Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

THIS SPEC IS OBSOLETE

Spec No: 001-94020

Spec Title: AN94020 - GETTING STARTED WITH PROC(TM) BLE

Replaced by: NONE

AN94020

Getting Started with PRoC™ BLE

Authors: Sai Prashanth Chinnapalli, Rahul Garg, Santhosh Kumar Vojjala

Associated Project: Yes

Associated Part Family: CYBL10X6X, CYBL1XX7X

Software Version: PSoC Creator™ 3.3 SP1 or later

Related Application Notes: click here.

To get the latest version of this application note, please visit www.cypress.com/go/AN94020

AN94020 introduces you to PRoC BLE, an ARM® Cortex™-M0 based Programmable Radio-on-Chip (PRoC) with Bluetooth Low Energy (BLE). It explores the PRoC BLE solution and development tools and shows you how to create your first BLE project using PSoC Creator™, the development tool for PRoC BLE. The application note also guides you to additional resources available to accelerate in-depth learning of PRoC BLE.

Contents

1 Introduction

PRoC BLE is a 32-bit, 48-MHz ARM Cortex-M0 BLE chip with CapSense®, 12-bit Analog to Digital Converter (ADC), four Timer/Counter/PWMs (TCPWMs), four additional 8-/16-bit PWMs, 36 GPIOs, two serial communication blocks (SCBs), Liquid Crystal Display (LCD) driver, and Inter-IC Sound (I2S). It provides a complete, programmable, and flexible solution for Human Interface Devices (HID), remote controls, toys, beacons, and wireless chargers. In addition to these applications, PRoC BLE provides a simple, low-cost way to add BLE connectivity into any system.

PRoC BLE includes a royalty-free BLE protocol stack (controller and host) compatible with the Bluetooth 4.2specification and comes with a BLE profile suite that includes all Bluetooth Special Interest Group (BT SIG)-adopted profiles (with a roadmap for updating for future specification revisions and profiles).

PRoC BLE offers low-power modes including 1.3 µA in Deep-Sleep mode with the watch crystal oscillator (WCO) on; 150 nA in Hibernate mode while retaining the SRAM contents, programmable logic, and the ability to wake up from an interrupt; and 60 nA in Stop mode while maintaining the wakeup capability on a GPIO interrupt.

The capacitive touch-sensing feature in PRoC BLE, known as CapSense, supports a wide variety of sensor types such as buttons, sliders, track pads, and proximity sensors. It offers a high signal-to-noise ratio and waterproofing to avoid false touches.

In addition to PRoC BLE, Cypress offers PSoC 4 BLE, 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. See the AN91267 - Getting Started with PSoC® 4 BLE application note for details. Cypress also offers EZ-BLE™ PRoC and EZ-BLE™ PSoC® modules, a small-footprint, fully certified, production-ready BLE module which significantly reduces design cycle time and helps to get to market faster. See Appendix A: BLE Device Family Comparison for a comparison of the BLE device families from Cypress.

This application note presents the basics of BLE that includes capabilities of the PRoC BLE device and an overview of development tools. For advanced application development, see the ***Designing BLE Applications*** and ***Creating a BLE Custom Profile*** application notes.

1.1 Prerequisites

Before you get started, make sure you have the development kit and required software.

Hardware

- BLE Pioneer Kit
- PC with Windows 7 or later
- Mobile phone with Android 5 or later, or iOS 8 or later

Software

- PSoC Creator 3.3 SP1 or later with PSoC Programmer 3.23.3 or later
- CySmart Host Emulation Tool or CySmart iOS/Android app

2 PRoC BLE Resources

Cypress provides a wealth of data at www.cypress.com to help you to select the right PRoC/PSoC device quickly and effectively integrate it into your design. If you are a first-time user of Cypress's PRoC/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.

The following is an abbreviated list for PRoC BLE:

- **Overview:** PSoC Portfolio, PRoC Portfolio
- **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 CYBL10X6X and CYBL1XX7X 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 PRoC BLE device family.
- **CapSense Design Guide:** Learn how to design capacitive touch-sensing applications with the PRoC 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 tool that enables you to test and debug your BLE Peripheral applications.

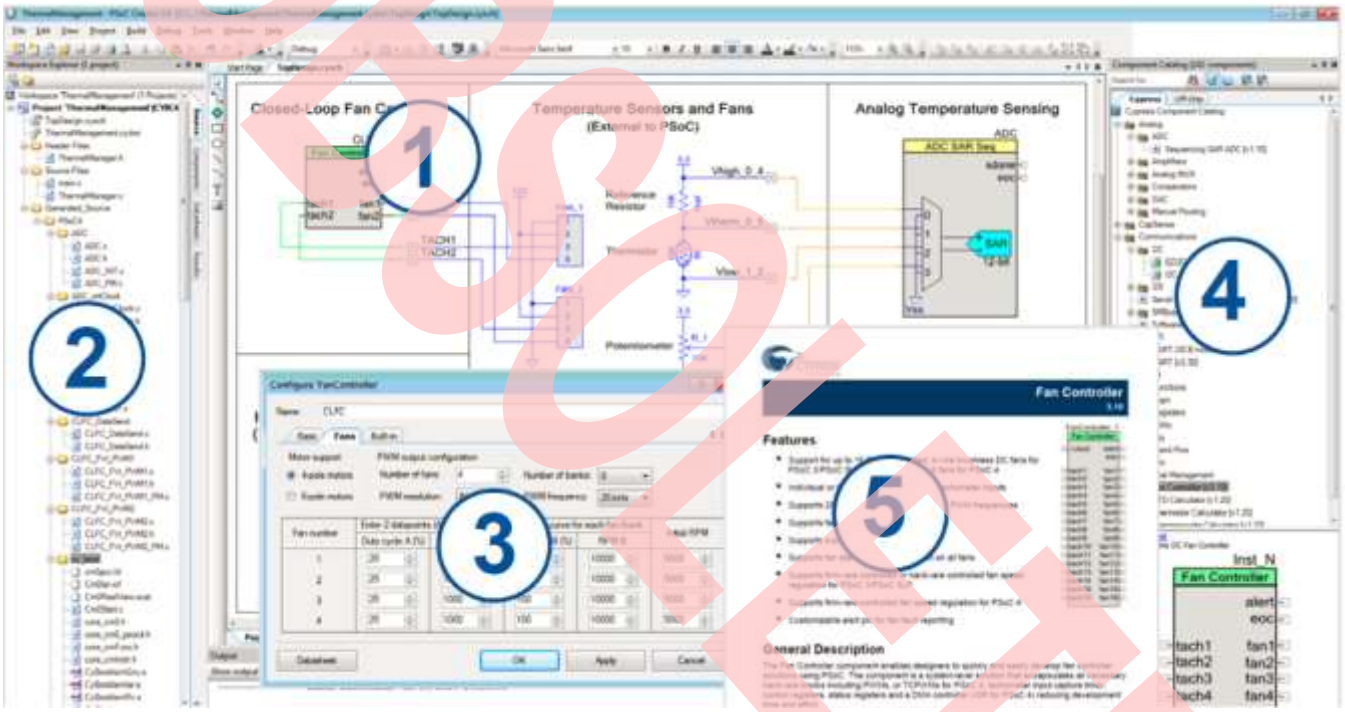
  See Appendix C: Cypress BLE Development Tools for an overview.
- **Training Videos:** Take a look at PSoC Creator and PSoC 4 BLE training video modules to get started with your first PSoC Creator-based PRoC BLE design.
- **Technical Support**
  - Frequently Asked Questions (FAQs): Learn more about our BLE ecosystem
  - 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.

2.1 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 PRoC BLE and PSoC 4 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 PRoC 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



2.2 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:

1. **Quick Start Guide**: Choose **Help** > **Documentation** > **Quick Start Guide**. This guide gives you the basics for developing PSoC Creator projects.
2. **Simple Component Code Examples**: Choose **File** > **Code Example**. These code examples demonstrate how to configure and use PSoC Creator Components.
3. **System Reference Guide**: Choose **Help** > **System Reference Guides**. This guide lists and describes the system functions provided by PSoC Creator.
4. **Component Datasheets**: Right-click a Component and select “Open Datasheet.”
5. **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**.

2.3 Code Examples

PSoC Creator includes a large number of code examples. These projects are available from the PSoC Creator Start Page, as Figure 2 shows.

Code examples can speed up your design process by starting you off with a complete design, instead of a blank page. The code examples 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 Code Example** 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 30 BLE code examples for your reference, 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.

Apart from PSoC Creator code examples, you can also find more BLE reference examples on this GitHub repository.

Figure 2. Code Examples in PSoC Creator

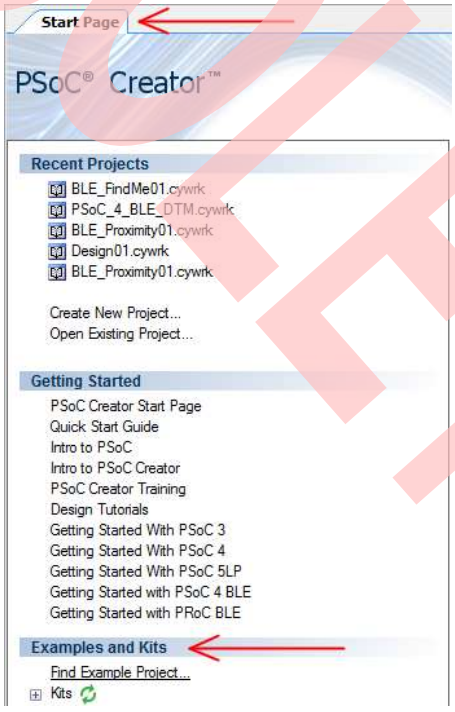
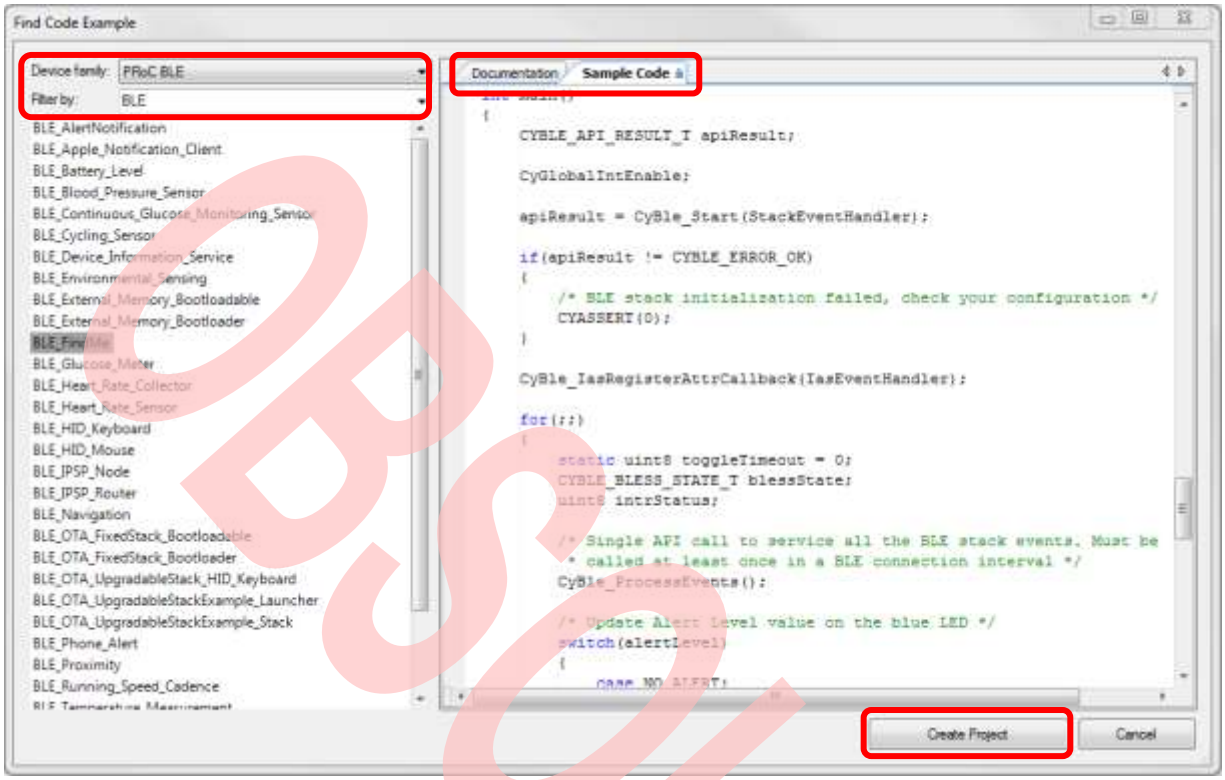


Figure 3. Code Example Projects with Sample Code



3 PRoC BLE Features

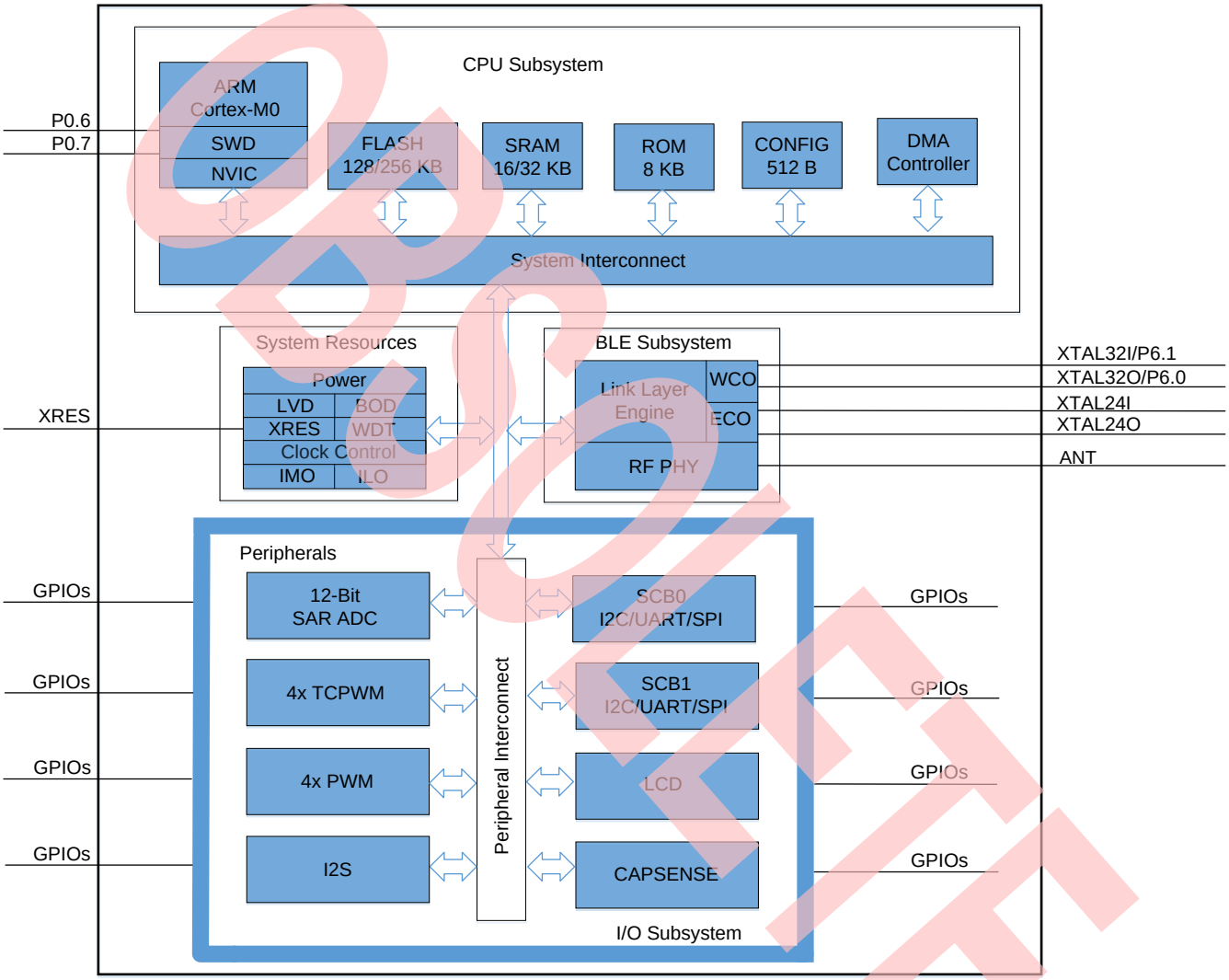
PRoC BLE is a feature-rich device family, which includes a CPU and memory subsystem, a BLE subsystem, a digital subsystem, and system resources. Figure 4 shows features available in the CYBL10X6X and CYBL1XX7X device families. The CYBL10X6X device family has 128 KB flash and CYBL1XX7X device family has 256 KB flash. PRoC BLE device features are listed below:

- 32-bit MCU subsystem
  - 48-MHz ARM® Cortex™-M0 CPU
  - Up to 256 KB flash and 32 KB SRAM
  - 8-channel direct memory access (DMA) controller
- CapSense® Touch Sensing with Two-Finger Gestures
  - One Cypress Capacitive Sigma-Delta™ (CSD) controller, which supports buttons, sliders, and touchpads with two-finger gesture detection capability
- Configurable Analog Peripherals
  - One 12-bit, 1-Msps SAR ADC
- Programmable Digital Peripherals
  - Four dedicated TCPWM blocks: 16-bit timer, counter, or PWM
    - Additional four 8-bit or two 16-bit PWMs
  - Two configurable serial communication blocks (SCBs): I2C master or slave, SPI master or slave, or UART
  - Inter-IC Sound(I2S) master interface
- Bluetooth Smart connectivity with Bluetooth 4.2

- 2.4-GHz BLE radio with integrated Balun
- Bluetooth 4.2 specification compliant controller and host implementation

See Appendix D: PRoC BLE Device for a brief description of the features. For in-depth information, see the PRoC BLE family datasheet, TRM, and application notes.

Figure 4. PRoC BLE Architecture



4 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 amount of data with low power consumption, making it the preferred wireless protocol for battery-operated devices.

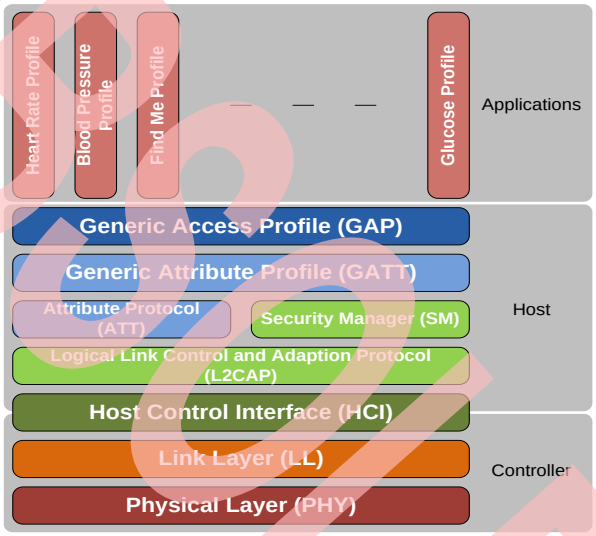
The BLE stack consists of the following:

- 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 that 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 Architecture



To develop a BLE application, you do not need a working knowledge of this complex protocol stack. Cypress provides an easy-to-configure, GUI-based BLE Component in PSoC Creator 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 BLE GAP and GATT layer configurations.

4.1 BLE Link Establishment

To establish a BLE link between two devices—the Cypress BLE Pioneer Kit and a smartphone, for example—you need to understand the two Generic Access Profile (GAP) device roles as Figure 6 shows:

- **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.

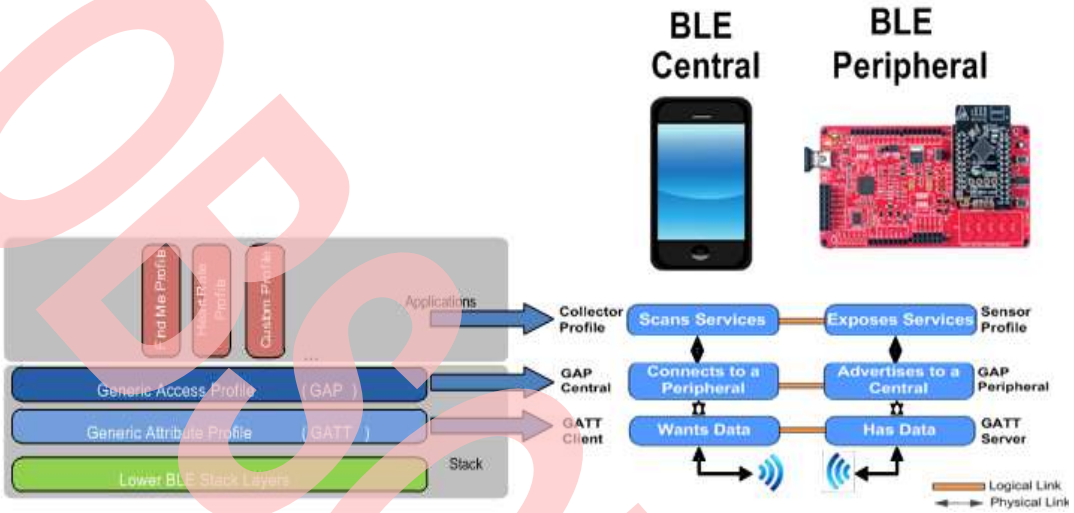
After the Central device establishes a connection with the Peripheral, both devices are said to be connected over a BLE link. On a connected BLE link, independent of the GAP role, the Generic Attribute Profile (GATT) defines two profile roles based on the source and destination of data:

- **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. BLE Application Overview



4.2 GATT Data Format

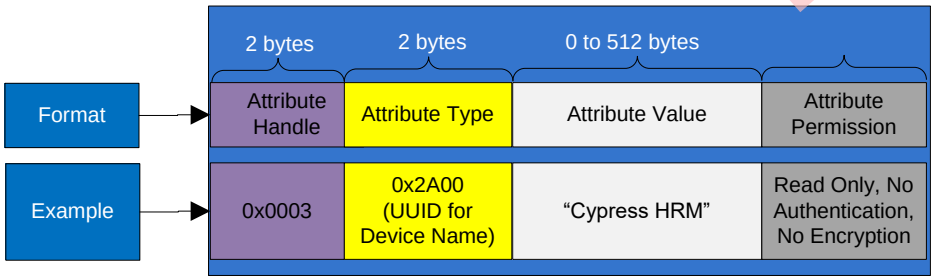
After understanding the BLE GAP/GATT roles, the next step is to understand how the data is stored on a GATT server and how is it retrieved by a GATT client. A GATT server uses Attributes, Characteristics, and Services to represent and abstract data in a BLE device. As you will see in this section, a Service contains one or more Characteristics and each Characteristic is composed of multiple Attributes that contain the actual data.

- **Attribute:** An Attribute is the fundamental data container of the 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 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. As a user, you only need to know the Attribute handle on which you want to perform a read/write operation; the rest of the details are abstracted by Cypress's BLE Component API functions.

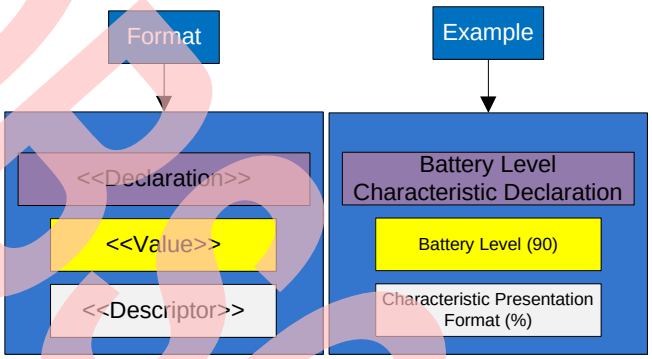
Figure 7. GATT Attribute Example



- **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. On/off state of a smart bulb is an example for a custom Characteristic.

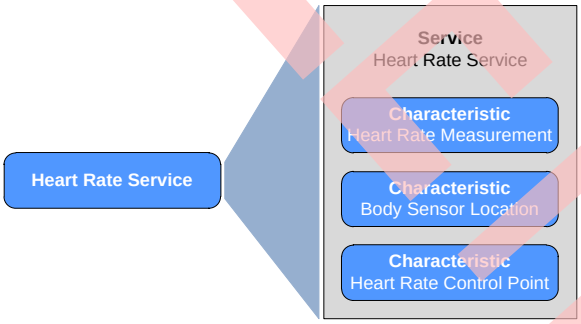
Figure 8. GATT Characteristic Example



- **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 the information related to 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. Smart bulb service is an example for a custom Service that contains the bulb on/off state Characteristic.

Figure 9. GATT Service Example

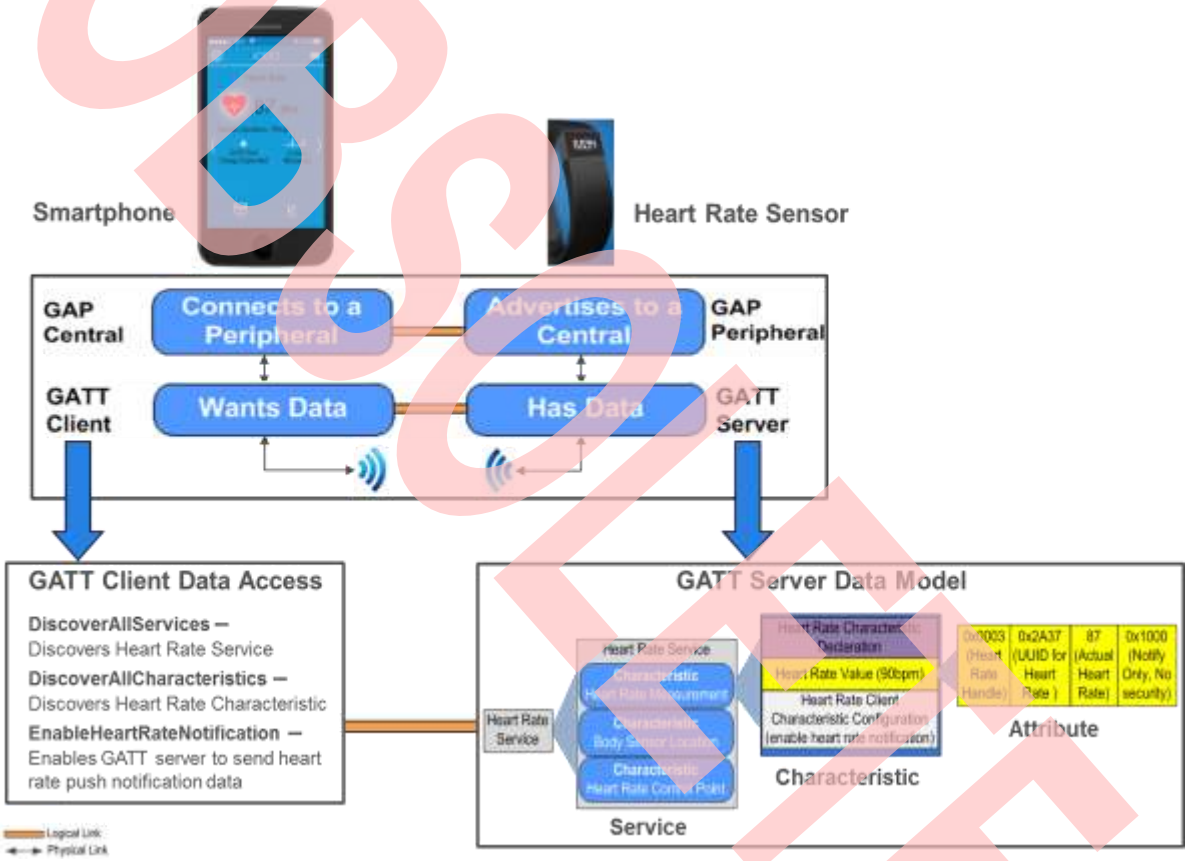


In addition to defining the data format, the GATT layer also defines a set of procedures or methods to discover and access data in the GATT layer. After establishing a BLE connection, a GATT client uses these GATT procedures to discover the complete attribute database of the connected GATT server. After the discovery is complete, the GATT client uses Characteristic read/write GATT procedures to perform read/write operations on the Attributes using the Attribute handle.

Figure 10 shows how the heart-rate data is modeled on a heart-rate sensor and the GATT procedures used by a smartphone to get the heart-rate data. A typical heart-rate sensor implements a GAP Peripheral role where it advertises and connects to a GAP Central device such as a smartphone. After establishing the connection, the heart-rate sensor device exposes a GATT server that encapsulates the heart-rate data. The GATT server supports a Heart Rate Service that contains the Heart Rate Measurement Characteristic, which stores the actual measured heart-rate value in the form of a Heart Rate Value Attribute.

On the smartphone side, after establishing the connection, a GATT client on the smartphone initially discovers all the Services, Characteristics, and Attributes supported by the connected GATT server using GATT discovery procedures. After completing the discovery, the smartphone GATT client uses GATT Characteristic procedures to enable push notifications for heart-rate data from the GATT server.

Figure 10. GATT Data Format



4.3 BLE Profile

A Profile in BLE 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 by the heart-rate monitoring device to create an interoperable heart-rate monitoring device. The Bluetooth SIG offers a set of predefined standard Profiles for commonly used BLE end applications. You can also 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:

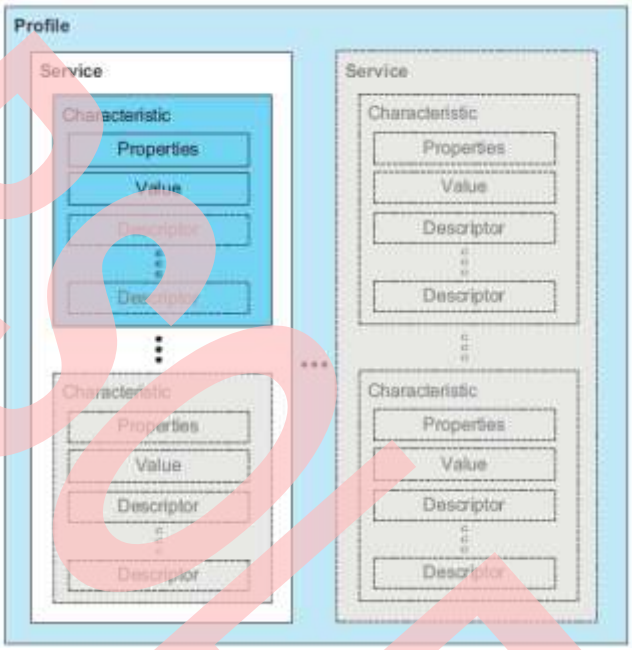
- **Sensor or server:** The Sensor Profile role is supported by the application that has data. The Sensor Profile specification defines the required roles (for example, GATT/GAP roles) and behavior (for example, advertisement interval or GATT Services to be supported) of the BLE device to support a Sensor application use case. Following the

Sensor Profile specification guarantees interoperability of the Sensor application with any other device that implements the corresponding Collector Profile. For example, a heart-rate monitoring device that implements the Heart Rate Sensor Profile will be interoperable with all smartphones that implement Heart Rate Collector Profile.

- **Collector or client:** The Collector Profile role is supported by the application that wants data. The Collector Profile specification defines the required BLE device roles and behavior to interoperate with and collect information from any device that implements the corresponding Sensor Profile specification.

A summary of the data abstraction and hierarchy in a BLE device is as shown in Figure 11.

Figure 11. BLE Data Hierarchy*



* Image courtesy of Bluetooth SIG

4.4 BLE Component

4.4.1 Features

The BLE Component in PSoC Creator abstracts the BLE protocol complexity into a simple and easy-to-use GUI and a few APIs. BLE Component features are listed below:

- Bluetooth 4.2compliant protocol stack
- Supports all GAP roles – Central, Peripheral, Broadcaster, and Observer. A limited simultaneous combination of GAP roles such as Central and Observer or Peripheral and Broadcaster is also supported.
- Supports all the Bluetooth SIG-adopted GATT-based Profiles and Services with at least one example code per supported Profile
- Custom Profile creation and usage made easy by the BLE Component GUI
- Supports L2CAP connection-oriented channels, Link Layer low-duty-cycle advertising and LE ping features
- Supports up to four bonded devices and eight device whitelist filter
- Supports all Bluetooth 4.2 security modes
- Supports link layer data length extension, link layer privacy, and LE secure connection features of Bluetooth 4.2 specification

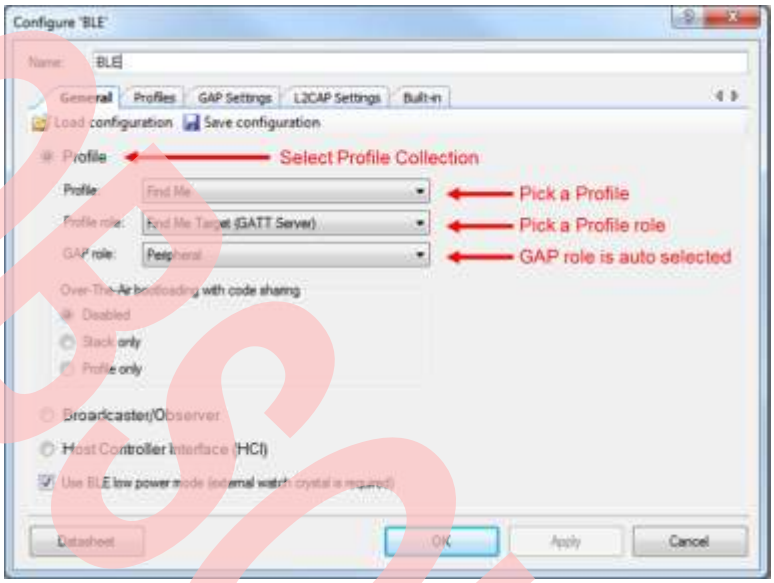
See the BLE Component datasheet for details.

4.4.2 Configuration

It takes five steps create an application based on the Bluetooth SIG-defined Profile (standard Profile) using the BLE Component.

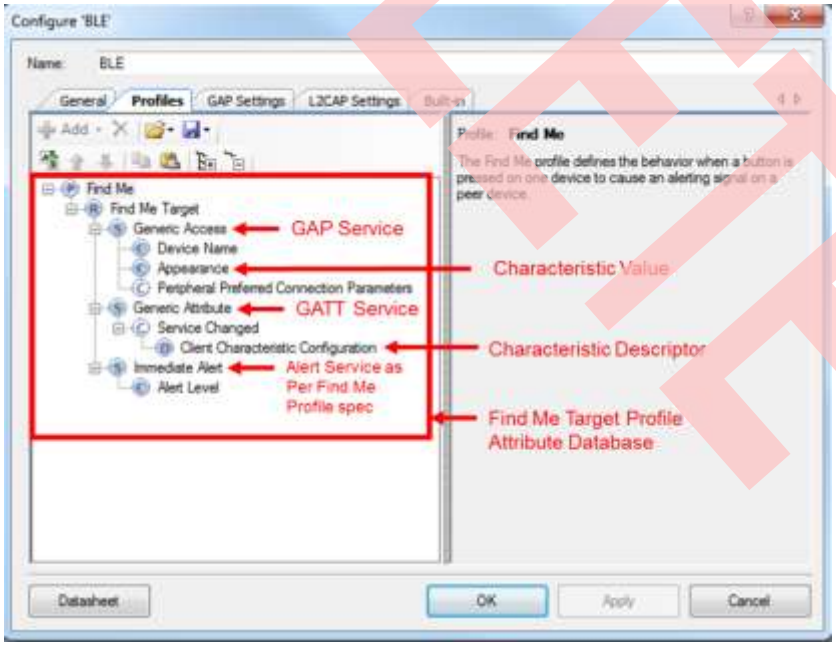
Step 1: Select a desired Profile and Profile role for your design. The BLE Component automatically selects the required GAP and GATT roles for the selected Profile role. For the Bluetooth SIG-defined Find Me Profile in the Target role shown in Figure 12, the GAP role is set to Peripheral and the GATT role is set to server per the Find Me Profile specification.

Figure 12. BLE Profile Configuration



Step 2: For the selected Profile role, all 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 based on your design. The settings in the **Profiles** tab form the Attribute database for the selected design.

Figure 13. BLE Attribute Database Configuration



Step 3: Configure the GAP general and advertisement data settings for your design as explained in section 4.1 (BLE Link Establishment) of this document, as shown in Figure 14 and Figure 15.

Figure 14. BLE GAP Configuration

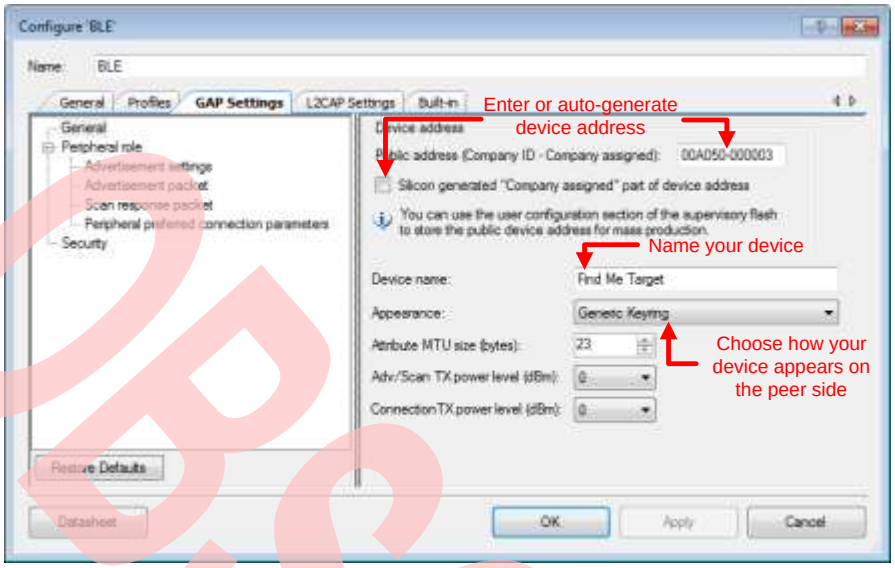
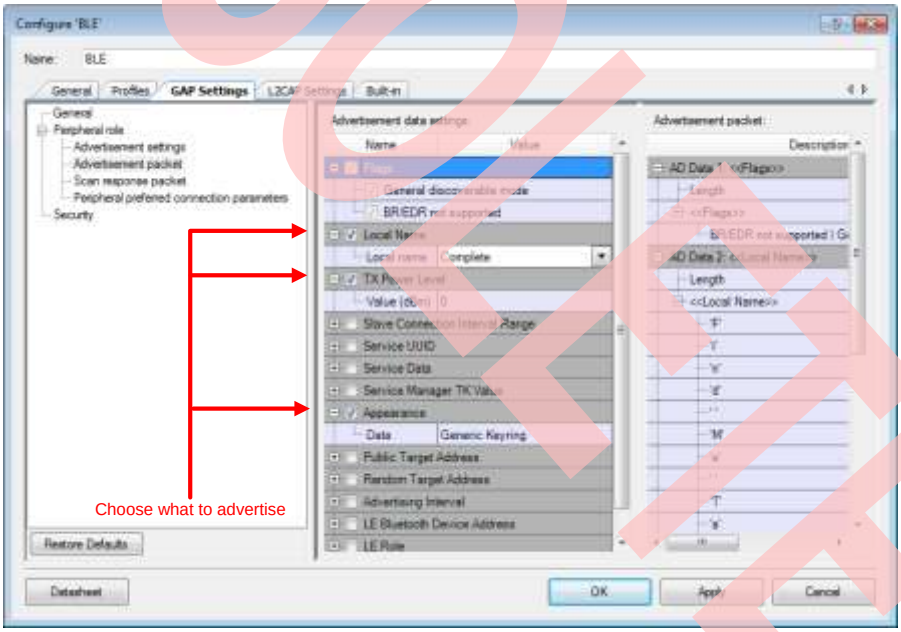


Figure 15. BLE Advertisement Data Configuration



Step 4: Write firmware to initialize the design you just configured. Register event handlers with the BLE Component to receive data and events. Event handler details are described later in this document.

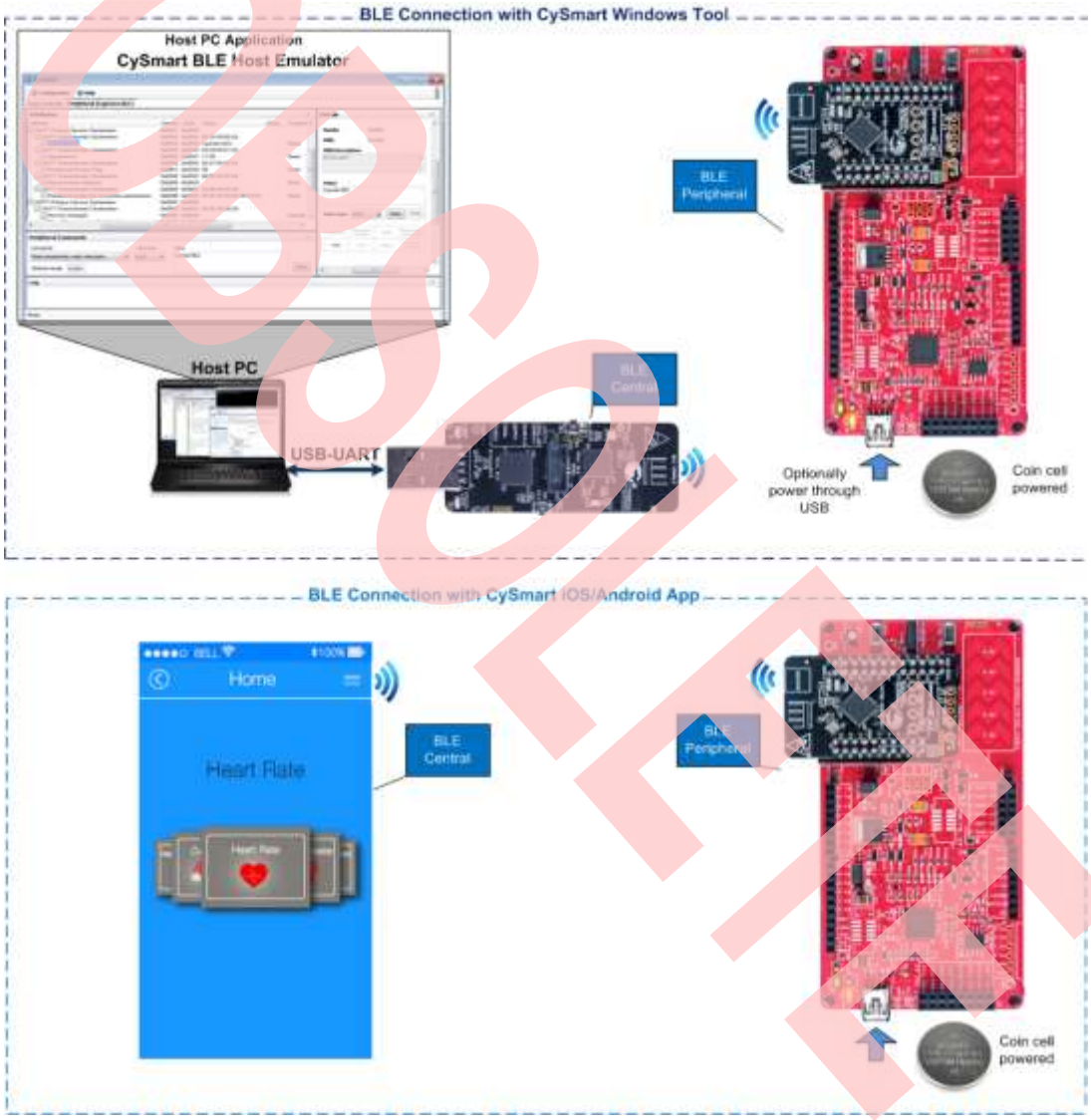
Step 5: Wait in the program main loop for an event from the BLE Component and take the necessary action or send data to the Central device using BLE Component API functions.

My First PRoC BLE Design section of this document will walk you through a step-by-step configuration of the BLE Component for creating a simple Peripheral application. See 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.

5 PRoC BLE Development Setup

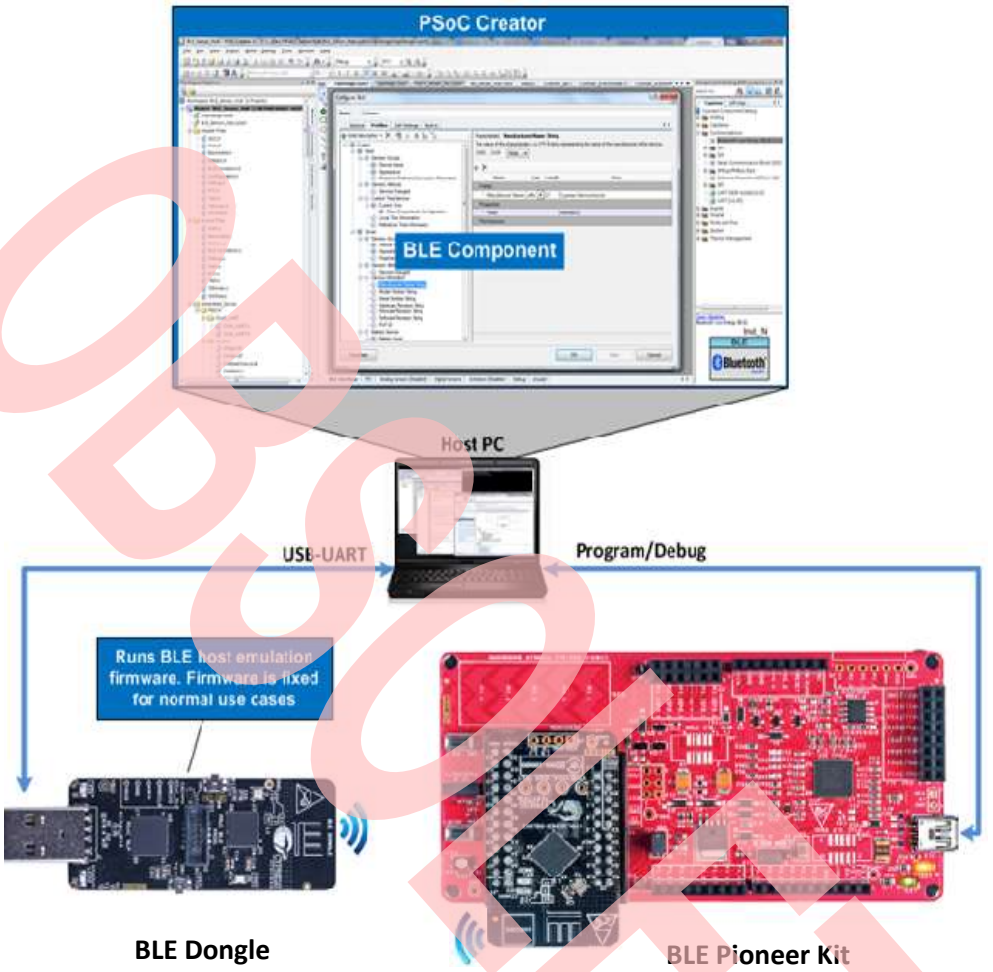
Figure 16 shows the hardware and software tools required for evaluating BLE Peripheral designs using the PRoC BLE device. In a typical use case, the BLE Pioneer Kit (red board with black module in Figure 16) is configured as a Peripheral that can communicate with a Central device such as CySmart iOS/Android app or CySmart Host Emulation Tool. The CySmart Host Emulation Tool also requires a BLE Dongle (black board in Figure 16) for its operation.

Figure 16. BLE Functional Setup



As shown in Figure 17, the BLE Dongle is preprogrammed to work with Windows CySmart Host Emulation Tool. The BLE Pioneer Kit has an on-board USB programmer that works with PSoC Creator for programming or debugging your BLE design. BLE Pioneer Kit can either be powered over the USB interface or by a coin-cell battery. Both the BLE Dongle and the BLE Pioneer Kit can be simultaneously connected to a common host PC for development and testing.

Figure 17. BLE Development Setup



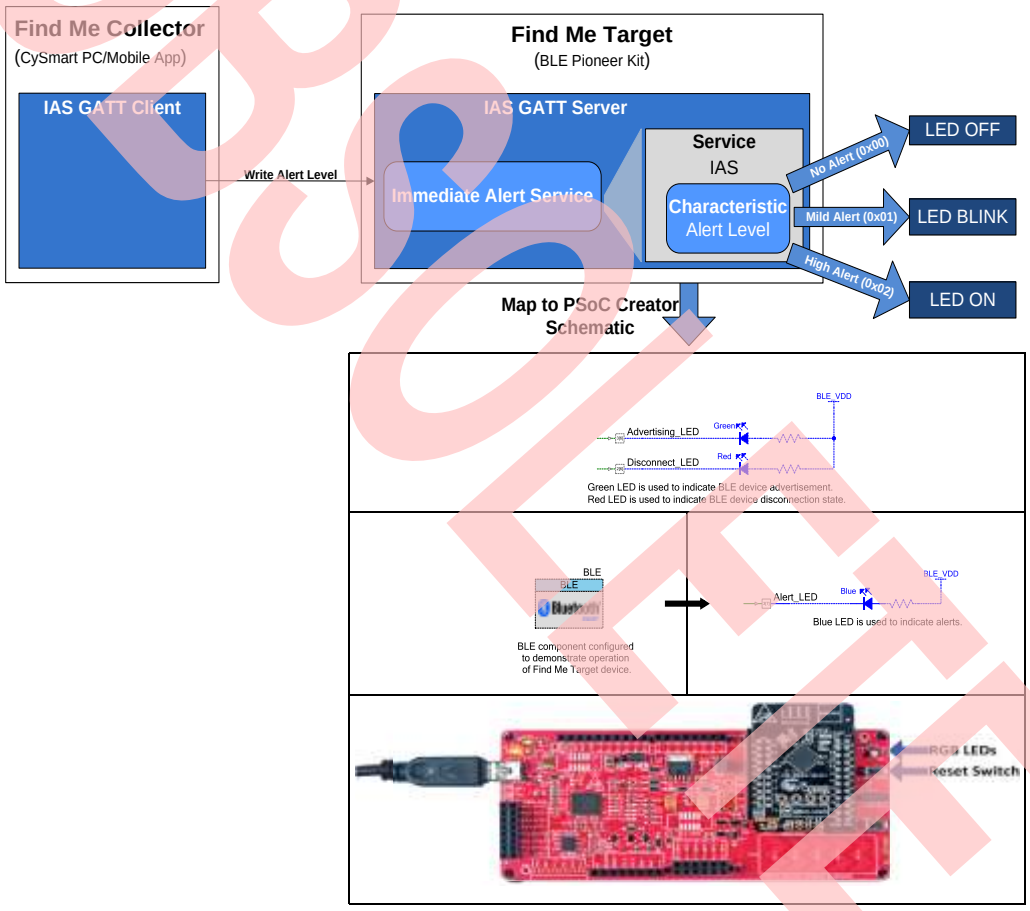
6 My First PRoC BLE Design

This section provides you with the step-by-step process for building a simple design with PRoC BLE using PSoC Creator on a BLE Pioneer Kit. Creating a simple Bluetooth SIG-defined standard Profile design is described in this section. For creating advanced standard or custom Profile designs, see *Designing BLE Applications* and *Creating a BLE Custom Profile* application notes.

6.1 About the Design

This design implements a BLE Find Me Profile (FMP) in the Target role that consists of an Immediate Alert Service (IAS). FMP and IAS are BLE standard Profile and Service respectively, defined by the Bluetooth SIG. Alert levels triggered by the Find Me Locator are indicated by varying the state of an LED on the BLE Pioneer Kit, as Figure 18 shows. Two status LEDs indicate the state of the BLE interface.

Figure 18. My First PRoC BLE Design



You can create your first PRoC BLE design in four steps:

Step 1. Create and configure the design in the PSoC Creator schematic page.

Step 2. Write the firmware to initialize and handle BLE events.

Step 3. Program the PRoC BLE device on the BLE Pioneer Kit.

Step 4. Test your design using the CySmart Host Emulation Tool or mobile app.

Note: The functional PSoC Creator project for the BLE example design described in this application note is distributed as part of PSoC code examples. You can choose to start with the code example, by selecting **File > Code Example**, and then selecting **Filter by** as **Find Me** > **BLE_FindMe**. If you use the code example, skip schematic configuration (step 1) and firmware development (step 2) mentioned above and go to programming (step 3) and testing the design (step 4).

6.2 Prerequisites

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

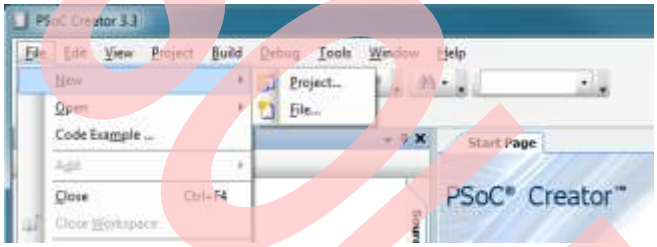
- PSoC Creator 3.3 SP1 or later with PSoC Programmer 3.23.3 or later
- CySmart Host Emulation Tool or CySmart iOS/Android app

6.3 Step 1: Create and 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 hardware and firmware design entry.

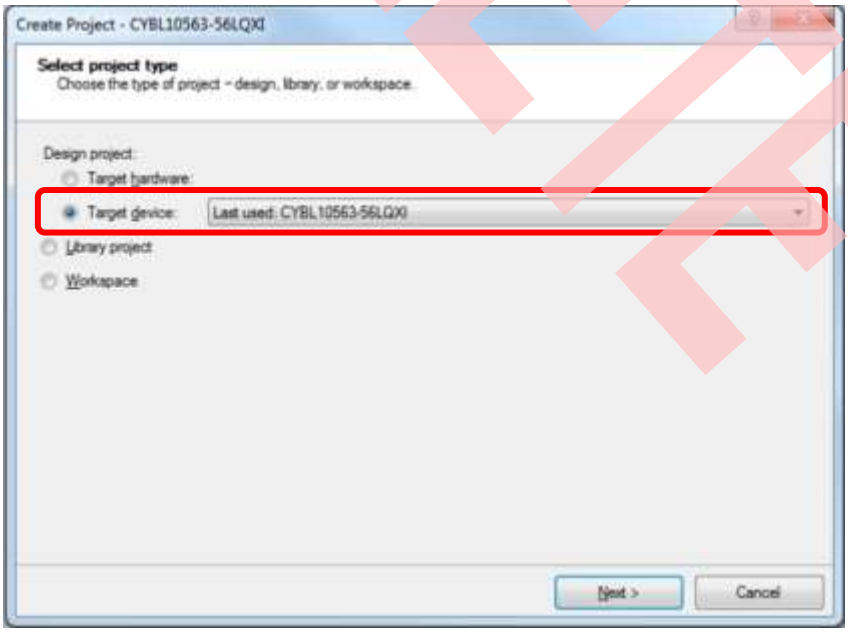
1. Install PSoC Creator 3.3 SP1 or later on your PC.
2. Start PSoC Creator, and choose **File** > **New** > **Project**, as Figure 19 shows.

Figure 19. Creating a New Project



3. Select the target device as “PRoC BLE” as shown in Figure 20 to select CYBL10563-56LQXI device used on BLE Pioneer kit and click **Next**. If you are using a custom PRoC BLE hardware or a different PRoC BLE part number, choose the “Launch Device Selector” option in Target device and select the appropriate part number.

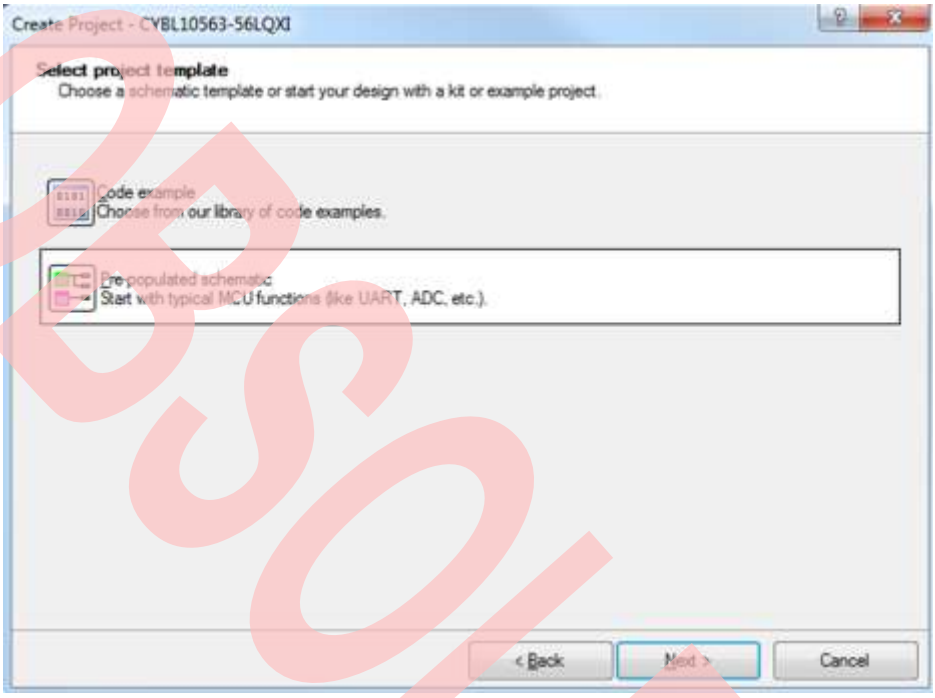
Figure 20. Selecting the Target Device



4. Select “Pre-populated schematic” project template as shown in Figure 21 and click **Next**.

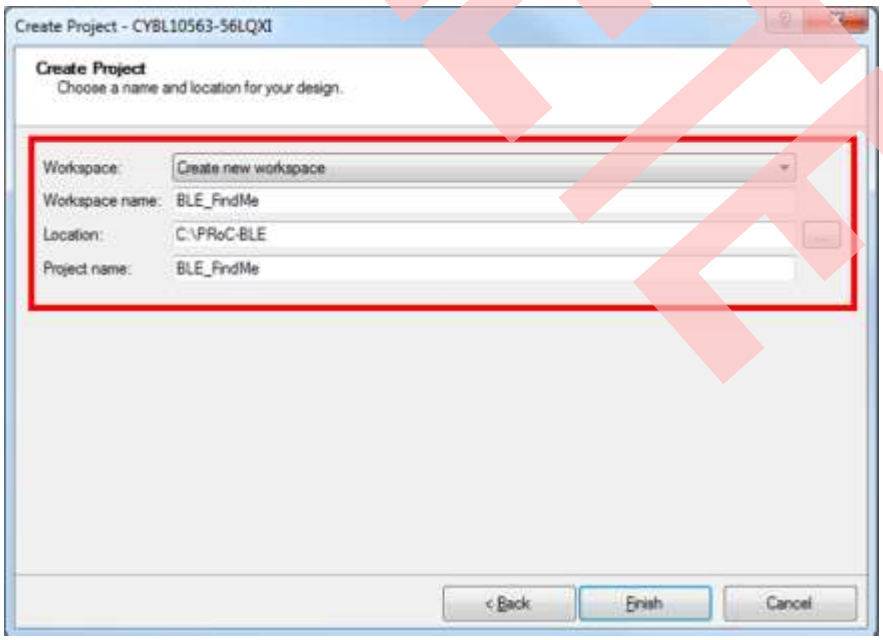
   Note: If you like to use the pre-built example for this design, select Code example in Figure 21 and click **Next**. In the code example selection window, select **Filter by** as **Find Me** > **BLE_FindMe** and skip to programming section directly.

Figure 21. Selecting the Project Template



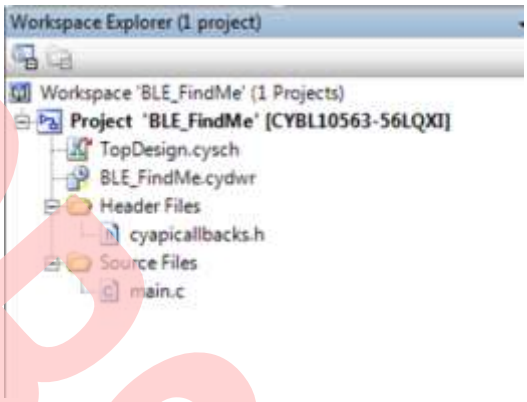
5. Provide the workspace and project name as shown in Figure 22. Choose an appropriate location for the new project, and then click **Finish**.

Figure 22. Create Project



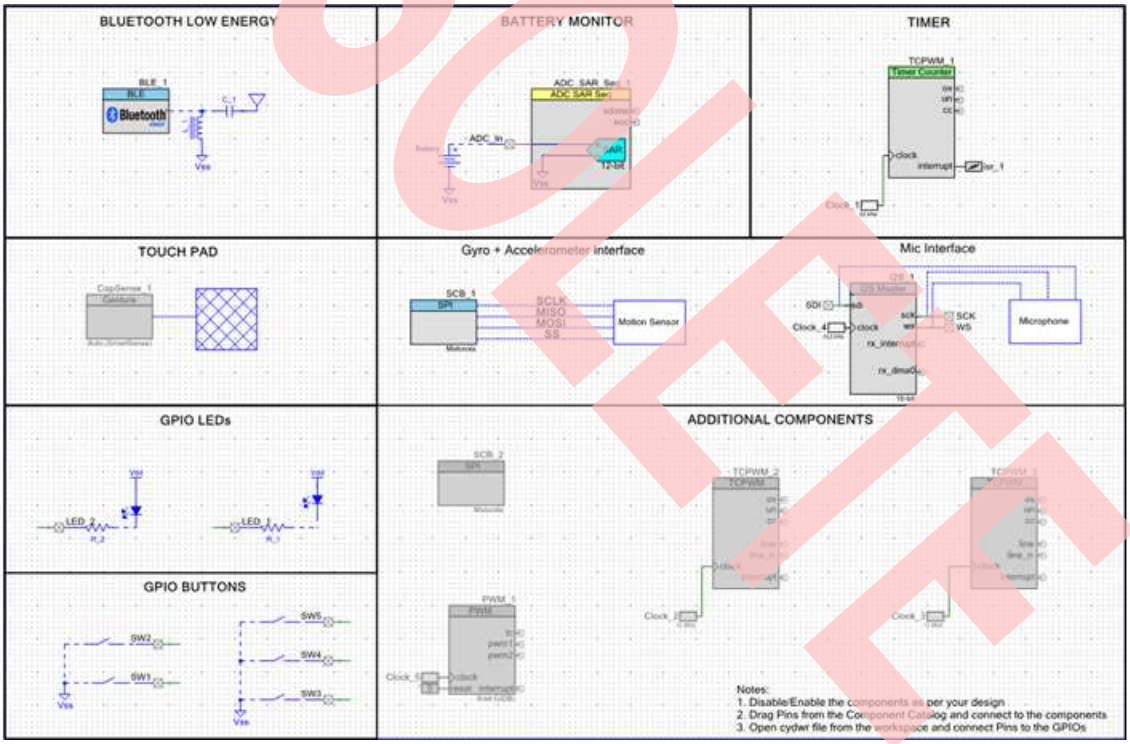
6. Creating a new project generates a project folder with a baseline set of files. You can view these files in the Workspace Explorer window, as Figure 23 shows.

Figure 23. Workspace Explorer



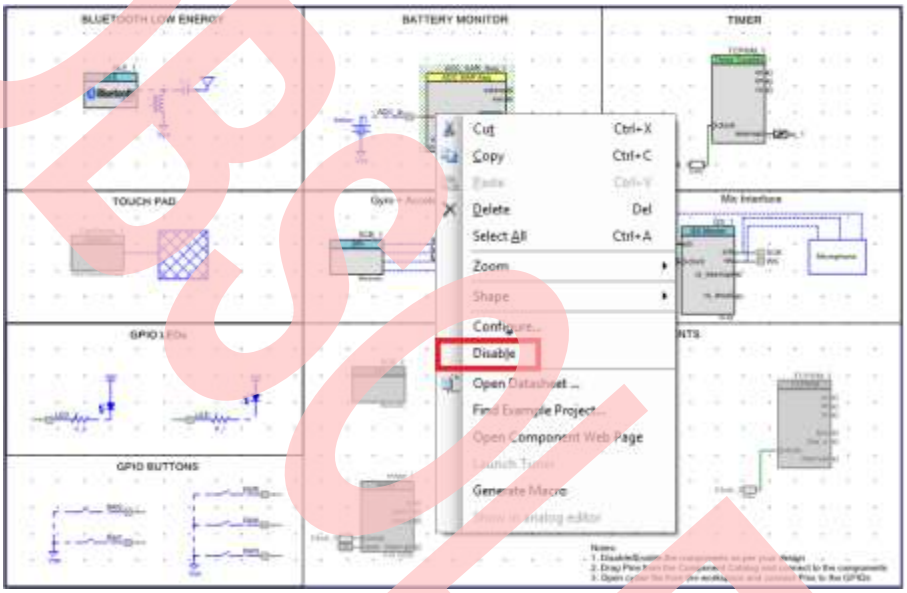
In the “TopDesign.cysch” a pre-populated schematic can be seen, as shown in Figure 24.

Figure 24. Pre-Populated Schematic



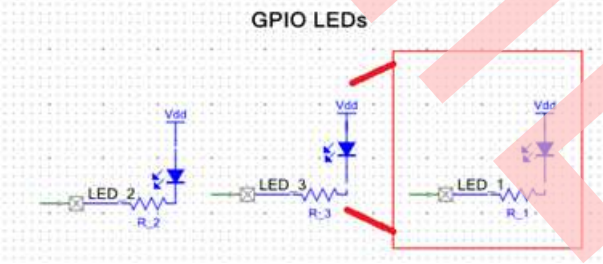
7. Every Component can be disabled (or deleted) by right-clicking on the Component and selecting the corresponding option (see Figure 25). For this design, disable the following Components:
   - *ADC_SAR_Seq_1* and *ADC_In* from the *"BATTERY MONITOR"* section
   - *All GPIO buttons (SW1-SW5)*
   - *TCPWM_1*, *Clock_1*, and *isr_1* from the *"TIMER"* section
   - *SCB_1* from *"Gyro + Accelerometer Interface"* section
   - *I2S_1*, *Clock_4*, *SCK*, *SDI,* and *WS* from the *"Mic Interface"* section

Figure 25. Enable/Disable Components



8. Add one more LED between LED_1 and LED_2. To add, increase the spacing between LED_1 and LED_2 schematics to fit the new LED. Select the LED_1 schematic, right-click **Copy**, and right-click **Paste** to create LED_3 as shown in Figure 26.

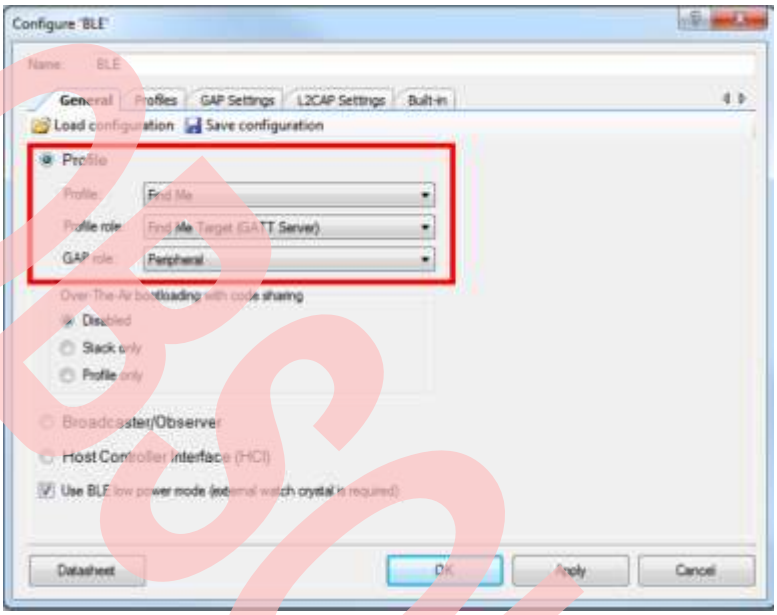
Figure 26. Adding One More LED



9. Double-click the BLE Component on the schematic to configure it as a “BLE Find Me Target” with the following properties:

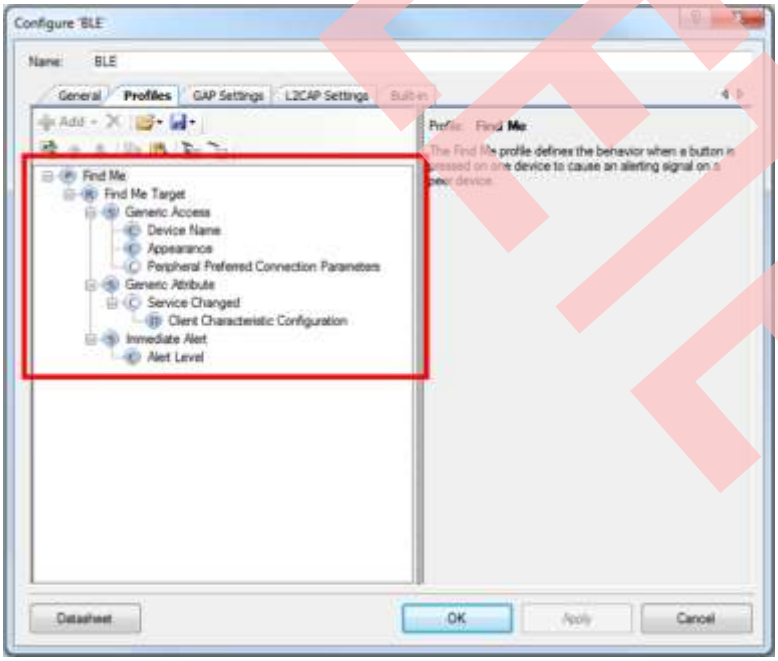
   - “Peripheral” for GAP role and “Find Me Target (GATT server)” for Profile role as shown in Figure 27.

Figure 27. BLE Component General Configuration



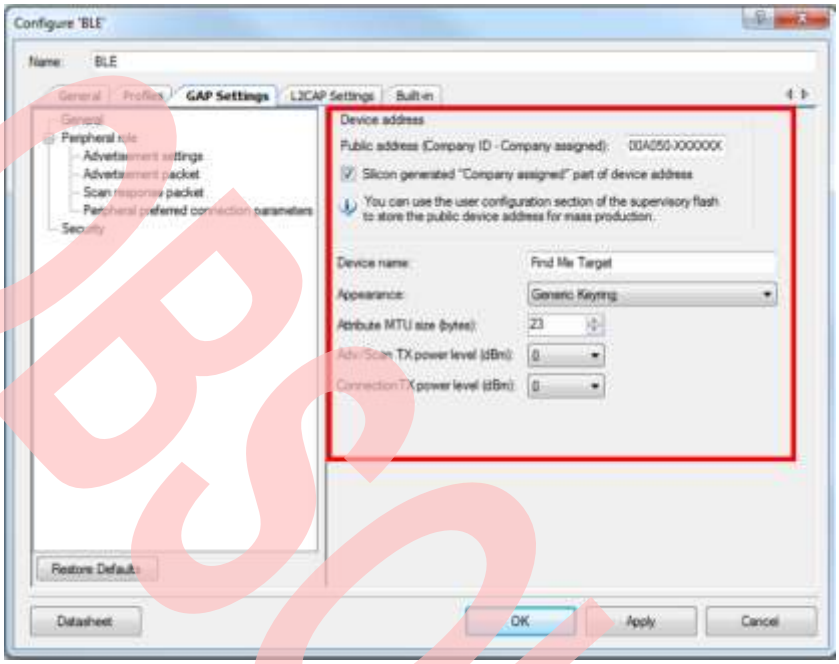
   - Services and characteristics for the Find Me profile are shown in Figure 28 (leave them to default values).

Figure 28. BLE Component Profiles Configuration



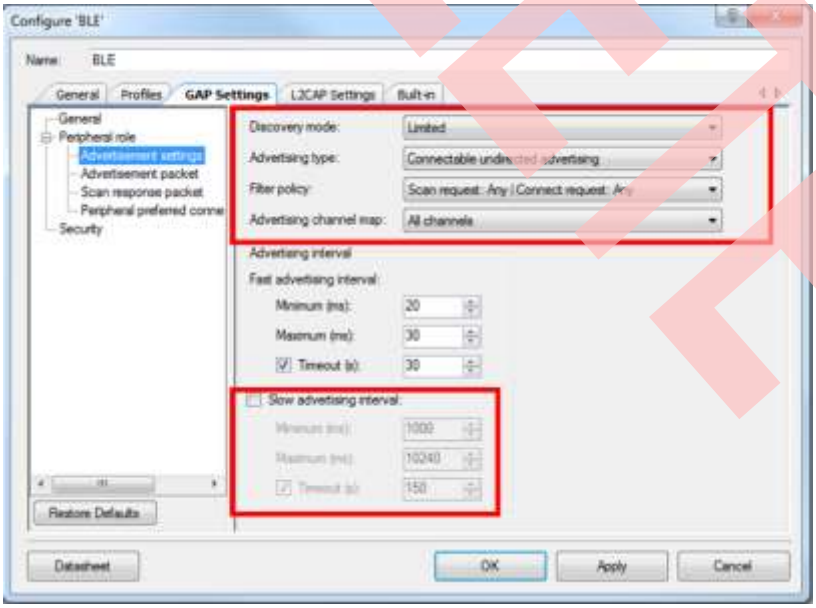
- GAP Device Name set to “Find Me Target” and Appearance set to “Generic Keying” as shown in Figure 29. Select the “Silicon generated” “Company assigned” part of device address.

Figure 29. BLE Component GAP General Settings



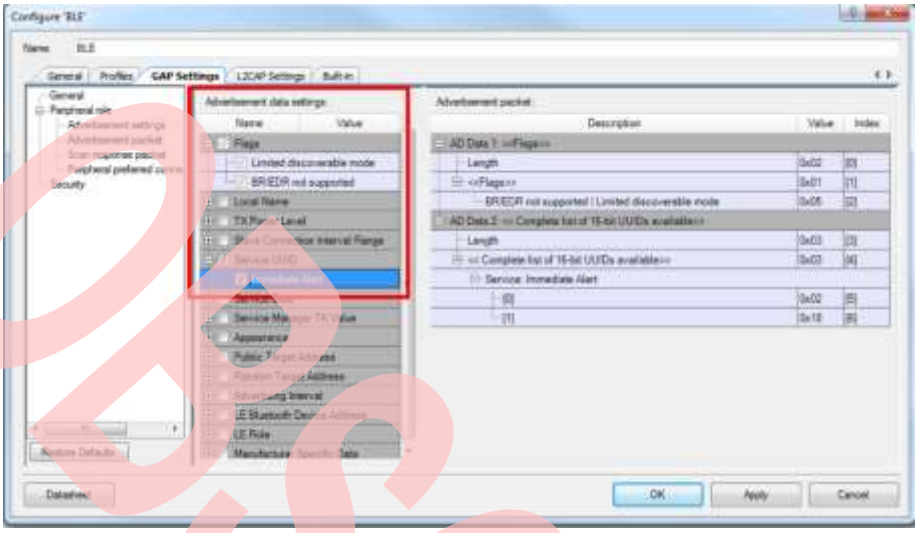
- Limited advertisement mode with an advertising timeout of 30 seconds and fast advertisement interval of 20 to 30 ms is shown in Figure 30. Fast advertising allows quick discovery and connection but consumes more power due to increased RF advertisement packets. Deselect the “Slow advertising interval” checkbox.

Figure 30. BLE Component GAP Advertisement Settings



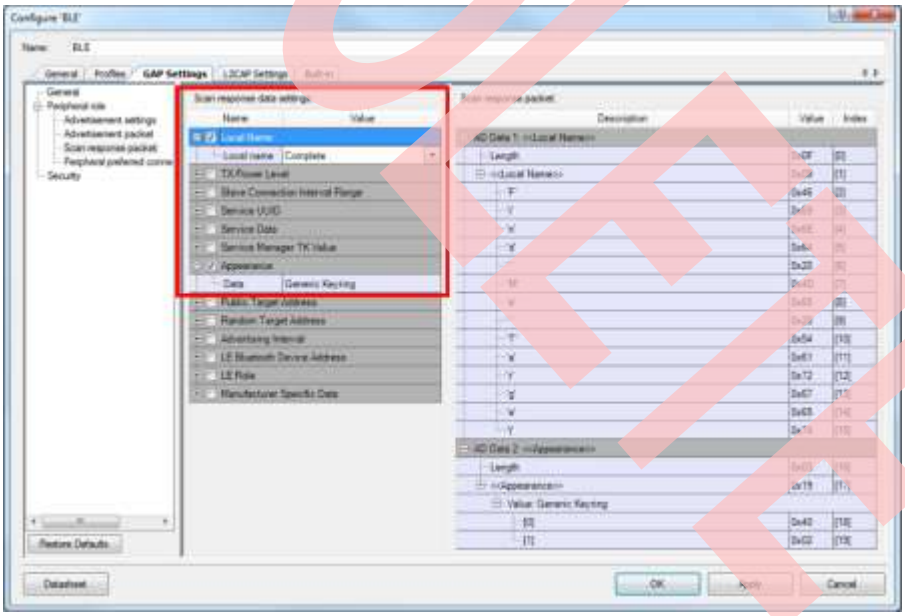
- Advertisement Packet with Immediate Alert Service is enabled as shown in Figure 31.

Figure 31. BLE Component GAP Advertisement Packet



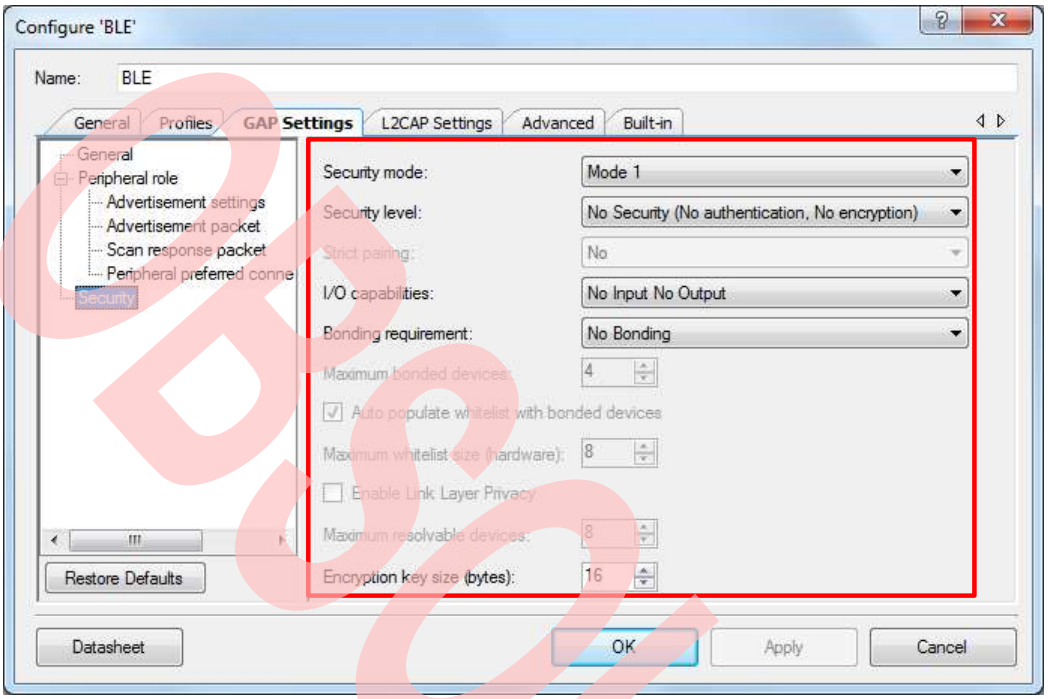
- Scan Response Packet with “Local Name” and “Appearance” fields are enabled as shown in Figure 32.

Figure 32. BLE Component GAP Scan Response Packet



- GAP security set to the lowest possible configuration that does not require authentication, encryption, authorization, or bonding for data exchange (Mode 1, No security), as shown in Figure 33.

Figure 33. BLE Component GAP Security Settings



10. Rename LED_1 and LED_2 to Advertising_LED and Disconnect_LED respectively with “External Terminal” selected, as shown in Figure 34 and Figure 35. Keep the “Initial drive state” set to “High(1),” as 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. These LEDs will be used to indicate BLE advertising and disconnected states.

Figure 34. LED_1 Configuration

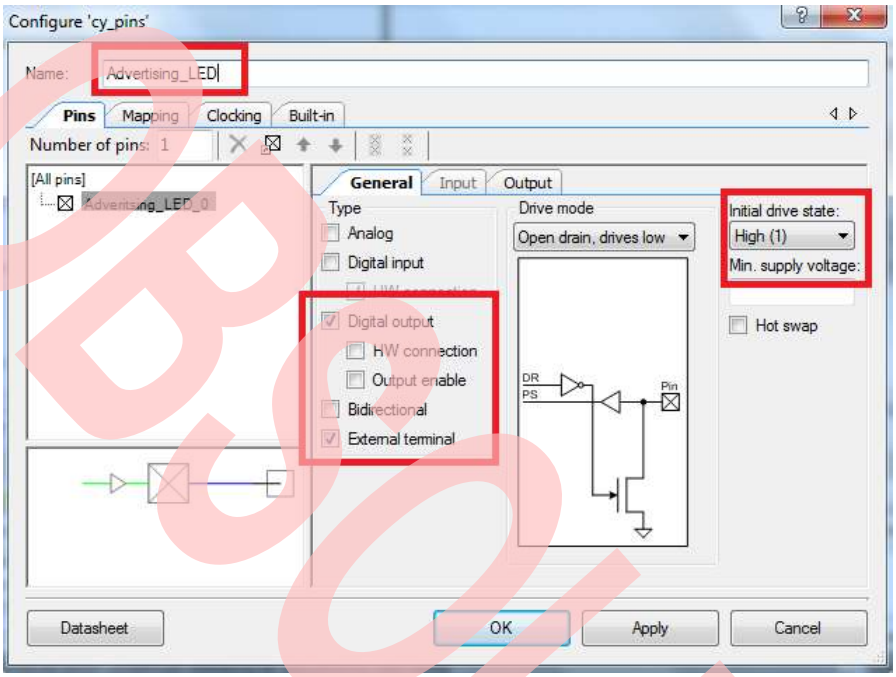
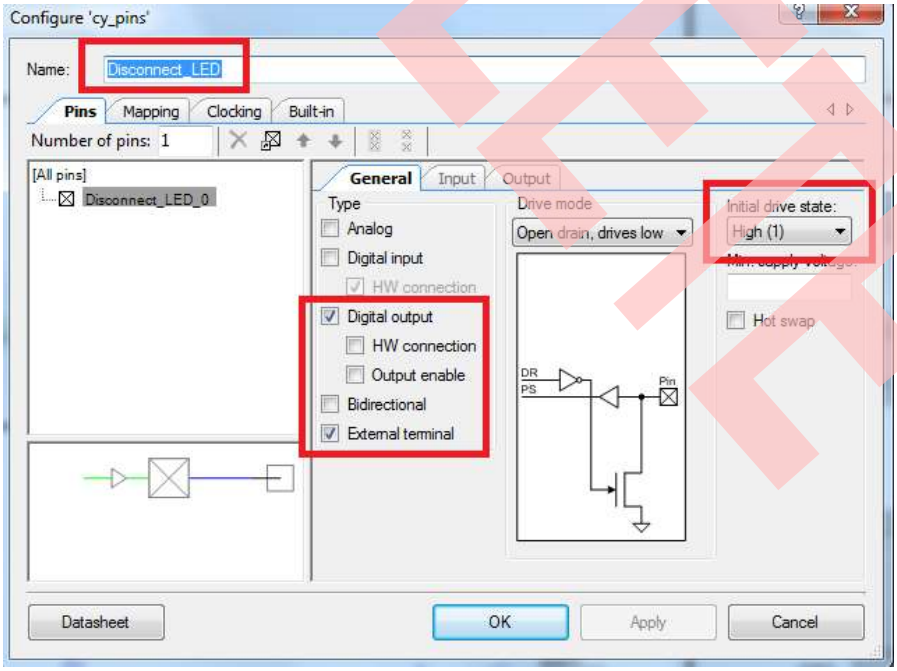
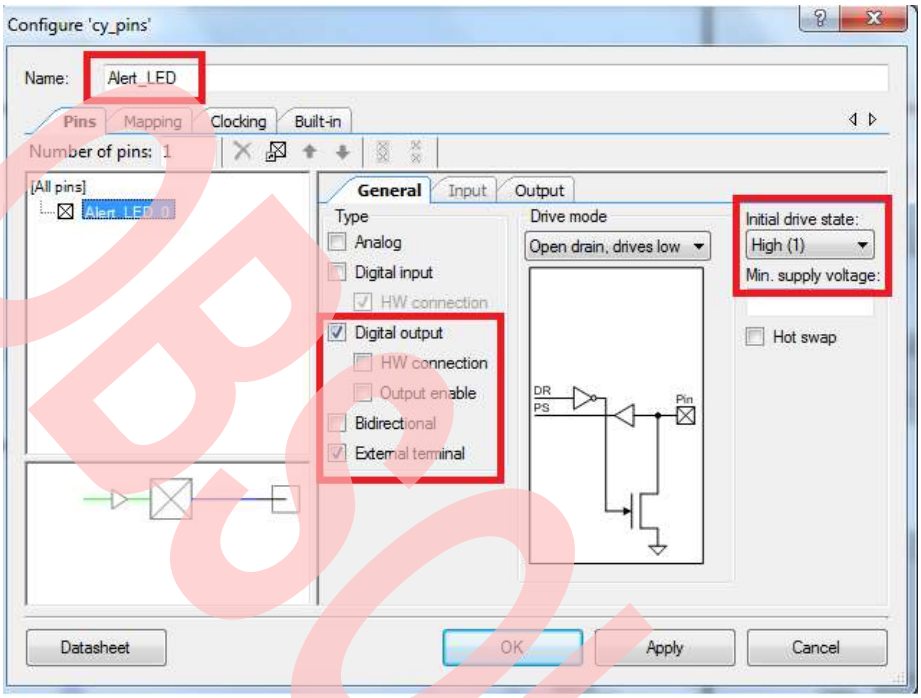


Figure 35. LED_2 Configuration



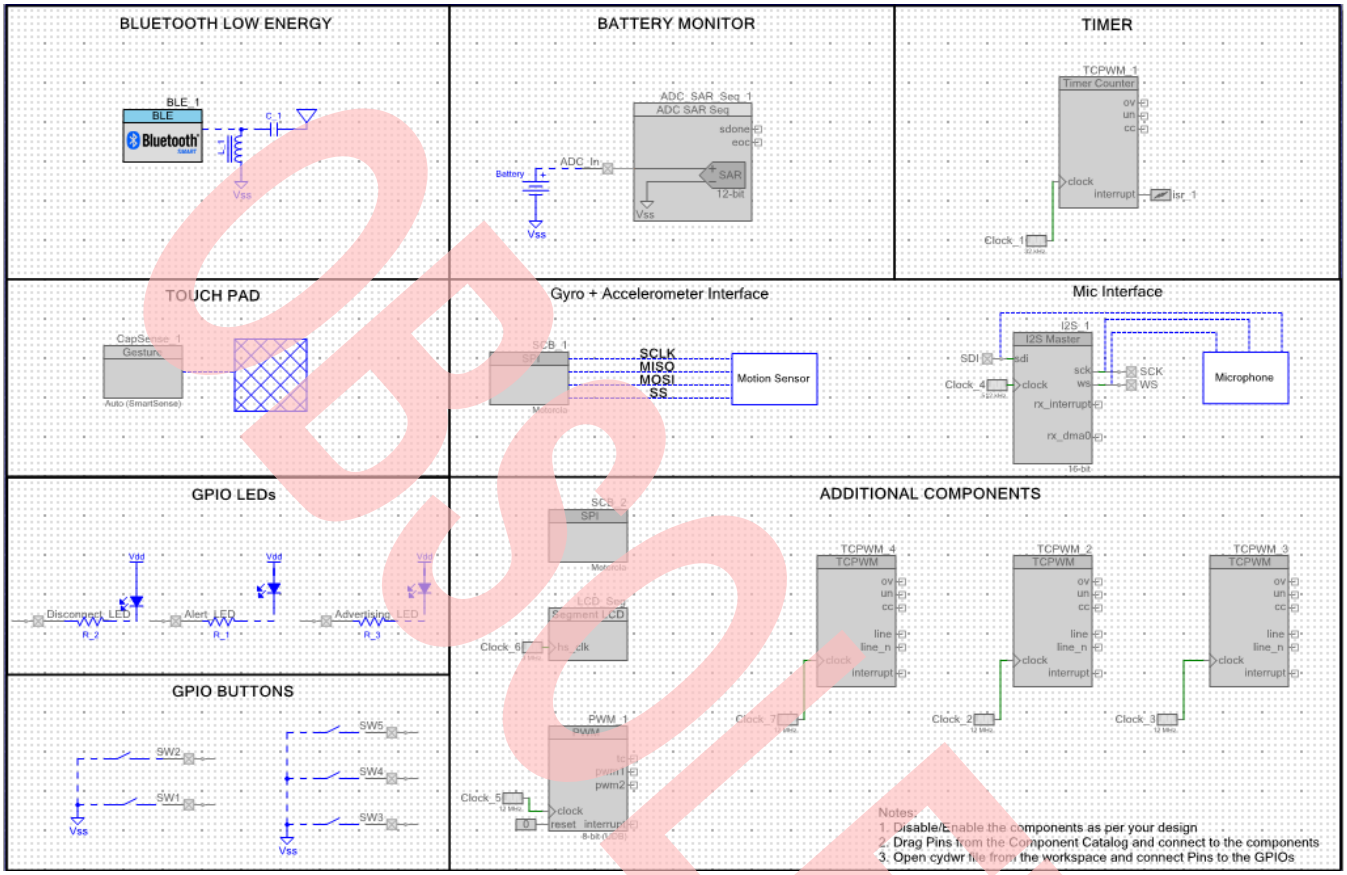
11. Rename LED_3 to “Alert_LED” with “External terminal” selected and “Initial drive state” set to “High(1),” as shown in Figure 36.

Figure 36. LED_3 Configuration



12. After completing the schematic configuration, your design should look similar to Figure 37.

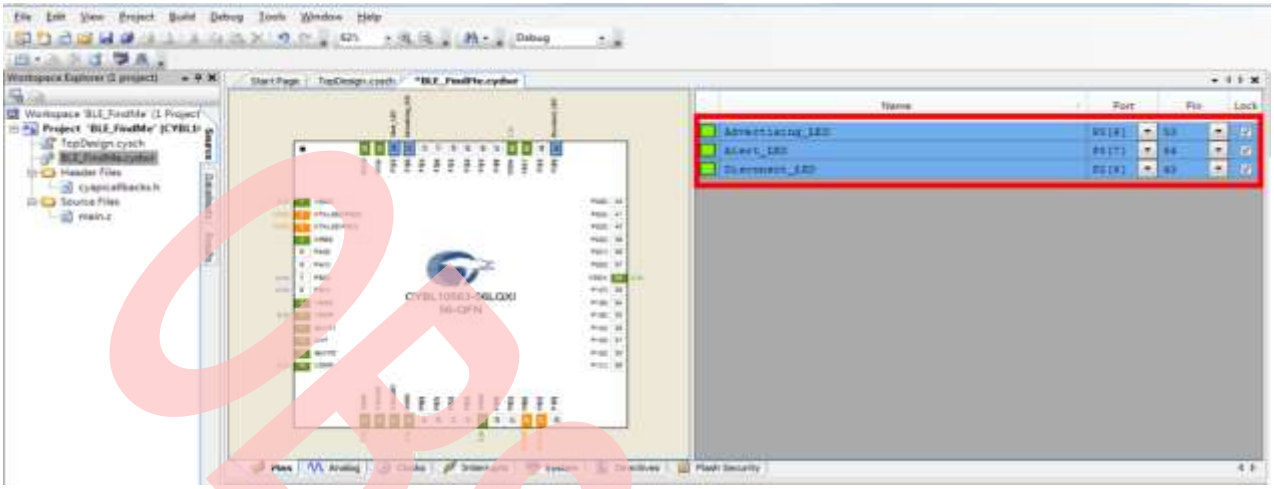
Figure 37. Schematic Configuration



Note: The blue dotted lines, the LED symbols, and resistor symbols shown in Figure 37 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.

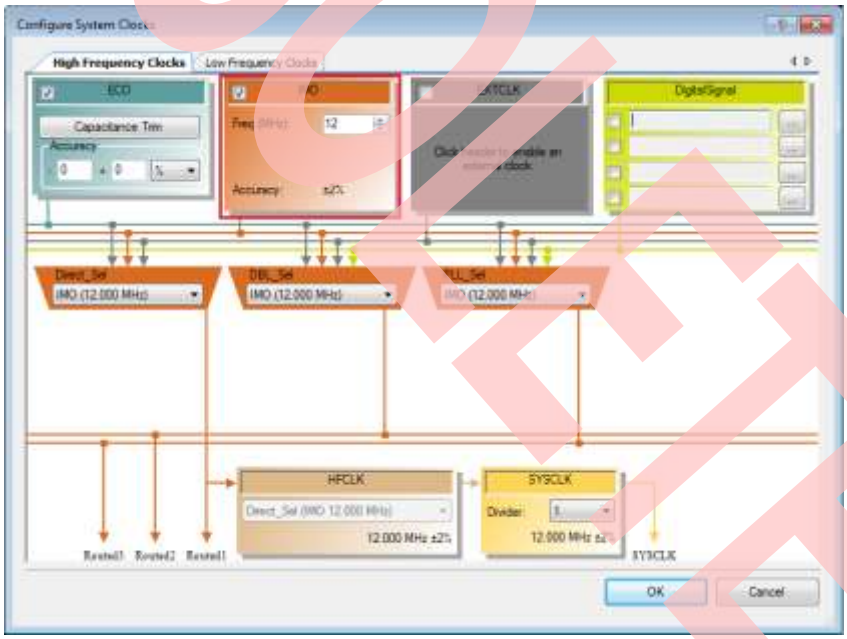
13. Open the file *BLE_FindMe.cydwr* (Design-Wide Resources) file from **Workspace Explorer** and click the **Pins** tab. You can use this tab to select the device pins for the outputs (Advertising_LED, Disconnect_LED, and Alert_LED). Figure 38 shows the pin configuration to connect the Advertising_LED, Disconnect_LED, and Alert_LED pins to the green, red, and blue LEDs on the BLE Pioneer Kit respectively.

Figure 38. Pin Selection



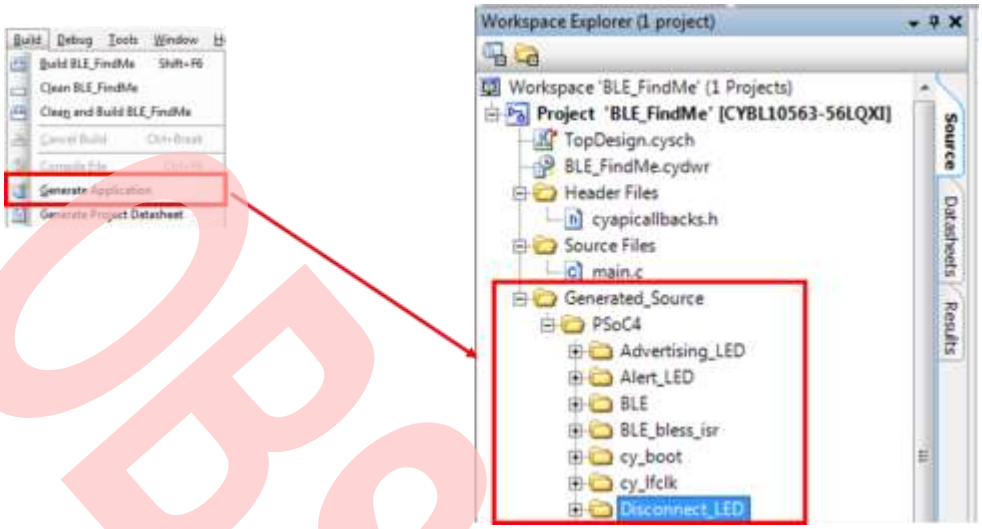
14. Similarly, in the **Clocks** tab under the *BLE_FindMe.cydwr* file, double-click on **IMO** to configure the internal main oscillator (IMO) to 12 MHz, as Figure 39 shows. Click **OK**.

Figure 39. Clock Configuration



15. Select "**Generate Application**" from the **Build** menu. Notice in the "**Workspace Explorer**" window that PSoC Creator automatically generates source code files for the BLE, Clock, and Digital Output/Input Pin Components, as shown in Figure 40.

Figure 40. Generated Source Files



6.4 Step 2: Write the Application Code

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

This section discusses details of these blocks with respect to the design that we configured in Step 1: Create and configure the Design. The code snippets provided in this section are to be placed inside the *main.c* file.

6.4.1 System Initialization

When the PRoC BLE device is reset, the firmware needs to perform initialization, which includes platform initialization, enabling global interrupts, and enabling all the components used in the design. After the system is initialized, the firmware initializes the BLE Component, which internally initializes the complete BLE subsystem. Figure 41 shows the flowchart for system initialization.

As a part of the BLE Component initialization, you must pass the event handler function that is called by the BLE stack to notify pending events. The BLE stack event handler shown in Figure 44 is registered as part of the BLE initialization. If the BLE Component initializes successfully, the firmware registers another event handler for specific IAS events and switches control to the main loop. The main loop code segment is explained in the Main Loop and Low Power Operation section. Figure 42 shows the firmware source code for system initialization.

Figure 41. System Initialization Flowchart

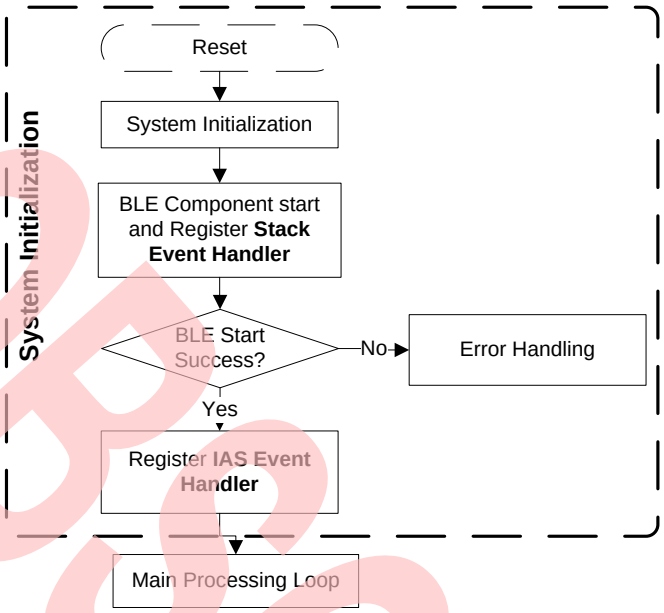


Figure 42. System Initialization Firmware

```
#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)

void StackEventHandler(uint32 event, void *eventParam);
void IasEventHandler(uint32 event, void *eventParam);

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);

  /* Place the main application loop here */

}
```

6.4.2 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 during the CyBle_Start API call as shown in Figure 42. The event handler must handle some basic events from the stack, and configure the stack to establish and maintain the BLE link. For the Find Me application that is discussed here, 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 43 and Figure 44.

Table 1. BLE Stack Events

BLE Stack Event Name	Event Description	Event Handler Action
CYBLE_EVT_STACK_ON	BLE firmware stack within the BLE Component initialized successfully	Start the advertisement and reflect the advertisement state on the LEDs
CYBLE_EVT_GAP_DEVICE_DISCONNECTED	BLE link is disconnected from the peer device	Start the re-advertisement and reflect the advertisement state on the LEDs
CYBLE_EVT_GAP_DEVICE_CONNECTED	BLE link is established with the peer device	Update the BLE link state on LEDs
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 43. BLE Stack Event Handler Flow Chart

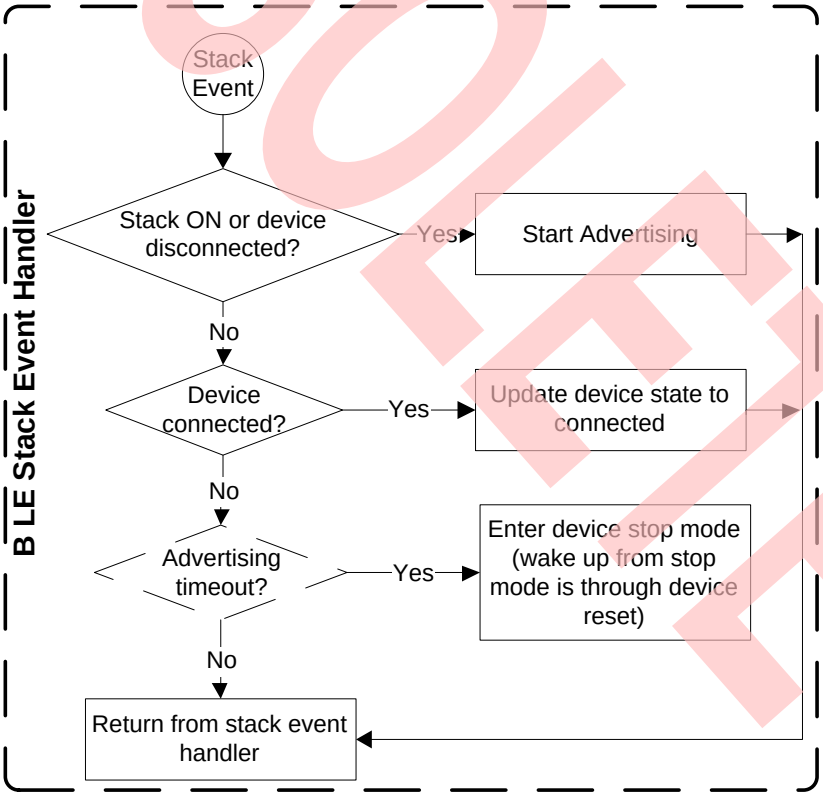


Figure 44. BLE Stack Event Handler Firmware

```
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;
    }
}
```

6.4.3 BLE Service-Specific Event Handler

In addition to the basic events listed in Table 1, the BLE Component 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 lets 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 45 and Figure 46.

Figure 45. BLE IAS Event Handler Flowchart

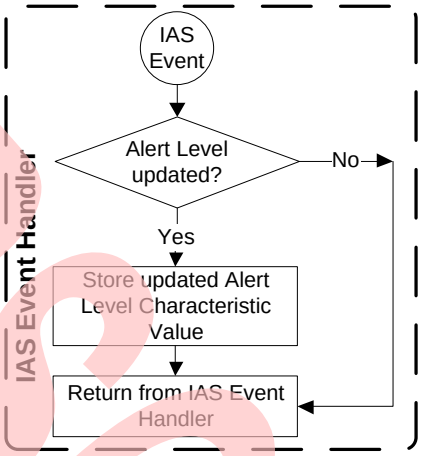


Figure 46. BLE IAS Event Handler Firmware

```
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);
    }
}
```

6.4.4 Main Loop and Low Power Operation

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 BLE subsystem (BLESS) block and the PRoC BLE device into low-power mode. Figure 47 and Figure 48 show the main loop flowchart and firmware. This main loop code should be placed inside the *main* function, after IasEventHandler function call is registered.

Figure 47. Main Loop Flowchart

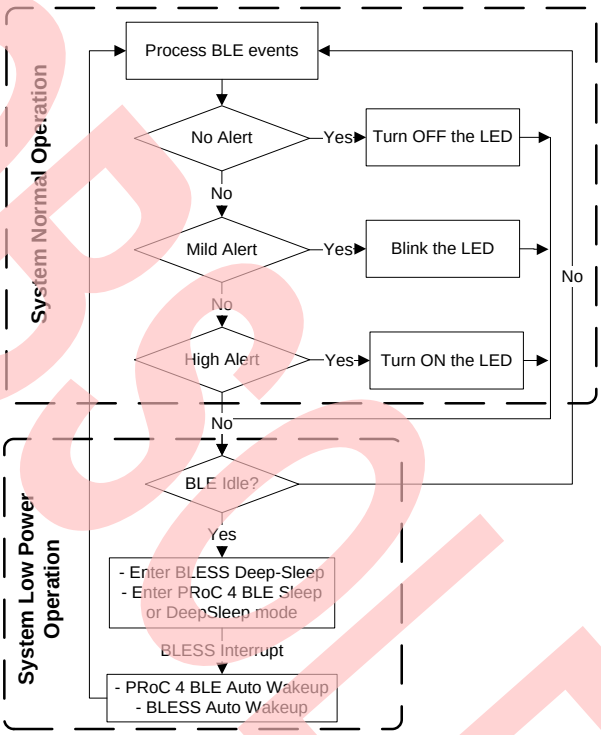


Figure 48. Firmware Main Loop

```
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 */
}
```

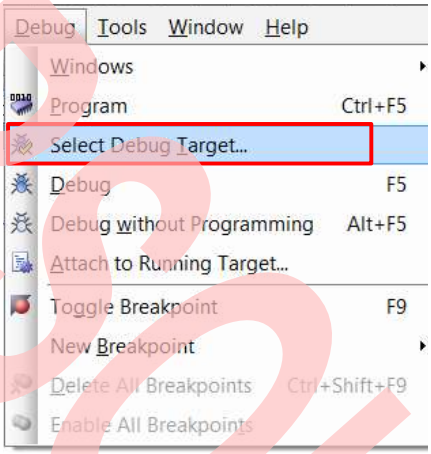
6.5 Step 3: Program the Device

This section shows how to program the device through the PSoC Creator. If you are using a development kit with a built-in programmer, connect the kit board to your computer using the USB cable. For other kits, refer to the kit user guide. If you are developing on your own hardware, you need a hardware debugger, like the Cypress CY8CKIT-002 MiniProg3, to program the device.

Note: The source project for this design is in PSoC Creator 3.3 SP1 or later under **File** > **Code Example**, select **Filter by** as **Find Me** > **BLE_FindMe**.

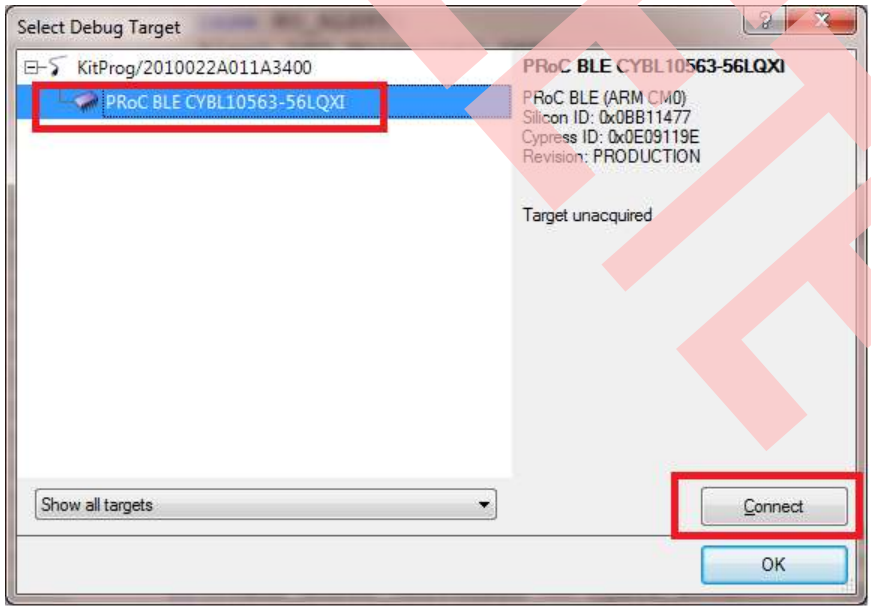
1. In PSoC Creator, choose **Debug** > **Select Debug Target**, as Figure 49 shows.

Figure 49. Selecting Debug Target



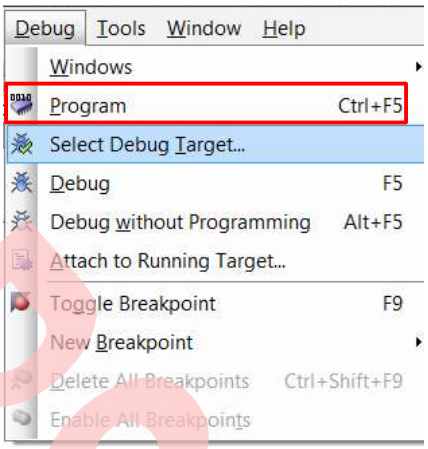
2. In the **Select Debug Target** dialog box, click **Port Acquire**, and then click **Connect**, as Figure 50 shows. Click **OK** to close the dialog box.

Figure 50. Connecting to a Device



3. Choose the **Debug** > **Program** to program the device with the project, as Figure 51 shows.

Figure 51. Programming the Device



You can view the programming status on the PSoC Creator status bar (lower-left corner of the window), as Figure 52 shows,

Figure 52. Programming Status

6.6 Step 4: Test Your Design

This section describes how to test your BLE design using the CySmart mobile app and PC application. The setup for testing your design using the BLE Pioneer Kit is shown in Figure 16.

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 53 and Figure 54.

Figure 53. Testing with CySmart iOS App

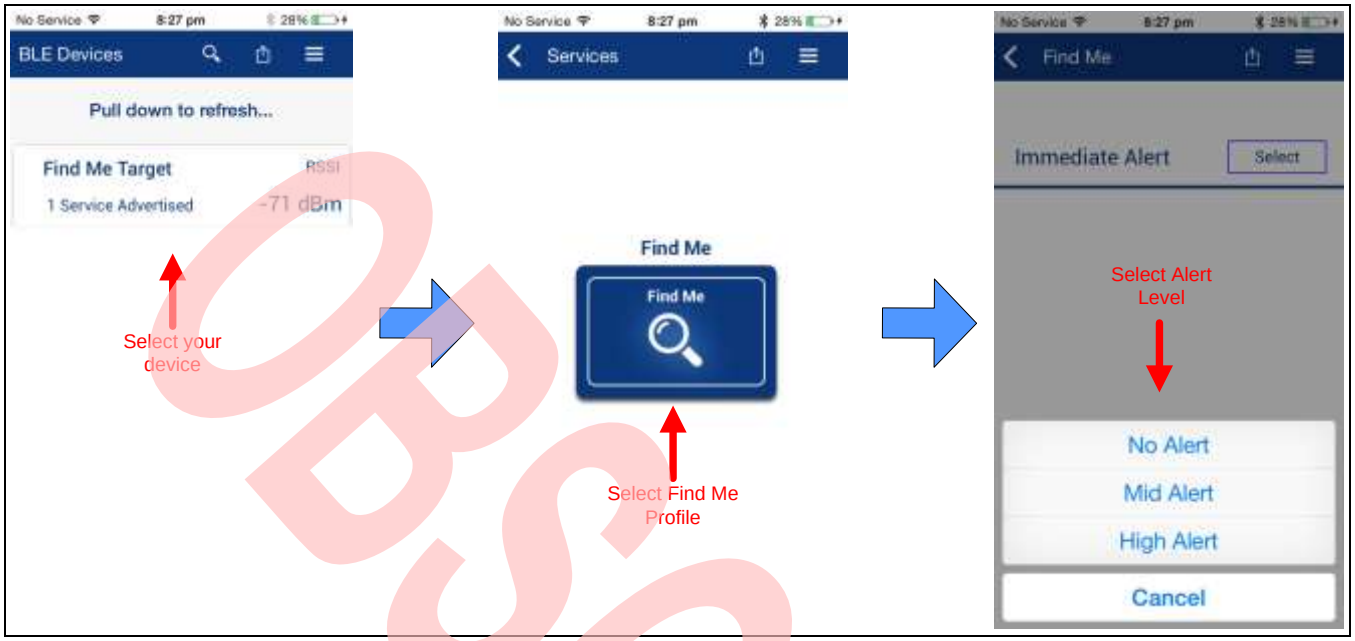
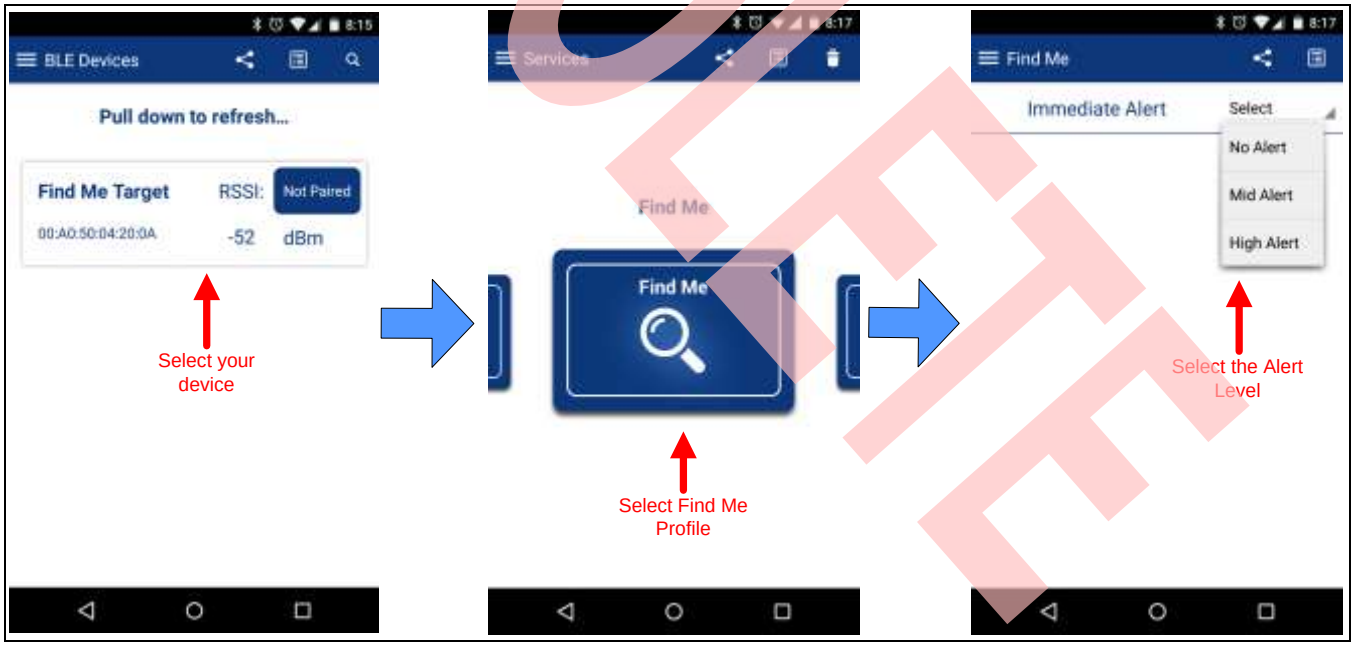


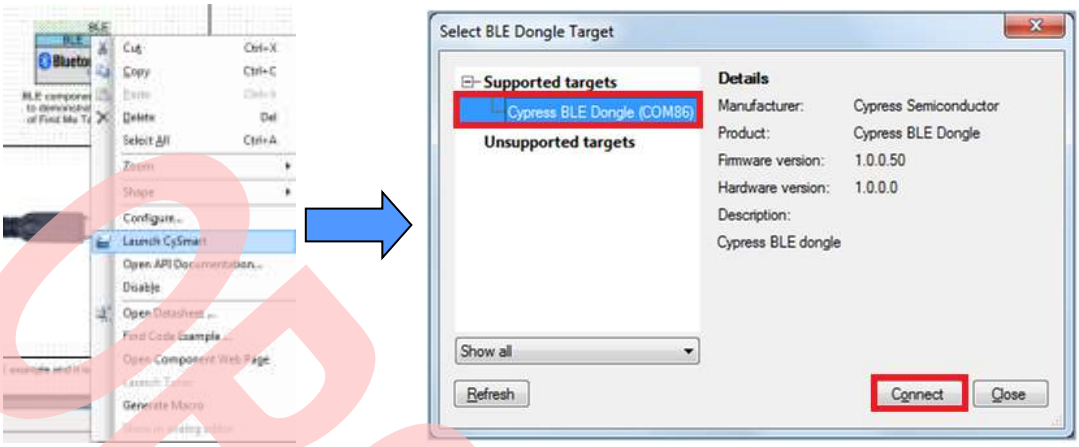
Figure 54. Testing with CySmart Android App



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.

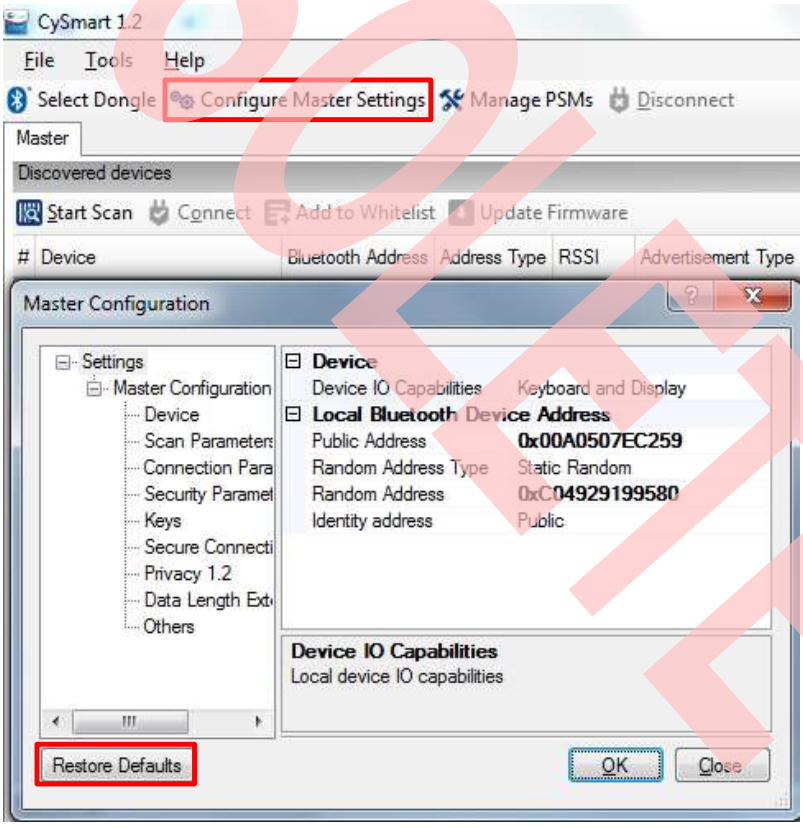
1. Connect the BLE Dongle to your Windows machine. Wait for the driver installation to be completed.
2. Launch the CySmart Host Emulation Tool; it automatically detects the BLE Dongle. Click **Refresh** if the BLE Dongle does not appear in the **Select BLE Dongle Target** pop-up window. Click **Connect,** as shown in Figure 55.

Figure 55. CySmart BLE Dongle Selection



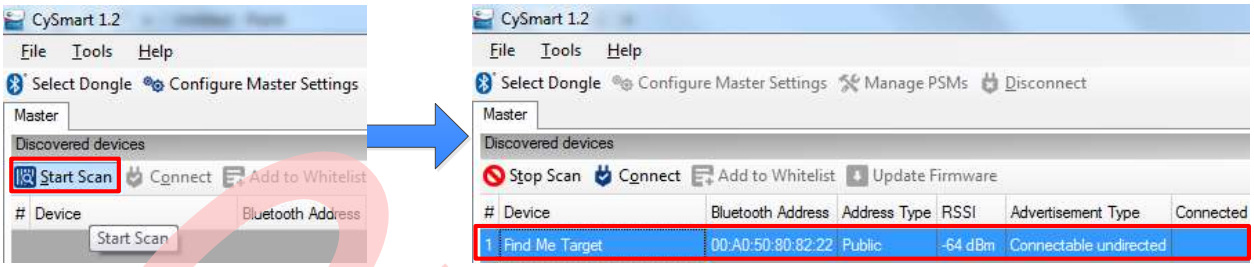
3. Select **Configure Master Settings** and restore the values to the default settings, as shown in Figure 56.

Figure 56. CySmart Master Settings Configuration



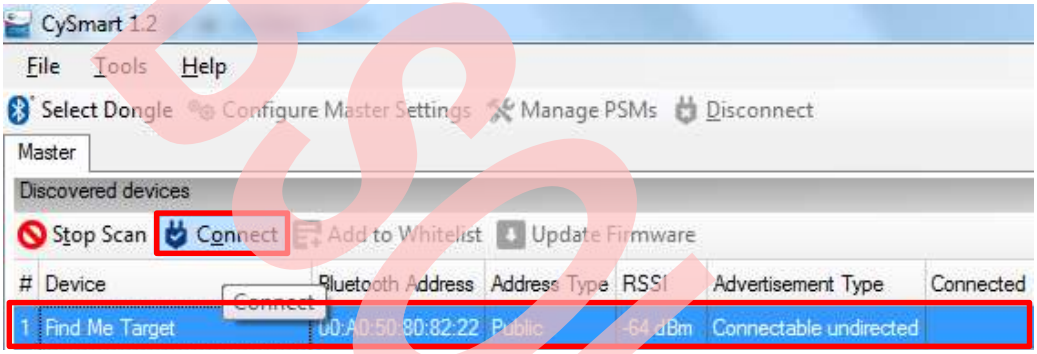
4. Press the reset switch on the BLE Pioneer Kit to start BLE advertisements from your design.
5. On the CySmart Host Emulation Tool, click **Start Scan**. Your device name should appear in the **Discovered devices** list, as shown in Figure 57.

Figure 57. CySmart Device Discovery



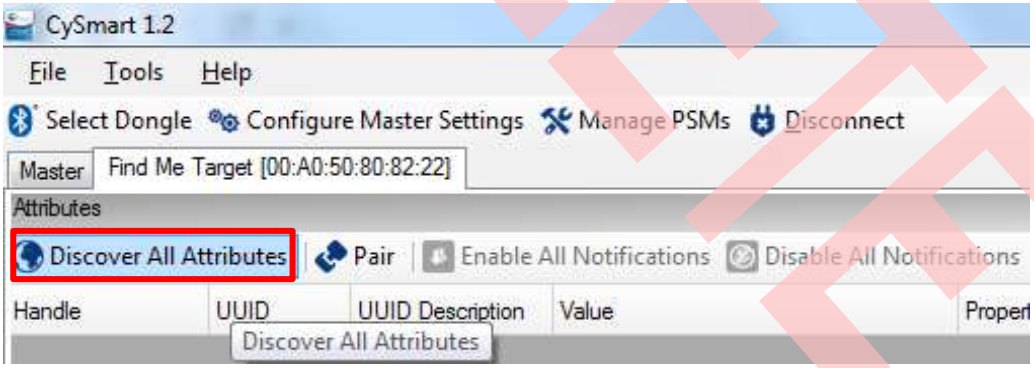
6. Select your device and click **Connect** to establish a BLE connection between the CySmart Host Emulation Tool and your device, as shown in Figure 58.

Figure 58. CySmart Device Connection



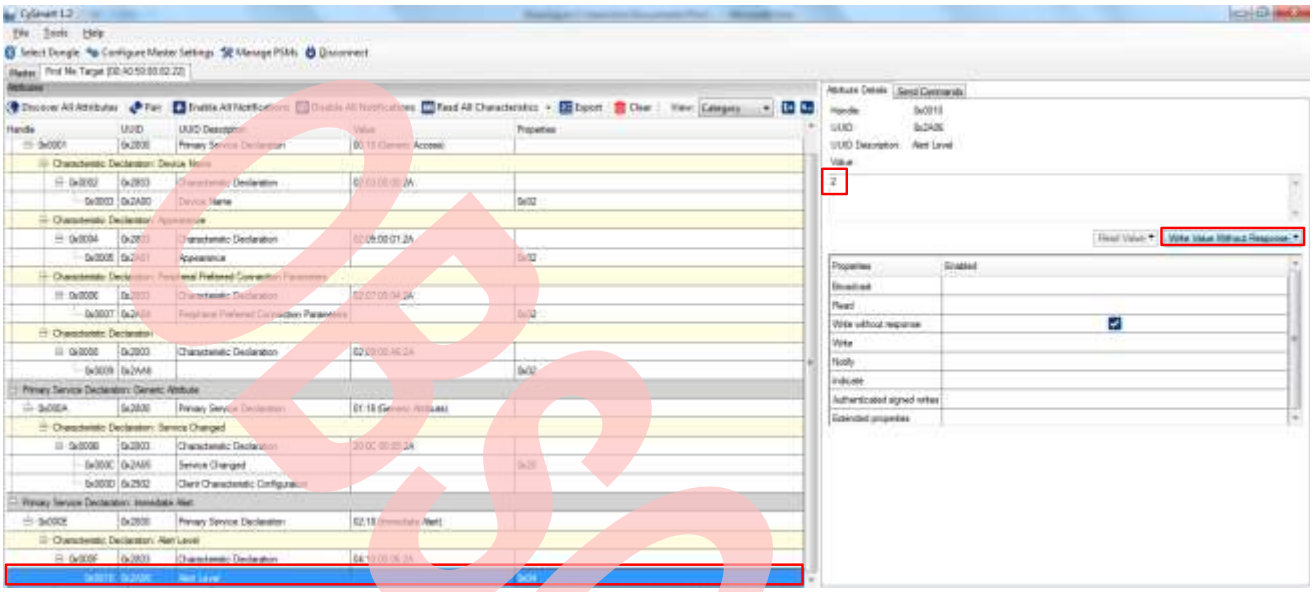
7. Once connected, discover all the Attributes on your design from the CySmart Host Emulation Tool, as shown in Figure 59.

Figure 59. CySmart Attribute Discovery



8. Write a value of 0, 1, or 2 to the Alert Level Characteristic under the Immediate Alert Service, as Figure 60 shows. Observe the state of the LED on your device change per your Alert Level Characteristic configuration.

Figure 60. Testing with CySmart Host Emulation Tool



7 Summary

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

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

8 Related Application Notes

- AN91445 – Antenna Design Guide
- AN91267 – Getting Started with PSoC® 4 BLE
- 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
- AN96841 – Getting Started With EZ-BLE™ PRoC™ Module
- AN95089 – PSoC 4/PRoC BLE Crystal Oscillator Selection and Tuning Techniques
- AN97060 - PSoC 4 BLE and PRoC BLE - Over-The-Air (OTA) Device Firmware Upgrade (DFU) Guide

About the Authors

Name: Sai Prashanth Chinnapalli (CSAI)

Title: Applications Engineer Staff

Background: Sai has a BSEE from Motilal Nehru National Institute of Technology, Allahabad, India

Name: Rahul Garg (RAHU)

Title: Systems Engineer Sr.

Background: Rahul has a B.Tech degree from Bharathi Vidyapeeth's College of Engineering, New Delhi, India

Name: Santhosh Kumar Vojjala

Title: Applications Engineer Staff

Background: Santhosh has a Master's degree from Indian Institute of Technology Bombay, India.

A 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-BLXXX	CY8C42x7-BLXXX	CYBL10X6X	CYBL1XX7X	CY8C41x8-BL	CY8C42x8-BL
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	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
Bluetooth 4.2 features	LE secure connection	LE secure connection	LE secure connection	LE secure connection, link layer privacy, and link layer data length extension	LE secure connection, link layer privacy, and link layer data length extension*	LE secure connection, link layer privacy, and link layer data length extension*
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	48-MHz ARM Cortex-M0 CPU with single-cycle multiply	24-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	256 KB	256 KB	256 KB
SRAM	16 KB	16 KB	16 KB	32 KB	32 KB	32 KB
GPIOs	Up to 36	Up to 36	Up to 36	Up to 36	Up to 36	Up to 36
CapSense	Up to 35 sensors	Up to 35 sensors	Up to 35 sensors	Up to 35 sensors	Up to 35 sensors	Up to 35 sensors
CapSense Gestures	On selected devices	On selected devices	On selected devices	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	12-bit, 1-Msps SAR ADC with sequencer	12-bit, 806-ksps 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	None	2 programmable opamps that are active in Deep-Sleep mode	4 programmable opamps that are active in Deep-Sleep mode
Comparators	2 low-power comparators with the wakeup feature	2 low-power comparators with the wakeup feature	None	None	2 low-power comparators with the wakeup feature	2 low-power comparators with the wakeup feature
Current DACs	One 7-bit, and one 8-bit	One 7-bit, and one 8-bit	None	None	One 7-bit, and one 8-bit	One 7-bit, and one 8-bit
Power Supply Range	1.9 V to 5.5 V	1.9 V to 5.5 V	1.9 V to 5.5 V	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	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	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

Features		Device Family				
	CY8C41x7-BLXXX	CY8C42x7-BLXXX	CYBL10X6X	CYBL1XX7X	CY8C41x8-BL	CY8C42x8-BL
Serial Communication	2 independent serial communication blocks (SCBs) with programmable I^2C, SPI, or UART	2 independent SCBs with programmable I^2C, SPI, or UART	1 or 2 independent SCBs with programmable I^2C, SPI, or UART	1 or 2 independent SCBs with programmable I^2C, SPI, or UART	2 independent serial communication blocks (SCBs) with programmable I^2C, SPI, or UART	2 independent SCBs with programmable I^2C, SPI, or UART
Timer Counter Pulse-Width Modulator (TCPWM)	4	4	4	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	None	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)
Additional Digital Peripherals (I^2S, PWM)	None	Yes (UDB-based digital peripherals on select devices)	Yes (fixed-function blocks on select devices)	Yes (fixed-function blocks on select devices)	None	Yes (UDB-based digital peripherals 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	3-MHz to 48-MHz IMO 32-kHz ILO 24-MHz ECO 32-kHz WCO	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
Power Supply Monitoring	Power-on reset (POR) Brown-out detection (BOD) Low-voltage detection (LVD)	POR BOD LVD	POR BOD LVD	POR BOD 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)	56-QFN (7.0 × 7.0 × 0.6 mm) and 76-WLCSP (4.04 × 3.87 × 0.55 mm)	56-QFN* (7.0 × 7.0 × 0.6 mm) and 76-WLCSP (4.04 × 3.87 × 0.55 mm)	56-QFN* (7.0 × 7.0 × 0.6 mm) and 76-WLCSP (4.04 × 3.87 × 0.55 mm)
DMA	None	None	None	Up to 8 channels	Up to 8 channels*	Up to 8 channels*

* = CY8C41x8-BL and CY8C42x8-BL family has two sub-families –BL4xx and –BL5xx. The –BL4xx family does not support Bluetooth 4.2 link layer privacy, link layer data length extension, and DMA engine, the –BL5xx series does.

B 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.

PRoC BLE: A fixed-function SoC with integrated BLE radio, which 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.

C Appendix C: Cypress BLE Development Tools

C.1 PRoC BLE Development Kit

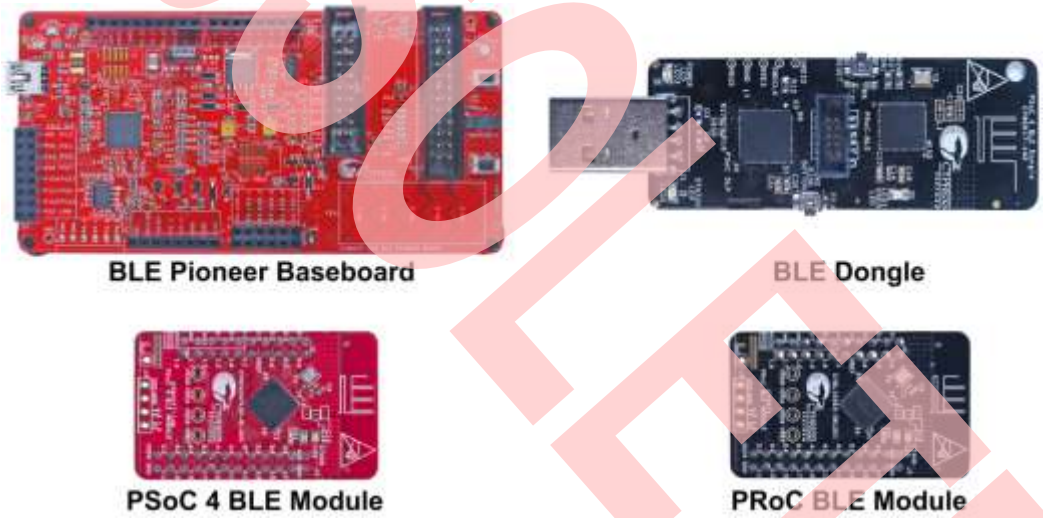
The BLE Pioneer Kit shown in Figure 61 is a BLE development kit from Cypress that supports both PRoC BLE and PRoC BLE family of devices. This kit consists of pluggable PRoC BLE (and PSoC 4 BLE) modules that connect to a BLE Pioneer Baseboard. The kit can be powered either with a coin-cell battery or 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 PRoC 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 to get the latest updates on the kit and download the kit design, example projects, and documentation files.

Figure 61. BLE Pioneer Kit



Note: In PRoC BLE development kit, the PRoC BLE device on the module (black colored module in Figure 61. BLE Pioneer kit has the CYBL10X6X device with 128KB of flash memory. The modules with devices having 256 KB of flash memory (CYBL10X7X) can be ordered separately through CY5676 webpage on Cypress website.

C.2 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. 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.
- Over-the-air (OTA) firmware upgrade of Cypress BLE Peripheral devices

Figure 62 and Figure 63 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 62. CySmart Host Emulation Tool Master Device Tab

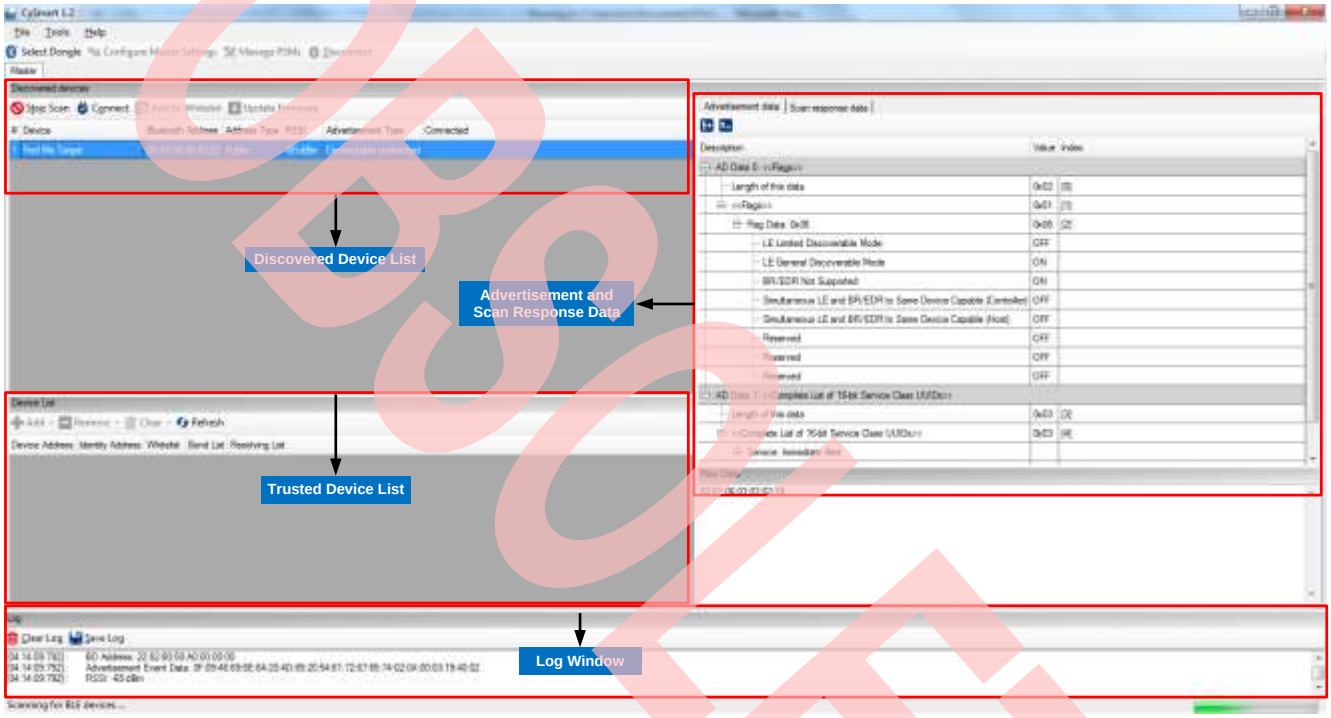
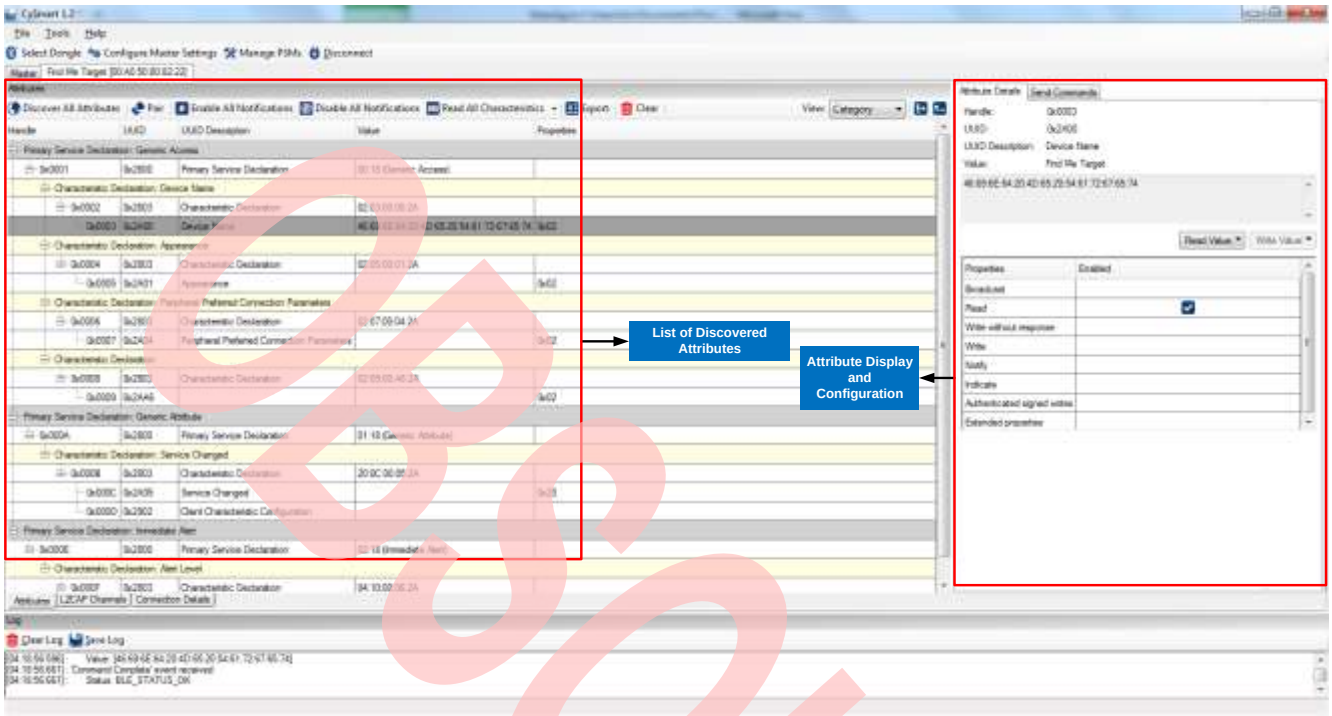


Figure 63. CySmart Host Emulation Tool Peripheral Device Attributes Tab



C.3 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 64 and Figure 65 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 64. CySmart iOS App Heart Rate Profile Example

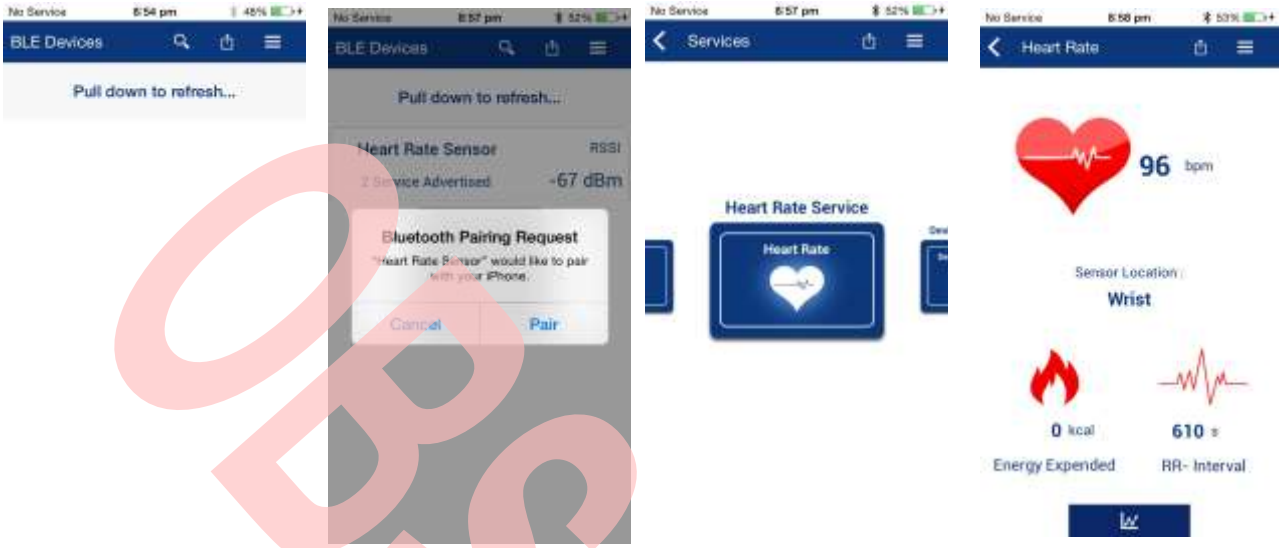
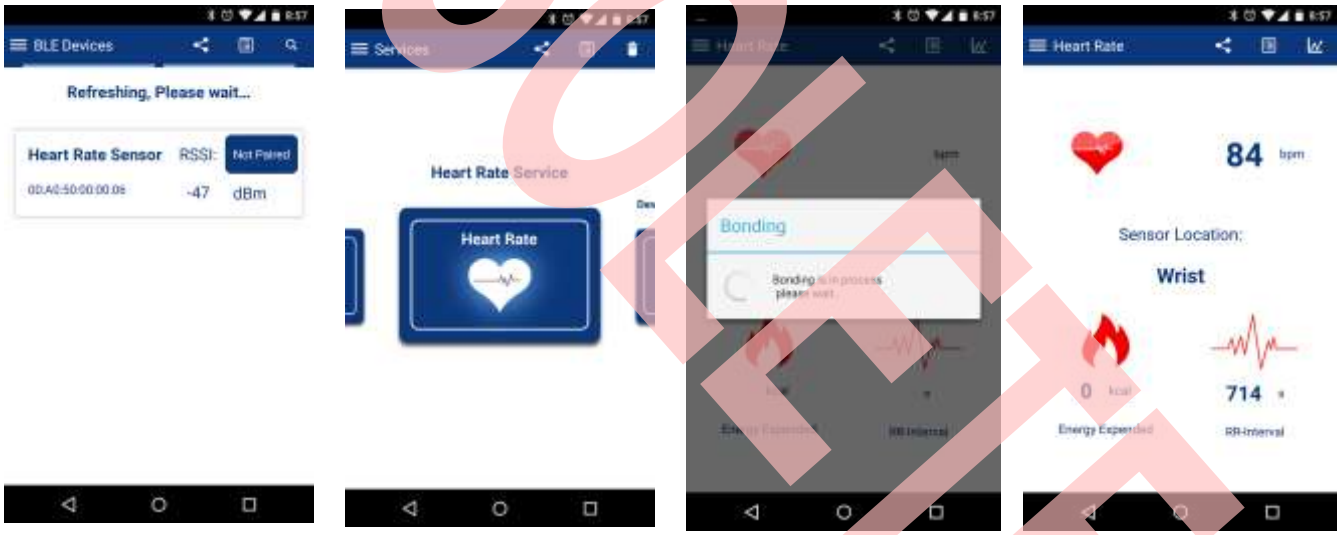


Figure 65. CySmart Android App Heart Rate Profile Example



D Appendix D: PRoC BLE Device

D.1 ARM Cortex-M0, Memory and DMA

PRoC 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 Cortex-M0 CPU supports single-cycle 32-bit multiplication. PRoC BLE has up to 32 KB of SRAM and up to 256 KB of flash memory; the flash memory includes a read accelerator that delivers 85% of single-cycle SRAM access performance on average.

A DMA engine, with eight channels, is provided on CYBL1XX7X family of devices, which can do 32-bit data transfer.

D.2 BLE Subsystem

The BLE subsystem (BLESS) consists of the BLE Link Layer engine and the Physical Layer. The Link Layer engine supports both master and slave roles. The Link Layer engine implements time-critical procedures related to device discovery and connection, and AES encryption in the hardware to reduce the power consumption. It requires minimal processor intervention to offer high performance. The key protocol elements such as Link Layer control and host protocols are implemented in firmware. The direct test mode (DTM) is included to test the radio performance using a standard Bluetooth tester. The physical layer consists of a modem and an RF transceiver that transmits and receives BLE packets at the rate of 1 Mbps over the 2.4-GHz ISM band. In the transmit direction, this block performs GFSK modulation and then converts the digital baseband signal of these BLE packets into radio frequency before transmitting them to air through an antenna. In the receive direction, this block converts an RF signal from the antenna to a digital bit stream after performing GFSK demodulation.

The RF transceiver contains an integrated balun, which provides a single-ended RF port pin to drive a 50-Ω antenna terminal through a pi-matching network. The output power is programmable from -18 dBm to +3 dBm to optimize the current consumption for different applications.

The BLE protocol stack provides the following features:

- Link Layer (LL)
  - Master and slave roles
  - 128-bit AES encryption
  - Low-duty-cycle advertising
  - Low Energy ping
  - LE Data Packet Length Extension (Bluetooth 4.2 feature)
  - Link Layer Privacy (with extended filter policy) (Bluetooth 4.2 feature)
  - LE Secure connections (Bluetooth 4.2 feature)
- BLE 4.2 single-mode host stack with
  - Logical Link Control and Adaptation Protocol (L2CAP), Attribute (ATT), and Security Manager (SM) protocols
  - Generic Attribute Profile (GATT) and Generic Access Profile (GAP)
- API access to GATT, GAP, and L2CAP
- L2CAP connection-oriented channel
- GAP features
  - Broadcaster, observer, peripheral, and central roles and procedures
  - Security mode 1: Level 1, 2, and 3
  - Security mode 2: Level 1 and 2
  - User-defined advertising data
  - Multiple-bond support
- GATT features
  - GATT client and server
  - Support for GATT sub procedures
  - 32-bit UUIDs
- Security
  - Pairing methods: Just Works, Passkey Entry, and Out of Band
  - Authenticated man-in-the-middle (MITM) protection and data signing
  - BLE privacy and signed data

- Support for all SIG-adopted BLE profiles
  The Cypress-supplied software (PSoC Creator) makes designing a BLE solution very easy. This GUI-based configuration tool allows you to configure the roles, features, and parameters of the BLE protocol stack. For more information, refer to the BLE Component datasheet.

D.3 Programmable Digital Peripherals

PRoC BLE provides a rich set of digital and analog peripherals. Digital peripherals include SCBs, TCPWMs, and I2S. Analog peripherals include a fast 12-bit SAR ADC, capacitive touch sensing (CapSense), and segment LCD direct drive.

D.3.1 Programmable SCBs

PRoC BLE has two independent run-time programmable SCBs with I²C, SPI, or UART interfaces. These SCBs support 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 I²C master and slave functionality
- SPI and I²C modes that allow operation without CPU intervention
- Low-power (Deep-Sleep) mode of operation for SPI and I²C protocols (using an external clock)

For more information, refer to the SCB Component datasheet.

D.3.2 Programmable TCPWMs/PWMs

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

For more information, refer to the TCPWM Component datasheet.

In addition to TCPWMs, some members of the family contain four additional PWMs. The PWM peripheral can be configured with 8- or 16-bit resolution. The PWM provides compare outputs to generate single or continuous timing and control signals in hardware. It also provides an easy method of generating complex real-time events accurately with minimal CPU intervention. A maximum of four 8-bit PWMs or two 16-bit PWMs are available.

For more information, refer to the PWM Component datasheet.

D.3.3 Inter-IC Sound Bus Block

PRoC BLE has an I2S block that operates in Master mode, supporting the transmitter (TX) and the receiver (RX), which have independent data byte streams. The block supports 8 to 32 data bits per sample with 16-, 32-, 48-, or 64-bit word select period. The maximum data rate is 96 kHz with 64-bit word select period. There are independent left and right channel FIFOs or interleaved stereo FIFOs are available.

For more information, refer to the I2S Component datasheet.

D.3.4 Segment LCD Direct Driver

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

PRoC 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 system power mode with a system current consumption as low as 7 µA.

For more information, see AN87391 – PSoC 4 Segment LCD Direct Drive.

D.3.5 Programmable GPIOs

PRoC BLE has as many as 36 programmable GPIO pins. You can configure the GPIOs for CapSense, LCD, analog, or digital signals. PRoC BLE GPIOs support multiple drive modes, drive strengths, and slew rates.

PRoC 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.

For more information, see PRoC™ BLE Technical Reference Manual.

D.4 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 longer lifetimes compared to mechanical buttons.

The CapSense feature in PRoC BLE offers a high signal-to-noise ratio and best-in-class waterproofing, and supports a wide variety of sensor types such as buttons, sliders, track pads, and proximity sensors.

The CapSense track pad with gestures has the following features:

- Supports 1-finger and 2-finger touch applications
- Supports up to 35 X/Y sensor inputs
- Includes a gesture-detection library:
  - 1-finger touch: tracing, pan, click, double-click
  - 2-finger touch: pan, click, zoom

A Cypress-supplied software Component makes capacitive sensing design very easy; the Component supports an automatic hardware tuning feature called SmartSense™.

For more information, see the CapSense Design Guide.

D.5 Configurable Analog Peripherals

D.5.1 SAR ADC with Hardware Sequencer

PRoC BLE has a 12-bit, 1-Msps SAR ADC. The input channels support programmable resolution and single-ended or differential input options. The number of channels is limited only by the number of available GPIOs.

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

For more information, refer to the Sequencing Successive Approximation Component datasheet.

D.6 System-Wide Resources

D.6.1 Power Supply and Monitoring

PRoC BLE is capable of operating from a single 1.9-V to 5.5-V supply. There are multiple internal regulators to support the various power modes, including the Active digital regulator, Quiet regulator, Deep-Sleep regulator, and Hibernate regulator.

PRoC BLE has three different types of voltage monitoring capabilities: Power-on reset (POR), Brown out detection (BOD), and Low-voltage detection (LVD).

For more information, see PRoC™ BLE Technical Reference Manual.

D.6.2 Clocking System

PRoC BLE has the following clock sources:

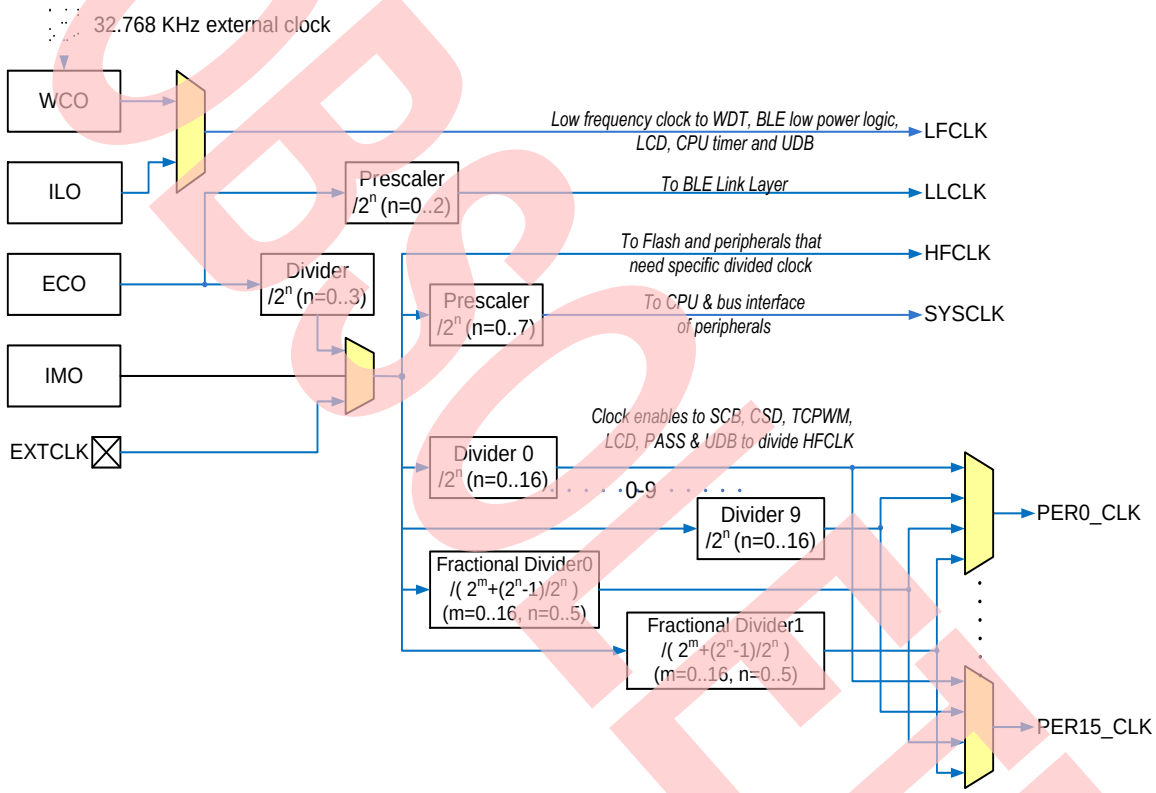
- **Internal main oscillator (IMO):** The IMO is the primary source of internal clocking in PRoC BLE. The CPU and all high-speed peripherals operate from the IMO. PRoC BLE has multiple peripheral clock dividers operating from the IMO, which generate clocks for the high-speed peripherals.
- **Internal low-speed oscillator (ILO)**: The ILO is a very-low-power oscillator, which primarily generates clocks for low-speed peripherals operating in the Deep-Sleep mode.
- **External crystal oscillator (ECO):** The ECO is used as the active clock for the BLE subsystem to meet the ±50-ppm clock accuracy requirement for the Bluetooth 4.2specification. The ECO includes a tunable load capacitor to

tune the crystal clock frequency by measuring the actual clock frequency. The high-accuracy ECO clock can also be used as a system clock.

- **Watch crystal oscillator (WCO):** The WCO is used as the sleep clock for the BLE subsystem to meet the ±500-ppm clock accuracy requirement for the Bluetooth 4.2 specification. The sleep clock provides accurate sleep timing and enables wakeup at specified advertisement and connection intervals. With the WCO and firmware, an accurate real-time clock (within the bounds of the 32.768-kHz crystal accuracy) can be realized.

Figure 66 shows the clocking architecture of a PRoC BLE device. For more information, see PRoC™ BLE Technical Reference Manual.

Figure 66 PRoC BLE Clocking System



D.6.3 Low-Power Mode

PRoC BLE supports the following five power modes as illustrated in Table 3.

Note: These modes are PRoC BLE device power modes, which are different from the power modes described in the BLE Subsystem 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 the 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. The WDT, LCD, I²C/SPI, link layer, and low-frequency clock (32-kHz ILO) are available. Interrupts from GPIO, WDT, or SCBs can cause a wakeup. The current consumption in this mode is 1.3 µA for all PRoC BLE devices in the family.
- **Hibernate mode:** This power mode provides a best-in-class current consumption of 150 nA while retaining SRAM, programmable logic, and the ability to wake up from an interrupt generated by 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.

Table 3: PRoC BLE Low-Power Modes

BLESS Modes	PRoC BLE System Power Modes				
	Active	Sleep	Deep-Sleep	Hibernate	Stop
Transmit	✓	✓	✘	✘	✘
Receive	✓	✓	✘	✘	✘
Idle	✓	✓	✘	✘	✘
Sleep	✓	✓	✘	✘	✘
Deep-Sleep	✓	✓	✓	✘	✘
Powered	✘	✘	✘	✓	✓

D.6.4 Device Security

PRoC BLE provides a number of options for the protection of 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.

D.7 Programmable GPIOs

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

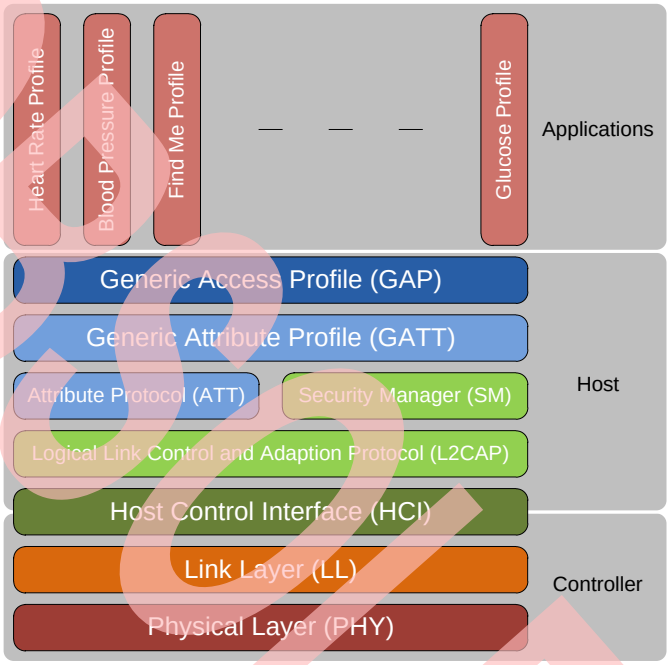
PRoC BLE offers an intelligent routing system that gives multiple choices for connecting an internal signal to a GPIO. This flexible routing simplifies circuit design and board layout.

E Appendix E: BLE Protocol

E.1 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 67 shows the BLE stack.

Figure 67. 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.

E.2 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.

E.3 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.

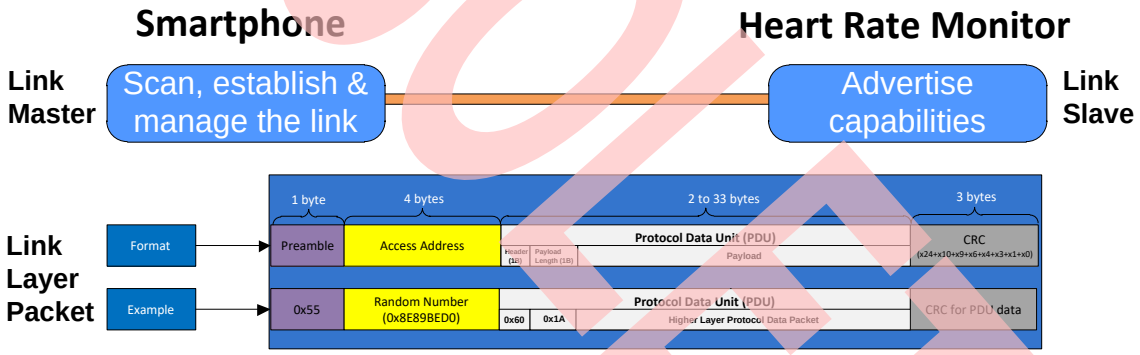
PSoC/PRoC BLE devices can operate in either configuration.

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 68). 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 68 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 68. BLE Link Layer Protocol



E.4 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.

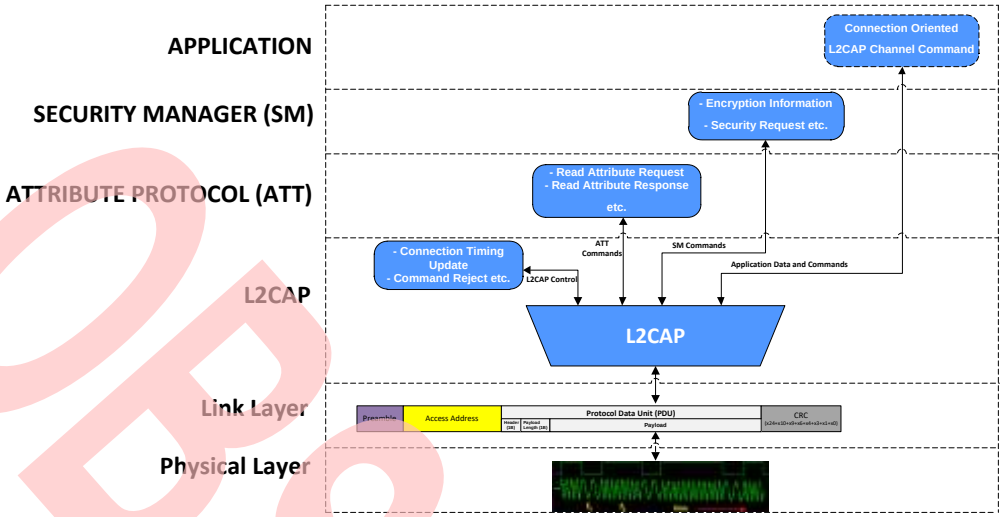
In PSoC/PRoC BLE devices, the HCI is just a firmware protocol layer that passes the messages and events between the controller and the host.

E.5 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 69. 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.

Figure 69. BLE L2CAP Layer



E.6 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.

E.7 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:

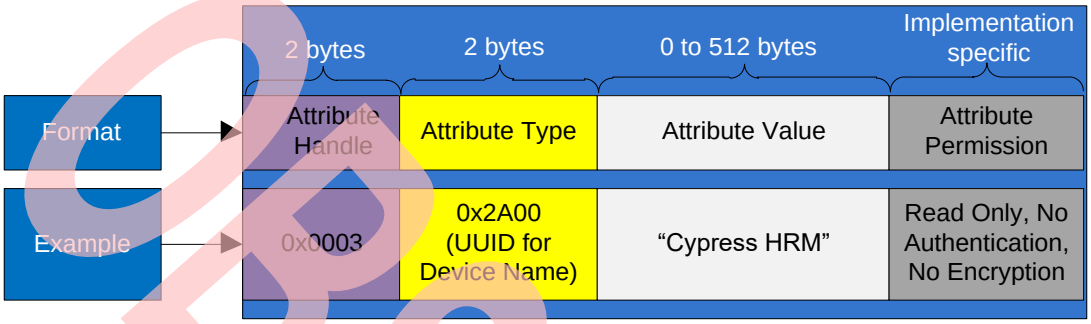
- **Attribute Handle:** The 16-bit address used to address and access an Attribute.
- **Attribute Type:** This specifies the type of data stored in an Attribute. It is represented by a 16-bit UUID defined by the Bluetooth SIG.

  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 70 shows the structure of a Device Name Attribute as an example.

Figure 70. Attribute Format Example



E.7.1 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:

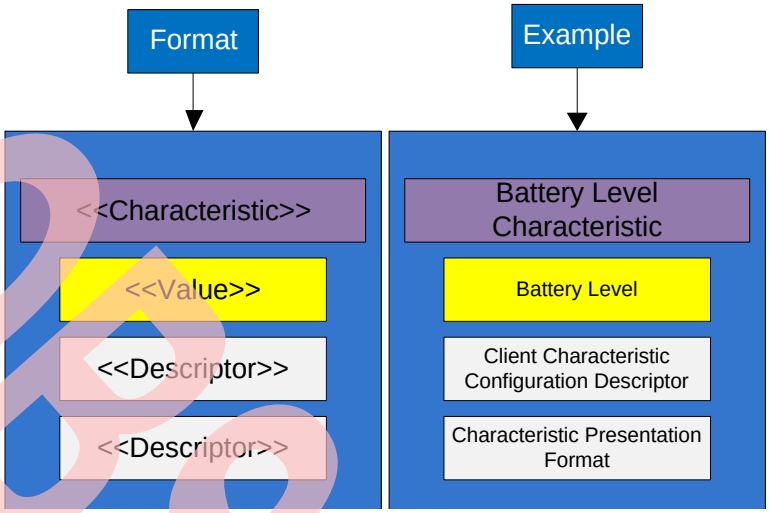
- **Characteristic:** A collection of Attributes that exposes the system information or meaningful data. A Characteristic consists of the following Attributes:
  - Characteristic Declaration Attribute: This defines the beginning of a Characteristic.
  - Characteristic Value Attribute: This holds the 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 71 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 71.
  - 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 71) 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 71. Characteristic Format and Example

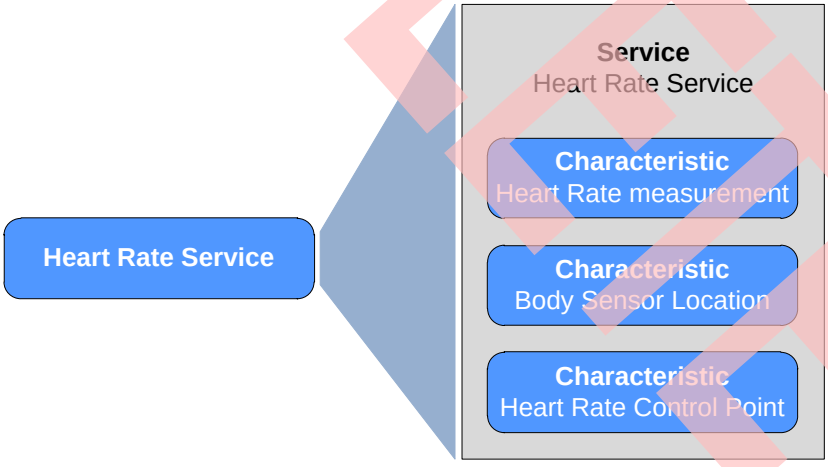


- **Service:** The type of Attribute that defines a function performed by the GATT server. A Service is a collection of Characteristics and can include other Services. The concept of a Service is used to establish the grouping of relative data and provide a data hierarchy. See Figure 72 for an example 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.

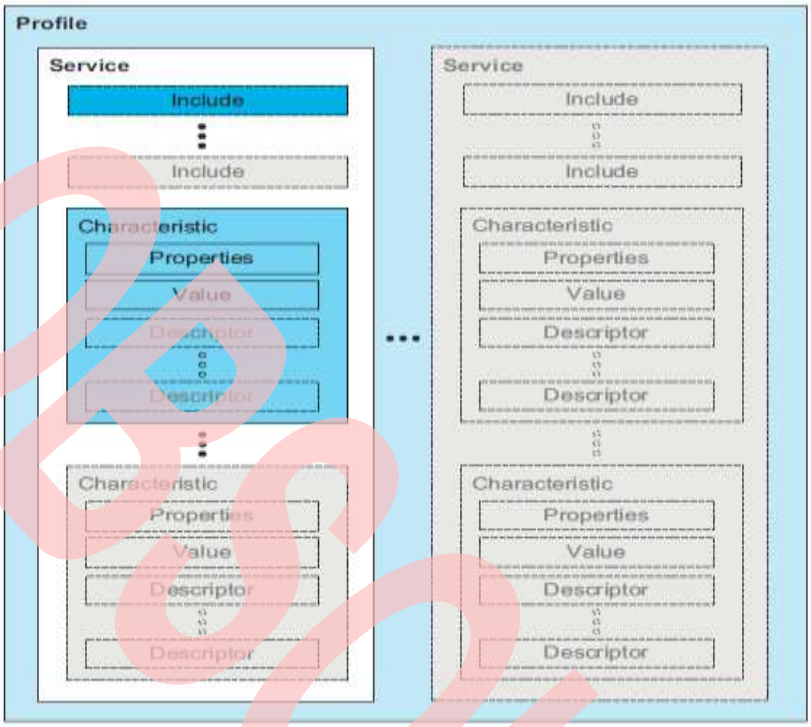
Figure 72. BLE Heart Rate Service Example



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 73 shows the data hierarchy using Attributes, Characteristics, Services, and Profiles defined previously in this section.

Figure 73. BLE Data Hierarchy*



* Image courtesy of Bluetooth SIG

E.7.2 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 71) 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.

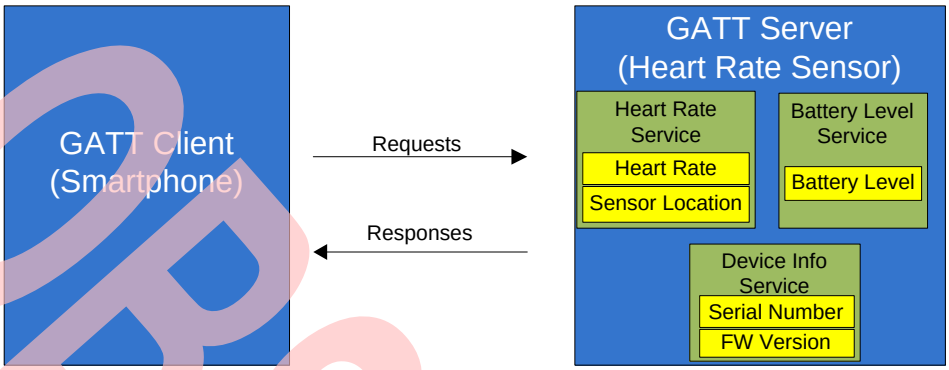
E.8 Generic Attribute Profile (GATT)

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 74 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 71.

Figure 74. 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.

E.9 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 76 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 75 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 75. BLE System Design

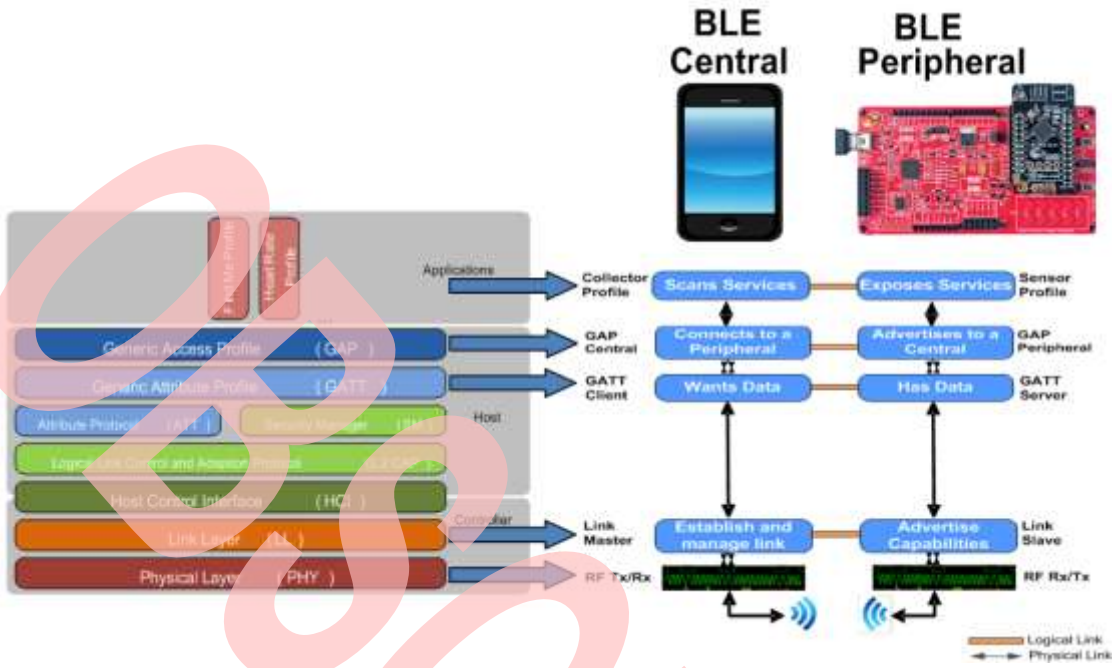
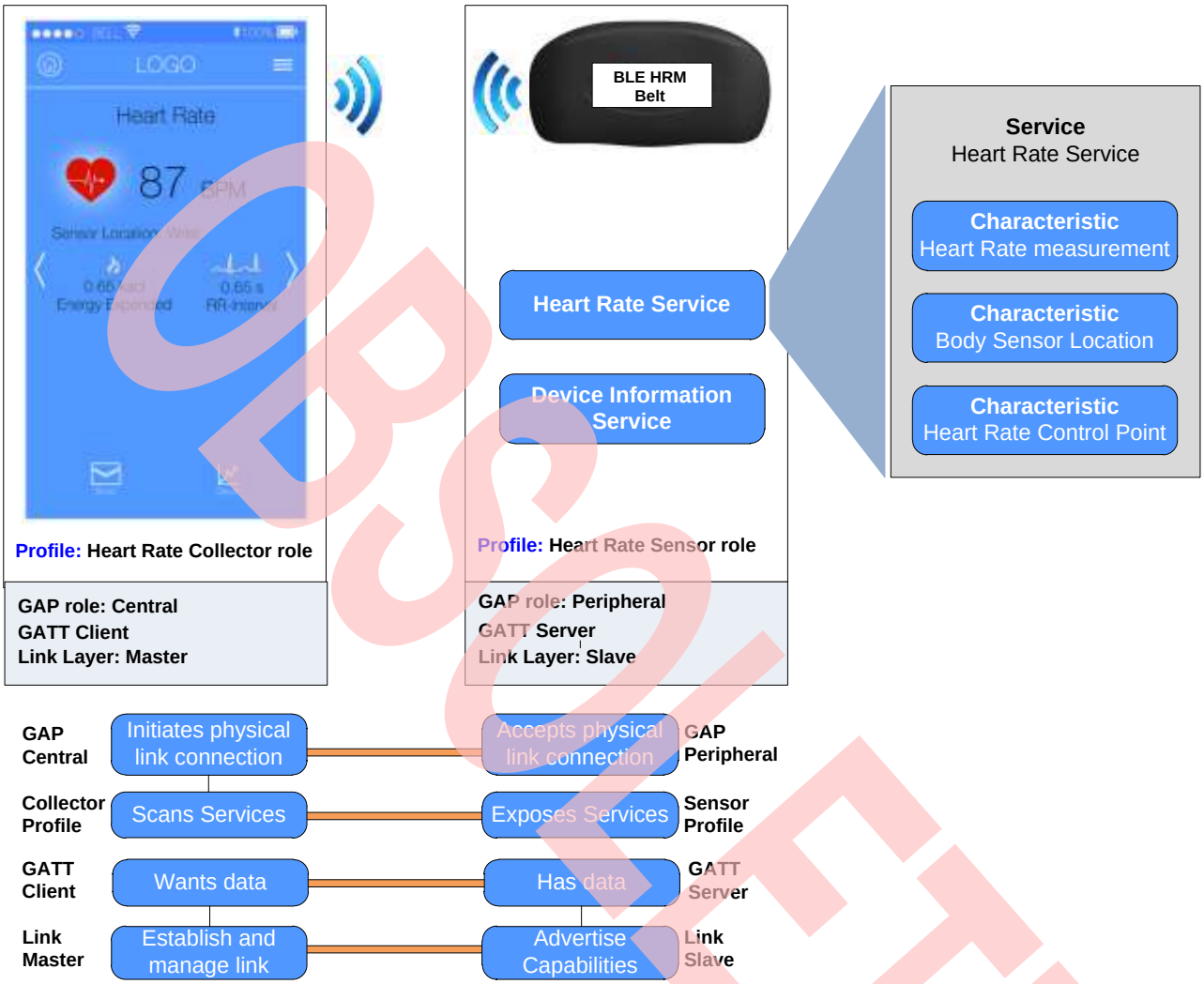


Figure 76 shows an example where a smartphone with a heart-rate app operates as a Central and a heart-rate sensor operates as a Peripheral. The heart-rate monitoring device implements the Heart-Rate Sensor Profile, while the smartphone receiving the data implements the Heart-Rate Collector Profile.

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 76. BLE Heart-Rate Monitor System



Document History

Document Title: AN94020 – Getting Started With PRoC™ BLE

Document Number: 001-94020

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	4566164	CSAI	11/10/2014	New Application Note
*A	4690155	CSAI	03/17/2015	Updated Table 1(changed CapSense value to “Yes” for CYBL10462-56LQXI).
*B	4770875	RAHU	06/19/2015	Document flow revised completely. Some sections moved to the Appendices. References added to CYBL10X7X device family and PSoC Creator 3.2. Updated screen shots as per PSoC Creator 3.2
*C	4992030	SKUV	10/28/2015	Updated multiple sections to incorporate customer feedback Updated screen shots based on PSoC Creator 3.3 and CySmart 1.1 Modified part numbers to CYBL1XX7X and CYBL10X6X
*D	5005641	SKUV	11/23/2015	Added Bluetooth 4.2 feature details Updated support for PSoC Creator 3.3 SP1 and CySmart 1.2
*E	5930488	ANKC	16/10/2017	Obsoleting the spec

Worldwide Sales and Design Support

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

Products

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

PSoC® Solutions

psoc.cypress.com/solutions

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

Cypress Developer Community

Community | Forums | Blogs | Video | Training

Technical Support

cypress.com/go/support

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

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