Notes	20
5.5	Use DMA to Move Data	8		Worldwide Sales and Design Support.....	22
6	Other Low-Power Mode Considerations.....	8		Products.....	22
6.1	Clocks.....	8		PSoC [®] Solutions	22
6.2	WDT.....	10		Cypress Developer Community.....	22
6.3	GPIOs.....	10		Technical Support	22
6.4	Deep-Sleep and Hibernate Regulators	11			

1 Introduction

PSoC 4 low-power modes allow you to reduce overall power consumption while retaining essential functionality, especially when implemented with other power-saving features and techniques. This application note describes the PSoC 4 low-power modes, provides information on active mode power-saving methods, and discusses other low-power considerations.

The document assumes that you are familiar with developing applications using PSoC Creator for PSoC 4. If you are new to PSoC 4, you can find an introduction in [AN79953 – Getting Started with PSoC 4](#). If you are new to PSoC Creator, see the [PSoC Creator home page](#). For more information on PSoC 4, see [Related Application Notes](#).

2 Power Mode Summary

PSoC 4 features five power modes of operation. In the order of power consumption and functionality, they are active, sleep, deep-sleep, hibernate, and stop. Table 1 lists the typical current and wakeup times for each power mode.

Table 1. Power Mode Specs

Power Mode	Current Range (typical) (V _{dd} = 3.3 V to 5.0 V)	Wakeup Time		
		PSoC 4200	PSoC 4200M	PSoC 4200L
Active	1.3 mA to 14 mA	–	–	–
Sleep	1.0 mA to 3 mA	0	0	0
Deep-sleep	1.3 μ A to 15 μ A	25 μ s	25 μ s	25 μ s
Hibernate	150 nA to 1 μ A	2 ms	0.7 ms	0.7 ms
Stop	20 nA to 80 nA	2 ms	2 ms	1.9 ms

The differences among the low-power modes include CPU and peripheral availability, wakeup and reset sources, power mode transition behaviors, and power consumption. Table 2 shows the PSoC 4 resources and their availability in different power modes.

Table 2. PSoC 4 Power Modes—Resources Availability

Subsystem	Active	Sleep	Deep-Sleep	Hibernate	Stop
CPU	On	Retention ¹	Retention	Off	Off
SRAM	On	On	Retention	Retention	Off
High-speed peripherals (SPI, UART, and so on)	On	On	Retention	Off	Off
Universal digital blocks (UDBs)	On	On	Retention ⁴	Off ⁵	Off
SPI slave and I ² C slave (SCB based)	On	On	On	Off	Off
High-speed clock (IMO, ECO, and PLLs)	On	On	Off	Off	Off
Low-speed clock (32 kHz) (ILO and WCO)	On	On	On	Off	Off
Brown-out detection	On	On	On	On	Off
CTBms (opamp and comparators)	On	On	On ³	Off	Off
ADC	On	On	Off	Off	Off
Low-power comparators	On	On	On	On	Off
GPIO (output state)	On	On	On	On	Frozen ²

¹ Retention: The configuration and state of the peripheral are retained. The peripheral continues its operation when the device enters active mode.

² Frozen: The configuration, mode, and state of all GPIOs in the system are locked. Changing the GPIO state is not possible until the device re-enters active mode and the pins are unlocked.

³ Some parts support a low-power mode called “deep-sleep.”

⁴ The state of any component that is based on UDBs is preserved by calling the component's `_Sleep()` function before entering deep-sleep, and the `_WakeUP()` function after exiting from deep-sleep.

⁵ When exiting hibernate, all components are reinitialized because the PSoC device goes through a reset.

3 Low-Power Mode Details

The four low-power modes of PSoC 4 are demonstrated in the AN86233_PowerModes example project included with this application note.

- Active mode is the normal operating mode in which all peripherals are available and the CPU is active.
- In sleep mode, all peripherals except the CPU are available.
- In deep-sleep mode, the CPU, most peripherals, and the MHz clocks are disabled.
- In hibernate mode, no clocks are available, but logical states are retained.
- In stop mode, the CPU, clocks, and all peripherals are halted, and logical states are not retained, but GPIO states are retained or frozen.

3.1 Sleep Mode

In sleep mode, the PSoC 4 ARM Cortex-M0 CPU does not run instructions; instead, it waits for an interrupt to occur. The SRAM is retained but not written to or read by the CPU. It can be accessed by DMA for those parts that support DMA. All other peripherals and clocks continue to run.

3.1.1 Sleep Mode Wakeup Sources

Any interrupt source in the device can be used to wake the device from sleep mode. All peripherals can remain active and generate interrupts.

3.1.2 Sleep Mode Transitions

Enter sleep mode using the API function `CySysPmSleep()`. This function configures the device for sleep. No other API calls are necessary.

Exit from sleep mode occurs when an interrupt is triggered. Upon exiting sleep, PSoC 4 re-enters active mode. The configuration of sleep wakeup sources requires only that their interrupts be enabled.

3.1.3 Sleep Mode Use Cases

Sleep mode should be used when peripherals such as the ADC, CapSense®, digital communication, or others must remain active, but CPU activity is not required. This can reduce current consumption between events such as ADC conversions and digital communication transactions.

3.2 Deep-Sleep Mode

In deep-sleep mode, the high-frequency clocks and peripherals that require high-frequency clocks are disabled. The high-frequency clocks include the internal main oscillator (IMO), external crystal oscillator (ECO), and phase-locked loops (PLLs). Note that the ECO and PLLs are not available on all PSoC 4 devices.

The internal low-speed oscillator (ILO) clock remains active and can be used to clock the watchdog timer (WDT), which can be used as a sleep timer to wake the system from deep-sleep. Some PSoC 4 devices also have an option to include a watch crystal oscillator (WCO) that can operate during deep-sleep.

The I²C block can continue operating as a slave to monitor the I²C bus, allowing wakeup upon I²C address match.

3.2.1 Deep-Sleep Mode Wakeup Sources

An I²C address match, the WDT, GPIO interrupts, and low-power comparator interrupts can wake the device from deep-sleep. The WDT block contains multiple counters that can be configured independently to generate interrupts, resets, or both. This allows the WDT to function as a sleep timer as well.

3.2.2 Deep-Sleep Mode Transitions

Enter deep-sleep mode using the API function `CySysPmDeepSleep()`. This function configures the device for deep-sleep. No other API calls are necessary, unless you want to turn off components that may operate in deep-sleep to save power and/or save their current state. In this case, use the component's specific API `_Sleep()` to store the current component state before calling `CySysPmDeepSleep()` and `_WakeUp()` after waking up from deep-sleep. Depending on your application, it may be faster to simply call the `_Stop()` function before entering deep-sleep and `_Start()` after waking.

Exit from deep-sleep mode occurs when an interrupt is triggered. Upon exiting deep-sleep, PSoC 4 re-enters active mode. The configuration of deep-sleep wakeup sources requires only that their interrupts be enabled.

3.2.3 Deep-Sleep Mode Use Cases

The deep-sleep mode should be used when the PSoc 4 high-performance analog and digital peripherals are not needed, but the device still needs to be able to wake up periodically using the WDT or upon an I²C address match.

Regular wakeup intervals can be used to periodically use active mode peripherals, such as the ADC, to take readings or scan CapSense button inputs.

3.3 Hibernate Mode

In hibernate mode, all clocks and synchronous peripherals in PSoc 4 are disabled. The pins and low-power comparators may remain active, and the SRAM and UDB register states are retained.

3.3.1 Hibernate Mode Wakeup Sources

Pin and low-power comparator interrupts can wake the device from hibernate mode. Any wakeup from hibernate causes a device reset, but SRAM and some register states are retained, allowing the wakeup reset reason to be detected. Although SRAM is retained, a reset by default initializes all variables to their user-defined initialized state or zero. To prevent a variable from getting reinitialized after waking from hibernate, use the `CY_NOINIT` attribute in the definition of the variable. This works only for global or static variables. To make sure a variable does get initialized, your code should determine if the reset was not caused by a wakeup from hibernation. After the return to active mode after a reset, you can call the `CySysPmGetResetReason()` API to determine if the reset was a wakeup from hibernate or any other reset condition. The following code shows how to define a non-initialized variable and determine if the reset was not due to a hibernate wakeup.

```
/* Define non-initialized global variable */
int32 CY_NOINIT testVarNoInit;

/* Inside of main() */
/* Initialize variable when PSoc is not reset from hibernation wakeup */
if( CySysPmGetResetReason() != CY_PM_RESET_REASON_WAKEUP_HIB )
{
    testVarNoInit = 0;
}
```

3.3.2 Hibernate Mode Transitions

Enter hibernate mode using the `CySysPmHibernate()` API function. This function configures the device for hibernate mode. No other API calls are necessary, since all components including clocks are powered down and will be reinitialized when exiting hibernate mode with a reset. The only exception is that the low-power comparator may remain active to cause a wakeup from hibernate.

Exit from hibernate mode occurs when a pin or low-power comparator interrupt is triggered. Upon exiting hibernate mode, PSoc 4 resets. After the return to active mode after reset, you can call the `CySysPmGetResetReason()` API function to detect the hibernate wakeup reset. The specific pin or comparator interrupt can then be detected using the component APIs, as those register states are retained.

Although not required, you have the option to lock the state of all I/O cells (GPIOs) by calling the `CySysPmfreezeIo()` API function before entering hibernate mode. The `CySysPmUnfreezeIo()` API function should be called after a wakeup from hibernate before the pins can again change state. Using these functions will make sure that there are no unexpected GPIO transitions during and after reset.

3.3.3 Hibernate Mode Use Cases

Hibernate mode should be used when periodic wakeup is not required, but the device must use less than a 1- μ A current. It may also be useful for a wakeup on an analog or digital signal transition, while requiring minimal current. Note that hibernate mode can be used effectively if your code is organized as a state machine, and the CPU can start executing the code from the previously known state before wakeup once the device wakes up from hibernate. You must use the `CY_NOINIT` attribute in the definition of the state variables to make sure that the state variables are not reinitialized after wakeup from hibernate. You can use the `CySysPmGetResetReason()` API function to detect the hibernate wakeup reset, as mentioned in the previous section, to ensure that the device was reset due to hibernate.

3.4 Stop Mode

Stop mode offers the lowest possible current consumption without removing power from the PSoC 4 supply pins. All peripherals are disabled, and the SRAM and register states are **not** retained. The device pins may be “frozen” and retain their drive modes and logical states. A dedicated wakeup pin, P0[7], is available to wake the device from stop mode.

3.4.1 Stop Mode Wakeup Sources

The dedicated wakeup pin P0[7] is the only wakeup source available in stop mode. You can set its input wakeup polarity to rising or falling edge by calling the `CySysPmSetWakeupPolarity()` API function.

3.4.2 Stop Mode Transitions

Enter stop mode using the `CySysPmStop()` API function. This function configures the device for stop mode, including freezing the I/O states. If you use the dedicated wakeup pin, set its input wakeup polarity before entering stop mode with the `CySysPmSetWakeupPolarity()` API.

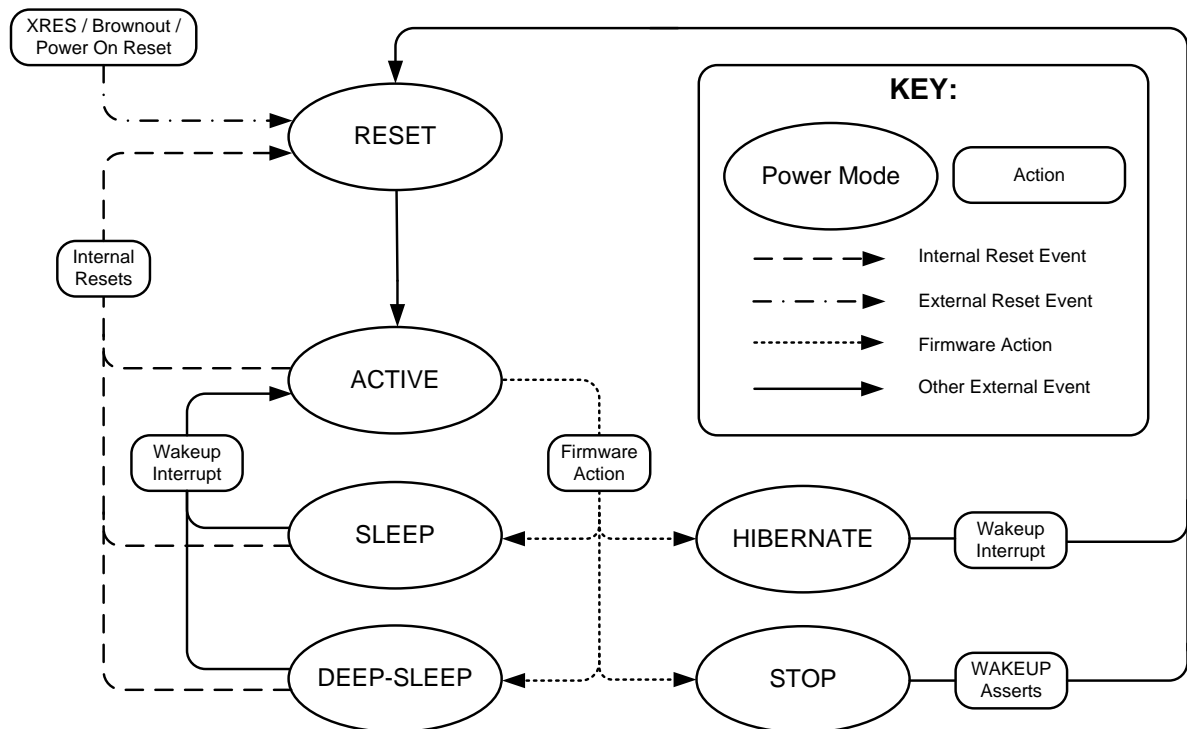
Exit from stop mode occurs when the dedicated pin wakeup is triggered, the reset signal goes low, or if power is cycled. Upon exiting stop mode, PSoC 4 resets. Upon return to active mode after reset, you can call the `CySysPmGetResetReason()` API function to determine if stop mode was exited by toggling the wakeup pin or power cycle. After a wakeup (pin) reset, the GPIO states remain frozen and must be unfrozen using the `CySysPmUnfreezeIo()` API function before the pin states can be changed. There is no need to call the `CySysPmfreezeIo()` API function before entering stop mode since the `CySysPmStop()` function freezes I/O cells implicitly.

3.4.3 Stop Mode Use Cases

Use stop mode when the absolute minimum power consumption and functionality are required. It is most useful in applications where a host controller or a user input such as a button press can trigger the dedicated wakeup pin, and the power supply topology does not allow disconnecting power from the device.

Figure 1 shows the transitions between the power modes in PSoC 4. Table 3 lists the available wakeup sources for each mode.

Figure 1. PSoC 4 Power Mode Transitions Diagram



4 Power Mode Wakeup Summary

All power modes exit upon two conditions: power cycling or driving the reset input low. All states are lost in either case for **all** power modes. Table 3 provides a summary of the events and hardware that can be used to exit the current low-power mode and return to active mode.

Table 3. Low-Power Modes and Wakeup Sources

Low-Power Mode	Wakeup Source	Wakeup Action
Sleep	Any interrupt source	Interrupt
	Any reset source	Device reset
Deep-Sleep	GPIO interrupt (PICU) ¹	Interrupt
	Low-power comparator	Interrupt
	SCB (I ² C address match)	Interrupt
	WDT ²	Interrupt or device reset
	XRES (external reset pin) ³	Device reset
Hibernate	GPIO interrupt (PICU)	Device reset
	Low-power comparator	Device reset
	XRES (external reset pin) ³	Device reset
Stop	WAKEUP pin (P0[7])	Device reset
	XRES (external reset pin) ³	Device reset

¹ PICU (port interrupt control unit): Each port contains a PICU to manage individual GPIO interrupts.

² The WDT can be configured to generate both interrupts and resets at different intervals. See the technical reference manual (TRM) for the applicable PSoC 4 device family.

³ XRES triggers a full system reset. All states including frozen I/Os are lost. In this case, the cause of wakeup is not detectable after the device restarts.

5 Power Reduction Techniques

In many applications, you can gain additional current reductions by properly managing internal PSoC 4 components as well as powering down external components. This section presents some techniques for doing so.

5.1 Turn Off Unused Components

The easiest way to reduce power in active mode is to turn off unused PSoC Creator Components.

Any Component that can be disabled in active or sleep mode includes 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 here.> */

/* Stop the Component. */
MyComponent_Stop();

```

Restart a Component by calling its Start function.

```

/* Start the Component. */
MyComponent_Start();

```

Any Component that must preserve its configuration data before powering down includes a `_Sleep()` function in its API. The `_Sleep()` function saves all necessary Component settings and then calls the `_Stop()` function. In some cases, the `_Sleep()` function does nothing but call `_Stop()`.

```

/* < Check task status here.> */

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

```

When a Component is put to sleep, it should be awakened 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.

```

/* Wake the Component. */
MyComponent_Wakeup();

```

`_Sleep()` and `_Stop()` both result in the same amount of power savings. The difference lies in whether or not the Component needs to resume from exactly where it left off.

5.2 Run Components at a Lower Speed

Clocked integrated circuits consume more current as their clock rates increase. This is because parasitic and designed capacitances are charged and discharged more rapidly, requiring more current. Reducing the operating frequency of PSoC 4 components can greatly reduce the current consumption. This technique can be applied to the Cortex-M0 CPU, SAR ADC, digital components, and others.

5.3 Reduce Supply Voltage

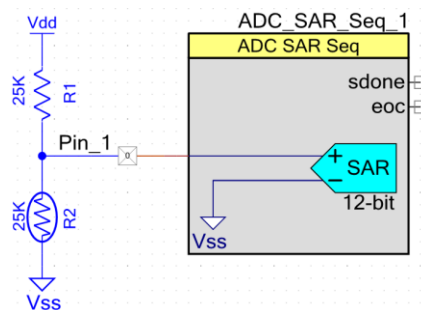
Reducing the supply voltage is probably the easiest thing you can do to reduce the overall power consumption. Even if the current stays the same, reducing the supply voltage from 5 V to 3 V reduces power consumption by 40 percent! Although the PSoC 4 device can operate below 1.8 V, you still need to consider the voltage requirements of other devices in the system.

5.4 Use PSoC Device to Gate Current Paths

Your PCB may contain other components that draw power; the PSoC device can be used to control the current consumed by 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 shown in [Figure 2](#). In this case, the PSoC device measures the temperature using the voltage on an analog pin, which changes as the thermistor resistance changes.

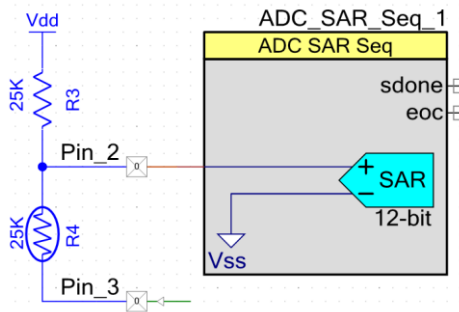
Figure 2. Typical Thermistor Application



The ADC can be turned off when not in use, but external components still consume power because the current path through the resistor and thermistor remains. A simple solution with PSoC 4 is to use a second pin as a switch to ground, as shown in Figure 3.

In this configuration, the current flow can be stopped by writing a '1' to Pin_3, allowing the pin to float. This removes the current consumption by reducing the voltage differential across the two resistors to 0 V. Writing a '0' resumes the current flow. The resource use of this power-saving feature is only one pin and a few lines of code.

Figure 3. Using a GPIO as a Ground Switch



5.5 Use DMA to Move Data

You can save power any time you offload a task from the CPU and either halt the CPU or let it do something else in parallel. The PSoC 4200M and PSoC 4200L devices have a DMA engine that can be used in active or sleep modes to transfer data with no CPU use.

6 Other Low-Power Mode Considerations

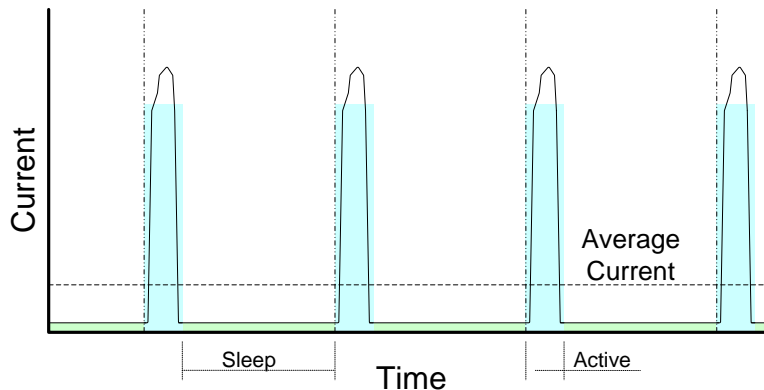
This section presents tips, tricks, and recommendations related to the use of PSoC 4 low-power modes.

6.1 Clocks

In some cases, running the clocks faster can 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 calculations, and then transmits the results to another device.

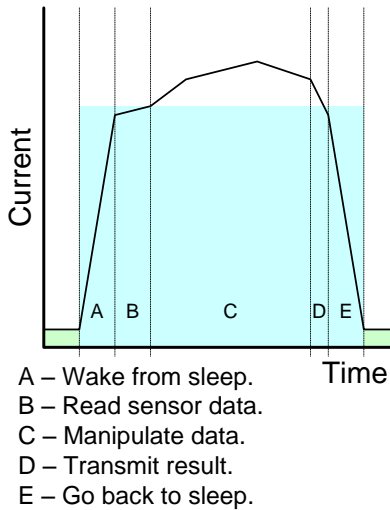
You can use sleep or deep-sleep mode to reduce the power when the PSoC device is idle, but the average current consumption is higher because of the time spent in active mode. Figure 4 is a representation of the current consumption of this example with the system clocks set at 3 MHz.

Figure 4. Example Current Profile with 3-MHz Clocks



Depending on the tasks or calculations 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 consumption because the PSoC device is in active mode for less time. Figure 5 is a representation of active mode timing, broken up into tasks.

Figure 5. Analysis of Tasks in Active Mode at 3 MHz



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. 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 6 shows. With a 12-MHz clock, the time spent in active mode is about half as long as with a 3-MHz clock. Figure 7 shows that the peak current consumption is greater when the clocks are faster, but the overall average is lower.

Figure 6. Analysis of Tasks in Active Mode at 12 MHz

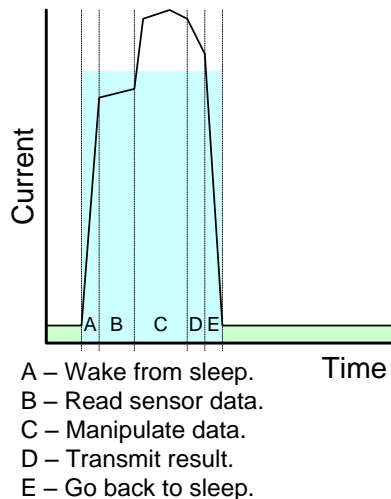
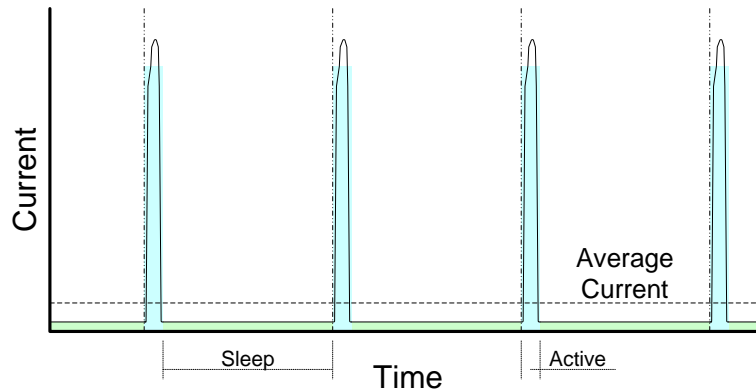


Figure 7. Example Current Profile with 12-MHz Clocks



You may also be able to reduce the peak active current by applying other suggestions in this application note. [AN86233_DeepSleepADC](#) is an example project that wakes once a second, takes a temperature reading with the ADC, transmits the data from a UART, and then goes back to deep-sleep.

6.2 WDT

The WDT can operate in active, sleep, and deep-sleep modes. The counters in the WDT can generate interrupts or resets, depending on the configuration and operating conditions. This allows the WDT to replace a traditional sleep timer in addition to guaranteeing reliable operation.

Increasing the WDT interval or disabling it entirely before entering low-power modes can reduce the time spent in active mode and the overall power consumption. The WDT is not active in hibernate or stop modes.

More information on the operation of the WDT and the associated API is available in the PSoC 4 TRM and the [PSoC Creator System Reference Guide](#).

6.3 GPIOs

The GPIOs can continue to drive the external circuitry when the PSoC device is in a low-power mode. This is helpful when you need to hold 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 Pins Component's `_Write()` API function to set it.

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

Configure all unused GPIOs to Analog HI-Z unless there is a specific reason to use a different drive mode. A Pins Component's port-wide drive mode may be set using the `_SetDriveMode()` API function.

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

The flexibility of PSoC 4 makes it easy to manage GPIO drive modes to prevent unwanted current leakage. See [AN86439 – PSoC 4 – Using GPIO Pins](#) for more information.

In stop mode, the GPIO drive modes and data registers may be “frozen.” The pin states are automatically frozen by the PSoC Creator stop mode entry API function, `CySysPmStop()`. They must be “unfrozen” after a wakeup reset to allow their states to change—call the `CySysPmUnfreezeIo()` API function.

6.4 Deep-Sleep and Hibernate Regulators

PSoC 4 has two low-power regulators that are used to maintain logic states in deep-sleep and hibernate:

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

Neither of these regulators powers the V_{CCD} supply. Each powers an internal power supply domain and is not brought out on a pin of the device.

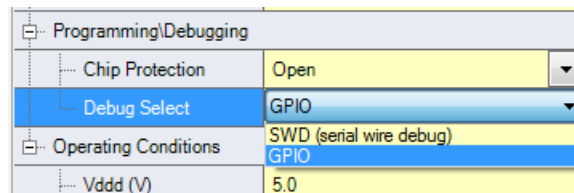
6.5 Debug Interface

PSoC 4 supports on-chip debugging. You may observe a higher current consumption than expected while in debug mode. This is normal because the programming and debug interface remains active in all low-power modes.

Power measurements may also be skewed if the debug pins are set to SWD mode and a MiniProg3 programmer/debugger is attached, even if the PSoC 4 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 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 8 shows.

Figure 8. Disable Debug Interface to Reduce Power



When the debug interface is disabled by setting these pins to GPIO mode, a reset must occur 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. If the pins are in SWD mode, one of two things should be done before entering a low-power mode to achieve minimum power: 1) pull the pins up or down using external components, or 2) change the pin mode to HI-Z for the SWD pins.

It is recommended that you set the debug interface pins to GPIO mode for any released product. For more information about programming and debugging, refer to the device datasheets and TRMs.

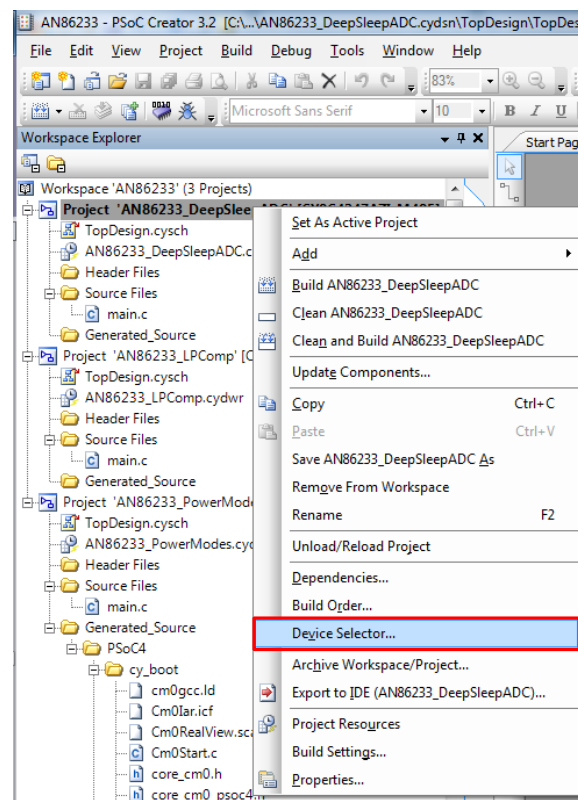
7 Example Projects

Three projects are included with this application note to demonstrate several of the concepts discussed. All the source code for each example is included in its *main.c* file and is heavily commented. It is recommended that you take the time to walk through each example's source code to learn the basic operation. These projects were designed for the CY8C4245AXI-483 device used in the CY8CKIT-042 and CY8CKIT-049 kits, but they can be easily adapted to other kits that use one of the 4100/4200/4100M/4200M devices. You may be required to change the pins connecting to external components and the device selection. Review the specific kit documentation for connections and device type.

To configure the projects for any other PSoC 4 device:

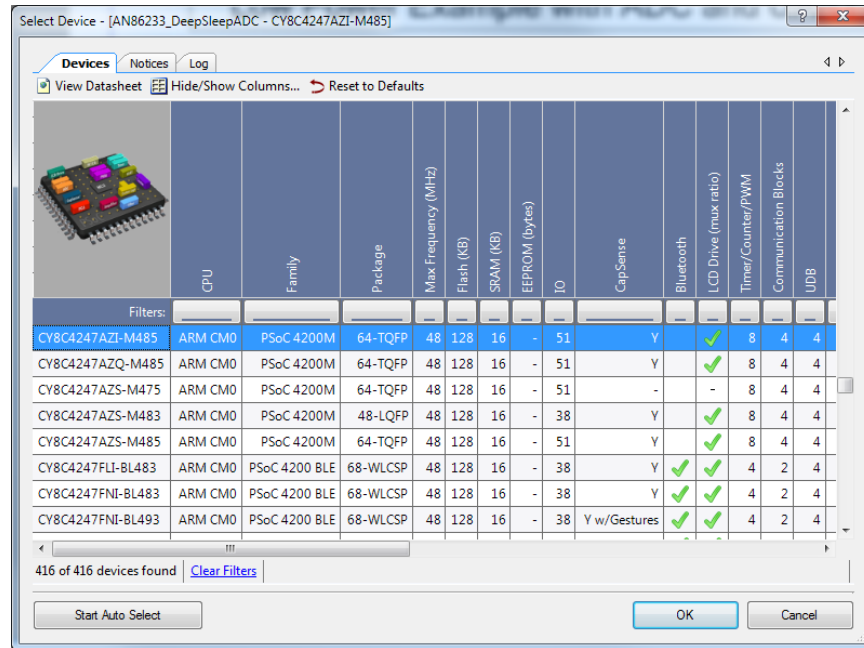
1. Open the project in PSoC Creator.
2. Select the option **Device Selector** from the context menu, as [Figure 9](#) shows.

Figure 9. Select the Device Selector Option



- Select the appropriate device from the pop-up menu that appears, as [Figure 10](#) shows.

Figure 10. Device Selector Window



Note: If you are using a development kit, right-click on the list of devices, select the option **Select Default Device**, and select the appropriate device.

7.1 AN86233_PowerModes

This project is designed to allow you to easily compare all four low-power modes with the active mode. You can edit the code in one place to set the constant `PM_TO_DEMO` to the desired power mode constant. [Code 1](#) shows an example of the code in which the deep-sleep mode to be demonstrated is selected.

Code 1. Selecting Power Mode

```

/* Power Modes Constants */
#define PM_SLEEP          1u
#define PM_DEEP_SLEEP    2u
#define PM_HIBERNATE     3u
#define PM_STOP          4u

/* Set PM_TO_DEMO to power mode */
#define PM_TO_DEMO    PM_DEEP_SLEEP
    
```

After this selection is made, build the project and program the device. The project starts in active mode until the switch on the development board is pressed. The LED will flash quickly according to the number of times assigned to the power mode constant. In this example, the LED will flash twice for deep-sleep.

The project will stay in the low-power mode until the switch is pressed again. It will then flash slowly the same number of times to let you know that the project is back to the active state. Using an ammeter, you can evaluate the current in each mode. To evaluate a different power mode, set `PM_TO_DEMO` to a different power mode constant, rebuild the project, and reprogram the PSoC device.

7.2 AN86233_LPComp

This project demonstrates how the low-power comparator (LPComp) can be used to wake the PSoC device from hibernate and return to active mode. Two 1-M Ω external resistors and a potentiometer or voltage source will need to be added to the development kits to complete the project. The two resistors form a voltage divider to provide a reference voltage that is one-half of Vdd. The potentiometer simulates an external changing voltage that will trigger an exit from hibernate back to active mode by causing an interrupt. An interrupt will occur any time the adjustable voltage from the potentiometer makes a transition from higher than the reference to lower than the reference, or vice versa.

After a reset, the project is in active mode. The LED is lit anytime the project is in active mode and turned OFF in hibernate. Pressing the switch will put the PSoC device in hibernate mode until the LPComp senses that the input voltage is changing from above to below the reference voltage (Vdd/2 in this case), or vice versa.

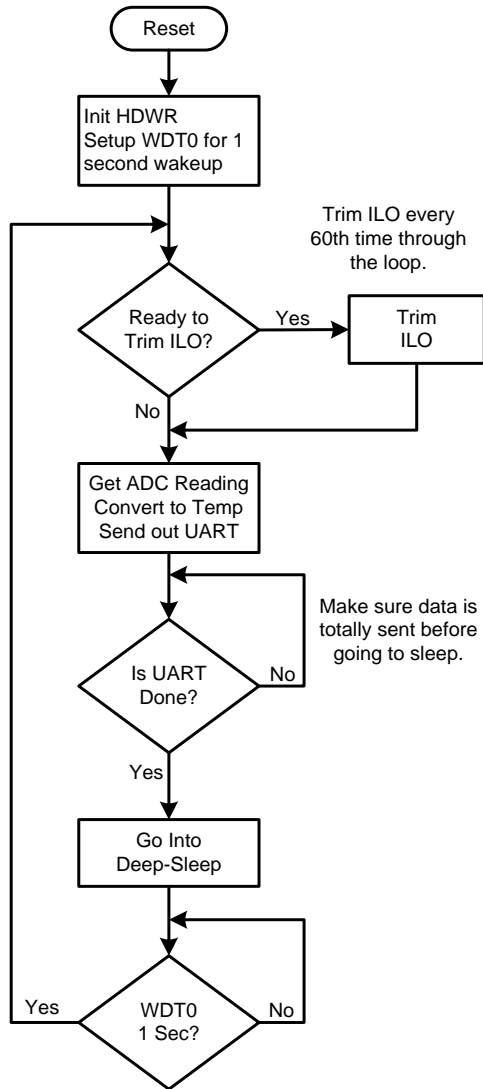
Feel free to change the code so that instead of going into hibernate when the switch is pressed, the device goes into sleep or deep-sleep. The operation should be identical, but the power consumption will differ among low-power modes.

7.3 AN86233_DeepSleepADC

This project is an example of how to periodically wake up, take a reading with the SAR ADC, do a transform on the data, send the data out the serial port (UART), and switch back to deep-sleep mode. It will perform this sequence about once a second but remain in deep-sleep 99.9 percent of the time between samples to reduce power by almost three orders of magnitude.

The project uses the ILO and the WDT (WDT0) to wake from deep-sleep once a second. The ILO is a low-power oscillator but is not very accurate, with a specification of 32 kHz -50 percent to $+100$ percent. The IMO, on the other hand, is much more accurate at ± 2 percent, but it requires more power and does not operate during deep-sleep. It always operates when the PSoC device is in active mode. You can use the IMO to calibrate the ILO with the ILO Trim Component. ILO Trim is used to adjust the WDT0 counter period so that it is within ± 10 percent. The large drift of the ILO is mainly a function of voltage and temperature, so you do not have to adjust the trim that often. In the example, it is trimmed every 60th time the Component wakes from deep-sleep or once a minute. You can change that from a fixed 60-second update to whenever the measured die temperature or supply voltage changed by more than a set value. See [Figure 11](#) for the program flow of the example.

Figure 11. AN86233_DeepSleepADC Program Flow

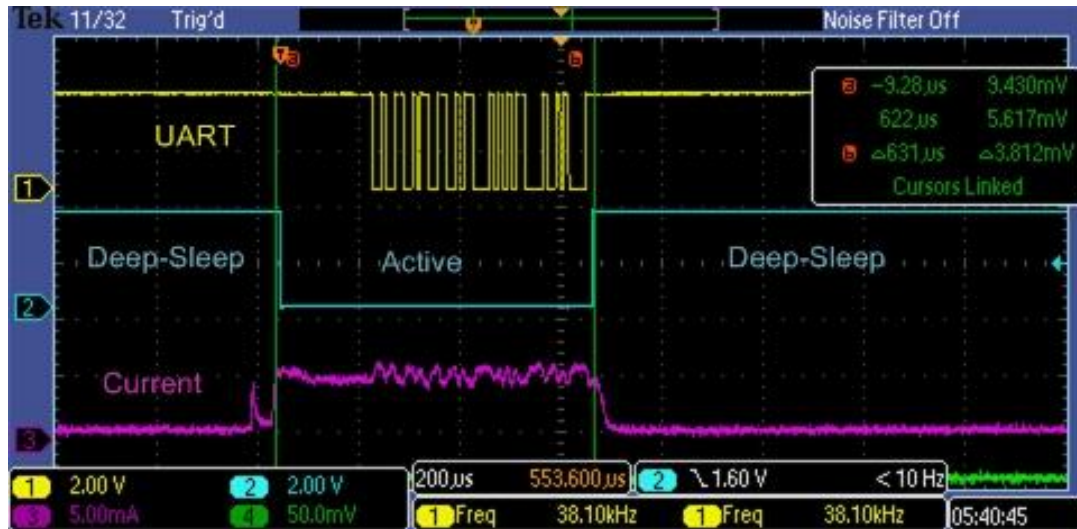


7.3.1 Average Power

To determine the average power of this example, you need to measure both the time and current in each mode, assuming that the duty cycle between power modes is consistent. This period of the firmware is about 1 second. So if you add up the time that the PSoC device stays in active and deep-sleep modes in one period, it would be about 1 second.

First, measure the current consumption and the time that the device stays in active power mode. Figure 12 shows that the duration in active mode is about 630 μs , and it consumes about 5000 μA (5.0 mA) during that time. You can also see the UART communication in Figure 12 and notice that the PSoC device does not return to deep-sleep until the UART has completed its transmission. The duration of the LED drive (active LOW) signal, along with the current measurement, is used to determine whether the device is in active mode versus deep-sleep mode. In deep-sleep mode, the current is about 3.5 μA , and the signal duration is about 999.4 ms.

Figure 12. Measuring Project Active Mode Parameters



If you average the power for both active and deep-sleep modes, as shown in Table 4, it is easy to see that using the deep-sleep low-power mode reduced the 5-mA current from active mode to an average of 6.7 μA , almost by three orders of magnitude!

Table 4. Average Power Calculation

	Deep-Sleep	Active	Total	Unit
Time (ms)	999.4	0.63	1000	ms
Current (μA)	3.5	5000		μA
Time \times I	3498	3150	6648	ms \times μA
Average Current	6.7			μA

To put all this into perspective, if this project were running on a 300-mAh battery without using the low-power mode, the battery life would be about 60 hours (300 mAh / 5 mA). With power mode cycling, the same 300-mAh battery would last almost 44,800 hours, or over 5 years!

Of course, the duty cycle time in both active and deep-sleep modes greatly affects the average power, but so does the power consumed in each of those modes. Cycling between two power modes does add some complexity to the design, but for battery-powered applications, it may be well worth it.

8 Hardware Configuration

Cypress provides different development kits for the PSoC 4 families (pioneer kits), including the CY8CKIT-042 for PSoC 4200, CY8CKIT-044 for PSoC 4200M, and CY8CKIT-046 for PSoC 4200L. Also available are low-cost kits with a small form factor (prototyping kits), such as CY8CKIT-049 for PSoC 4200 and CY8CKIT-043 for PSoC 4200M. These kits offer different features, but all include a user switch and LED connected to the PSoC 4 device. You must make the appropriate pin assignments for each kit for the projects to function properly. [Table 5](#) provides the pin mapping across the PSoC 4 pioneer kits; [Table 6](#) provides the pin mapping across the PSoC 4 prototyping kits.

Table 5. Pin Mapping Across PSoC 4 Pioneer Kits

Function	CY8CKIT-042 (PSoC 4200)	CY8CKIT-044 (PSoC 4200M)	CY8CKIT-046 (PSoC 4200L)
Red LED (Active LOW)	P1[6]	P0[6]	P5[2]
Switch (Active LOW)	P0[7]	P0[7]	P0[7]
UART Tx*	P4[1]	P7[1]	P3[1]

*Other compatible pins also can be used to connect to the UART Tx. On CY8CKIT-044 and CY8CKIT-046, the pins listed in the table are hardwired to the onboard KitProg USB-to-UART Bridge.

Table 6. Pin Mapping Across PSoC 4 Prototyping Kits

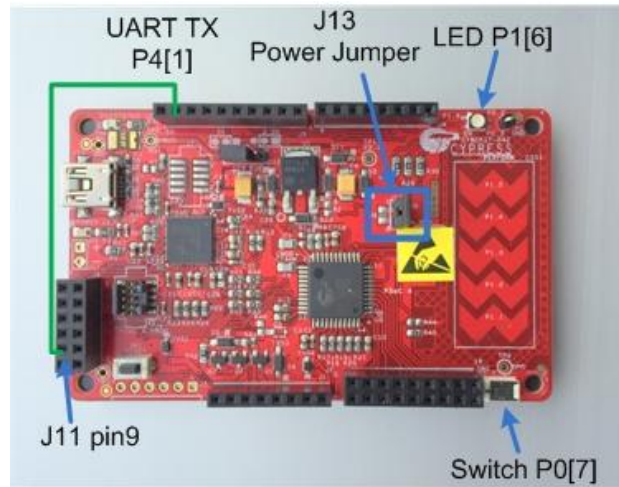
Function	CY8CKIT-049 (PSoC 4200)	CY8CKIT-043 (PSoC 4200M)
Blue LED (Active HIGH)	P1[6]	P1[6]
Switch (Active LOW)	P0[7]	P0[7]
UART Tx*	P4[1]	P7[1]

*Other compatible pins also can be used to connect to the UART Tx. On CY8CKIT-049 and CY8CKIT-043, the pins listed in the table are hardwired to the onboard USB-to-UART Bridge.

8.1 CY8CKIT-042 Kit (PSoC 4200)

This kit includes a built-in debugger and connectors that are compatible with many common Arduino shield boards. The most important feature of this kit pertaining to this application note and measuring current is the power jumper, J13. To measure the current of the PSoC device, remove this jumper and place an ammeter across the pins. By doing so, you are measuring only the current of the PSoC 4 device and not the rest of the board. This board also includes the LED and the switch required to exercise these examples. Another feature of the board is a USB-to-Serial port adapter. By connecting a wire from PSoC P4[1] to J11 pin 9, as shown in [Figure 13](#), you can use your favorite terminal emulator to monitor the serial output of the AN86233_DeepSleepADC project. The baud rate is 115.2 kbps, 8-bit data, and no parity. For more information on using the USB-to-Serial port feature, refer to the [CY8CKIT-042 – PSoC 4 Pioneer Kit Guide](#).

Figure 13. CY8CKIT-042 Kit Hardware

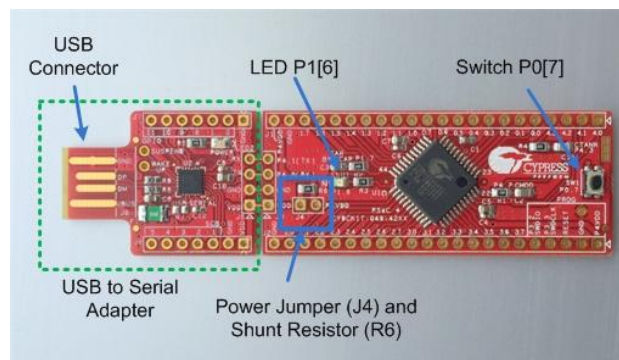


The CY8CKIT-044 and CY8CKIT-046 are similar to the CY8CKIT-042 and have similar connections for the switch, LED, and UART Tx lines. Refer to the kit guides provided on the respective Cypress web pages for more information on these kits.

8.2 CY8CKIT-049-42xx Kit (PSoC 4200)

These low-cost kits provide access to all the PSoC 4 pins and include a user LED and switch that are compatible with the CY8CKIT-042. Power can be provided from one of the “P4VDD” pins on the side or through the USB connector at the end. [Figure 14](#) shows the kit hardware.

Figure 14. CY8CKIT-049-42xx Kit Hardware



This kit does not have a built-in debugger, but it includes a USB-to-Serial adapter that can be used to bootload the project. The projects must first be modified to include a bootloadable Component to use this interface. If you have a PSoC MiniProg3 Program and Debug Kit (CY8CKIT-002), you can connect it directly to a user-installed connector at the bottom-right corner of the board. For more information on how to use the USB-to-Serial adapter for bootloading, download the kit documentation from the Cypress website.

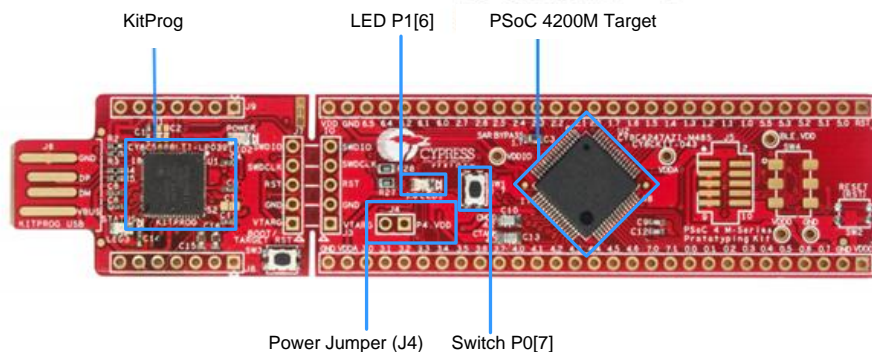
To measure the project current with the kit when the kit is powered from the USB port, remove zero-ohm resistor R6 and add pins or wires in J4 that can be connected to an ammeter or digital multimeter (DMM).

The USB-to-Serial adapter provides a bootloader interface as well as a serial port adapter to display the output of example project AN86233_DeepSleepADC.

8.3 CY8CKIT-043 Kit (PSoC 4200M)

This kit includes a built-in debugger and has a form factor similar to that of CY8CKIT-049. Power can be provided from one of the “P4VDD” pins on the side or through the USB connector at the end. To measure the project current with the kit when the kit is powered from the USB port, remove zero-ohm resistor R22 and add pins or wires in J4 that can be connected to an ammeter or DMM. [Figure 15](#) shows the kit hardware.

Figure 15. CY8CKIT-043 Hardware



This board also contains the LED and switch required to exercise these examples. Another feature it includes is a USB-to-Serial port adapter. You can use your favorite terminal emulator to monitor the serial output of the AN86233_DeepSleepADC project without making any special hardware connection. The baud rate is 115.2 kbps, 8-bit data, and no parity. For more information on using the USB-to-Serial port feature, refer to the [KitProg User Guide](#).

8.4 Measuring Current with a DMM

When using a DMM to measure the current, it is important to know the value of the shunt resistor in the DMM. DMMs have one or more (shunt) resistors between the current inputs. These resistors can range from less than an ohm to more than 10 k Ω . There is no standard value for the shunt resistor between brands or even models from the same vendor. It is important to review your meter's manual and learn the value of the shunt resistor because there will always be a voltage drop across this shunt. This means that the PSoC device will not see the same voltage as that which you thought you were supplying. If the shunt resistor in your meter is 1 Ω or less, you will see only a few millivolts of drop when measuring PSoC 4 current, and you can ignore it. If the shunt resistor is 1 k Ω , which some vendors use for low-current measurements, a 1-mA current will result in a drop of 1 V! Also, when changing ranges, be careful that the DMM does not do a break-before-make, or the power will be cycled and your project will be reset.

As an alternative to relying on the DMM shunt, both kits include a place for a shunt resistor. The CY8CKIT-049-42xx has a zero-ohm shunt (R6). This can be replaced by a small resistor and measurement of the voltage across the shunt with a voltmeter. You can then easily determine the current. A shunt between 1 Ω and 100 Ω should work well for most applications. The CY8CKIT-042 also has place for a shunt resistor (R6) next to J13.

8.5 Approximating Power Consumption

The device datasheet and the Component datasheets provide sufficient information to estimate the power consumption for a given project. To simplify this process, a spreadsheet has been provided that includes typical power requirements for a wide range of internal components. This spreadsheet, *PSoC4_Power_Estimator.xlsx*, is located on the [AN86233 page](#). Because 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 you read the “Instructions” tab before entering your data.

9 Summary

Power consumption can make the difference between a good idea and a successful design. By taking advantage of the many power-saving features available in PSoC 4, you can optimize your design and ensure that it consumes the lowest amount of power.

10 Related Application Notes

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

- [AN79953 – Getting Started with PSoC 4](#)
- [AN85951 – PSoC 4 CapSense Design Guide](#)
- [AN86439 – PSoC 4 – Using GPIO Pins](#)
- [AN90114 – PSoC 4000 Family Low-Power System Design Techniques](#)
- [AN77900 – PSoC 3 and PSoC 5LP Low-Power Modes and Power Reduction Techniques](#)

About the Authors

Name: Kannan Sadasivam

Title: Applications Engineer Staff

Background: Kannan earned a bachelor's degree in electronics and communications from Model Engineering College. He has written a number of articles on mixed-signal design and embedded systems.

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.

Document History

Document Title: AN86233 – PSoC® 4 Low-Power Modes and Power Reduction Techniques

Document Number: 001-86233

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3973447	MAXK	04/18/2013	New application note
*A	3996246	MKEA	05/09/2013	Updated links Updated example project section
*B	4703186	MEH	04/09/2015	Replaced example project with three other projects Added Table 1 of typical current and wakeup times Made changes to Table 2 Updated content on UnFreezelo() and Freezelo() APIs Updated low-power APIs Updated the CY8CKIT-049-42xx Kit section Included power calculation spreadsheet Many minor changes Sunset update
*C	4803916	MEH	06/19/2015	Clarified retention of UDBs when exiting Deep-Sleep and Hibernate in Table 2. Cleaned up several minor typos
*D	4890073	RNJT	08/24/2015	Updated the document for PSoC 4200L Updated the power calculation spreadsheet
*E	5068783	RNJT	12/31/2015	Updated the projects to PSoC Creator 3.3 SP1

Worldwide Sales and Design Support

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

Products

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

PSoC[®] Solutions

psoc.cypress.com/solutions

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

Cypress Developer Community

[Community](#) | [Forums](#) | [Blogs](#) | [Video](#) | [Training](#)

Technical Support

cypress.com/go/support

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



Cypress Semiconductor Phone : 408-943-2600
198 Champion Court Fax : 408-943-4730
San Jose, CA 95134-1709 Website : www.cypress.com

© Cypress Semiconductor Corporation, 2013-2015. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

This Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.