AN90114

PSoC® 4000 Family Low-Power System Design Techniques

Authors: Ranjith M and Nidhin M S

Associated Project: Yes

Associated Part Family: CY8C40xx

Software Version: PSoC Creator™ 4.1

Related Application Notes: For a complete list of the application notes, click here.

To get the latest version of this application note, or the associated project file, please visit http://www.cypress.com/go/AN90114.

AN90114 introduces the low-power modes offered by the PSoC® 4000 family and teaches the methods to design low-power systems. Major topics include device power modes and system-level power reduction techniques. This application note also includes an example project to demonstrate a low-power CapSense® system design.

Contents

1 Introduction

This application note introduces the low-power modes available in the PSoC 4000 (CY8C40xx) family of devices and the techniques to reduce overall system power consumption. PSoC 4000 is a cost-optimized, entry-level PSoC 4 device that offers 32-bit performance at 8-bit prices, along with PSoC value-added features such as CapSense and programmable peripherals. See AN86233 – PSoC 4 Low-Power Modes and Power Reduction Techniques to learn about the power modes and power reduction techniques for all the other devices in the PSoC 4 family.

This document assumes that you are familiar with the PSoC 4 architecture, PSoC 4 CapSense, and application development for PSoC 4 using the Cypress PSoC Creator™ Integrated Design Environment (IDE). For an introduction to PSoC 4, read AN79953 – Getting Started with PSoC 4. If you are new to PSoC 4 CapSense, refer to the PSoC 4 CapSense Design Guide. If you are new to PSoC Creator, see the PSoC Creator home page.

The initial sections of this application note cover the PSoC 4000 power modes and power reduction techniques in detail. If you are not interested in theoretical discussions and want to evaluate the example project, proceed to the Example Project: Low-Power CapSense With Periodic Scan.

2 Power Modes

This section explains in detail the power modes available in PSoC 4000 devices. The operation and power consumption of these power modes are slightly different from those available in other PSoC 4 families because of architectural differences in the PSoC 4000 power subsystem.

From here on, this document will use the term “PSoC 4” to refer to the PSoC 4000 family of devices, unless otherwise specified.

PSoC 4 features three modes of operation: Active, Sleep, and Deep-Sleep. These power modes have different power consumption and peripheral availability, as shown in Table 1.

2.1 Active

Active is the primary operating mode of the device and is the default mode at boot. In this power mode, the high-frequency internal clock (HFCLK), which is derived from the internal main oscillator (IMO) is on, and the CPU and all the peripherals are operational unless they are specifically disabled by firmware. Active mode typically consumes more power than other modes, as all the blocks in the chip are operational.

Active mode can transition to any other mode. Any valid interrupt or reset event from the other modes returns the PSoC 4 device to Active mode.

The device consumes approximately 3.2 mA in Active mode with the CPU running at 12 MHz. For more detailed specifications on power, see the PSoC 4000 Family datasheet.

2.2 Sleep

Sleep mode is a low-power mode similar to Active mode in that all the peripherals in the device are functional. The clock to the CPU, system clock (SYSCLK), is off during Sleep mode. The CPU can wake up from Sleep mode on interrupts. In this mode, all peripherals such as CapSense or I^2C interface can operate while the CPU is off to reduce the device power consumption. The device consumes approximately 1.4 mA in Sleep mode at 12-MHz HFCLK with I^2C wakeup and watchdog timer (WDT) enabled.

2.3 Deep-Sleep

Deep-Sleep is the lowest power mode available in the PSoC 4 device. In this mode, almost all peripherals including CPU, IMO, CapSense, and TCPWM are either off or in retention state.

Basic peripherals such as I^2C, WDT, and GPIO, are operational in this mode. The I^2C block can wake up the device from Deep-Sleep mode on an address match, depending on the configuration. The internal low-speed oscillator (ILO) clock source is also functional in Deep-Sleep mode. A WDT implemented in the clock block allows periodic wake up from Deep-Sleep. The device consumes about 2.5 µA in Deep-Sleep mode. Refer to the device datasheet for more detailed power numbers.

Table 1. PSoC 4 Power Mode Peripheral Availability

Peripheral	Active	Sleep	Deep-Sleep
CPU	On	Retention*	Retention
SRAM	On	Retention	Retention
SYSTICK timer	On	On	Off
IMO	On	On	Off
ILO	On	On	On
WDT	On	On	On
TCPWM	On	On	Off
I^2C	On	On	On**
CapSense	On	On	Off
GPIO	On	On	On
Power consumption***	~3.2 mA at 12 MHz	~1.4 mA at 12 MHz	~2.5 µA

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

**Only the I²C slave mode is available in the Deep-Sleep power mode.

*** Refer to the device datasheet for more detailed power numbers.

2.4 Mode Entry and Wakeup Sources

PSoC Creator provides a set of functions called “application programming interfaces (APIs)” that abstracts the register-level operations to enter the low-power modes. Table 2 shows the different APIs to enter the low-power modes explained in the previous section. The table also identifies wakeup sources to exit the low-power mode and return to Active mode.

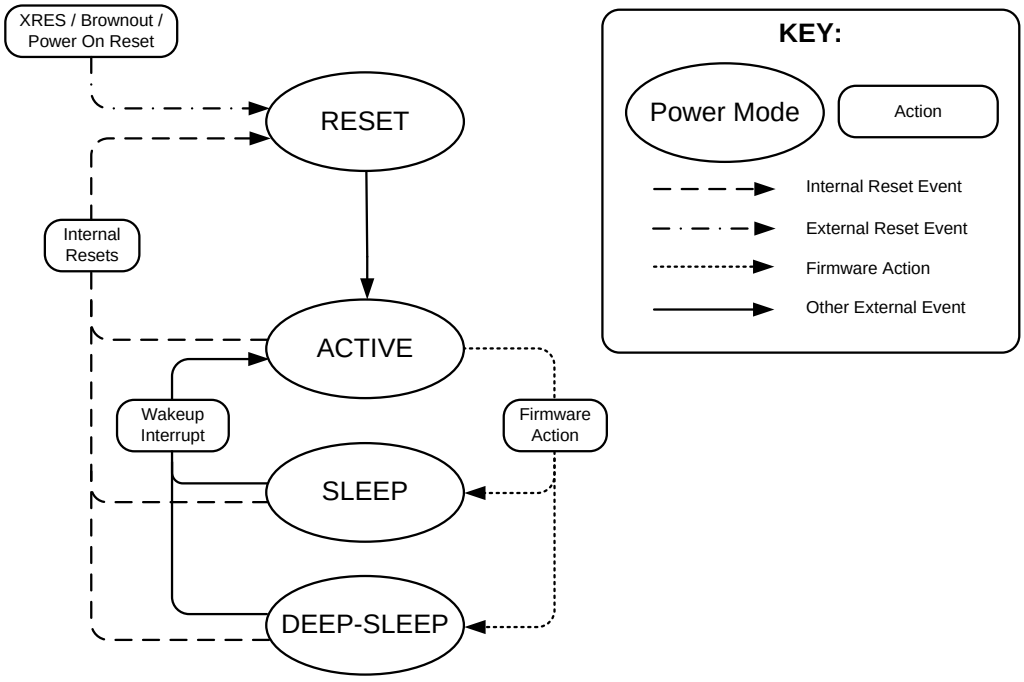
Figure 1 shows possible transitions between power modes.

See the PSoC 4000 Family Architecture TRM for more details on these low-power modes and wakeup sources. Refer to AN90799 – PSoC 4 Interrupts for details on the interrupts available in PSoC 4.

Table 2. Mode Transition Details

Low-Power Mode	API to Enter Low-Power Mode	Wakeup Source	Wakeup Action
Sleep	CySysPmSleep()	Any interrupt source	Interrupt
		Any reset source	Reset
Deep-Sleep	CySysPmDeepSleep()	GPIO interrupt	Interrupt
		I²C address match	Interrupt
		Watchdog timer	Interrupt/Reset
		XRES (external reset pin), brownout	Reset

Figure 1.Power Mode Transitions



3 Low-Power System Design Considerations - General

In many applications, you can gain additional current reductions by the proper usage of PSoC 4 peripherals. This section presents these techniques.

3.1 Turn Off Unused Components

The easiest way to reduce power in active mode is to turn off unused components.

Any PSoC Creator Component that can be disabled in Active or Sleep mode has a `_Stop()` function in its API. This function immediately halts all operations 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. */
Component_Stop();
```

Restart a Component by calling its Start function:

```
/* Start the Component. */
Component_Start();
```

Any PSoC Creator 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 some cases, the `_Sleep()` function does nothing but call `_Stop()`.

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

/* Sleep the Component. */
Component_Sleep();
```

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.

```
/* Wake the Component. */
Component_Wakeup();
```

`_Sleep()` and `_Stop()` functions both result in the same amount of power savings. The difference is whether the Component needs to resume from exactly where it left off.

3.2 GPIOs in Low Power

GPIOs can continue to drive when the PSoC 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 operations. If holding a digital output pin at logic 1 or 0 is the best option, then use the Pin Component's `_Write()` API function to set it.

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

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

```
/* Set MyPin to Alg Hi-Z for low power. */
Pin_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.

3.3 Is Your Debug Interface Running?

PSoC 4 supports on-chip debug. You may observe higher current consumption than you expect while in debug mode. This is normal, because the programming and debug interfaces remains active in all low-power modes.

3.4 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 current consumption. This technique can be applied to the Cortex-M0 CPU and digital components such as TCPWM and I^2C.

However, in some cases, running the CPU faster can actually result in a lower average current consumption. An example is in applications in which the CPU spends most of its time in Deep-Sleep mode and periodically wakes up to Active mode to perform operations. This technique of dynamically switching the power modes is explained in the next section.

In such applications, a higher CPU clock reduces the time spent in the Active mode, which in turn reduces the average power consumption.

3.5 Dynamically Switching the Power Modes

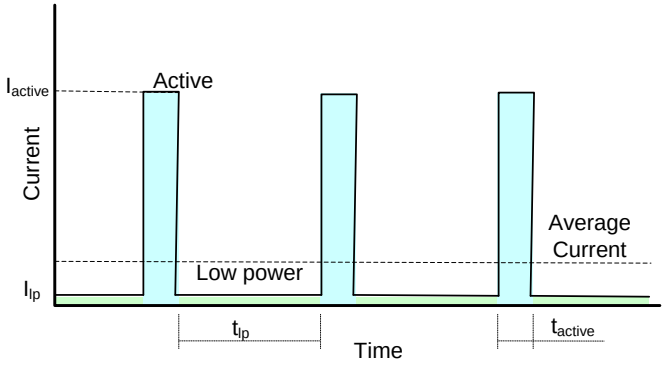
This section explains the techniques to reduce the average power consumption of the device by dynamically switching device power modes. Average power consumption is important in battery-powered applications to get long battery life.

In certain voltage sources such as coin cells, peak power consumption is also equally important due to the output impedance of the source. In such cases, you should also reduce the peak power using the techniques mentioned in previous sections.

A system optimized for low average power spends most of the time in the lower-power modes, while ensuring reliable operation. You can reduce the average power of PSoC 4 by ensuring that it is in the lowest-power mode (Deep Sleep) during most of its operation. To perform the operations required by the system, the device must wake up from Deep-Sleep mode. Typical applications use one of the following methods.

1. Periodically wake up and perform the required operations and remain in the lowest-power mode for most of the time (See Figure 2). Watchdog timer is used to generate periodic wake-up events.
2. Remain in the lowest power mode unless an external event such as a GPIO interrupt or an I^2C address match needs attention.

Figure 2. Timing Diagram of a Typical Low-Power System



If the device periodically wakes up from a low-power mode and spends a fixed amount of time in Active mode, you can calculate the approximate average current consumed by the device as:

$$\text{Average current} = \frac{((t_{active} * I_{active}) + (t_{lp} * I_{lp}))}{t_{active} + t_{lp}} \text{ -- (1)}$$

t_{active} = Time spent in Active mode

I_{active} = Active mode current

t_{lp} = Time spent in low-power mode

I_{lp} = Low-power mode current

You can use the following wake up sources to wake up the device from Deep Sleep.

3.5.1 Watchdog Timer

The watchdog timer is a free-running, 16-bit up-counter, which is clocked by the 40-kHz ILO. The counter has a 16-bit match register and 4-bit ignore MSB bit fields associated with it. The WDT can generate an interrupt on every match, but it does not reset the count to 0. You can configure the WDT to reset the device if three consecutive interrupts are unserviced. To avoid a device reset, the firmware must clear the WDT interrupt before generation of the three interrupts. The WDT in reset mode provides latch-up protection for the device.

You can also configure the WDT as a wakeup timer in applications where high accuracy of the timer is not required. Use this WDT capability to put the PSoC 4 device into Deep-Sleep or Sleep for a specific amount of time and periodically wake up the device to perform the required operations.

3.5.2 GPIO Interrupt

PSoC 4 can wake up from an interrupt generated by the rising edge, the falling edge, or both the edges of an input digital signal to a GPIO. Using the GPIO interrupt, a host processor or a user button can wake up the PSoC device by providing a trigger on the GPIO. GPIO-based wakeup consumes the lowest possible current in Deep-Sleep mode.

3.5.3 I^2C Address Match

The PSoC 4 device has a fixed-function I^2C block that can operate as a master or a slave. When the I^2C is configured as a slave, it can wake up the PSoC device from Deep-Sleep mode. This method allows a remote master to wake up the target PSoC device through I^2C address match. PSoC can then complete the required operations in Active mode, send the results to the I^2C master, and return to Deep-Sleep mode.

I^2C communication allows the device to be in any power mode regardless of the power mode or operating mode of the host controller. No sync is needed between the host and the slave (PSoC 4). The slave can wake up only when needed, complete the action, and go back to Deep Sleep. It also helps to avoid unwanted wake up events when the host is communicating with other devices on the I^2C bus.

Refer to AN90799 – PSoC 4 Interrupts for details on how to configure these interrupt sources.

4 Low-Power System Design Considerations - CapSense

CapSense is one of the most important blocks in the PSoC 4 device. CapSense UI may have stringent power consumption regulations as the UI is associated with a battery-powered device, in most cases.

This section explains techniques to achieve maximum efficiency in terms of power, while helping to ensure reliable and glitch-free operation.

4.1 Use Deep-Sleep Mode Between Scans

CapSense sensors are typically scanned at a certain frequency depending on the required touch response. You can reduce the average power by ensuring that the system is in the lowest power mode (Deep-Sleep) between the CapSense scans, as described in the Dynamically Switching the Power Modes section. Following techniques show how to use Deep-Sleep mode while proving good touch response.

4.1.1 Periodic Scan Using WDT

You can use the WDT to put the PSoC 4 device into Deep-Sleep for a specified amount of time and occasionally wake up to scan the CapSense sensors. This technique is useful since human interaction with UI systems is generally slow, and the CapSense inactivity during the Deep-Sleep interval will not be recognizable by the user.

The frequency at which the system wakes up for a CapSense application is determined by the human response time. Normally, a valid finger touch can last from several tens of milliseconds to several hundred milliseconds. For such a system, PSoC 4 need not wake up every millisecond and scan for a finger touch.

4.1.2 Using Proximity Sensor or Ganged Sensors

You can use a dedicated proximity sensor or gang the existing sensors to create a proximity sensor to reduce the power consumption further as follows. CapSense component allows you to combine multiple sensors to create a ganged sensor.

The device remains in Deep-Sleep mode for a specific amount of time and wakes up occasionally. On wakeup, the device combines all the sensors and scans them together like a proximity sensor or uses a separate proximity sensor to detect a finger in proximity. If proximity is detected, the device remains in the Active mode and scans individual sensors to determine the source of touch. Otherwise, the device goes back into Deep-Sleep, reducing the average power consumption.

When ganged sensor or the proximity sensor is being scanned, users may not frequently operate the panel. Therefore, the device may scan sensors less frequently. Upon first touch, the device can switch to a different scan frequency where the sensors are scanned at a faster rate.

There are advantages and disadvantages when using a dedicated proximity sensor instead of ganging the sensors together and scanning.

Ganging multiple sensors together can significantly increase the parasitic capacitance, thereby increasing the required resolution and scan time. This, in turn, increases the average power consumption. If the parasitic capacitance exceeds 35 pF, tuning the sensors to detect finger proximity reliably becomes difficult. The dedicated proximity sensor, on the other hand, is reliable and tuned easily.

The dedicated proximity sensor can surround the CapSense sensors and give a proximity range of approximately 1.5 to 2 times the diameter of the proximity loop. Using a dedicated proximity sensor consumes an additional pin. Ganging the sensors removes that requirement.

You cannot use the proximity sensing technique if the sensors are physically at a great distance from one another. In this case, ganging the sensors together is the best alternative for wakeup and the sensor scan approach. Using this approach instead of scanning all the sensors at full resolution during each wakeup can significantly reduce the system's power consumption.

The Example Project: Low-Power CapSense With Periodic Scan demonstrates the technique of using a ganged sensor to reduce the power consumption of a periodically scanned CapSense system.

4.2 Enter and Remain in Sleep Mode During CapSense Scan

CapSense scan is non-blocking. CPU intervention is not required between the start and end of a CapSense scan. You can put the device into Sleep mode after initiating a scan to save power. When the CapSense Sigma Delta (CSD) hardware completes the scan, it generates an interrupt to return the device to Active mode.

CapSense is a high-sensitivity analog system. Therefore, in PSoC 4000 devices, sudden changes in the device current may increase the noise present in the CapSense raw counts. Any change in CPU activity, including mode transitions, can cause noise in the raw counts. To avoid this noise, you should always use Sleep mode while a CapSense scan is in progress. If an interrupt other than CapSense causes a wake-up from the Sleep mode during a CapSense scan, discard the scan results and initiate another scan. The Example Project: Low-Power CapSense With Periodic Scan illustrates this technique.

4.3 Reduce Scan Time

Reducing the scan time reduces power consumption, as the device can spend more time in low-power modes.

A sensor's scan time depends on the required sensitivity and the sensor's parasitic capacitance. Refer to the PSoC 4 CapSense Design Guide to learn how to reduce the parasitic capacitance and hence the scan time.

You can use manual tuning to optimize the scan time and get strict control over CapSense performance. The example project in this application note use manual tuning. The CapSense Design Guide provides guidance to tune CapSense manually.

5 Example Project: Low-Power CapSense With Periodic Scan

An example project accompanies this application note, demonstrating the use of the concepts presented in this document to reduce the average power consumption of a CapSense system. This section provides an overview of the example project and methods to test the project and validate the results.

The following are required for testing the example project.

- CY8CKIT-040: This is the PSoC 4 Development Kit with the PSoC 4000 part. The kit also comes with a touchpad shield.
- PSoC Creator software installed on your PC: You can download the latest version of PSoC Creator from the PSoC Creator web page.
- Ammeter with averaging capability for measuring power consumption

This project uses the touchpad shield available with the CY8CKIT-040 to simulate a wakeup on touch feature. Figure 3 shows the firmware flowchart of this project.

PSoC 4 spends most of the time in Deep-Sleep mode and periodically (at 200-ms intervals) wakes up to check for a finger touch. To reduce the time spent by the PSoC device in Active mode, some of the touchpad sensors are ganged together and scanned at a low resolution immediately after the device wakes up from Deep-Sleep mode. If the PSoC device detects a touch or the proximity of finger, it scans all the touchpad sensors one by one to determine the exact finger position. PSoC continues the scan at 30-ms periodic intervals to maintain a good touch response and spends the time between two consecutive scans in Deep-Sleep mode. If the touchpad widget is inactive for 100 consecutive scans (about 3 seconds), the device halts the scan of the touchpad sensors and resumes the periodic scan of the proximity sensor at 200-ms intervals.

This project uses Sleep mode during CapSense scans to reduce the power consumption and to help ensure reliable CapSense operation. If an interrupt other than CapSense causes a wakeup from Sleep mode during a CapSense scan, the firmware discards the scan results and initiates another scan. The project includes a GPIO interrupt input to test this feature. The firmware uses a set of flags set in the interrupt service routines (ISRs) in *main.c*, *CapSense_INT.c, isr_pin.c,* and *isr_WDT.c* to detect if a non-CapSense interrupt caused the wakeup. Refer to AN90799 – PSoC 4 Interrupts to learn more about the ISRs used in PSoC 4.

The touch data is encoded in the form of (x,y) coordinates, with the origin of the coordinate system starting at the top-left corner of the touchpad shield and extending positively right and down. The maximum value of the touch coordinate is (100, 100). The finger touch data is used to obtain the (x,y) coordinates of the touch on the touchpad.

This data is sent to the Bridge Control Panel (BCP) UI through the software UART.

Figure 3. Firmware Flowchart of Example Project

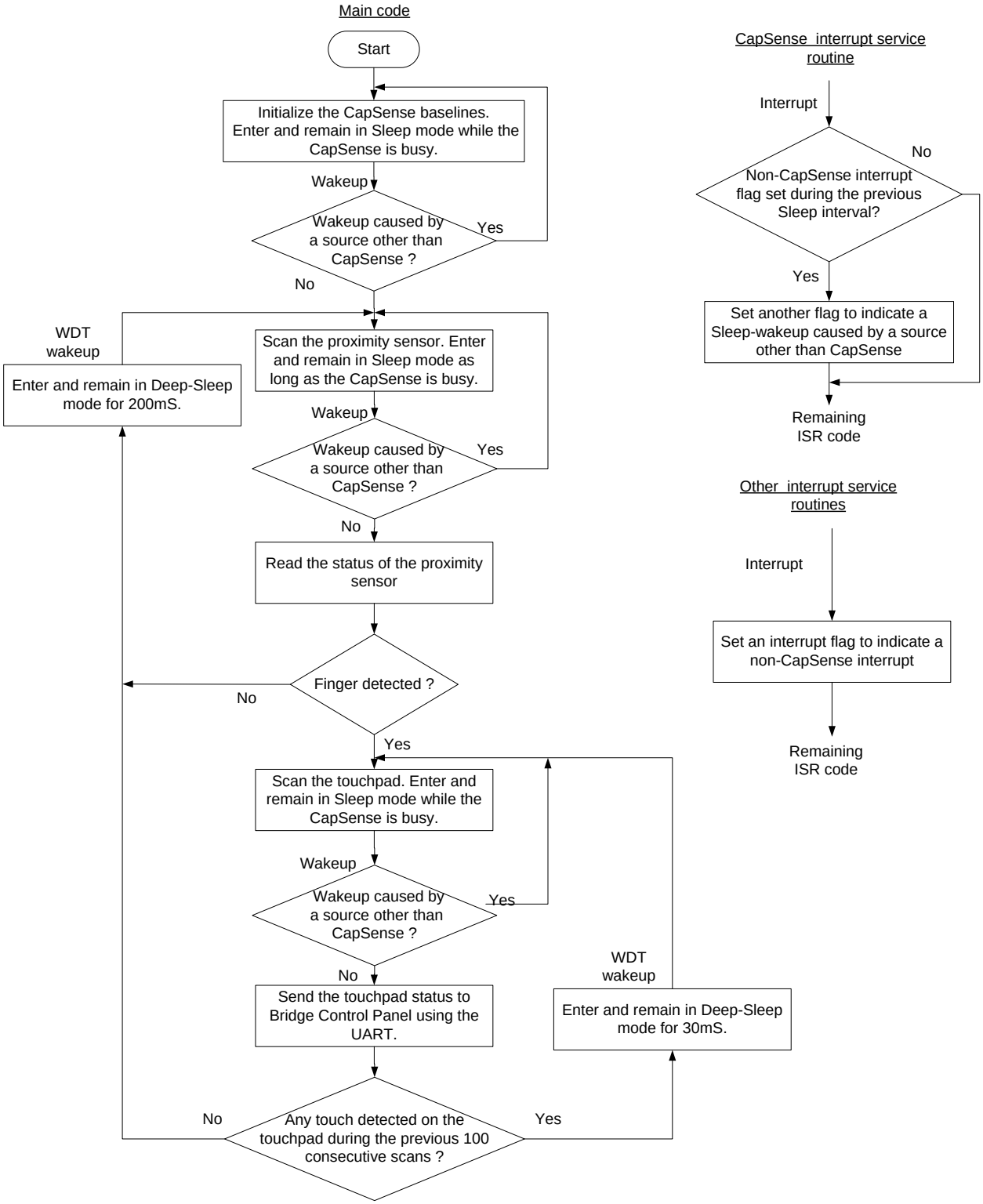
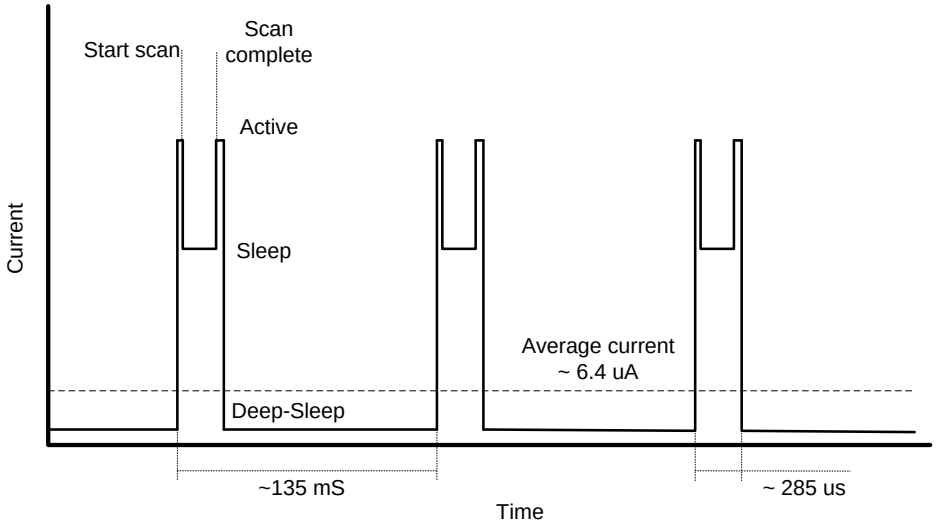


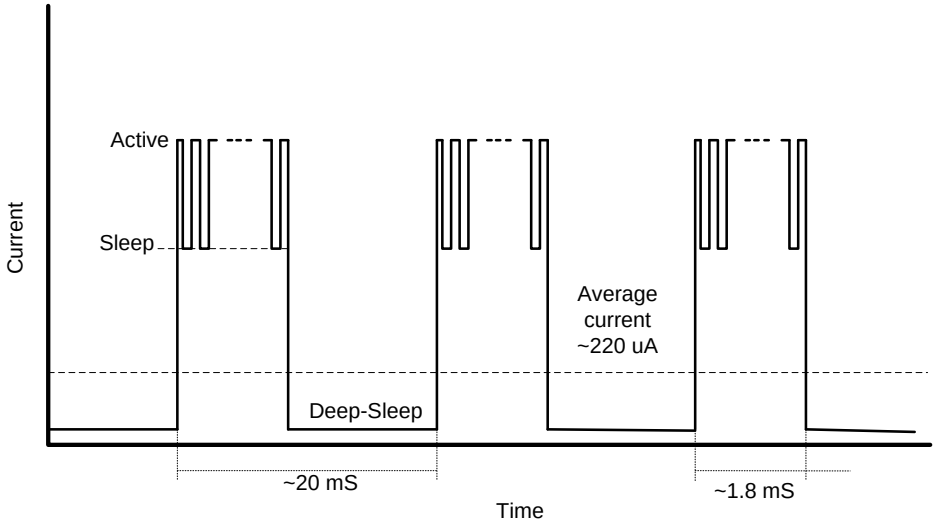
Figure 4 and Figure 5 show the timing diagrams for the current consumption in the two scenarios: periodic scan of the proximity sensor at 200-ms intervals when there is no touch, and periodic scan of the touchpad sensors at 30-ms intervals after the proximity sensor detects a finger touch.

Figure 4. Timing Diagram in the Absence of Touch



When there is no touch, the device spends approximately 2 ms in Active and Sleep modes, and approximately 200 ms in Deep-Sleep mode during one scan cycle, as Figure 4 shows. The average current consumption is approximately 6.5 µA.

Figure 5. Timing Diagram With Touch



Note: PSoC 4000 devices have ILO tolerance ranging from -50% to 100%. Therefore, Deep-Sleep times mentioned in Figure 4 and Figure 5 may vary from device to device. Also, if SmartSense is used, Active times vary depending on the sensor parasitic capacitance (Cp).

Since the touchpad comprises multiple row and column sensors, the device periodically switches between Active and Sleep modes during the scan of the touchpad widget. The average current consumption is approximately 220 µA when a finger is present on the touchpad.

The actual power consumption depends on the percentage of time a finger is present on the widgets. For example if this project spends 1% of time with finger touch and 99% of time on standby, the average current consumption becomes 28.5 µA.

Note: The touchpad widget has inherently high scan times due to the large number of sensors and high-parasitic capacitance. The power numbers can vary depending on the type of sensors used in a design. For example, if you use four buttons instead of the touchpad, the current consumption (for 1% finger touch) can be lower than 15 µA.

A UI with just one button can have a current consumption lower than 5 µA. In this case, extra widgets such as proximity sensors or ganged sensors are not required.

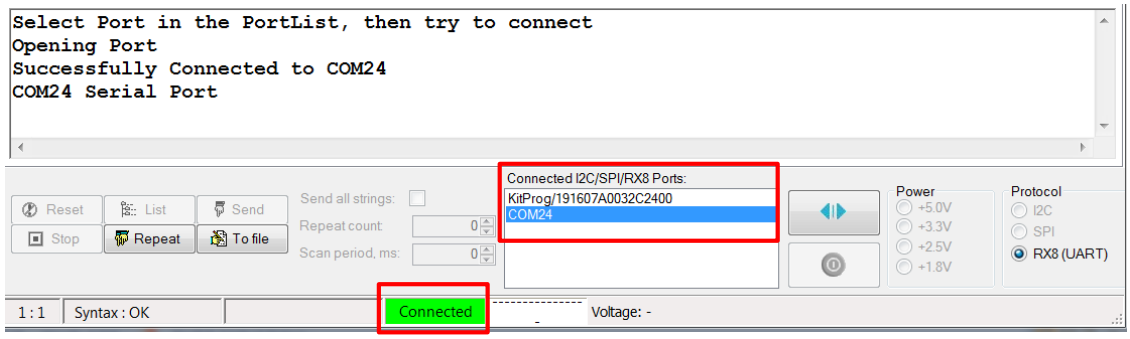
5.1 Testing the Project

You can test this project using CY8CKIT-040, which comes with a baseboard and a 6x5 trackpad shield. The trackpad is assigned a resolution of 100 in x and y directions. This means that CapSense can recognize 100 different points of touch in each direction (x and y).

PSoC sends the output data using a software UART component. You can use the Bridge Control Panel (BCP) software provided with PSoC Creator, along with the KitProg USB-UART bridge in CY8CKIT-040, to view the output.

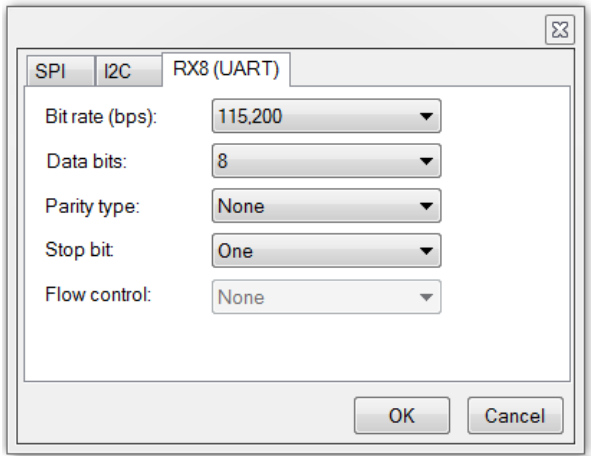
1. Open, build, and program the project “Low-Power CapSense with Periodic Scan” to CY8CKIT-040.
2. Open the BCP by choosing **Start menu > Programs > Cypress > Bridge Control Panel > Bridge Control Panel.**
3. Connect to the COM port with the highest number, displayed in the **Connected I2C/SPI/RX8 Ports,** as Figure 6 shows.

Figure 6. Connecting to the USB-UART Bridge



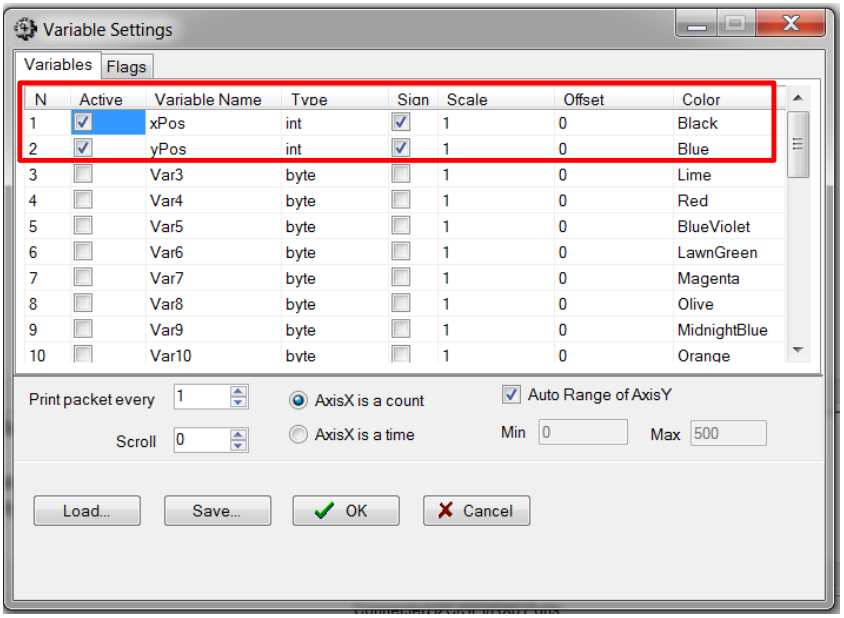
4. Open the UART configuration window from the menu by choosing **Tools > Protocol Configuration** and select the **RX8 (UART)** tab. Configure the parameters as Figure 7 shows.

Figure 7. UART Protocol Configuration



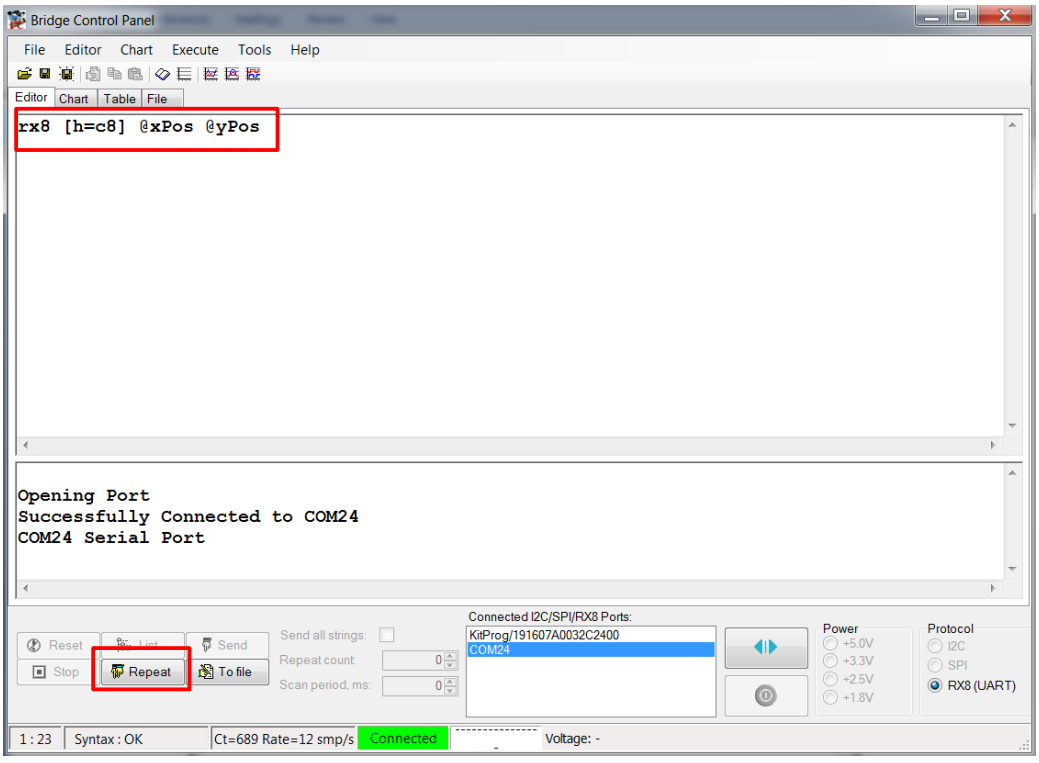
5. Open the variable configuration window from the menu by choosing **Chart > Variable Settings** and configure two variables, **xPos and yPos,** as Figure 8 shows.

Figure 8. Variable Configuration



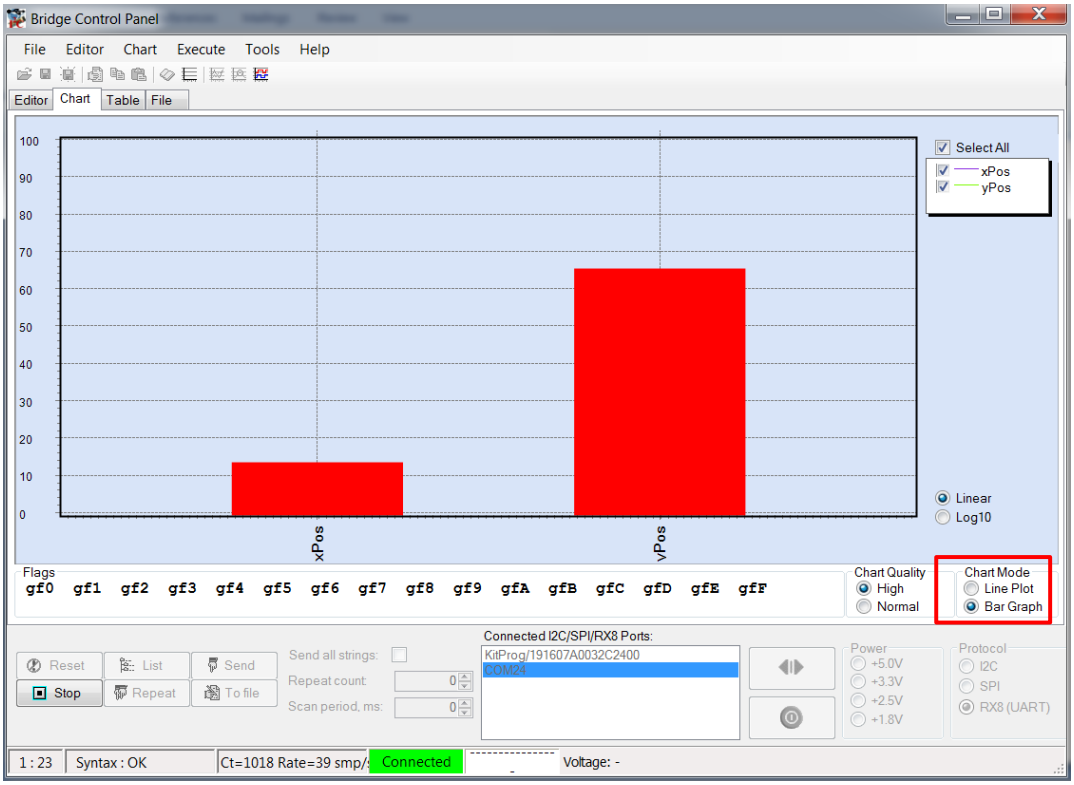
6. Type the command **rx8 [h=c8] @xPos @yPos** in the BCP **Editor** window, and click the **Repeat** button, as Figure 9 shows.

Figure 9. Entering and Sending a Command



7. Select the **Chart Mode Bar Graph** in the BCP **Chart** window, as Figure 10 shows. Touch the trackpad shield to observe the (x,y) coordinates of the finger touch on the BCP.

Figure 10. Viewing the Output



5.1.1 Kit Settings for Power Measurement

CY8CKIT-040 provides a power measurement jumper, J13, to measure the power consumed by the PSoC 4 device exclusively. You can remove this jumper and connect an ammeter across its terminals to measure the power consumed by the device.

5.2 Modifying the Example Project

You can easily modify the example project to suit your application. If you are using a custom PCB with a widget other than touchpad, follow these steps:

1. Open the CapSense Component, remove the touchpad widget, and add the required widgets. Refer to Section 5.1.2 of the PSoC 4 CapSense Design Guide to understand the configuration of CapSense widgets.
2. Reassign the pins in the PSoC Creator design-wide resources (cydwr) file according to the PCB you are using.
3. Retune the CapSense widgets including any ganged sensors in the project. Refer to Section 5.2 of the CapSense Design Guide to understand how to tune CapSense widgets.
4. Replace CapSense widget APIs according to the widgets used. See the CapSense Component datasheet (right click on the Component and select “Open Datasheet”) for a list of APIs related to each widget.

Follow these steps to use a dedicated proximity sensor instead of a ganged sensor:

1. Open the CapSense Component, select the proximity widget, and set the number of dedicated proximity elements to ’1’.. Uncheck all the other sensors that are included in the proximity widget. Refer to sections 5.1.2 and 5.1.3 of the CapSense Design Guide for more details.
2. Reassign the pins in the PSoC Creator design-wide resources (cydwr) file according to the PCB you are using.
3. Retune the proximity sensor.

Follow these steps to interface the project with a host that uses an I^2C master:

1. Go to the “I^2C” tab in the TopDesign window and configure the I^2C component according the requirements of the host that you are using.

2. Change the value of “I2C_ENABLE” macro in the *main.h* file to ’1’.

   The project is configured to interrupt the host after a scan is finished. The host can then read the touch coordinates as two 8-bit integers, from the I^2C buffer. You can modify the *main.c* according to the requirements of the host used.

   Note: Unlike other interrupts, the I^2C component uses the I2C_Slave_SetCustomInterruptHandler() API for entering user code into the I^2C ISR. Refer to the I^2C component datasheet (right click on the Component and select “open datasheet”) for more details.

   You can also refer to the I^2C example projects (right click on the Component and select “find example project”) for firmware examples.

3. Reassign the pins in the PSoC Creator design-wide resources (cydwr) file if required. Make sure that host interrupt and I^2C pins are properly connected to the host.

6 Summary

AN90114 introduced the low-power modes offered by the PSoC 4000 family and explained the methods to design low-power systems. Major topics included device power modes and system-level power reduction techniques.

Finally, the example project demonstrated how to achieve very low-power consumption while maintaining a good touch response in CapSense based applications.

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 help ensure that it consumes the lowest amount of power possible.

7 Related Application Notes

- AN86233 – PSoC® 4 Low-Power Modes and Power Reduction Techniques
- AN77900 – PSoC® 3 and PSoC 5LP Low-Power Modes and Power Reduction Techniques
- AN79953 – Getting Started with PSoC® 4
- AN90799 – PSoC® 4 Interrupts
- PSoC 4® CapSense Design Guide

About the Authors

Name:	Nidhin M S
Title:	Applications Engineer
Background:	Nidhin graduated from GEC Thrissur with a Bachelor's degree in Electronics and Communication Engineering. His technical interests are analog signal processing, low-power design, and capacitive touch sensing.
Name:	Ranjith M
Title:	Applications Engineer Senior
Background:	Ranjith graduated from GEC, Thrissur with a Bachelor's Degree in Electronics and Communications Engineering.

Document History

Document Title: AN90114 - PSoC® 4000 Family Low-Power System Design Techniques

Document Number: 001-90114

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	4396284	RNJT	06/05/2014	New Application Note
*A	5700260	RNJT	04/26/2017	Updated logo and copyright
*B	5767361	VKVK	06/13/2017	Added a note to Figure 5 Updated template

Worldwide Sales and Design Support

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

Products

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

PSoC® Solutions

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

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