

## Getting Started with PSoC® 4 BLE

**Author:** Krishnaprasad M V (KRIS)

**Associated Project:** Yes

**Associated Part Family:** CY8C41x7-BL, CY8C42x7-BL

**Software Version:** PSoC Creator™ 3.1 SP1

**Related Application Notes:** For a complete list of the application notes, [click here](#).

To get the latest version of this application note, please visit  
<http://www.cypress.com/go/AN91267>.

AN91267 introduces you to PSoC® 4 BLE, an ARM® Cortex™-M0 based Programmable System-on-Chip (PSoC) that integrates a Bluetooth Low Energy (BLE) radio system. This application note helps you explore the PSoC 4 BLE architecture and development tools and shows how easily you can create a BLE design using PSoC Creator™, the development tool for PSoC 4 BLE. It also guides you to more resources to accelerate in-depth learning about PSoC 4 BLE.

## Contents

Introduction .....	2	ARM Cortex-M0 and Memory.....	46
More Information .....	2	Programmable Digital Peripherals.....	46
PSoC Creator .....	3	Programmable Analog.....	47
PSoC Creator Help .....	3	System-Wide Resources.....	48
Code Examples .....	4	Programmable GPIOs.....	50
PSoC 4 BLE Features .....	5	Appendix E: BLE Protocol.....	51
BLE Overview .....	6	Overview .....	51
BLE Link Establishment.....	7	Physical Layer (PHY) .....	51
GATT Data Format.....	7	Link Layer (LL) .....	51
BLE Profile .....	9	Host Control Interface (HCI) .....	52
BLE Component .....	10	Logical Link Control and Adaptation Protocol (L2CAP) .....	52
PSoC 4 BLE Development Setup.....	12	Security Manager (SM) .....	53
My First PSoC 4 BLE Design .....	14	Attribute Protocol (ATT) .....	53
About the Design .....	14	Generic Attribute Profile (GATT) .....	56
Prerequisites .....	14	Generic Access Profile (GAP) .....	57
Part 1: Configure the Design .....	15	Worldwide Sales and Design Support.....	61
Part 2: Write the Firmware.....	26		
Part 3: Program the Device .....	32		
Part 4: Test Your Design .....	34		
Summary .....	38		
Related Application Notes .....	38		
Appendix A: BLE Device Family Comparison .....	39		
Appendix B: Cypress Terms of Art .....	41		
Appendix C: Cypress BLE Development Tools .....	42		
CySmart Host Emulation Tool .....	42		
CySmart Mobile App .....	44		
Appendix D: PSoC 4 BLE Device .....	45		
Bluetooth Low Energy Subsystem (BLESS).....	45		

## Introduction

BLE is an ultra-low-power wireless standard defined by the Bluetooth Special Interest Group (SIG) for short-range communication. It features a physical layer, protocol stack, and application use cases, all designed and optimized for the lowest power consumption.

The Cypress PSoC 4 BLE device is a programmable embedded system-on-chip that integrates a BLE radio, programmable analog and digital peripherals, memory, and an ARM Cortex-M0 microcontroller on a single chip.

PSoC 4 BLE provides a cost-effective alternative to the combination of an MCU and an external BLE radio. In addition to reducing the overall system cost and size, the programmable analog and digital subsystems allow flexibility and fine-tuning of the design using PSoC Creator, the schematic-based design tool for designing applications for PSoC 4 BLE.

PSoC 4 BLE simplifies the RF board design with its integrated balun circuit, which reduces the number of external components required for antenna matching. The BLE protocol stack library is integrated with PSoC Creator and is free of cost. It can be easily configured using a simple graphical user interface (GUI), allowing you to jump-start your BLE design in minutes.

PSoC 4 BLE offers a current consumption of 150 nA while retaining the SRAM contents, programmable logic, and the ability to wake up from an interrupt. It consumes only 1.3 µA in Deep-Sleep mode while maintaining the BLE link active. A combination of these low-power modes and flexible analog and digital peripherals provides a single chip solution with best-in-class system power consumption for battery-operated BLE designs such as wearable fitness monitors and wireless sensor interfaces.

Cypress's capacitive touch-sensing feature in PSoC 4 BLE, known as CapSense®, offers unprecedented signal-to-noise ratio, best-in-class waterproofing, and a wide variety of sensor types such as buttons, sliders, track pads, and proximity sensors that are gaining popularity in wearable electronic devices such as activity monitors and health and fitness equipment.

In addition to PSoC 4 BLE, Cypress offers PRoC™ BLE, a fixed-function BLE device that focuses on human interface device designs such as keyboards, mice, and remote controls. See the [Getting Started with PRoC BLE](#) application note for details. See [Appendix A: BLE Device Family Comparison](#) for a comparison of the BLE device families from Cypress.

The Cypress PSoC portfolio also contains PSoC 1, PSoC 3, PSoC 4, and PSoC 5LP, which do not support BLE. These devices offer different system architectures and analog and digital peripherals. For more information, refer to the [Cypress Platform PSoC Solutions Roadmap](#).

## More Information

Cypress provides a wealth of data at [www.cypress.com](http://www.cypress.com) to help you to select the right PSoC device and quickly and effectively integrate it into your design. If you are a first-time user of Cypress's PSoC family of products, it is recommended that you read [Appendix B: Cypress Terms of Art](#) for a list of commonly used terms. For a comprehensive list of resources, see [KBA86521, How to Design with PSoC 3, PSoC 4, and PSoC 5LP](#).

Following is an abbreviated list for PSoC 4 BLE:

- **Overview:** [PSoC Portfolio](#), [PSoC Roadmap](#)
- **Product Selectors:** [PSoC 1](#), [PSoC 3](#), [PSoC 4](#), or [PSoC 5LP](#). In addition, [PSoC Creator](#) includes a device selection tool.
- **Datasheets:** Describe and provide electrical specifications for the [PSoC 41XX-BL](#) and [PSoC 42XX-BL](#) device families.
- **Application Notes and Code Examples:** Cover a broad range of topics, from basic to advanced level. Many of the application notes include code examples. PSoC Creator provides additional code examples—see [Code Examples](#).
- **Technical Reference Manuals (TRMs):** Provide detailed descriptions of the architecture and registers in each PSoC 4 BLE device family.
- **CapSense Design Guide:** Learn how to design capacitive touch-sensing applications with the PSoC 4 BLE family of devices.
- **Development Tools**
  - [CY8CKIT-042-BLE Bluetooth Low Energy \(BLE\) Pioneer Kit](#) is an easy-to-use and inexpensive development platform for BLE. This kit includes connectors for Arduino™ compatible shields and Digilent® Pmod™ daughter cards.
  - [CySmart BLE Host Emulation Tool](#) for [Windows](#), [iOS](#), and [Android](#) is an easy-to-use GUI that enables you to test and debug your BLE Peripheral applications.

See [Appendix C: Cypress BLE Development Tools](#) for an overview.

### • Technical Support

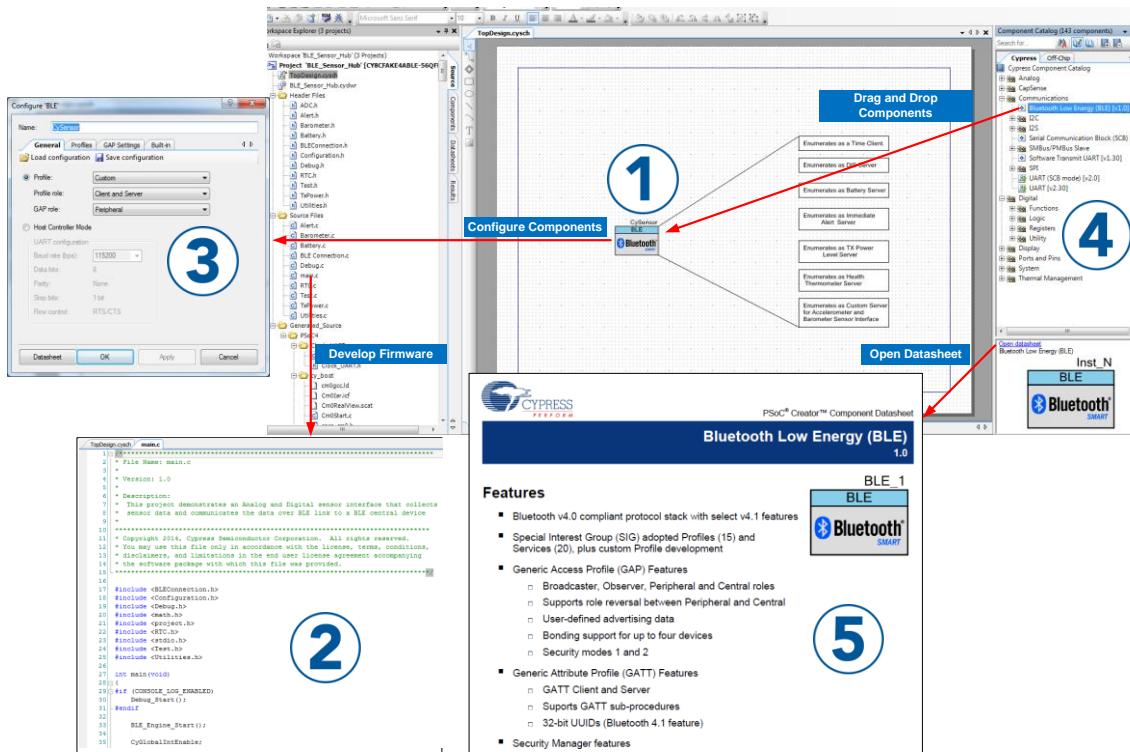
- [Frequently Asked Questions \(FAQs\)](#): Learn more about our BLE ecosystem
- [BLE Forum](#): See if your question is already answered by fellow developers on the [PSoC 4 BLE](#) and [PRoC BLE](#) forums.
- Cypress support: Still no luck? Visit our [support](#) page and create a [technical support case](#) or contact a [local sales representative](#). If you are in the United States, you can talk to our technical support team by calling our toll-free number: +1-800-541-4736. Select option 8 at the prompt.

## PSoC Creator

PSoC Creator is a free Windows-based Integrated Design Environment (IDE). It enables you to design hardware and firmware systems concurrently, based on PSoC 4 BLE and PROc BLE. As Figure 1 shows, with PSoC Creator, you can:

1. Drag and drop Components to build your hardware system design in the main design workspace.
2. Co-design your application firmware with the PSoC hardware.
3. Configure the Components using configuration tools.
4. Explore the library of more than 100 Components.
5. Review the Component datasheets.

Figure 1. PSoC Creator Schematic Entry and Components



## PSoC Creator Help

Visit the [PSoC Creator](#) home page to download and install the latest version of PSoC Creator. Then launch PSoC Creator and navigate to the following items:

- **Quick Start Guide:** Choose **Help > Documentation > Quick Start Guide**. This guide gives you the basics for developing PSoC Creator projects.
- **Simple Component example projects:** Choose **File > Open > Example projects**. These example projects demonstrate how to configure and use PSoC Creator Components.
- **Starter designs:** Choose **File > New > Project > PSoC 4 Starter Designs**. These starter designs demonstrate the unique features of PSoC 4 BLE.
- **System Reference Guide:** Choose **Help > System Reference > System Reference Guide**. This guide lists and describes the system functions provided by PSoC Creator.
- **Component datasheets:** Right-click a Component and select “Open Datasheet.” Visit the [PSoC 4 BLE Component Datasheets](#) page for a list of all PSoC 4 BLE Component datasheets.
- **Document Manager:** PSoC Creator provides a document manager to help you to easily find and review document resources. To open the document manager, choose the menu item **Help > Document Manager**.

## Code Examples

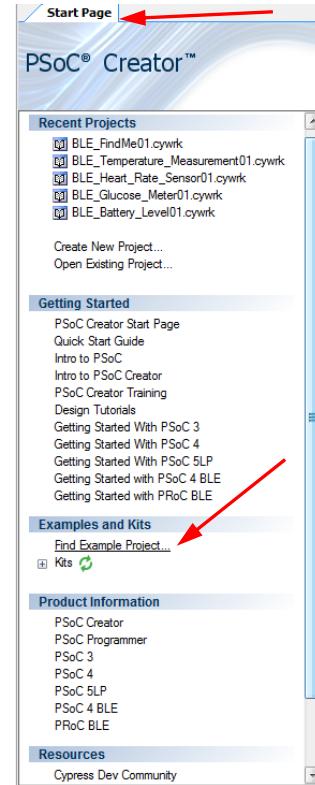
PSoC Creator includes a large number of code example projects. These projects are available from the PSoC Creator Start Page, as [Figure 2](#) shows.

Example projects can speed up your design process by starting you off with a complete design, instead of a blank page. The example projects also show how PSoC Creator Components can be used for various applications. Code examples and datasheets are included, as [Figure 3](#) shows.

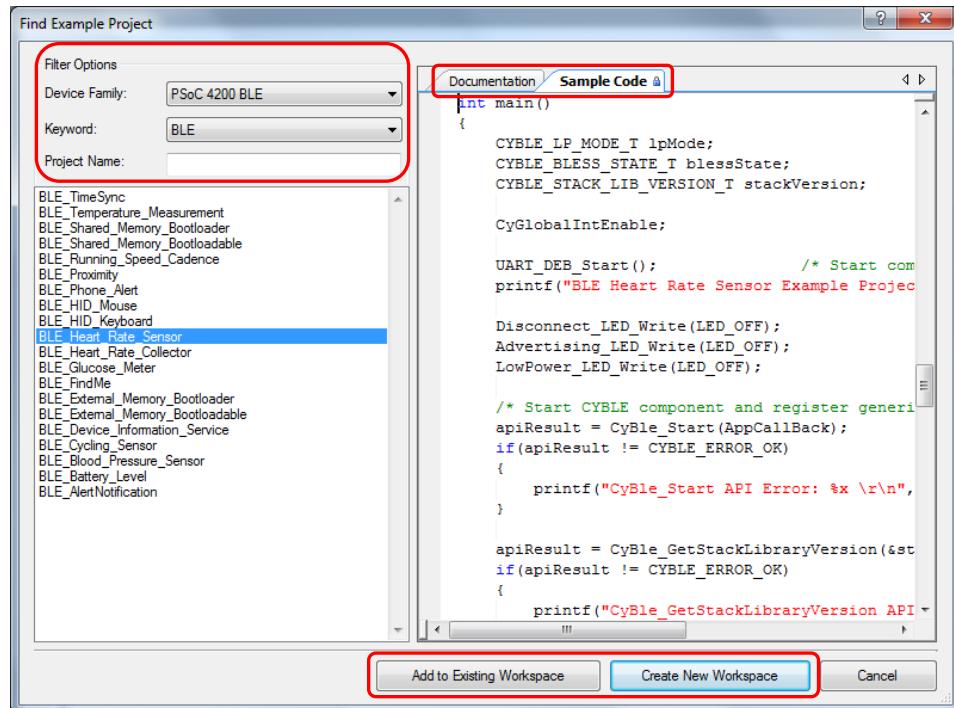
In the **Find Example Project** dialog shown in [Figure 3](#), you have several options:

- Filter for examples based on architecture or device family, that is, PSoC 4, PSoC 4 BLE, PRoC BLE, and so on; category; or keyword.
- Select from the menu of examples offered based on the **Filter Options**. There are more than 20 BLE example projects for you to get started, as shown in [Figure 3](#).
- Review the datasheet for the selection (on the **Documentation** tab)
- Review the code example for the selection. You can copy and paste code from this window to your project, which can help speed up code development.
- Or create a new project (and a new workspace if needed) based on the selection. This can speed up your design process by starting you off with a complete basic design. You can then adapt that design to your application.

[Figure 2. Code Examples in PSoC Creator](#)



[Figure 3. Code Example Projects with Sample Code](#)



## PSoC 4 BLE Features

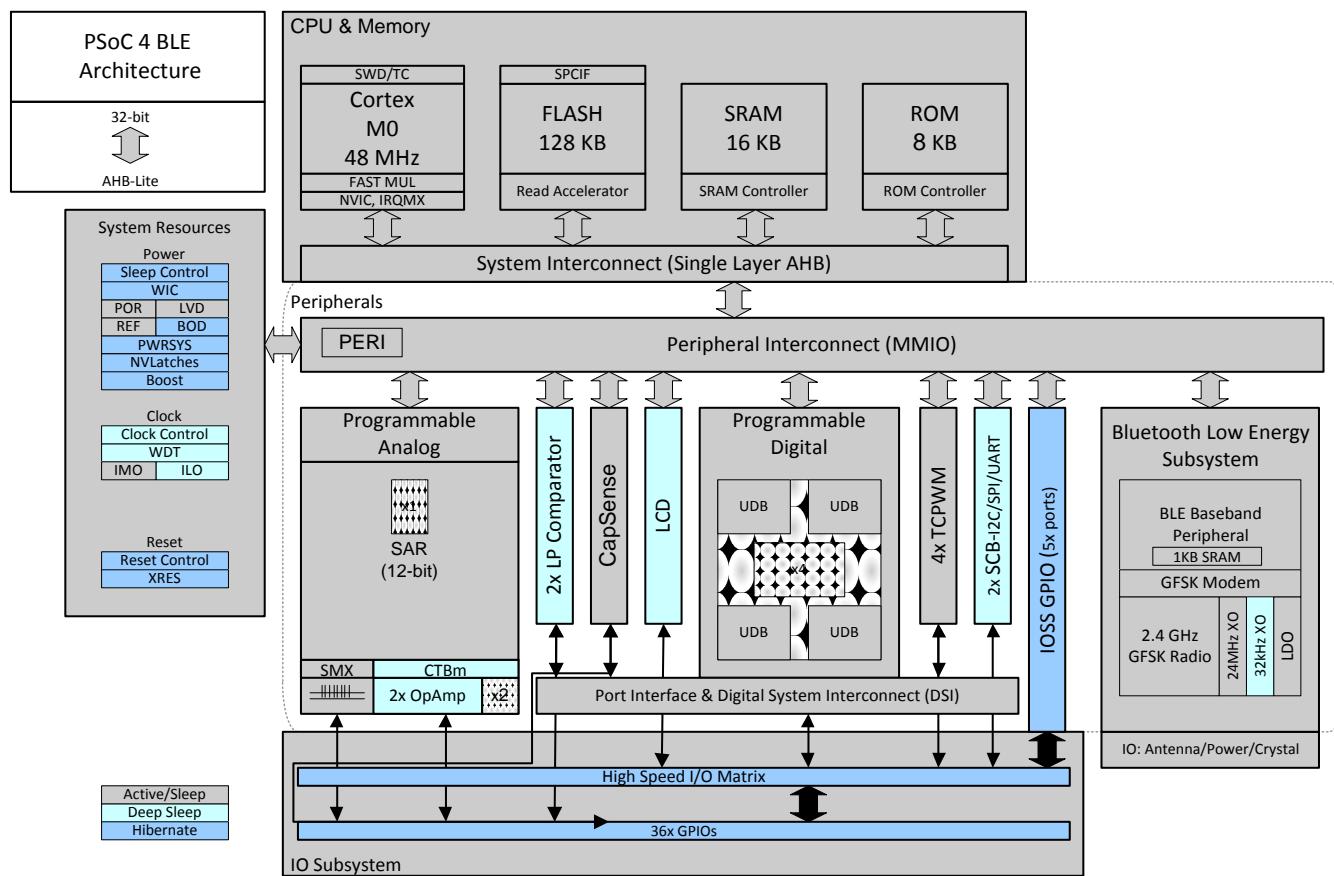
The PSoC 4 BLE product family is based on Cypress's PSoC 4100/4200 family of devices. PSoC 4 BLE features include a **CPU and memory subsystem**, a **BLE subsystem**, a **digital subsystem**, an **analog subsystem**, and **system resources**, as Figure 4 shows. Figure 4 shows the features available in the CY8C4247-BL family of devices. Subsets of these features are available in other device families; see [Table 2](#).

If you are familiar with the PSoC 4100/4200 family of devices, PSoC 4 BLE supports all the features supported by PSoC 4100/4200 and adds the following:

- BLE radio with link-layer hardware
- External crystal oscillator and watch crystal oscillator
- Four opamps (compared to two in the PSoC 4100/4200 family) that are active in device low-power (Deep-Sleep) mode
- 128 KB of flash and 16 KB of SRAM (compared to 32 KB of flash and 4 KB of SRAM in PSoC 4100/4200 family)

See [Appendix D: PSoC 4 BLE Device](#) for a brief description of the features. For in-depth information, see the PSoC 4 BLE family [datasheet](#), [TRM](#), and [application notes](#).

Figure 4. PSoC 4 BLE Architecture (CY8C4247-BL)



## BLE Overview

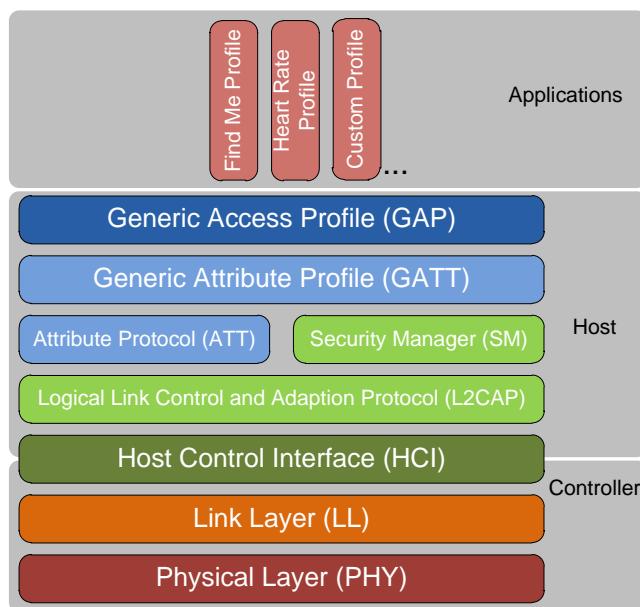
BLE or Bluetooth Smart™ is a low-power, short-range, low-data-rate wireless communication protocol that is defined by the Bluetooth SIG. As shown in [Figure 5](#), BLE has a layered protocol stack that is designed to efficiently transfer a small chunk of data with low power consumption, making it the de facto wireless protocol for battery-operated devices.

The BLE stack consists of the following:

- 2.4-GHz RF physical layer (PHY) with a 1-Mbps data rate
- Link Layer (LL) that defines the timing and packet format for PHY
- Host Control Interface (HCI) that links the hardware controller (PHY + LL) layer with the firmware host layer of the stack
- Logical Link Control and Adaptation Protocol (L2CAP) that acts as a packet assembly/disassembly and protocol multiplexer layer
- Attribute Protocol (ATT) that defines how the application data is organized and accessed
- Security Manager (SM) that provides a toolbox for secure data exchange over the BLE link
- Generic Attribute Profile (GATT) that defines methods to access data defined by the ATT layer.
- Generic Access Profile (GAP) that provides an application oriented interface which defines if the device acts as a BLE link master or slave and configures the underlying layers accordingly.

See [Appendix E: BLE Protocol](#) for a detailed description of the BLE protocol.

Figure 5. BLE Protocol Stack



To develop an application using BLE, you do not need a working knowledge of this complex protocol stack. Cypress provides an easy-to-configure, GUI-based BLE Component that abstracts the protocol complexity. To get started with BLE, it is sufficient to understand:

- BLE link establishment procedure
- Application data representation and abstraction
- Application requirements mapping to GAP and GATT layer configurations using Cypress's BLE Component GUI.

## BLE Link Establishment

To establish a BLE link between two devices—the BLE Pioneer Kit and a smartphone, for example—you need to understand the two GAP device roles:

- **GAP Peripheral:** A device that advertises its presence and accepts connection from a GAP Central device. A BLE Pioneer Kit that implements a heart-rate measurement function is an example of a GAP Peripheral device.
- **GAP Central:** A device that scans for advertisements from GAP Peripherals and establishes a connection with them. A smartphone that connects to a heart-rate measurement device is an example of a GAP Central device.

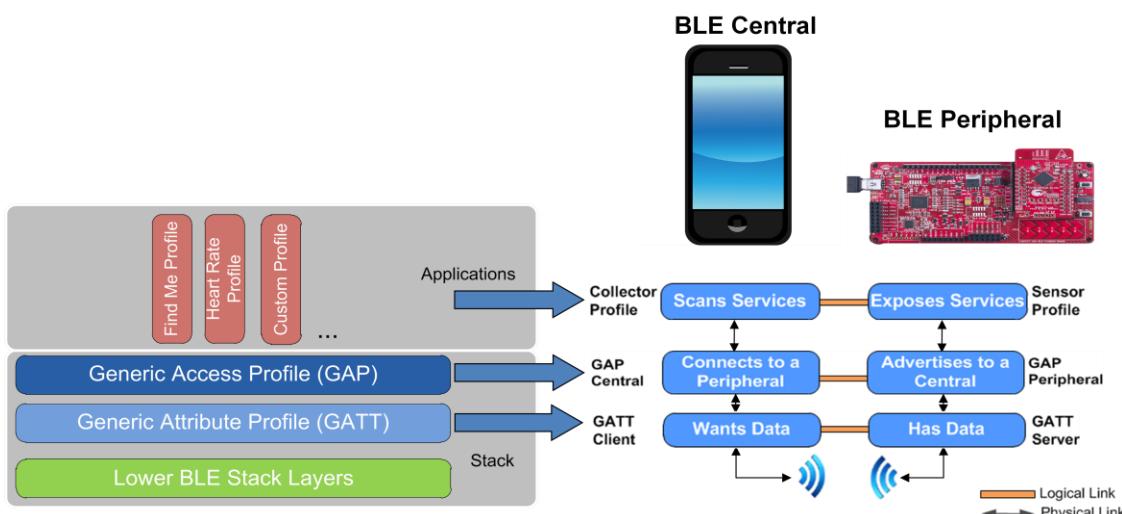
After the Central device establishes a connection with the Peripheral, both devices are said to be connected over a BLE link. On a connected BLE link, independent of the GAP role, GATT defines two roles based on the source and destination of data:

- **GATT server:** A GATT server is a device that contains data or state. When configured by a GATT client, it sends data to the GATT client or modifies its local state. For example, a heart-rate measurement device is a GATT server that sends heart-rate data to a smartphone GATT client.
- **GATT client:** A GATT client is a device that configures the state of a GATT server or receives data from a GATT server. For example, a smartphone that receives heart-rate information from the heart-rate device is a GATT client.

After establishing a BLE link, the GATT client discovers all the data present on the GATT server. Once it is discovered, the GATT client can configure and/or read/write data from the GATT server.

Figure 6 shows an example of a BLE Pioneer Kit configured in the GAP Peripheral/GATT server role that is connected to a smartphone configured as a GAP Central/GATT client device. The GAP and GATT role configuration on each device is defined by the application use cases or [BLE Profile](#). For example, a device that measures heart rate should be configured as a GAP Peripheral device with a GATT server that exposes heart-rate data, and a smartphone that reads the heart-rate information from a heart-rate device should be configured as a GAP Central device with a GATT client.

Figure 6. BLE Application Overview



## GATT Data Format

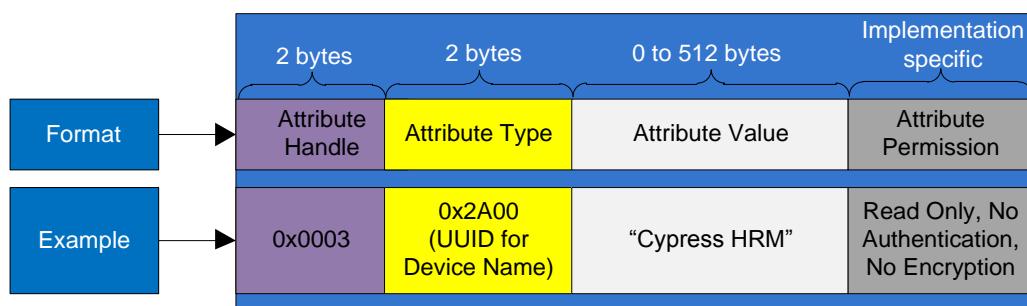
A GATT server uses Attributes, Characteristics, and Services to represent and abstract data in a BLE device.

- **Attribute:** An Attribute is the fundamental data container of the ATT/GATT layer that represents a discrete piece of information. The structure of an Attribute consists of the following, as shown in [Figure 7](#).
  - Attribute Handle: Used to address the Attribute
  - Attribute Type: A 16-bit Universally Unique Identifier (UUID) assigned by the Bluetooth SIG that specifies what is contained in the Attribute
  - Attribute Value: Contains the actual data
  - Attribute Permission: Specifies read/write and security requirements for the Attribute

Heart-rate measurement, battery level, battery level units, and device name are a few examples of an Attribute.

A GATT server consists of a number of Attributes that are stored in a firmware database called the “Attribute database.” The GATT client performs read/write operations on one or more Attributes in the GATT server’s Attribute database using the Attribute handle.

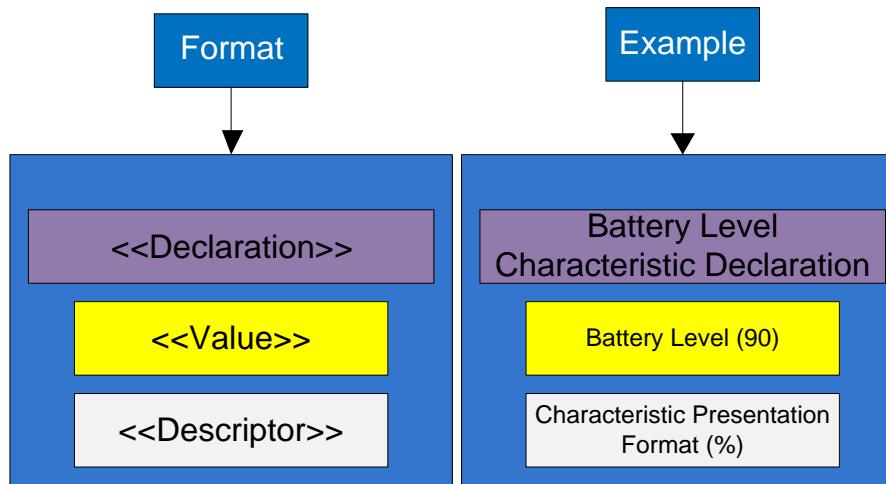
Figure 7. GATT Attribute Example



- **Characteristic:** A Characteristic is composed of multiple discrete Attributes that when combined define system information or meaningful data. A Characteristic consists of a Characteristic Declaration Attribute, a Characteristic Value Attribute, and optionally one or more Characteristic Descriptor Attributes, as shown in [Figure 8](#). Combining the Battery Level Attribute of “90” and Battery Level Descriptor Attribute of “%” provides the battery level information of a system as 90%.

The Bluetooth SIG offers a set of predefined [standard Characteristics](#) that you can use in your application. Or you can define your own custom Characteristics.

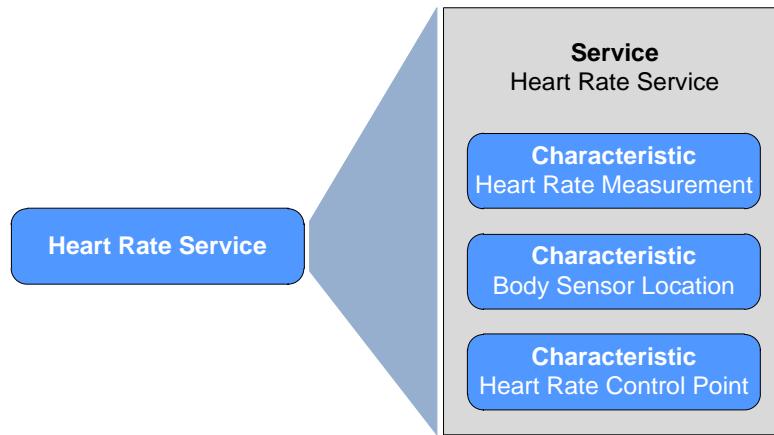
Figure 8. GATT Characteristic Example



- **Service:** A Service is composed of one or more related Characteristics that define a particular function or feature of a device. [Figure 9](#) shows an example of a Heart Rate Service that has three Characteristics describing information related to the heart-rate measurement.

The Bluetooth SIG offers a set of predefined [standard Services](#) for implementing commonly used BLE device functionalities. Or you can define your own custom Services that consist of standard or custom Characteristics.

Figure 9. GATT Service Example



## BLE Profile

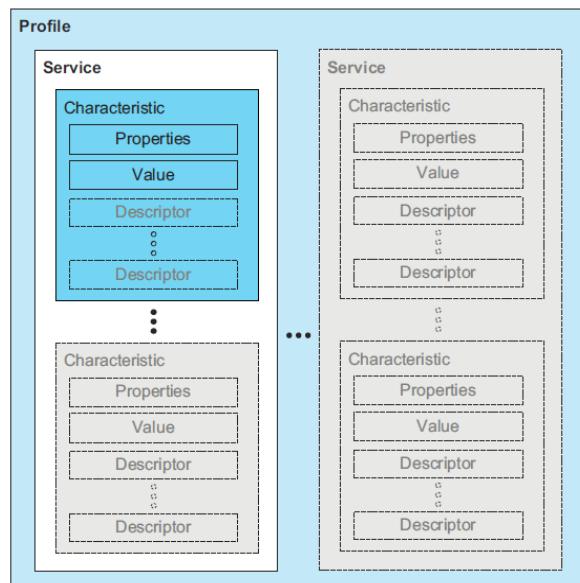
A BLE Profile is a specification that guarantees application-level interoperability between Profile-compliant devices. It defines the role and configuration of different BLE layers and GATT Service(s) to be supported to create a specific end application or use case. For example, if you want to implement a heart-rate monitoring device, the BLE Heart Rate Profile defines the required GAP, and GATT roles and the GATT Services to be supported to create an interoperable heart-rate measurement device. The Bluetooth SIG offers a set of predefined [Standard Profiles](#) for commonly used BLE end applications. Or you can create your own custom Profile that consists of standard or custom Services.

As shown in [Figure 6](#), similar to the GATT layer, the Profile defines two application roles:

- **Sensor or server:** Application that has data
- **Collector or client:** Application that wants data

A summary of the data abstraction and hierarchy in a BLE device is shown in [Figure 10](#).

Figure 10. BLE Data Hierarchy\*



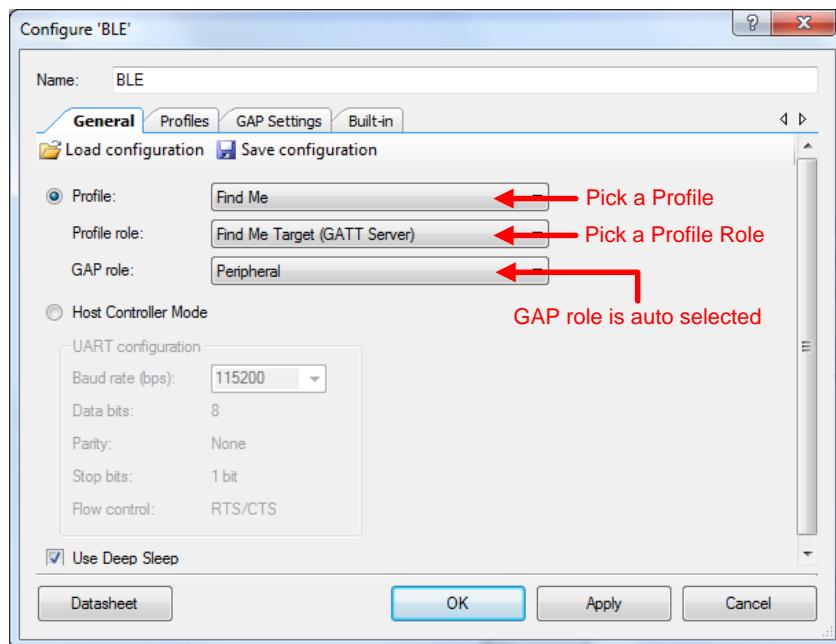
\* Image courtesy of Bluetooth SIG

## BLE Component

The BLE Component in PSoC Creator abstracts the BLE protocol complexity into a simple and easy-to-use GUI and a few APIs. It takes five steps to configure the BLE Component as a standard BLE Peripheral:

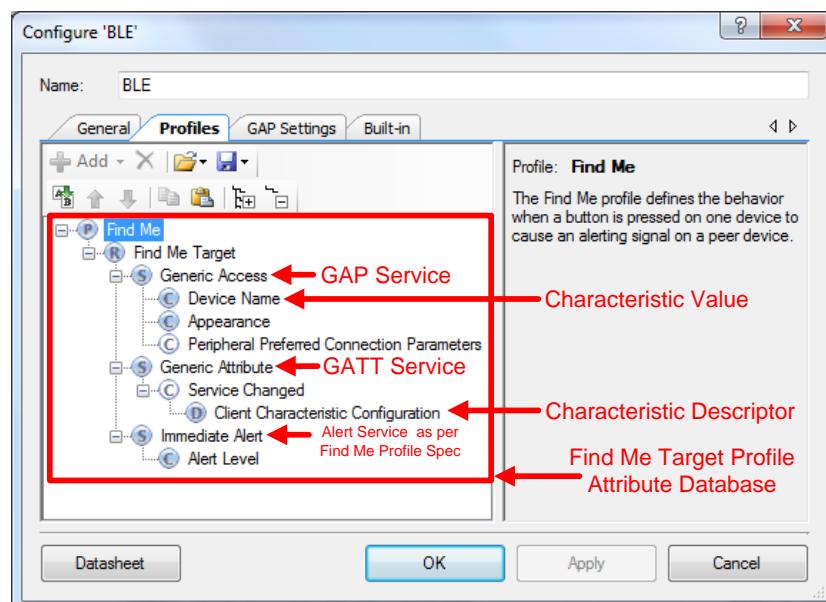
**Step 1:** Select a desired Profile and Profile role for your design. The BLE Component will automatically select the required GAP and GATT roles for the selected Profile role, as shown in [Figure 11](#).

Figure 11. BLE Profile Configuration



**Step 2:** For the selected Profile role, all the supported GATT Services and their corresponding Characteristics are auto-generated per the Profile specification. Verify and/or edit the Service and Characteristic values if required. The configurations in the **Profiles** tab form the Attribute database for the selected design.

Figure 12. BLE Attribute Database Configuration



**Step 3:** Configure the GAP general and advertisement data settings for your design, as shown in Figure 13 and Figure 14.

Figure 13. BLE GAP Configuration

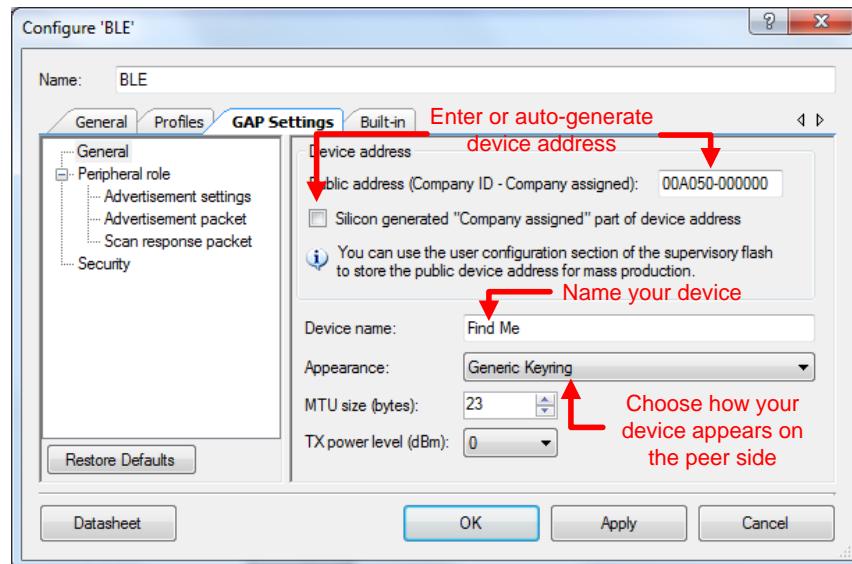
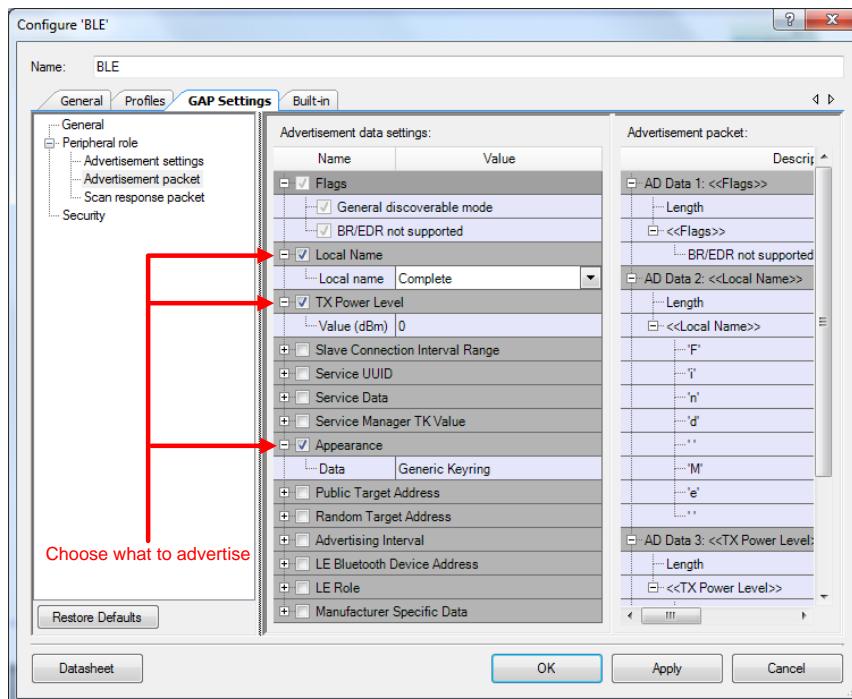


Figure 14. BLE Advertisement Data Configuration



**Step 4:** Write firmware to initialize the design you just configured. Register application layer event handlers with the BLE Component to receive events and data from the BLE Component. There are two types of event handlers: a generic event handler to handle GAP, GATT, and other generic events and a Service specific event handler to handle Service specific events, with one event handler per supported Service.

**Step 5:** Wait in the program main loop for an event from the BLE Component and take the necessary action or send data to the Central device using BLE Component APIs.

The [My First PSoC 4 BLE Design](#) section will walk you through a step-by-step configuration of the BLE Component for creating a simple Peripheral application. Refer to application notes [AN91184](#) and [AN91162](#) for a step-by-step description of how to use the BLE Component to develop applications using BLE standard and custom Profiles.

## PSoC 4 BLE Development Setup

Figure 15 shows the hardware and software tools required for evaluating BLE Peripheral designs using the PSoC 4 BLE device. The BLE Pioneer Kit (red board in Figure 15) acts as a Peripheral that can communicate with either a CySmart iOS/Android app or CySmart Host Emulation Tool that acts as a Central device. The CySmart Host Emulation Tool also requires a BLE Dongle (black board in Figure 15) for its operation. The development system setup required for programming and debugging a BLE design is shown in Figure 16.

Figure 15. BLE Functional Setup

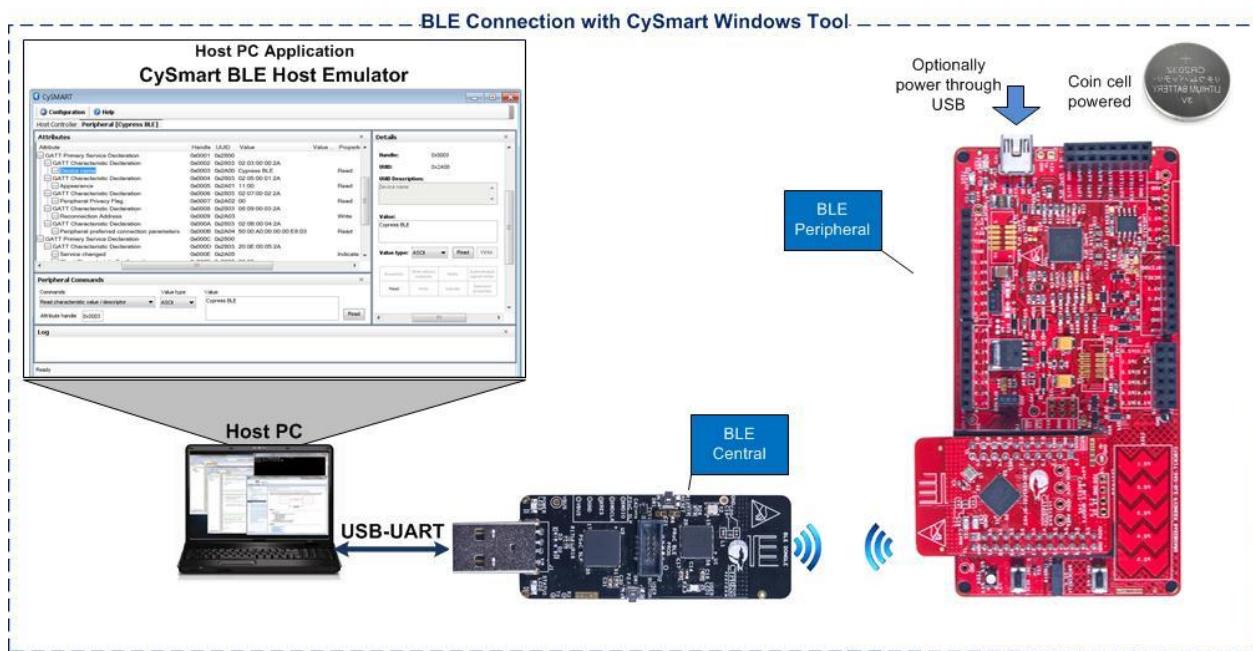
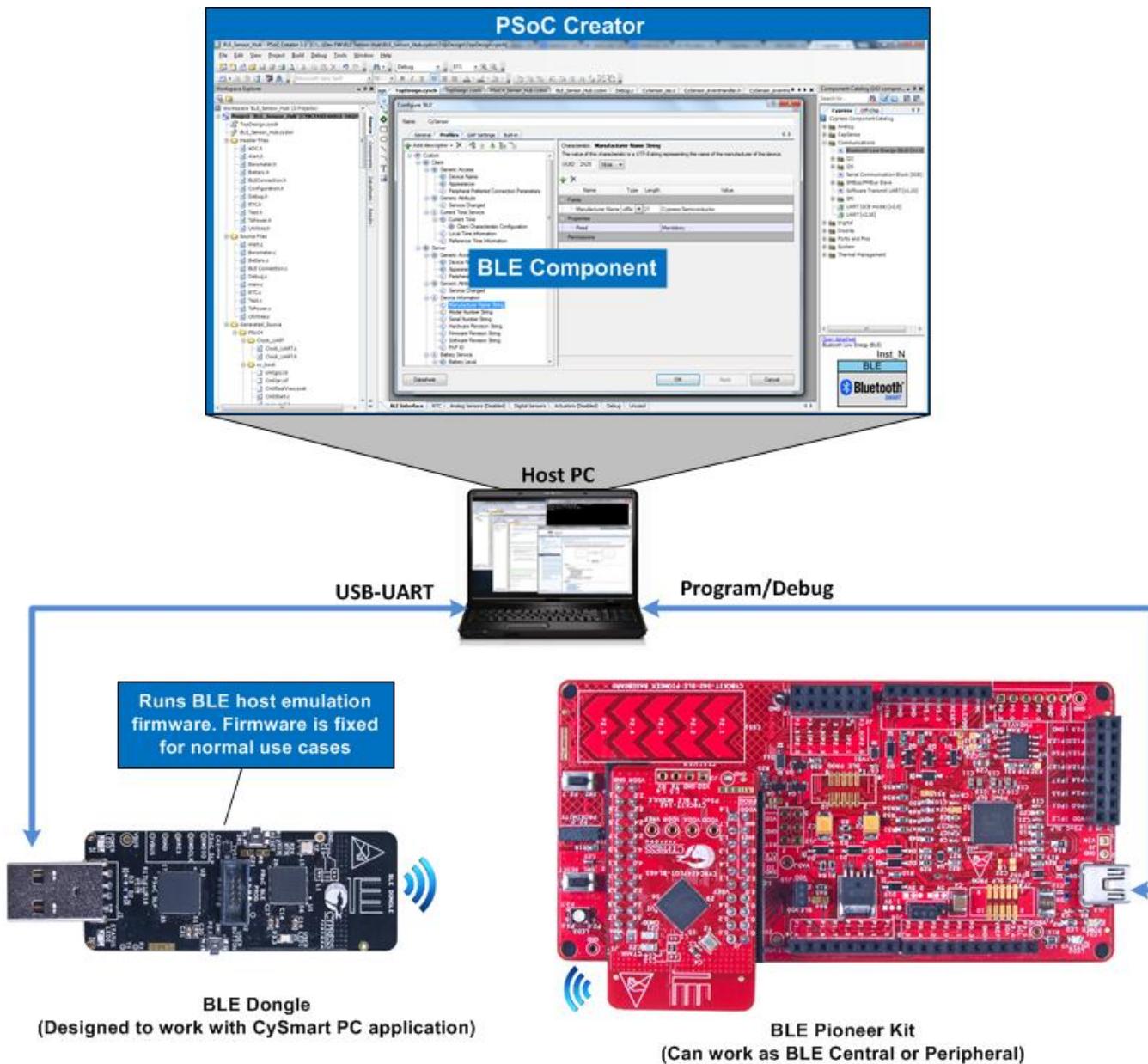


Figure 16. BLE Development Setup



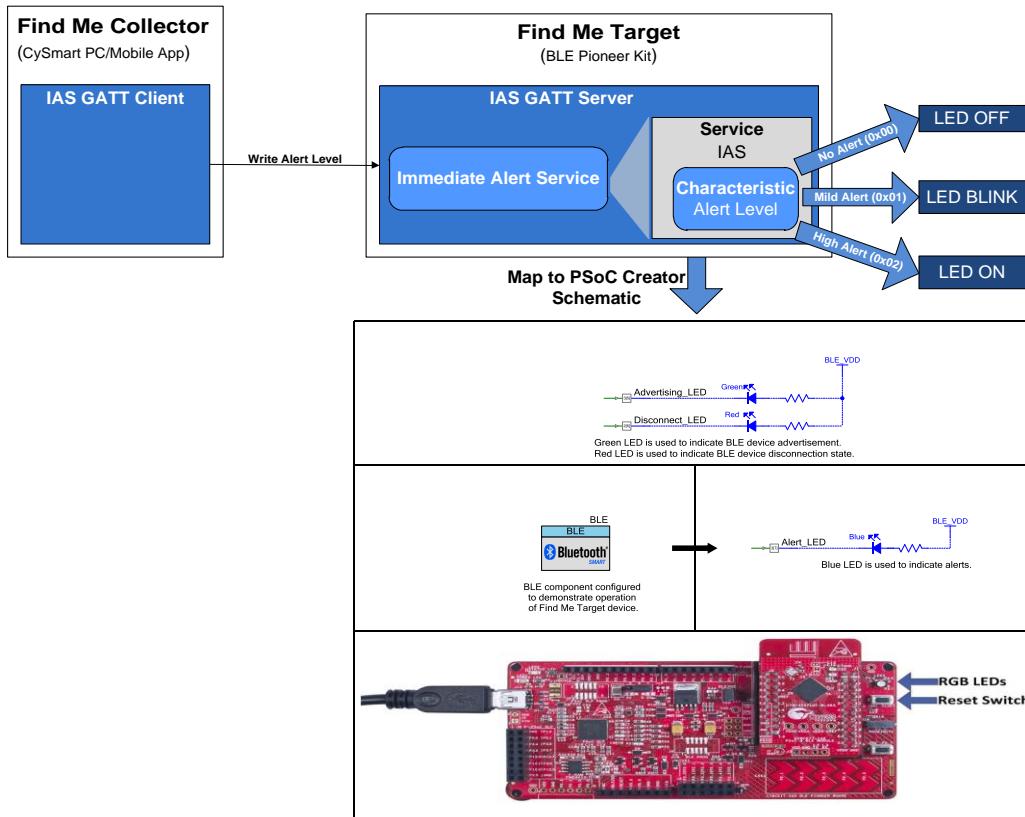
## My First PSoC 4 BLE Design

This section gives you a step-by-step process for building a simple BLE Pioneer Kit-based design with the PSoC 4 BLE device using PSoC Creator.

### About the Design

This design implements a BLE [Find Me Profile](#) in the Target role that consists of an Immediate Alert Service ([IAS](#)). Alert levels triggered by the Find Me Locator are indicated by varying the state of a LED on the BLE Pioneer Kit, as [Figure 17](#) shows. Two status LEDs indicate the state of the BLE interface.

Figure 17. My First PSoC 4 BLE Design



### Prerequisites

Before you get started with the implementation, make sure you have a [BLE Pioneer Kit](#) and have installed the following software:

- [PSoC Creator 3.1 SP1](#) or later with [PSoC Programmer 3.22.0](#) or later
- [CySmart Host Emulation Tool](#) or CySmart iOS/Android app

You can create your first PSoC 4 BLE design in four steps:

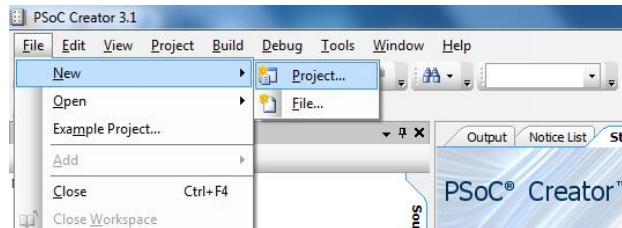
1. Configure the design in the PSoC Creator schematic page.
2. Write the firmware to initialize and handle BLE events.
3. Program the PSoC 4 BLE device on the BLE Pioneer Kit.
4. Test your design using the CySmart Host Emulation Tool or mobile app.

## Part 1: Configure the Design

This section takes you on a step-by-step guided tour of the design process. It starts with creating an empty project and guides you through schematic design entry. You can [skip this section](#) if you simply wish to try the example project provided with this application note without going through the build process.

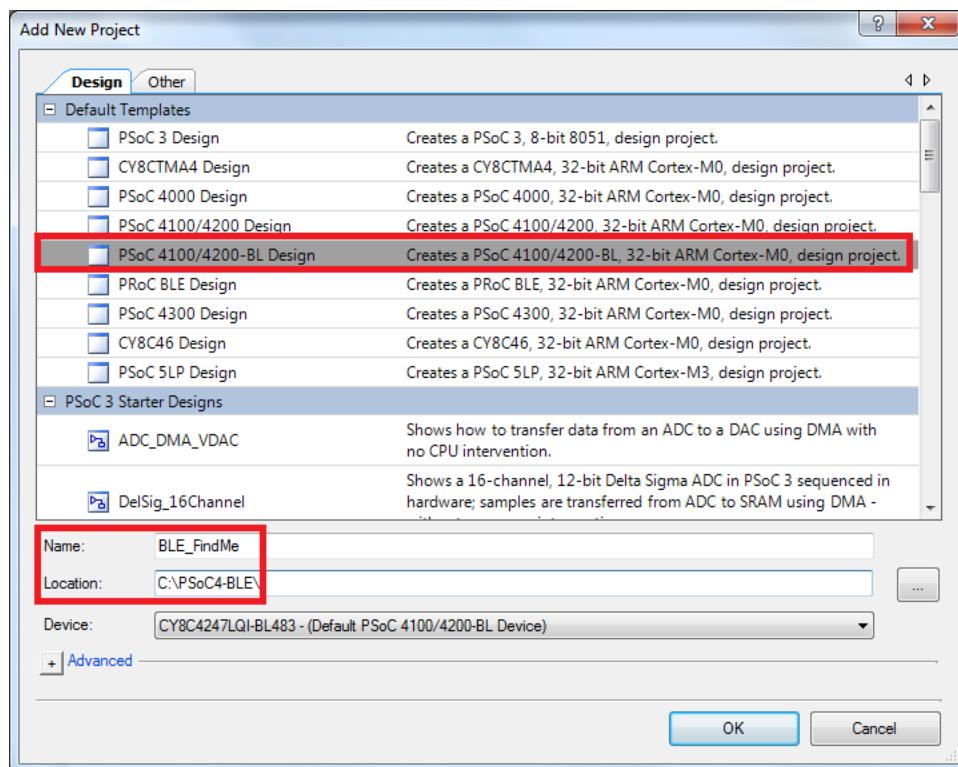
1. Install PSoC Creator 3.1 SP1 or later on your PC.
2. Start PSoC Creator, and from the **File** menu, choose **New > Project**, as [Figure 18](#) shows.

Figure 18. Creating a New Project



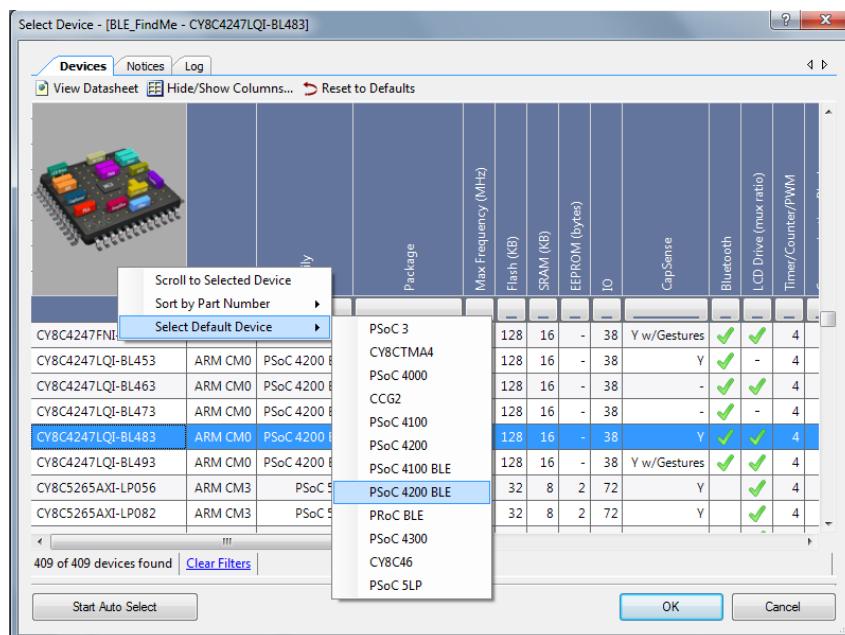
3. Select the project template “PSoC 4100/4200-BL Design,” and give the project a name such as “BLE\_FindMe,” as [Figure 19](#) shows. Choose an appropriate location for your new project, and then click **OK**.

Figure 19. Naming the New Project



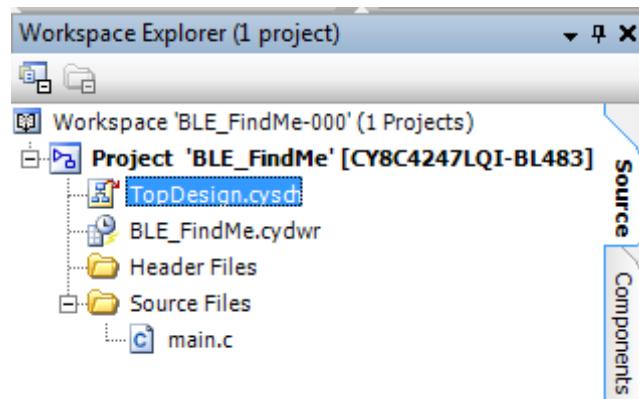
4. Go to **Project > Device Selector** and select the device. If you are using a development kit, read the part number from the kit or refer to the kit guide for the part number. You can right click on the device selector's **Devices** tab area and click on **Select Default Device for PSoC 4200 BLE** family to select CY8C4247LQI-BL483 device used on **BLE Pioneer Kit** as shown in [Figure 20](#).

Figure 20. Device Selection



5. Creating a new project generates a project folder with a baseline set of files. You can view these files in the **Workspace Explorer** window, as [Figure 21](#) shows. Open the project schematic file *TopDesign.cysch* by double-clicking it.

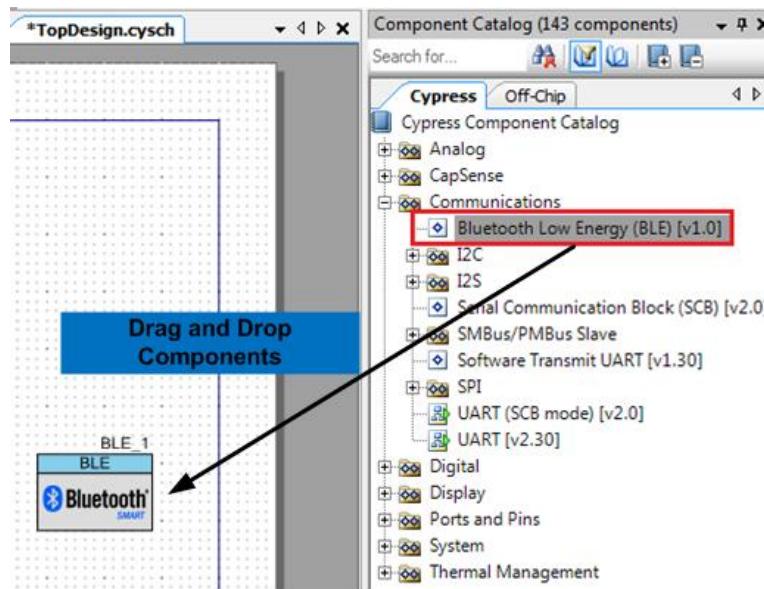
Figure 21. Opening TopDesign Schematic



6. In the project schematic, you can create analog, digital, and communication designs by dragging and dropping Components and wiring them. For the current design, you need two Components, BLE and Pin, as [Figure 17](#) shows.

7. Drag one Bluetooth Low Energy (BLE) Component from the **Component Catalog** onto the schematic, as [Figure 22](#) shows.

Figure 22. Location of the BLE Component



8. Double-click the BLE Component on the schematic to configure it as a “BLE Find Me Target” with the following properties:
- GAP Peripheral role with Find Me Target (GATT server) configuration
  - GAP Device Name set to “Find Me Target” and Appearance set to “Generic Keyring.” This configures the device name and type that will appear when another device attempts to discover your device.
  - Limited advertisement mode with an advertising timeout of 30 seconds and a fast advertisement interval of 20 to 30 ms. Fast advertising allows quick discovery and connection but consumes more power due to increased RF advertisement packets.
  - Advertisement Packet with Immediate Alert Service enabled and Scan Response Packet with Local Name, Tx Power Level, and Appearance fields enabled.
  - GAP security set to the lowest possible configuration that does not require authentication, encryption, or authorization for data exchange (Mode 1, No Security)

[Figure 23](#) to [Figure 29](#) show the BLE Component screenshots for this configuration.

**Note** You do not need to change the default configuration of the BLE Component in the **Profiles** tab for this design.

Figure 23. BLE Component General Configuration

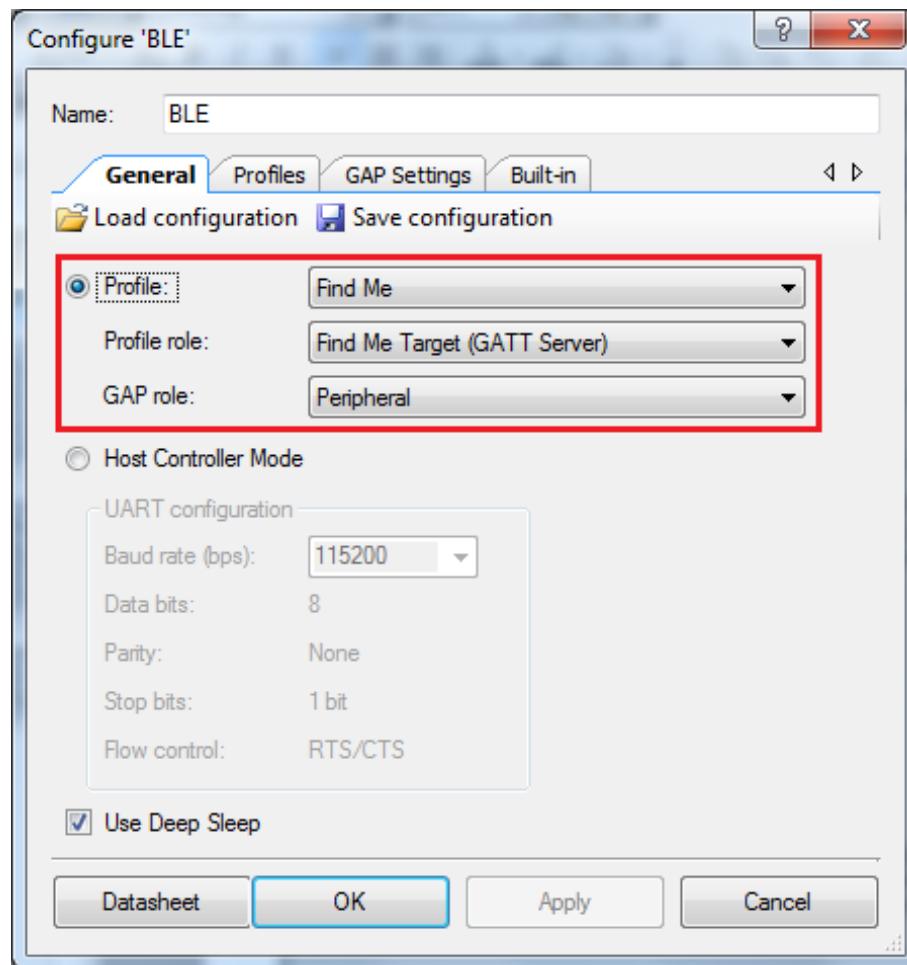


Figure 24. BLE Component GAP General Settings

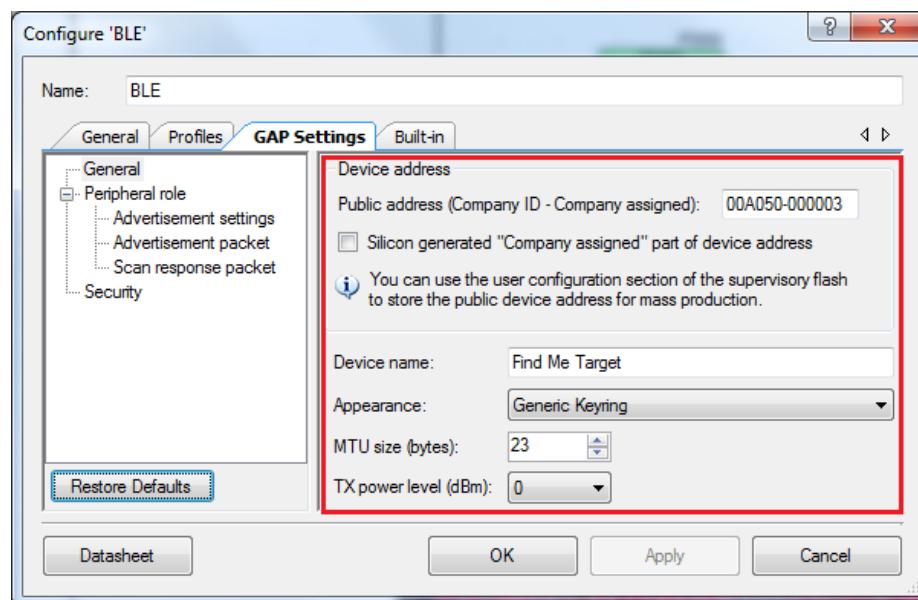


Figure 25. BLE Component GAP Advertisement Settings

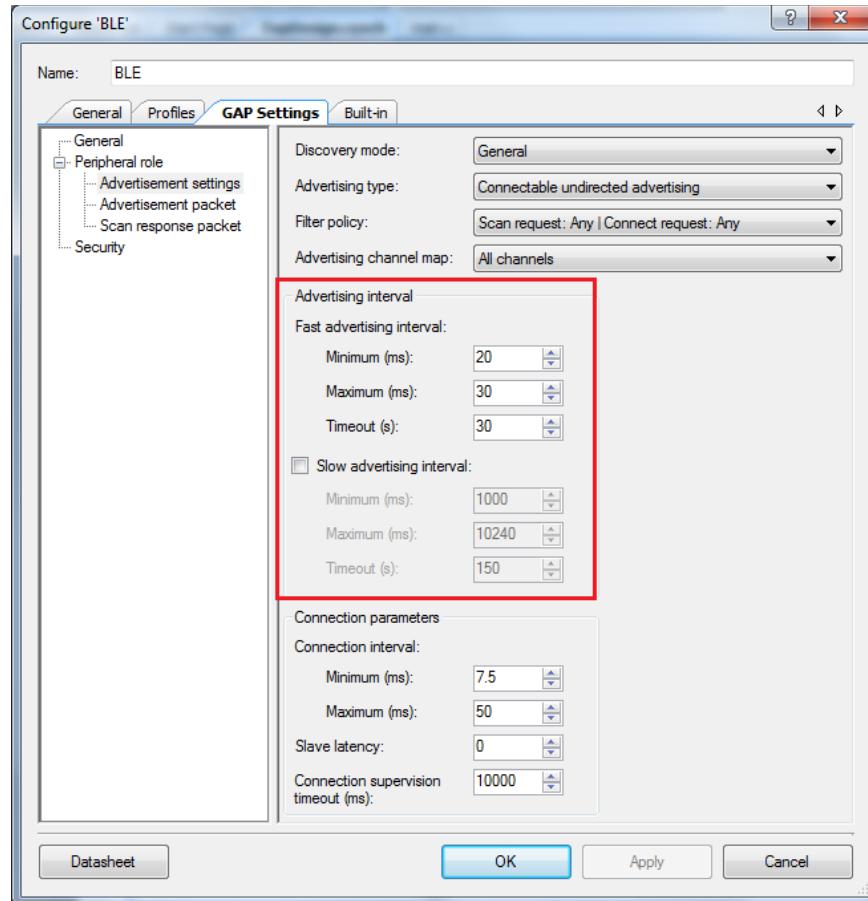


Figure 26. BLE Component GAP Advertisement Packet

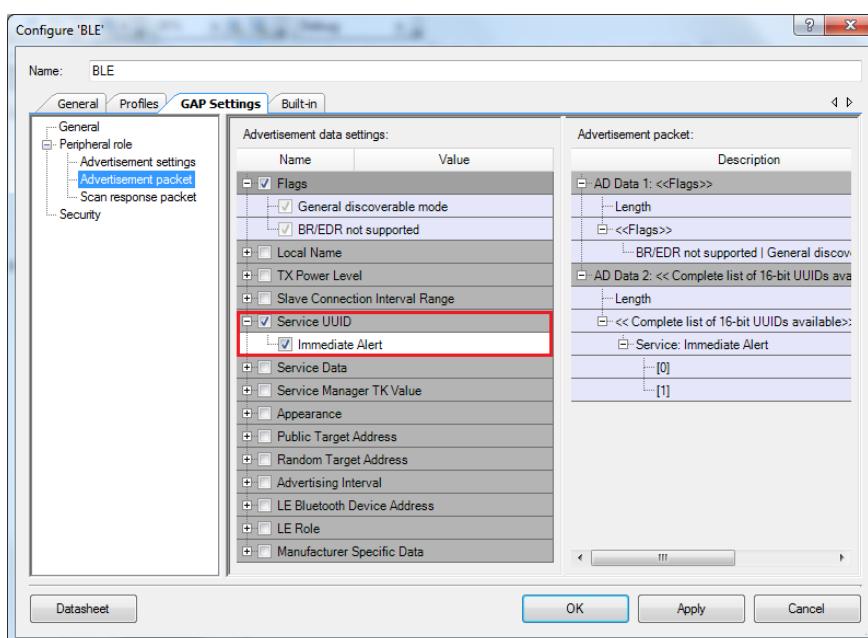


Figure 27. BLE Component GAP Scan Response Packet

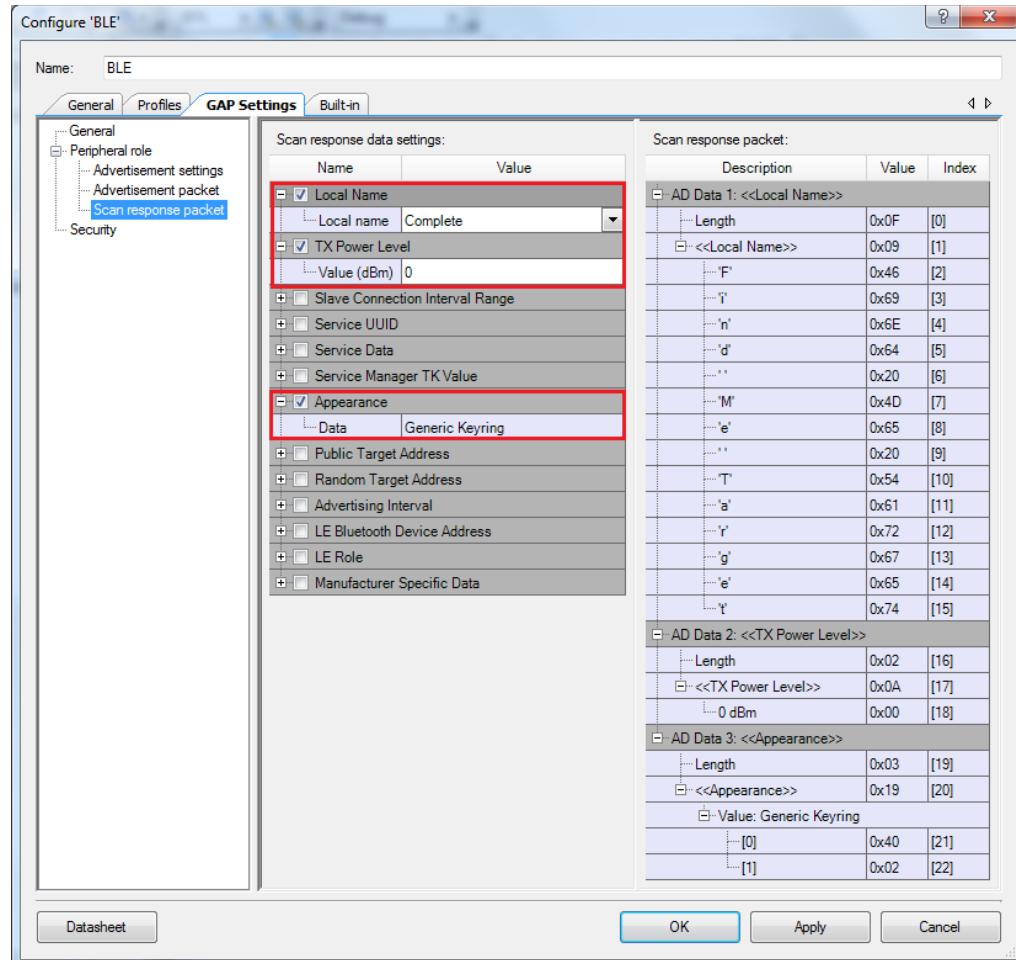


Figure 28. BLE Component GAP Security Settings

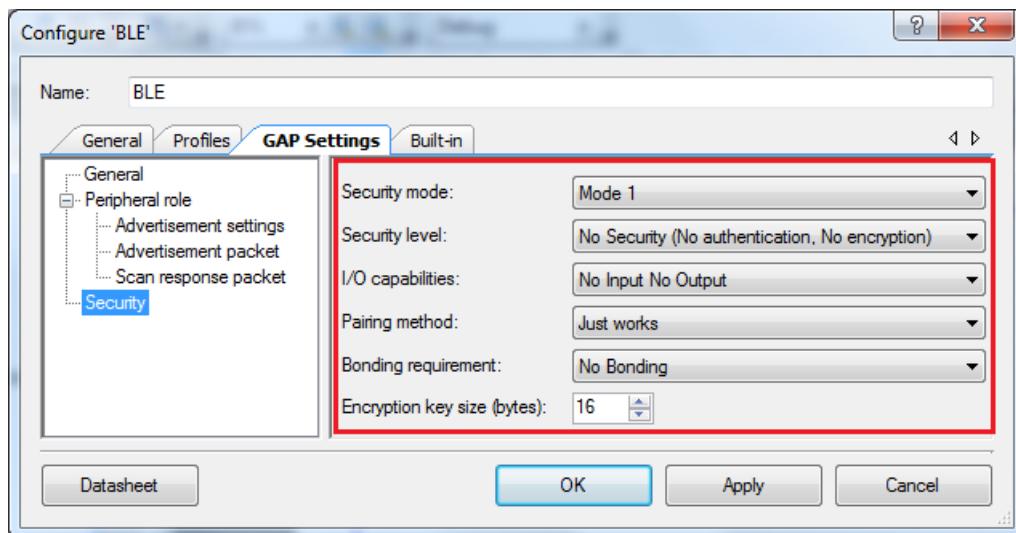
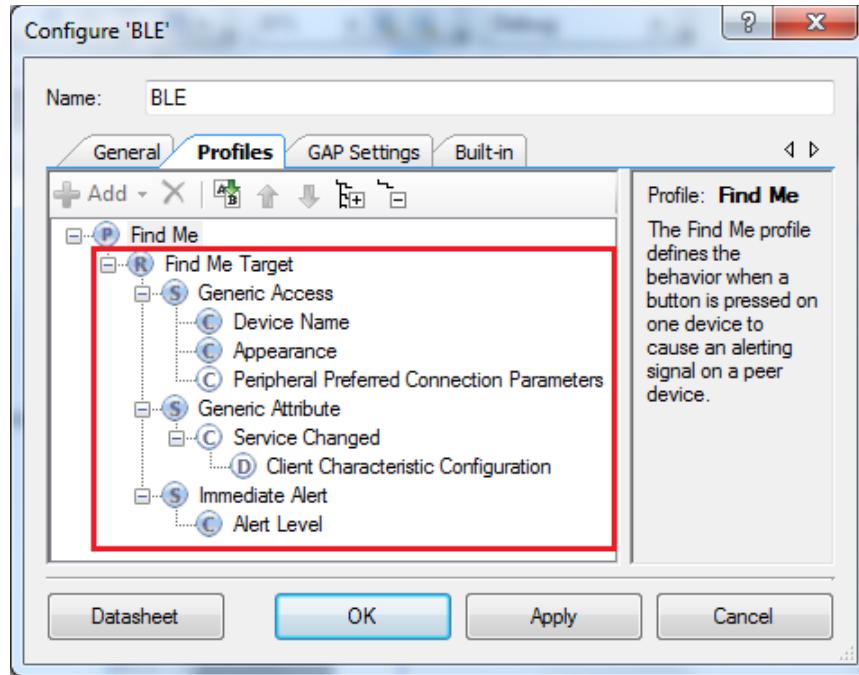


Figure 29. BLE Component Profiles Configuration



- Drag and drop a Digital Output Pin Component. Change the name to Alert\_LED, as Figure 30 and Figure 31 show.

Figure 30. Location of the Digital Output Pin Component

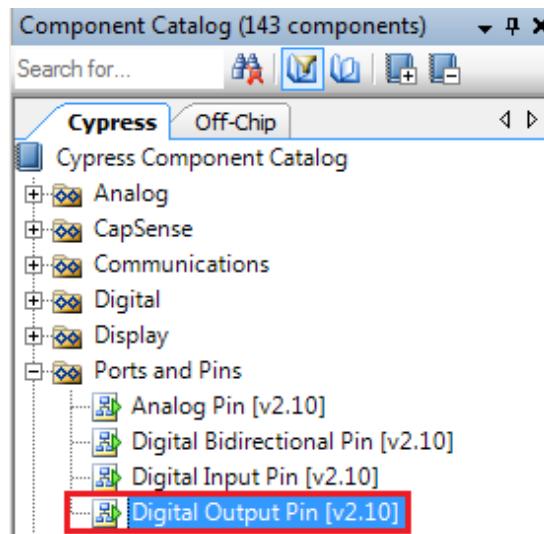
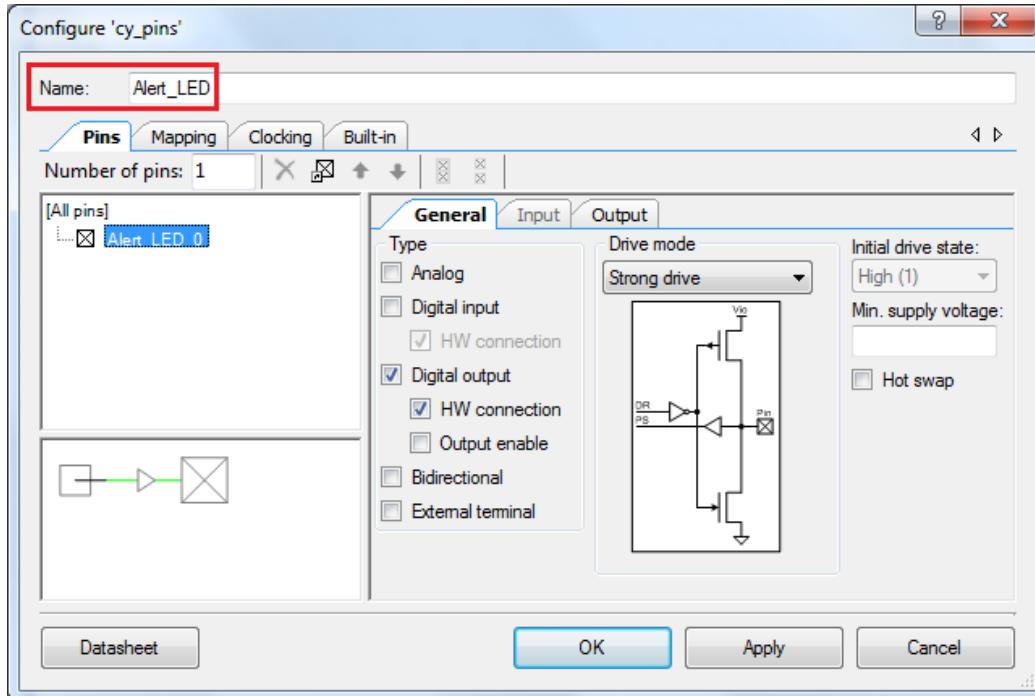


Figure 31. Renaming a Pin Component



10. Add two more Digital Output Pin Components named Advertising\_LED and Disconnect\_LED to your design (see [Figure 30](#) and [Figure 31](#)) with **HW Connection** deselected and **Initial drive state** set to “High(1),” as [Figure 32](#) and [Figure 33](#) show. These pins will be used to drive the BLE advertising and disconnection state indicator LEDs. The LEDs on the BLE Pioneer Kit are active LOW; that is, the high pin-drive state turns off the LEDs and the low pin-drive state turns them on.

Figure 32. Advertising LED Configuration

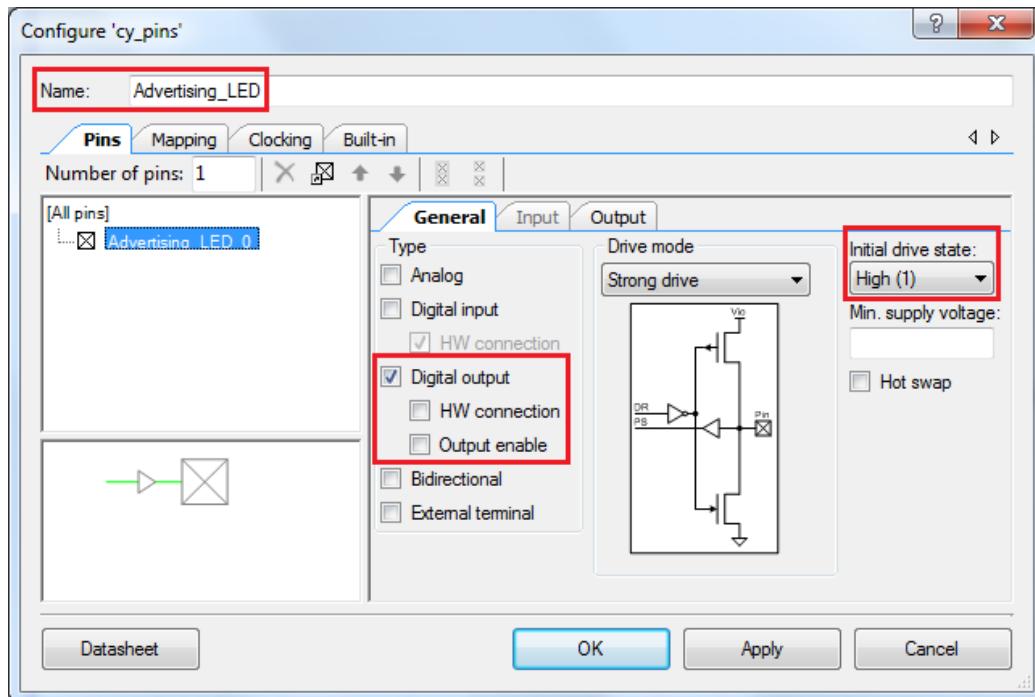
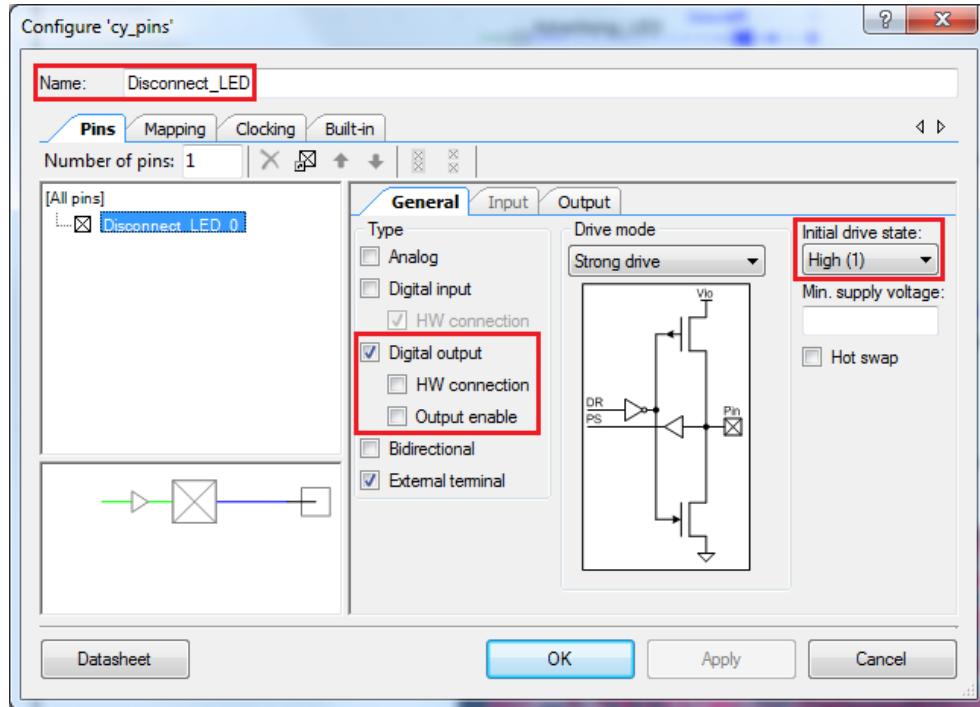
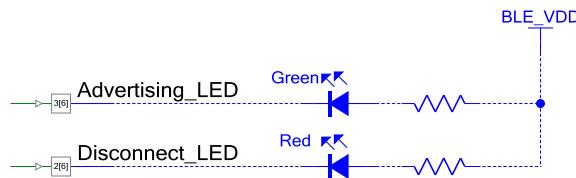


Figure 33. Disconnection LED Configuration

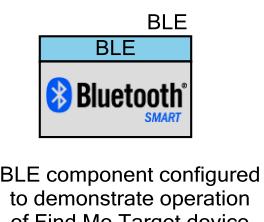


11. After completing the schematic configuration, your design should look similar to Figure 34.

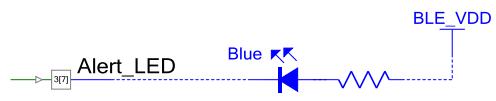
Figure 34. Schematic Configuration



Green LED is used to indicate BLE device advertisement.  
Red LED is used to indicate BLE device disconnection state.



BLE component configured to demonstrate operation of Find Me Target device.

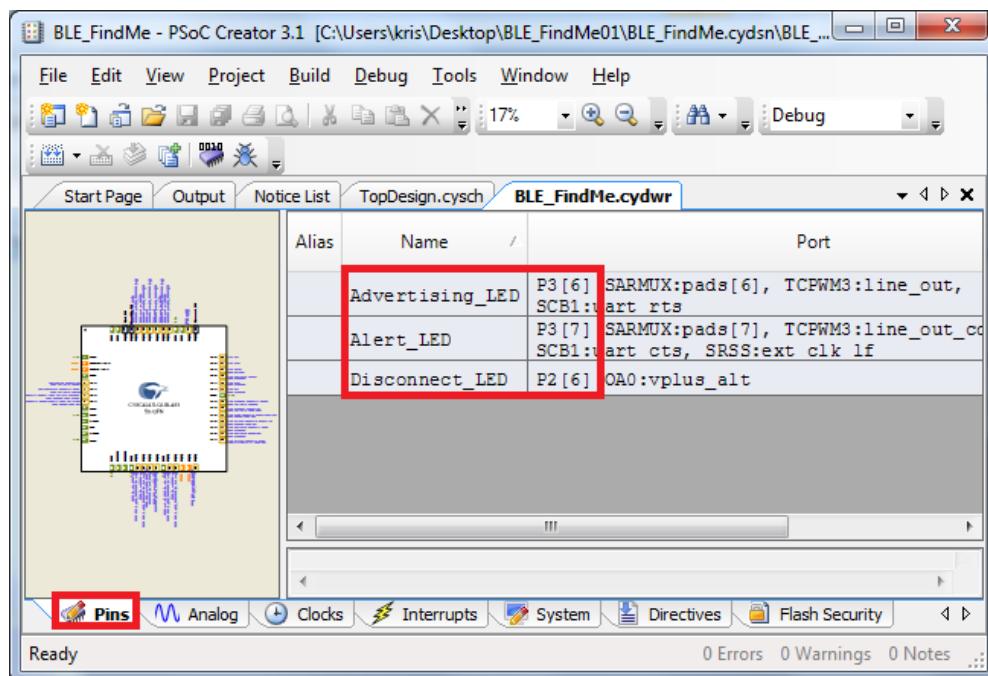


Blue LED is used to indicate alerts.

**Note** The blue dotted lines, the LED symbols, and resistor symbols shown in Figure 34 are off-chip PSoC Creator Components that are present only for descriptive purposes and are not required for the functioning of your design. You can add off-chip Components to your design by dragging and dropping the required off-chip Components on to your project schematic page from PSoC Creator's off-chip Component Catalog.

12. Open the file *BLE\_FindMe.cydwr* (Design-Wide Resources) file from **Workspace Explorer** and click the **Pins** tab. You can use this tab to select the device pins for the outputs (Advertising\_LED, Disconnect\_LED, and Alert\_LED). [Figure 35](#) shows the pin configuration to connect the Advertising\_LED, Disconnect LED, and Alert\_LED pins to the green, red, and blue LEDs on the BLE Pioneer Kit respectively.

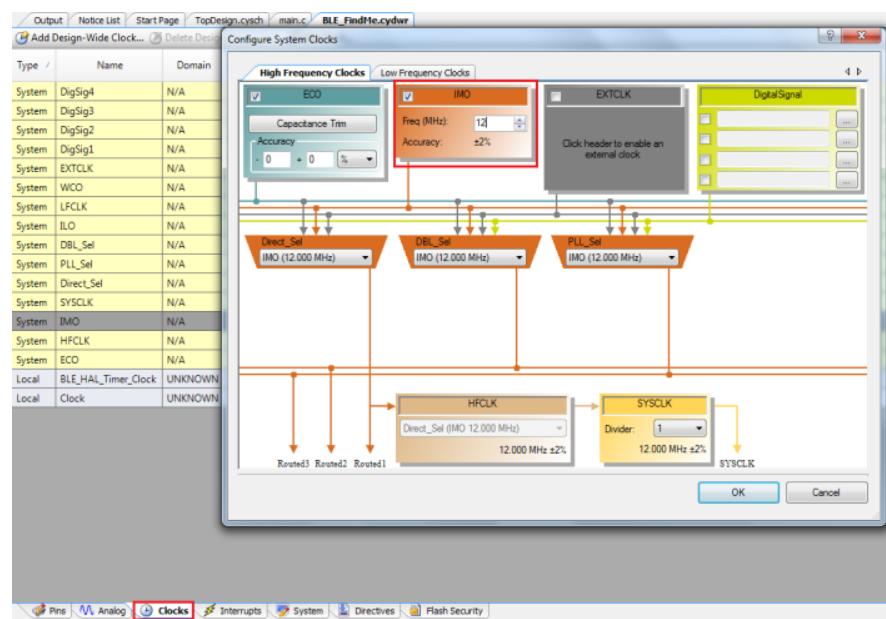
Figure 35. Pin Selection



If you are using your own board or a development kit with no LEDs, select the appropriate pins. You can connect external LEDs to the selected pins, as [Figure 17](#) shows.

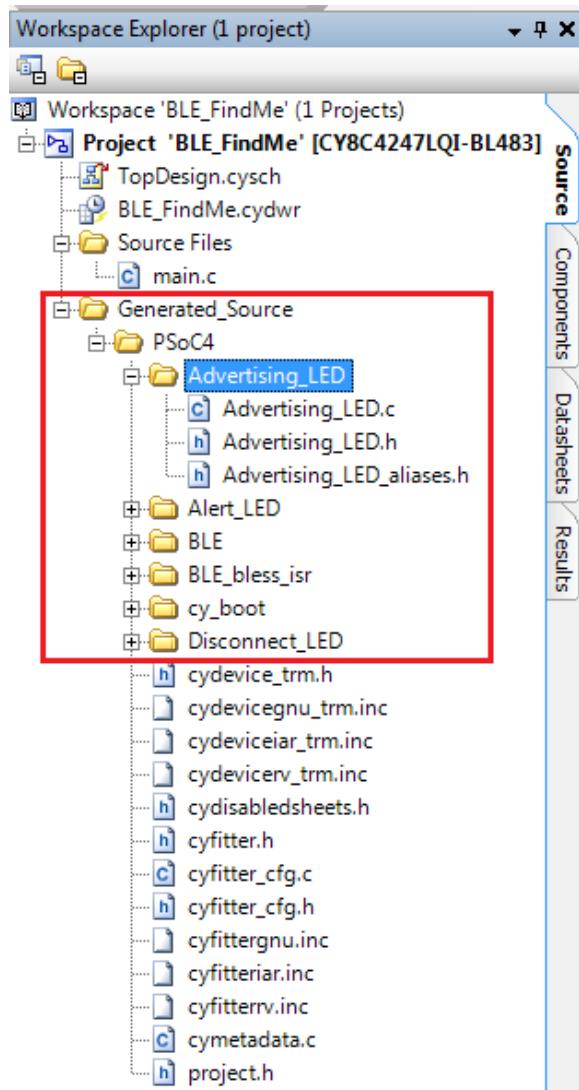
13. Similarly, in the **Clocks** tab under the *BLE\_FindMe.cydwr* file, configure the internal main oscillator (IMO) to 12 MHz, as [Figure 36](#) shows.

Figure 36. Clock Configuration



14. Select **Generate Application** from the **Build** menu. Notice in the **Workspace Explorer** window that PSoC Creator automatically generates source code files for the BLE and Digital Output Pin Components, as [Figure 37](#) shows.

Figure 37. Generated Source Files



## Part 2: Write the Firmware

Four main firmware blocks are required for designing BLE standard Profile applications using PSoC Creator:

- System initialization
- BLE stack event handler
- BLE service-specific event handler
- Main loop and low-power implementation

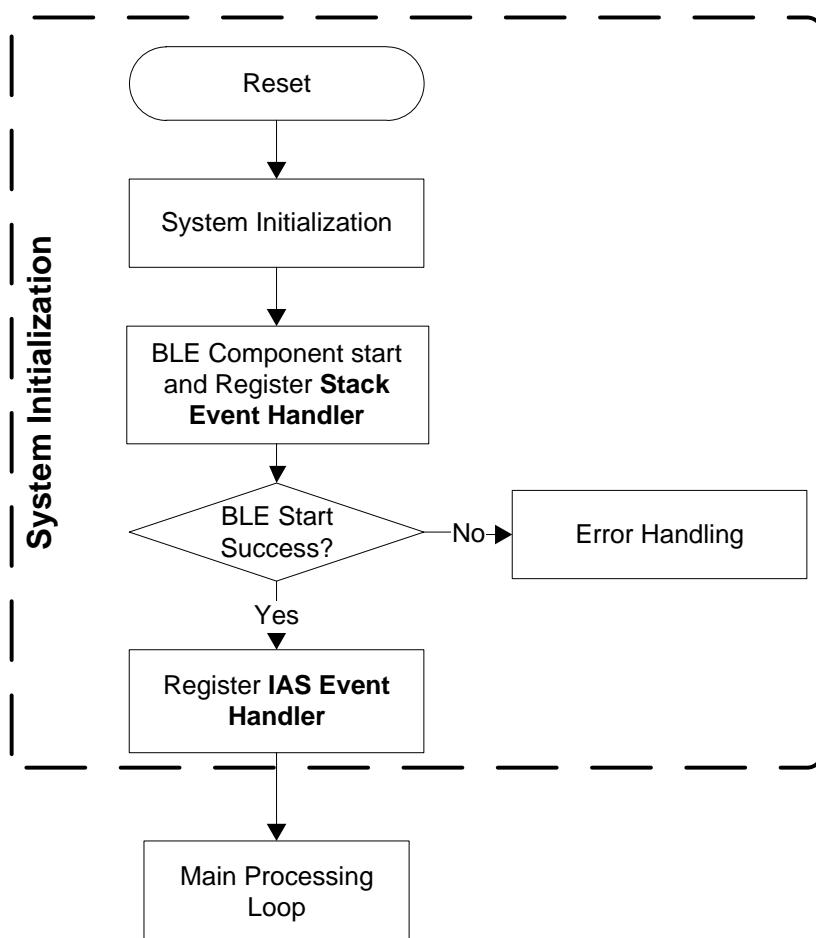
The following sections discuss these blocks with respect to the design that you configured in [Part 1: Configure the Design](#).

### System Initialization

When the PSoC 4 BLE device is reset, the firmware first performs the system initialization, which includes enabling global interrupts and enabling other Components used in the design. After the system is initialized, the firmware initializes the BLE Component, which internally initializes the complete BLE subsystem.

As a part of the BLE Component initialization, you must pass the event-handler function, which will be called by the BLE stack to notify pending events. The BLE stack event handler shown in [Code 2](#) is registered as a part of the BLE initialization. If the BLE Component initializes successfully, the firmware registers another event handler for the IAS-specific events and switches control to the main loop. [Figure 38](#) and [Code 1](#) show the flow chart and the firmware source code for system initialization.

Figure 38. System Initialization Flow Chart



## Code 1. System Initialization Firmware

```

#include <Project.h>

#define LED_ON (0u)
#define LED_OFF (1u)

#define NO_ALERT (0u)
#define MILD_ALERT (1u)
#define HIGH_ALERT (2u)

#define LED_TOGGLE_TIMEOUT (100u)

uint8 alertLevel;

int main()
{
  CYBLE_API_RESULT_T apiResult;

  CyGlobalIntEnable;

  apiResult = CyBle_Start(StackEventHandler);

  if(apiResult != CYBLE_ERROR_OK)
  {
    /* BLE stack initialization failed, check your configuration */
    CYASSERT(0);
  }

  CyBle_IasRegisterAttrCallback(IasEventHandler);
}
  
```

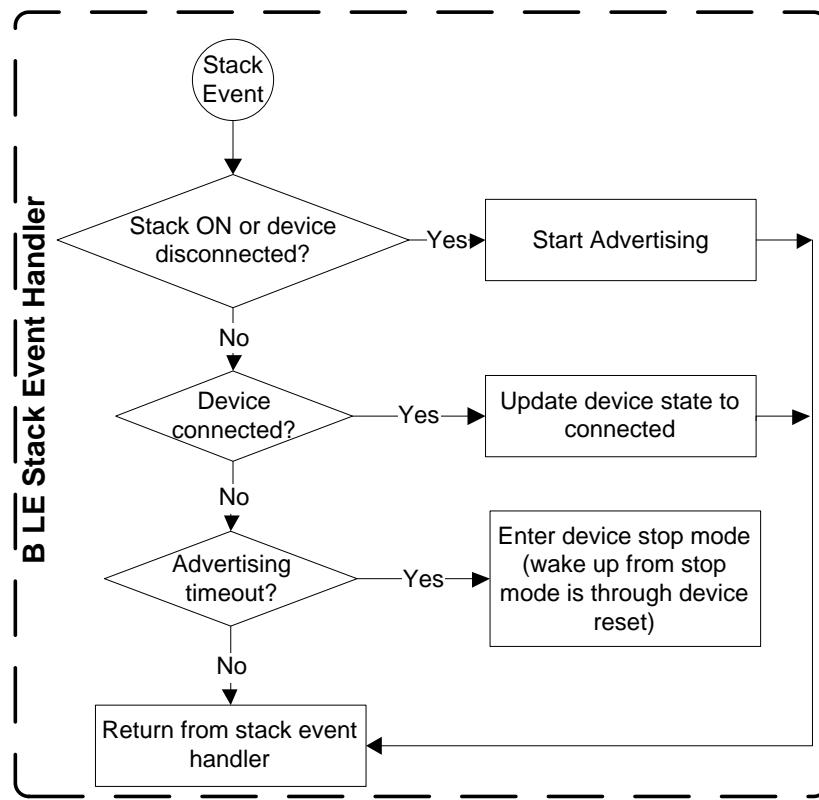
### BLE Stack Event Handler

The BLE stack within the BLE Component generates events to provide the BLE interface status and data to the application firmware through the BLE stack event handler registered by you. The event handler must handle a few basic events from the stack, such as device connection and stack on, and configure the stack accordingly to establish and maintain the BLE link. For the Find Me Target application that you are creating, the BLE stack event handler must process all the events described in Table 1. The flow chart and the firmware for handling BLE stack events are shown in [Figure 39](#) and [Code 2](#).

Table 1. BLE Stack Events

BLE Stack Event Name	Event Description	Event Handler Action
CYBLE_EVT_STACK_ON	BLE stack initialization completed successfully.	Start advertisement and reflect the advertisement state on the LED.
CYBLE_EVT_GAP_DEVICE_DISCONNECTED	BLE link with the peer device is disconnected.	Restart advertisement and reflect the advertisement state on the LED.
CYBLE_EVT_GAP_DEVICE_CONNECTED	BLE link with the peer device is established.	Update the BLE link state on the LED.
CYBLE_EVT_GAPP_ADVERTISEMENT_START_STOP	BLE stack advertisement start/stop event.	Configure the device in Stop mode if the advertisement has timed out.

Figure 39. BLE Stack Event Handler Flow Chart



Code 2. BLE Stack Event Handler Firmware

```

void StackEventHandler(uint32 event, void *eventParam)
{
    switch(event)
    {
        /* Mandatory events to be handled by Find Me Target design */
        case CYBLE_EVT_STACK_ON:
        case CYBLE_EVT_GAP_DEVICE_DISCONNECTED:
            /* Start BLE advertisement for 30 seconds and update link
             * status on LEDs */
            CyBle_GappStartAdvertisement(CYBLE_ADVERTISING_FAST);
            Advertising_LED_Write(LED_ON);
            alertLevel = NO_ALERT;
            break;

        case CYBLE_EVT_GAP_DEVICE_CONNECTED:
            /* BLE link is established */
            Advertising_LED_Write(LED_OFF);
            Disconnect_LED_Write(LED_OFF);
            break;

        case CYBLE_EVT_GAPP_ADVERTISEMENT_START_STOP:
            if(CyBle_GetState() == CYBLE_STATE_DISCONNECTED)
            {
    
```

```

/* Advertisement event timed out, go to low power
 * mode (Stop mode) and wait for device reset
 * event to wake up the device again */
Advertising_LED_Write(LED_OFF);
Disconnect_LED_Write(LED_ON);
CySysPmSetWakeupPolarity(CY_PM_STOP_WAKEUP_ACTIVE_HIGH);
CySysPmStop();

/* Code execution will not reach here */
}
break;

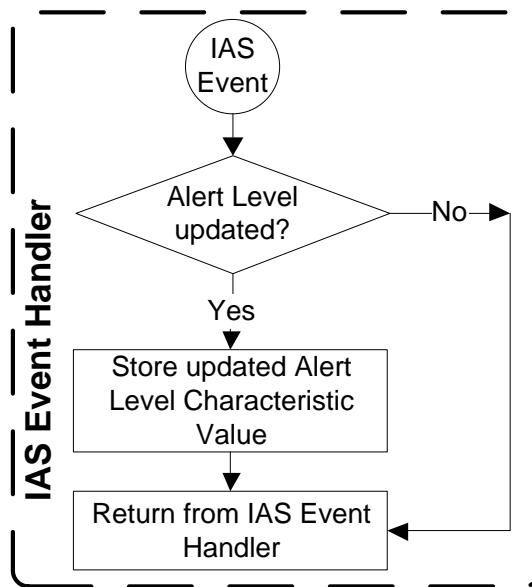
default:
break;
}
}

```

### BLE Service-Specific Event Handler

The BLE Component also generates events corresponding to each of the Services supported by your design. For the Find Me Target application that you are creating, the BLE Component will generate IAS events that will let the application know if the Alert Level Characteristic is updated with a new value. The flow chart and the firmware for handling BLE IAS events are shown in [Figure 40](#) and [Code 3](#) respectively.

Figure 40. BLE IAS Event Handler Flow Chart



## Code 3. BLE IAS Event Handler Firmware

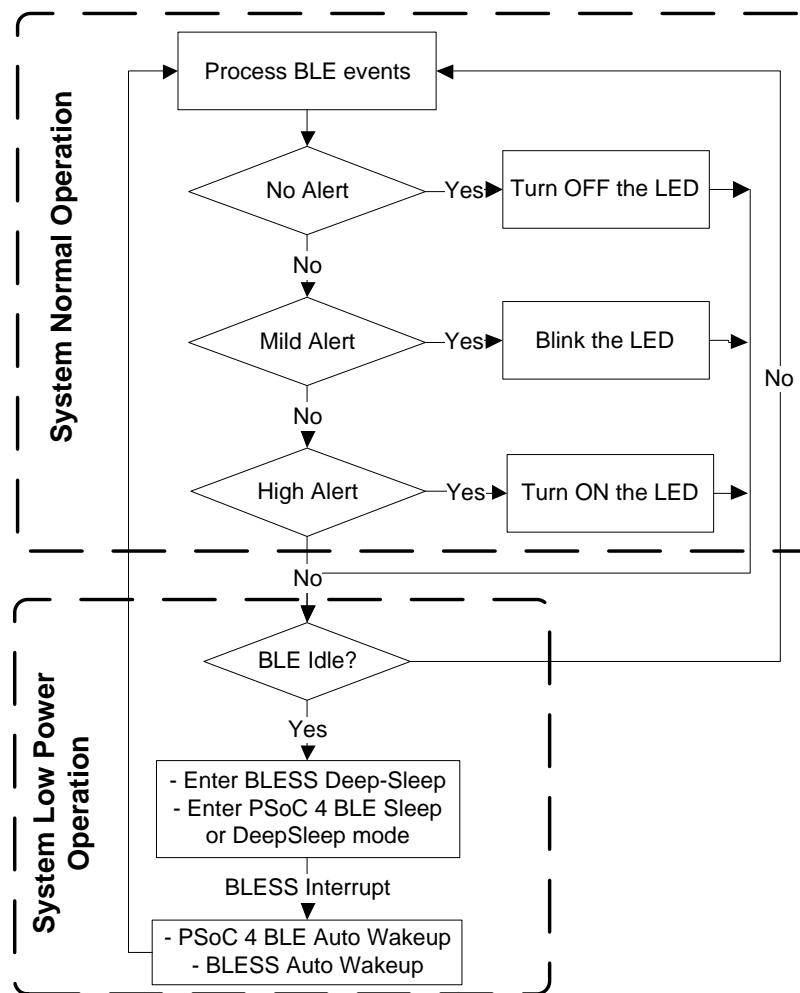
```

void IasEventHandler(uint32 event, void *eventParam)
{
    /* Alert Level Characteristic write event */
    if(event == CYBLE_EVT_IASS_WRITE_CHAR_CMD)
    {
        /* Read the updated Alert Level value from the GATT database */
        CyBle_IassGetCharacteristicValue(CYBLE_IAS_ALERT_LEVEL,
            sizeof(alertLevel), &alertLevel);
    }
}
    
```

**Main Loop and Low-Power Implementation**

The main loop firmware in your design must periodically service the BLE stack-processing event, update the blue alert LED state per the IAS Alert Level Characteristic value and configure the BLESS block and the PSoC 4 BLE system into low-power mode between consecutive BLE advertisement and connection intervals. The main loop flow chart and the firmware are shown in [Figure 41](#) and [Code 4](#).

Figure 41. Firmware Main Loop Flow Chart



## Code 4. Main Loop Firmware

```

for(;;)
{
    static uint8 toggleTimeout = 0;
    CYBLE_BLESS_STATE_T blessState;
    uint8 intrStatus;

    /* Single API call to service all the BLE stack events. Must be
     * called at least once in a BLE connection interval */
        CyBle_ProcessEvents();

    /* Update Alert Level value on the blue LED */
    switch(alertLevel)
    {
        case NO_ALERT:
            Alert_LED_Write(LED_OFF);
            break;

        case MILD_ALERT:
            toggleTimeout++;
            if(toggleTimeout == LED_TOGGLE_TIMEOUT)
            {
                /* Toggle alert LED after timeout */
                Alert_LED_Write(Alert_LED_Read() ^ 0x01);
                toggleTimeout = 0;
            }
            break;

        case HIGH_ALERT:
            Alert_LED_Write(LED_ON);
            break;
    }

    /* Configure BLESS in Deep-Sleep mode */
    CyBle_EnterLPM(CYBLE_BLESS_DEEPSLEEP);

    /* Prevent interrupts while entering system low power modes */
    intrStatus = CyEnterCriticalSection();

    /* Get the current state of BLESS block */
    blessState = CyBle_GetBleSsState();

    /* If BLESS is in Deep-Sleep mode or the XTAL oscillator is turning on,
     * then PSoC 4 BLE can enter Deep-Sleep mode (1.3uA current consumption) */
    if(blessState == CYBLE_BLESS_STATE_ECO_ON ||
       blessState == CYBLE_BLESS_STATE_DEEPSLEEP)
    {
        CySysPmDeepSleep();
    }
    else if(blessState != CYBLE_BLESS_STATE_EVENT_CLOSE)
    {
        /* If BLESS is active, then configure PSoC 4 BLE system in
         * Sleep mode (~1.6mA current consumption) */
        CySysPmSleep();
    }
}

```

```

}
else
{
    /* Keep trying to enter either Sleep or Deep-Sleep mode */
}
CyExitCriticalSection(intrStatus);

/* BLE link layer timing interrupt will wake up the system from Sleep
 * and Deep-Sleep modes */
}

```

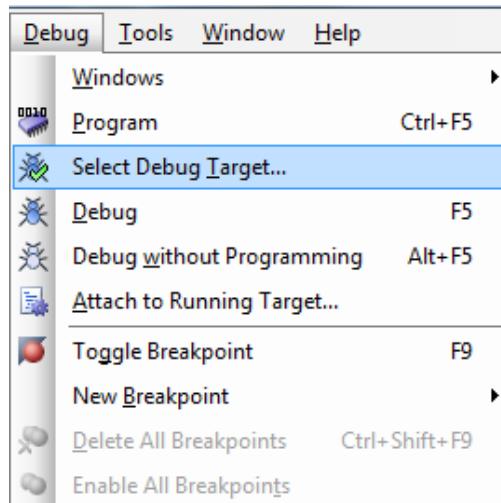
### Part 3: Program the Device

This section shows how to program the PSoC 4 BLE device. If you are using a development kit with a built-in programmer (the BLE Pioneer Kit, for example), connect the kit board to your computer using the USB cable. For other kits, refer to the kit guide.

**Note** The source project for this design is in PSoC Creator 3.1 SP1 or later under **File > Example Project** > select **Keyword** as **Find Me > BLE\_FindMe**.

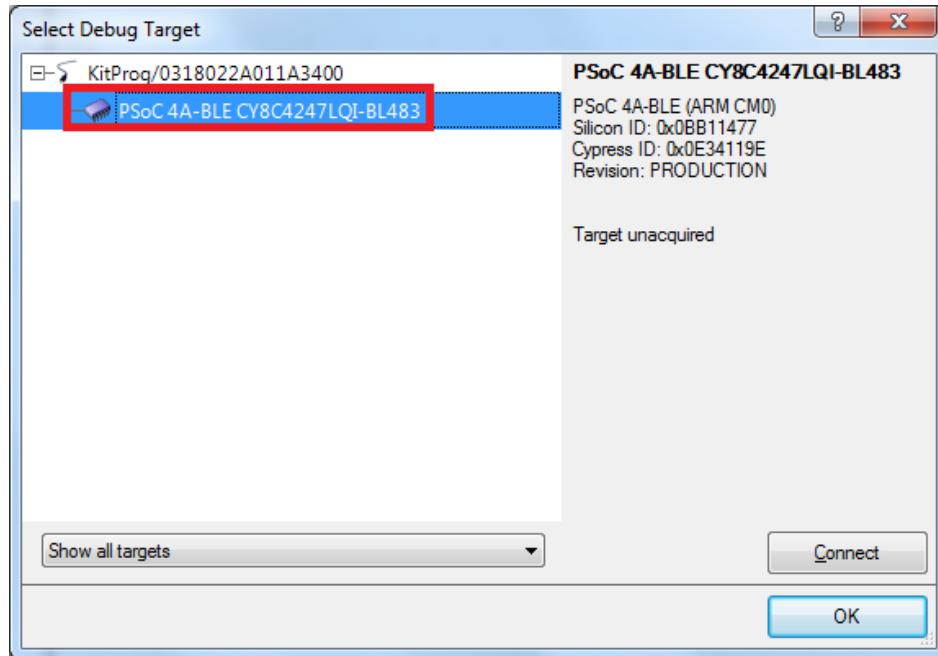
If you are developing on your own hardware, you need a hardware debugger, for example, a Cypress CY8CKIT-002 MiniProg3. In PSoC Creator, choose **Debug > Select Debug Target**, as [Figure 42](#) shows.

Figure 42. Selecting Debug Target



1. In the **Select Debug Target** dialog box, click **Port Acquire**, and then click **Connect**, as [Figure 43](#) shows. Click **OK** to close the dialog box.

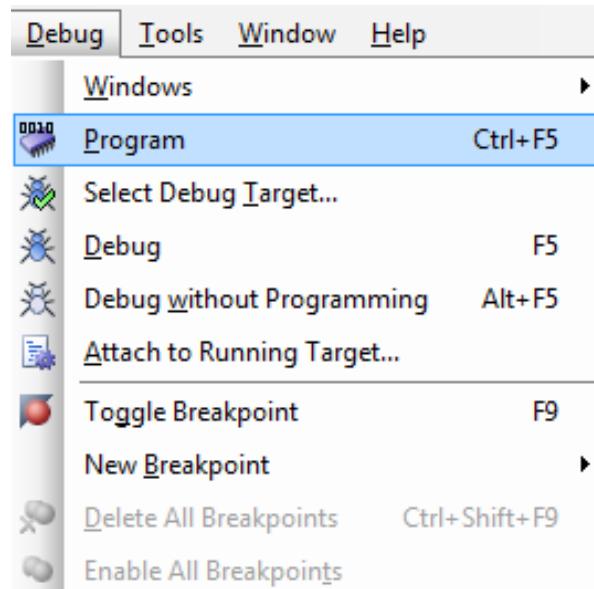
Figure 43. Connecting to a Device



If you are using your own hardware, make sure the **Port Setting** configuration in the **Select Debug Target** window for your programming hardware is configured per your setup.

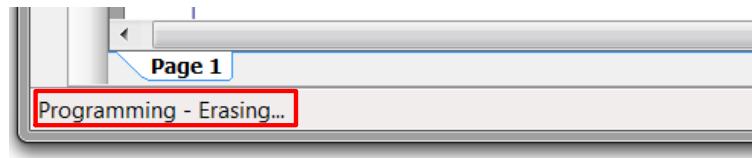
2. In PSoC Creator, choose **Debug > Program** to program the device with the project, as Figure 44 shows.

Figure 44. Programming the Device



3. You can view the programming status on the PSoC Creator status bar (lower left corner of the window), as Figure 45 shows,

Figure 45. Programming Status



## Part 4: Test Your Design

This section describes how to test your BLE design using the CySmart mobile apps and PC tool. The setup for testing your design using the BLE Pioneer Kit is shown in [Figure 15](#).

1. Turn on Bluetooth on your iOS or Android device.
2. Launch the CySmart app.
3. Press the reset switch on the BLE Pioneer Kit to start BLE advertisements from your design.
4. Pull down the CySmart app home screen to start scanning for BLE Peripherals, your device will now appear in the CySmart app home screen. Select your device to establish a BLE connection.
5. Select the “Find Me” Profile from the carousel view.
6. Select one of the Alert Level values on the Find Me **Profile** screen and observe the state of the LED on your device change per your selection.

A step-by-step configuration screenshot of the CySmart mobile app is shown in [Figure 46](#) and [Figure 47](#).

Figure 46. Testing with CySmart iOS App

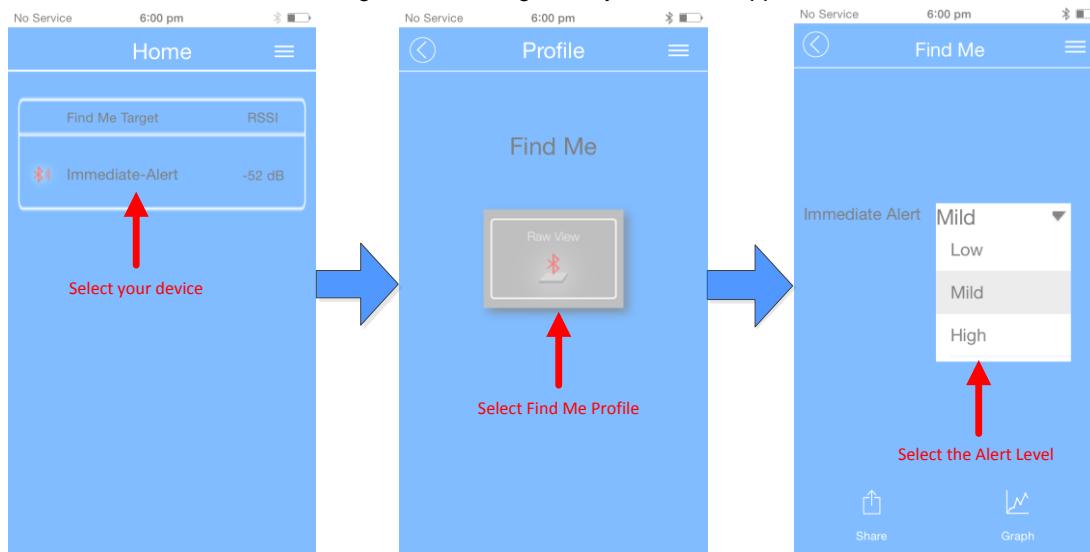
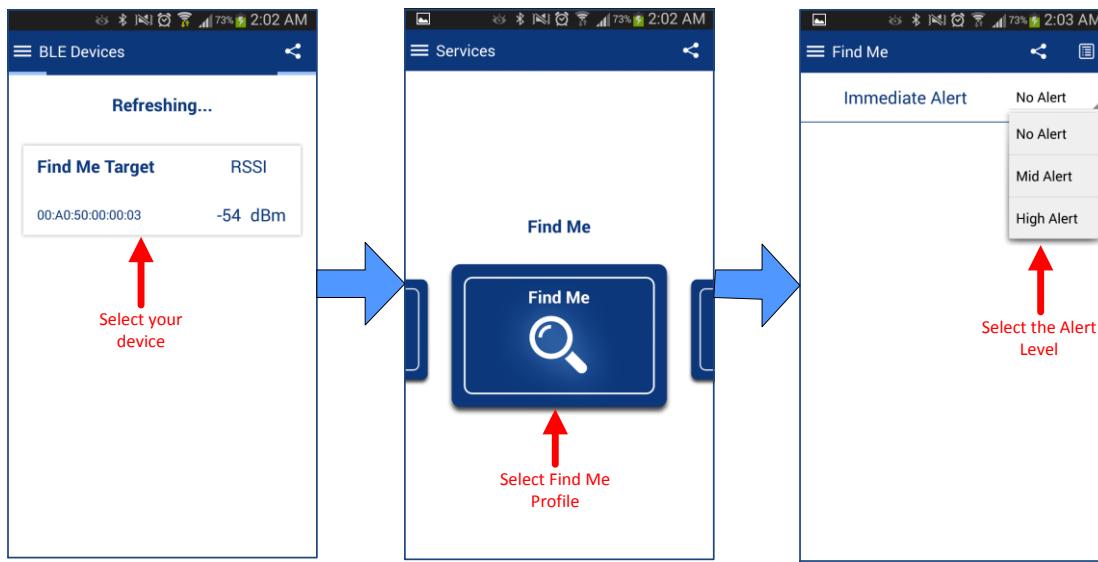


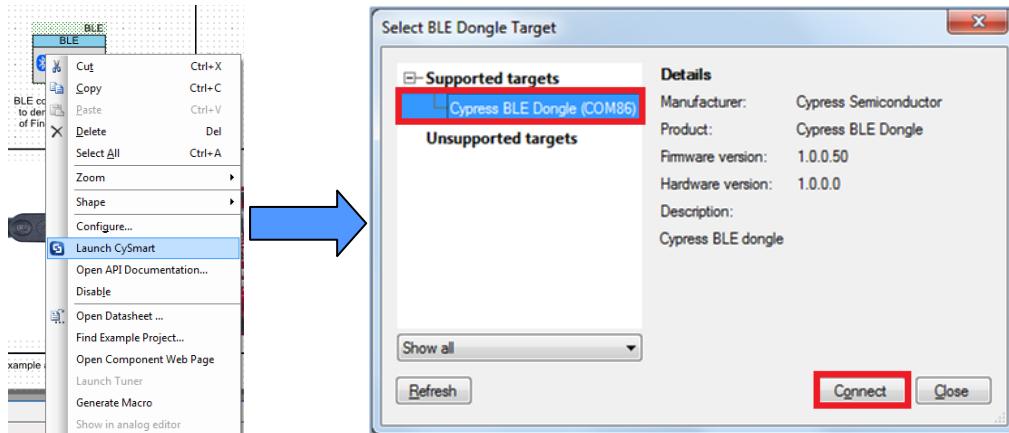
Figure 47. Testing with CySmart Android App



Similar to the CySmart mobile app, you can also use the CySmart Host Emulation Tool to establish a BLE connection with your design and perform read or write operations on BLE Characteristics.

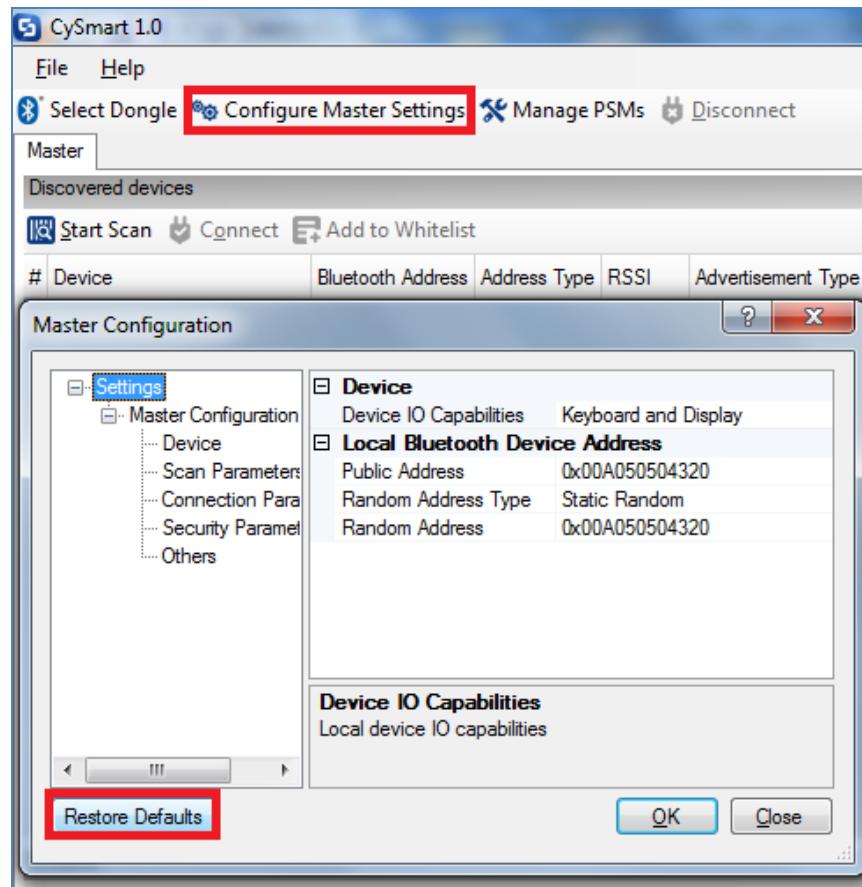
1. Connect the BLE Dongle to your Windows machine. Wait for the driver installation to be completed.
2. Launch the CySmart Host Emulation Tool; it automatically detects the BLE Dongle. Click **Refresh** if the BLE Dongle does not appear in the **Select BLE Dongle Target** pop-up window. Click **Connect**, as shown in Figure 48.

Figure 48. CySmart BLE Dongle Selection



3. Select **Configure Master Settings** and restore the values to the default settings, as shown in Figure 49.

Figure 49. CySmart Master Settings Configuration



4. Press the reset switch on the BLE Pioneer Kit to start BLE advertisements from your design.

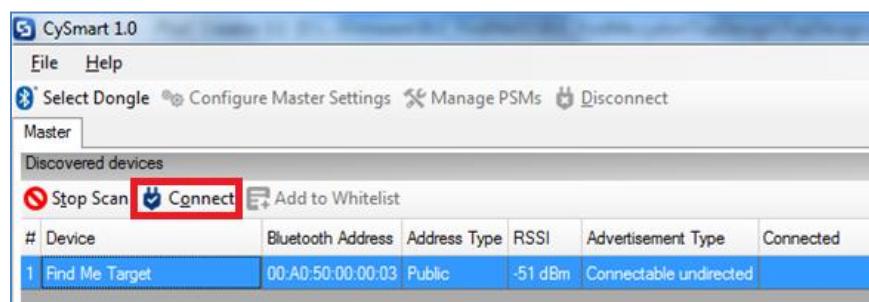
5. On the CySmart Host Emulation Tool, click **Start Scan**. Your device name should appear in the **Discovered devices** list, as shown in [Figure 50](#).

Figure 50. CySmart Device Discovery



6. Select your device and click **Connect** to establish a BLE connection between the CySmart Host Emulation Tool and your device, as shown in [Figure 51](#).

Figure 51. CySmart Device Connection



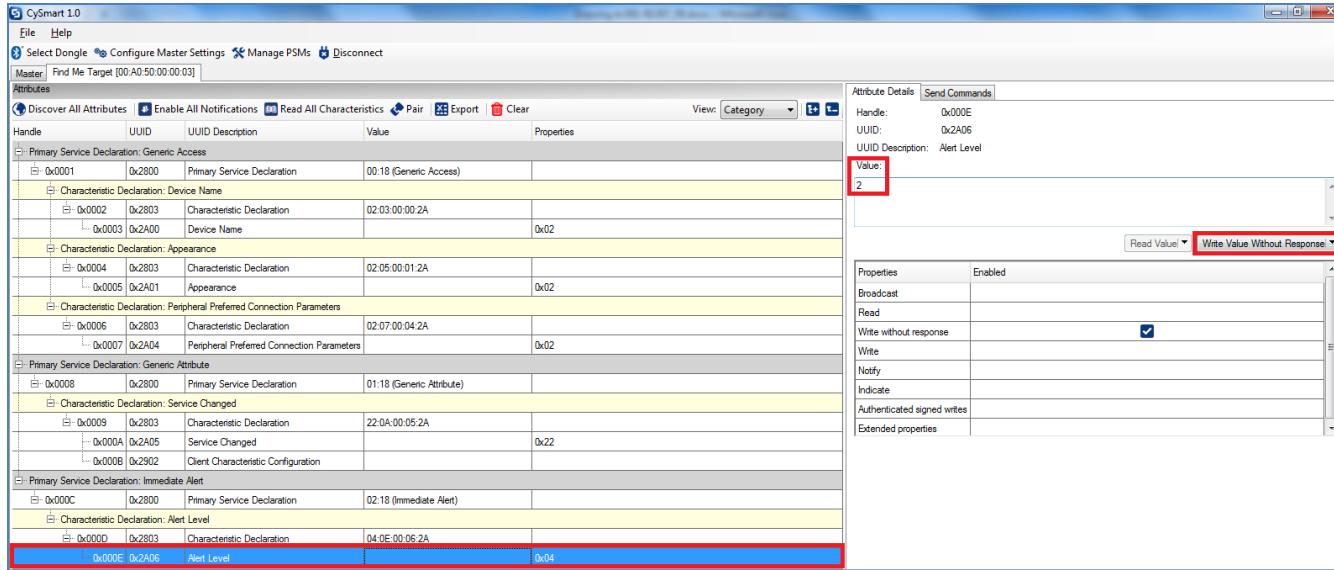
7. Once connected, discover all the Attributes on your design from the CySmart Host Emulation Tool, as shown in [Figure 52](#).

Figure 52. CySmart Attribute Discovery



8. Write a value of 0, 1, or 2 to the Alert Level Characteristic under the Immediate Alert Service, as [Figure 53](#) shows. Observe the state of the LED on your device change per your Alert Level Characteristic configuration.

Figure 53. Testing with CySmart Host Emulation Tool



## Design Source

The functional PSoC Creator project for the BLE example design described in this application note is not distributed on the application note webpage, but is part of PSoC Creator example projects. You can find the functional PSoC Creator project for this design in PSoC Creator 3.1 SP1 or later by selecting **File > Example Project**, and then selecting **Keyword** as **Find Me > BLE\_FindMe**

## Summary

This application note explored the basics of the BLE protocol and PSoC 4 BLE device architecture and development tools. PSoC 4 BLE is a truly programmable embedded system-on-chip, integrating BLE radio, configurable analog and digital peripheral functions, memory, and an ARM Cortex-M0 microcontroller on a single chip. Because of the integrated features and low-power modes, PSoC 4 BLE is an ideal choice for battery-operated wearable, health, and fitness BLE applications.

This application note also guided you to a comprehensive collection of resources to accelerate in-depth learning about PSoC 4 BLE.

## Related Application Notes

[AN94020 – Getting Started with PRoC BLE](#)

[AN91445 – Antenna Design Guide](#)

[AN91184 – PSoC 4 BLE - Designing BLE Applications](#)

[AN91162 – Creating a BLE Custom Profile](#)

[AN92584 – Designing for Low Power and Estimating Battery Life for BLE Applications](#)

## About the Author

Name: Krishnaprasad M V (KRIS).

Title: Applications Engineer Senior Staff

Background: Krishnaprasad has a BSEE from National Institute of Engineering, Mysore, India.

## Appendix A: BLE Device Family Comparison

Table 2 summarizes the features and capabilities of the BLE device family from Cypress.

Table 2. BLE Device Families

Features	Device Family		
	CY8C41x7-BL	CY8C42x7-BL	CYBL10X6X
BLE Subsystem	BLE radio and link-layer hardware blocks with Bluetooth 4.2-compatible protocol stack	BLE radio and link-layer hardware blocks with Bluetooth 4.2-compatible protocol stack	BLE radio and link-layer hardware blocks with Bluetooth 4.2-compatible protocol stack
CPU	24-MHz ARM Cortex-M0 CPU with single-cycle multiply	48-MHz ARM Cortex-M0 CPU with single-cycle multiply	48-MHz ARM Cortex-M0 CPU with single-cycle multiply
Flash Memory	128 KB	128 KB	128 KB
SRAM	16 KB	16 KB	16 KB
GPIOs	Up to 36	Up to 36	Up to 36
CapSense	Up to 35 sensors	Up to 35 sensors	Up to 35 sensors
CapSense Gestures	On selected devices	On selected devices	On selected devices
ADC	12-bit, 806-ksps SAR ADC with sequencer	12-bit, 1-MspS SAR ADC with sequencer	12-bit, 1-MspS SAR ADC with sequencer
Opamps	2 programmable opamps that are active in Deep-Sleep mode	4 programmable opamps that are active in Deep-Sleep mode	None
Comparators	2 low-power comparators with the wakeup feature	2 low-power comparators with the wakeup feature	None
Current DACs	One 7-bit, and one 8-bit	One 7-bit, and one 8-bit	None
Power Supply Range	1.9 V to 5.5 V	1.9 V to 5.5 V	1.9 V to 5.5 V
Low-Power Modes	Deep-Sleep mode at 1.3 µA Hibernate mode at 150 nA Stop mode at 60 nA	Deep-Sleep mode at 1.3 µA Hibernate mode at 150 nA Stop mode at 60 nA	Deep-Sleep mode at 1.3 µA Hibernate mode at 150 nA Stop mode at 60 nA
Segment LCD Drive	4-COM, 32-segment LCD drive on select devices	4-COM, 32-segment LCD drive on select devices	4-COM, 32-segment LCD drive on select devices
Serial Communication	2 independent serial communication blocks (SCBs) with programmable I <sup>2</sup> C, SPI, or UART	2 independent SCBs with programmable I <sup>2</sup> C, SPI, or UART	1 or 2 independent SCBs with programmable I <sup>2</sup> C, SPI, or UART
Timer Counter Pulse-Width Modulator (TCPWM)	4	4	4
Universal Digital Blocks (UDBs)	None	4, each with 8 macrocells and one data path. Can be used to synthesize additional digital peripherals (Timer, Counter, PWM) or communication interfaces (UART, SPI)	None
Additional Digital Peripherals (I <sup>2</sup> S, PWM)	None	Yes (UDB-based digital peripherals on select devices)	Yes (fixed-function blocks on select devices)
Clocks	3-MHz to 24-MHz IMO 32-kHz ILO 24-MHz ECO 32-kHz WCO	3-MHz to 48-MHz IMO 32-kHz ILO 24-MHz ECO 32-kHz WCO	3-MHz to 48-MHz IMO 32-kHz ILO 24-MHz ECO 32-kHz WCO

Features	Device Family		
	CY8C41x7-BL	CY8C42x7-BL	CYBL10X6X
Power Supply Monitoring	Power-on reset (POR) Brown-out detection (BOD) Low-voltage detection (LVD)	POR BOD LVD	POR BOD LVD
Package	56-QFN (7.0 × 7.0 × 0.6 mm) and 68-WLCSP (3.52 × 3.91 × 0.55 mm)	56-QFN (7.0 × 7.0 × 0.6 mm) and 68-WLCSP (3.52 × 3.91 × 0.55 mm)	56-QFN (7.0 × 7.0 × 0.6 mm) and 68-WLCSP (3.52 × 3.91 × 0.55 mm)

## Appendix B: Cypress Terms of Art

This section lists the most commonly used terms that you might encounter while working with Cypress's PSoC family of devices.

**Component Configuration Tool:** Simple GUI in PSoC Creator that is embedded in each Component. It is used to customize the Component parameters and is accessed by right-clicking a Component.

**Components:** Free embedded ICs represented by an icon in PSoC Creator software. These are used to integrate multiple ICs and system interfaces into one PSoC Component that is inherently connected to the MCU via the main system bus. For example, the BLE Component creates Bluetooth Smart products in minutes. Similarly, you can use the Programmable Analog Components for sensors.

**MiniProg3:** A programming hardware for development that is used to program PSoC devices on your custom board or PSoC development kits that do not support a built-in programmer.

**PSoC:** A programmable, embedded design platform that includes a CPU, such as the 32-bit ARM Cortex-M0, with both analog and digital programmable blocks. It accelerates embedded system design with reliable, easy-to-use solutions, such as touch sensing, and enables low-power designs.

**PSoC 4 BLE:** A PSoC 4 IC with an integrated BLE radio that includes a royalty-free BLE protocol stack compatible with the Bluetooth 4.2 specification.

**PSoC Creator:** PSoC 3, PSoC 4, and PSoC 5LP Integrated Design Environment (IDE) software that installs on your PC and allows concurrent hardware and firmware design of PSoC systems, or hardware design followed by export to other popular IDEs.

**PSoC Programmer:** A flexible, integrated programming application for programming PSoC devices. PSoC Programmer is integrated with PSoC Creator to program PSoC 3, PSoC 4, PRoC, and PSoC 5LP designs.

## Appendix C: Cypress BLE Development Tools

### PSoC 4 BLE Development Kit

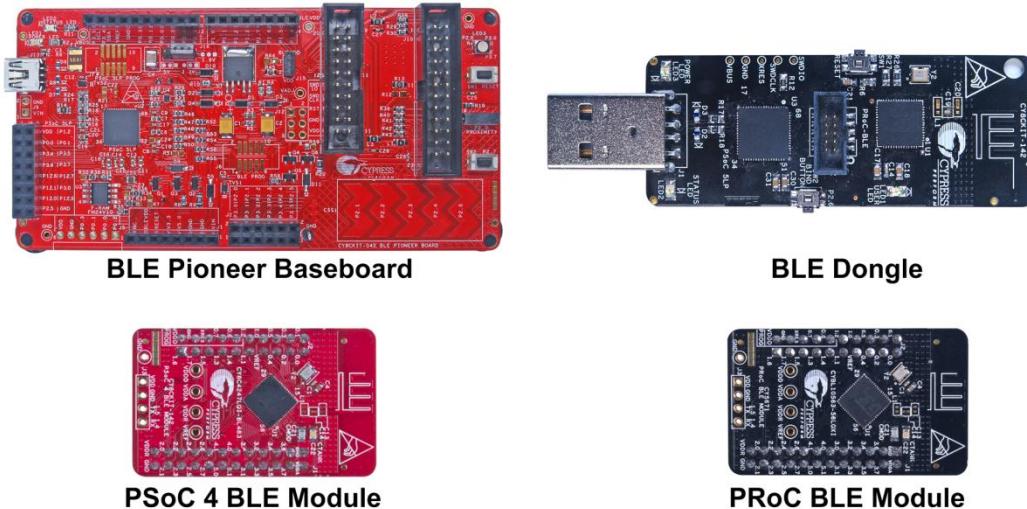
The BLE Pioneer Kit shown in [Figure 54](#) is a BLE development kit from Cypress that supports both PSoC 4 BLE and PRoC BLE family of devices. This kit consists of pluggable PSoC 4 BLE (and PRoC BLE) modules that connect to a BLE Pioneer Baseboard. The kit can be powered either with a coin-cell battery or through the USB interface.

The BLE Pioneer Baseboard and the BLE module combination enables you to develop battery-operated low-power BLE designs that work in conjunction with standard, Arduino Uno connector-compliant shields or the onboard PSoC 4 BLE device capabilities such as the CapSense user interface.

The BLE Pioneer Baseboard has an onboard programming and debugging interface, making the BLE design debugging through the PSoC Creator IDE quick and easy. The BLE Pioneer Kit has an additional header that supports interfacing with Pmod™ daughter cards from third-party vendors such as Digilent®. The kit also has a BLE Dongle that acts as a BLE link master and works with CySmart Host Emulation Tool to provide a BLE host emulation platform on non-BLE Windows PCs.

The kit consists of a set of BLE example projects and documentation that help you get started on developing your own BLE applications. Visit [www.cypress.com/go/CY8CKIT-042-BLE](http://www.cypress.com/go/CY8CKIT-042-BLE) to get the latest updates on the kit and download the kit design, example projects, and documentation files.

Figure 54. BLE Pioneer Kit



### CySmart Host Emulation Tool

The CySmart Host Emulation Tool is a Windows application that emulates a BLE Central device using the BLE Pioneer Kit's BLE Dongle; see [Figure 15](#). It is installed as part of the BLE Pioneer Kit installation and can be launched from right-click options in the BLE Component. It provides a platform for you to test your PSoC 4 BLE Peripheral implementation over GATT or L2CAP connection-oriented channels by allowing you to discover and configure the BLE Services, Characteristics, and Attributes on your Peripheral.

Operations that you can perform with CySmart Host Emulation Tool include, but are not limited to:

- Scan BLE Peripherals to discover available devices to which you can connect.
- Discover available BLE Attributes including Services and Characteristics on the connected Peripheral device.
- Perform read and write operations on Characteristic values and descriptors.
- Receive Characteristic notifications and indications from the connected Peripheral device.
- Establish a bond with the connected Peripheral device using BLE Security Manager procedures.
- Establish a BLE L2CAP connection-oriented session with the Peripheral device and exchange data per the Bluetooth 4.2 specification.

Figure 55 and Figure 56 show the user interface of CySmart Host Emulation Tool. For more information on how to set up and use this tool, see the CySmart user guide from the **Help** menu.

Figure 55. CySmart Host Emulation Tool Master Device Tab

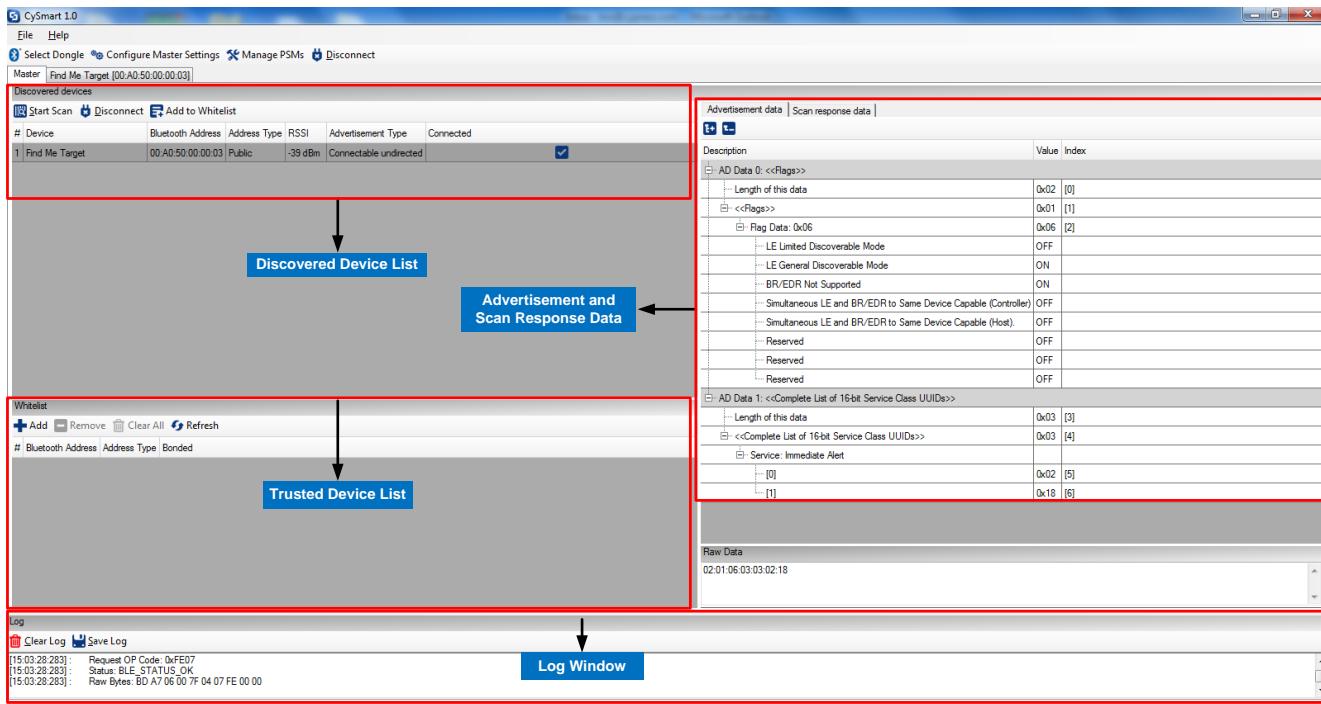
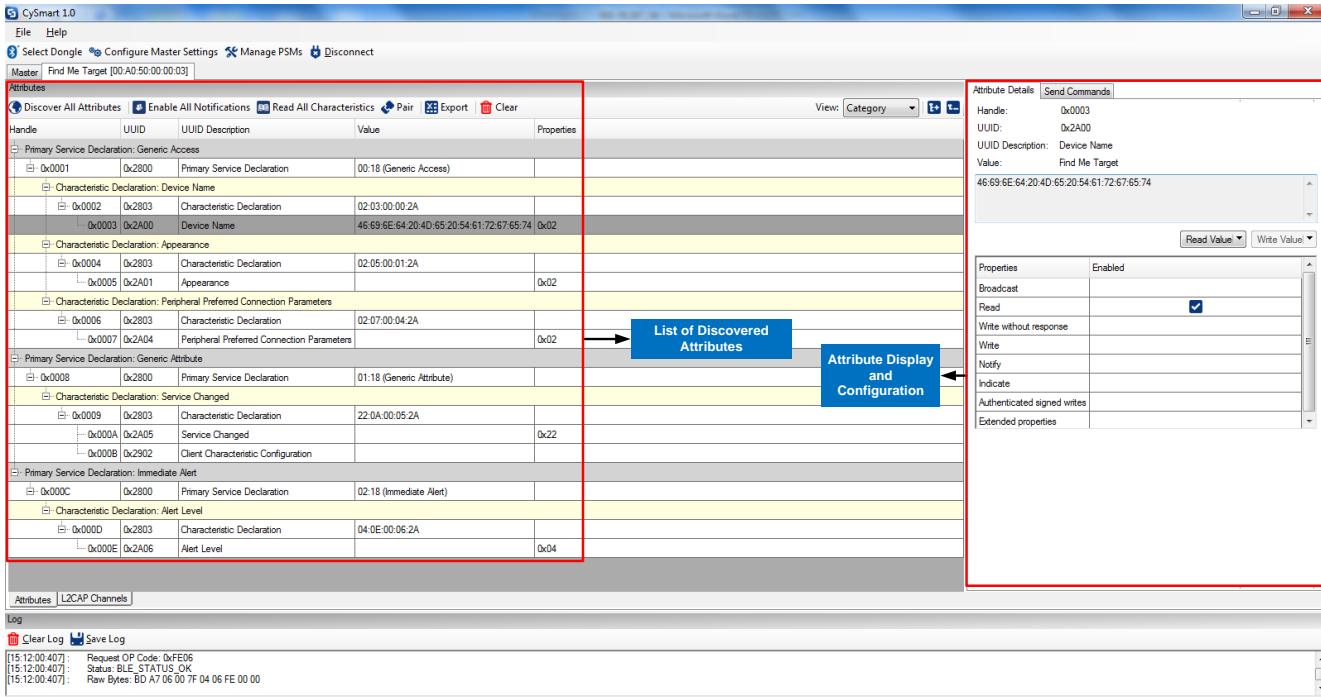


Figure 56. CySmart Host Emulation Tool Peripheral Device Attributes Tab



## CySmart Mobile App

In addition to the PC tool, you can download the CySmart mobile app for iOS or Android from the respective app stores. This app uses the iOS Core Bluetooth framework and the Android built-in platform framework for BLE respectively to configure your BLE-enabled smartphone as a Central device that can scan and connect to Peripheral devices.

The mobile app supports SIG-adopted BLE standard Profiles through an intuitive GUI and abstracts the underlying BLE Service and Characteristic details. In addition to the BLE standard Profiles, the app demonstrates a custom Profile implementation using Cypress's LED and CapSense demo examples. [Figure 57](#) and [Figure 58](#) shows a set of CySmart app screenshots for the Heart Rate Profile user interface. For a description of how to use the app with BLE Pioneer Kit example projects, see the BLE Pioneer Kit guide.

Figure 57. CySmart iOS App Heart Rate Profile Example

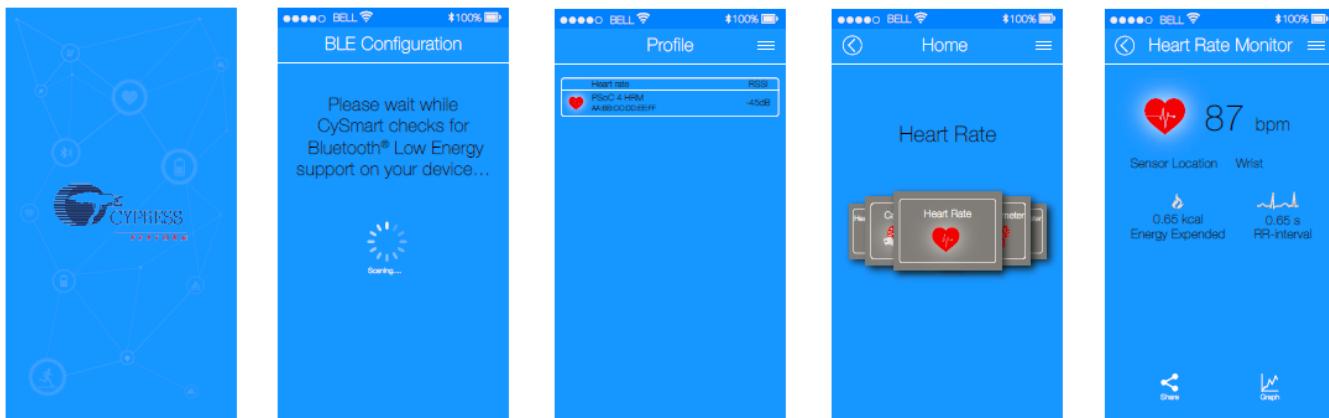
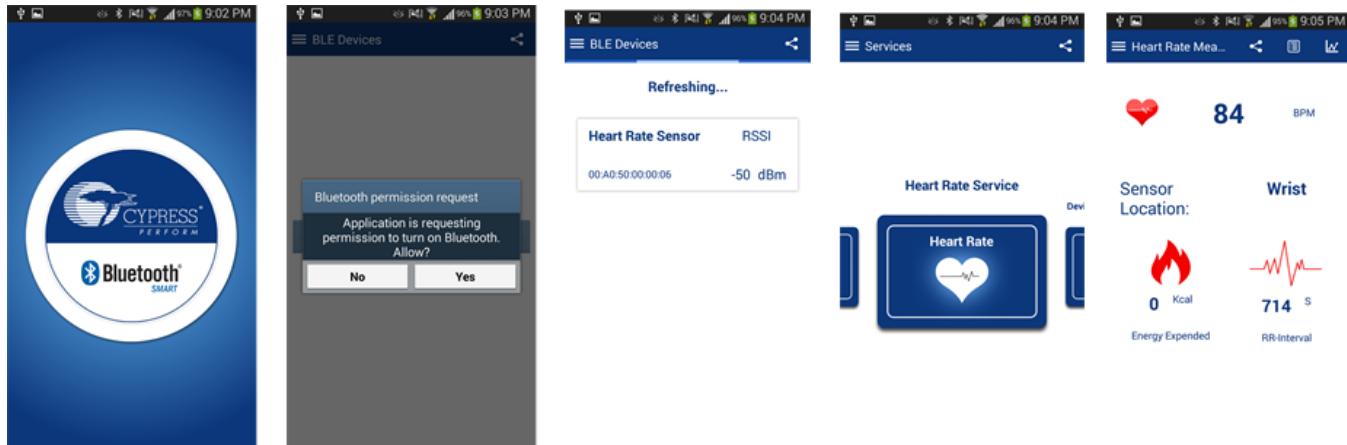


Figure 58. CySmart Android App Heart Rate Profile Example



## Appendix D: PSoC 4 BLE Device

### Bluetooth Low Energy Subsystem (BLESS)

The BLE subsystem contains the physical layer (PHY) and link-layer engine with an embedded AES-128 security engine.

The physical layer consists of a digital PHY and RF transceiver compliant with the Bluetooth 4.2 specification. The transceiver transmits and receives GFSK packets at 1 Mbps over the 2.4-GHz ISM band. The baseband controller is a composite hardware/firmware implementation that supports both master and slave modes. The key protocol elements such as HCI and link control are implemented in firmware, while time-critical functions such as encryption, CRC, data whitening, and access code correlation are implemented in hardware.

The BLESS is Bluetooth 4.2 compliant with support for all the features of the Bluetooth 4.0 specification and some additional features of the Bluetooth 4.2 specification such as low-duty-cycle advertising, LE ping, and L2CAP connection-oriented channels. The BLESS block also contains an external crystal oscillator (ECO) and watch crystal oscillator (WCO) that are required for generating an accurate RF frequency and keeping the time between successive connection intervals on the BLE link respectively. The ECO and the WCO from the BLESS block can also be used as high-frequency and low-frequency clock sources for the PSoC 4 BLE system. See the [Clocking System](#) section for details.

The BLESS supports five functional power modes: Deep-Sleep, Sleep, Idle, Transmit, and Receive.

**Note** The power modes discussed in this section are specific to the BLESS block. For PSoC 4 BLE system power modes, see [Low-Leakage Power Modes](#).

#### Deep-Sleep Mode

Deep-Sleep mode is the lowest power functional mode supported by the BLESS. In this mode, the radio is off. This mode is entered for maximum power saving during an advertising or connection interval after the packet transmission and reception is complete. The ECO can be turned off in this mode for power saving; the WCO, which is the low-frequency clock, is on for maintaining the BLE link-layer timing reference logic. The CPU controls the entry to and exit from this state.

#### Sleep Mode

In Sleep mode, the radio is off. The block maintains all the configurations. The ECO and WCO are turned on, but the clock to the core BLESS logic is turned off. The CPU controls the entry to and exit from this state.

#### Idle Mode

The Idle mode is the preparation state for the Transmit and Receive states. In this state, the radio is turned off but the link-layer clock is enabled for the link-layer logic so that the CPU starts the protocol state machines.

#### Transmit Mode

Transmit mode is the active functional mode; all the blocks within BLESS are powered on. The link-layer clock is enabled to complete the logic within the link layer and RF-PHY. In this mode, RF-PHY gets 1-Mbps of serial data from the link layer and transmits the 2.4-GHz GFSK-modulated data to the antenna port. BLE enters Transmit mode from Idle mode.

#### Receive Mode

This mode enables the BLESS to move into the receive state to perform BLE-specific receiver operations. RF-PHY translates the 1-Mbps data received from the RF analog block and forwards it to the link-layer controller after demodulation.

A summary of the BLESS power modes and operational sub-blocks is shown in [Table 3](#).

Table 3. BLESS Power Modes

BLESS Power Mode	ECO	WCO	RF Tx	RF Rx	BLESS Core
Deep-Sleep	Off	On	Off	Off	Off
Sleep	On	On	Off	Off	Off
Idle	On	On	Off	Off	On
Transmit	On	On	On	Off	On
Receive	On	On	Off	On	On

## ARM Cortex-M0 and Memory

PSoC 4 BLE has a 32-bit ARM Cortex-M0 CPU, capable of operating at a maximum frequency of 48 MHz, providing a 43-DMIPS performance. The CPU supports single-cycle 32-bit multiplication. PSoC 4 BLE has 16 KB of SRAM and 128 KB of flash memory that can service most of the BLE application use; the flash includes a read accelerator. The device also provides 512 bytes of supervisory flash area for you to store user-specific data such as BLE device address and encryption keys.

## Programmable Digital Peripherals

PSoC 4 BLE provides a rich set of digital peripherals including programmable serial communication blocks (SCBs), timer counter pulse width modulators (TCPWMs), and programmable logic arrays called universal digital blocks (UDBs).

### Programmable SCBs

PSoC 4 BLE has independent run-time programmable SCBs with I<sup>2</sup>C, SPI, or UART. The SCB supports the following features:

- Standard SPI master and slave functionality with Motorola®, Texas Instruments®, and National Semiconductor® protocols
- Standard UART functionality with smart-card reader, Local Interconnect Network (LIN), and Infrared Data Association (IrDA) protocols
- Standard I<sup>2</sup>C master and slave functionality
- SPI and EZI<sup>2</sup>C mode, which allows operation without CPU intervention
- Low-power (Deep-Sleep) mode of operation for SPI and I<sup>2</sup>C protocols (using an external clock)

For more information, refer to the [PSoC 4 SCB Component datasheet](#).

### Programmable TCPWMs

PSoC 4 BLE has four programmable 16-bit TCPWM blocks. Each TCPWM can implement a 16-bit timer, counter, PWM, or quadrature decoder. TCPWMs provide complementary outputs and selectable start, reload, stop, count, and capture event signals. The PWM mode supports center-aligned, edge, and pseudo random operations.

For more information, refer to the [PSoC 4 TCPWM Component datasheet](#).

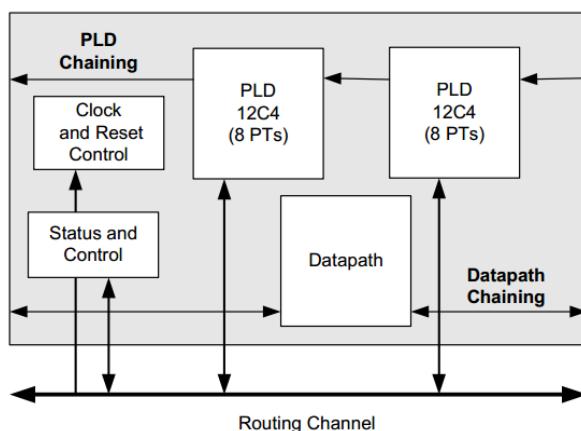
### Universal Digital Blocks

UDBs are programmable logic blocks that provide functionalities similar to CPLD and FPGA blocks, as [Figure 59](#) shows. UDBs allow you to create a variety of digital functions such as timer, counter, PWM, pseudo random sequence (PRS), CRC, shift register, SPI, UART, I2S, and custom combinational and sequential logic circuits.

Each UDB has two programmable logic devices (PLDs), each with 12 inputs and 8 product terms. PLDs can form registered or combinational sum-of-products logic. Additionally, an 8-bit single-cycle arithmetic logic unit (ALU), known as a “datapath,” is present in each UDB. The datapath helps with the efficient implementation of functions such as timer, counter, PWM, and CRC.

UDBs also provide a switched digital signal interconnect (DSI) fabric that allows signals from peripherals and ports to be routed to and through the UDBs for communication and control.

Figure 59. Universal Digital Block Diagram



You do not necessarily need to know any hardware description language (HDL) to use UDBs. PSoC Creator, Cypress's development tool for PSoC 4 BLE, can generate the required function for you from a schematic. If required, advanced users can implement custom logic on UDBs using Verilog.

For more information, refer to the following application notes.

- [AN62510 – Implementing State Machines with PSoC 3, PSoC 4, and PSoC 5LP](#)
- [AN82156 – PSoC 3, PSoC 4, and PSoC 5LP – Designing PSoC Creator Components with UDB Datapaths](#)
- [AN82250 – PSoC 3, PSoC 4, and PSoC 5LP – Implementing Programmable Logic Designs with Verilog](#)

## Applications

The use of programmable digital peripherals in BLE applications is shown in [Table 4](#).

Table 4. Applications of Programmable Digital Peripherals

Applications	Digital Peripherals
Sensor Hub	I <sup>2</sup> C (digital sensor interface), PWM (actuators), I2S (voice input)
Health and Fitness	PWM (user interface), Counter (waveform peak measurement), SPI (external memory interface)
Industrial	UART (modbus), Counter (event counting)
Home Automation	PWM (garage door control), PWM (lighting control)

## Programmable Analog

PSoC 4 BLE provides the industry's best-in-class analog integration. The analog system includes Continuous Time Block mini (CTBm) blocks, a fast 12-bit SAR ADC, low-power comparators, capacitive touch-sensing (CapSense), and a segment LCD direct drive.

### Continuous Time Block mini (CTBm)

PSoC 4 BLE contains two CTBm blocks, each consisting of two programmable opamps and a switch matrix. You can configure each opamp individually as a comparator, voltage follower, or an opamp with external feedback. If required, the CTBm block can be configured to function even in device Deep-Sleep mode.

For more information, refer to the following Component datasheets:

- [PSoC 4 Opamp](#)
- [PSoC 4 Voltage Comparator](#)

### SAR ADC with Hardware Sequencer

PSoC 4 BLE has a 12-bit, 1-MspS Successive Approximation Register (SAR) ADC with input channels that support programmable resolution and single-ended or differential input options. The number of GPIOs limits the number of ADC input channels that can be implemented.

The SAR ADC has a hardware sequencer that can perform an automatic scan on as many as eight channels without CPU intervention. It also supports preprocessing operations such as accumulation and averaging of the output data on these eight channels.

You can trigger a scan with a variety of methods, such as firmware, timer, pin, or UDB, giving you additional design flexibility.

For more information, refer to the [PSoC 4 SAR ADC Component datasheet](#).

### Low-Power Comparators

PSoC 4 BLE devices have low-power comparators capable of operating in all system power modes except the [Stop](#) mode. In a power-sensitive design, when the device goes into low-power modes, you can use the low-power comparator to monitor analog inputs and generate an interrupt that can wake up the system.

For more information, refer to the [PSoC 4 Low-Power Comparator Component datasheet](#).

## Capacitive Touch Sensing (CapSense)

Capacitive touch sensors use human-body capacitance to detect the presence of a finger on or near a sensor. Capacitive sensors are aesthetically superior, easy to use, and have long lifetimes.

The CapSense feature in PSoC 4 BLE offers unprecedented signal-to-noise ratio; best-in-class liquid tolerance; and a wide variety of sensor types such as buttons, sliders, track pads, and proximity sensors.

A Cypress-supplied software Component makes capacitive sensing design very easy; the Component supports an automatic hardware-tuning feature called SmartSense™ and provides a gesture-recognition library for trackpads and proximity sensors.

Two current DACs (IDACs), one 7-bit and one 8-bit, in the CapSense block are available for general-purpose use if capacitive sensing is not used. The comparator in the CapSense block is also available for general-purpose use.

For more information, see the [PSoC 4 CapSense Design Guide](#).

## Segment LCD Direct Driver

Most low-power, portable, handheld devices such as glucose meters, multimeters, and blood pressure monitors use a segment LCD to display information. Segment LCDs typically require an external driver to interface with a microcontroller. PSoC 4 BLE includes an integrated low-power LCD driver that can directly drive segment LCD glass.

PSoC 4 BLE can drive LCDs with as many as 4 common and 32 segment electrodes. The segment LCD driver can retain a static display in Deep-Sleep mode with a system current consumption as low as 7  $\mu$ A.

For more information, see [AN87391 – PSoC 4 Segment LCD Direct Drive](#).

## Applications

The use of programmable analog peripherals in different BLE applications is listed in [Table 5](#).

Table 5. Applications of Programmable Analog Peripherals

Applications	Analog Peripherals
Health and Fitness	Opamp: TIA (heart-rate measurement) Opamp: Follower (analog- reference buffer) ADC (sampling heart-rate signal) CapSense (user interface on a wrist band) Segment LCD (display on a wrist band)
Sensor Hub	ADC (analog sensor interface) Analog Mux (multiple-sensor input) Opamps (signal amplifier) Segment LCD (UI)
Industrial	ADC: Differential mode (temperature measurement) IDAC (temperature-sensor drive) Low-Power Comparator (wakeup on threshold detection) Opamp: PGA (4-mA to 20-mA current-loop system)
Home Automation	Opamp: PGA (motion sensor, light sensor) Comparator (door sensors) ADC and opamp filter (smoke detector)

## System-Wide Resources

This section explains the system-wide resources available for all peripherals in PSoC 4 BLE.

### Low-Leakage Power Modes

PSoC 4 BLE offers the following power modes. Note that these are PSoC 4 BLE device power modes, which are different from the power modes described in the [Bluetooth Low Energy Subsystem \(BLESS\)](#) section.

- **Active mode:** This is the primary mode of operation. In this mode, all peripherals are available.

- **Sleep mode:** In this mode, the CPU is in Sleep mode, SRAM is in retention, and all peripherals are available. Any interrupt wakes up the CPU and returns the system to Active mode.
- **Deep-Sleep mode:** In this mode, the high-frequency clock (IMO) and all high-speed peripherals are off. Optionally, the low-frequency clocks (32-kHz ILO and WCO) and low-speed peripherals are available. Interrupts from low-speed, asynchronous, or low-power analog peripherals can cause a wakeup. The current consumption in this mode is 1.3  $\mu$ A.
- **Hibernate mode:** This power mode provides a best-in-class current consumption of 150 nA while retaining SRAM and the ability to wake up from an interrupt generated by a low-power comparator or a GPIO.
- **Stop mode:** This power mode retains the GPIO states. Wakeup is possible from a fixed WAKEUP pin. The current consumption in this mode is only 60 nA.

You can use a combination of Sleep, Deep-Sleep, Hibernate, and Stop modes in a battery-operated BLE system to achieve best-in-class system power with longer battery life.

Table 6 shows the dependency between PSoC 4 BLE system power modes and BLESS power modes. All these dependencies are handled by simple APIs; see the [Main Loop and Low-Power](#) section for an example.

In a typical BLE application such as heart-rate monitoring, the PSoC 4 BLE device will be in Active mode while measuring the heart rate, in Sleep mode while the BLE radio is transmitting or receiving packets, in Deep-Sleep mode between consecutive BLE connection intervals, and in Hibernate or Stop mode on BLE advertisement timeout.

Table 6. PSoC 4 BLE Power Modes

BLESS Modes	PSoC 4 BLE System Power Modes				
	Active	Sleep	Deep-Sleep	Hibernate	Stop
Transmit	✓	✓	✗	✗	✗
Receive	✓	✓	✗	✗	✗
Idle	✓	✓	✗	✗	✗
Sleep	✓	✓	✗	✗	✗
Deep-Sleep	✓	✓	✓	✗	✗
Powered	✗	✗	✗	✓	✓

## Power Supply and Monitoring

PSoC 4 BLE is capable of operating from a single 1.9-V to 5.5-V supply. There are multiple internal regulators to support the different device power modes. PSoC 4 BLE has three types of voltage-monitoring capabilities: POR, BOD, and LVD.

## Clocking System

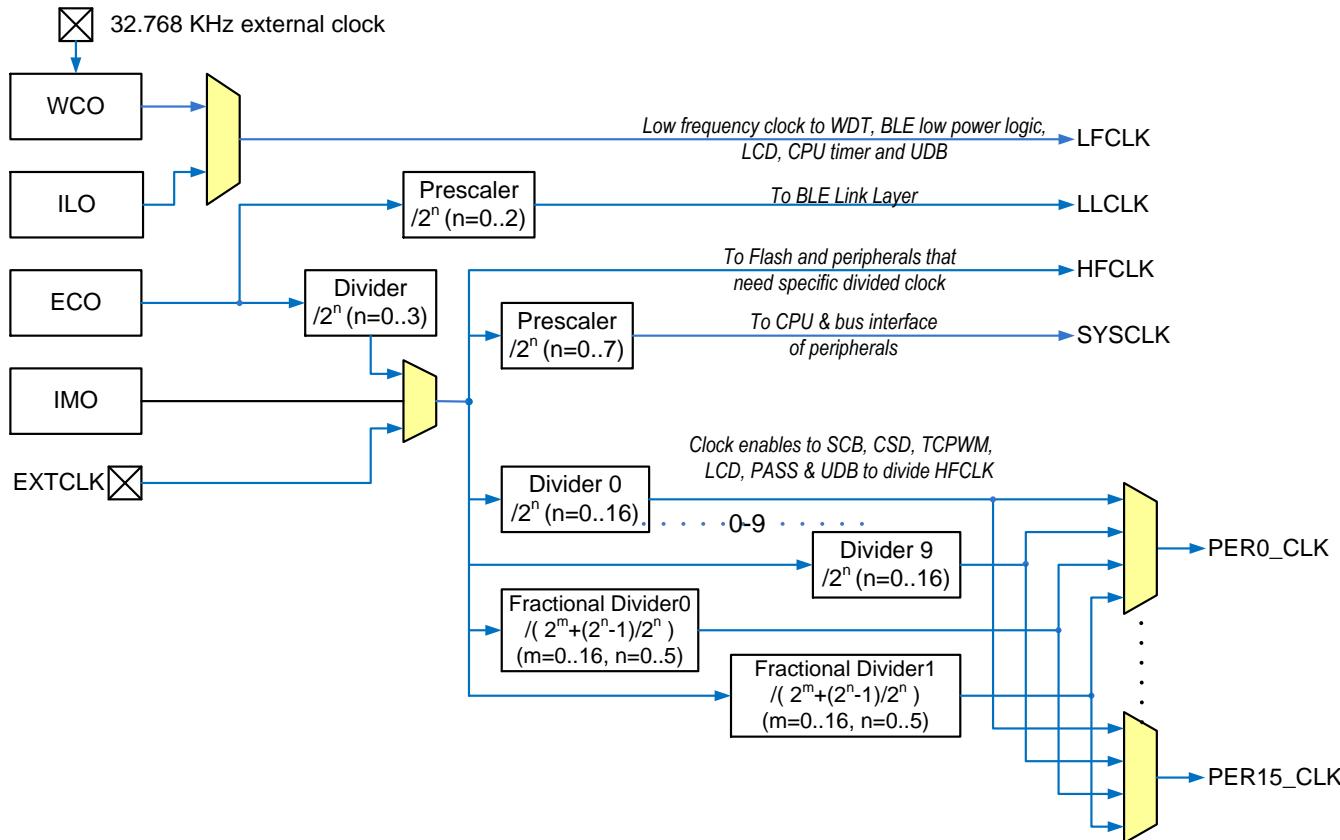
PSoC 4 BLE has the following clock sources:

- **Internal main oscillator (IMO):** The IMO is the primary source of internal clocking in PSoC 4 BLE. The CPU and all high-speed peripherals can operate from the IMO or an external crystal oscillator (ECO). PSoC 4 BLE has multiple peripheral clock dividers operating from either the IMO or the ECO, which generate clocks for high-speed peripherals. The IMO can generate clocks in the range of 3 MHz to 48 MHz in 1-MHz increments with an accuracy of  $\pm 2$  percent.
- **Internal low-speed oscillator (ILO):** The ILO is a very-low-power 32-kHz oscillator, which primarily generates clocks for low-speed peripherals operating in Deep-Sleep mode except the BLESS (see [WCO](#)).
- **External crystal oscillator (ECO):** The external crystal oscillator with a built-in tunable crystal load capacitance is used to generate a highly accurate 24-MHz clock. It is primarily used to clock the BLE subsystem that generates the RF clocks. The high-accuracy ECO clock can also be used as a clock source for the PSoC 4 BLE device's high-frequency clock (HFCLK).

- **Watch crystal oscillator (WCO):** The 32-kHz WCO is used as one of the sources for LFCLK (along with ILO). WCO is used to accurately maintain the time interval for BLE advertising and connection events. Similar to ILO, WCO is also available in all modes except the Hibernate and Stop modes.

Figure 60 shows the clocking architecture of a PSoC 4 BLE device.

Figure 60. PSoC 4 BLE Clocking System



### Device Security

PSoC 4 BLE provides a number of options to protect the flash memory from unauthorized access or copying. Each row of flash has a single protection bit; these bits are stored in a supervisory flash row.

### Programmable GPIOs

The I/O system provides an interface between the CPU and the peripherals and the outside world. PSoC 4 BLE has up to 36 programmable GPIO pins. You can configure the GPIOs for CapSense, LCD, analog, or digital signals. PSoC 4 BLE GPIOs support multiple drive modes, drive strengths, and slew rates.

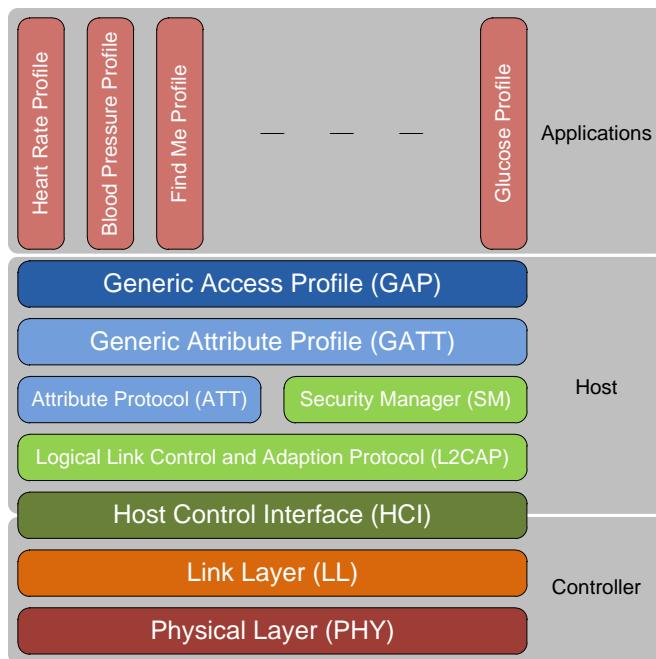
PSoC 4 BLE offers an intelligent routing system that gives multiple choices for connecting an internal signal to a GPIO. This flexible routing simplifies circuit design and board layout.

## Appendix E: BLE Protocol

### Overview

BLE, also known as Bluetooth Smart, was introduced by the Bluetooth SIG as a low-power wireless standard operating in the 2.4-GHz ISM band. [Figure 61](#) shows the BLE stack.

Figure 61. BLE Architecture



The BLE stack can be subdivided into three groups:

- **Controller:** A physical device that encodes the packet and transmits it as radio signals. On reception, the controller decodes the radio signals and reconstructs the packet.
- **Host:** A software stack consisting of various protocols and Profiles (Security Manager, Attribute Protocol, and so on) that manages how two or more devices communicate with one another.
- **Application:** A use case that uses the software stack and the controller to implement a particular functionality.

The following sections provide an overview of the multiple layers of the BLE stack, using the standard Heart Rate and Battery Service as examples. For a detailed BLE architecture description, see the Bluetooth 4.2 specification or the training videos on the [Bluetooth Developer](#) website.

### Physical Layer (PHY)

The physical layer transmits or receives digital data at 1 Mbps using Gaussian frequency-shift keying (GFSK) modulation in the 2.4-GHz ISM band. The BLE physical layer divides the ISM band into 40 RF channels with a channel spacing of 2 MHz, 37 of which are data channels and 3 are advertisement channels.

### Link Layer (LL)

The link layer implements key procedures to establish a reliable physical link (using an acknowledgement and flow-control-based architecture) and features that help make the BLE protocol robust and low power. Some link layer functions include:

- Advertising, scanning, creating, and maintaining connections to establish a physical link

- 24-bit CRC and AES-128-bit encryption for robust and secure data exchange
- Establishing fast connections and low-duty-cycle advertising for low-power operation
- Adaptive Frequency Hopping (AFH), which changes the communication channel used for packet transmission so that the interference from other devices is reduced

At the link layer, two roles are defined:

- **Master:** A smartphone is an example that configures the link layer in the master configuration.
- **Slave:** A heart-rate monitor device is an example that configures the link layer in the slave configuration.

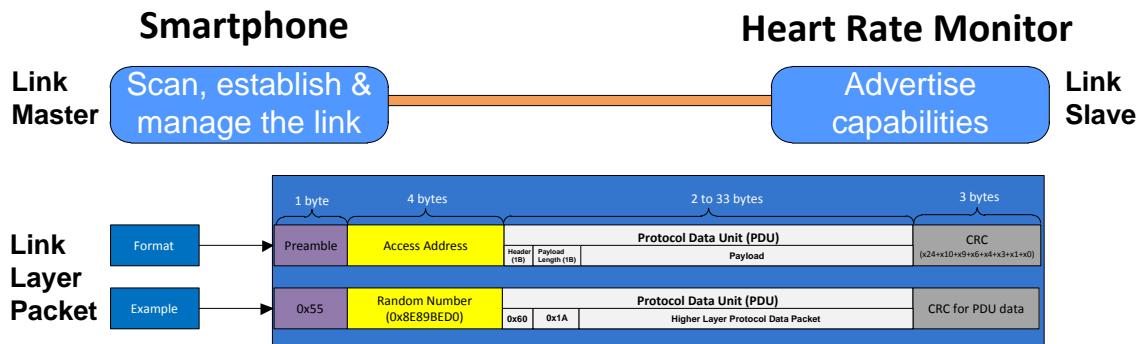
PSoC/PRoC BLE devices can operate in either configuration.

The link-layer slave is the one that advertises its presence to another link-layer master. A link-layer master receives the advertisement packets and can choose to connect to the slave based on the request from an application (see [Figure 62](#)). In this example implementation of a heart-rate monitor application, a heart-rate monitor device acts as the slave and sends the data to a smartphone, which acts as the master. A smartphone app then can display the reading on the smartphone.

PSoC/PRoC BLE devices implement the time-critical and processor-intensive parts of the link layer such as advertising, CRC, and AES encryption in hardware. Link-layer control operations such as entering the advertisement state and starting encryption are implemented in firmware.

[Figure 62](#) shows the BLE link-layer packet structure and sizes of the individual fields in the link-layer packet. The link-layer packet carries all upper layer data in its payload field. It has a 4-byte access address that is used to uniquely identify communications on a physical link, and ignore packets from a nearby BLE device operating in the same RF channel. 24-bit CRC provides data robustness.

Figure 62. BLE Link Layer Protocol



## Host Control Interface (HCI)

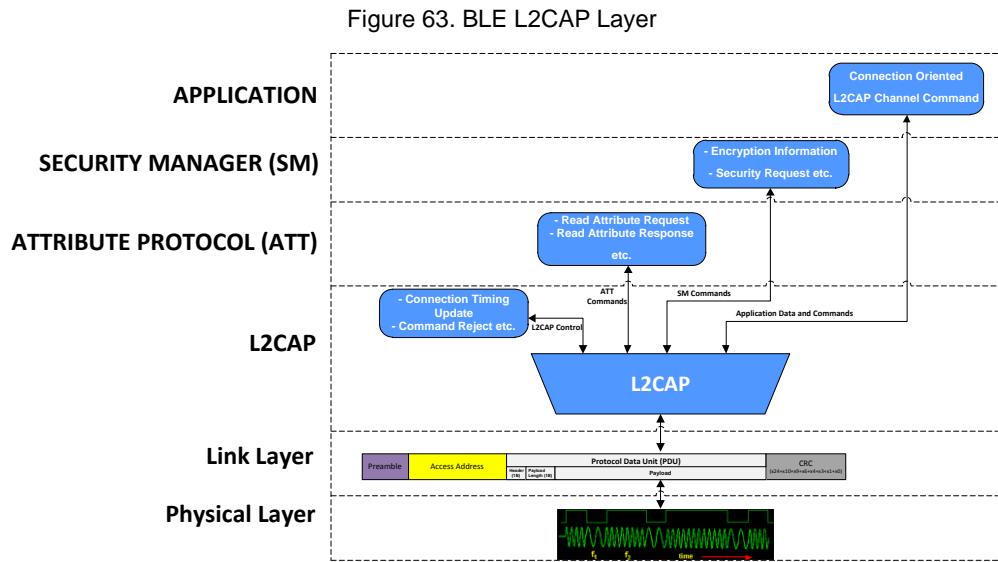
The HCI is the standard-defined interface between the host and the controller. It allows the host and the controller to exchange information such as commands, data, and events over different physical transports such as USB or UART. The HCI requires a physical transport only when the controller and the host are different devices.

In PSoC/PRoC BLE devices, the HCI is just a firmware protocol layer that passes the messages and events between the controller and the host.

## Logical Link Control and Adaptation Protocol (L2CAP)

L2CAP provides protocol multiplexing, segmentation, and reassembly services to upper-layer protocols. Segmentation breaks the packet received from the upper layer into smaller packets that the link layer can transmit, while reassembly combines the smaller packets received from the link layer into a meaningful packet. The L2CAP layer supports three protocol channel IDs for [Attribute Protocol \(ATT\)](#), [Security Manager \(SM\)](#), and L2CAP control, as shown in [Figure 63](#). Bluetooth 4.2 allows direct data channels through the L2CAP connection-oriented channels on top of these protocol channels.

The L2CAP and the layers above it are implemented in firmware in PSoC/PRoC BLE.



## Security Manager (SM)

The SM layer defines the methods used for pairing, encryption, and key distribution.

- **Pairing** is the process to enable security features. In this process, two devices are authenticated, the link is encrypted, and then the encryption keys are exchanged. This enables the secure exchange of data over the BLE interface without being snooped on by a silent listener on the RF channel.
  - **Bonding** is the process in which the keys and the identity information exchanged during the pairing process are saved. After devices are bonded, they do not have to go through the pairing process again when reconnected.
- BLE uses 128-bit [AES](#) for data encryption.

## Attribute Protocol (ATT)

There are two GATT roles in BLE that you should know to understand the ATT and GATT layers:

- **GATT server:** A GATT server contains the data or information. It receives requests from a GATT client and responds with data. For example, a heart-rate monitor GATT server contains heart-rate information; a BLE HID keyboard GATT server contains user key press information.
- **GATT client:** A GATT client requests and/or receives data from a GATT server. For example, a smartphone is a GATT client that receives heart-rate information from the heart-rate GATT server; a laptop is a GATT client that receives key-press information from a BLE keyboard.

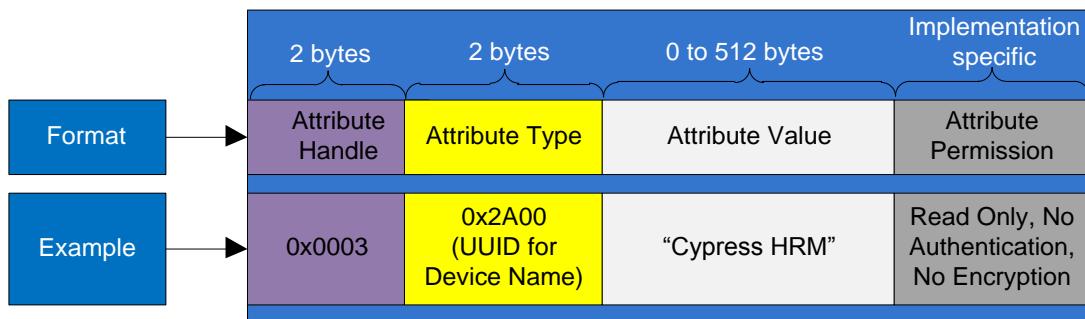
ATT forms the basis of BLE communication. This protocol enables the GATT client to find and access data or Attributes on the GATT server. For more details about the GATT client and server architecture, refer to [Generic Attribute Profile \(GATT\)](#).

An Attribute is the fundamental data container in the ATT/GATT layer, which consists of the following:

- **Attribute Handle:** The 16-bit address used to address and access an Attribute.
- **Attribute Type:** This specifies the type of data stored in an Attribute. It is represented by a 16-bit UUID defined by the Bluetooth SIG.  
For example, the 16-bit UUID of the Heart-Rate Service is 0x180D; the UUID for the Device Name Attribute is 0x2A00. Visit the Bluetooth [web page](#) for a list of 16-bit UUIDs assigned by the SIG.
- **Attribute Value:** This is the actual data stored in the Attribute.

- **Attribute Permission:** This specifies the Attribute access, authentication, and authorization requirements. Attribute permission is set by the higher layer specification and is not discoverable through the Attribute protocol. Figure 64 shows the structure of a Device Name Attribute as an example.

Figure 64. Attribute Format Example



### Attribute Hierarchy

Attributes are the building blocks for representing data in ATT/GATT. Attributes can be broadly classified into the following two groups to provide hierarchy and abstraction of data:

- **Characteristic:** A collection of Attributes that exposes the system information or meaningful data. A Characteristic consists of the following Attributes:

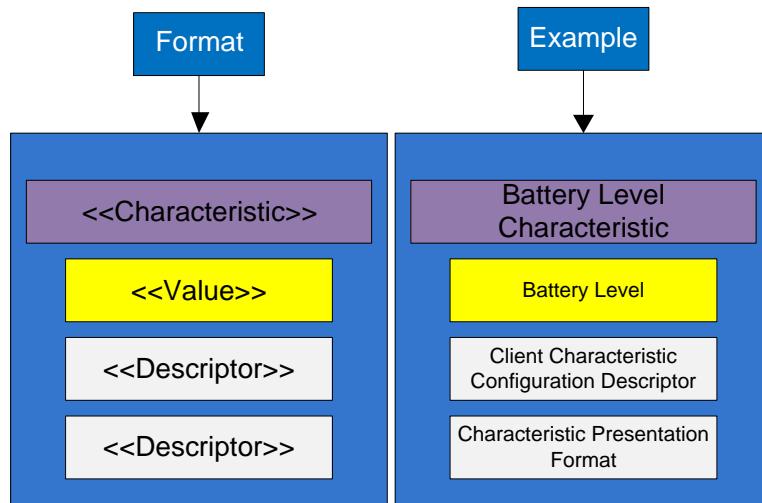
- Characteristic Declaration Attribute: This defines the beginning of a Characteristic.
- Characteristic Value Attribute: This holds the actual data.
- Characteristic Descriptor Attributes: These are optional Attributes, which provide additional information about the Characteristic value.

"Battery Level" is an example of a Characteristic in the Battery Service (BAS). Representing the battery level in percentage values is an example of a Characteristic descriptor.

Figure 65 shows the structure of a Characteristic with Battery Level as an example.

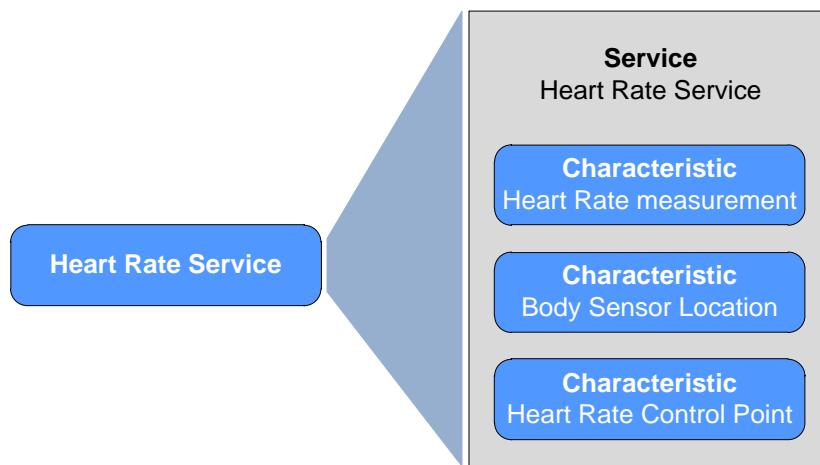
- The first part of a Characteristic is the declaration of the Characteristic (it marks the beginning of a Characteristic) indicated by the Battery Level Characteristic in Figure 65.
- Next is the actual Characteristic value or the real data, which in the case of the Battery Level Characteristic is the current battery level. The battery level is expressed as a percentage of full scale, for example "65," "90," and so on.
- Characteristic descriptors provide additional information that is required to make sense of the Characteristic value. For example, the Characteristic Presentation Format Descriptor for Battery Level indicates that the battery level is expressed as a percentage. Therefore, when "90" is read, the GATT client knows this is 90 percent and not 90 mV or 90 mAh. Similarly, the Valid Range Characteristic descriptor (not shown in Figure 65) indicates that the battery level range is between 0 and 100 percent.
- A Client Characteristic Configuration Descriptor (CCCD) is another commonly used Characteristic descriptor that allows a GATT client to configure the behavior of a Characteristic on the GATT server. When the GATT client writes a value of 0x01 to the CCCD of a Characteristic, it enables asynchronous notifications (described in the next section) to be sent from the GATT server. In the case of a Battery Level Characteristic, writing 0x01 to the Battery Level CCCD enables the Battery Service to notify its battery level periodically or on any change in battery-level value.

Figure 65. Characteristic Format and Example



- **Service:** The type of Attribute that defines a function performed by the GATT server. A Service is a collection of Characteristics and can include other Services. The concept of a Service is used to establish the grouping of relative data and provide a data hierarchy. See [Figure 66](#) for an example of a Heart Rate Service ([HRS](#)).  
 A Service can be of two types: A primary Service or a secondary Service. A primary Service exposes the main functionality of the device, while the secondary Service provides additional functionality. For example, in a heart-rate monitoring device, the HRS is a primary Service and BAS is a secondary Service.  
 A Service can also include other Services that are present on the GATT server. The entire included Services become part of the new Service.

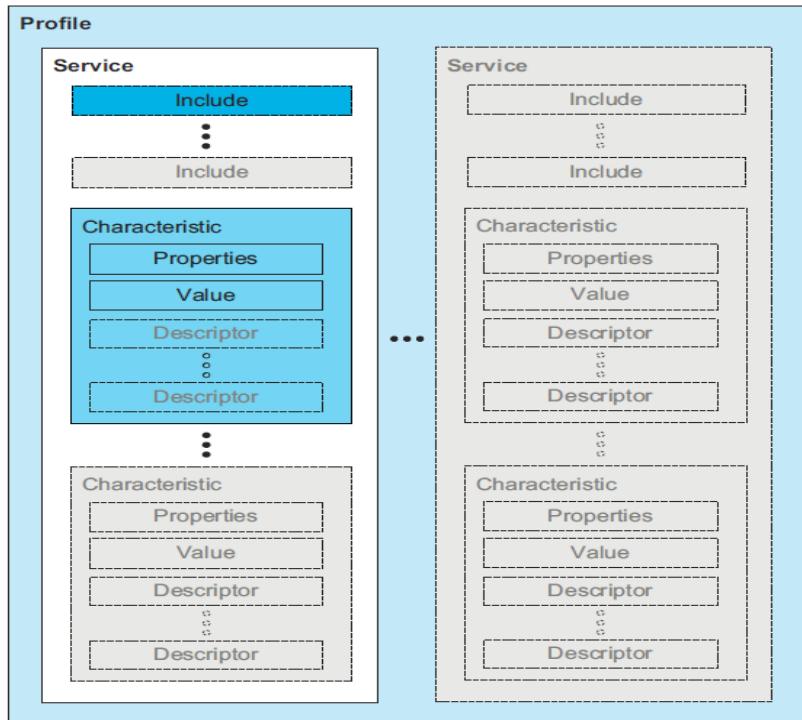
Figure 66. BLE Heart Rate Service Example



The word “Profile” in BLE is a collection of Services and their behavior that together perform a particular end application. A Heart Rate Profile ([HRP](#)) is an example of a BLE Profile that defines all the required Services for creating a heart-rate monitoring device. See the [Generic Access Profile \(GAP\)](#) section for details.

[Figure 67](#) shows the data hierarchy using Attributes, Characteristics, Services, and Profiles defined previously in this section.

Figure 67. BLE Data Hierarchy\*



\* Image courtesy of Bluetooth SIG

### Attribute Operations

Attributes defined in the previous section are accessed using the following five basic methods:

- **Read Request:** The GATT client sends this request to the GATT server to read an Attribute value. For every request, the GATT server sends a response to the GATT client. A smartphone reading the Battery-Level Characteristic of a heart-rate monitor device (see Figure 65) is an example of a Read Request.
- **Write Request:** The GATT client sends this request to the GATT server to write an Attribute value. The GATT server responds to the GATT client, indicating whether the value was written. A smartphone writing a value of 0x01 to the CCCD of a Battery Level Characteristic to enable notifications is an example of a Write Request.
- **Write Command:** The GATT client sends this command to the GATT server to write an Attribute value. The GATT server does not send any response to this command. For example, the BLE Immediate Alert Service (IAS) uses a Write Command to trigger an alert (turn on an LED, ring a buzzer, drive a vibration motor, and so on) on an IAS Target device (for example, a BLE key fob) from an IAS locator (for example, a smartphone).
- **Notification:** The GATT server sends this to the GATT client to notify it of a new value for an Attribute. The GATT client does not send any confirmation for a notification. For example, a heart-rate monitor device sends heart-rate measurement notifications to a smartphone when its CCCD is written with a value of 0x01.
- **Indication:** The GATT server sends this type of message. The GATT client always confirms it. For example, a BLE Health Thermometer Service (HTS) uses indications to reliably send the measured temperature value to a health thermometer collector, such as a smartphone.

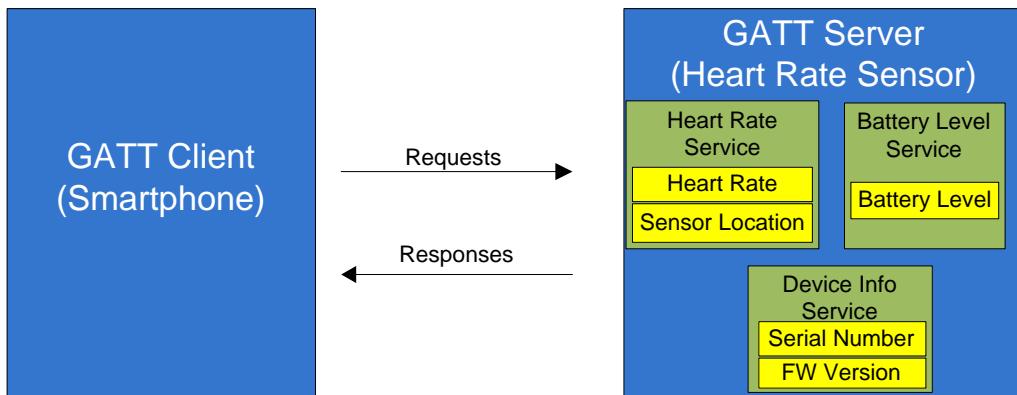
### Generic Attribute Profile (GATT)

The GATT defines the ways in which Attributes can be found and used. The GATT operates in one of two roles:

- **GATT client:** The device that requests the data (for example, a smartphone).
- **GATT server:** The device that provides the data (for example, a heart-rate monitor)

Figure 68 shows the client-server architecture in the GATT layer using a heart-rate monitoring device as an example. The heart-rate monitoring device exposes multiple Services (HRS, BAS, and Device Information Service); each Service consists of one or more Characteristics with a Characteristic value and descriptor, as shown in Figure 65.

Figure 68. GATT Client-Server Architecture



After the BLE connection is established at the link-layer level, the GATT client (which initially knows nothing about the connected BLE device) initiates a process called “service discovery.” As part of the service discovery, the GATT client sends multiple requests to the GATT server to get a list of all the available Services, Characteristics, and Attributes in the GATT server. When service discovery is complete, the GATT client has the required information to modify or read the information exposed by the GATT server using the Attribute operations described in the previous section.

## Generic Access Profile (GAP)

The GAP layer provides device-specific information such as the device address; device name; and the methods of discovery, connection, and bonding. The Profile defines how a device can be discovered, connected, the list of Services available, and how the Services can be used. Figure 70 shows an example of a Heart Rate Profile.

The GAP layer operates in one of four roles:

- **Peripheral:** This is an advertising role that enables the device to connect with a GAP Central. After a connection is established with the Central, the device operates as a slave. For example, a heart-rate sensor reporting the measured heart rate to a remote device operates as a GAP Peripheral.
- **Central:** This is the GAP role that scans for advertisements and initiates connections with Peripherals. This GAP role operates as the master after establishing connections with Peripherals. For example, a smartphone retrieving heart-rate measurement data from a Peripheral (heart-rate sensor) operates as a GAP Central.
- **Broadcaster:** This is an advertising role that is used to broadcast data. It cannot form BLE connections and engage in data exchange (no request/response operations). This role works similar to a radio station in that it sends data continuously whether or not anyone is listening; it is a one-way data communication. A typical example of a GAP Broadcaster is a beacon, which continuously broadcasts information but does not expect any response.
- **Observer:** This is a listening role that scans for advertisements but does not connect to the advertising device. It is the opposite of the Broadcaster role. It works similar to a radio receiver that can continuously listen for information but cannot communicate with the information source. A typical example of a GAP Observer is a smartphone app that continuously listens for beacons.

Figure 69 shows a generic BLE system with Cypress’s BLE Pioneer Kit as the Peripheral and a smartphone as the Central device. The interaction between BLE protocol layers and their roles on the Central and the Peripheral devices are also shown.

Figure 69. BLE System Design

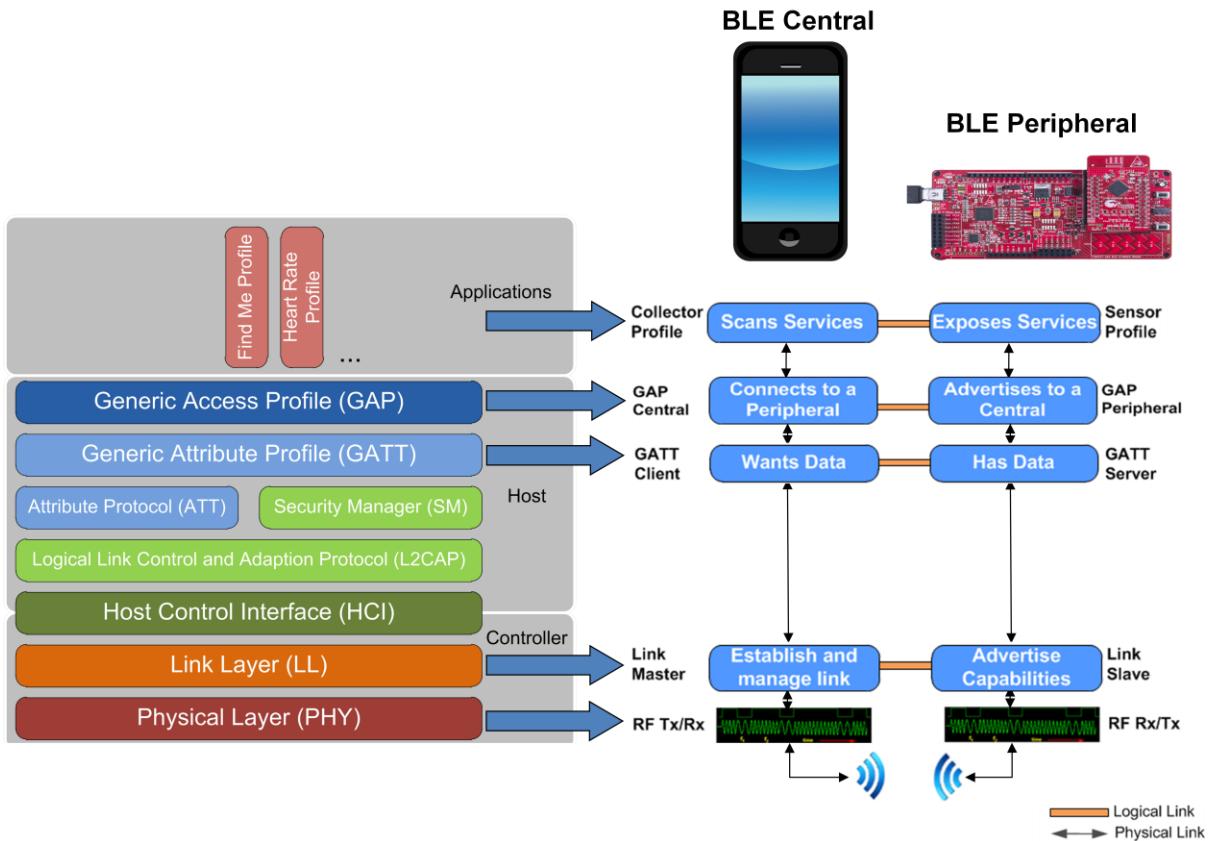
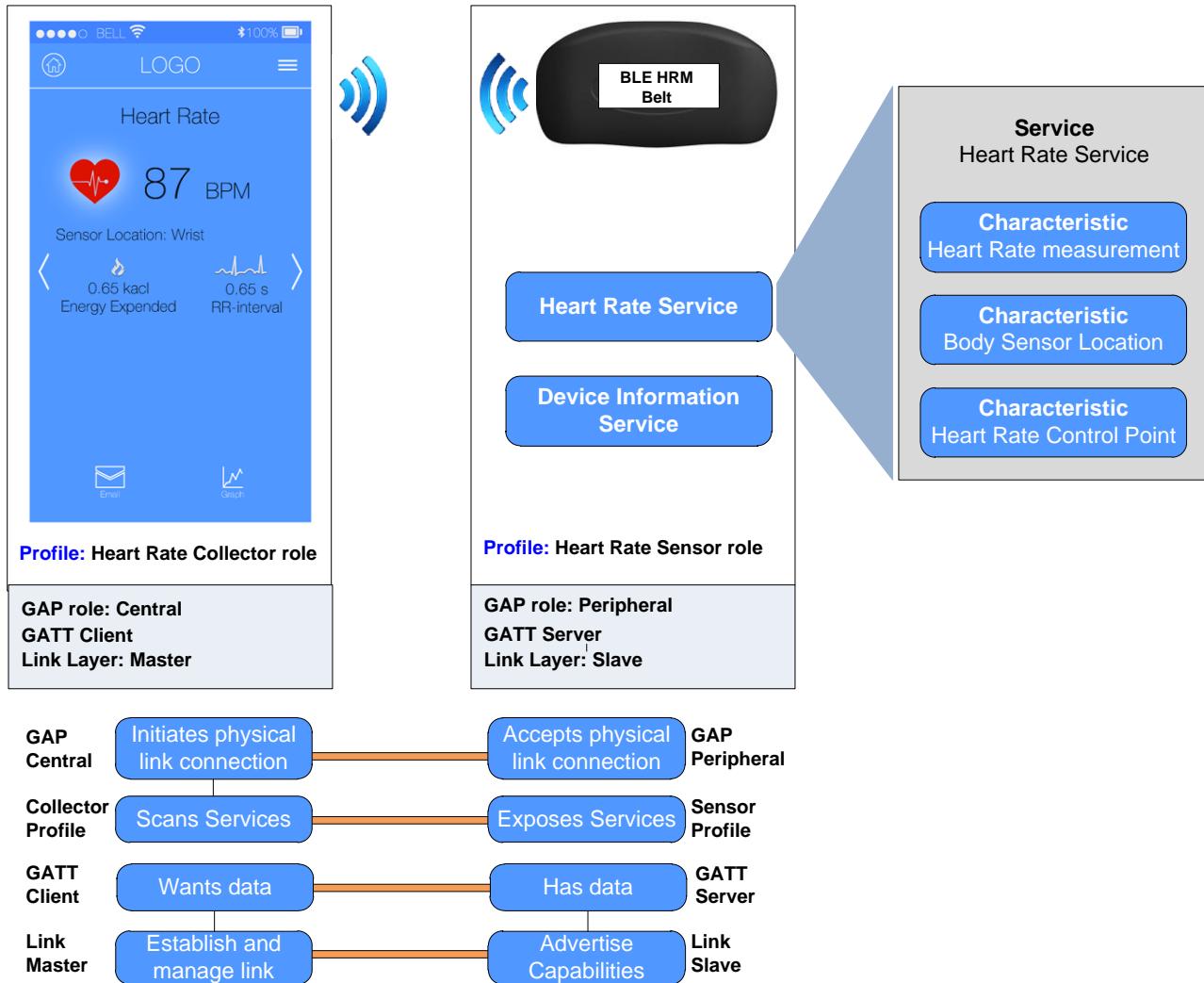


Figure 70 shows an example where a smartphone with a heart-rate app operates as a Central and a heart-rate sensor operates as a Peripheral. The heart-rate monitoring device implements the Heart-Rate Sensor Profile, while the smartphone receiving the data implements the Heart-Rate Collector Profile.

In this example, the Heart-Rate Sensor Profile implements two standard Services. The first is a Heart Rate Service that comprises three Characteristics (the Heart Rate Measurement Characteristic, the Body Sensor Location Characteristic, and the Heart Rate Control Point Characteristic). The second Service is a Device Information Service. At the link layer, the heart-rate measurement device is the slave and the smartphone is the master. See the Bluetooth developer portal for a detailed description of the Heart Rate [Service](#) and [Profile](#).

Figure 70. BLE Heart-Rate Monitor System



## Document History

Document Title: AN91267 – Getting Started with PSoC® 4 BLE

Document Number: 001-91267

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	4564230	KRIS	11/07/2014	New Application Note
*A	4567888	KRIS	11/12/2014	Fixed hyperlinks and document formatting
*B	4683692	KRIS	03/27/2015	Major reformatting including moving the BLE Protocol and PSoC 4 BLE device details to Appendix. Example design simplified and added Deep-Sleep low-power mode.

## 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	<a href="http://cypress.com/go/automotive">cypress.com/go/automotive</a>
Clocks & Buffers	<a href="http://cypress.com/go/clocks">cypress.com/go/clocks</a>
Interface	<a href="http://cypress.com/go/interface">cypress.com/go/interface</a>
Lighting & Power Control	<a href="http://cypress.com/go/powerpsoc">cypress.com/go/powerpsoc</a>
Memory	<a href="http://cypress.com/go/memory">cypress.com/go/memory</a>
PSoC	<a href="http://cypress.com/go/psoc">cypress.com/go/psoc</a>
Touch Sensing	<a href="http://cypress.com/go/touch">cypress.com/go/touch</a>
USB Controllers	<a href="http://cypress.com/go/usb">cypress.com/go/usb</a>
Wireless/RF	<a href="http://cypress.com/go/wireless">cypress.com/go/wireless</a>

### PSoC® Solutions

[psoc.cypress.com/solutions](http://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](http://cypress.com/go/support)

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



© Cypress Semiconductor Corporation, 2014-2015. 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.