# Getting Started with PSoC® 4 BLE

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

**Associated PSoC Creator™ Project: Yes**

**Associated Part Family: CY8C41x7-BL, CY8C42x7-BL, CY8C41x8-BL, CY8C42x8-BL**

**Software Version: PSoC Creator 3.3 SP1 or later**

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

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

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

## Contents

# 1 Introduction

The Cypress PSoC 4 BLE device is a programmable embedded system-on-chip that integrates a Bluetooth Low Energy (BLE) radio, programmable analog and digital peripherals, memory, and an ARM Cortex-M0 microcontroller on a single chip. BLE is an ultra-low-power wireless standard defined by the Bluetooth Special Interest Group (SIG) for short-range communication. It features a physical layer, protocol stack, and application use cases, all designed and optimized for low power consumption.

PSoC 4 BLE provides a cost-effective and small footprint alternative to the combination of an MCU and a BLE radio. The programmable analog and digital subsystems allow flexibility and dynamic fine-tuning of the design using PSoC Creator, the schematic-based design tool for developing PSoC 4 BLE applications. To develop a BLE application, you do not need a working knowledge of BLE complex protocol stack. Cypress provides an easy-to-configure, free-of-cost graphical user interface (GUI) based BLE Component in PSoC Creator that abstracts the protocol complexity. BLE Component can be configured in minutes using a GUI, enabling you to jump-start your BLE design. PSoC 4 BLE offers a current consumption of 150 nA while retaining the SRAM contents, programmable logic, and the ability to wake up from an interrupt. It consumes 1.3 µA in Deep-Sleep mode while maintaining an active BLE link. A combination of these low-power modes provides a best-in-class system power consumption for battery-operated BLE designs such as wearable fitness monitors and wireless sensor interfaces.

PSoC 4 BLE simplifies the RF board design with its integrated balun circuit, which reduces the number of external components required for antenna matching.

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

This application note presents the basics of BLE that includes capabilities of PSoC 4 BLE device and overview of development tools. For advanced application development, refer to Designing BLE Applications and Creating a BLE Custom Profile application note.

## 1.1 Prerequisites

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

### 1.1.1 Hardware

- BLE Pioneer Kit
- PC/Laptop with Windows 7 or later (If using the CySmart Host Emulation Tool PC application)
- Mobile phone with Android 5 or later or iOS 8 or later (If using the CySmart iOS/Android app)

### 1.1.2 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 PSoC 4 BLE Resources

Cypress provides a wealth of data at www.cypress.com to help you to select the right PSoC device and quickly and effectively integrate it into your design. If you are new to PSoC, 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 PSoC 4 BLE:

- **Overview:** PSoC Portfolio, PSoC RF Roadmap
- **Product Selectors:** PSoC 1, PSoC 3, PSoC 4, or PSoC 5LP. In addition, PSoC Creator includes a device selection tool.
- **Datasheets** describe and provide electrical specifications for the PSoC 4XX7-BL and PSoC 4XX8-BL device families.
- **Application Notes and Code Examples** cover a broad range of topics, from basic to advanced level. Many of the application notes include code examples. PSoC Creator provides additional code examples—see Code Examples.
- **Technical Reference Manuals (TRMs)** provide detailed descriptions of the architecture and registers in each PSoC 4 BLE device family.
- **CapSense Design Guide:** Learn how to design capacitive touch-sensing applications with the PSoC 4 BLE family of devices.
- **Development Tools**
  - CY8CKIT-042-BLE Bluetooth Low Energy (BLE) Pioneer Kit is an easy-to-use and inexpensive development platform for BLE. This kit includes connectors for Arduino™ compatible shields and Digilent® Pmod™ daughter cards.
  - CySmart BLE Host Emulation Tool for Windows, iOS, and Android is an easy-to-use app that enables you to test and debug your BLE Peripheral applications.

  See Appendix C: Cypress BLE Development Tools for an overview.
- **Training Videos:** A video is worth a million words—here's a suggested list of PSoC 4 BLE videos to get you started with your first BLE design:
  - Introduction to PSoC 4 BLE
  - Getting to know PSoC Creator
  - BLE Pioneer Kit
  - Introduction to BLE
  - Configuring your first BLE design
  - Writing firmware for your first BLE design

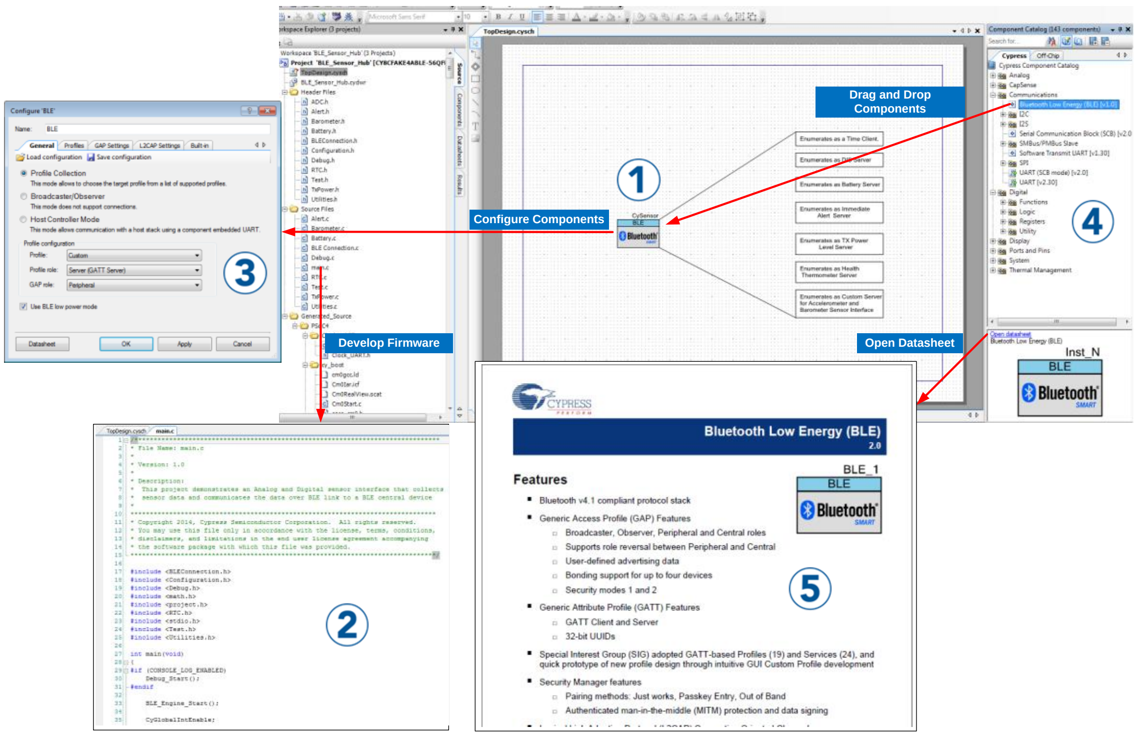
  Take a look at PSoC Creator and PSoC 4 BLE training video modules for more videos.
- **Technical Support**
  - Frequently Asked Questions (FAQs): Learn more about our BLE ecosystem
  - Forum: See if your question is already answered by fellow developers on the PSoC 4 BLE and PRoC BLE forums.
  - Cypress 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 PSoC 4 BLE and PRoC BLE. As Figure 1 shows, with PSoC Creator, you can:

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

Figure 1. PSoC Creator Schematic Entry and Components



Configure 'BLE'

Name: BLE

General | Profiles | GAP Settings | L2CAP Settings | Built-in

Load configuration | Save configuration

(●) Profile Collection
This mode allows to choose the target profile from a list of supported profiles.

( ) Broadcaster/Observer
This mode does not support connections.

( ) Host Controller Mode
This mode allows communication with a host stack using a component embedded UART.

Profile configuration
Profile: Custom
Profile role: Server (GATT Server)
GAP role: Peripheral

[✓] Use BLE low power mode

Datasheet | OK | Apply | Cancel

```
/*****************************************************************************
* File Name: main.c
*
* Version: 1.0
*
* Description:
*  This project demonstrates an Analog and Digital sensor interface that collects
*  sensor data and communicates the data over BLE link to a BLE central device
*
*****************************************************************************
* Copyright 2014, Cypress Semiconductor Corporation.  All rights reserved.
* You may use this file only in accordance with the license, terms, conditions,
* disclaimers, and limitations in the end user license agreement accompanying
* the software package with which this file was provided.
*****************************************************************************/

#include <BLEConnection.h>
#include <Configuration.h>
#include <Debug.h>
#include <math.h>
#include <project.h>
#include <RTC.h>
#include <stdio.h>
#include <Test.h>
#include <Utilities.h>

int main(void)
{
#if (CONSOLE_LOG_ENABLED)
    Debug_Start();
#endif

    BLE_Engine_Start();

    CyGlobalIntEnable;
```

Bluetooth Low Energy (BLE)
2.0

BLE_1

Features

- Bluetooth v4.1 compliant protocol stack
- Generic Access Profile (GAP) Features
  - Broadcaster, Observer, Peripheral and Central roles
  - Supports role reversal between Peripheral and Central
  - User-defined advertising data
  - Bonding support for up to four devices
  - Security modes 1 and 2
- Generic Attribute Profile (GATT) Features
  - GATT Client and Server
  - 32-bit UUIDs
- Special Interest Group (SIG) adopted GATT-based Profiles (19) and Services (24), and quick prototype of new profile design through intuitive GUI Custom Profile development
- Security Manager features
  - Pairing methods: Just works, Passkey Entry, Out of Band
  - Authenticated man-in-the-middle (MITM) protection and data signing

## 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:

- **Quick Start Guide**: Choose **Help** > **Documentation** > **Quick Start Guide**. This guide gives you the basics for developing PSoC Creator projects.
- **Simple Component Code Examples**: Choose **File** > **Code Example**. These code examples demonstrate how to configure and use PSoC Creator Components.
- **System Reference Guide**: Choose **Help** > **System Reference Guides**. This guide lists and describes the system functions provided by PSoC Creator.
- **Component Datasheets**: Right-click a Component and select “Open Datasheet.” Visit the PSoC 4 BLE Component Datasheets page for a list of all PSoC 4 BLE Component datasheets.
- **Document Manager**: PSoC Creator provides a document manager to help you to easily find and review document resources. To open the document manager, choose the menu item **Help** > **Document Manager**.

## 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.

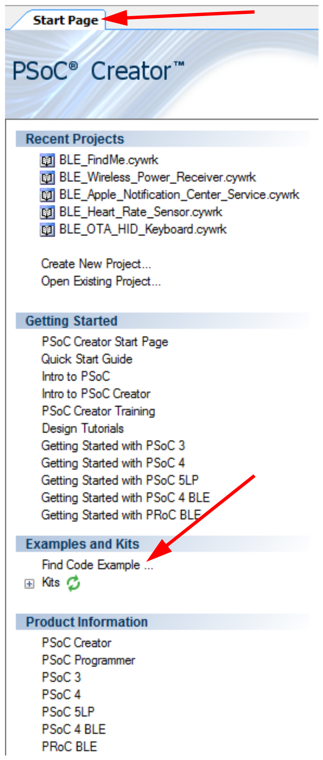
Code examples can speed up your design process by starting you off with a complete design, instead of a blank page. The code examples also show how PSoC Creator Components can be used for various applications. Code examples and datasheets are included, as Figure 3 shows.

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



Start Page

PSoC® Creator™

Recent Projects
- BLE_FindMe.cywrk
- BLE_Wireless_Power_Receiver.cywrk
- BLE_Apple_Notification_Center_Service.cywrk
- BLE_Heart_Rate_Sensor.cywrk
- BLE_OTA_HID_Keyboard.cywrk

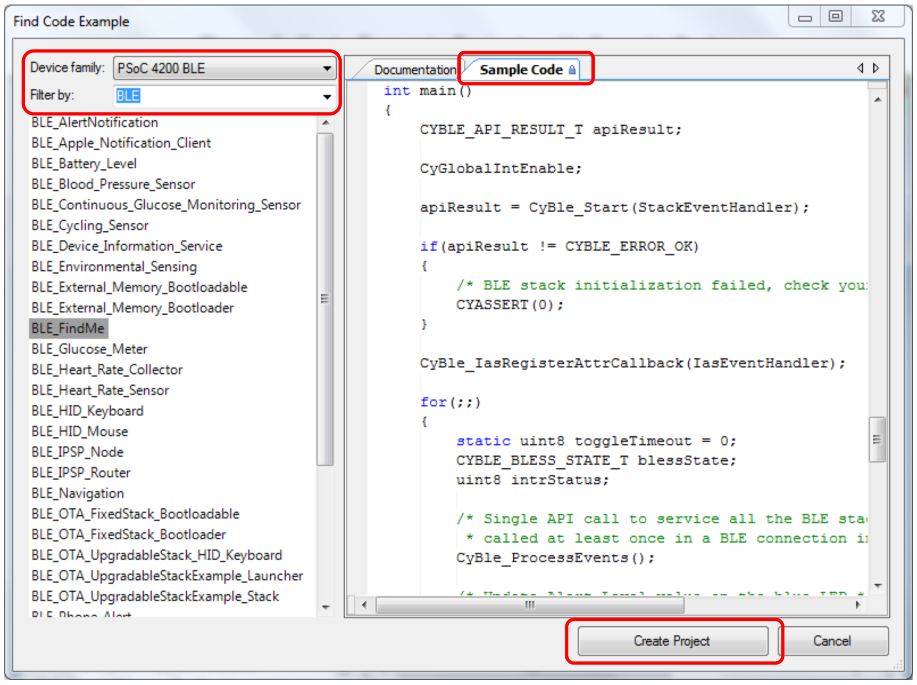
Create New Project...
Open Existing Project...

Getting Started
- PSoC Creator Start Page
- Quick Start Guide
- Intro to PSoC
- Intro to PSoC Creator
- PSoC Creator Training
- Design Tutorials
- Getting Started with PSoC 3
- Getting Started with PSoC 4
- Getting Started with PSoC 5LP
- Getting Started with PSoC 4 BLE
- Getting Started with PRoC BLE

Examples and Kits
- Find Code Example ...
- Kits

Product Information
- PSoC Creator
- PSoC Programmer
- PSoC 3
- PSoC 4
- PSoC 5LP
- PSoC 4 BLE
- PRoC BLE

Figure 3. Code Examples with Sample Code



Find Code Example

Device family: PSoC 4200 BLE
Filter by: BLE

- BLE_AlertNotification
- BLE_Apple_Notification_Client
- BLE_Battery_Level
- BLE_Blood_Pressure_Sensor
- BLE_Continuous_Glucose_Monitoring_Sensor
- BLE_Cycling_Sensor
- BLE_Device_Information_Service
- BLE_Environmental_Sensing
- BLE_External_Memory_Bootloadable
- BLE_External_Memory_Bootloader
- BLE_FindMe
- BLE_Glucose_Meter
- BLE_Heart_Rate_Collector
- BLE_Heart_Rate_Sensor
- BLE_HID_Keyboard
- BLE_HID_Mouse
- BLE_IPSP_Node
- BLE_IPSP_Router
- BLE_Navigation
- BLE_OTA_FixedStack_Bootloadable
- BLE_OTA_FixedStack_Bootloader
- BLE_OTA_UpgradableStack_HID_Keyboard
- BLE_OTA_UpgradableStackExample_Launcher
- BLE_OTA_UpgradableStackExample_Stack

Documentation | Sample Code

```
int main()
{
    CYBLE_API_RESULT_T apiResult;

    CyGlobalIntEnable;

    apiResult = CyBle_Start(StackEventHandler);

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

    CyBle_IasRegisterAttrCallback(IasEventHandler);

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

        /* Single API call to service all the BLE sta
         * called at least once in a BLE connection i
        CyBle_ProcessEvents();
```

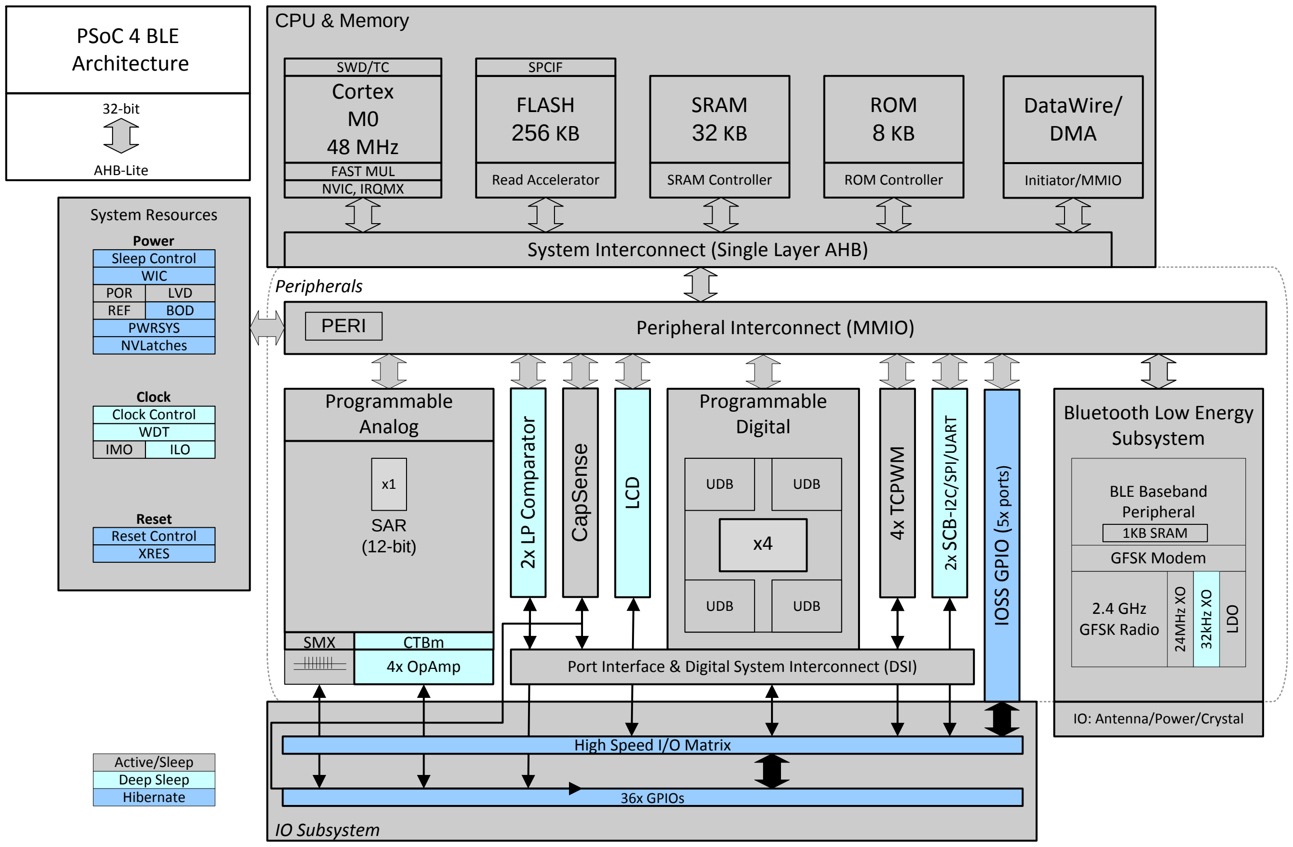
Create Project | Cancel

## 3 PSoC 4 BLE Features

As shown in Figure 4, PSoC 4 BLE device features:

- 32-bit MCU subsystem
  - 48-MHz ARM® Cortex™-M0 CPU
  - Up to 256 KB flash and 32 KB SRAM
- CapSense® with SmartSense™ Auto-tuning
  - One Cypress Capacitive Sigma-Delta™ (CSD) controller with touchpad capability
- Programmable analog front end (AFE)
  - Four opamps, configurable as PGAs, comparators, filters, etc.
  - One 12-bit, 1-Msps SAR2 ADC
- Programmable digital logic
  - Four Universal Digital Blocks (UDBs): custom digital peripherals
  - Four configurable TCPWM blocks: 16-bit timer, counter or PWM
  - Two configurable serial communication blocks (SCBs): I2C master or slave, SPI master or slave, or UART
- Bluetooth Smart connectivity with Bluetooth 4.2
  - 2.4-GHz BLE radio with integrated Balun
  - Bluetooth 4.2 specification-compliant controller and host implementation

Figure 4. PSoC 4 BLE Architecture (CY8C4248-BL)



See Appendix D: PSoC 4 BLE Device for a brief description of the features. For in-depth information, see the PSoC 4 BLE family datasheet, TRM, and application notes. Figure 4 shows the features available in the CY8C4248-BL family of devices. Subsets of these features are available in other device families; see Table 2.

## 4 PSoC is More Than a BLE MCU

Figure 5 shows that a typical MCU contains a CPU (such as 8051 or an ARM Cortex) with a set of peripheral functions such as ADCs, DACs, UARTs, SPIs, BLE, and general I/O, all linked to the CPU's register interface. Within the MCU, the CPU is the "heart" of the device – the CPU manages everything from setup to data movement to timing. Without the CPU, the MCU cannot function.

Figure 6 shows that PSoC is quite different. With PSoC, the CPU, analog, digital, and I/O are equally important resources in a programmable system. *It is the system's interconnect and programmability that is the heart of PSoC – not the CPU.* The peripheral analog and digital are interconnected with a highly configurable matrix of signal and data bus meshing that allows you to create custom designs that meet your application requirements. *You can program PSoC to emulate an MCU, but you cannot program an MCU to emulate PSoC.*

Figure 5. Block Diagram of a Typical BLE MCU

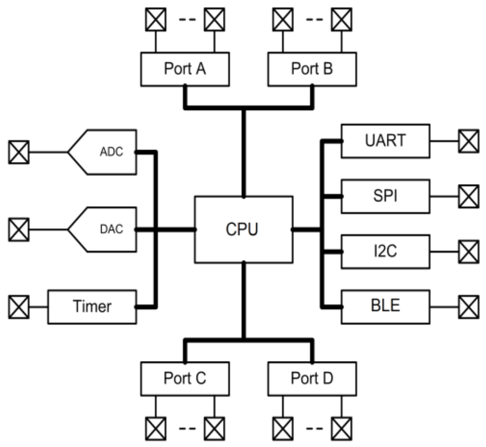
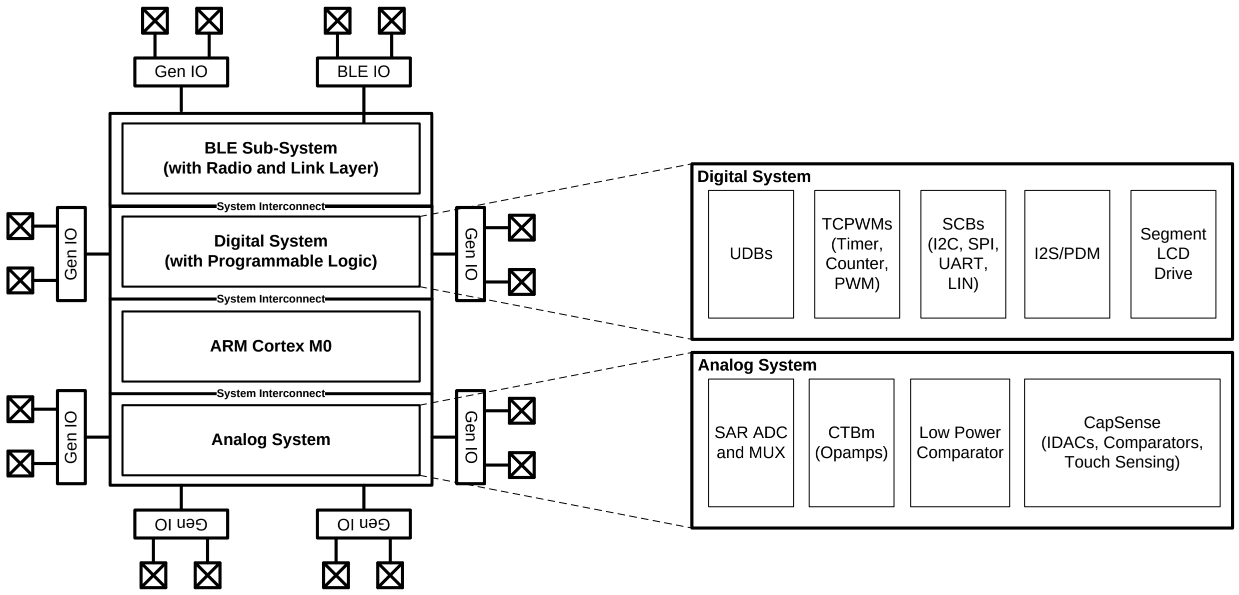


Figure 6. PSoC 4 BLE Block Diagram



A typical MCU requires CPU firmware to process state machines, use a timer for timing, and drive an output pin. Thus, the functional path is almost always through the CPU. However, with PSoC, asynchronous parallel processing is possible. You can configure a PSoC to have elements that operate independently from the CPU. PSoC Creator makes the interface between the analog, digital and programmable interconnect blocks seamless and all the analog and digital peripherals are provided in the form of virtual ICs called Components in PSoC Creator. The example project in this application note will describe how to use PSoC Creator and its Components.

The PSoC also has programmable digital blocks known as Universal Digital Blocks (UDBs). PSoC Creator also provides several Components made out of UDBs, such as UART, SPI, I2S, Timer, PWM, Counter, Digital Gates (AND, OR, NOT, XOR, and so on), and many more. You can create custom state machines and digital logic using the UDBs in PSoC Creator. The method to create custom PSoC Creator Components is provided in the PSoC Creator Component Author Guide.

## 5 BLE Overview

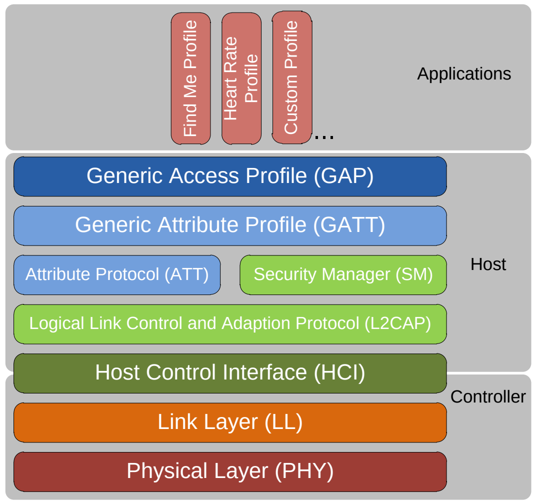
BLE or Bluetooth Smart$^{TM}$ is a low-power, short-range, low-data-rate wireless communication protocol that is defined by the Bluetooth SIG. As shown in Figure 7, BLE has a layered protocol stack that is designed to efficiently transfer a small amount of data with low power consumption, making it the 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 determines whether 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 7. BLE Protocol Stack



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 the following:

- BLE link establishment procedure
- Application data representation and abstraction
- Mapping of application requirements to BLE GAP and GATT layer configurations.

## 5.1 BLE Link Establishment

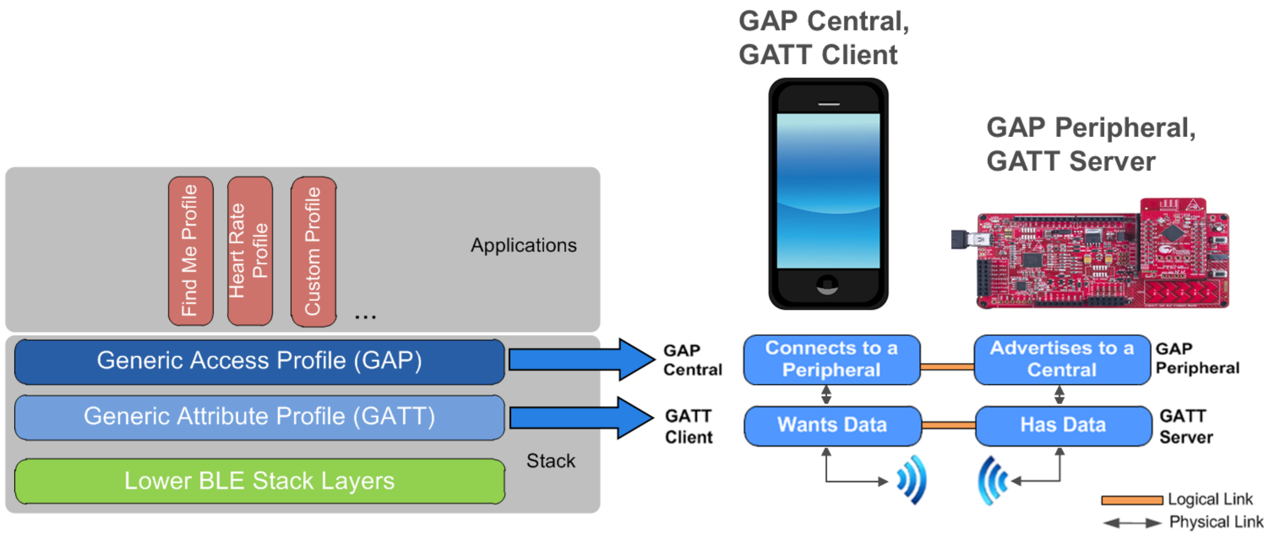
To establish a BLE link between two devices—the Cypress BLE Pioneer Kit and a smartphone, for example—you need to understand the two Generic Access Profile (GAP) device roles as Figure 8 shows:

- **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 as Figure 8 shows:

- **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. Similarly, a smart bulb is a GATT server that contains the state of the bulb (on/off) that can be configured by a smart switch 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.

Figure 8. BLE Application Overview



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

## 5.2 GATT Data Format

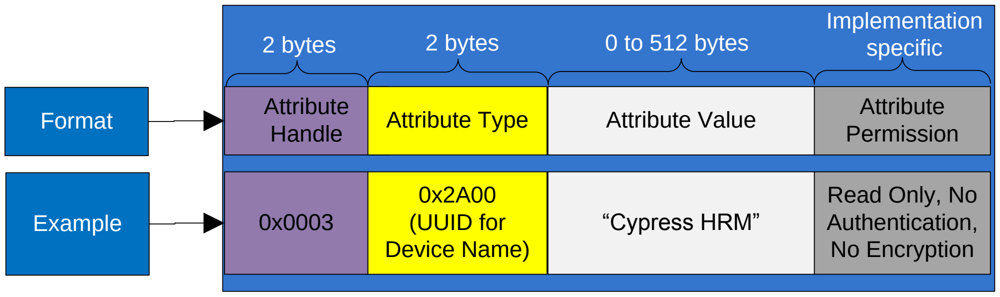
After understanding the BLE GAP/GATT roles, next step is to understand how the data is stored on a GATT server and how it is retrieved by a GATT client. A GATT server uses Attributes, Characteristics, and Services to represent and abstract data in a BLE device. As you will see in this section, a Service 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 9.
  - Attribute Handle: Used to address the Attribute
  - Attribute Type: A 16-bit Universally Unique Identifier (UUID) assigned by the Bluetooth SIG that specifies the data contained in the Attribute
  - Attribute Value: Contains the actual data
  - Attribute Permission: Specifies read/write permissions 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 “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; other details are abstracted by Cypress’s BLE Component API functions.

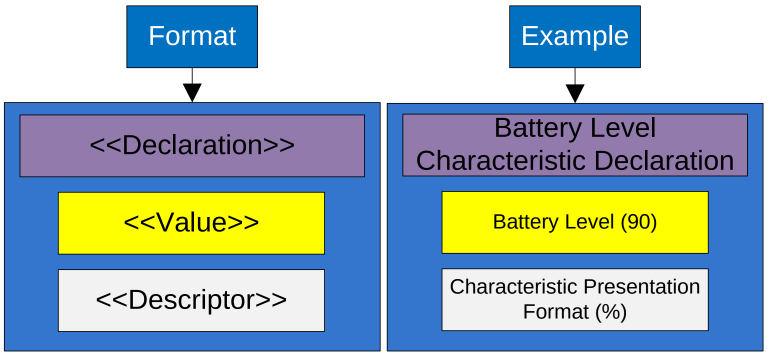
Figure 9. GATT Attribute Example



- **Characteristic:** A Characteristic is composed of multiple discrete Attributes that when combined, define the 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 10. For example, 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. In addition, you can define your own custom Characteristics. On/off state of a smart bulb is an example for a custom Characteristic.

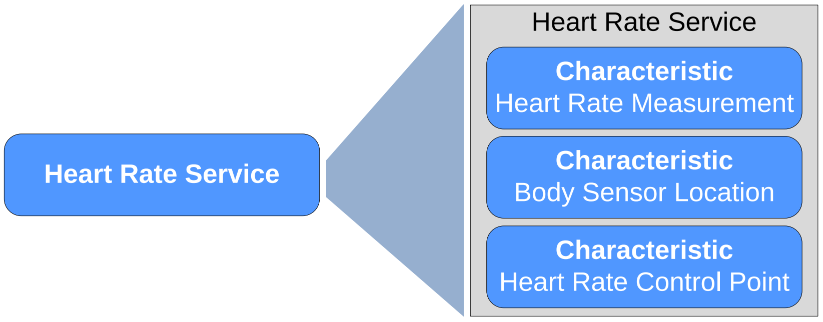
Figure 10. GATT Characteristic Example


Format
Example
<<Declaration>>
<<Value>>
<<Descriptor>>
Battery Level Characteristic Declaration
Battery Level (90)
Characteristic Presentation Format (%)

- **Service:** A Service is composed of one or more related Characteristics that define a particular function or feature of a device. Figure 11 shows an example (Heart Rate Service) that has three Characteristics describing the information related to measuring the heart-rate.

The Bluetooth SIG offers a set of predefined standard Services for implementing commonly used BLE device functionalities. In addition, 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 11. GATT Service Example

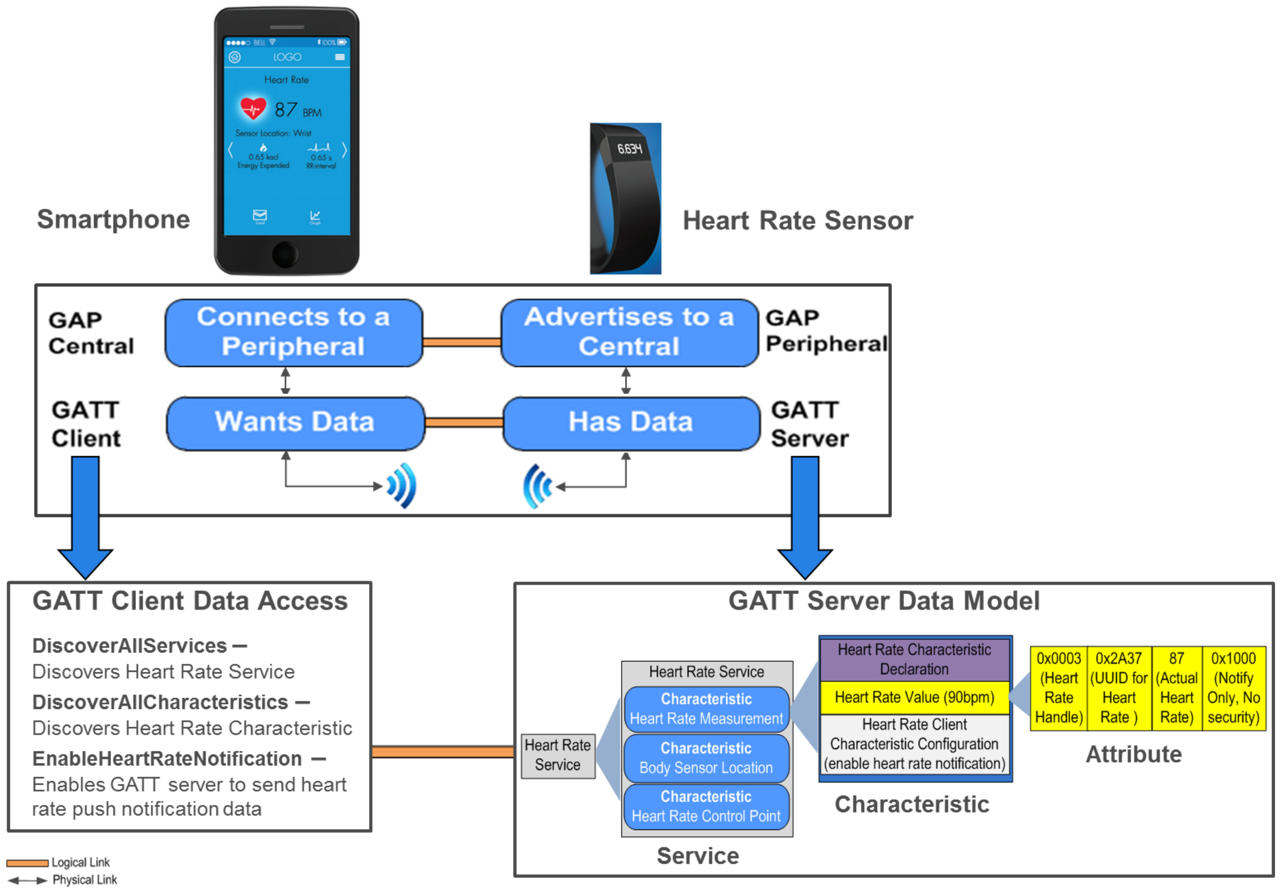


In addition to defining the data format, GATT layer also defines a set of procedures or methods to discover and access data in the GATT layer. After establishing a BLE connection, a GATT client uses these GATT procedures to discover the complete attribute database of the connected GATT server. After 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 12 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 12. GATT Data Format


Smartphone
Heart Rate Sensor
GAP Central
Connects to a Peripheral
Advertises to a Central
GAP Peripheral
GATT Client
Wants Data
Has Data
GATT Server
GATT Client Data Access
DiscoverAllServices – Discovers Heart Rate Service
DiscoverAllCharacteristics – Discovers Heart Rate Characteristic
EnableHeartRateNotification – Enables GATT server to send heart rate push notification data
GATT Server Data Model
Heart Rate Service
Heart Rate Service
Characteristic Heart Rate Measurement
Characteristic Body Sensor Location
Characteristic Heart Rate Control Point
Service
Heart Rate Characteristic Declaration
Heart Rate Value (90bpm)
Heart Rate Client Characteristic Configuration (enable heart rate notification)
Characteristic
0x0003 (Heart Rate Handle)
0x2A37 (UUID for Heart Rate )
87 (Actual Heart Rate)
0x1000 (Notify Only, No security)
Attribute
Logical Link
Physical Link

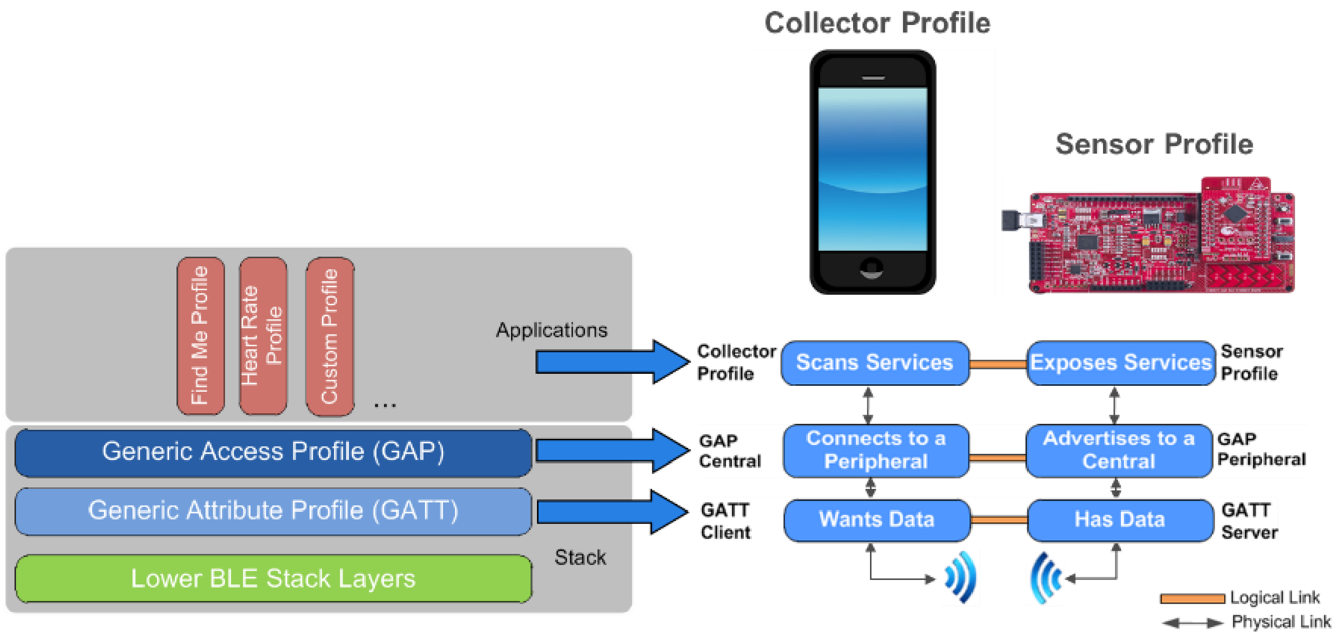
## 5.3 BLE Profile

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, in the case of a heart-rate monitoring device, the BLE Heart Rate Profile defines the required GAP, 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. In addition, you can create your own custom Profile that consists of standard or custom Services.

As shown in Figure 13, similar to the GATT layer, the Profile defines two application roles:

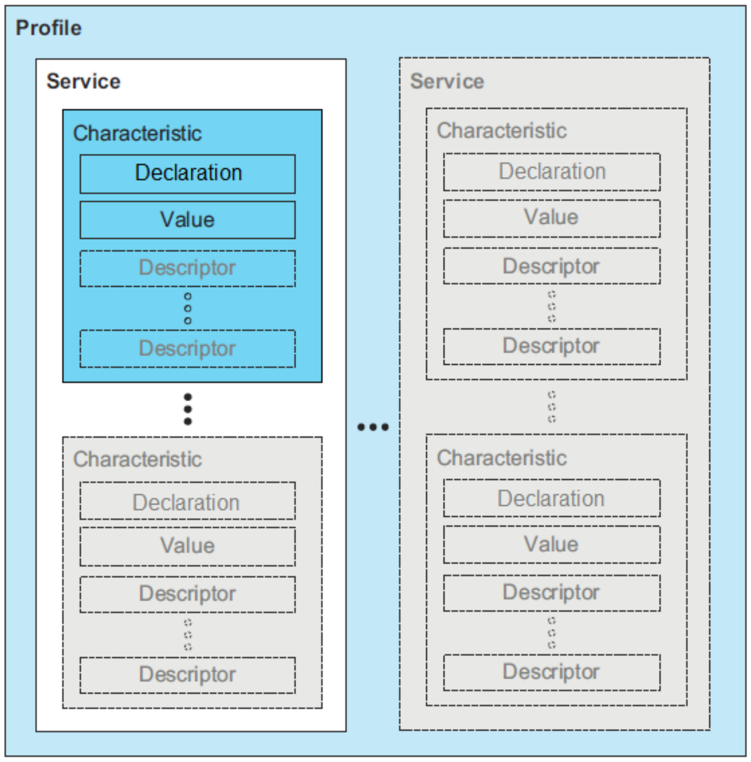
- **Sensor or server:** The Sensor Profile role is supported by the application that has data. The Sensor Profile specification defines the required roles (for example, GATT/GAP roles) and behavior (for example, advertisement interval, GATT Services to be supported) of the BLE device to support a Sensor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

Figure 13. BLE Profile Example



A summary of the data abstraction and hierarchy in a BLE device is shown in Figure 14.

Figure 14. BLE Data Hierarchy*



* Image courtesy of Bluetooth SIG

## 5.4 BLE Component

### 5.4.1 Features

The BLE Component in PSoC Creator abstracts the BLE protocol into a simple and easy-to-use GUI and an API with just a few functions. BLE Component features:

- Bluetooth 4.2-compliant protocol stack
- Supports all the GAP roles – Central, Peripheral, Broadcaster, and Observer. A limited simultaneous combination of GAP roles such as Central and Observer, Peripheral and Broadcaster are 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 4 bonded devices and 8 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

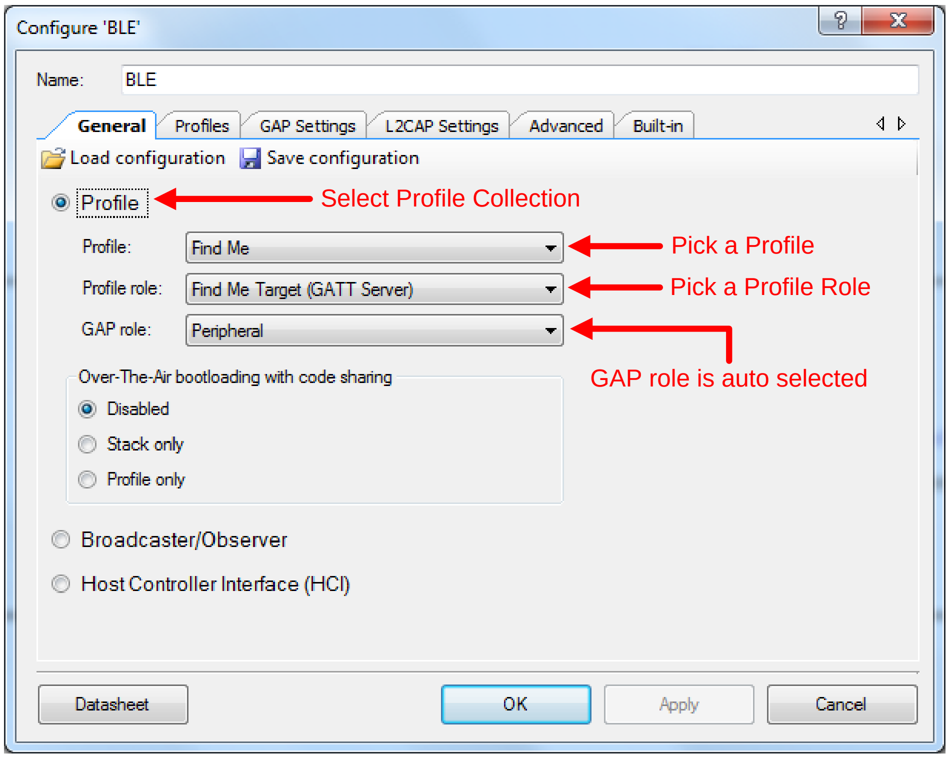
See the BLE Component datasheet for details.

### 5.4.2 Configuration

It takes five steps to create a Bluetooth SIG defined Profile (standard Profile) based application using the BLE component. For example, to create a Find Me Target example described in My First PSoC 4 BLE Design section, follow these steps:

**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 Target role shown in Figure 15, the GAP role is set to Peripheral and the GATT role is set to server as per the Find Me Profile specification.

Figure 15. BLE Component Profile Configuration


Configure 'BLE'

Name: BLE

[✓ Profile]

Profile: Find Me

Profile role: Find Me Target (GATT Server)

GAP role: Peripheral

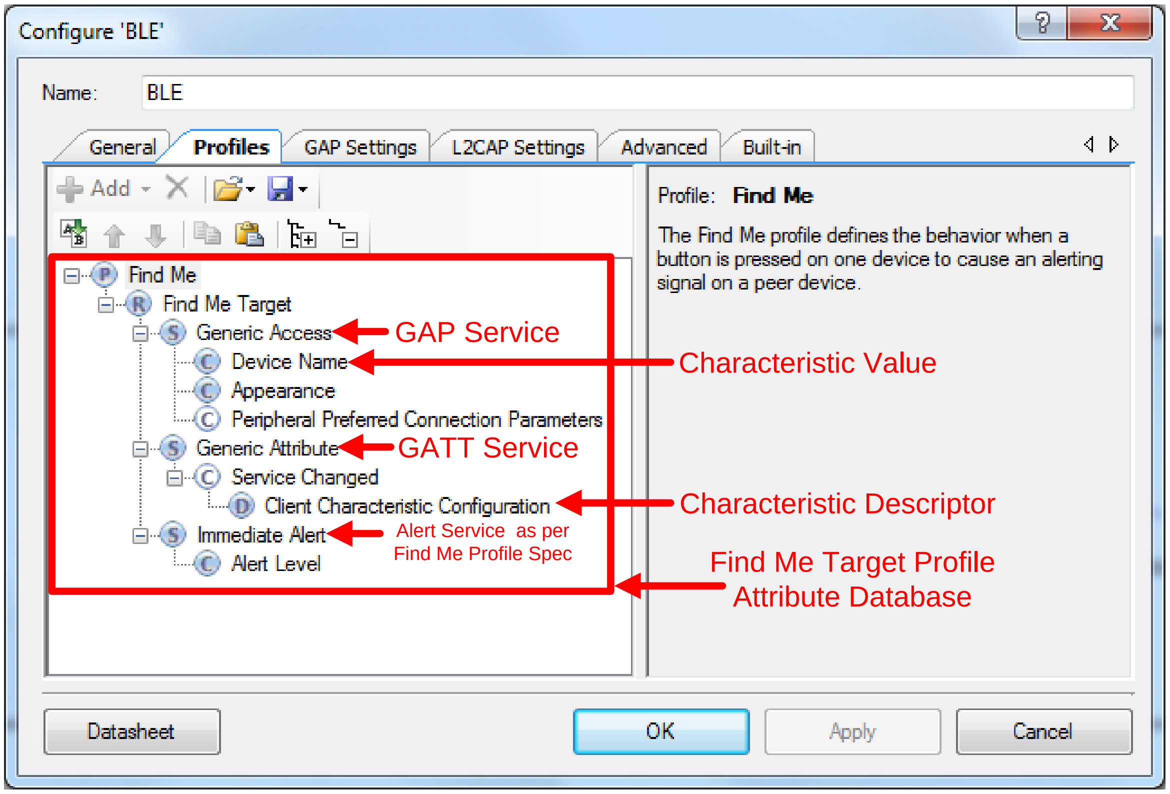
Over-The-Air bootloading with code sharing: [✓ Disabled] Stack only Profile only

Broadcaster/Observer

Host Controller Interface (HCI)

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

Figure 16. BLE Attribute Database Configuration

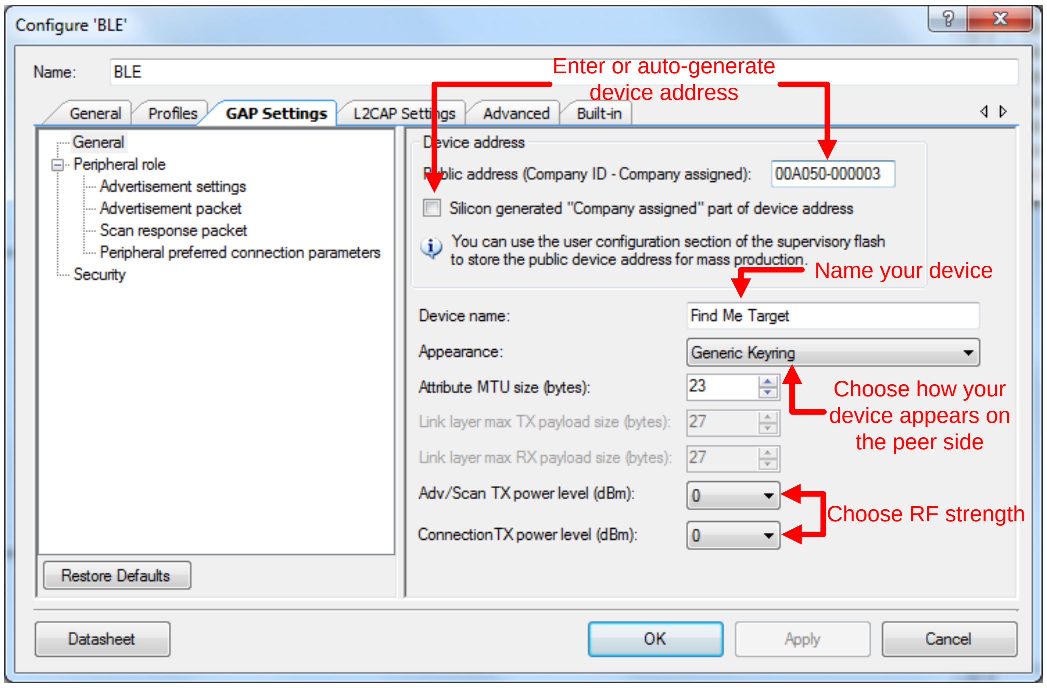


Profile: **Find Me**

The Find Me profile defines the behavior when a button is pressed on one device to cause an alerting signal on a peer device.

**Step 3:** Configure the GAP general and advertisement data settings for your design as explained at the top of page 9, as shown in Figure 17 and Figure 18.

Figure 17. BLE GAP Configuration


Enter or auto-generate device address
Name your device
Choose how your device appears on the peer side
Choose RF strength

Configure 'BLE'

Name: BLE

General | Profiles | **GAP Settings** | L2CAP Settings | Advanced | Built-in

- General
- Peripheral role
  - Advertisement settings
  - Advertisement packet
  - Scan response packet
  - Peripheral preferred connection parameters
- Security

Device address

Public address (Company ID - Company assigned): 00A050-000003

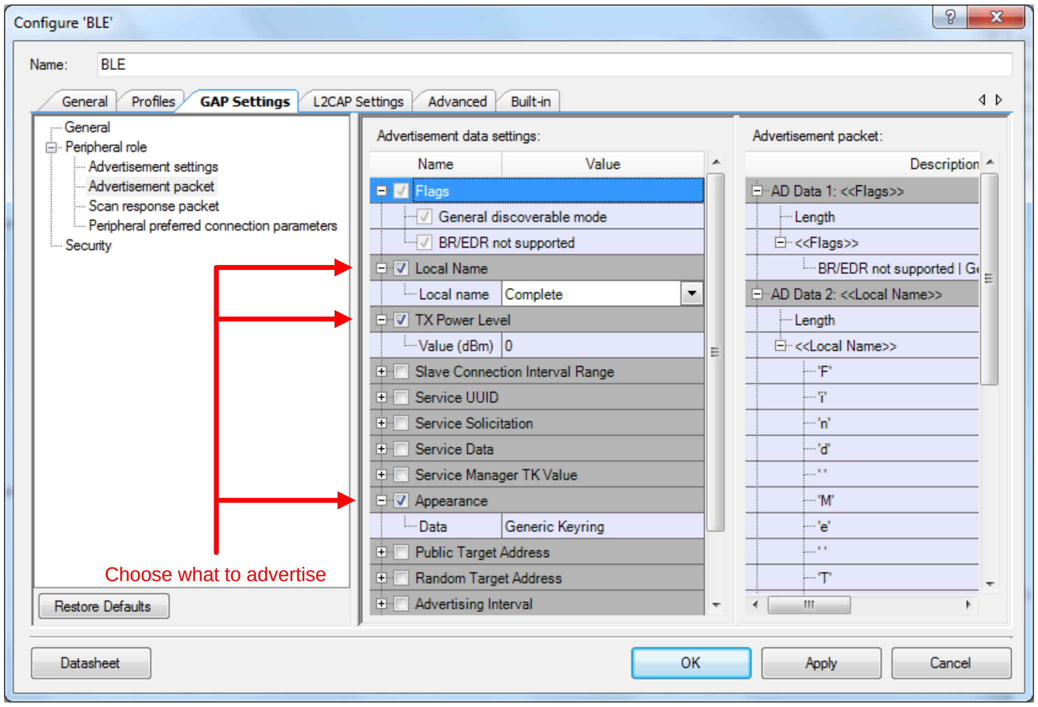
☐ Silicon generated "Company assigned" part of device address

You can use the user configuration section of the supervisory flash to store the public device address for mass production.

| | |
|---|---|
| Device name: | Find Me Target |
| Appearance: | Generic Keyring |
| Attribute MTU size (bytes): | 23 |
| Link layer max TX payload size (bytes): | 27 |
| Link layer max RX payload size (bytes): | 27 |
| Adv/Scan TX power level (dBm): | 0 |
| Connection TX power level (dBm): | 0 |

Restore Defaults | Datasheet | OK | Apply | Cancel

Figure 18. BLE Advertisement Data Configuration



Configure 'BLE'

Name: BLE

General | Profiles | **GAP Settings** | L2CAP Settings | Advanced | Built-in

- General
- Peripheral role
  - Advertisement settings
  - Advertisement packet
  - Scan response packet
  - Peripheral preferred connection parameters
- Security

Advertisement data settings:

| Name | Value |
|---|---|
| ☑ Flags | |
| ☑ General discoverable mode | |
| ☑ BR/EDR not supported | |
| ☑ Local Name | |
| Local name | Complete |
| ☑ TX Power Level | |
| Value (dBm) | 0 |
| ☐ Slave Connection Interval Range | |
| ☐ Service UUID | |
| ☐ Service Solicitation | |
| ☐ Service Data | |
| ☐ Service Manager TK Value | |
| ☑ Appearance | |
| Data | Generic Keyring |
| ☐ Public Target Address | |
| ☐ Random Target Address | |
| ☐ Advertising Interval | |

Advertisement packet:

| Description |
|---|
| AD Data 1: <<Flags>> |
| Length |
| <<Flags>> |
| BR/EDR not supported \| G |
| AD Data 2: <<Local Name>> |
| Length |
| <<Local Name>> |
| 'F' |
| 'i' |
| 'n' |
| 'd' |
| ' ' |
| 'M' |
| 'e' |
| ' ' |
| 'T' |

Restore Defaults | Datasheet | OK | Apply | Cancel

**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.

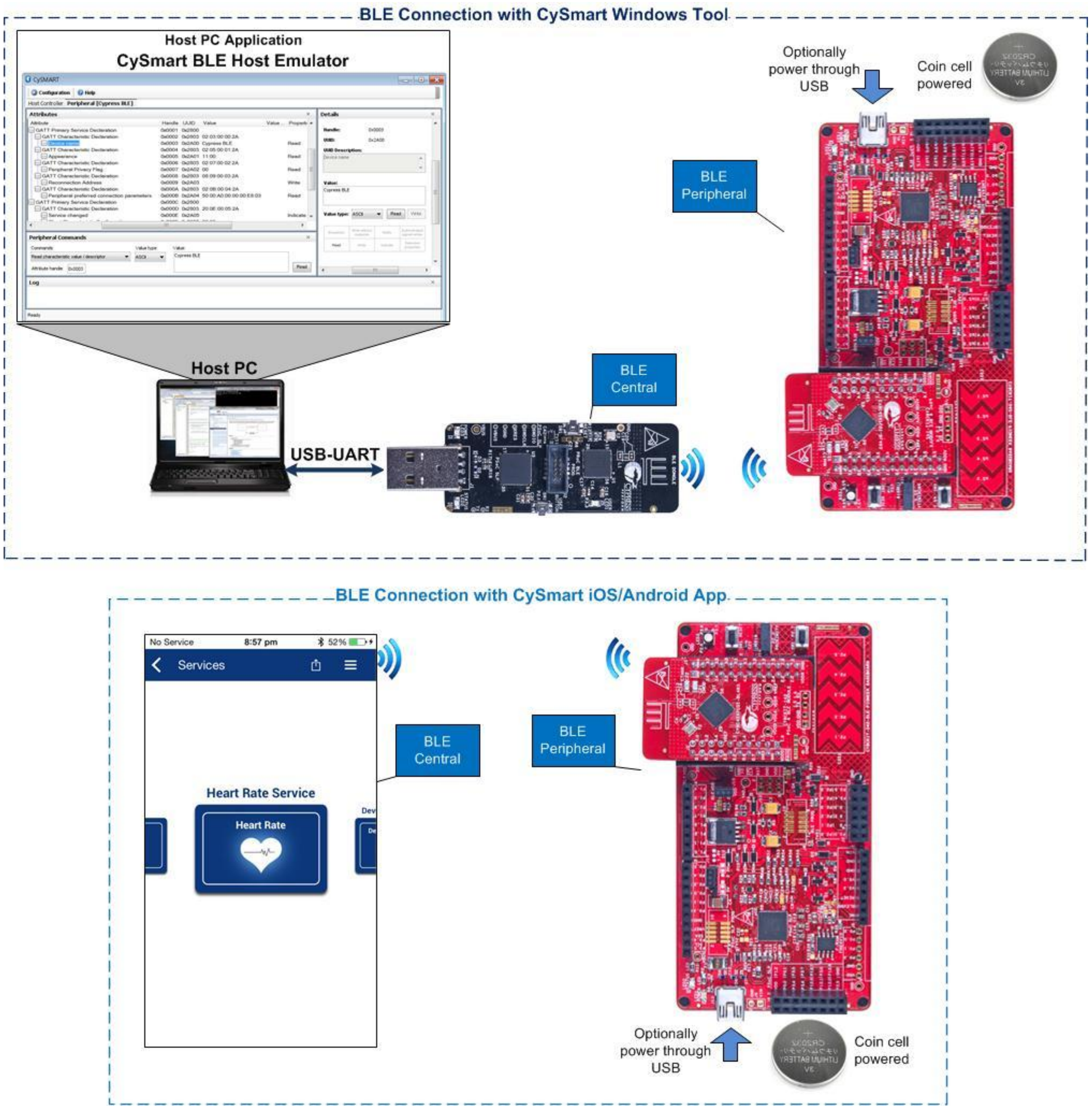
**Step 5:** Wait in the program main loop for an event from the BLE Component and take the necessary action or send data to the Central device using BLE Component API functions.

The My First PSoC 4 BLE Design section will walk you through a step-by-step configuration of the BLE Component for creating a simple Peripheral application. 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.

## 6 PSoC 4 BLE Development Setup

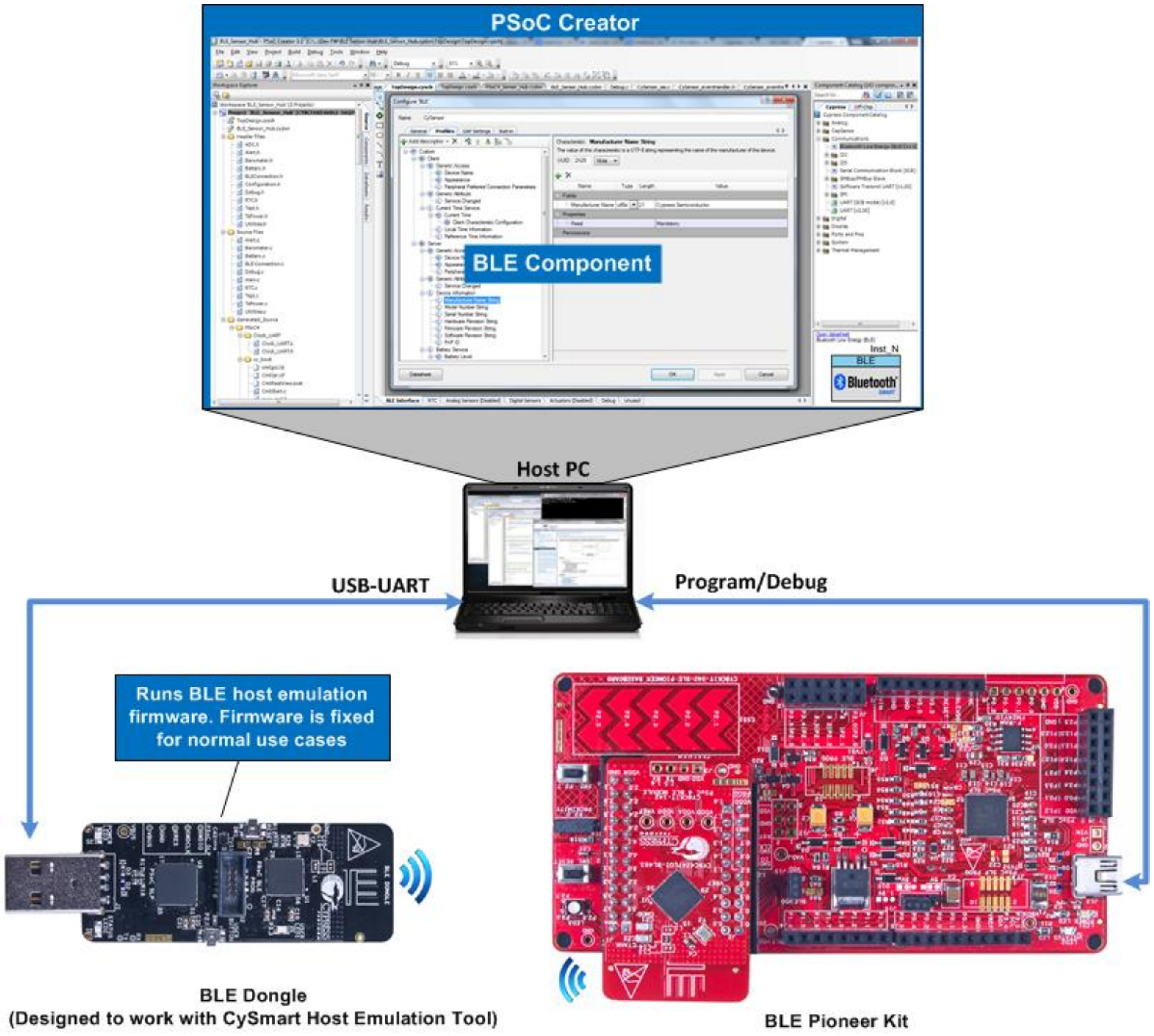
Figure 19 shows the hardware and software tools required for evaluating BLE Peripheral designs using the PSoC 4 BLE device. In a typical use case, the BLE Pioneer Kit (red board in Figure 19) is configured as a Peripheral that can communicate with a Central device such as CySmart iOS/Android app or CySmart Host Emulation Tool. The CySmart Host Emulation Tool also requires a BLE Dongle (black board in Figure 19) for its operation.

Figure 19. BLE Functional Setup



As shown in Figure 20, the BLE Dongle is pre-programmed 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 simultaneously be connected to a common host PC for development and testing.

Figure 20. BLE Development Setup


PSoC Creator
BLE Component
Host PC
USB-UART
Program/Debug
Runs BLE host emulation firmware. Firmware is fixed for normal use cases
BLE Dongle
(Designed to work with CySmart Host Emulation Tool)
BLE Pioneer Kit

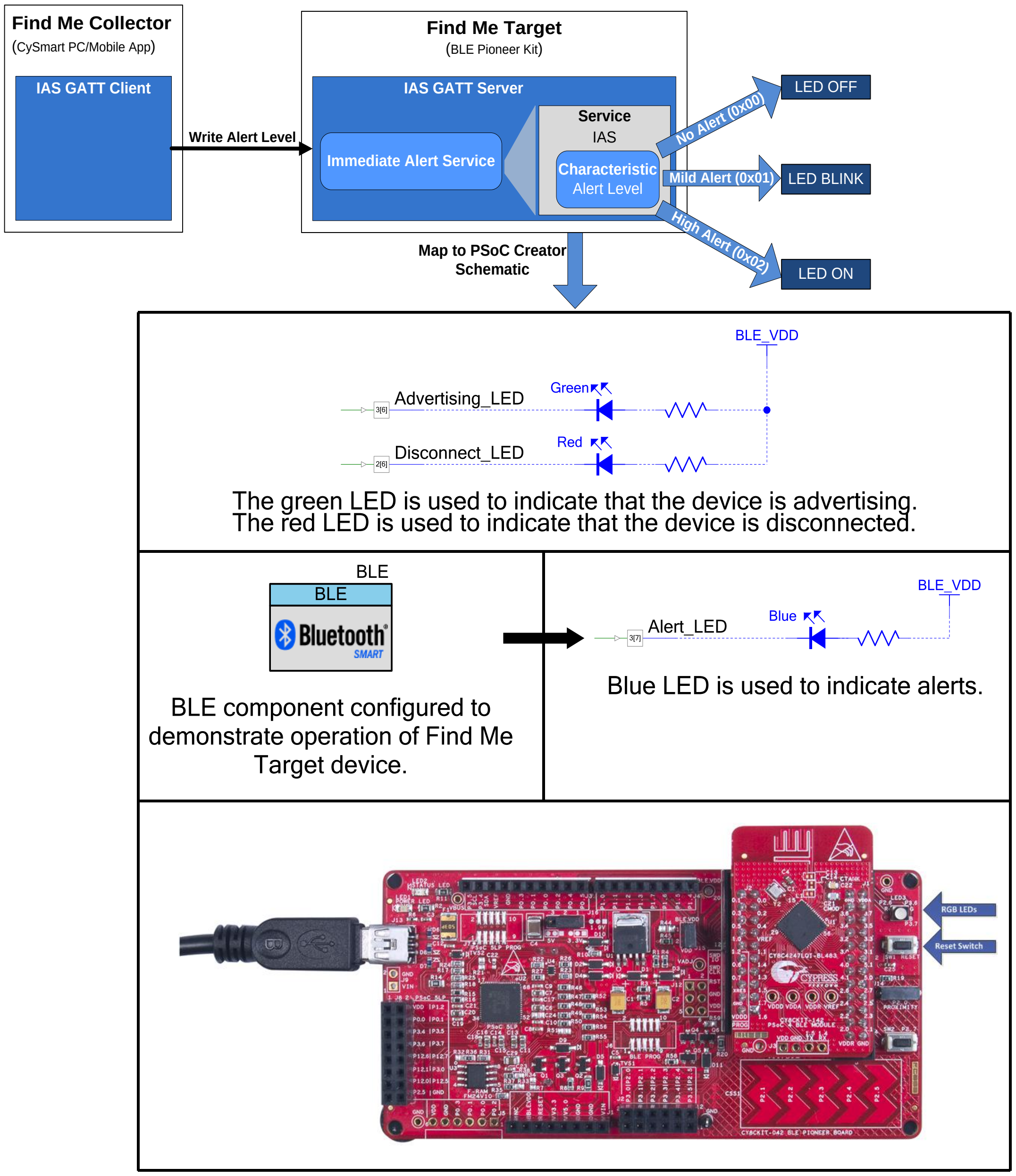
# 7 My First PSoC 4 BLE Design

This section gives you a step-by-step process for building a simple BLE Pioneer Kit–based design with the PSoC 4 BLE device using PSoC Creator. A simple Bluetooth SIG defined standard Profile design creation is described in this section. For creating advanced standard or custom Profile designs, refer to Designing BLE Applications and Creating a BLE Custom Profile application note.

## 7.1 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 21 shows. Two status LEDs indicate the state of the BLE interface.

Figure 21. My First PSoC 4 BLE Design



Create your first PSoC 4 BLE design in four stages:

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

**Note** The functional PSoC Creator project for the BLE example design described in this application note is distributed as part of PSoC Creator 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 (stage 1) and firmware development (stage 2) mentioned above and go to programming (stage 3) and testing the design (stage 4).

## 7.2 Prerequisites

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

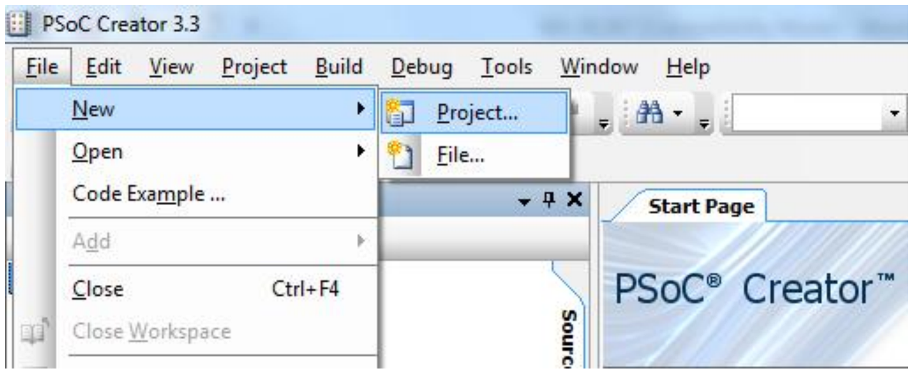
- PSoC Creator 3.3 SP1 or later with PSoC Programmer 3.23.3 or later
- CySmart Host Emulation Tool or CySmart iOS/Android app

## 7.3 Stage 1: Create the Design

This section takes you on a step-by-step guided tour of the design process. It starts with creating an empty project and guides you through schematic design entry.

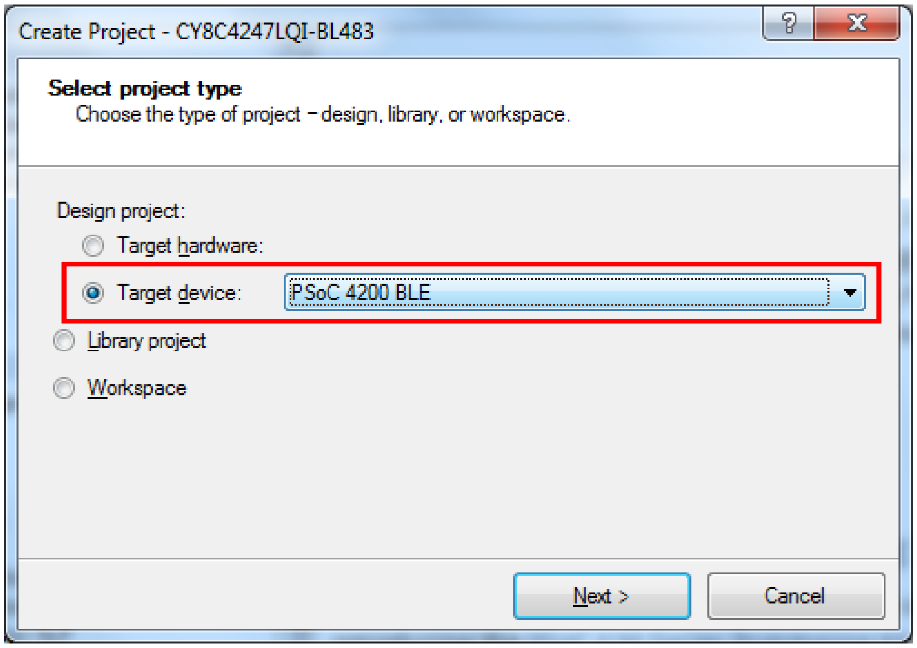
1. Install PSoC Creator 3.3 SP1 or later on your PC.
2. Start PSoC Creator, and from the **File** menu, choose **New** > **Project**, as Figure 22 shows.

Figure 22. Creating a New Project



3. Select the target device as “PSoC 4200 BLE” as shown in Figure 23 to select CY8C4247LQI-BL483 device used on BLE Pioneer Kit and click **Next**. If you are using a custom PSoC 4 BLE hardware or a different PSoC 4 BLE part number, choose the “Launch Device Selector” option in Target device and select the appropriate part number.

Figure 23. Selecting the Target Device



Create Project - CY8C4247LQI-BL483

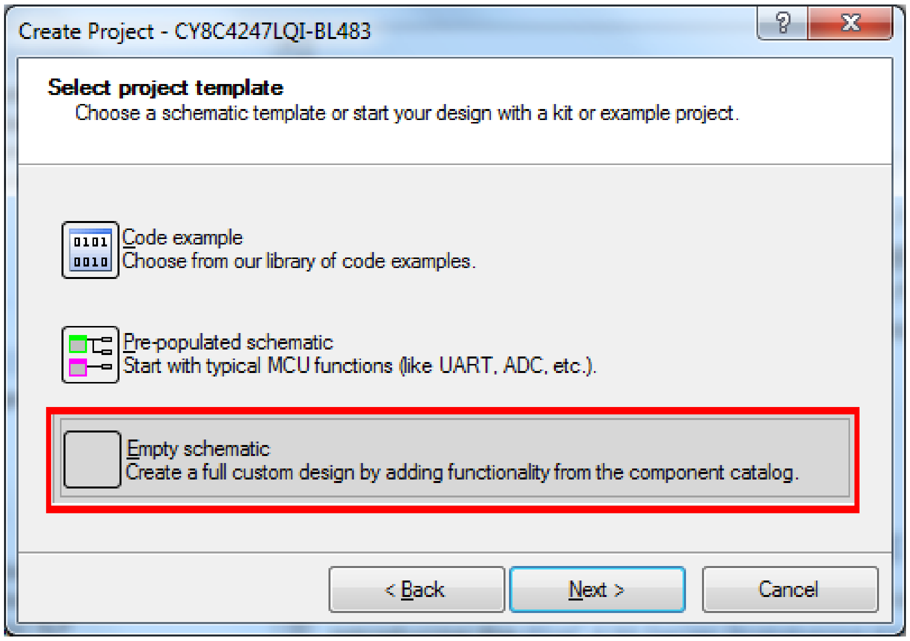
**Select project type**
Choose the type of project – design, library, or workspace.

Design project:
- Target hardware:
- [✓ Target device:] PSoC 4200 BLE
- Library project
- Workspace

4. Select an “Empty Schematic” project template as shown in Figure 24 and click **Next**.

**Note**: If you like to use the pre-built example for this design, select Code example in Figure 24 and click **Next**. In the code example selection window, select **Filter by** as **Find Me > BLE_FindMe** and skip to programming section directly.

Figure 24. Selecting the Project Template



Create Project - CY8C4247LQI-BL483

**Select project template**
Choose a schematic template or start your design with a kit or example project.

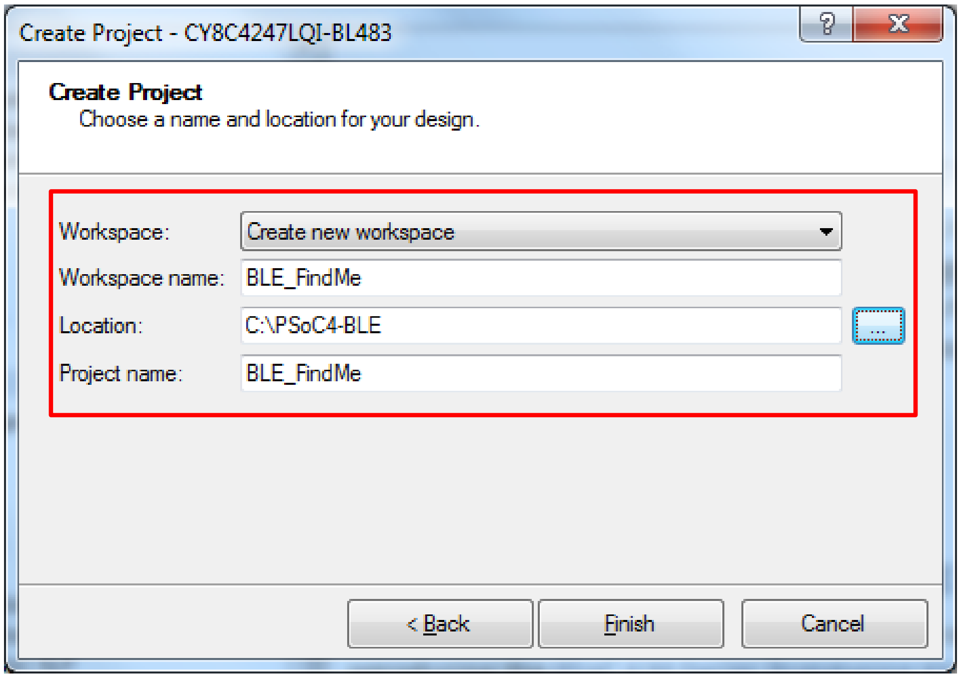
Code example
Choose from our library of code examples.

Pre-populated schematic
Start with typical MCU functions (like UART, ADC, etc.).

Empty schematic
Create a full custom design by adding functionality from the component catalog.

5. Give the workspace and the project a name such as “BLE_FindMe,” as Figure 25 shows. Choose an appropriate location for your new project, and then click **Finish**.

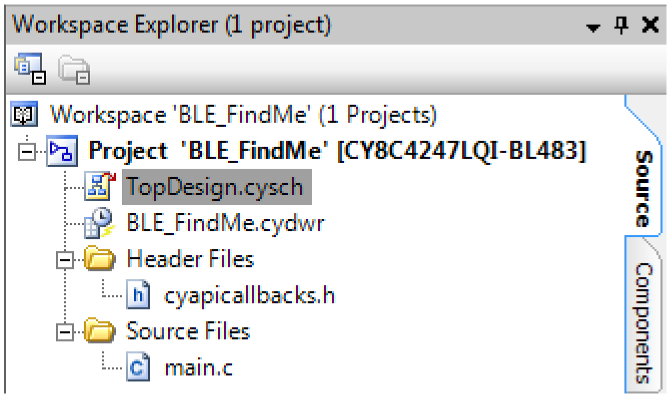
Figure 25. Project Naming


Create Project - CY8C4247LQI-BL483
Create Project
Choose a name and location for your design.
< Back
Finish
Cancel

Workspace: Create new workspace
Workspace name: BLE_FindMe
Location: C:\PSoC4-BLE
Project name: BLE_FindMe

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 26 shows.

Figure 26. Opening TopDesign Schematic

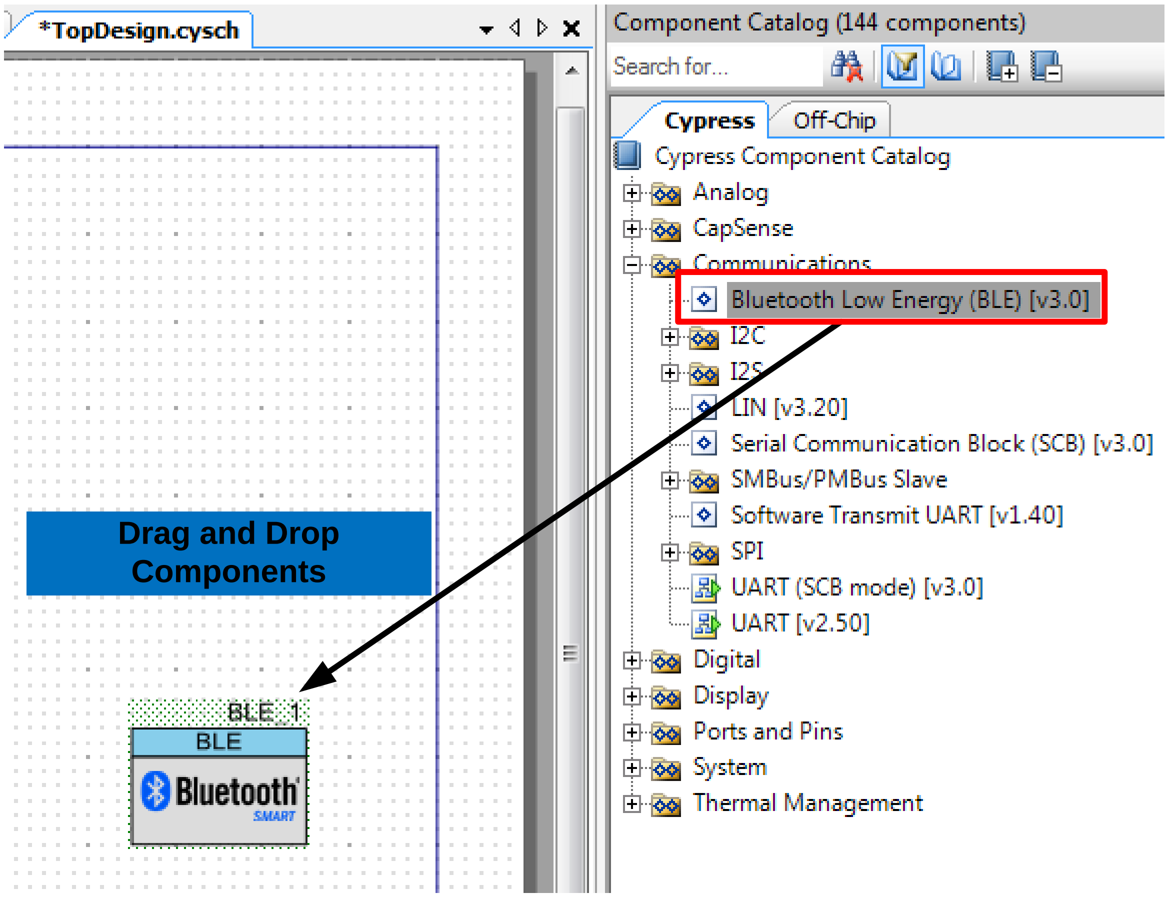


```
Workspace 'BLE_FindMe' (1 Projects)
  Project 'BLE_FindMe' [CY8C4247LQI-BL483]
    TopDesign.cysch
    BLE_FindMe.cydwr
    Header Files
      cyapicallbacks.h
    Source Files
      main.c
```

6. Open the project schematic file *TopDesign.cysch* by double-clicking it.
7. In the project schematic, you can create analog, digital, and communication designs by dragging and dropping Components and wiring them. For the current design, you need two Components—BLE and Pin—as Figure 21 shows.

8. Drag one Bluetooth Low Energy (BLE) Component from the **Component Catalog** onto the schematic, as Figure 27 shows.

Figure 27. Location of the BLE Component



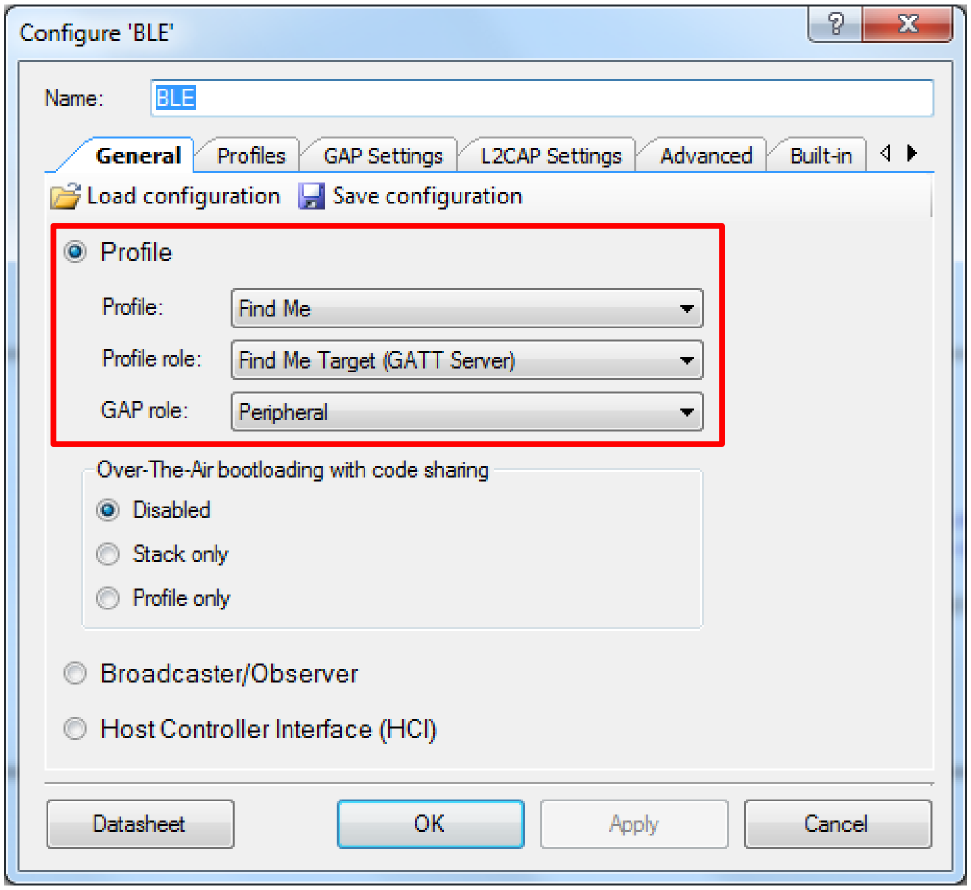
9. Double-click the BLE Component on the schematic to configure it as a “BLE Find Me Target” with the following properties:

- GAP Peripheral role with Find Me Target (GATT Server) configuration as shown in Figure 28.
- GAP Device Name set to “Find Me Target”, Appearance set to “Generic Keyring”, and check the Silicon generated “Company assigned” part of device address as shown in Figure 29. This configures the device name and type that appears when another device attempts to discover your device and assigns a unique BLE device address to your device.
- Limited advertisement mode with an advertising timeout of 30 seconds and a fast advertisement interval of 20 to 30 ms as shown in Figure 30. Fast advertising allows quick discovery and connection but consumes more power due to increased RF advertisement packets. De-select the “Slow advertising interval” checkbox.
- Advertisement Packet with Immediate Alert Service enabled as shown in Figure 31 and Scan Response Packet with Local Name, Tx Power Level, and Appearance fields enabled as shown in Figure 32.
- GAP security set to the lowest possible configuration that does not require authentication, encryption, or authorization for data exchange (Mode 1, No Security). Set the I/O capabilities to “No Input No Output” and Bonding requirement to “No Bonding”.

Figure 28 to Figure 34 show the BLE Component screenshots for this configuration.

**Note:** You do not need to change the default configuration of the BLE Component in the **GAP Settings > Peripheral preferred connection parameters**, **Profiles, L2CAP Settings,** and **Advanced** tab for this design.

Figure 28. BLE Component General Configuration



Name: BLE

[✓ Profile]

Profile: Find Me

Profile role: Find Me Target (GATT Server)

GAP role: Peripheral

Over-The-Air bootloading with code sharing

[✓ Disabled]

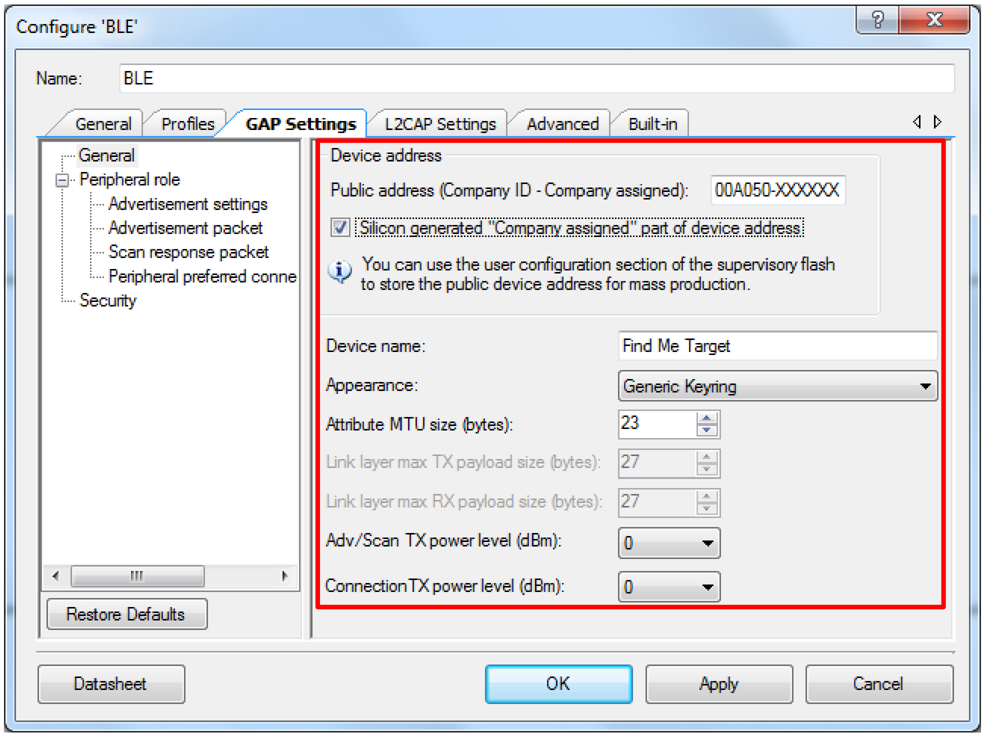
Stack only

Profile only

Broadcaster/Observer

Host Controller Interface (HCI)

Figure 29. BLE Component GAP General Settings



Configure 'BLE'

Name: BLE

General | Profiles | GAP Settings | L2CAP Settings | Advanced | Built-in

- General
- Peripheral role
  - Advertisement settings
  - Advertisement packet
  - Scan response packet
  - Peripheral preferred conne
- Security

Device address

Public address (Company ID - Company assigned): 00A050-XXXXXX

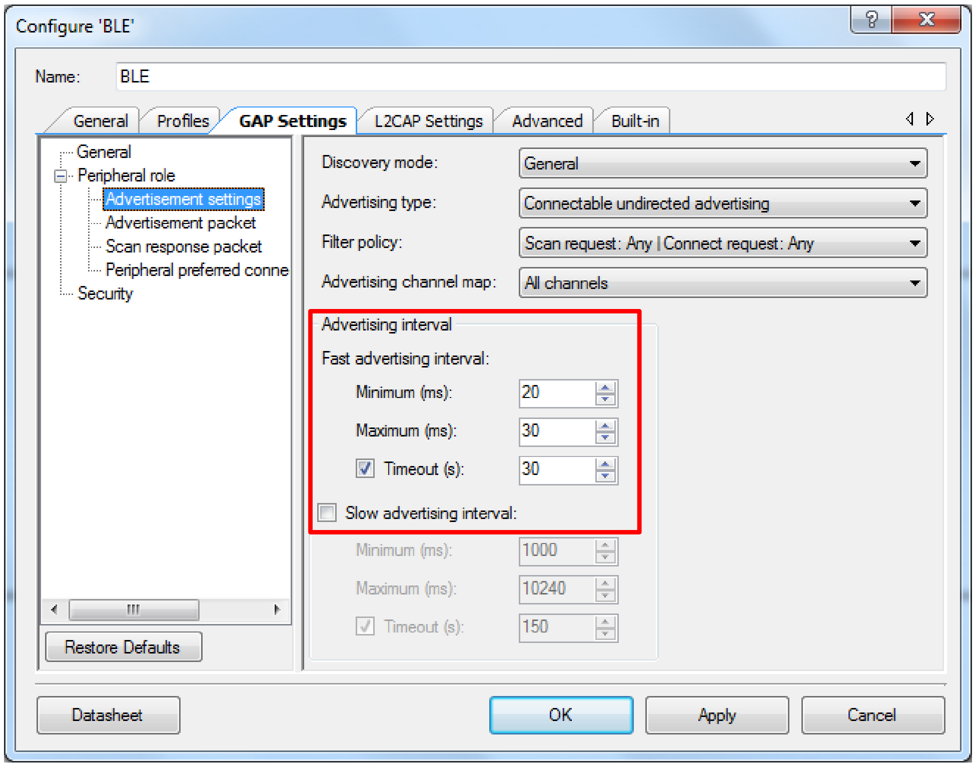
[✓] Silicon generated "Company assigned" part of device address

You can use the user configuration section of the supervisory flash to store the public device address for mass production.

| | |
|---|---|
| Device name: | Find Me Target |
| Appearance: | Generic Keyring |
| Attribute MTU size (bytes): | 23 |
| Link layer max TX payload size (bytes): | 27 |
| Link layer max RX payload size (bytes): | 27 |
| Adv/Scan TX power level (dBm): | 0 |
| Connection TX power level (dBm): | 0 |

Restore Defaults | Datasheet | OK | Apply | Cancel

Figure 30. BLE Component GAP Advertisement Settings



Configure 'BLE'

Name: BLE

General | Profiles | GAP Settings | L2CAP Settings | Advanced | Built-in

- General
- Peripheral role
  - Advertisement settings
  - Advertisement packet
  - Scan response packet
  - Peripheral preferred conne
- Security

| | |
|---|---|
| Discovery mode: | General |
| Advertising type: | Connectable undirected advertising |
| Filter policy: | Scan request: Any \| Connect request: Any |
| Advertising channel map: | All channels |

Advertising interval

Fast advertising interval:

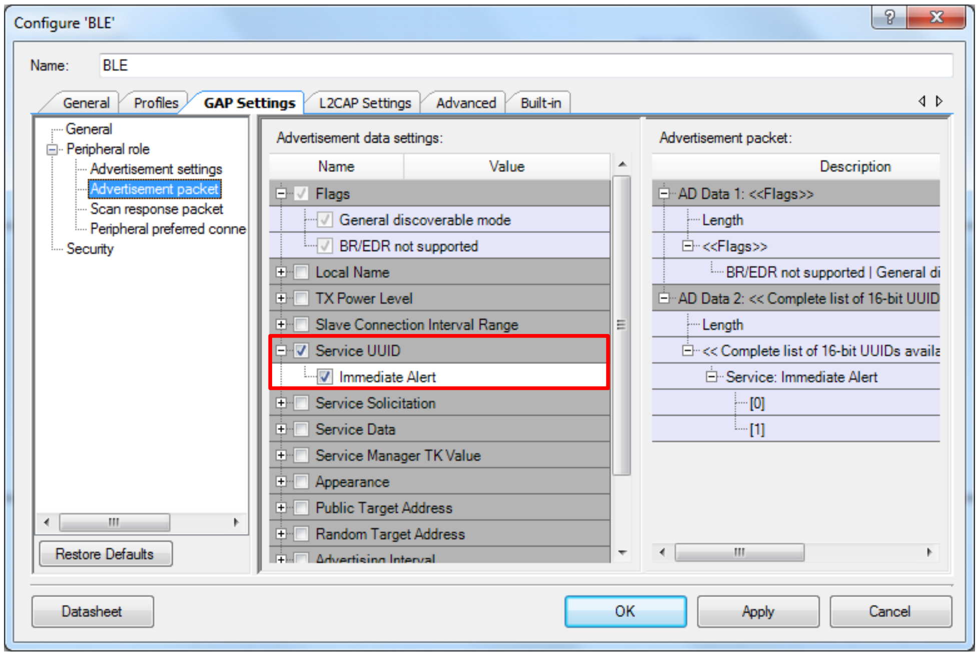
| | |
|---|---|
| Minimum (ms): | 20 |
| Maximum (ms): | 30 |
| [✓] Timeout (s): | 30 |

[ ] Slow advertising interval:

| | |
|---|---|
| Minimum (ms): | 1000 |
| Maximum (ms): | 10240 |
| [✓] Timeout (s): | 150 |

Restore Defaults | Datasheet | OK | Apply | Cancel

Figure 31. BLE Component GAP Advertisement Packet



Configure 'BLE'

Name: BLE

General | Profiles | **GAP Settings** | L2CAP Settings | Advanced | Built-in

- General
- Peripheral role
  - Advertisement settings
  - Advertisement packet
  - Scan response packet
  - Peripheral preferred conne
- Security

Restore Defaults

Advertisement data settings:

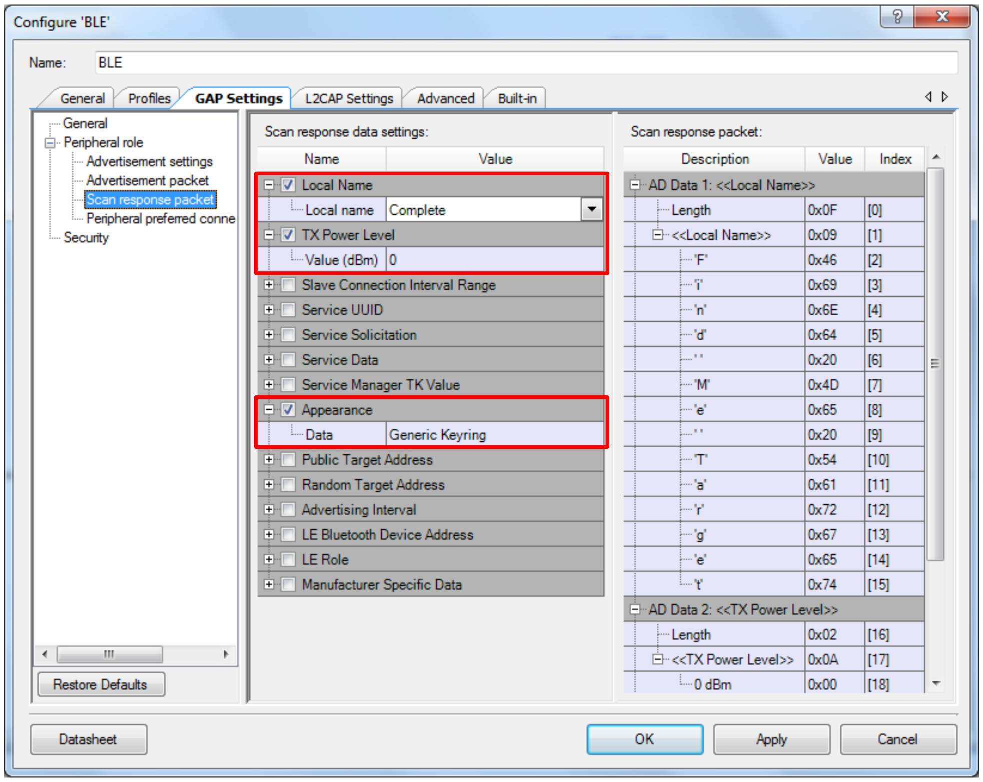
| Name | Value |
|---|---|
| ✓ Flags | |
| ✓ General discoverable mode | |
| ✓ BR/EDR not supported | |
| Local Name | |
| TX Power Level | |
| Slave Connection Interval Range | |
| ✓ Service UUID | |
| ✓ Immediate Alert | |
| Service Solicitation | |
| Service Data | |
| Service Manager TK Value | |
| Appearance | |
| Public Target Address | |
| Random Target Address | |
| Advertising Interval | |

Advertisement packet:

| Description |
|---|
| AD Data 1: <<Flags>> |
| Length |
| <<Flags>> |
| BR/EDR not supported \| General di |
| AD Data 2: << Complete list of 16-bit UUID |
| Length |
| << Complete list of 16-bit UUIDs availa |
| Service: Immediate Alert |
| [0] |
| [1] |

Datasheet | OK | Apply | Cancel

Figure 32. BLE Component GAP Scan Response Packet



Configure 'BLE'

Name: BLE

General | Profiles | **GAP Settings** | L2CAP Settings | Advanced | Built-in

- General
- Peripheral role
  - Advertisement settings
  - Advertisement packet
  - Scan response packet
  - Peripheral preferred conne
- Security

Restore Defaults

Scan response data settings:

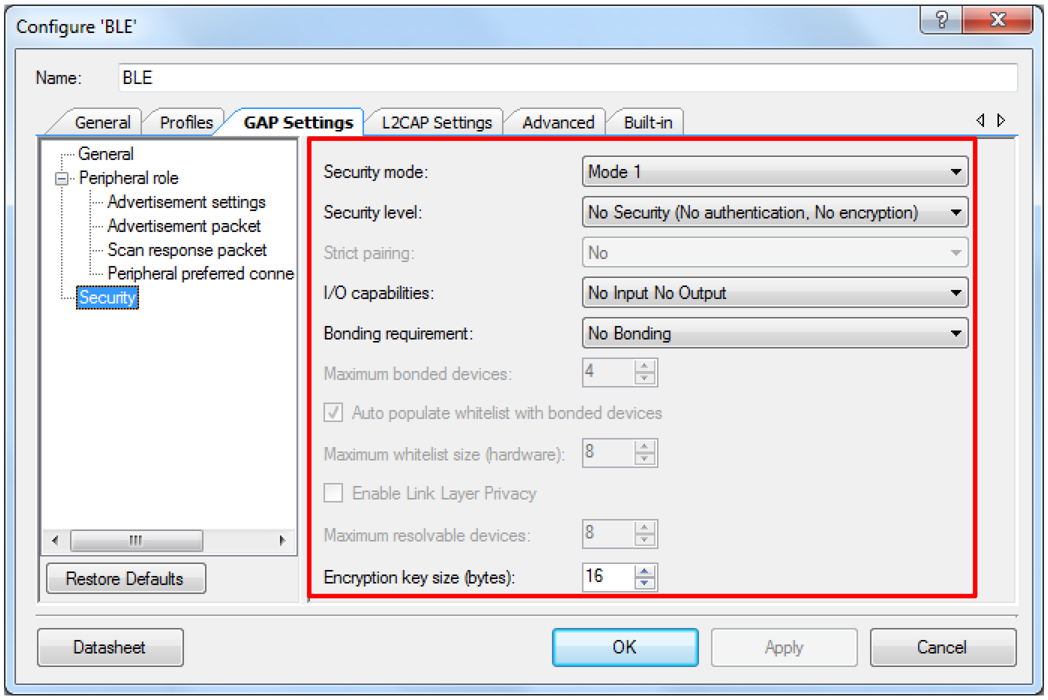
| Name | Value |
|---|---|
| ✓ Local Name | |
| Local name | Complete |
| ✓ TX Power Level | |
| Value (dBm) | 0 |
| Slave Connection Interval Range | |
| Service UUID | |
| Service Solicitation | |
| Service Data | |
| Service Manager TK Value | |
| ✓ Appearance | |
| Data | Generic Keyring |
| Public Target Address | |
| Random Target Address | |
| Advertising Interval | |
| LE Bluetooth Device Address | |
| LE Role | |
| Manufacturer Specific Data | |

Scan response packet:

| Description | Value | Index |
|---|---|---|
| AD Data 1: <<Local Name>> | | |
| Length | 0x0F | [0] |
| <<Local Name>> | 0x09 | [1] |
| 'F' | 0x46 | [2] |
| 'i' | 0x69 | [3] |
| 'n' | 0x6E | [4] |
| 'd' | 0x64 | [5] |
| ' ' | 0x20 | [6] |
| 'M' | 0x4D | [7] |
| 'e' | 0x65 | [8] |
| ' ' | 0x20 | [9] |
| 'T' | 0x54 | [10] |
| 'a' | 0x61 | [11] |
| 'r' | 0x72 | [12] |
| 'g' | 0x67 | [13] |
| 'e' | 0x65 | [14] |
| 't' | 0x74 | [15] |
| AD Data 2: <<TX Power Level>> | | |
| Length | 0x02 | [16] |
| <<TX Power Level>> | 0x0A | [17] |
| 0 dBm | 0x00 | [18] |

Datasheet | OK | Apply | Cancel

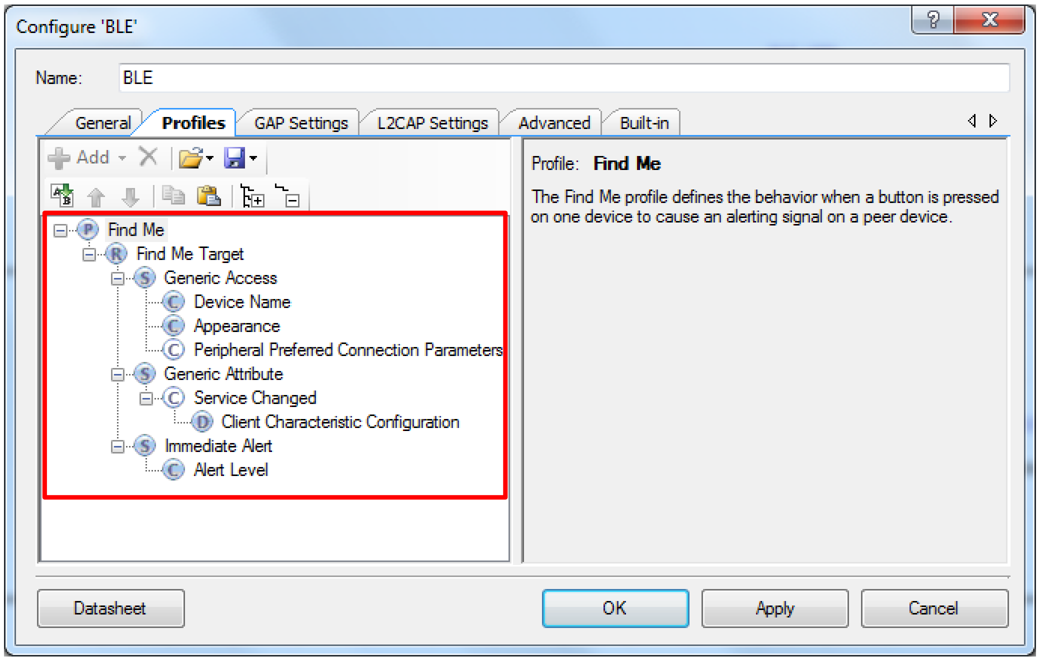
Figure 33. BLE Component GAP Security Settings



Name: BLE

Security mode: Mode 1
Security level: No Security (No authentication, No encryption)
Strict pairing: No
I/O capabilities: No Input No Output
Bonding requirement: No Bonding
Maximum bonded devices: 4
☑ Auto populate whitelist with bonded devices
Maximum whitelist size (hardware): 8
☐ Enable Link Layer Privacy
Maximum resolvable devices: 8
Encryption key size (bytes): 16

Figure 34. BLE Component Profiles Configuration



Name: BLE

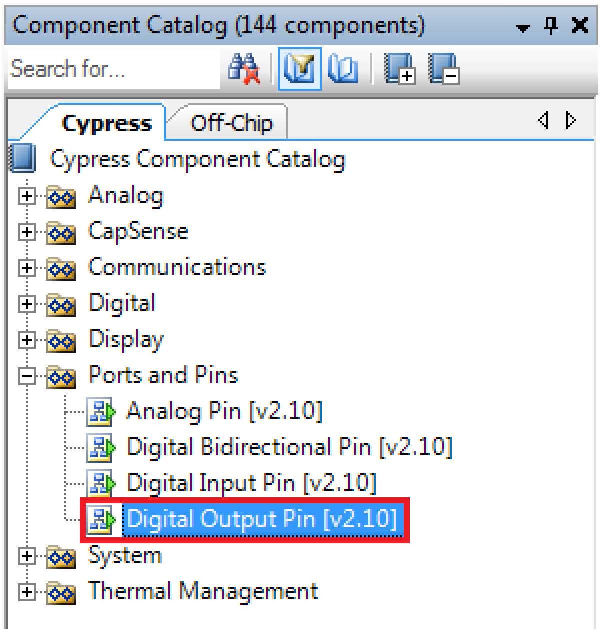
- Find Me
  - Find Me Target
    - Generic Access
      - Device Name
      - Appearance
      - Peripheral Preferred Connection Parameters
    - Generic Attribute
      - Service Changed
        - Client Characteristic Configuration
    - Immediate Alert
      - Alert Level

Profile: **Find Me**

The Find Me profile defines the behavior when a button is pressed on one device to cause an alerting signal on a peer device.

10. Drag and drop a Digital Output Pin Component as shown in Figure 35.

Figure 35. Location of the Digital Output Pin Component


Component Catalog (144 components)

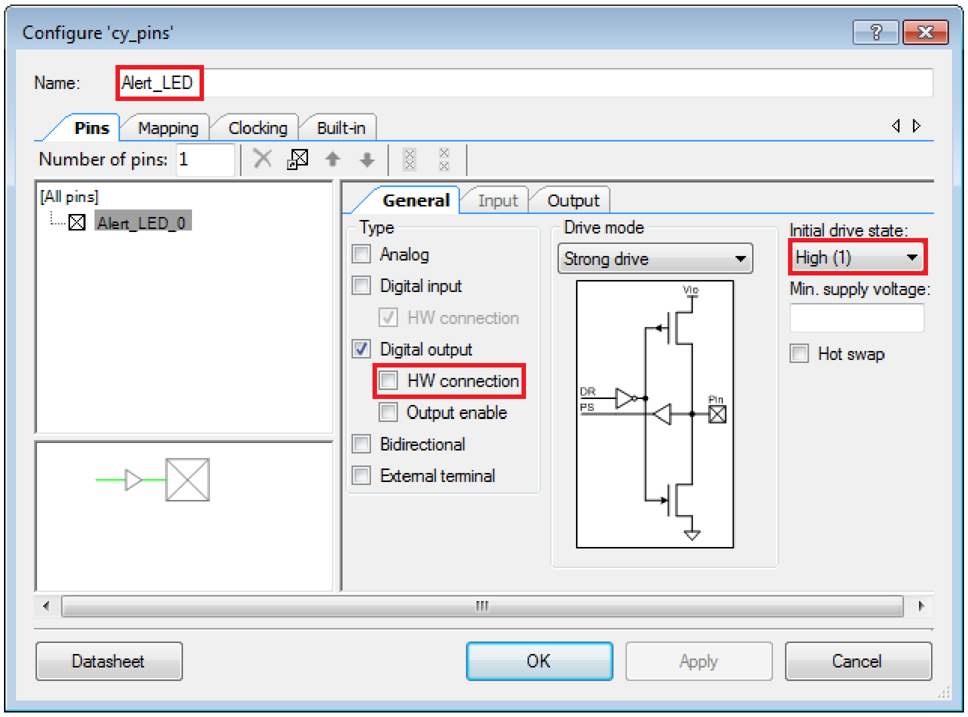
Search for...

Cypress | Off-Chip

- Cypress Component Catalog
  - Analog
  - CapSense
  - Communications
  - Digital
  - Display
  - Ports and Pins
    - Analog Pin [v2.10]
    - Digital Bidirectional Pin [v2.10]
    - Digital Input Pin [v2.10]
    - Digital Output Pin [v2.10]
  - System
  - Thermal Management

11. Double-click the Component and make the following changes as shown in Figure 36:
    - Deselect **HW Connection**.
    - Change the name to **Alert_LED**.
    - Set **Initial drive state** to “High(1)”.

Figure 36. Renaming a Pin Component


Configure 'cy_pins'

Name: Alert_LED

Pins | Mapping | Clocking | Built-in

Number of pins: 1

[All pins]
Alert_LED_0

General | Input | Output

Type
- [ ] Analog
- [ ] Digital input
  - [✓] HW connection
- [✓] Digital output
  - [ ] HW connection
  - [ ] Output enable
- [ ] Bidirectional
- [ ] External terminal

Drive mode: Strong drive

Initial drive state: High (1)

Min. supply voltage:

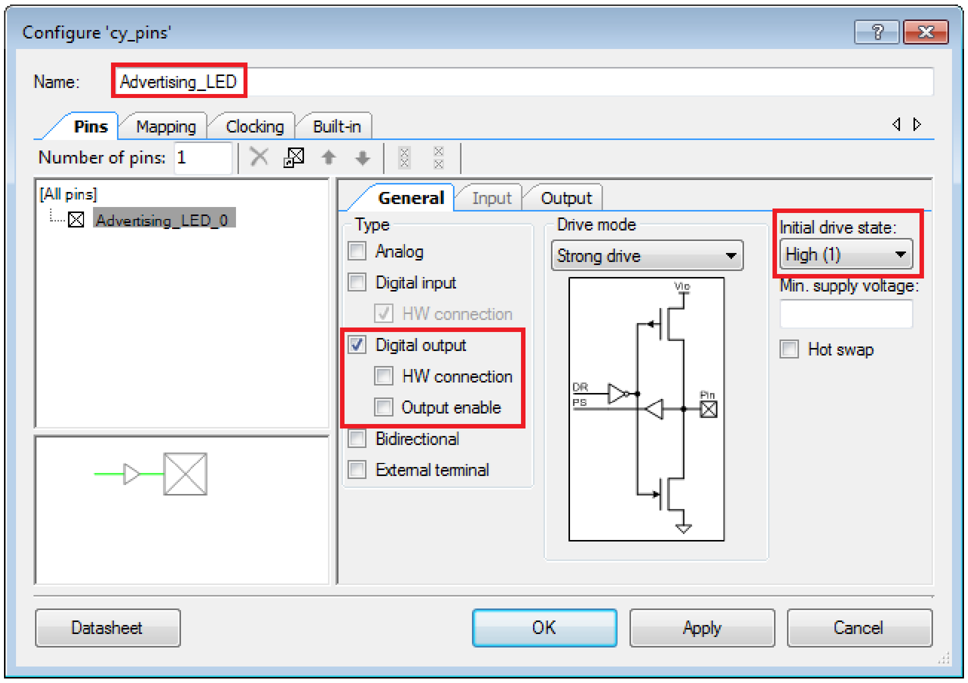
[ ] Hot swap

12. Add two more Digital Output Pin Components. Double-click the Components and make the following changes as Figure 37 and Figure 38 show:

    - Rename the Components as Advertising_LED and Disconnect_LED
    - Deselect **HW Connection.**
    - Set **Initial drive state** to “High(1)”.

    These pins will be used to drive the BLE advertising and disconnection state indicator LEDs. The LEDs on the BLE Pioneer Kit are active LOW; that is, the high pin-drive state turns off the LEDs and the low pin-drive state turns them on.

Figure 37. Advertising LED Configuration


Configure 'cy_pins'

Name: Advertising_LED

Number of pins: 1

[All pins]
Advertising_LED_0

Type
- [ ] Analog
- [ ] Digital input
  - [✓] HW connection
- [✓] Digital output
  - [ ] HW connection
  - [ ] Output enable
- [ ] Bidirectional
- [ ] External terminal

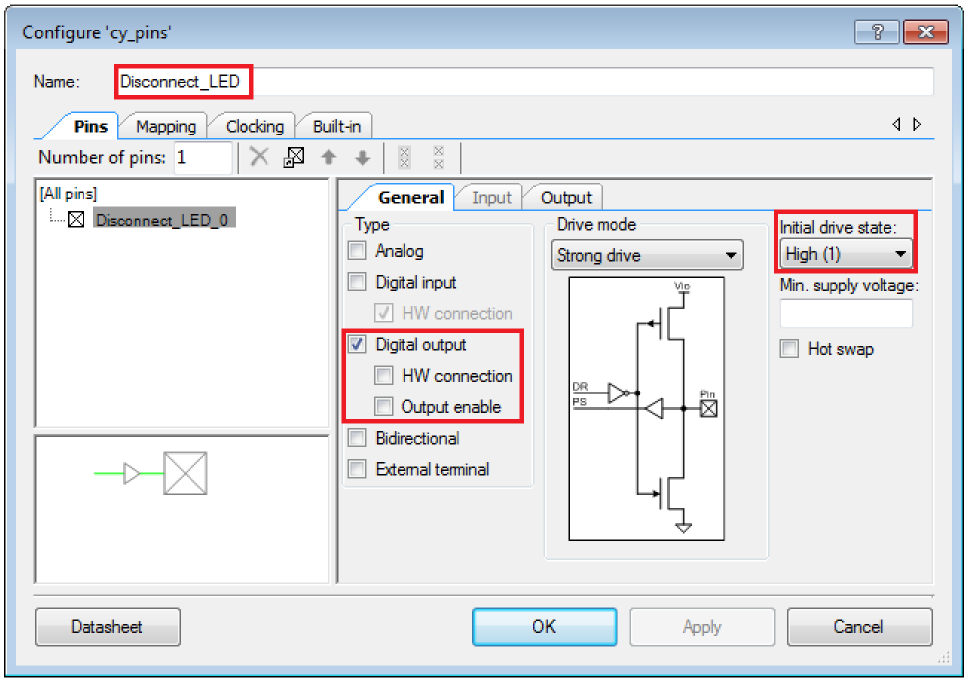
Drive mode: Strong drive

Initial drive state: High (1)

Min. supply voltage:

- [ ] Hot swap

Figure 38. Disconnection LED Configuration



Name: Disconnect_LED

Number of pins: 1

[All pins]
Disconnect_LED_0

Type
- [ ] Analog
- [ ] Digital input
  - [✓] HW connection
- [✓] Digital output
  - [ ] HW connection
  - [ ] Output enable
- [ ] Bidirectional
- [ ] External terminal

Drive mode: Strong drive

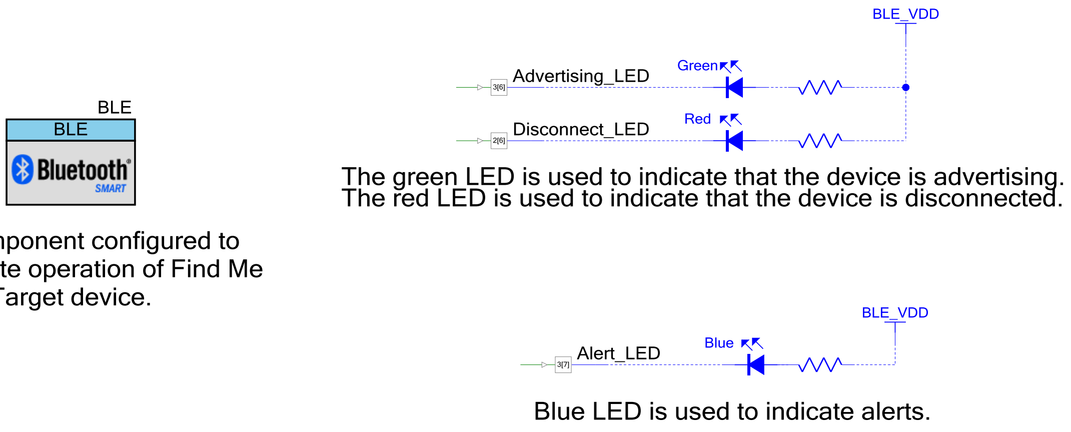
Initial drive state: High (1)

Min. supply voltage:

- [ ] Hot swap

13. After completing the schematic configuration, your design should look similar to Figure 39.

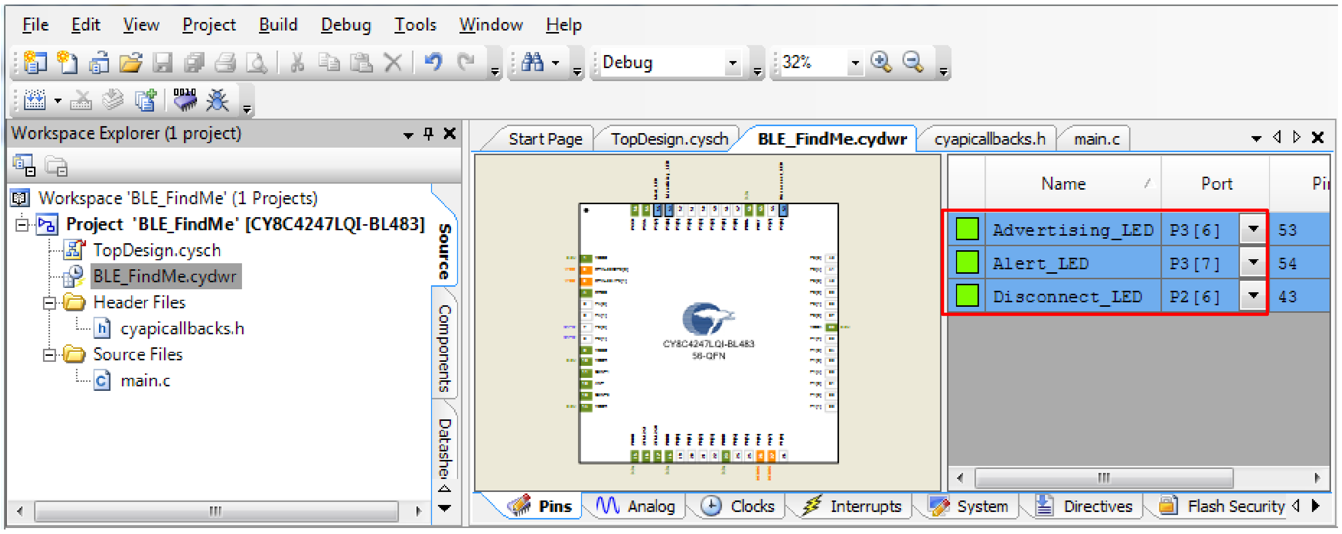
Figure 39. Schematic Configuration


BLE
BLE
Bluetooth SMART
BLE component configured to demonstrate operation of Find Me Target device.
BLE_VDD
Advertising_LED
Green
Disconnect_LED
Red
The green LED is used to indicate that the device is advertising.
The red LED is used to indicate that the device is disconnected.
BLE_VDD
Alert_LED
Blue
Blue LED is used to indicate alerts.

**Note** The blue dotted lines, the LED symbols, and resistor symbols shown in Figure 39 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.

14. 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 40 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 40. Pin Selection

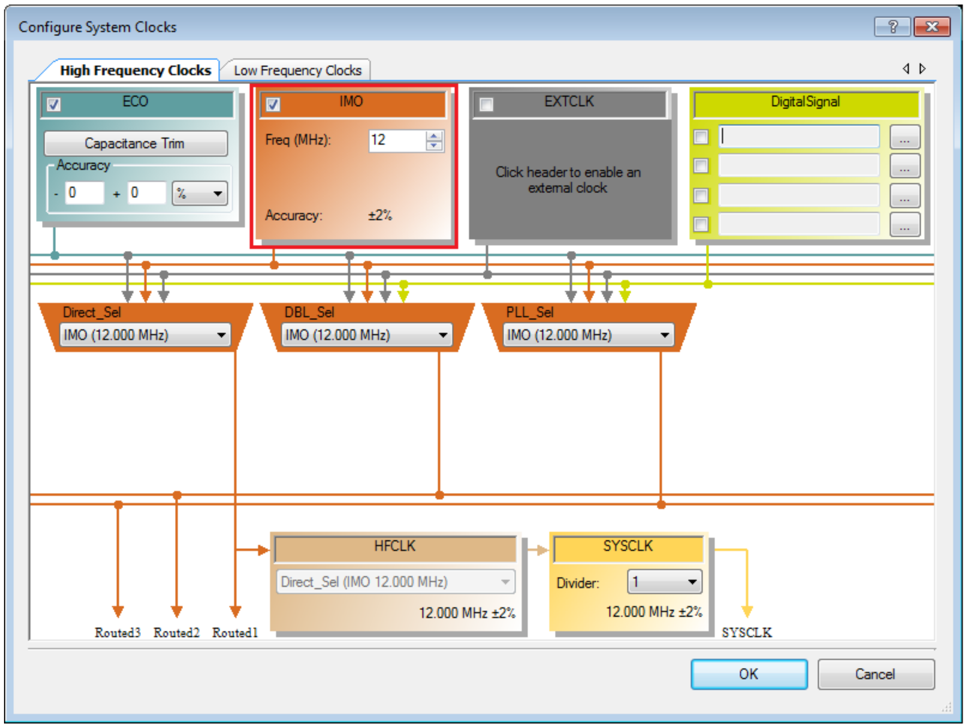


| | Name | Port | Pi |
|---|---|---|---|
| 🟩 | Advertising_LED | P3[6] | 53 |
| 🟩 | Alert_LED | P3[7] | 54 |
| 🟩 | Disconnect_LED | P2[6] | 43 |

If you are using your own board or a development kit with no LEDs, select the appropriate pins. You can connect external LEDs to the selected pins, as Figure 21 shows.

15. 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 41 shows. Click **OK**.

Figure 41. Clock Configuration



IMO: ✓ Freq (MHz): 12, Accuracy: ±2%

ECO: ✓ Accuracy - 0 + 0 %

Direct_Sel: IMO (12.000 MHz)

DBL_Sel: IMO (12.000 MHz)

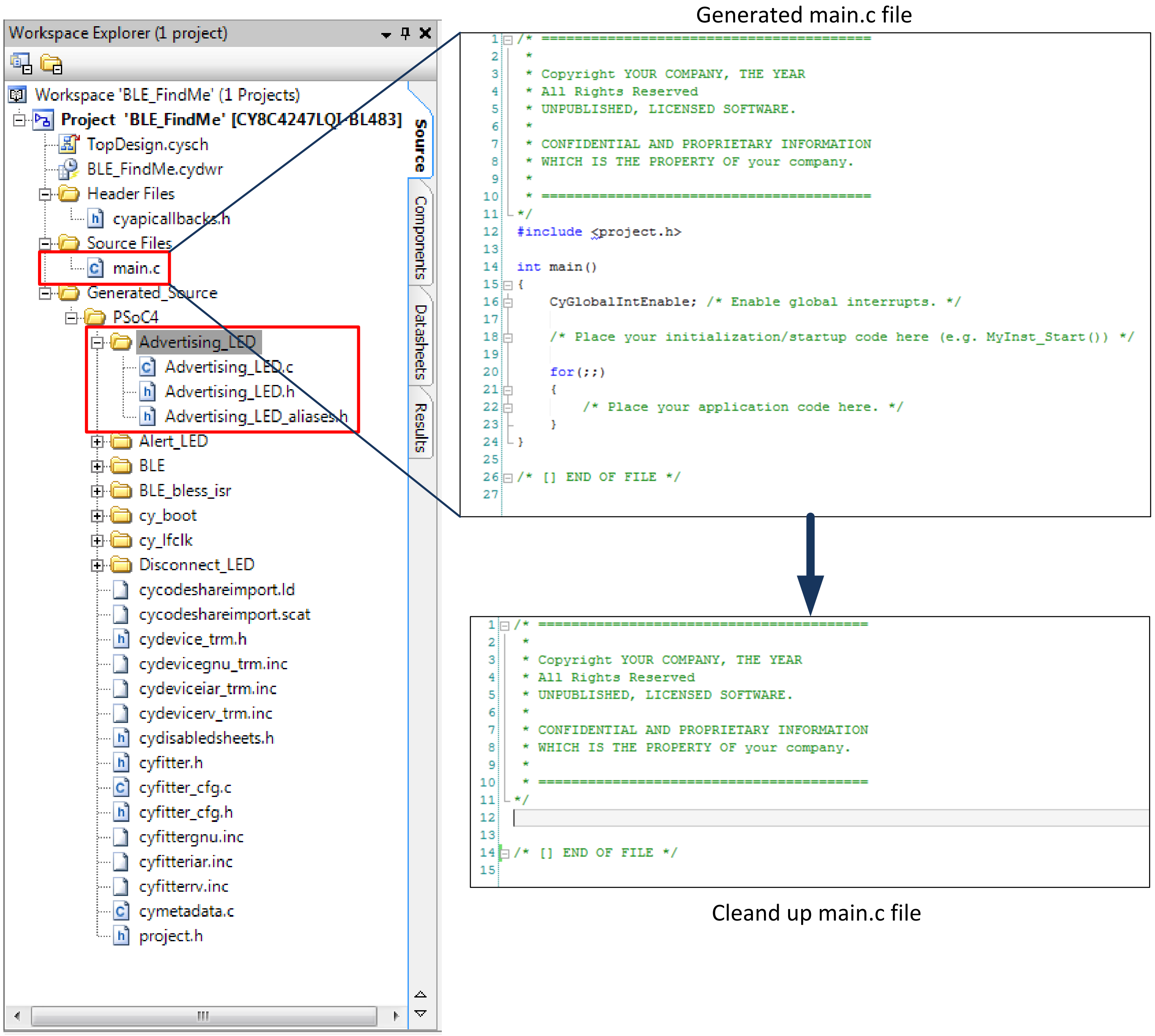
PLL_Sel: IMO (12.000 MHz)

HFCLK: Direct_Sel (IMO 12.000 MHz) 12.000 MHz ±2%

SYSCLK: Divider: 1 12.000 MHz ±2%

16. Select **Generate Application** from the **Build** menu. Notice in the **Workspace Explorer** window that PSoC Creator automatically generates source code files for the BLE and Digital Output Pin Components, as Figure 42 shows. Clear the contents of generated main.c file before proceeding to next section.

Figure 42. Generated Source Files



```
/* ========================================
 *
 * Copyright YOUR COMPANY, THE YEAR
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF your company.
 *
 * ========================================
*/
#include <project.h>

int main()
{
    CyGlobalIntEnable; /* Enable global interrupts. */

    /* Place your initialization/startup code here (e.g. MyInst_Start()) */

    for(;;)
    {
        /* Place your application code here. */
    }
}

/* [] END OF FILE */
```

```
/* ========================================
 *
 * Copyright YOUR COMPANY, THE YEAR
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF your company.
 *
 * ========================================
*/


/* [] END OF FILE */
```

## 7.4 Stage 2: Write the Firmware

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

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

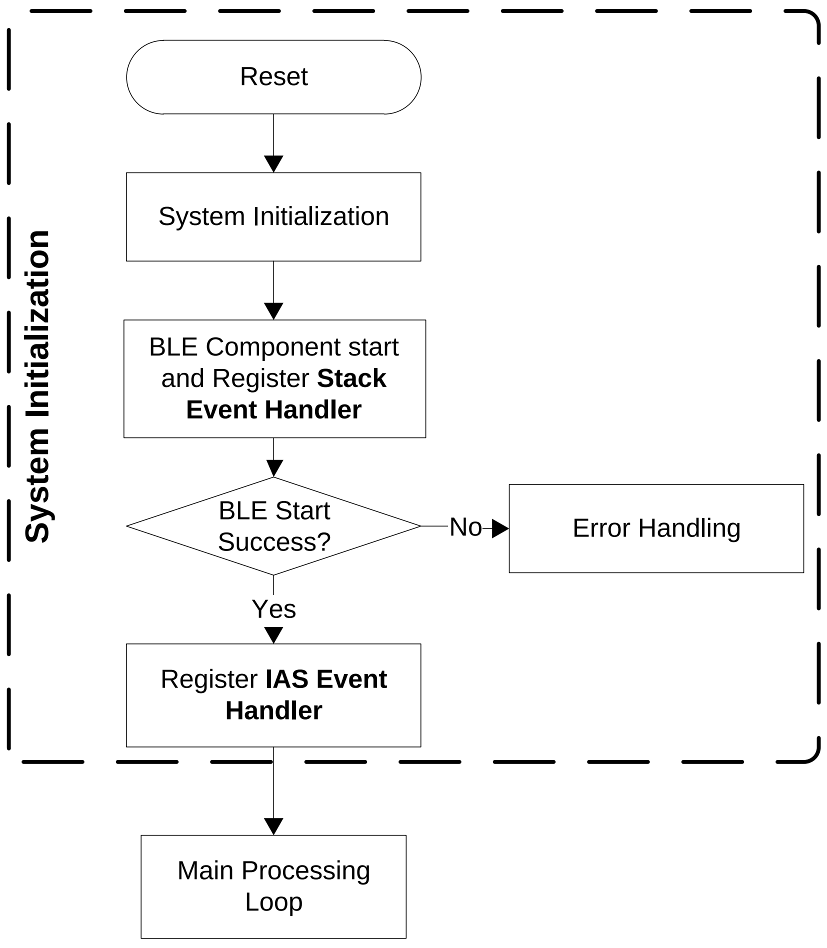
The following sections discuss these blocks with respect to the design that you configured in Stage 1: Create the Design. The code snippets provided in this section are to be placed inside the main.c file shown in Figure 42.

### 7.4.1 System Initialization

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

As a part of the BLE Component initialization, you must pass the event-handler function, which will be called by the BLE stack to notify pending events. The BLE stack event handler shown in Code 2 is registered as a part of the BLE initialization. If the BLE Component initializes successfully, the firmware registers another event handler for the events specific to the Immediate Alert Service (IAS) and switches control to the main loop. Figure 43 and Code 1 show the flow chart and the firmware source code for system initialization.

Figure 43. System Initialization Flow Chart



Code 1. System Initialization Firmware

```
#include <project.h>

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

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

#define LED_TOGGLE_TIMEOUT        (100u)

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 */
}
```

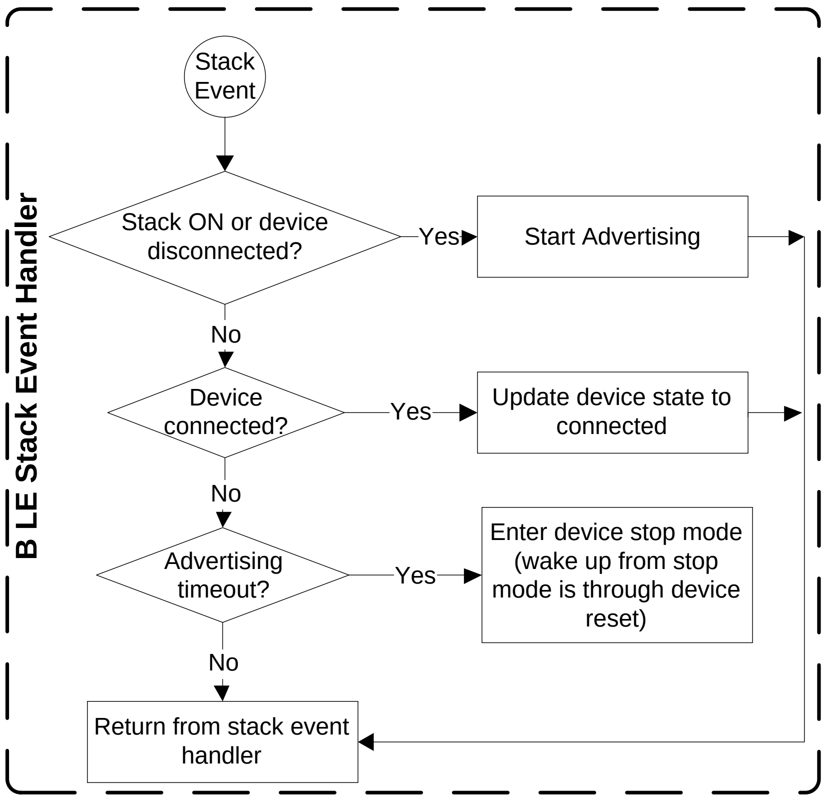
### 7.4.2 BLE Stack Event Handler

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 by you. The event handler must handle a few basic events from the stack, such as device connection and stack on, and configure the stack accordingly to establish and maintain the BLE link. For the Find Me Target application that you are creating, the BLE stack event handler must process all the events described in Table 1. The flow chart and the firmware for handling BLE stack events are shown in Figure 44 and Code 2.

Table 1. BLE Stack Events

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

Figure 44. BLE Stack Event Handler Flow Chart


B LE Stack Event Handler
Stack Event
Stack ON or device disconnected?
Yes
Start Advertising
No
Device connected?
Yes
Update device state to connected
No
Advertising timeout?
Yes
Enter device stop mode (wake up from stop mode is through device reset)
No
Return from stack event handler

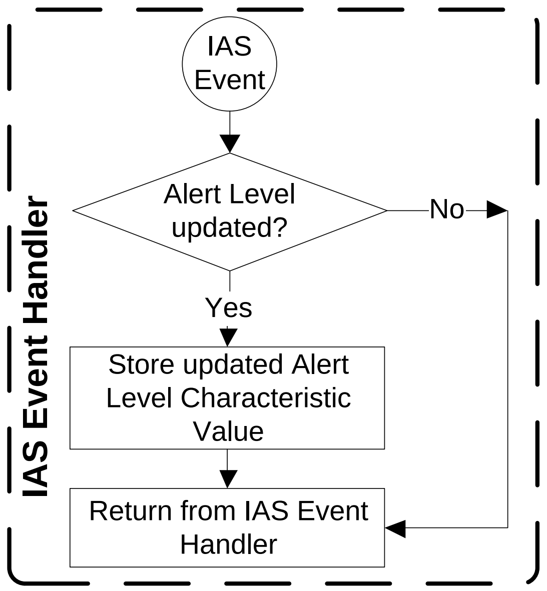
Code 2. BLE Stack Event Handler Firmware

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

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

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

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

        default:
        break;
    }
}
```

### 7.4.3 BLE Service-Specific Event Handler

The BLE Component also generates events corresponding to each of the Services supported by your design. For the Find Me Target application that you are creating, the BLE Component will generate IAS events that will let the application know if the Alert Level Characteristic is updated with a new value. The flow chart and the firmware for handling BLE IAS events are shown in Figure 45 and Code 3 respectively.

Figure 45. BLE IAS Event Handler Flow Chart


IAS Event Handler
IAS Event
Alert Level updated?
No
Yes
Store updated Alert Level Characteristic Value
Return from IAS Event Handler

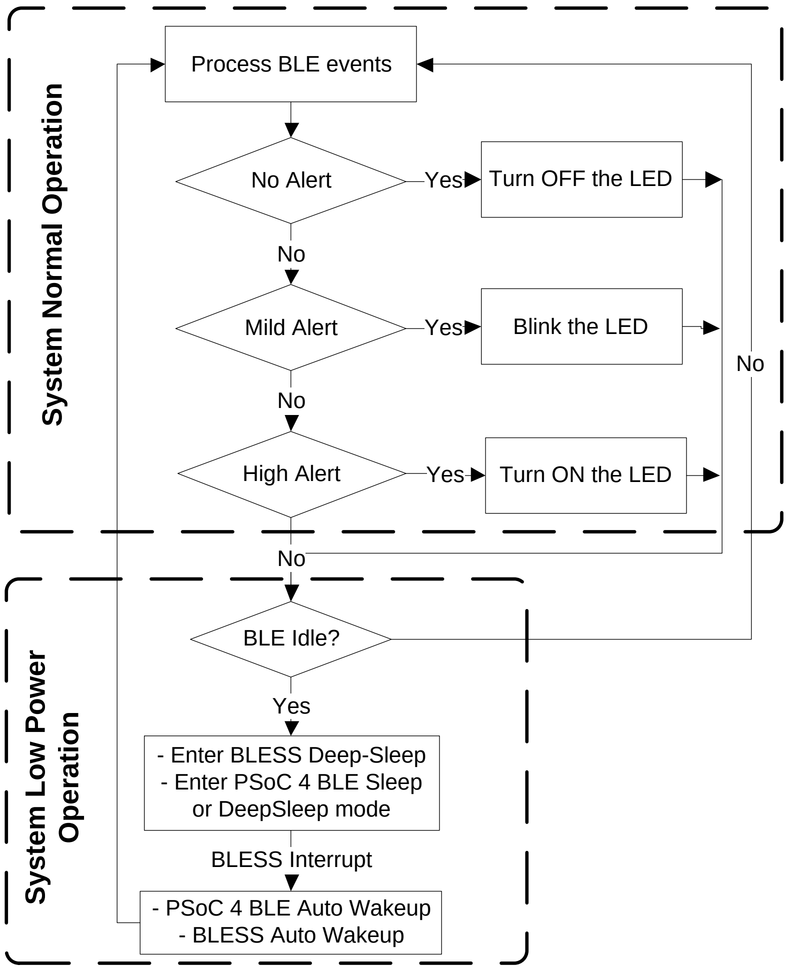
Code 3. BLE IAS Event Handler Firmware

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

### 7.4.4 Main Loop and Low-Power Implementation

The main loop firmware in your design must periodically service the BLE stack-processing event, update the blue alert LED state per the IAS Alert Level Characteristic value and configure the BLE subsystem (BLESS) block and the PSoC 4 BLE device into the low-power mode. The main loop flow chart and the firmware are shown in Figure 46 and Code 4. You should place the block of code in Code 4 inside the main function shown in Code 1 after the IasEventHandler function callback is registered.

Figure 46. Firmware Main Loop Flow Chart



Code 4. Main Loop Firmware

```
/* Place the main application loop here */

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 */
}
```

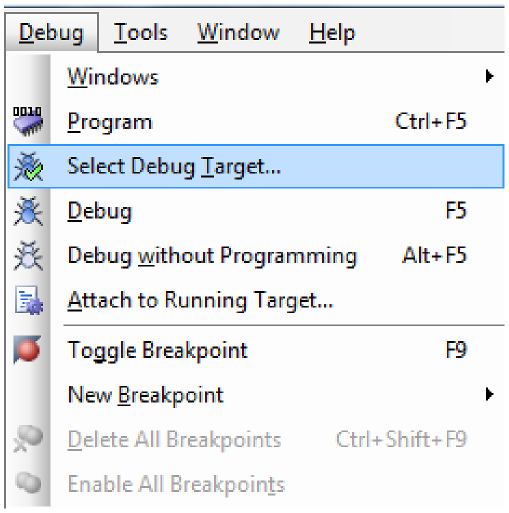
## 7.5 Stage 3: Program the Device

This section shows how to program the PSoC 4 BLE device. If you are using a development kit with a built-in programmer (the BLE Pioneer Kit, for example), connect the kit board to your computer using the USB cable. For other kits, refer to the kit guide. If you are developing on your own hardware, you need a hardware debugger, for example, a Cypress CY8CKIT–002 MiniProg3.

**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 .**

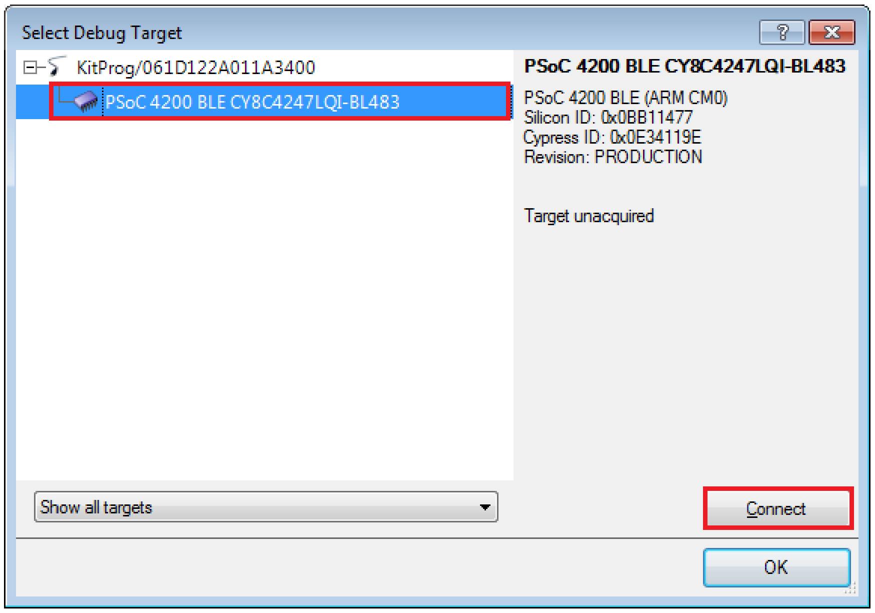
1. In PSoC Creator, choose **Debug** > **Select Debug Target**, as Figure 47 shows.

Figure 47. Selecting Debug Target



2. In the **Select Debug Target** dialog box, click **Port Acquire**, and then click **Connect**, as Figure 48 shows. Click **OK** to close the dialog box.

Figure 48. Connecting to a Device



PSoC 4200 BLE CY8C4247LQI-BL483

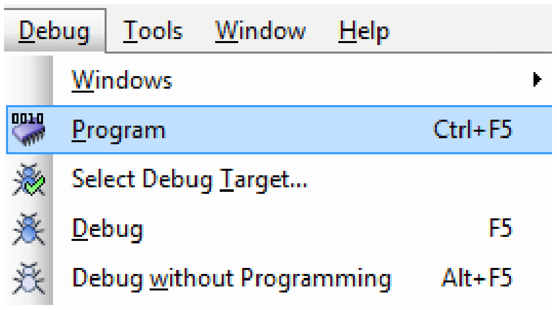
PSoC 4200 BLE (ARM CM0)
Silicon ID: 0x0BB11477
Cypress ID: 0x0E34119E
Revision: PRODUCTION

Target unacquired

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

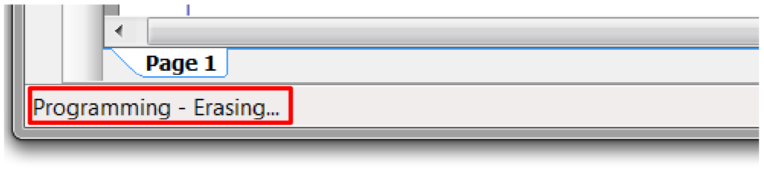
3. In PSoC Creator, choose **Debug** > **Program** to program the device with the project, as Figure 49 shows.

Figure 49. Programming the Device



4. You can view the programming status on the PSoC Creator status bar (lower left corner of the window), as Figure 50 shows,

Figure 50. Programming Status



Programming - Erasing...

## 7.6 Stage 4: Test Your Design

This section describes how to test your BLE design using the CySmart mobile apps and PC tool. The setup for testing your design using the BLE Pioneer Kit is shown in Figure 19.

See the following appendixes for more information on these tools:

- CySmart Host Emulation Tool
- CySmart Mobile App

### 7.6.1 With CySmart Mobile App

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 51 and Figure 52.

Figure 51. Testing with CySmart iOS App

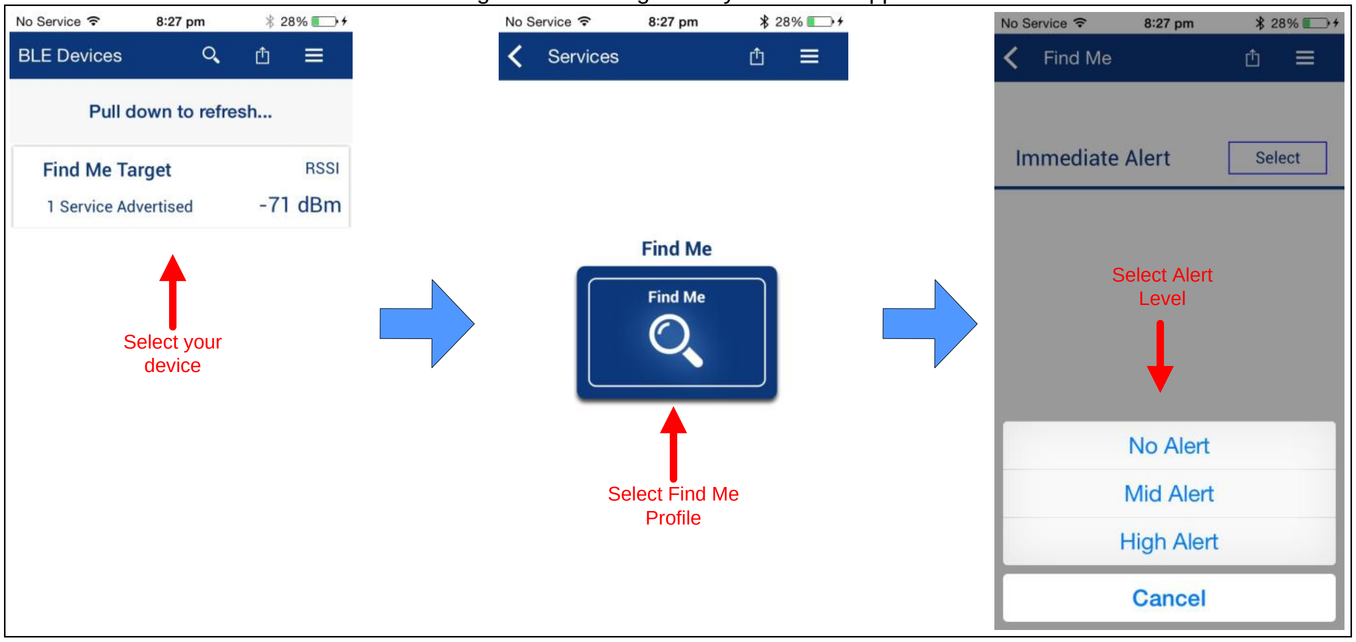
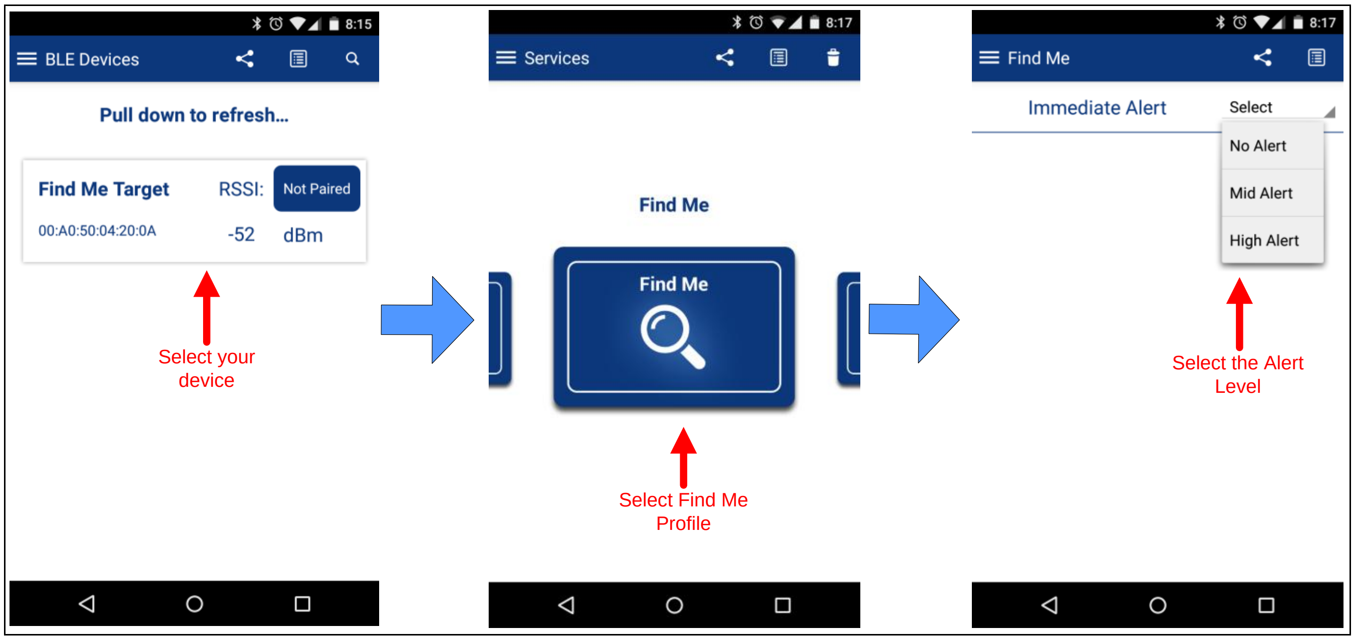


Figure 52. Testing with CySmart Android App

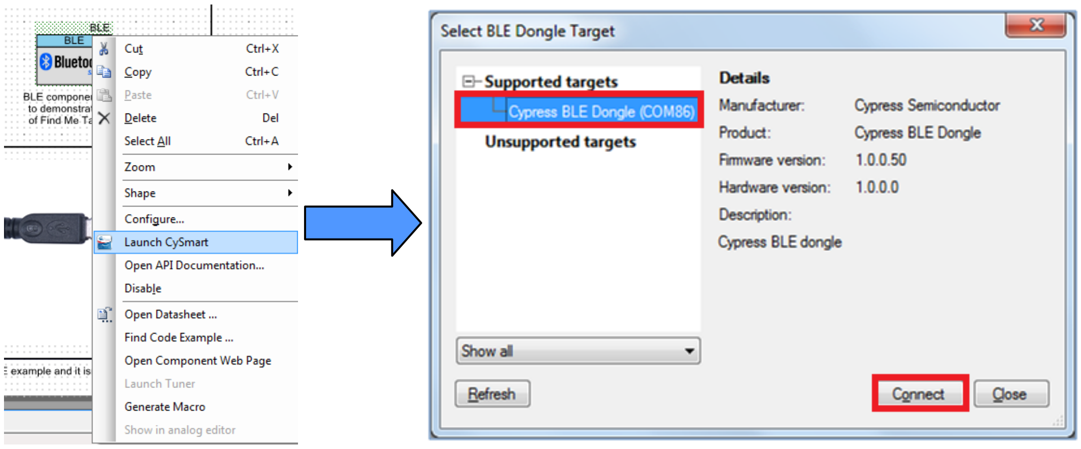


### 7.6.2 With CySmart Host Emulation Tool

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 53.

Figure 53. CySmart BLE Dongle Selection



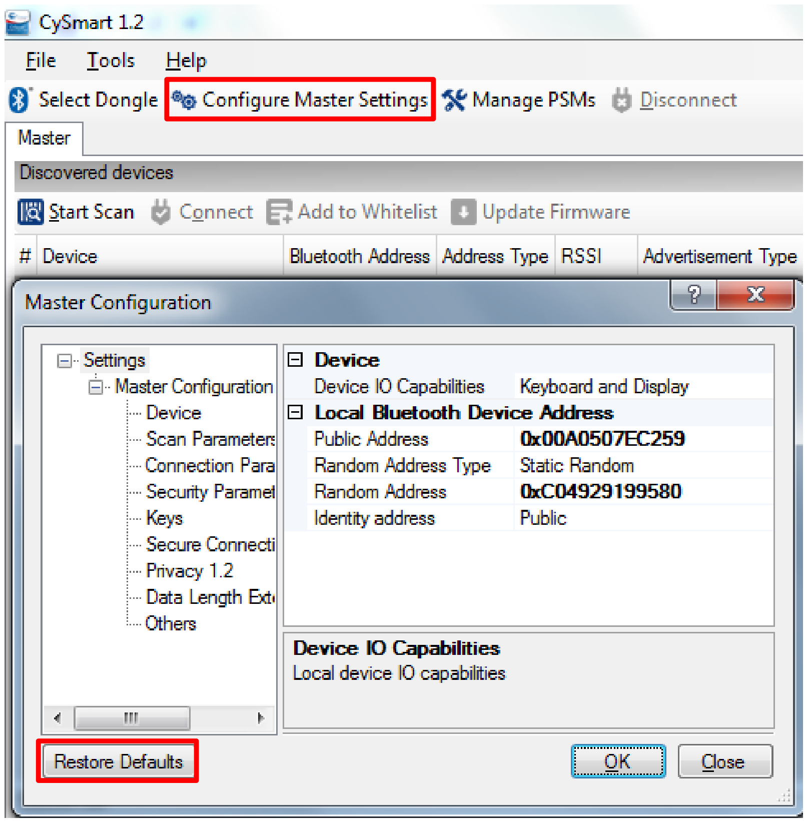
Details

| | |
|---|---|
| Manufacturer: | Cypress Semiconductor |
| Product: | Cypress BLE Dongle |
| Firmware version: | 1.0.0.50 |
| Hardware version: | 1.0.0.0 |
| Description: | |

Cypress BLE dongle

3. Select **Configure Master Settings** and restore the values to the default settings, as shown in Figure 54.

Figure 54. CySmart Master Settings Configuration

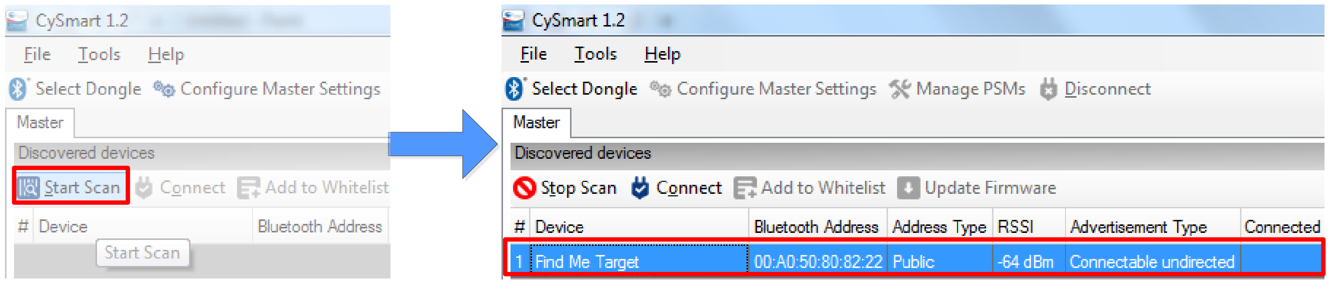


| | |
|---|---|
| **Device** | |
| Device IO Capabilities | Keyboard and Display |
| **Local Bluetooth Device Address** | |
| Public Address | **0x00A0507EC259** |
| Random Address Type | Static Random |
| Random Address | **0xC04929199580** |
| Identity address | Public |

**Device IO Capabilities**
Local device IO capabilities

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 55.

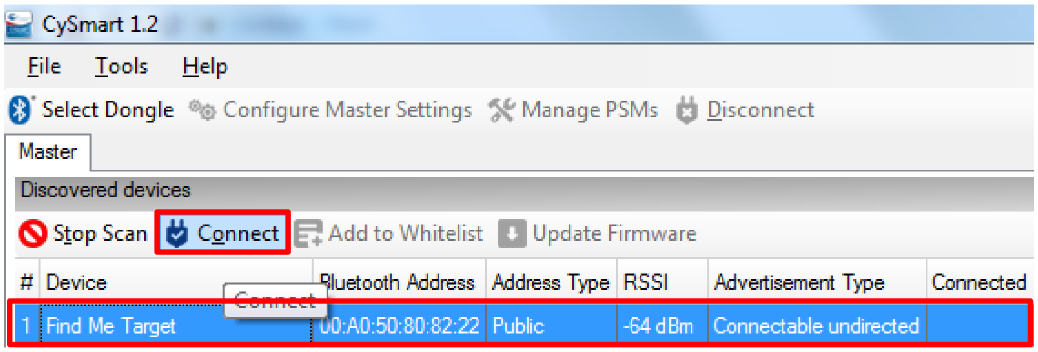
Figure 55. CySmart Device Discovery


CySmart 1.2
File Tools Help
Select Dongle Configure Master Settings
Master
Discovered devices
Start Scan Connect Add to Whitelist
# Device Bluetooth Address
Start Scan
CySmart 1.2
File Tools Help
Select Dongle Configure Master Settings Manage PSMs Disconnect
Master
Discovered devices
Stop Scan Connect Add to Whitelist Update Firmware

| # | Device | Bluetooth Address | Address Type | RSSI | Advertisement Type | Connected |
|---|---|---|---|---|---|---|
| 1 | Find Me Target | 00:A0:50:80:82:22 | Public | -64 dBm | Connectable undirected | |

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 56.

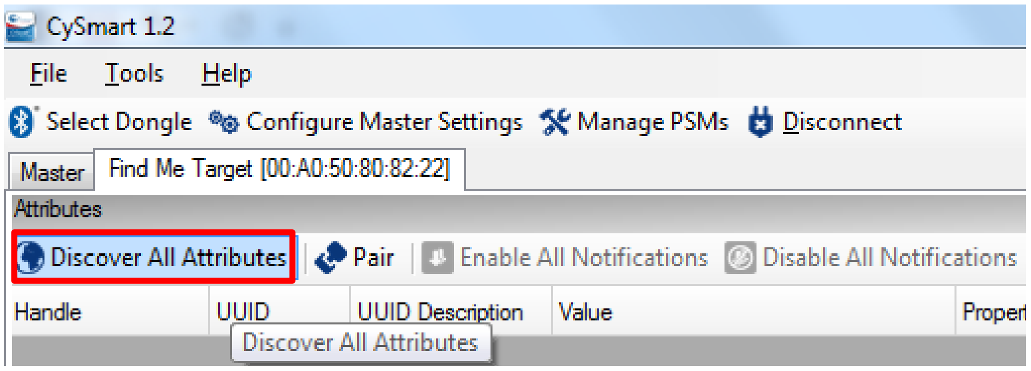
Figure 56. CySmart Device Connection


CySmart 1.2
File Tools Help
Select Dongle Configure Master Settings Manage PSMs Disconnect
Master
Discovered devices
Stop Scan Connect Add to Whitelist Update Firmware
Connect

| # | Device | Bluetooth Address | Address Type | RSSI | Advertisement Type | Connected |
|---|---|---|---|---|---|---|
| 1 | Find Me Target | 00:A0:50:80:82:22 | Public | -64 dBm | Connectable undirected | |

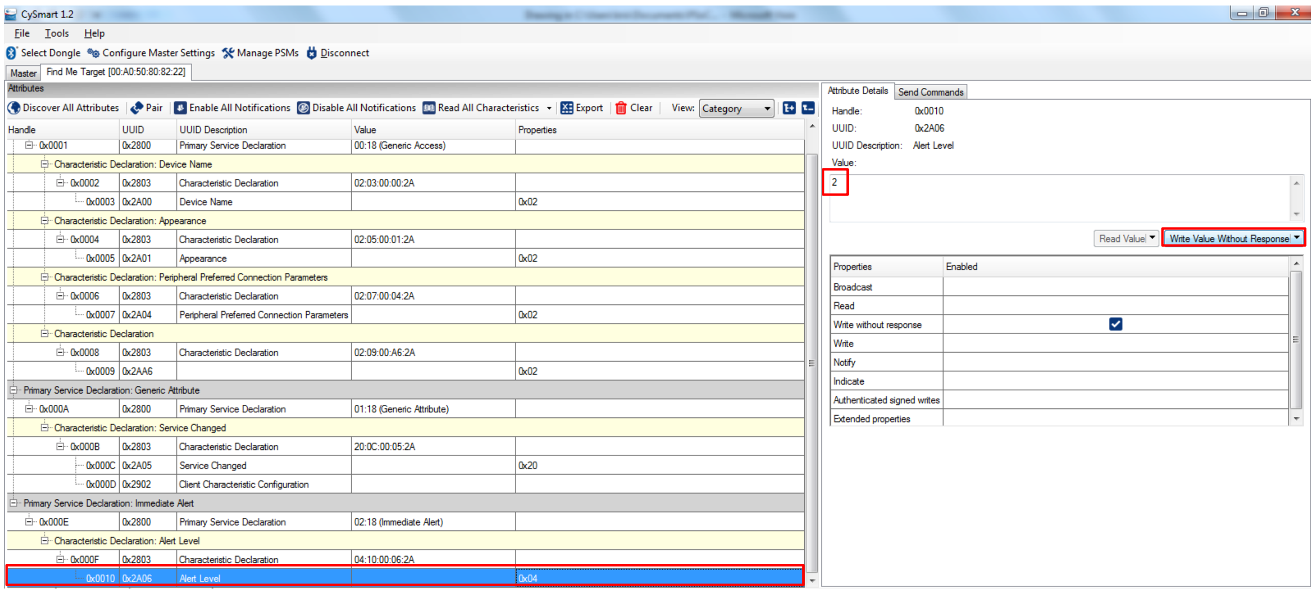
7. Once connected, discover all the Attributes on your design from the CySmart Host Emulation Tool, as shown in Figure 57.

Figure 57. CySmart Attribute Discovery



8. Write a value of 0, 1, or 2 to the Alert Level Characteristic under the Immediate Alert Service, as Figure 58 shows. Observe the state of the LED on your device change per your Alert Level Characteristic configuration.

Figure 58. Testing with CySmart Host Emulation Tool



| Handle | UUID | UUID Description | Value | Properties |
|---|---|---|---|---|
| 0x0001 | 0x2800 | Primary Service Declaration | 00:18 (Generic Access) | |
| Characteristic Declaration: Device Name | | | | |
| 0x0002 | 0x2803 | Characteristic Declaration | 02:03:00:00:2A | |
| 0x0003 | 0x2A00 | Device Name | | 0x02 |
| Characteristic Declaration: Appearance | | | | |
| 0x0004 | 0x2803 | Characteristic Declaration | 02:05:00:01:2A | |
| 0x0005 | 0x2A01 | Appearance | | 0x02 |
| Characteristic Declaration: Peripheral Preferred Connection Parameters | | | | |
| 0x0006 | 0x2803 | Characteristic Declaration | 02:07:00:04:2A | |
| 0x0007 | 0x2A04 | Peripheral Preferred Connection Parameters | | 0x02 |
| Characteristic Declaration | | | | |
| 0x0008 | 0x2803 | Characteristic Declaration | 02:09:00:A6:2A | |
| 0x0009 | 0x2AA6 | | | 0x02 |
| Primary Service Declaration: Generic Attribute | | | | |
| 0x000A | 0x2800 | Primary Service Declaration | 01:18 (Generic Attribute) | |
| Characteristic Declaration: Service Changed | | | | |
| 0x000B | 0x2803 | Characteristic Declaration | 20:0C:00:05:2A | |
| 0x000C | 0x2A05 | Service Changed | | 0x20 |
| 0x000D | 0x2902 | Client Characteristic Configuration | | |
| Primary Service Declaration: Immediate Alert | | | | |
| 0x000E | 0x2800 | Primary Service Declaration | 02:18 (Immediate Alert) | |
| Characteristic Declaration: Alert Level | | | | |
| 0x000F | 0x2803 | Characteristic Declaration | 04:10:00:06:2A | |
| 0x0010 | 0x2A06 | Alert Level | | 0x04 |

Handle: 0x0010
UUID: 0x2A06
UUID Description: Alert Level
Value:
2

| Properties | Enabled |
|---|---|
| Broadcast | |
| Read | |
| Write without response | ✓ |
| Write | |
| Notify | |
| Indicate | |
| Authenticated signed writes | |
| Extended properties | |

## 8 Summary

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

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

## 9 Related Application Notes

AN94020 - Getting Started with PRoC BLE

AN96841 - Getting Started With EZ-BL PRoC Module

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

AN95089 - PSoC 4/PRoC BLE Crystal Oscillator Selection and Tuning Techniques

AN91445 - Antenna Design Guide

AN97060 - PSoC 4 BLE and PRoC BLE - Over-The-Air (OTA) Device Firmware Upgrade (DFU) Guide

## About the Author

| | |
|---|---|
| Name: | Krishnaprasad M V (KRIS). |
| Title: | Applications Engineer Senior Staff |
| Background: | Krishnaprasad has a BSEE from National Institute of Engineering, Mysore, India. |

## 10 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** | **CYBL10xxx*** ** | **CY8C41x8-BL** | **CY8C42x8-BL** |
| BLE Subsystem | BLE radio and link-layer hardware blocks with Bluetooth 4.1-compatible protocol stack | BLE radio and link-layer hardware blocks with Bluetooth 4.1-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, 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 | 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 | 256 KB | 256 KB | 256 KB |
| SRAM | 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 |
| CapSense | 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 |
| 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, 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 | 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 | 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 | 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 |
| Low-Power Modes | Deep-Sleep mode at 1.3 µA<br>Hibernate mode at 150 nA<br>Stop mode at 60 nA | Deep-Sleep mode at 1.3 µA<br>Hibernate mode at 150 nA<br>Stop mode at 60 nA | Deep-Sleep mode at 1.3 µA<br>Hibernate mode at 150 nA<br>Stop mode at 60 nA | Deep-Sleep mode at 1.3 µA<br>Hibernate mode at 150 nA<br>Stop mode at 60 nA | Deep-Sleep mode at 1.3 µA<br>Hibernate mode at 150 nA<br>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 |

| Features | Device Family | | | | |
|---|---|---|---|---|---|
| | **CY8C41x7-BLxxx** | **CY8C42x7-BLxxx** | **CYBL10xxx*** ** | **CY8C41x8-BL** | **CY8C42x8-BL** |
| Serial Communication | 2 independent serial communication blocks (SCBs) with programmable I$^2$C, SPI, or UART | 2 independent SCBs with programmable I$^2$C, SPI, or UART | 2 independent SCBs with programmable I$^2$C, SPI, or UART | 2 independent serial communication blocks (SCBs) with programmable I$^2$C, SPI, or UART | 2 independent SCBs with programmable I$^2$C, SPI, or UART |
| Timer Counter Pulse-Width Modulator (TCPWM) | 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 | 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$^2$S, PWM) | None | Yes (UDB-based digital peripherals 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<br>32-kHz ILO<br>24-MHz ECO<br>32-kHz WCO | 3-MHz to 48-MHz IMO<br>32-kHz ILO<br>24-MHz ECO<br>32-kHz WCO | 3-MHz to 48-MHz IMO<br>32-kHz ILO<br>24-MHz ECO<br>32-kHz WCO | 3-MHz to 24-MHz IMO<br>32-kHz ILO<br>24-MHz ECO<br>32-kHz WCO | 3-MHz to 48-MHz IMO<br>32-kHz ILO<br>24-MHz ECO<br>32-kHz WCO |
| Power Supply Monitoring | Power-on reset (POR)<br>Brown-out detection (BOD)<br>Low-voltage detection (LVD) | POR<br>BOD<br>LVD | POR<br>BOD<br>LVD | Power-on reset (POR)<br>Brown-out detection (BOD)<br>Low-voltage detection (LVD) | POR<br>BOD<br>LVD |
| Package | 56-QFN (7.0 × 7.0 × 0.6 mm) and<br>68-WLCSP (3.52 × 3.91 × 0.55 mm) | 56-QFN (7.0 × 7.0 × 0.6 mm) and<br>68-WLCSP (3.52 × 3.91 × 0.55 mm) | 56-QFN* (7.0 × 7.0 × 0.6 mm) and<br>76-WLCSP (4.04 × 3.87 × 0.55 mm) | 56-QFN* (7.0 × 7.0 × 0.6 mm) and<br>76-WLCSP (4.04 × 3.87 × 0.55 mm) | 56-QFN* (7.0 × 7.0 × 0.6 mm) and<br>76-WLCSP (4.04 × 3.87 × 0.55 mm) |
| DMA | None | None | Up to 8 channels | Up to 8 channels** | Up to 8 channels** |

* = CY8C41x8-BL and CY8C42x8-BL (256K FLASH) QFN packages are pin-to-pin compatible with CY8C41x7-BL and CY8C42x7-BL (128K FLASH) QFN package, respectively.
** = CY8C41x8-BL and CY8C42x8-BL family has two sub-families –BL4xx and –BL5xx. The –BL4xx series does not support Bluetooth 4.2 link layer privacy, link layer data length extension, and DMA engine; the –BL5xx series does.
*** = CY8CBL10xxx PRoC family of devices have multiple sub-families and the features described here are for the super set device. See AN94020 - Getting Started with PRoC BLE for details.

# 11 Appendix B: Cypress Terms of Art

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

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

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

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

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

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

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

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

# 12 Appendix C: Cypress BLE Development Tools

## PSoC 4 BLE Pioneer Kit

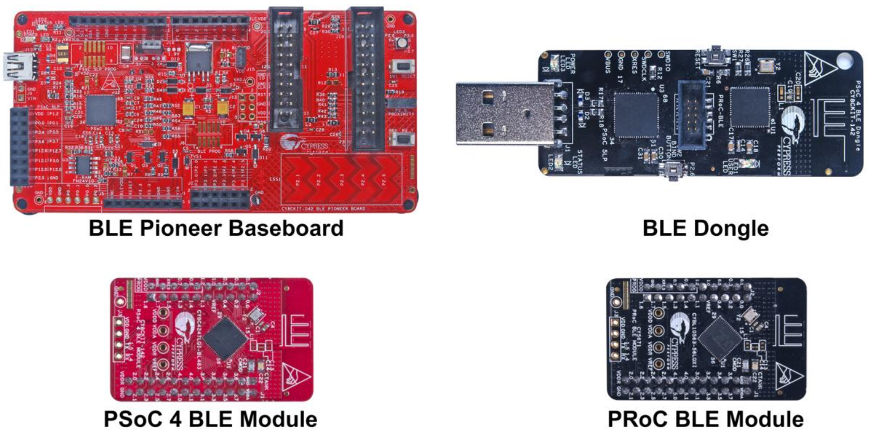
The BLE Pioneer Kit shown in Figure 59 is a BLE development kit from Cypress that supports both PSoC 4 BLE and PRoC BLE family of devices. This kit consists of pluggable PSoC 4 BLE (and PRoC BLE) modules that connect to a BLE Pioneer Baseboard. The kit can be powered either with a coin-cell battery or through the USB interface.

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

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

The kit consists of a set of BLE example projects and documentation that help you get started on developing your own BLE applications. Visit BLE Pioneer Kit webpage to get the latest updates on the kit and download the kit design, example projects, and documentation files.

Figure 59. BLE Pioneer Kit



## 12.1 CySmart Host Emulation Tool

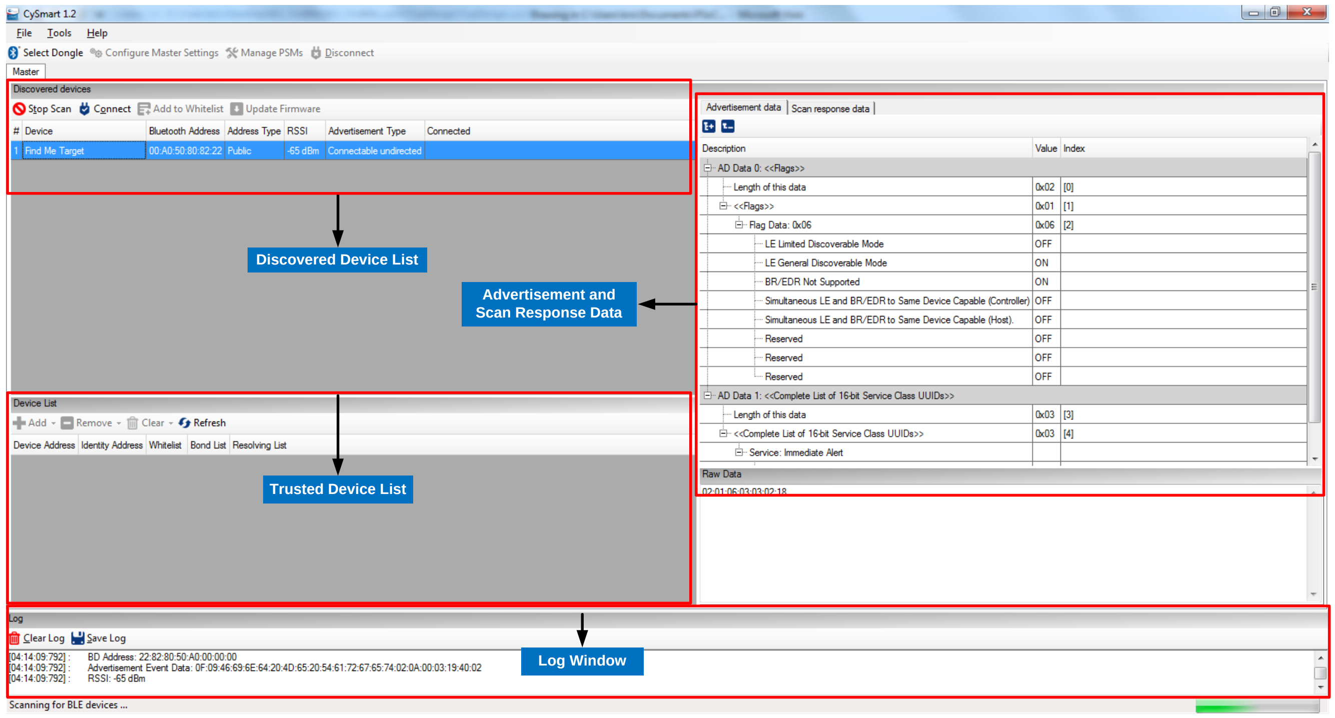
The CySmart Host Emulation Tool is a Windows application that emulates a BLE Central device using the BLE Pioneer Kit's BLE Dongle; see Figure 19. It is installed as part of the BLE Pioneer Kit installation and can be launched from right-click options in the BLE Component. It provides a platform for 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 60 and Figure 61 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 60. CySmart Host Emulation Tool Master Device Tab



| # | Device | Bluetooth Address | Address Type | RSSI | Advertisement Type | Connected |
|---|---|---|---|---|---|---|
| 1 | Find Me Target | 00:A0:50:80:82:22 | Public | -65 dBm | Connectable undirected | |

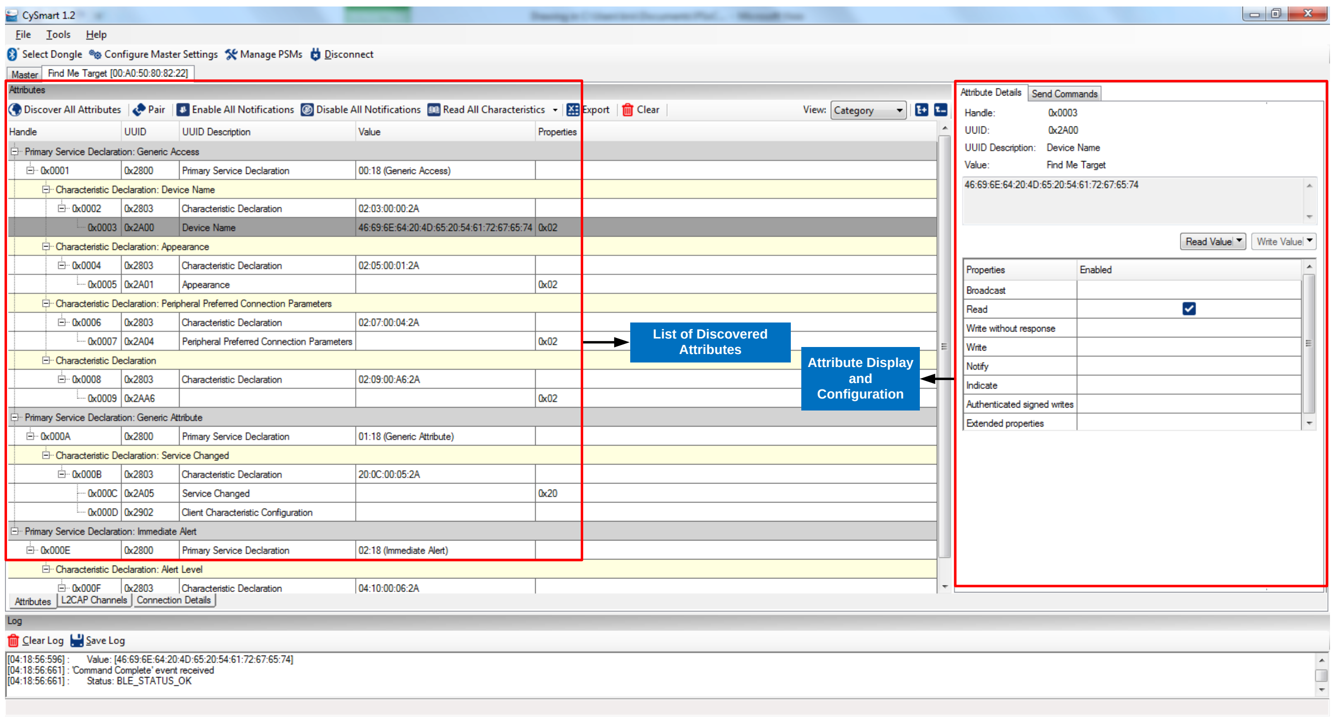
| Device Address | Identity Address | Whitelist | Bond List | Resolving List |
|---|---|---|---|---|

| Description | Value | Index |
|---|---|---|
| AD Data 0: <<Flags>> | | |
| Length of this data | 0x02 | [0] |
| <<Flags>> | 0x01 | [1] |
| Flag Data: 0x06 | 0x06 | [2] |
| LE Limited Discoverable Mode | OFF | |
| LE General Discoverable Mode | ON | |
| BR/EDR Not Supported | ON | |
| Simultaneous LE and BR/EDR to Same Device Capable (Controller) | OFF | |
| Simultaneous LE and BR/EDR to Same Device Capable (Host). | OFF | |
| Reserved | OFF | |
| Reserved | OFF | |
| Reserved | OFF | |
| AD Data 1: <<Complete List of 16-bit Service Class UUIDs>> | | |
| Length of this data | 0x03 | [3] |
| <<Complete List of 16-bit Service Class UUIDs>> | 0x03 | [4] |
| Service: Immediate Alert | | |

```
02:01:06:03:03:02:18
```

```
[04:14:09:792] :     BD Address: 22:82:80:50:A0:00:00:00
[04:14:09:792] :     Advertisement Event Data: 0F:09:46:69:6E:64:20:4D:65:20:54:61:72:67:65:74:02:0A:00:03:19:40:02
[04:14:09:792] :     RSSI: -65 dBm
```

Figure 61. CySmart Host Emulation Tool Peripheral Device Attributes Tab



| Handle | UUID | UUID Description | Value | Properties |
|---|---|---|---|---|
| Primary Service Declaration: Generic Access | | | | |
| 0x0001 | 0x2800 | Primary Service Declaration | 00:18 (Generic Access) | |
| Characteristic Declaration: Device Name | | | | |
| 0x0002 | 0x2803 | Characteristic Declaration | 02:03:00:00:2A | |
| 0x0003 | 0x2A00 | Device Name | 46:69:6E:64:20:4D:65:20:54:61:72:67:65:74 | 0x02 |
| Characteristic Declaration: Appearance | | | | |
| 0x0004 | 0x2803 | Characteristic Declaration | 02:05:00:01:2A | |
| 0x0005 | 0x2A01 | Appearance | | 0x02 |
| Characteristic Declaration: Peripheral Preferred Connection Parameters | | | | |
| 0x0006 | 0x2803 | Characteristic Declaration | 02:07:00:04:2A | |
| 0x0007 | 0x2A04 | Peripheral Preferred Connection Parameters | | 0x02 |
| Characteristic Declaration | | | | |
| 0x0008 | 0x2803 | Characteristic Declaration | 02:09:00:A6:2A | |
| 0x0009 | 0x2AA6 | | | 0x02 |
| Primary Service Declaration: Generic Attribute | | | | |
| 0x000A | 0x2800 | Primary Service Declaration | 01:18 (Generic Attribute) | |
| Characteristic Declaration: Service Changed | | | | |
| 0x000B | 0x2803 | Characteristic Declaration | 20:0C:00:05:2A | |
| 0x000C | 0x2A05 | Service Changed | | 0x20 |
| 0x000D | 0x2902 | Client Characteristic Configuration | | |
| Primary Service Declaration: Immediate Alert | | | | |
| 0x000E | 0x2800 | Primary Service Declaration | 02:18 (Immediate Alert) | |
| Characteristic Declaration: Alert Level | | | | |
| 0x000F | 0x2803 | Characteristic Declaration | 04:10:00:06:2A | |

| | |
|---|---|
| Handle: | 0x0003 |
| UUID: | 0x2A00 |
| UUID Description: | Device Name |
| Value: | Find Me Target |

```
46:69:6E:64:20:4D:65:20:54:61:72:67:65:74
```

| Properties | Enabled |
|---|---|
| Broadcast | |
| Read | ✓ |
| Write without response | |
| Write | |
| Notify | |
| Indicate | |
| Authenticated signed writes | |
| Extended properties | |

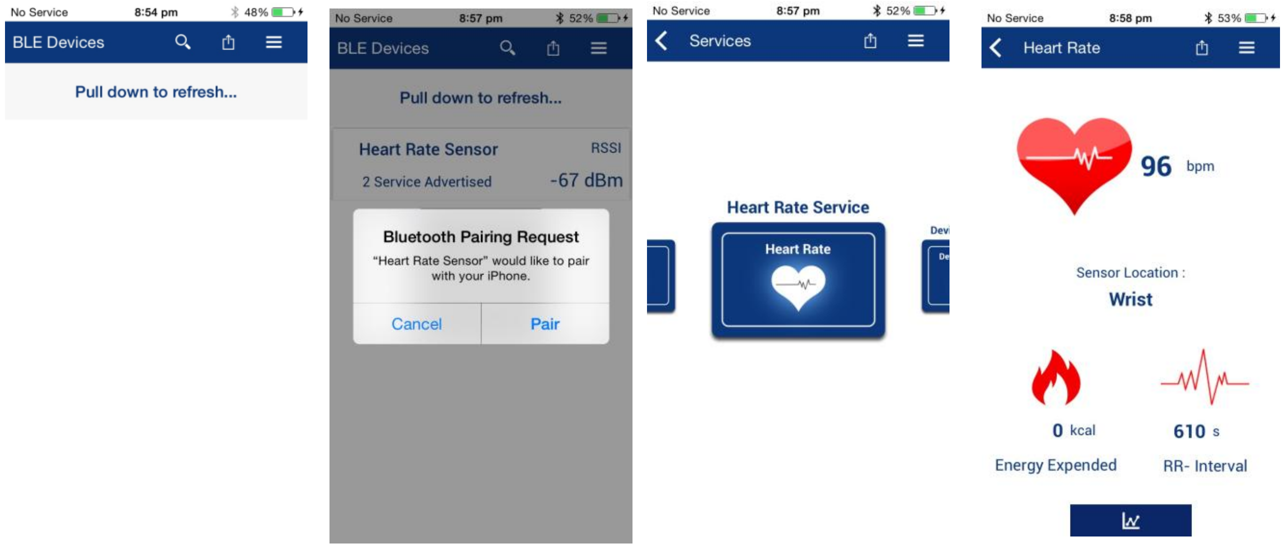
```
[04:18:56:596] :     Value: [46:69:6E:64:20:4D:65:20:54:61:72:67:65:74]
[04:18:56:661] : 'Command Complete' event received
[04:18:56:661] :     Status: BLE_STATUS_OK
```

## 12.2 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.

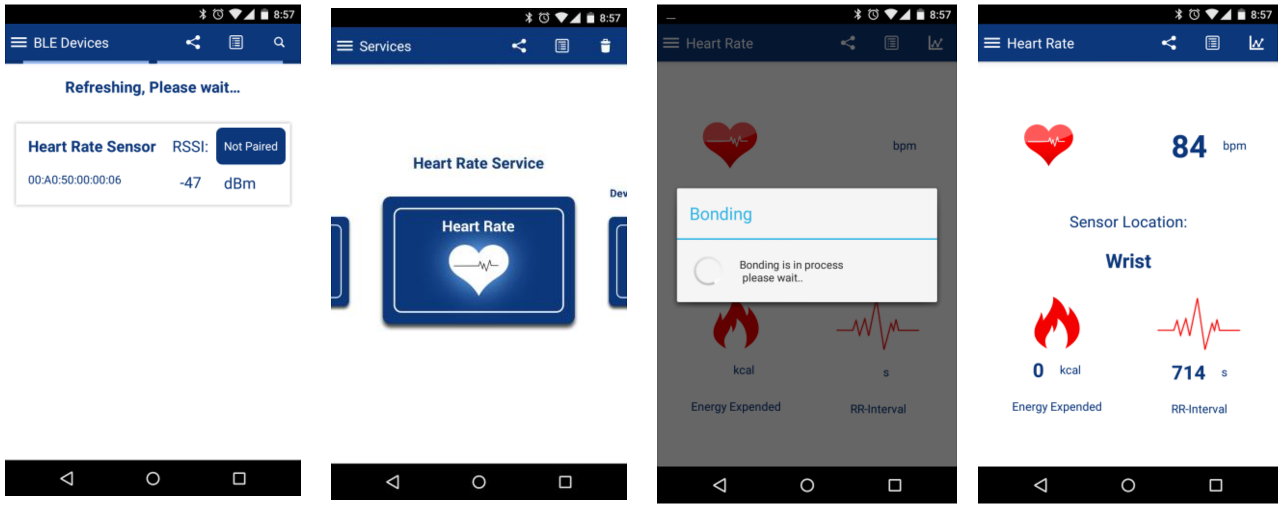
The mobile app supports SIG-adopted BLE standard Profiles through an intuitive GUI and abstracts the underlying BLE Service and Characteristic details. In addition to the BLE standard Profiles, the app demonstrates a custom Profile implementation using Cypress's LED and CapSense demo examples. Figure 62 and Figure 63 show 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 62. CySmart iOS App Heart Rate Profile Example



"Heart Rate Sensor" would like to pair with your iPhone.

Figure 63. CySmart Android App Heart Rate Profile Example



Bonding is in process please wait..

# 13 Appendix D: PSoC 4 BLE Device

## 13.1 Bluetooth Low Energy Subsystem (BLESS)

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

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

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

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

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

### 13.1.1 Deep-Sleep Mode

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

### 13.1.2 Sleep Mode

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

### 13.1.3 Idle Mode

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

### 13.1.4 Transmit Mode

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

### 13.1.5 Receive Mode

This mode enables the BLESS to move into the receive state to perform BLE-specific receiver operations. RF-PHY translates the 1-Mbps data received from the RF analog block and forwards it to the link-layer controller after demodulation.A summary of the BLESS power modes and operational sub-blocks is shown in Table 3.

Table 3. BLESS Power Modes

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

## 13.2 ARM Cortex-M0, Memory, and DMA

PSoC 4 BLE has a 32-bit ARM Cortex-M0 CPU, capable of operating at a maximum frequency of 48 MHz, providing a 43-DMIPS performance. The CPU supports single-cycle 32-bit multiplication.

PSoC 4 BLE has 16 KB/32 KB of SRAM and 128 KB/256 of flash memory that can service most of the BLE application use; the flash includes a read accelerator. The device also provides 512 bytes of supervisory flash area for you to store user-specific data such as BLE device address and encryption keys.

A DMA engine, with eight channels, is provided on PSoC 4XX8-BL family of devices that can do 32-bit transfers among peripherals, among memory, or between peripherals and memory.

## 13.3 Programmable Digital Peripherals

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

### 13.3.1 Programmable SCBs

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

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

For more information, refer to the PSoC 4 SCB Component datasheet.

### 13.3.2 Programmable TCPWMs

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

For more information, refer to the PSoC 4 TCPWM Component datasheet.

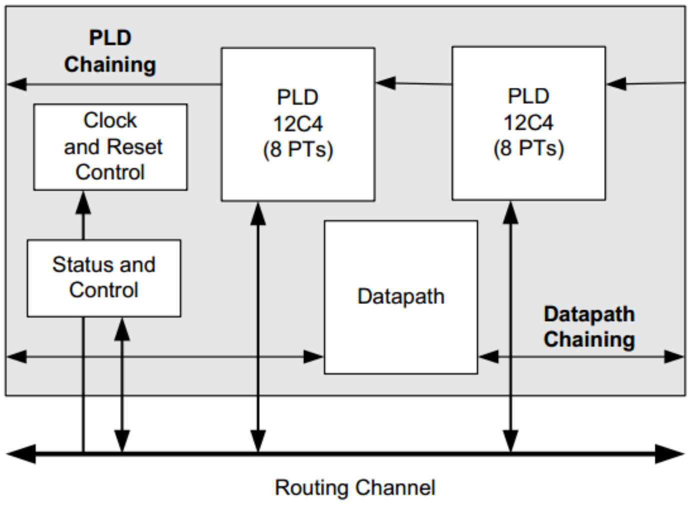
### 13.3.3 Universal Digital Blocks

UDBs are programmable logic blocks that provide functionalities similar to CPLD and FPGA blocks, as Figure 64 shows. UDBs allow you to create a variety of digital functions such as timer, counter, PWM, pseudo random sequence (PRS), CRC, shift register, SPI, UART, I2S, and custom combinational and sequential logic circuits.

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

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

Figure 64. Universal Digital Block Diagram



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

For more information, refer to the following application notes.

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

#### 13.3.4 Applications

The use of programmable digital peripherals in BLE applications is shown in Table 4.

Table 4. Applications of Programmable Digital Peripherals

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

## 13.4 Programmable Analog

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

#### 13.4.1 Continuous Time Block mini (CTBm)

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

For more information, refer to the following Component datasheets:

- PSoC 4 Opamp
- PSoC 4 Voltage Comparator

### 13.4.2 SAR ADC with Hardware Sequencer

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

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

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

For more information, refer to the PSoC 4 SAR ADC Component datasheet.

### 13.4.3 Low-Power Comparators

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

For more information, refer to the PSoC 4 Low-Power Comparator Component datasheet.

### 13.4.4 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 long lifetimes.

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

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

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

For more information, see the PSoC 4 CapSense Design Guide.

### 13.4.5 Segment LCD Direct Driver

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

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

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

### 13.4.6 Applications

The use of programmable analog peripherals in different BLE applications is listed in Table 5.

Table 5. Applications of Programmable Analog Peripherals

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

## 13.5 System-Wide Resources

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

### 13.5.1 Low-Leakage Power Modes

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

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

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

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

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

Table 6. PSoC 4 BLE Power Modes

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

### 13.5.2 Power Supply and Monitoring

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

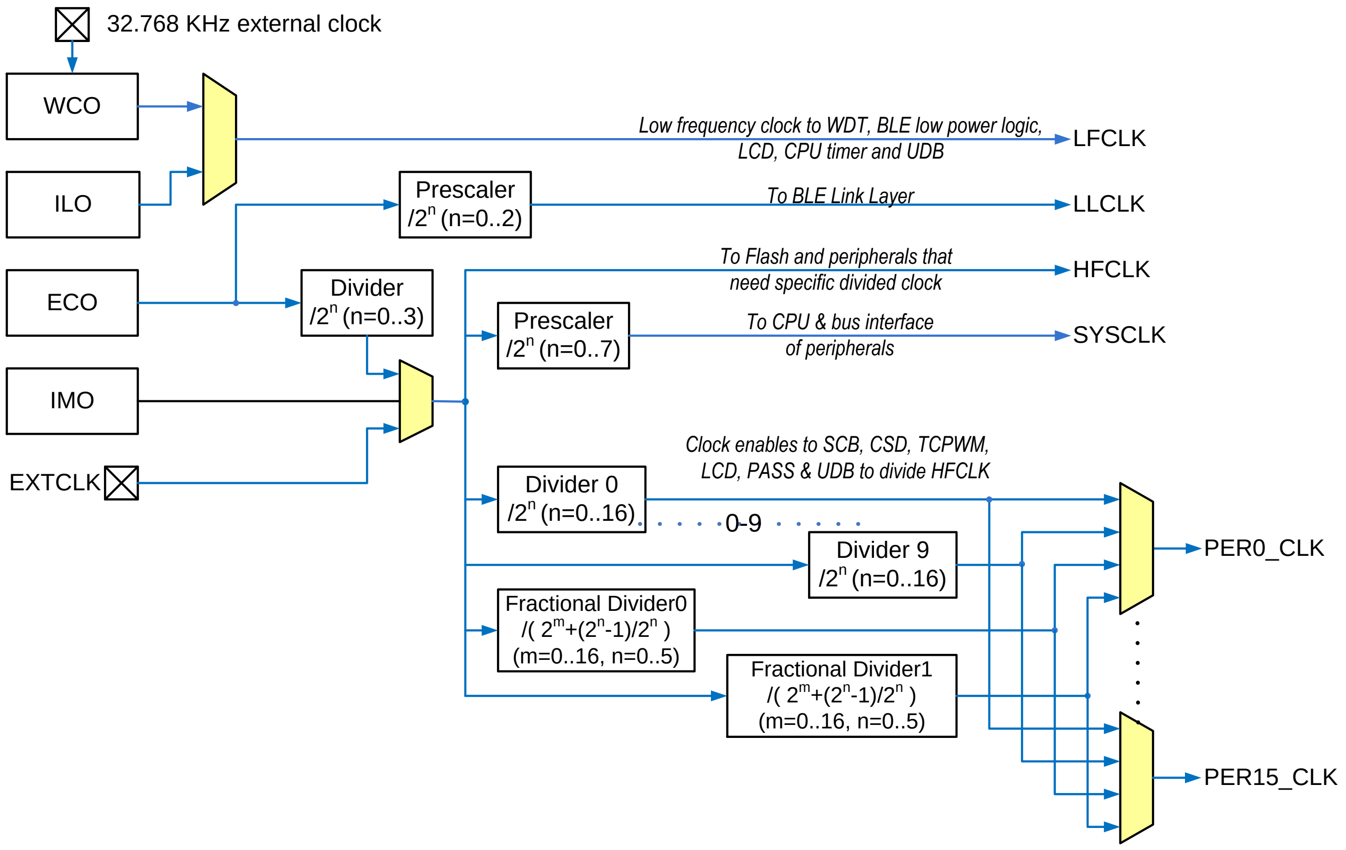
### 13.5.3 Clocking System

PSoC 4 BLE has the following clock sources:

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

Figure 65 shows the clocking architecture of a PSoC 4 BLE device.

Figure 65. PSoC 4 BLE Clocking System


32.768 KHz external clock
WCO
ILO
ECO
IMO
EXTCLK
Low frequency clock to WDT, BLE low power logic, LCD, CPU timer and UDB
LFCLK
Prescaler /2^n (n=0..2)
To BLE Link Layer
LLCLK
Divider /2^n (n=0..3)
To Flash and peripherals that need specific divided clock
HFCLK
Prescaler /2^n (n=0..7)
To CPU & bus interface of peripherals
SYSCLK
Clock enables to SCB, CSD, TCPWM, LCD, PASS & UDB to divide HFCLK
Divider 0 /2^n (n=0..16)
0-9
Divider 9 /2^n (n=0..16)
Fractional Divider0 /( 2^m+(2^n-1)/2^n ) (m=0..16, n=0..5)
Fractional Divider1 /( 2^m+(2^n-1)/2^n ) (m=0..16, n=0..5)
PER0_CLK
PER15_CLK

### 13.5.4 Device Security

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

## 13.6 Programmable GPIOs

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

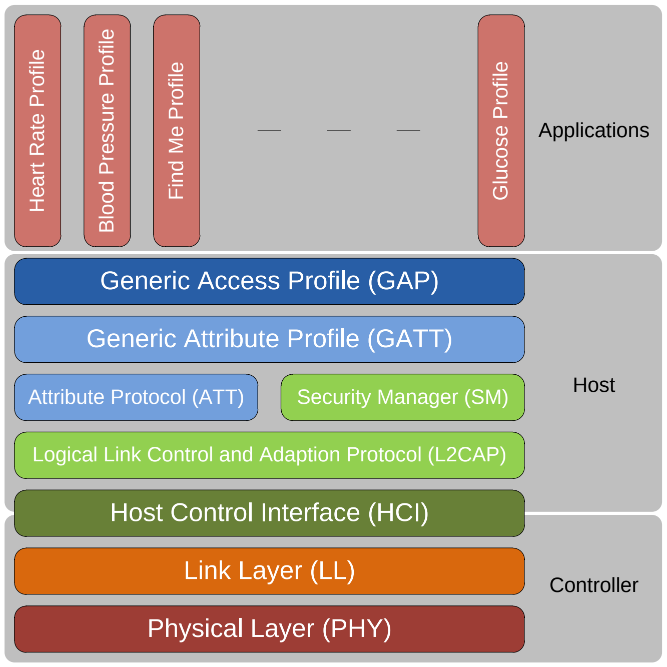
PSoC 4 BLE offers an intelligent routing system that gives multiple choices for connecting an internal signal to a GPIO. This flexible routing simplifies circuit design and board layout.

# 14 Appendix E: BLE Protocol

## 14.1 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 66 shows the BLE protocol stack.

Figure 66. 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.

## 14.2 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.

## 14.3 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.

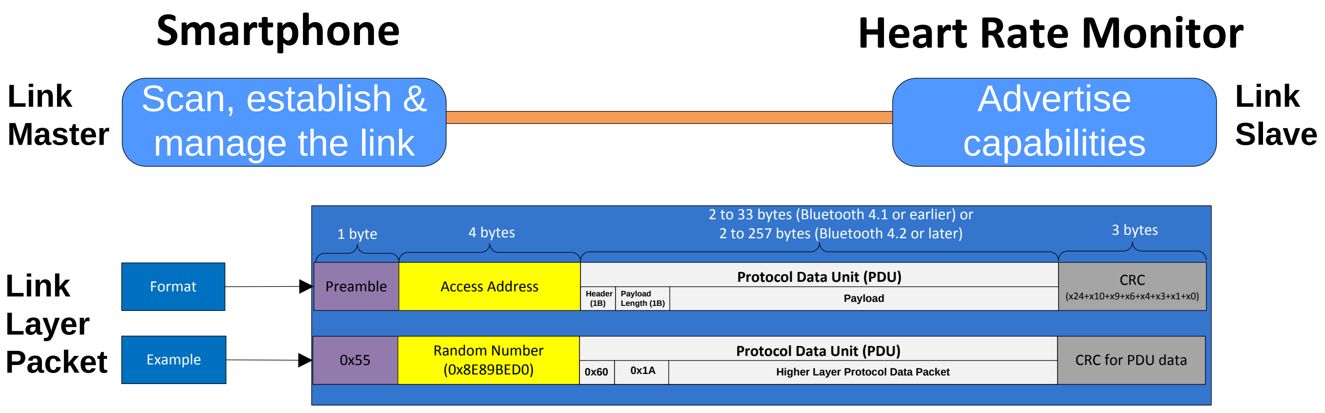
PSoC/PRoC BLE devices can operate in either configuration.

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 67). 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 67 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 67. BLE Link Layer Protocol



## 14.4 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.

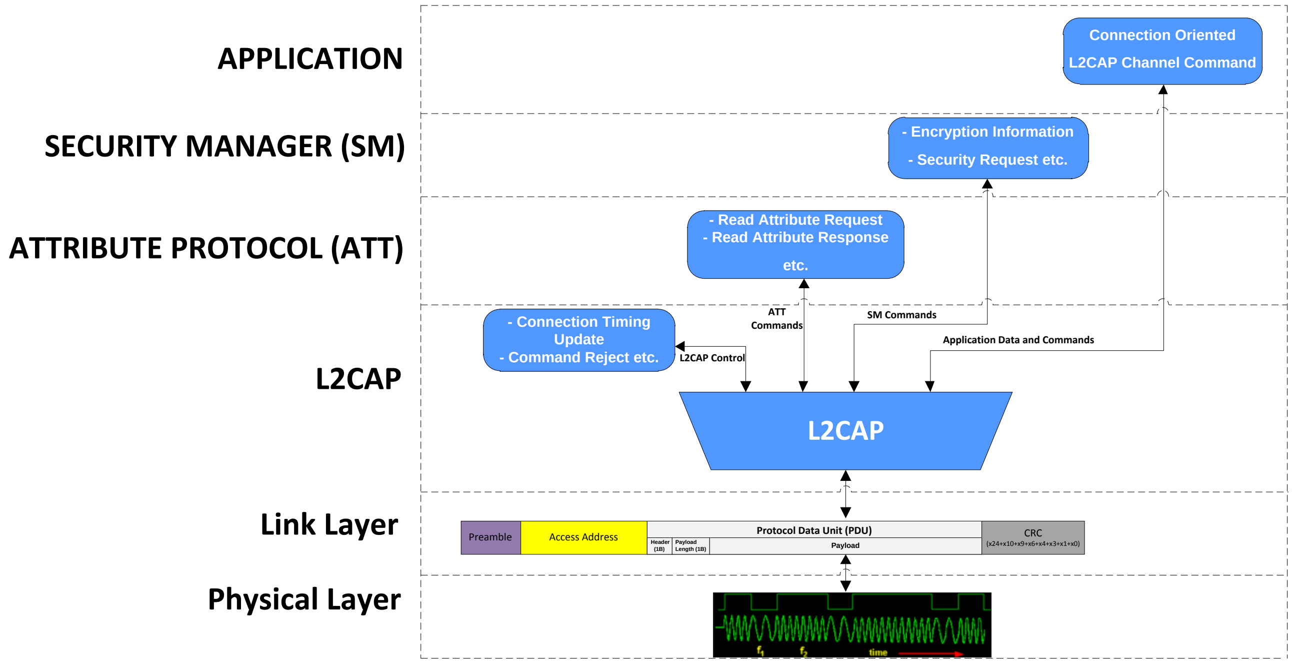
In PSoC/PRoC BLE devices, the HCI is just a firmware protocol layer that passes the messages and events between the controller and the host.

## 14.5 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 68. 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 68. BLE L2CAP Layer



## 14.6 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.

## 14.7 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).

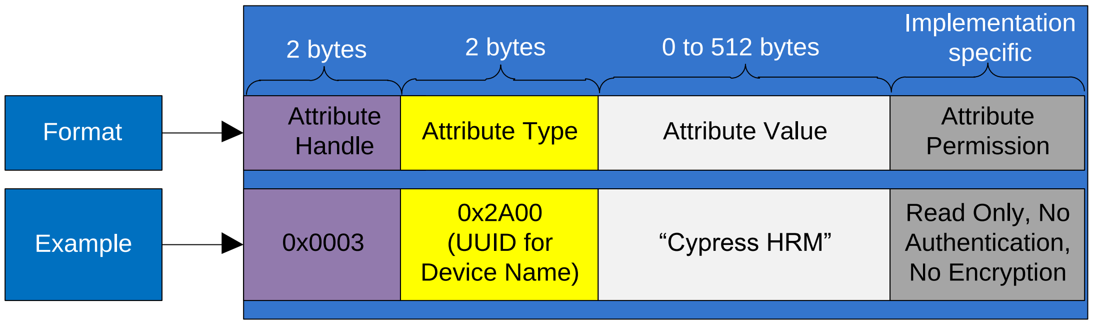
An Attribute is the fundamental data container in the ATT/GATT layer, which consists of the following:

- **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 webpage 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 69 shows the structure of a Device Name Attribute as an example.

Figure 69. Attribute Format Example



### 14.7.1 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:

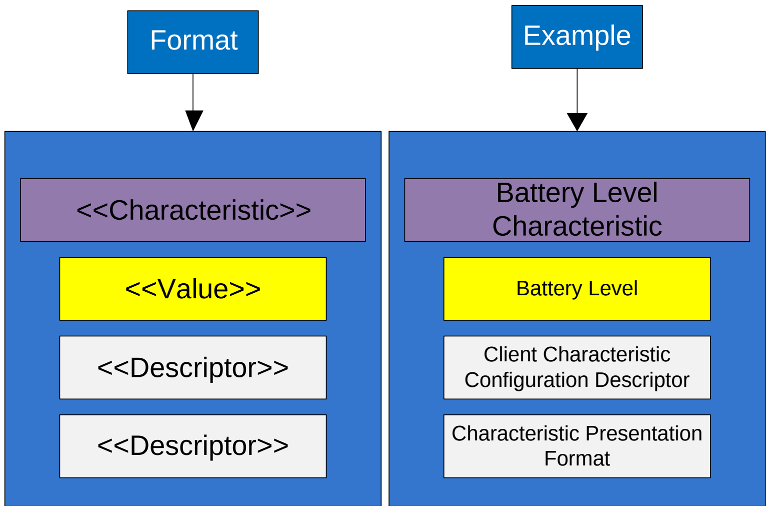
- **Characteristic:** A collection of Attributes that exposes the system information or meaningful data. A Characteristic consists of the following Attributes:
  - Characteristic Declaration Attribute: This defines the beginning of a Characteristic.
  - Characteristic Value Attribute: This holds the 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 70 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 70.
- 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 70) 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 70. Characteristic Format and Example

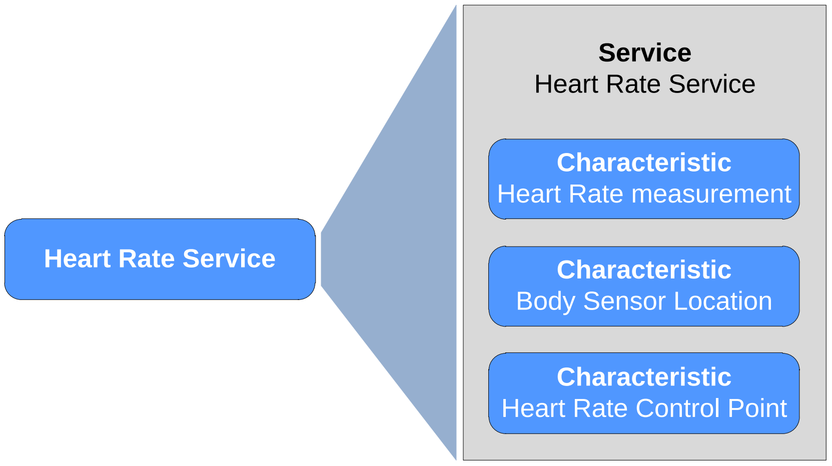

Format
Example
<<Characteristic>>
<<Value>>
<<Descriptor>>
<<Descriptor>>
Battery Level Characteristic
Battery Level
Client Characteristic Configuration Descriptor
Characteristic Presentation Format

- **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 71 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.

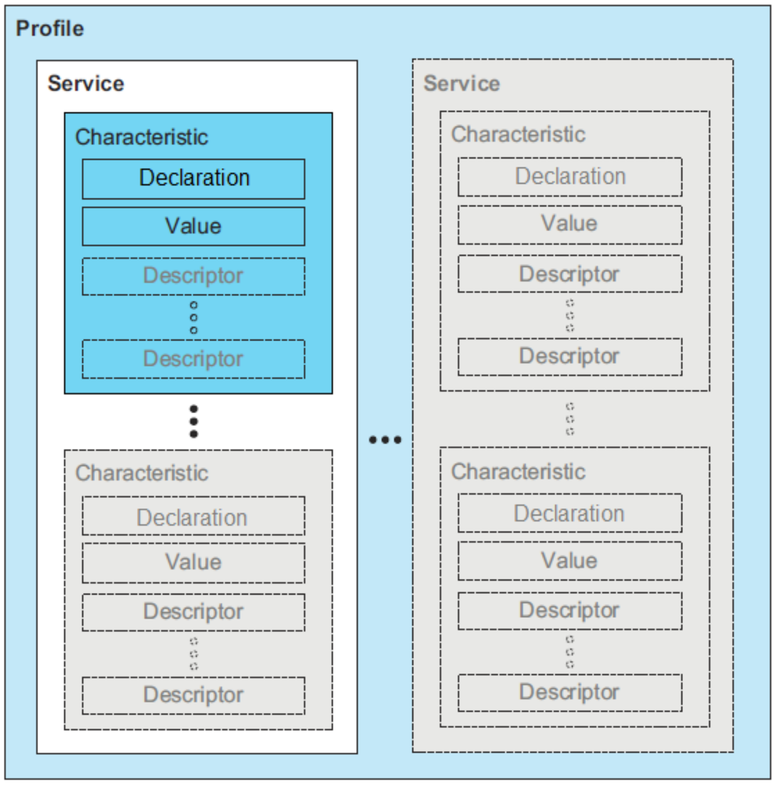
Figure 71. BLE Heart Rate Service Example



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

Figure 72. BLE Data Hierarchy*



* Image courtesy of Bluetooth SIG

### 14.7.2 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 70) 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.

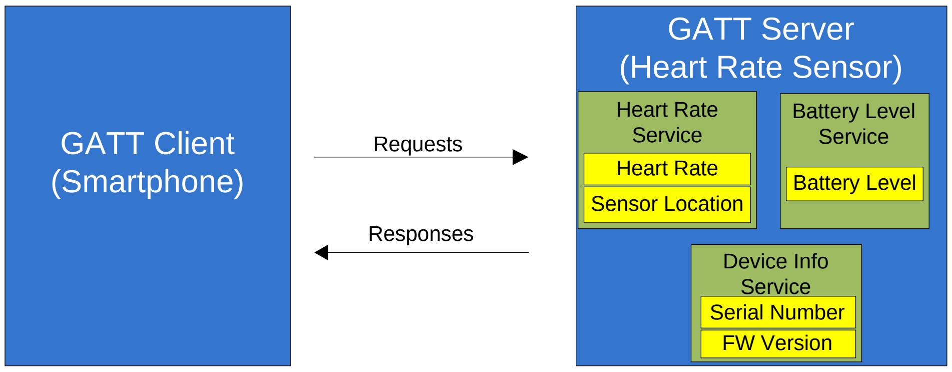
## 14.8 Generic Attribute Profile (GATT)

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 73 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 70.

Figure 73. 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.

## 14.9 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 75 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 74 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 74. BLE System Design

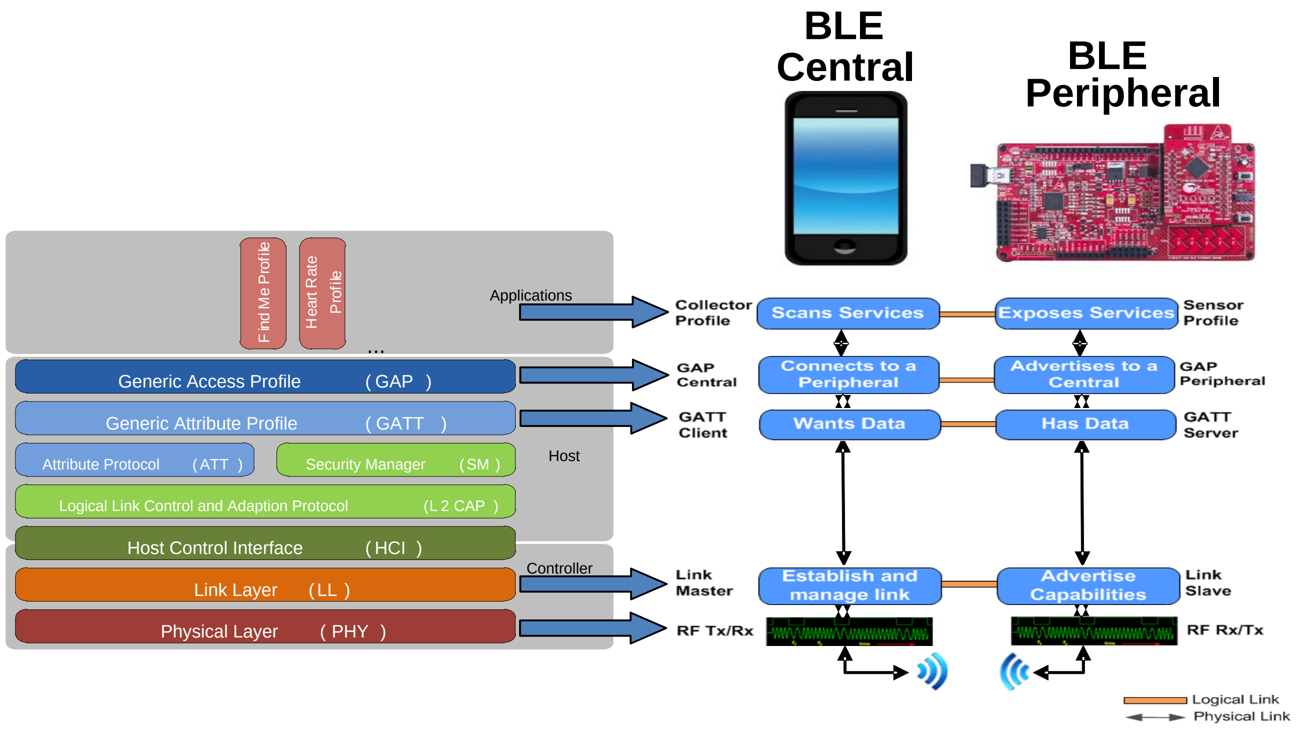
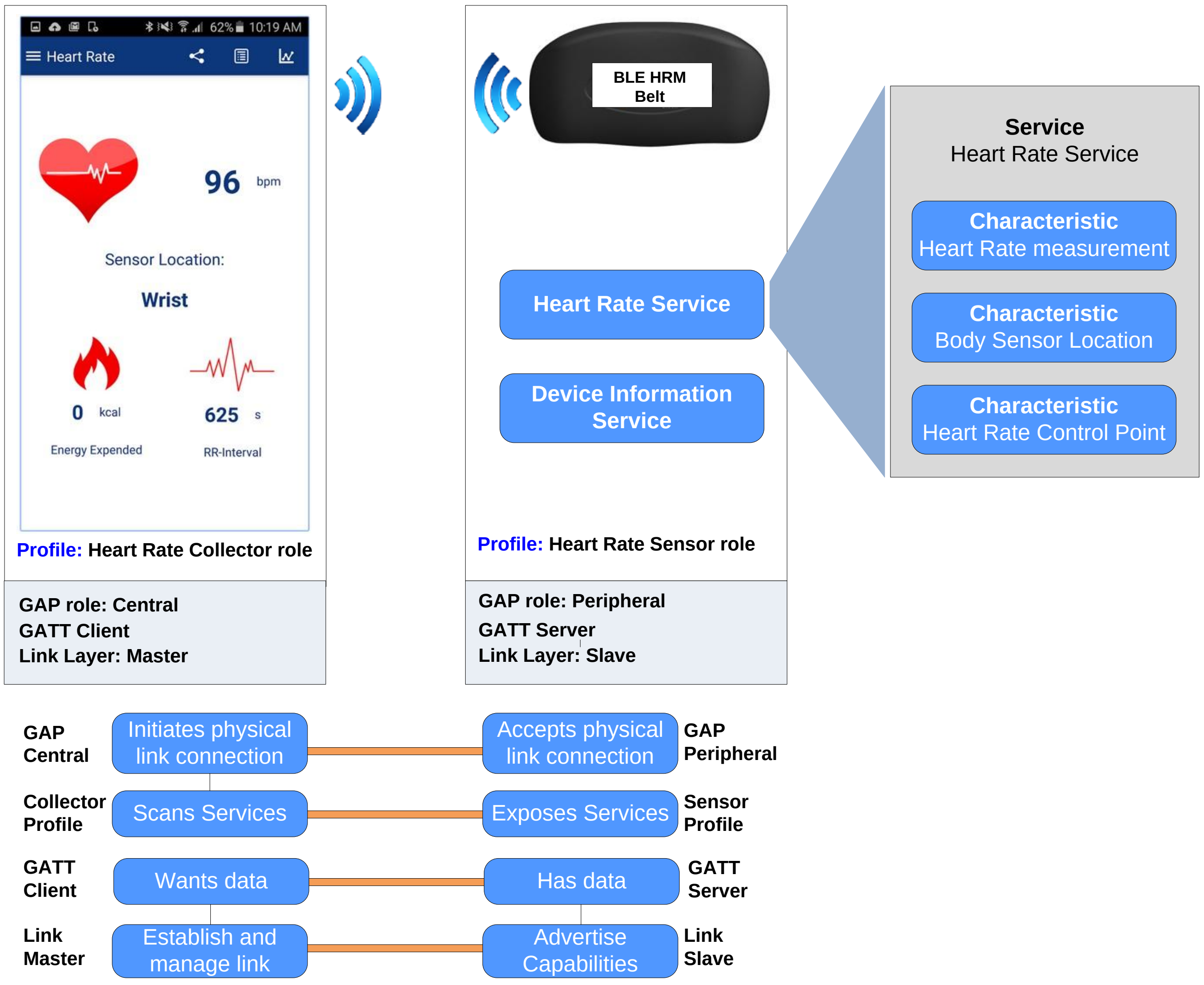


Figure 75 shows an example where a smartphone with a heart-rate app operates as a Central and a heart-rate sensor operates as a Peripheral. The heart-rate monitoring device implements the Heart-Rate Sensor Profile, while the smartphone receiving the data implements the Heart-Rate Collector Profile.

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 75. BLE Heart-Rate Monitor System



# Document History

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

Document Number: 001-91267

| Revision | ECN | Orig. of Change | Submission Date | Description of Change |
|---|---|---|---|---|
| ** | 4564230 | KRIS | 11/07/2014 | New Application Note |
| *A | 4567888 | KRIS | 11/12/2014 | Fixed hyperlinks and document formatting |
| *B | 4683692 | KRIS | 03/27/2015 | Major reformatting including moving the BLE Protocol and PSoC 4 BLE device details to Appendix.<br>Example design simplified and added Deep-Sleep low-power mode. |
| *C | 4758078 | KRIS | 05/11/2015 | Updated Software Version as “PSoC Creator™ 3.1 SP3” in page 1.<br>Replaced “PSoC Creator 3.1 SP1” with “PSoC Creator 3.1 SP3” in all instances across the document.<br>Updated My First PSoC 4 BLE Design:<br>Updated Stage 1: Create the Design:<br>Updated description.. |
| *D | 4767158 | ROIT | 05/15/2015 | Added support for PSoC 4 BLE 256K<br>Updated Software Versions as “PSoC Creator 3.2”<br>Updated BLE component GUI screenshots<br>Updated CySmart iOS/Android App screenshots<br>Updated Table 2 in Appendix A with WLCSP dimensions |
| *E | 4962932 | KRIS | 10/15/2015 | Rearranged/updated contents of multiple sections to address customer feedback<br>Added PSoC is More Than a BLE MCU section<br>Updated BLE Overview section to clarify GATT roles and data representation<br>Updated to support PSoC Creator 3.3 and CySmart 1.1 tool<br>Updated PSoC Creator and CySmart screenshots |
| *F | 5005244 | KRIS | 11/06/2015 | Added Bluetooth 4.2 device details<br>Updated to support PSoC Creator 3.3 SP1 and CySmart 1.2 tool |

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