Getting started with PSoC™ 63 MCU & Bluetooth® LE on ModusToolbox™

About this document

Scope and purpose

This document introduces you to the PSoC™ 63 MCU with Bluetooth® Low Energy connectivity, a dual-CPU Arm® Cortex®-M4 CPU at 150-MHz as the primary processor, an Arm® Cortex®-M0+ at 100-MHz that supports low-power operations and integrates a Bluetooth® Low Energy 5.2 system, the latest-generation of CAPSENSE™ technology, and a host of security features. This application note helps you get started with the PSoC™ 63 MCU with Bluetooth® Low Energy and with an overview of the device architecture, development kits, and software development tools. It shows how to create a simple Bluetooth® Low Energy application using Eclipse IDE for ModusToolbox™ and the AIROC™ Bluetooth® Host Software. It also guides you to more resources available online to accelerate your learning on PSoC™ 63 MCU.

Note: *AIROC™ CY8CKIT-062-BLE, CY8CKIT-063-BLE, and CYBLE-416045-EVAL are referred to as PSoC™ 6 Bluetooth® LE in this document.*

Intended audience

This document is intended for anyone who uses the PSoC™ 63 MCU with Bluetooth® Low Energy connectivity.

Table of contents

1 Introduction

PSoC™ 63 MCU with Bluetooth® LE connectivity, hereafter called PSoC™ 6 Bluetooth® LE, is an ultra-low-power PSoC™ device specifically designed for wearables and Internet of Things (IoT) products. It establishes a new low-power standard for today's "always-on" applications. The PSoC™ 6 Bluetooth® LE device is a programmable embedded system-on-chip that integrates the following on a single chip:

- Dual-CPU microcontroller: CM4 and CM0+
- Bluetooth® LE 5.2 subsystem
- Programmable analog and digital peripherals
- Up to 1 MB of flash and 288 KB of SRAM
- Fourth-generation CAPSENSE™ technology

PSoC™ 6 Bluetooth® LE is suitable for a variety of power-sensitive connected applications such as:

- Smart watches and fitness trackers
- Connected medical devices
- Smart home sensors and controllers
- Smart home appliances
- Gaming controllers
- Sports, smartphone, and virtual reality (Va qR) accessories
- Industrial sensor nodes
- Industrial logic controllers
- Advanced remote controllers

PSoC™ 6 Bluetooth® LE provides a cost-effective and small-footprint alternative to the combination of an MCU and a Bluetooth® LE radio. The ModusToolbox™ software environment supports PSoC™ 63 MCU application development with a set of tools for configuring the device, setting up peripherals, and complementing your projects with world-class middleware. To develop a Bluetooth® LE application, ModusToolbox™ provides a Bluetooth® configurator tool to easily set up Bluetooth® for your application which uses the AIROC™ Bluetooth® stack.

Bluetooth® LE is an ultra-low-power wireless standard defined by the Bluetooth® Special Interest Group (SIG) for short-range communication. PSoC™ 6 Bluetooth® LE integrates a Bluetooth® LE 4.2 radio and a royalty-free protocol stack with enhanced security, privacy, and throughput compliant with the Bluetooth® LE 5.2 specification.

Fourth-generation capacitive touch-sensing feature in PSoC™ 6 Bluetooth® LE devices, known as CAPSENSE™, offers unprecedented signal-to-noise ratio (SNR); best-in-class waterproofing; and a wide variety of sensor types such as buttons, sliders, trackpads, and proximity sensors. CAPSENSE™ user interfaces are gaining popularity in wearable electronic devices such as activity monitors and health and fitness equipment. The CAPSENSE™ solution works in noisy environments and in the presence of liquids.

PSoC™ 6 Bluetooth® LE enables ultra-low-power connected applications with an integrated solution.

Figure 1 illustrates the fitness tracker application block diagram for a real-world use case using PSoC™ 63 MCU.

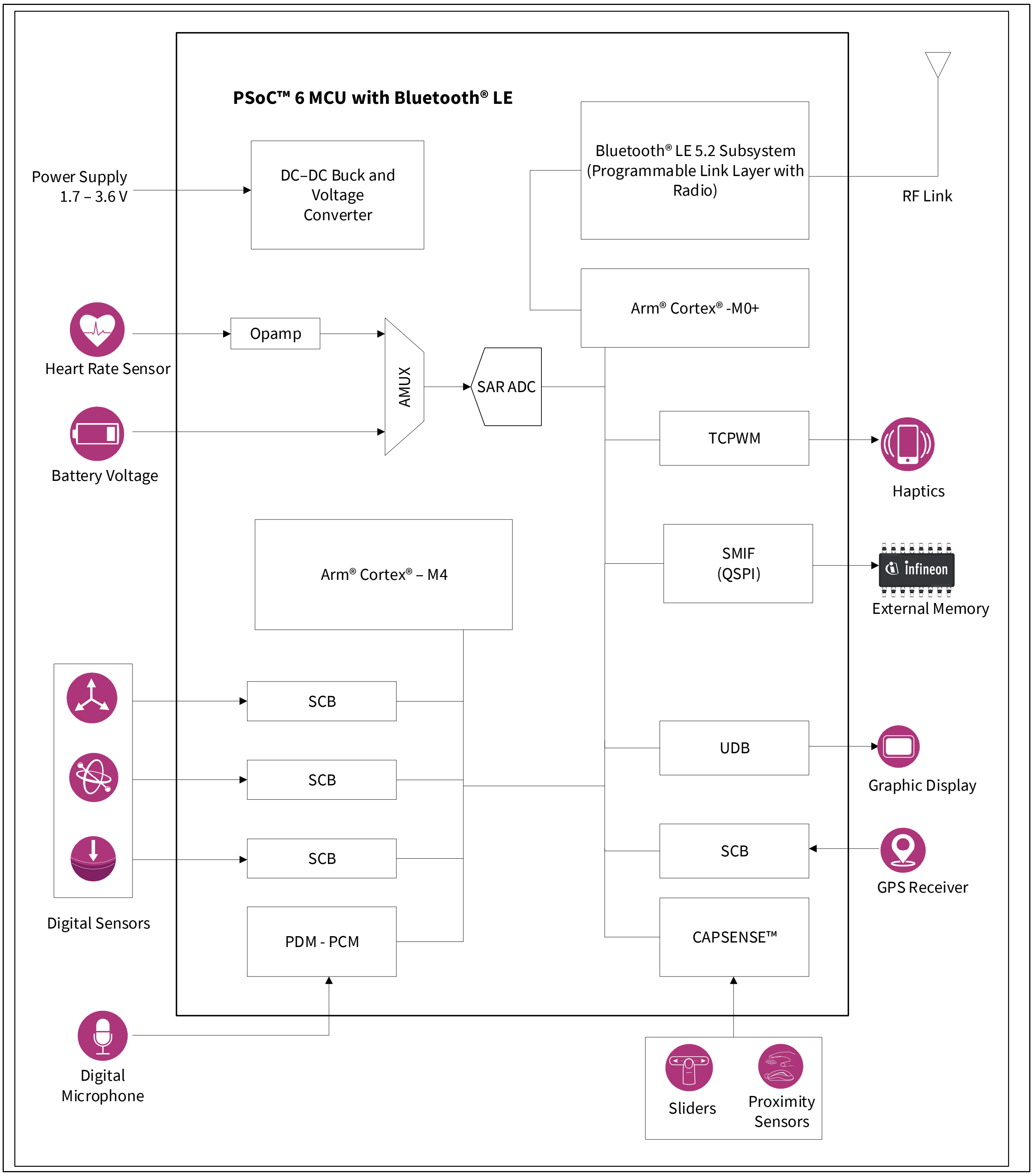


Figure 1 **Fitness tracker application block diagram**

Introduction

PSoC™ 63 MCU is a highly capable and flexible solution. For example, the real-world use case in Figure 1 takes advantage of these features:

- A low – power Bluetooth® LE 5.2 system that can sustain up to four simultaneous connections
- A buck converter for ultra-low-power operation
- An analog front end (AFE) within the device to condition and measure heart rate sensor outputs and to monitor battery voltage
- Serial Communication Blocks (SCBs) to interface with multiple digital sensors including the global positioning system (GPS) module
- A Pulse-Density Modulation (PDM) Pulse Code Modulation (PCM) hardware engine and digital microphone interface for voice
- CAPSENSE™ technology for reliable touch and proximity sensing
- A serial memory interface (SMIF) that supports an interface to Quad-Serial Peripheral Interface (QSPI) – enabled external memory
- Digital logic (UDB) and peripherals (TCPWM) to drive the display and haptics

This application note will help you get started with the PSoC™ 63 MCU device and covers the following:

- Overview of the development ecosystem support for the PSoC™ 63 MCU device, including software development platforms, hardware evaluation platforms, code examples, application notes, and other related technical documents.
- Tutorial on how to develop applications using the CY8CKIT-062-BLE device in Eclipse IDE for ModusToolbox™ by going through the step-by-step process of creating a simple Bluetooth® Low Energy-based application from scratch.

This application note does not cover the Bluetooth® Low Energy or classic Bluetooth® protocol. It assumes that the reader is already familiar with the basics of these protocols.

2 Development ecosystem

2.1 PSoC™ resources

A wealth of data is available at www.infineon.com to help you select the right PSoC™ device and quickly and effectively integrate it into your design. The following is an abbreviated list of resources for PSoC™ 63 MCU:

- Datasheet describes and provides electrical specifications for PSoC™ 63 MCU.
- Applications notes and Code examples cover a broad range of topics, from basic to advanced levels. You can browse our collection of code examples.
- Technical reference manuals (TRMs) provide detailed descriptions of the architecture and registers in each device family.
- PSoC™ 63 MCU programming specification provides the information necessary to program the nonvolatile memory of PSoC™ 63 MCU devices
- CAPSENSE™ design guides: Learn how to design capacitive touch-sensing applications with PSoC™ devices.

Development tools

- CY8CKIT-062-BLE PSoC™ 6-Bluetooth® LE pioneer kit is a development kit that supports the PSoC™ 63 series MCU that supports Bluetooth® LE connectivity.
- CY8CPROTO-063-BLE PSoC™ 6-Bluetooth® LE prototyping kit is a low–cost hardware platform that enables the design and debugging of PSoC™ 63 MCUs.
- CYBLE-416045-EVAL EZ-BLE Arduino evaluation board is a fully certified Bluetooth® LE module based on Infineon's PSoC™ 63 series MCU. It can be used as a standalone evaluation kit or can be combined with Arduino-compatible shields.

2.2 Firmware/application development

For application development with PSoC™ 63 MCU, use the ModusToolbox™ development platform. The software includes configuration tools, low-level drivers, middleware libraries, and other packages that enable you to create MCU and wireless applications. All tools run on Windows, macOS, and Linux. ModusToolbox™ includes an Eclipse IDE, which provides an integrated flow with all the ModusToolbox™ tools. Other IDEs such as Visual Studio Code, IAR Embedded Workbench, and Arm® MDK (µVision®) are also supported.

ModusToolbox™ software supports stand-alone devices and middleware configurators. Use the configurators to set the configuration of different blocks in the device and generate code that can be used in firmware development. The software supports all PSoC™ 63 MCUs. It is recommended that you use ModusToolbox™ software for all application development for PSoC™ 63 MCUs. See the ModusToolbox™ tools package user guide for more information.

For firmware development with PSoC™ 6 Bluetooth® LE, you will be mainly using board support package (BSP) and hardware abstraction layer (HAL) for PSoC™ 63 MCU features and AIROC™ BTSTACK library for the Bluetooth® feature.

Libraries and enablement software are available on the GitHub site.

ModusToolbox™ tools and resources can also be used on the command line. See the build system chapter in the ModusToolbox™ tools package user guide for detailed documentation.

2.2.1 ModusToolbox™ software

ModusToolbox™ software provides support for many types of devices and environments. From a practical standpoint, ModusToolbox™ software delivers in various ways, such as installation resources, code examples, BSPs, and libraries; you only use the resources you need. When you create applications, you use these resources and interact with the hardware through the HAL and/or the peripheral driver library (PDL).

Figure 2 shows the high-level view of the tools included in the ModusToolbox™ software.

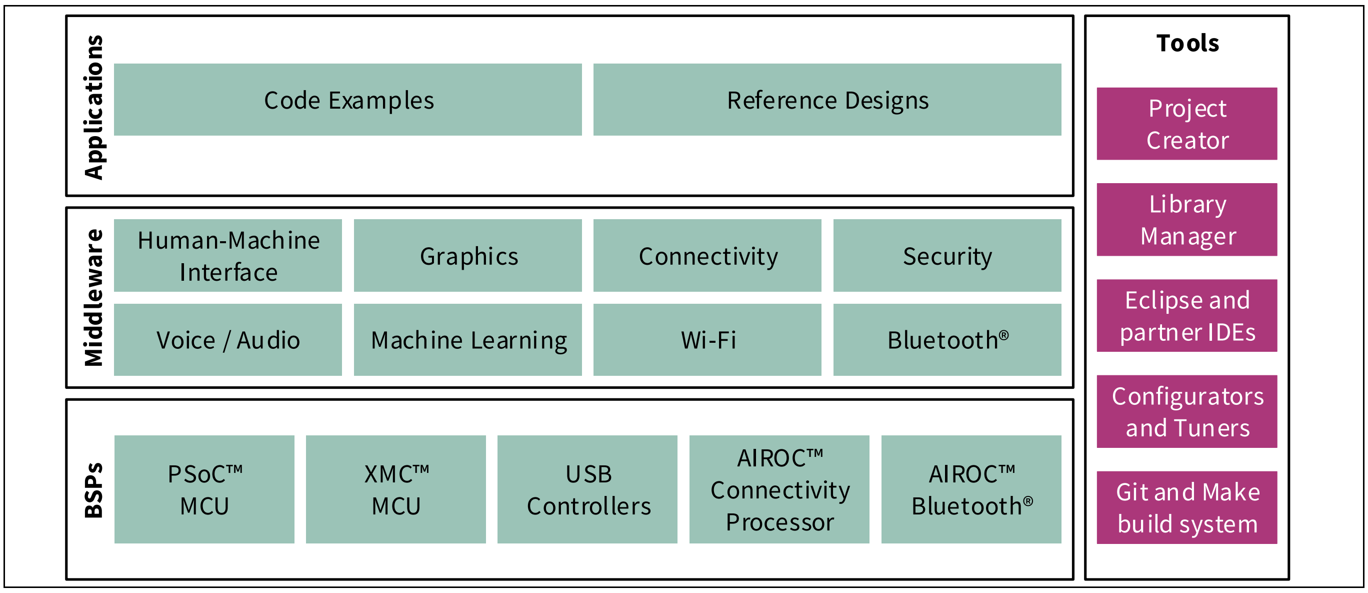


Figure 2 ModusToolbox™ software overview

The ModusToolbox™ tools package installer includes the design configurators and tools, and the build system infrastructure.

The build system infrastructure includes the new project creation wizard that can be run independently of the Eclipse IDE, the make infrastructure, and other tools. This means you choose your compiler, IDE, RTOS, and ecosystem without compromising usability or access to our industry-leading CAPSENSE™ (Human-Machine Interface), AIROC™ Wi-Fi and Bluetooth®, security, and various other features.

2.2.2 Getting started with ModusToolbox™

Visit the ModusToolbox™ home page to download and install the latest version of the ModusToolbox™. See the ModusToolbox™ installation guide document in the Documentation tab of the ModusToolbox™ home page for information on installing the ModusToolbox™ software. After installing, launch ModusToolbox™ and navigate to the following items:

- **User guide**: The detailed user guide at **Help** > **Eclipse IDE for ModusToolbox™ Documentation** > **user guide**.
- These documents are also available in the **Documentation** tab of the ModusToolbox™ home page.

2.2.3 Eclipse IDE for ModusToolbox™

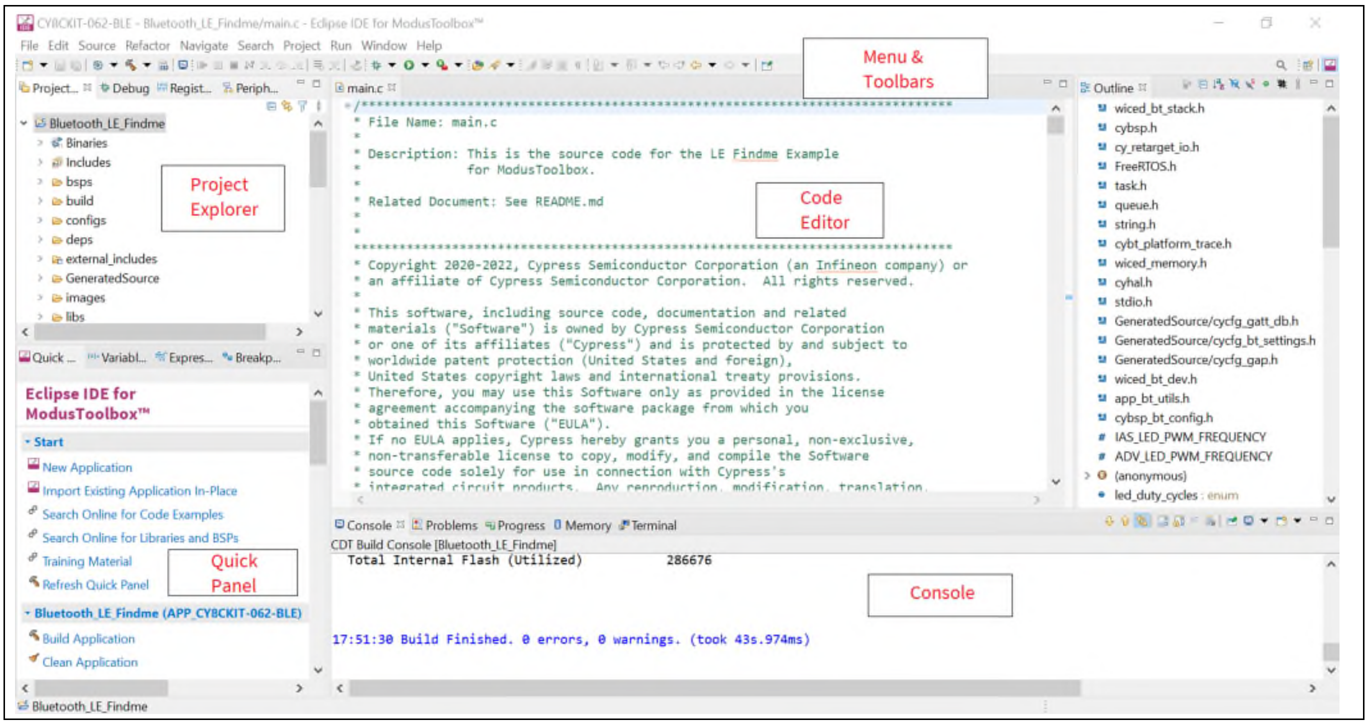
Eclipse IDE for ModusToolbox™ is based on the Eclipse IDE "Oxygen" version. It uses several plugins, including the Eclipse C/C++ Development Tools (CDT) plugin. The Eclipse Survival Guide provides tips and hints for using Eclipse IDE for ModusToolbox™.

The IDE contains Eclipse-standard menus and toolbars, plus various panels such as the Project Explorer, Code Editor, and Console, as shown in Figure 3. One difference from the standard Eclipse IDE is the "ModusToolbox™ Perspective." This perspective provides the "Quick Panel," a "News View," and adds tabs to the Project Explorer.

The top-level entity that you ultimately program to a device is called an application in the IDE. The application consists of one or more Eclipse projects. The IDE handles all dependencies between projects automatically. It also provides hooks for launching various tools provided by the software development kits (SDKs).

With Eclipse IDE for ModusToolbox™, you can:

1. Create a new application based on a list of starter applications filtered by kit or device or browse the collection of code examples online.
2. Configure device resources to build your hardware system design in the workspace.
3. Add software components or middleware.
4. Develop your application firmware.


CY8CKIT-062-BLE - Bluetooth_LE_Findme/main.c - Eclipse IDE for ModusToolbox™
File Edit Source Refactor Navigate Search Project Run Window Help
Menu & Toolbars
Project Explorer
Code Editor
Quick Panel
Console
Eclipse IDE for ModusToolbox™
Start
New Application
Import Existing Application In-Place
Search Online for Code Examples
Search Online for Libraries and BSPs
Training Material
Refresh Quick Panel
Bluetooth_LE_Findme (APP_CY8CKIT-062-BLE)
Build Application
Clean Application

```
/*******************************************************************************
 * File Name: main.c
 *
 * Description: This is the source code for the LE Findme Example
 *              for ModusToolbox.
 *
 * Related Document: See README.md
 *
 *
 *******************************************************************************
 * Copyright 2020-2022, Cypress Semiconductor Corporation (an Infineon company) or
 * an affiliate of Cypress Semiconductor Corporation.  All rights reserved.
 *
 * This software, including source code, documentation and related
 * materials ("Software") is owned by Cypress Semiconductor Corporation
 * or one of its affiliates ("Cypress") and is protected by and subject to
 * worldwide patent protection (United States and foreign),
 * United States copyright laws and international treaty provisions.
 * Therefore, you may use this Software only as provided in the license
 * agreement accompanying the software package from which you
 * obtained this Software ("EULA").
 * If no EULA applies, Cypress hereby grants you a personal, non-exclusive,
 * non-transferable license to copy, modify, and compile the Software
 * source code solely for use in connection with Cypress's
 * integrated circuit products.  Any reproduction, modification, translation,
```

```
CDT Build Console [Bluetooth_LE_Findme]
  Total Internal Flash (Utilized)          286676

17:51:30 Build Finished. 0 errors, 0 warnings. (took 43s.974ms)
```

Figure 3 Eclipse IDE for ModusToolbox™ overview

2.3 PSoC™ 6 Bluetooth® LE software resources

This section details the additional software resources required to start with firmware development for PSoC™ 6 Bluetooth® LE. See the PSoC™ 63 MCU software resources section of AN228571 to understand the software resources, such as configurators and different libraries required for PSoC™ 63 MCU.

AIROC™ BTSTACK

Infineon's AIROC™ BTSTACK is a software implementation of Bluetooth® core 5.2 Host protocol stack. The stack is hosted as a library on Infineon's GitHub. The stack library includes both Bluetooth® BR/EDR and Bluetooth® LE host and provides API for it. The application can choose to use Bluetooth® LE or both BR/EDR + LE. The stack is available for different Arm® cores such as CM4 and CM33 and can be used with three toolchains Arm®, GCC, and IAR. For PSoC™ 6 Bluetooth® LE will be using only the Bluetooth® LE build of the stack and the corresponding LE API.

The AIROC™ BTSTACK provides API for Bluetooth® LE host layers:

- L2CAP
- GATT
- GAP
- SMP

In addition to API related to Bluetooth® LE host stack layers, the AIROC™ BTSTACK also provides functions to configure Bluetooth® LE parameters such as PHY, device address, etc. It has utility functions for memory management.

The application uses the AIROC™ BTSTACK API extensively to implement the Bluetooth® functionality required.

Bluetooth® porting layer

AIROC™ BTSTACK requires a porting layer specific to the device it is running on. Porting layer sets up the physical transport required for HCI traffic, memory, threads, and other OS constructs required by the stack library.

Porting layer for Infineon Bluetooth® devices is hosted on GitHub as a library called btstack-integration with source. btstack-integration caters to various Bluetooth® devices with different hardware platforms such as PSoC™ 6 Bluetooth® LE, CYW20829, and PSoC™ 6 + CYW43xxx. It provides a component for each platform and the application can include the respective component to get the functionality.

In PSoC™ 6 Bluetooth® LE, the AIROC™ BTSTACK runs on the CM4 core, and the controller stack runs on the CM0+ core. A hardware block called inter-processor communication (IPC) is used as HCI. Therefore, the application uses the component called BLESS-IPC in btstack-integration.

2.4 FreeRTOS support with ModusToolbox™

Adding native FreeRTOS support to a ModusToolbox™ application project is like adding any middleware library. You can include the FreeRTOS middleware in your application by using the Library Manager. If using the Eclipse IDE, select the application project and click the **Library Manager** link in the **Quick Panel**. Click **Add Library** and select **freertos** from the **Core** dialog.

The *.mtb* file pointing to the FreeRTOS middleware is added to the application project's *deps* directory. The middleware content is also downloaded and placed inside the corresponding folder called **freertos**. The default location is in the shared asset repo named mtb_shared. To continue working with FreeRTOS, follow the steps in the Quick Start section of FreeRTOS documentation.

2.5 Configurators

ModusToolbox™ software provides graphical applications called configurators that make it easier to configure a hardware block. For example, instead of having to search through all the documentation to configure a Serial Communication Block as a UART with a desired configuration, open the appropriate configurator and set the baud rate, parity, and stop bits. After saving the hardware configuration, the tool generates the "C" code to initialize the hardware with the desired configuration.

There are two types of configurators: BSP configurators that configure items that are specific to the MCU hardware and library configurators that configure options for middleware libraries.

Configurators are independent of each other, but they can be used together to provide flexible configuration options. They can be used stand-alone, in conjunction with other tools, or within a complete IDE. Configurators are used for:

- Setting options and generating code to configure drivers
- Setting up connections such as pins and clocks for a peripheral
- Setting options and generating code to configure middleware

For PSoC™ 63 MCU applications, the available configurators include:

- **Device configurator:** Set up the system (platform) functions, pins, and the basic peripherals (e.g., UART, timer, PWM).
- **CAPSENSE™ configurator and tuner:** Configure CAPSENSE™ and generate the required code and tune CAPSENSE™ applications.
- **ML configurator:** To fit the pre-trained model of choice to the target device with a set of optimization parameters (only available as a part of a separate pack).
- **USB configurator:** Configure USB settings and generate the required code.
- **QSPI configurator:** Configure external memory and generate the required code.
- **Smart I/O configurator:** Configure Smart I/O pins.
- **Bluetooth® configurator:** Configure the Bluetooth® settings.
- **SegLCD configurator:** Configure and generate the required structures for the SegLCD driver.

Each of the above configurators creates its files (e.g., *design.cycapsense* for CAPSENSE™). BSP configurator files (e.g., *design.modus* or *design.cycapsense*) are provided as part of the BSP with default configurations while library configurators (e.g., *design.cybt*) are provided by the application. When an application is created based on Infineon BSP, the application makes use of BSP configurator files from the Infineon BSP repo. You can customize/create all the configurator files based on your application requirement using ModusToolbox™ software. See BSP Assistant to create your custom BSP. See ModusToolbox™ help for more details.

In the next section, see a detailed look at using the configurators as part of a Bluetooth® Low Energy application creation exercise.

2.6 PSoC™ 63 MCU development kits

Table 1 **PSoC™ 63 MCU development kits**

Product line	Development kits
Connectivity	PSoC™ 6 Bluetooth® LE pioneer kit (CY8CKIT-062-BLE) PSoC™ 6 Bluetooth® LE prototyping kit (CY8CPROTO-063-BLE) Arduino evaluation board (CYBLE-416045-EVAL)

3 Device features

PSoC™ 63 MCUs have extensive features as shown in Figure 4. The following is a list of major features. For more information, see the PSoC™ 63 MCU page.

- **MCU subsystem**
  - Powerful dual-core architecture with Arm® Cortex®-M4F core as the primary processor at up to 150 MHz, Arm® Cortex®-M0+ at up to 100 MHz
  - Up to 1 MB of flash with an additional 32 KB for EEPROM emulation and 32-KB supervisory flash
  - Up to 288 KB of SRAM with selectable Deep Sleep retention granularity at 32-KB retention boundaries
  - Inter-processor communication supported in hardware
  - DMA controllers
- **Security features**
  - Cryptography accelerators and true random number generator function
  - One-time programmable eFUSE for secure key storage
  - “Secure Boot” with hardware hash-based authentication
- **I/O subsystem**
  - Up to 78 GPIOs that can be used for analog, digital, CAPSENSE™, or segment LCD functions
  - Programmable drive modes, drive strength, and slew rates
  - Two ports with smart I/Os that can implement boolean operations

Device features

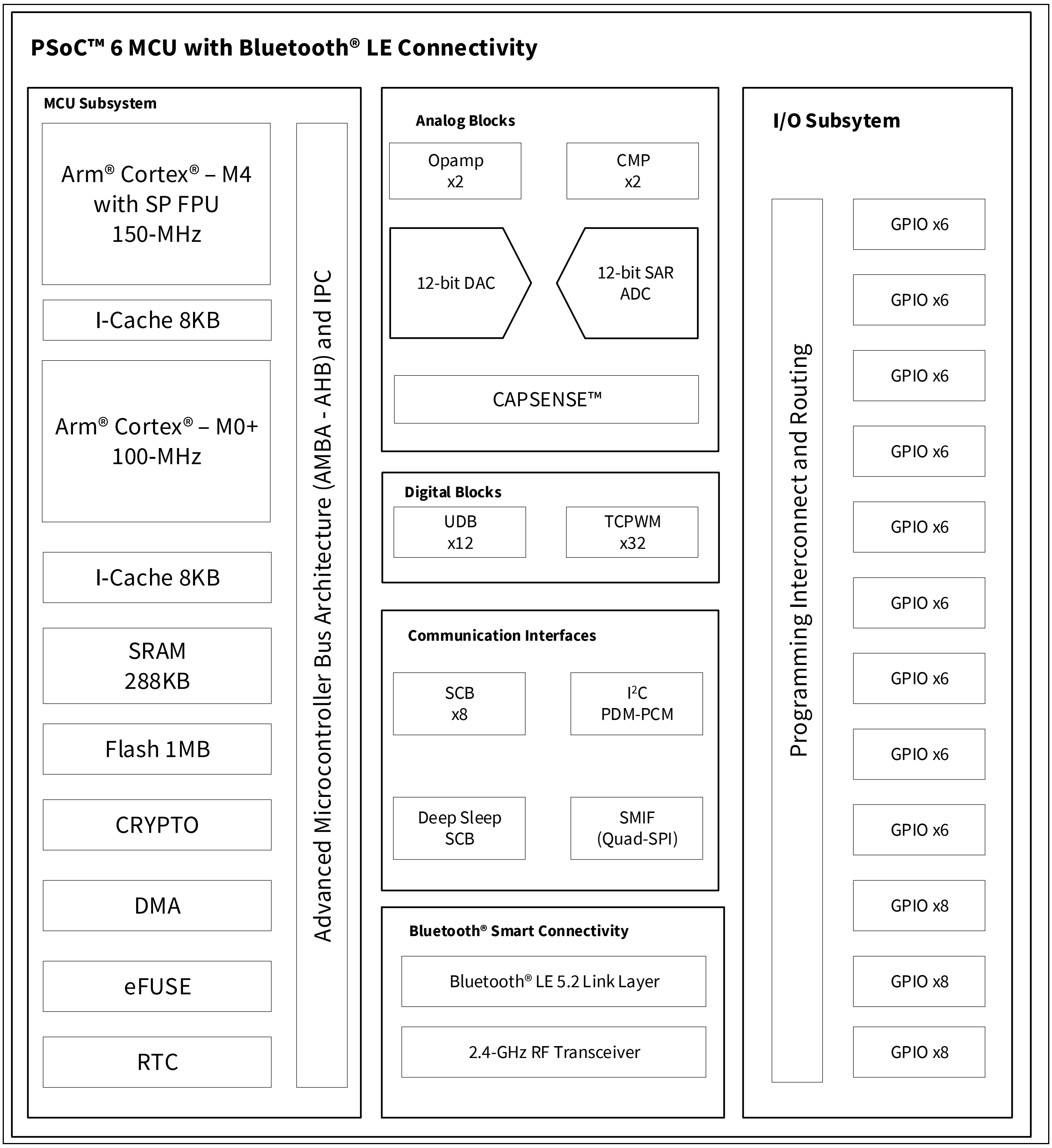


Figure 4 PSoC™ 63 MCU with Bluetooth® LE connectivity block diagram

Device features

- **Programmable analog blocks**
  - Two opamps of 6-MHz gain bandwidth (GBW), configurable as programmable gain amplifiers (PGA), comparators, or filters
  - Two low-power comparators with less than 300 nA current consumption that are operational in Deep Sleep and Hibernate modes
  - One 12-bit, 1-Msps SAR ADC with a 16-channel sequencer
  - One 12-bit voltage mode DAC
- **CAPSENSE™ with SmartSense auto-tuning**
  - Supports capacitive sigma-delta (CSD) and CAPSENSE™ transmit/receive (CSX)
  - Provides best-in-class SNR, liquid tolerance, and proximity sensing
- **Programmable digital blocks, communication interfaces**
  - 12 UDBs for custom digital peripherals
  - 32 TCPWM blocks as 16-bit/32-bit timer, counter, PWM, or quadrature decoder
  - Nine SCBs configurable as I^{2}C master or slave, SPI master or slave, or UART
  - Audio subsystem with one I^{2}S interface and two PDM channels
  - SMIF interface with support for execute-in-place from external qual SPI flash memory and on-the-fly encryption and decryption
- **Bluetooth® subsystem**
  - Complies with Bluetooth® core specification version 5.2
  - Includes support for Bluetooth® LE and LE 2 Mbps.
  - Programmable TX output power up to +4 dBm
  - Excellent receiver sensitivity (-95 dBm for Bluetooth® LE 1 Mbps)
  - Link layer engine supports four connections simultaneously
- **Operating voltage range, power domains, and low-power modes**
  - Device operating voltage: 1.71 V to 3.6 V
  - User-selectable core logic operation at either 1.1 V or 0.9 V
  - Multiple on-chip regulators: Low-drop out (LDO for Active, Deep Sleep modes), single input multiple output (SIMO) buck converter
  - Active, Low-power Active, Sleep, Low-Power Sleep, Deep Sleep, and Hibernate modes for fine power management
  - Deep Sleep mode with operational Bluetooth® LE link: 4.5-µA typical current at 3.3 V with 64-KB SRAM retention
  - An “always on” backup power domain with built-in RTC, power management integrated circuit (PMIC) control, and limited SRAM backup

For more information on the device, including the electrical specifications, see the device datasheet of PSoC™ 63 MCUs respectively.

4 My first PSoC™ 6 Bluetooth® Low Energy application

This section provides step-by-step instructions to build a simple Bluetooth® Low Energy-based application for the PSoC™ 6 Bluetooth® LE device using the Eclipse IDE for ModusToolbox™. A Bluetooth® SIG-defined standard profile called Find Me Profile (FMP) is implemented in the design. The steps covered in this section are:

- Part 1: Create a new application
- Part 2: Configure design resources
- Part 3: Write the application code
- Part 4: Build, program, and test your design

These instructions require that you use a particular code example (*Bluetooth® LE Find Me Profile* in this case). However, the extent to which you use the code example (CE) depends on the path you follow through these instructions. Note that the terms Code Example (CE) and application mean the same thing in the context of this document. A Code Example (CE) is simply an existing ModusToolbox™ application that serves a specific purpose or functionality.

Defined two paths through these instructions depending on what you need to learn:

Path	**"Using CE directly" path (Evaluate existing code example (CE) directly)**	**"Working from Scratch" path (Use existing code example (CE) as a reference only)**
Best For	A few are new to the tool or device and want to see how it all works quickly.	A few who want the hands-on experience to learn to develop PSoC™ 6 based Bluetooth® applications in ModusToolbox™.

What you need to do for each path is clearly defined at the start of each part of the instructions.

If you start from scratch and follow all instructions in this application note, you use the code example as a reference while following the instructions. Working from scratch helps you learn the design process and takes more time. Alternatively, you can evaluate the existing code example directly to get acquainted with the PSoC™ 6 Bluetooth® LE application development flow in a short time.

It would help if you started by reading Prerequisites and About the design in both cases.

4.1 Prerequisites

Ensure that you have the following items for this exercise.

- ModusToolbox™ 3.0 or later version installed on your PC
- CY8CKIT-062-BLE PSoC™ 6 Bluetooth® LE pioneer kit with Kitprog3. See the KitProg3 user guide on how to update the KitProg firmware
- This design is developed for the CY8CKIT-062-BLE PSoC™ 6 Bluetooth® LE pioneer kit. If you wish to use other hardware, you must adapt the instructions to your needs.
- AIROC™ Bluetooth® Connect iOS/Android app or any Android or iOS app that supports the Immediate Alert Service (IAS).

Scan the following QR codes from your mobile phone to download the AIROC™ Bluetooth® Connect app.

iOS	Android

4.2 About the design

This design implements a Bluetooth® Low Energy Find Me Profile (FMP) that consists of an Immediate Alert Service (IAS). FMP and IAS are Bluetooth® Low Energy standard Profile and Service respectively, as defined by the Bluetooth® SIG.

The design uses the two LEDs (red LED and orange LED) on the CY8CKIT-062-BLE kit. The orange LED (LED8) displays the IAS alert level – no alert (LED OFF), mild alert (LED blinking), or high alert (LED ON). The red LED (LED9) indicates whether the Peripheral device (PSoC™ 63 MCU) is advertising (LED blinking), connected (LED ON), or disconnected (LED OFF). In addition, a debug UART interface uses sending the Bluetooth® stack and application trace messages.

An iOS/Android mobile device or a PC can act as the Bluetooth® Low Energy Central device, connecting to the Peripheral device.

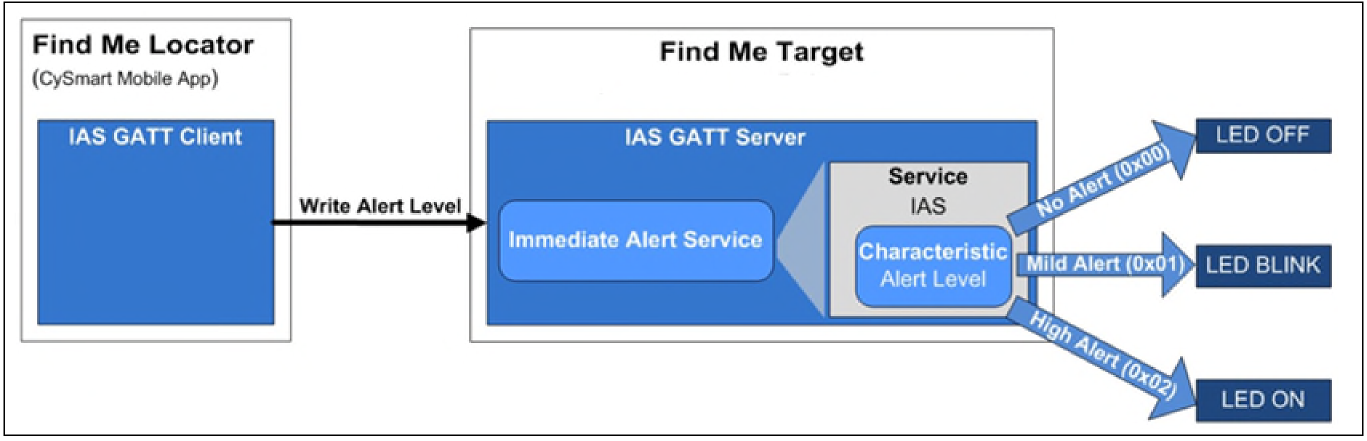


Figure 5 Find Me Profile (FMP) application using PSoC™ 63 MCU

4.3 Part 1: Create a new application

This part takes you step-by-step through creating a new ModusToolbox™ application. Before performing the steps in this section, decide whether you want to create and run the code example as-is or you would instead learn how to create an application from scratch. Depending on your choice, the steps you need to follow are as shown below:

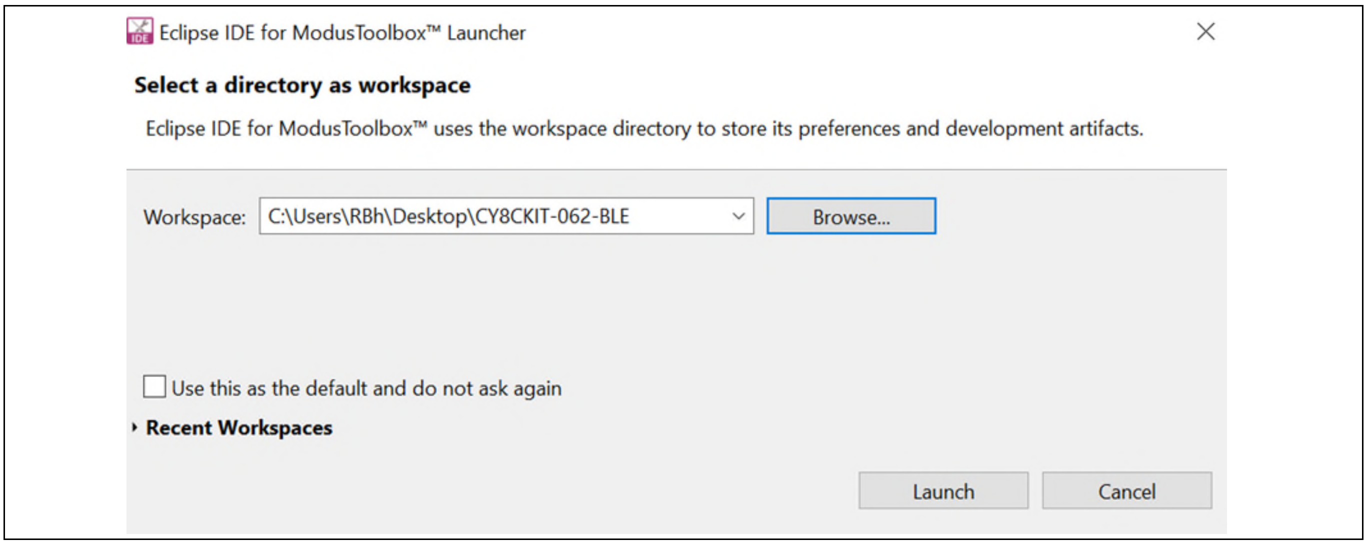
Path	**"Using CE directly" path (Evaluate existing code example (CE) directly)**	**"Working from Scratch" path (Use existing code example (CE) as reference only)**
Actions	Follow the sections 4.3.1, 4.3.2, 4.3.3, and 4.3.4. Ignore section 4.3.5.	Follow the sections 4.3.1, 4.3.2, 4.3.3, and 4.3.5. Ignore section 4.3.4.

Launch ModusToolbox™ and get started.

4.3.1 Select a new workspace

At launch, ModusToolbox™ presents a dialog to choose a directory for use as the workspace directory. The workspace directory is used to store workspace preferences and development artifacts such as device configuration and application source code.

You can choose an existing empty directory by clicking the **Browse** button, as Figure 6 shows. Alternatively, you can type in a directory name to be used as the workspace directory along with the complete path, and ModusToolbox™ will create the directory for you.


Eclipse IDE for ModusToolbox™ Launcher

Select a directory as workspace

Eclipse IDE for ModusToolbox™ uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\Users\RBh\Desktop\CY8CKIT-062-BLE

Figure 6 Select a directory as workspace

4.3.2 Create a new ModusToolbox™ application

Click **New Application** in the Start group of the Quick Panel. Alternatively, you can choose **File** > **New** > **ModusToolbox™ Application** (Figure 7).

The Eclipse IDE for ModusToolbox™ Application window appears.

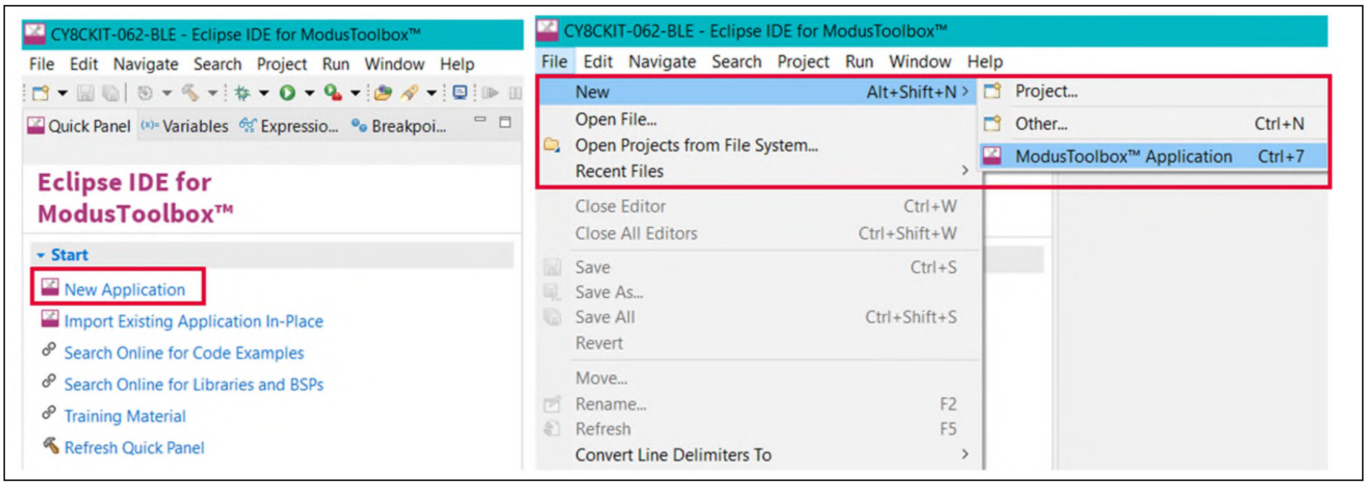
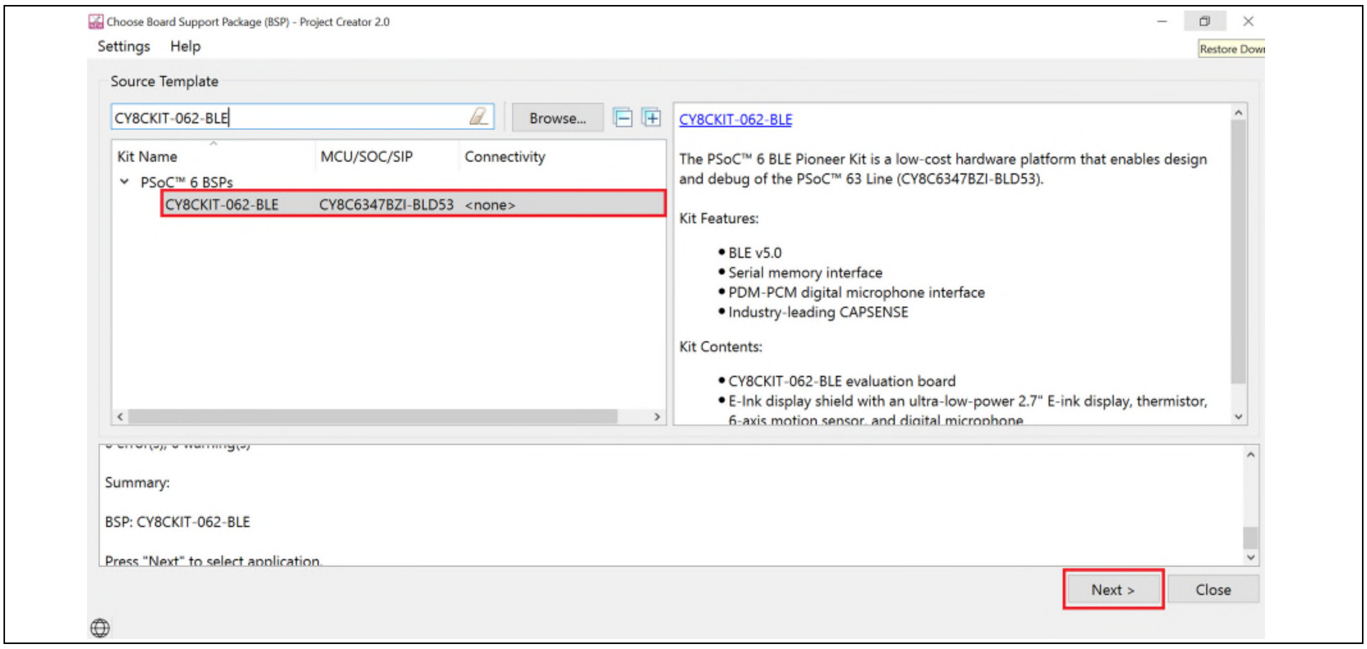

CY8CKIT-062-BLE - Eclipse IDE for ModusToolbox™
File Edit Navigate Search Project Run Window Help
Quick Panel
Variables
Expressio...
Breakpoi...
Eclipse IDE for ModusToolbox™
Start
New Application
Import Existing Application In-Place
Search Online for Code Examples
Search Online for Libraries and BSPs
Training Material
Refresh Quick Panel
New Alt+Shift+N
Open File...
Open Projects from File System...
Recent Files
Close Editor Ctrl+W
Close All Editors Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Project...
Other... Ctrl+N
ModusToolbox™ Application Ctrl+7

Figure 7 **Create a new ModusToolbox™ application**

4.3.3 Select PSoC™ 63 MCU-based target hardware

ModusToolbox™ presents the list of Infineon kits to start your application development. In this case, we want to develop an application on the CY8CKIT-062-BLE evaluation board that uses the PSoC™ 63-line device. Select **CY8CKIT-062-BLE** and click **Next** (Figure 8).



CY8CKIT-062-BLE

The PSoC™ 6 BLE Pioneer Kit is a low-cost hardware platform that enables design and debug of the PSoC™ 63 Line (CY8C6347BZI-BLD53).

Kit Features:

- BLE v5.0
- Serial memory interface
- PDM-PCM digital microphone interface
- Industry-leading CAPSENSE

Kit Contents:

- CY8CKIT-062-BLE evaluation board
- E-Ink display shield with an ultra-low-power 2.7" E-ink display, thermistor, 6-axis motion sensor, and digital microphone

Summary:

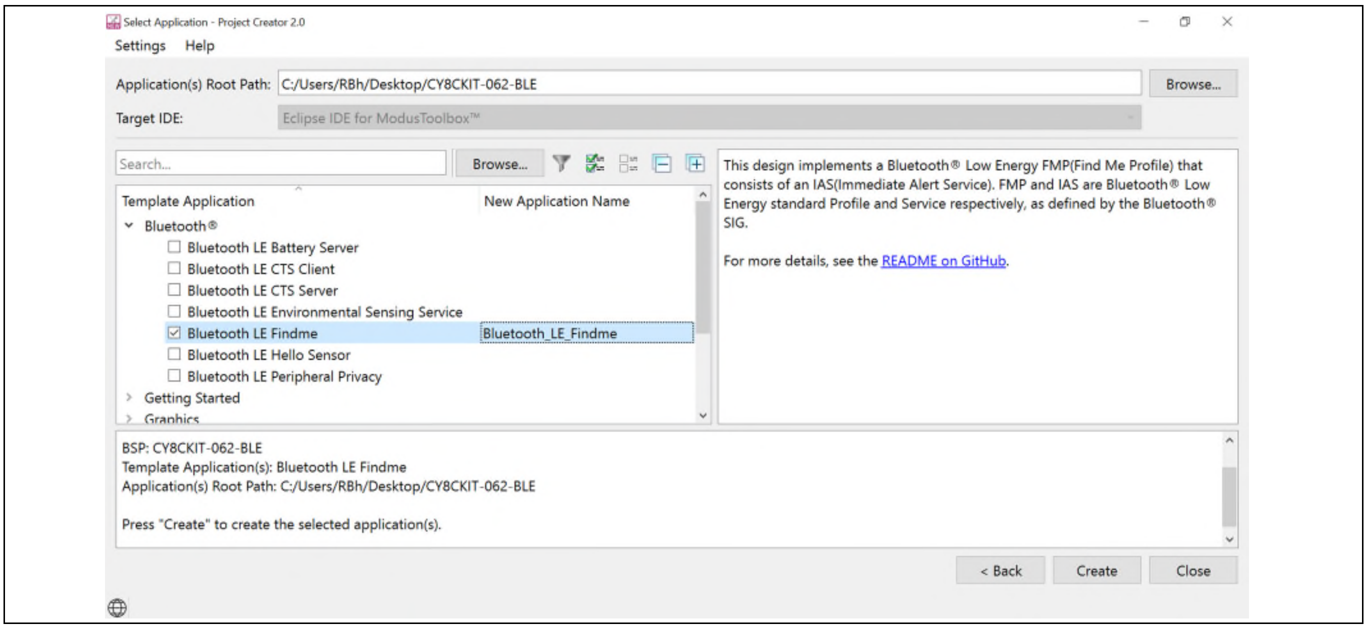
BSP: CY8CKIT-062-BLE

Press "Next" to select application.

Figure 8 **Choose target hardware**

4.3.4 Create the Bluetooth® LE Find Me code example (applicable only for the "Using CE directly" flow)

Here, you **Create** an existing code example into Eclipse IDE for ModusToolbox™. Use this feature to create the Bluetooth® LE Find Me code example for the *Using CE directly* flow. Figure 9 shows the **Select Application** dialog of the project creator tool. Select the Bluetooth® LE Find Me application, and optionally, in the 'New application Name' field, change the name of the application. Click on **Create** and wait for the application to get downloaded and created in the workspace. Click on **Close** to complete the application creation process.



Application(s) Root Path: C:/Users/RBh/Desktop/CY8CKIT-062-BLE

Target IDE: Eclipse IDE for ModusToolbox™

Template Application	New Application Name
Bluetooth®	
☐ Bluetooth LE Battery Server	
☐ Bluetooth LE CTS Client	
☐ Bluetooth LE CTS Server	
☐ Bluetooth LE Environmental Sensing Service	
☑ Bluetooth LE Findme	Bluetooth_LE_Findme
☐ Bluetooth LE Hello Sensor	
☐ Bluetooth LE Peripheral Privacy	
Getting Started	
Graphics	

This design implements a Bluetooth® Low Energy FMP(Find Me Profile) that consists of an IAS(Immediate Alert Service). FMP and IAS are Bluetooth® Low Energy standard Profile and Service respectively, as defined by the Bluetooth® SIG.

For more details, see the README on GitHub.

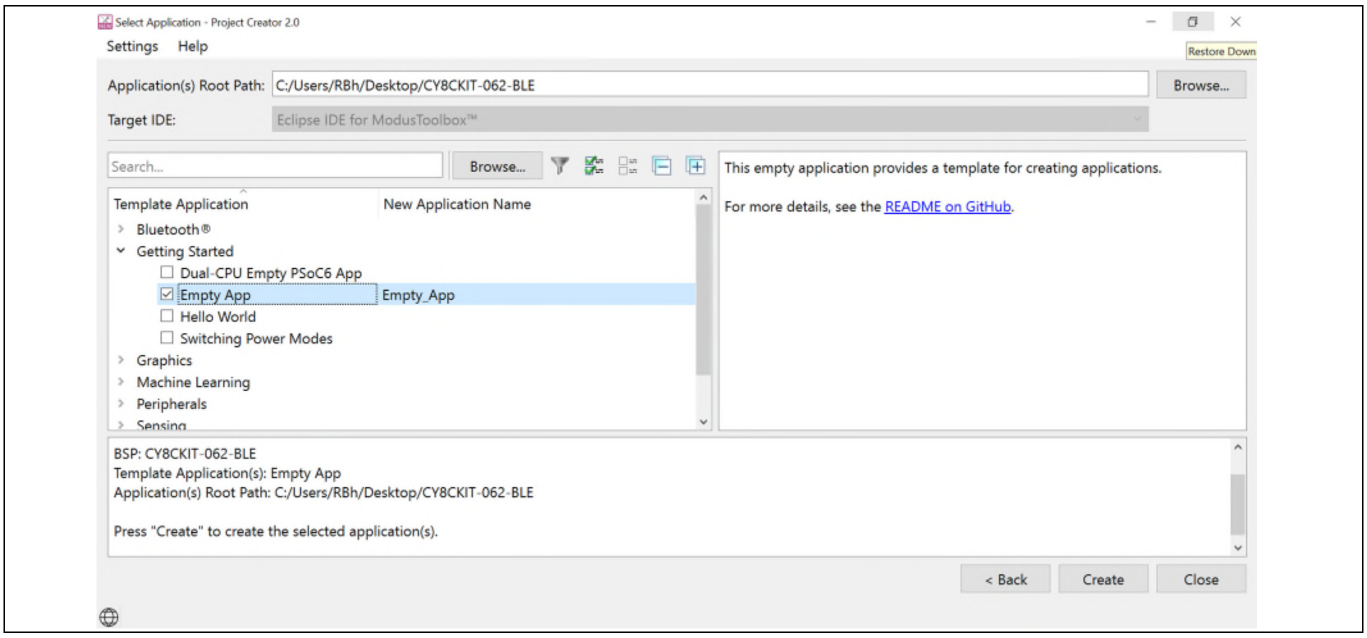
```
BSP: CY8CKIT-062-BLE
Template Application(s): Bluetooth LE Findme
Application(s) Root Path: C:/Users/RBh/Desktop/CY8CKIT-062-BLE

Press "Create" to create the selected application(s).
```

Figure 9 Create Bluetooth® LE Find Me code example

You have successfully created a new ModusToolbox™ application for CY8CKIT-062-BLE.

4.3.5 Select a starter application and create the application (Applicable only for "Working from Scratch" flow)

Here, you use an existing template application as the starting point for the *Working from Scratch* development flow. In the **Select Application** dialog shown in Figure 10, select **Empty_ App**. In the **Name** field, type in a name for the application and click **Next**; the application summary dialog appears. Click on **Create** and wait for the application to get downloaded and created in the workspace. Click on **Close** to complete the application creation process.



Application(s) Root Path: C:/Users/RBh/Desktop/CY8CKIT-062-BLE

Target IDE: Eclipse IDE for ModusToolbox™

Empty App: Empty_App

This empty application provides a template for creating applications.

For more details, see the README on GitHub.

BSP: CY8CKIT-062-BLE
Template Application(s): Empty App
Application(s) Root Path: C:/Users/RBh/Desktop/CY8CKIT-062-BLE

Press "Create" to create the selected application(s).

Figure 10 Starter application window

You have successfully created a new ModusToolbox™ application for CY8CKIT-062-BLE.

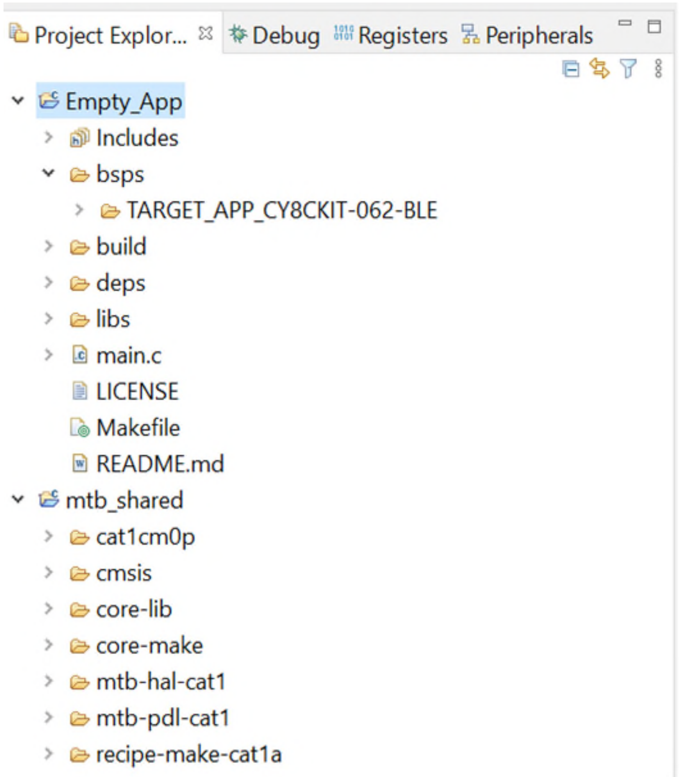
4.4 Part 2: Configure design resources

In this step, you will configure the design resources for your application and generate the configuration code.

Path	"Using CE directly" path (Evaluate existing Code Example (CE) directly)	"Working from Scratch" path (Use existing Code Example (CE) as reference only)
Actions	Read and understand all steps. The CE has the resource configurations done, therefore, you need not perform any of the steps in this section.	Perform all steps

The **Empty_App** application template has all the resources available on the CY8CKIT–062–BLE kit pre-configured and ready for use. These resources include user LEDs, push buttons, and communication peripherals (Bluetooth®, UART, I^{2}C, and SPI). The template application also contains a default application code snippet that initializes the device and the Bluetooth® stack and prints a status message on the Peripheral UART (PUART) interface.

Before proceeding further, a quick tour of the ModusToolbox™ Project Explorer is in order. Figure 11 shows the ModusToolbox™ Project Explorer view after the template application is created.



```
Empty_App
    Includes
    bsps
        TARGET_APP_CY8CKIT-062-BLE
    build
    deps
    libs
    main.c
    LICENSE
    Makefile
    README.md
mtb_shared
    cat1cm0p
    cmsis
    core-lib
    core-make
    mtb-hal-cat1
    mtb-pdl-cat1
    recipe-make-cat1a
```

Figure 11 Project Explorer view

- The files provided by the BSP are in the *bsps* folder and are listed under TARGET_<*bsp_name*> sub-folders. All the input files for the device and peripheral configurators are in the *config* folder inside the BSP.
- The *GeneratedSource* folder in the BSP contains the files that are generated by the configurators and are prefixed with *cycfg_*. These files contain the design configuration as defined by the BSP. From ModusToolbox™ 3.x or later, you can directly customize the configurator files of BSP for your application rather than overriding the default design configurator files with custom design configurator files since BSPs are completely owned by the application.
- The BSP folder also contains the linker scripts and the start-up code for the PSoC™ 63 MCU used on the board.
- The *build* folder contains all the artifacts resulting from a build of the project. The output files are organized by target BSPs.
- The *deps* folder contains .mtb files, which provide the locations from which ModusToolbox™ pulls the libraries that are directly referenced by the application. These files typically each contain the GitHub location of a library. The *.mtb* files also contain a git Commit Hash or Tag that tells which version of the library is to be fetched and a path as to where the library should be stored locally.

  For example, here, *retarget-io.mtb* points to `mtb://retarget-io#latest-v1.X#$$ASSET_REPO$$/retarget-io/latest-v1.X.` The variable `$$ASSET REPO$$` points to the root of the shared location which defaults to mtb_shared. If the library must be local to the application instead of shared, use `$$LOCAL$$`instead of `$$ASSET REPO$$.`
- The *libs* folder also contains .mtb files. In this case, they point to libraries that are included indirectly as a dependency of a BSP or another library. For each indirect dependency, the Library Manager places a .mtb file in this folder. These files have been populated based on the targets available in the *deps* folder.

For example, the BSP CY8CKIT-062-BLE populates the libs folder with the following .mtb files: cmsis.mtb, core-lib.mtb, core-make.mtb, mtb-hal-cat1.mtb, mtb-pdl-cat1.mtb, cat1cm0p.mtb, reciepe-make-cat1a.mtb.

The *libs* folder contains the file *mtb.mk,* which stores the relative paths of all the libraries required by the application. The build system uses this file to find all the libraries required by the application. Everything in the *libs* folder is generated by the Library Manager, therefore, you should not manually edit anything in that folder.

- An application contains a Makefile which is in the application's root folder. This file contains the set of directives that the make tool uses to compile and link the application project. There can be more than one project in an application. In that case, there is a Makefile at the application level and one inside each project.
- By default, when creating a new application or adding a library to an existing application and specifying it as shared, all libraries are placed in a *mtb_shared* directory adjacent to the application directories.

  The *mtb_shared* folder is shared between different applications within a workspace. Different applications may use different versions of shared libraries if necessary.

Now, let's get into how to configure the design resources in the template application.

4.4.1 Configure hardware resources

Bluetooth® LE Find Me code example uses two LEDs. We use HAL functions to configure and initialize the GPIOs routed to the LEDs on the board. The BSP provides aliases for GPIO pins routed to different components on the board. These aliases can be found in the file *bsps* > *TARGET_<BSP-name>> config* > *GeneratedSource* > *cycfg_pins.h.* For the two LEDs, we have the following aliases provided in the BSP:

Table 2 Pin mapping details

Pin	Alias	Purpose
P1.5	CYBSP_USER_LED	Mapped to the Orange LED (LED8) on the kit. Indicates IAS Alert Level.
P13.7	CYBSP_USER_LED2	Mapped to the Red LED (LED9) on the kit. Indicates the Advertising/Connected state of the Bluetooth® Low Energy peripheral device.

The mentioned aliases are generated through **design.modus** file which can be found in the *bsps/TARGET_<BSP-name>/config.* You can view the alias setting in the Device Configurator tool. The Device Configurator is used to enable/configure the peripherals and the pins used in the application. To launch the Device Configurator, double-click the *design.modus* file or click on **Device Configurator** in the Quick Panel, as shown in Figure 12. This file is used by the graphical configurators, which generate the configuration firmware. This firmware is stored in the application's *GeneratedSource* folder. Any time you change the Device Configurator, click **File > Save** to save the updated configuration.

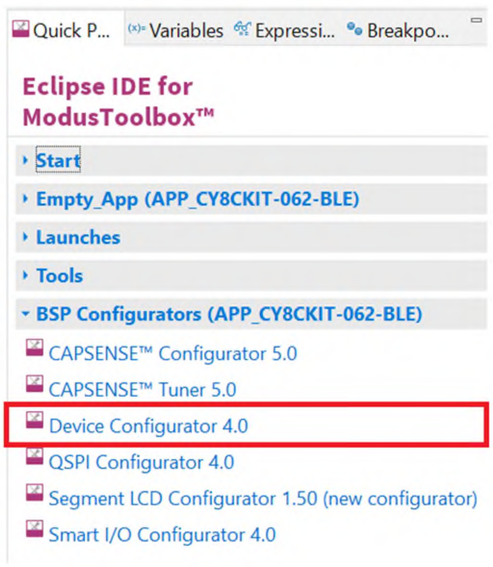
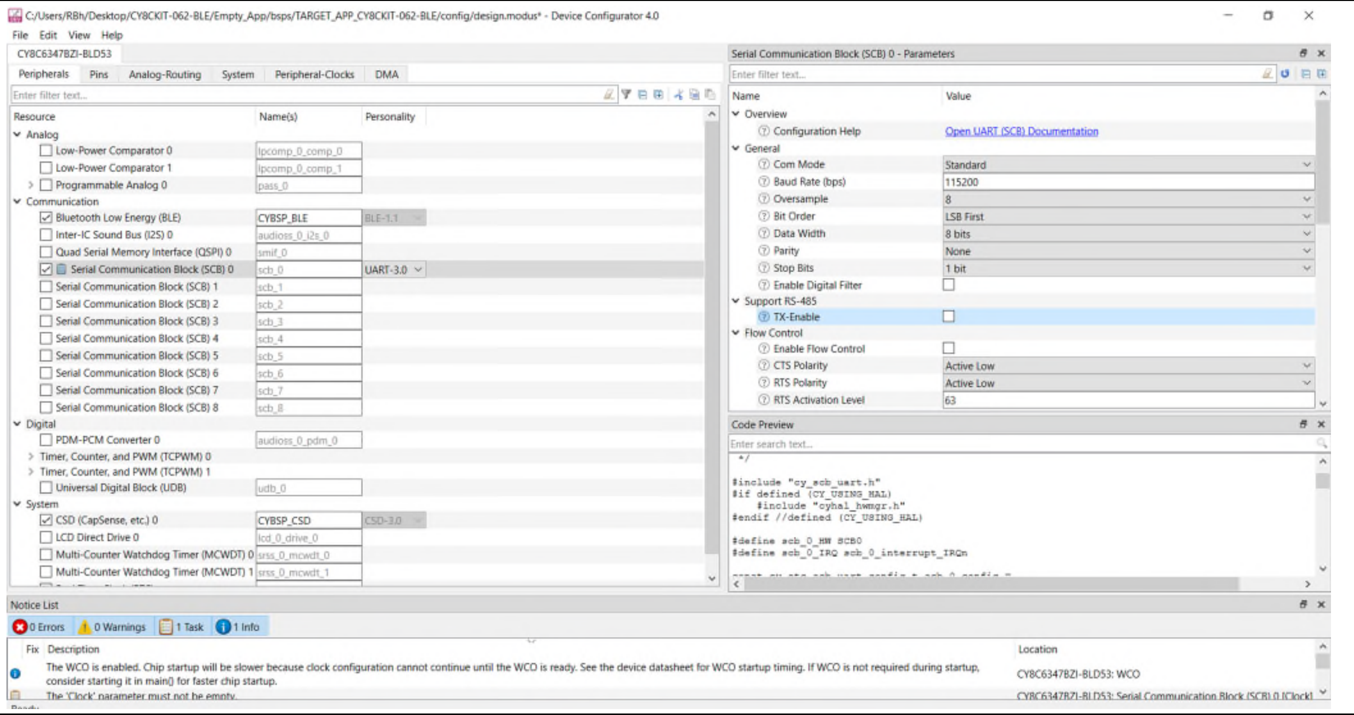

Quick P...
Variables
Expressi...
Breakpo...
Eclipse IDE for ModusToolbox™
Start
Empty_App (APP_CY8CKIT-062-BLE)
Launches
Tools
BSP Configurators (APP_CY8CKIT-062-BLE)
CAPSENSE™ Configurator 5.0
CAPSENSE™ Tuner 5.0
Device Configurator 4.0
QSPI Configurator 4.0
Segment LCD Configurator 1.50 (new configurator)
Smart I/O Configurator 4.0

Figure 12 **Quick Panel view**

Figure 13 shows the Device Configurator view showing the Peripherals view for this application.


C:/Users/RBh/Desktop/CY8CKIT-062-BLE/Empty_App/bsps/TARGET_APP_CY8CKIT-062-BLE/config/design.modus* - Device Configurator 4.0
File
Edit
View
Help
CY8C6347BZI-BLD53
Peripherals
Pins
Analog-Routing
System
Peripheral-Clocks
DMA
Enter filter text...

Resource	Name(s)	Personality
Analog		
☐ Low-Power Comparator 0	lpcomp_0_comp_0	
☐ Low-Power Comparator 1	lpcomp_0_comp_1	
☐ Programmable Analog 0	pass_0	
Communication		
✓ Bluetooth Low Energy (BLE)	CYBSP_BLE	BLE-1.1
☐ Inter-IC Sound Bus (I2S) 0	audioss_0_i2s_0	
☐ Quad Serial Memory Interface (QSPI) 0	smif_0	
✓ Serial Communication Block (SCB) 0	scb_0	UART-3.0
☐ Serial Communication Block (SCB) 1	scb_1	
☐ Serial Communication Block (SCB) 2	scb_2	
☐ Serial Communication Block (SCB) 3	scb_3	
☐ Serial Communication Block (SCB) 4	scb_4	
☐ Serial Communication Block (SCB) 5	scb_5	
☐ Serial Communication Block (SCB) 6	scb_6	
☐ Serial Communication Block (SCB) 7	scb_7	
☐ Serial Communication Block (SCB) 8	scb_8	
Digital		
☐ PDM-PCM Converter 0	audioss_0_pdm_0	
Timer, Counter, and PWM (TCPWM) 0		
Timer, Counter, and PWM (TCPWM) 1		
☐ Universal Digital Block (UDB)	udb_0	
System		
✓ CSD (CapSense, etc.) 0	CYBSP_CSD	CSD-3.0
☐ LCD Direct Drive 0	lcd_0_drive_0	
☐ Multi-Counter Watchdog Timer (MCWDT) 0	srss_0_mcwdt_0	
☐ Multi-Counter Watchdog Timer (MCWDT) 1	srss_0_mcwdt_1	

Serial Communication Block (SCB) 0 - Parameters

Name	Value
Overview	
Configuration Help	Open UART (SCB) Documentation
General	
Com Mode	Standard
Baud Rate (bps)	115200
Oversample	8
Bit Order	LSB First
Data Width	8 bits
Parity	None
Stop Bits	1 bit
Enable Digital Filter	☐
Support RS-485	
TX-Enable	☐
Flow Control	
Enable Flow Control	☐
CTS Polarity	Active Low
RTS Polarity	Active Low
RTS Activation Level	63

Code Preview

```
 */

#include "cy_scb_uart.h"
#if defined (CY_USING_HAL)
    #include "cyhal_hwmgr.h"
#endif //defined (CY_USING_HAL)

#define scb_0_HW SCB0
#define scb_0_IRQ scb_0_interrupt_IRQn
```

Notice List

0 Errors | 0 Warnings | 1 Task | 1 Info

Fix	Description	Location
	The WCO is enabled. Chip startup will be slower because clock configuration cannot continue until the WCO is ready. See the device datasheet for WCO startup timing. If WCO is not required during startup, consider starting it in main() for faster chip startup.	CY8C6347BZI-BLD53: WCO
	The 'Clock' parameter must not be empty.	CY8C6347BZI-BLD53: Serial Communication Block (SCB) 0 [Clock]

Figure 13 **Device Configurator**

The **Device Configurator** provides a set of **Resources Categories** tabs. Here you can choose between different resources available in the device such as peripherals, pins, and clocks from the **List of Resources**.

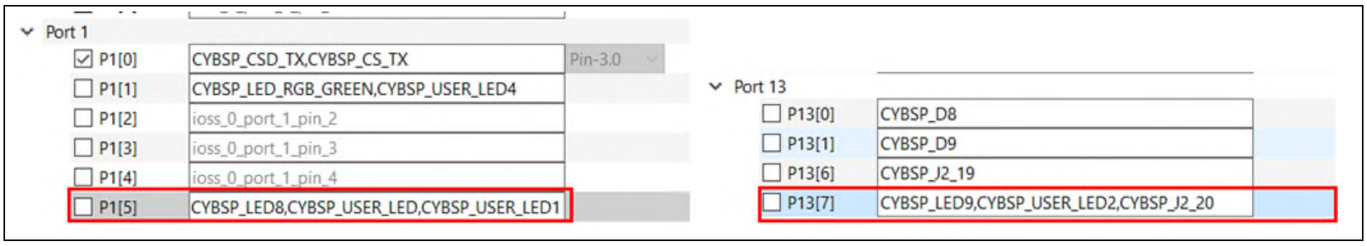
You can choose how a resource behaves by choosing a **Personality** for the resource. For example, a **Serial Communication Block (SCB)** resource can have **EZI2C, I2C, SPI, or UART** personalities. The **Alias** is your name for the resource, which is used in firmware development. One or more aliases can be specified by using a comma to separate them (with no spaces).

The **Parameters** panel is where you enter the configuration parameters for each enabled resource and the selected personality. The **Code Preview** panel shows the configuration code generated per the configuration parameters selected. This code is populated in the cycfg_ files in the GeneratedSource folder. The Parameters panel and Code Preview panel may be displayed as tabs instead of separate windows, but the contents will be the same.

Any errors, warnings, and information messages arising out of the configuration are displayed in the Notices panel.

Currently, the device configurator supports configurations using a PDL source. If you choose to use HAL libraries in your application then you do not need to do any device configuration changes here.

Click on **Pins** and expand the sections: **Port 1** and **Port 13**. Under Port 1, you can see the alias CYBSP_USER_LED against Pin 1.5, and under Port 13, you can see the alias CYBSP_USER_LED2 against Pin 13.7 as shown in Figure 14.



Port 1

☑ P1[0]	CYBSP_CSD_TX,CYBSP_CS_TX	Pin-3.0
☐ P1[1]	CYBSP_LED_RGB_GREEN,CYBSP_USER_LED4	
☐ P1[2]	ioss_0_port_1_pin_2	
☐ P1[3]	ioss_0_port_1_pin_3	
☐ P1[4]	ioss_0_port_1_pin_4	
☐ P1[5]	CYBSP_LED8,CYBSP_USER_LED,CYBSP_USER_LED1	

Port 13

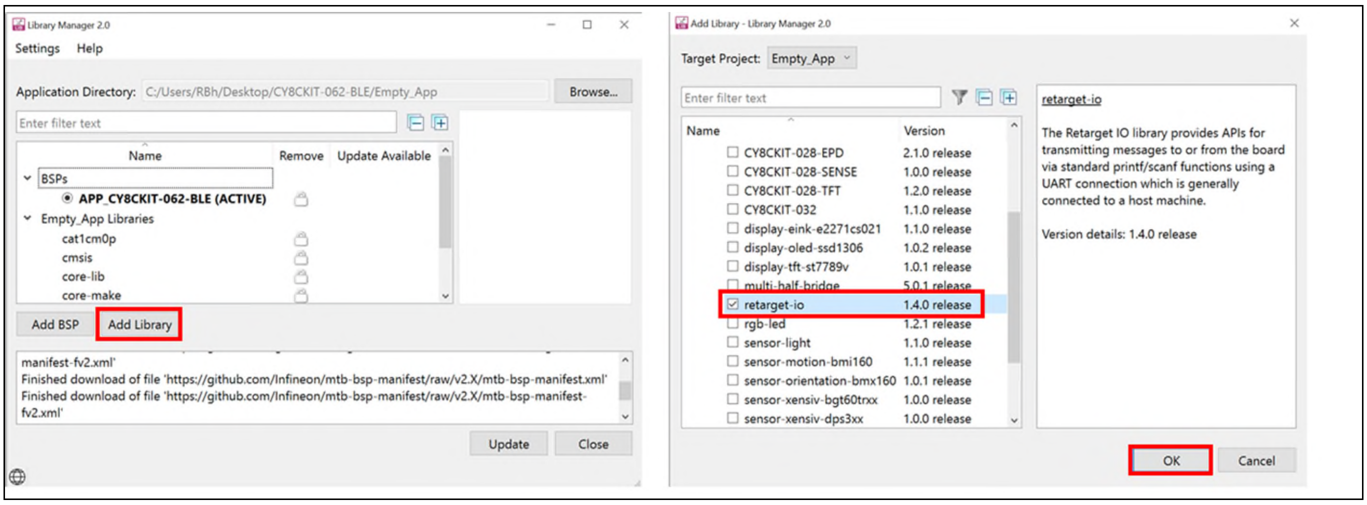
☐ P13[0]	CYBSP_D8
☐ P13[1]	CYBSP_D9
☐ P13[6]	CYBSP_J2_19
☐ P13[7]	CYBSP_LED9,CYBSP_USER_LED2,CYBSP_J2_20

Figure 14 Device configurator – pin aliases

4.4.2 Add libraries and middleware

ModusToolbox™ provides a 'Library Manager' dialog to select various middleware components for developing Bluetooth® applications. To launch the Library Manager dialog, in the **Quick Panel**, click the **Library Manager**. Left-click on **Add Library** to add the required libraries and middleware for your application. For Bluetooth® LE Find Me, follow the below steps to add the required libraries.

- In this step, you will add the retarget-io middleware to redirect standard input and output streams to the UART configured by the BSP. The initialization of the middleware will be done in *main.c*. After clicking on **Add Library,** select **Peripheral > retarget-io** (see Figure 15 for this option).



Application Directory: C:/Users/RBh/Desktop/CY8CKIT-062-BLE/Empty_App

BSPs
- APP_CY8CKIT-062-BLE (ACTIVE)

Empty_App Libraries
- cat1cm0p
- cmsis
- core-lib
- core-make

```
manifest-fv2.xml'
Finished download of file 'https://github.com/Infineon/mtb-bsp-manifest/raw/v2.X/mtb-bsp-manifest.xml'
Finished download of file 'https://github.com/Infineon/mtb-bsp-manifest/raw/v2.X/mtb-bsp-manifest-
fv2.xml'
```

Target Project: Empty_App

Name	Version
☐ CY8CKIT-028-EPD	2.1.0 release
☐ CY8CKIT-028-SENSE	1.0.0 release
☐ CY8CKIT-028-TFT	1.2.0 release
☐ CY8CKIT-032	1.1.0 release
☐ display-eink-e2271cs021	1.1.0 release
☐ display-oled-ssd1306	1.0.2 release
☐ display-tft-st7789v	1.0.1 release
☐ multi-half-bridge	5.0.1 release
☑ retarget-io	1.4.0 release
☐ rgb-led	1.2.1 release
☐ sensor-light	1.1.0 release
☐ sensor-motion-bmi160	1.1.1 release
☐ sensor-orientation-bmx160	1.0.1 release
☐ sensor-xensiv-bgt60trxx	1.0.0 release
☐ sensor-xensiv-dps3xx	1.0.0 release

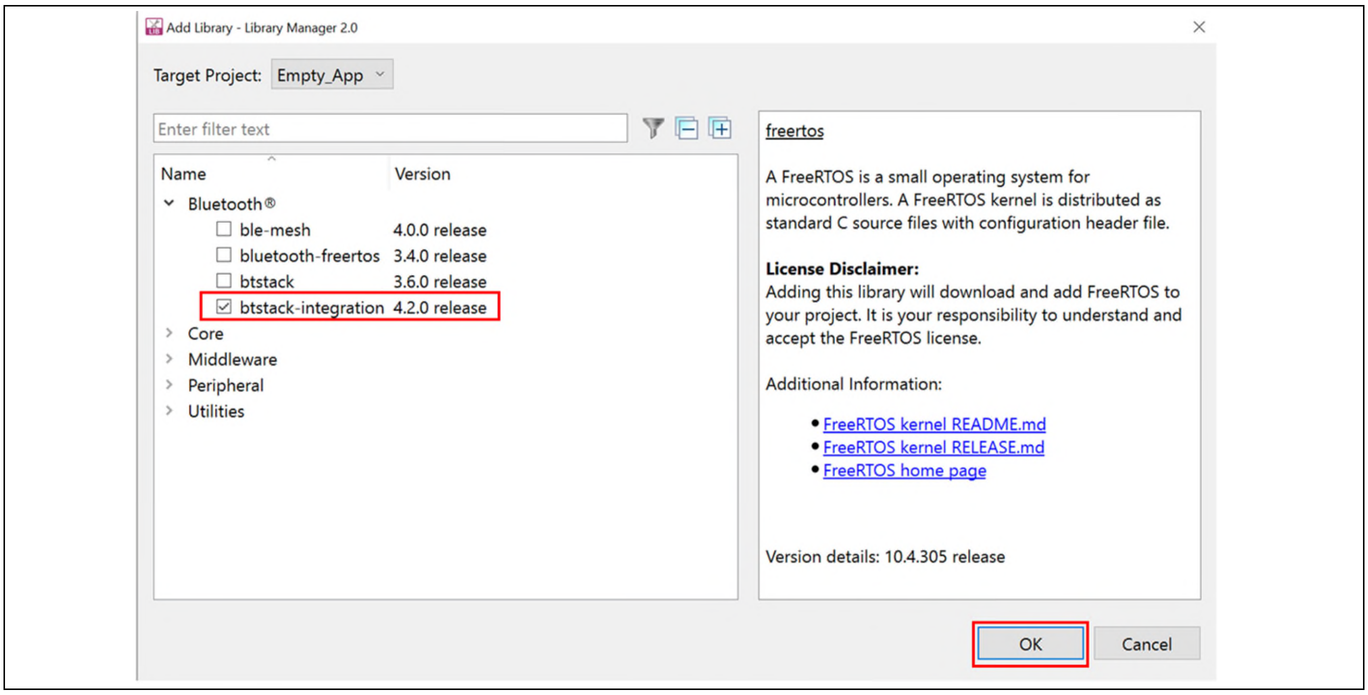
retarget-io

The Retarget IO library provides APIs for transmitting messages to or from the board via standard printf/scanf functions using a UART connection which is generally connected to a host machine.

Version details: 1.4.0 release

Figure 15 Add the retarget-io middleware

- Next, you add the AIROC™ BTSTACK which is a software implementation of Bluetooth® core 5.2 Host protocol stack. AIROC™ BTSTACK requires a porting layer specific to the device it is running on. Therefore, you will add the btstack integration porting layer which sets up the physical transport required for HCI traffic, memory, threads, and other OS constructs required by the stack library.
  Click on **Add Library** and select **Bluetooth® > btstack-integration** (see Figure 16 for this option).
  Note that selecting btstack-integration will in turn select the required version of **btstack**. We do not have to explicitly select btstack unless a specific version is required. Along with btstack, btstack-integration also adds dependency libraries **abstraction-rtos** and **freertos**.



Target Project: Empty_App

Name	Version
Bluetooth®	
☐ ble-mesh	4.0.0 release
☐ bluetooth-freertos	3.4.0 release
☐ btstack	3.6.0 release
☑ btstack-integration	4.2.0 release

freertos

A FreeRTOS is a small operating system for microcontrollers. A FreeRTOS kernel is distributed as standard C source files with configuration header file.

License Disclaimer:
Adding this library will download and add FreeRTOS to your project. It is your responsibility to understand and accept the FreeRTOS license.

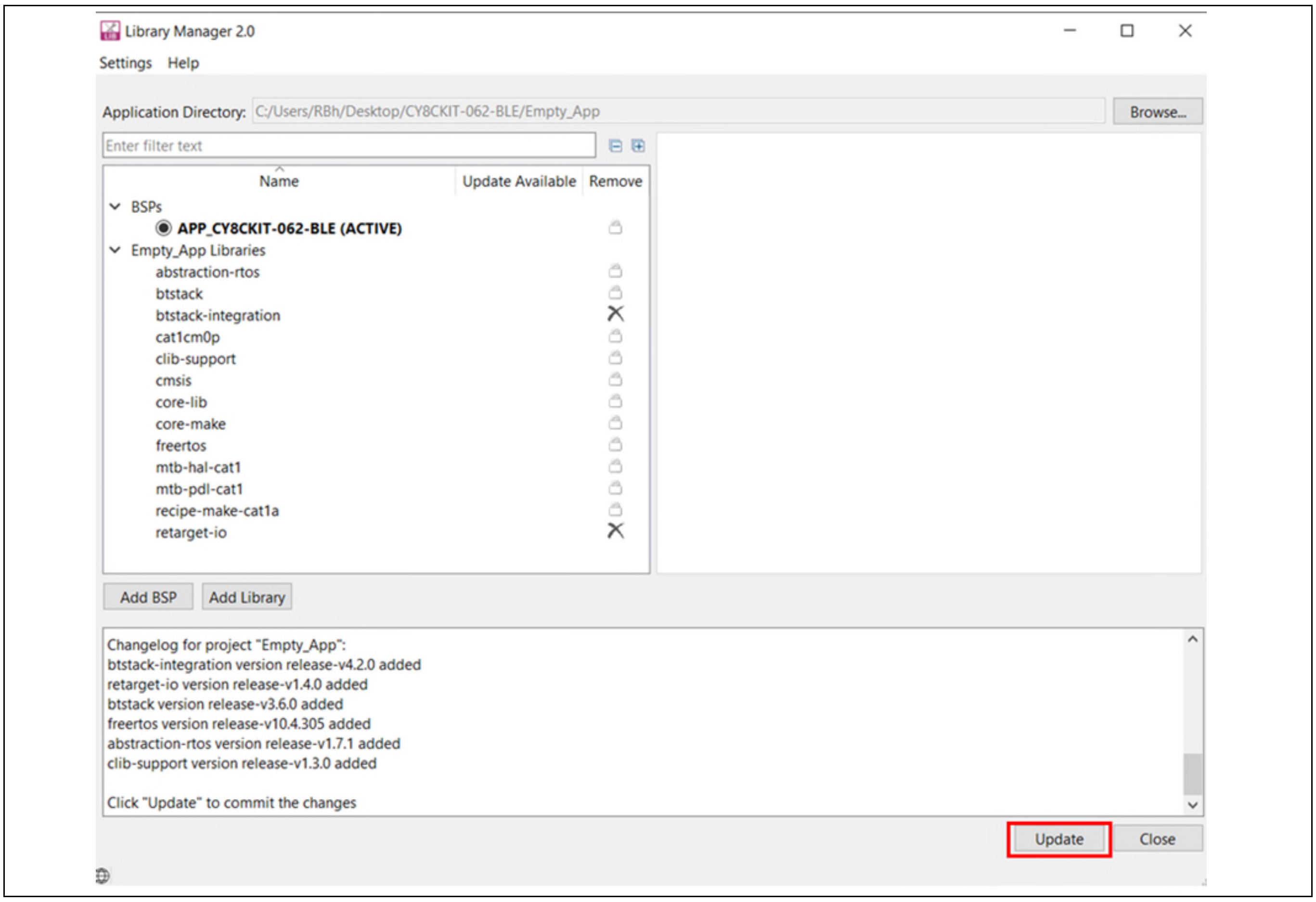
Additional Information:

- FreeRTOS kernel README.md
- FreeRTOS kernel RELEASE.md
- FreeRTOS home page

Version details: 10.4.305 release

Figure 16 Add btstack library and btstack-integration middleware

- You have selected all the required libraries. To add them to the project, click on **OK** and then **Update**. Figure 17 shows all the libraries selected and their dependency libraries. The files necessary to use the retarget-io middleware are added in the *mtb_shared* > *retarget_io* folder, and the *.mtb* file is added to the deps folder. Similarly, you can find other libraries under the respective folder in the mtb_shared folder.



```
Changelog for project "Empty_App":
btstack-integration version release-v4.2.0 added
retarget-io version release-v1.4.0 added
btstack version release-v3.6.0 added
freertos version release-v10.4.305 added
abstraction-rtos version release-v1.7.1 added
clib-support version release-v1.3.0 added

Click "Update" to commit the changes
```

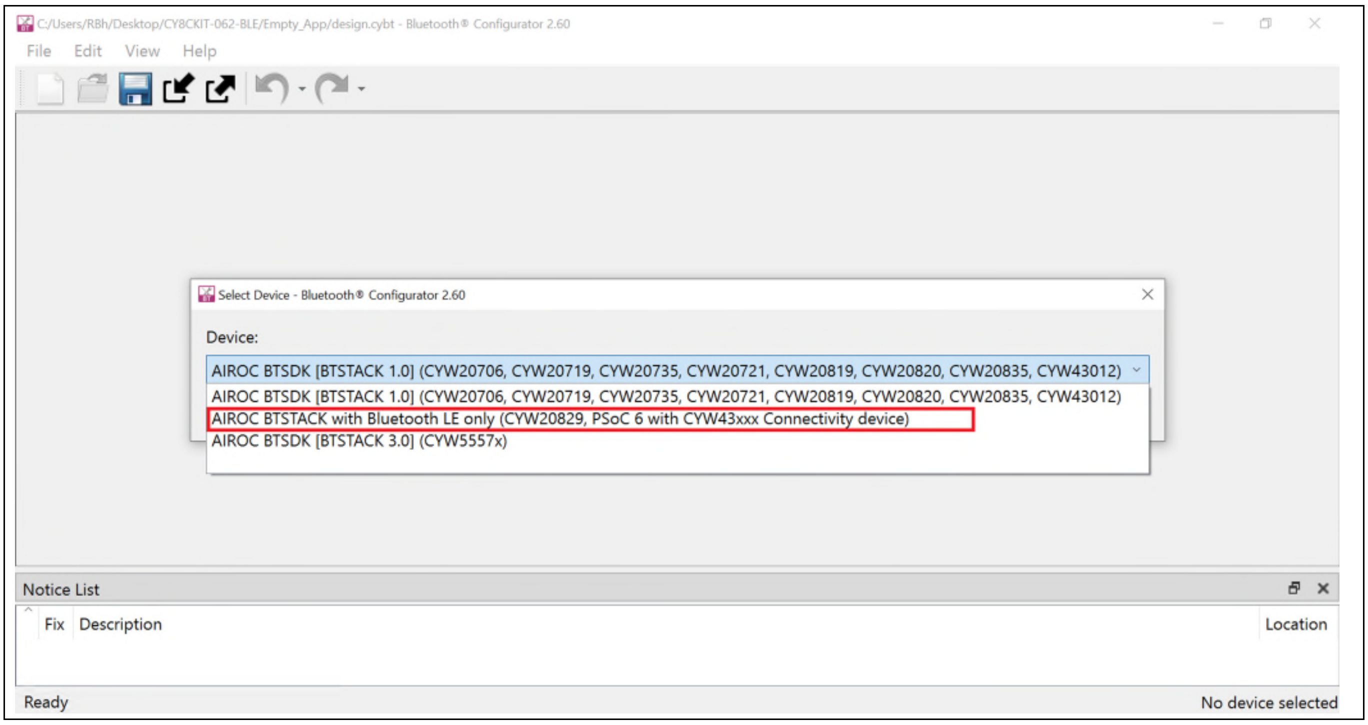
Figure 17 Libraries for Bluetooth® LE Find Me code example

4.4.3 Bluetooth® Configurator

The Bluetooth® peripheral has an additional configurator called the Bluetooth® Configurator that is used to create the Bluetooth® Low Energy configuration structure and GATT database for the application. The Bluetooth® LE configuration structure generated will be used by the application during stack initialization.

For the Find Me Profile application, you need to generate a GATT database and Bluetooth® settings to initialize the host btstack corresponding to the Find Me Target role of the CY8CKIT-062-BLE device. To launch the Bluetooth® Configurator, In **Quick Panel** click on the **Bluetooth® Configurator** under the **Library Configurator** section, and then click on the **Bluetooth® Configurator** button.

Note: *Bluetooth® configurator tool generates Bluetooth® configuration structure for a specific Bluetooth® stack. Up to bt-configurator version 2.6, there are options to generate the configuration for AIROC™ BTSTACK, AIROC™ BTSDK, and PSoC™ 6 BLESS middleware. Because PSoC™ 6 Bluetooth® LE device, no longer supports BLESS middleware from MTB version 3.0 and uses AIROC™ BTSTACK, use the option 'AIROC™ BTSTACK with Bluetooth® LE only' while creating a* *new configuration file. The Bluetooth® configurator will be updated in coming versions of MTB to support PSoC™ 6 Bluetooth® LE using AIROC™ BTSTACK.*


C:/Users/RBh/Desktop/CY8CKIT-062-BLE/Empty_App/design.cybt - Bluetooth® Configurator 2.60
File
Edit
View
Help
Select Device - Bluetooth® Configurator 2.60
Notice List
Fix
Description
Location
Ready
No device selected

Device:

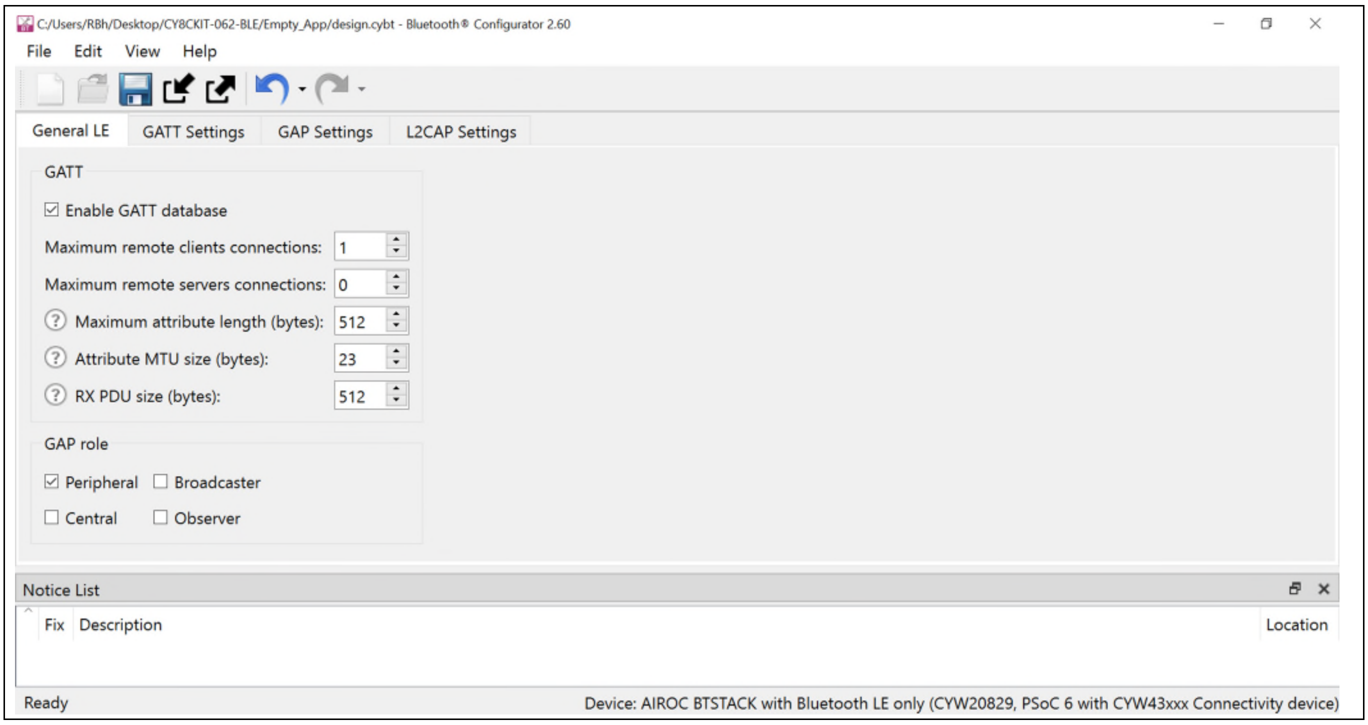
AIROC BTSDK [BTSTACK 1.0] (CYW20706, CYW20719, CYW20735, CYW20721, CYW20819, CYW20820, CYW20835, CYW43012)

AIROC BTSDK [BTSTACK 1.0] (CYW20706, CYW20719, CYW20735, CYW20721, CYW20819, CYW20820, CYW20835, CYW43012)
AIROC BTSTACK with Bluetooth LE only (CYW20829, PSoC 6 with CYW43xxx Connectivity device)
AIROC BTSDK [BTSTACK 3.0] (CYW5557x)

Figure 18 Selecting Bluetooth® Stack for PSoC™ devices

Set the **General LE** properties as shown in Figure 19.

- Enable the GATT database
- **Maximum remote client connections** set to 1. This will configure the Bluetooth® LE stack appropriately
- Confirm that **Peripheral** is selected as the **GAP role**. This sets the device to act as a Bluetooth® LE Peripheral device and respond to central device requests.


C:/Users/RBh/Desktop/CY8CKIT-062-BLE/Empty_App/design.cybt - Bluetooth® Configurator 2.60

General LE | GATT Settings | GAP Settings | L2CAP Settings

GATT
- ☑ Enable GATT database
- Maximum remote clients connections: 1
- Maximum remote servers connections: 0
- Maximum attribute length (bytes): 512
- Attribute MTU size (bytes): 23
- RX PDU size (bytes): 512

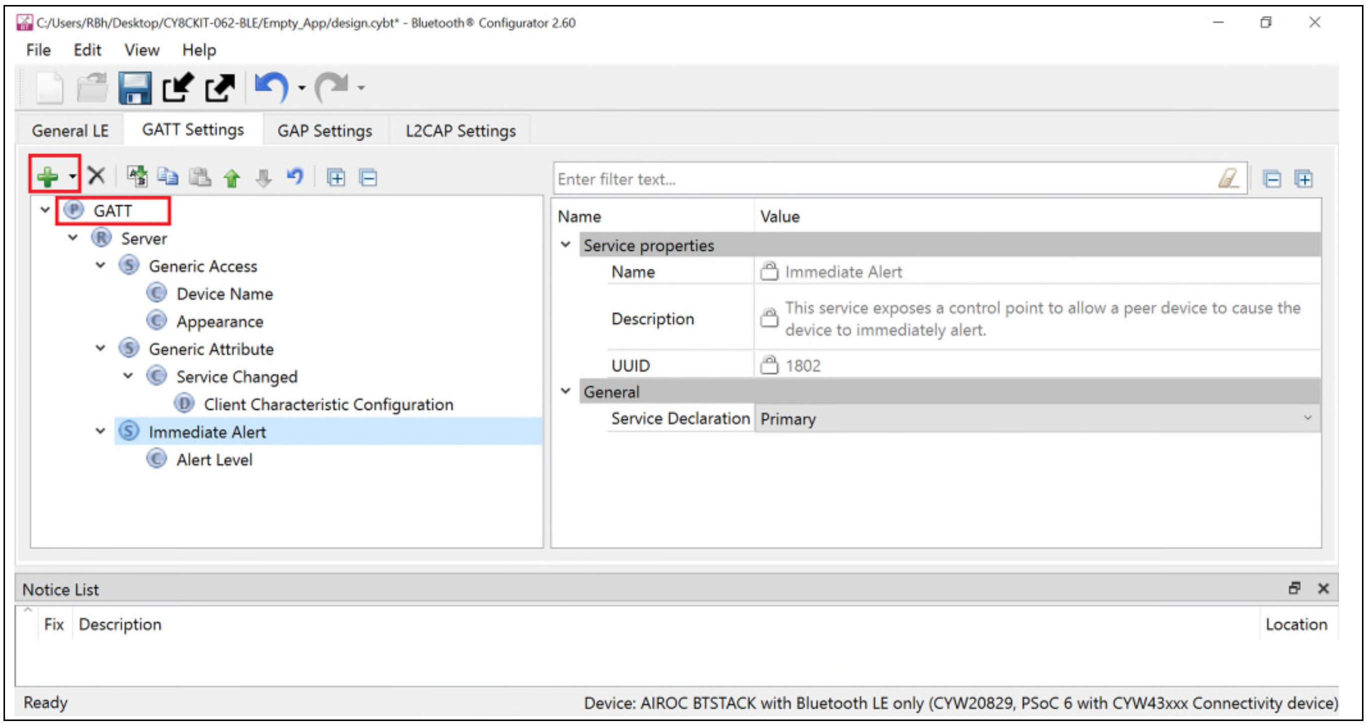
GAP role
- ☑ Peripheral ☐ Broadcaster
- ☐ Central ☐ Observer

Notice List: Fix | Description | Location

Ready — Device: AIROC BTSTACK with Bluetooth LE only (CYW20829, PSoC 6 with CYW43xxx Connectivity device)

Figure 19 General LE configuration

To add the Find Me Target profile, select GATT Settings Tab, click **GATT** profile (in Figure 20), and then click the **+** icon. Select the **Find Me Target (GATT Server)** profile from the drop-down menu, as shown in Figure 21.


C:/Users/RBh/Desktop/CY8CKIT-062-BLE/Empty_App/design.cybt* - Bluetooth® Configurator 2.60

General LE | GATT Settings | GAP Settings | L2CAP Settings

- GATT
  - Server
    - Generic Access
      - Device Name
      - Appearance
    - Generic Attribute
      - Service Changed
        - Client Characteristic Configuration
    - Immediate Alert
      - Alert Level

Name	Value
Service properties	
Name	Immediate Alert
Description	This service exposes a control point to allow a peer device to cause the device to immediately alert.
UUID	1802
General	
Service Declaration	Primary

Ready — Device: AIROC BTSTACK with Bluetooth LE only (CYW20829, PSoC 6 with CYW43xxx Connectivity device)

Figure 20 Bluetooth® configurator

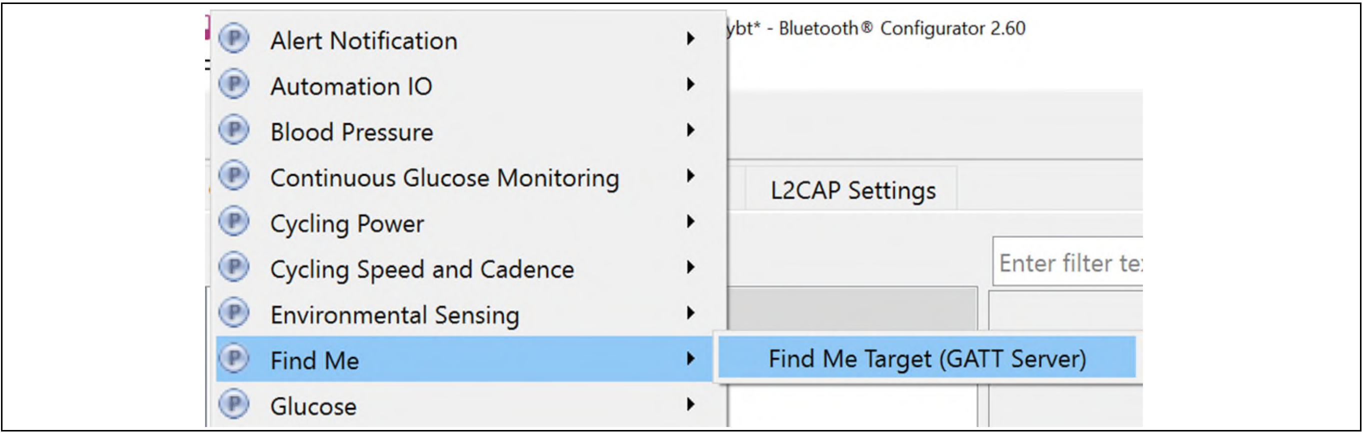
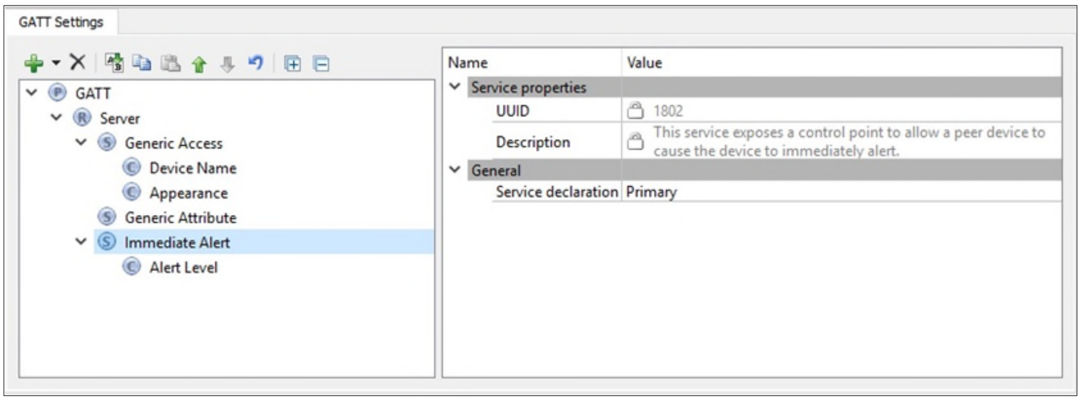


Figure 21 **Adding Find Me Target profile**

Figure 22 shows the GATT database view once the Find Me Target Server has been added. Note that the Immediate Alert Service corresponding to the Find Me Target profile has been added. Click **File** > **Save** in the Configurator window or click the Save icon. The configurator stores the GATT database in the source files *cycfg_gatt_db.c* and *cycfg_gatt_db.h* in the *GeneratedSource* folder.



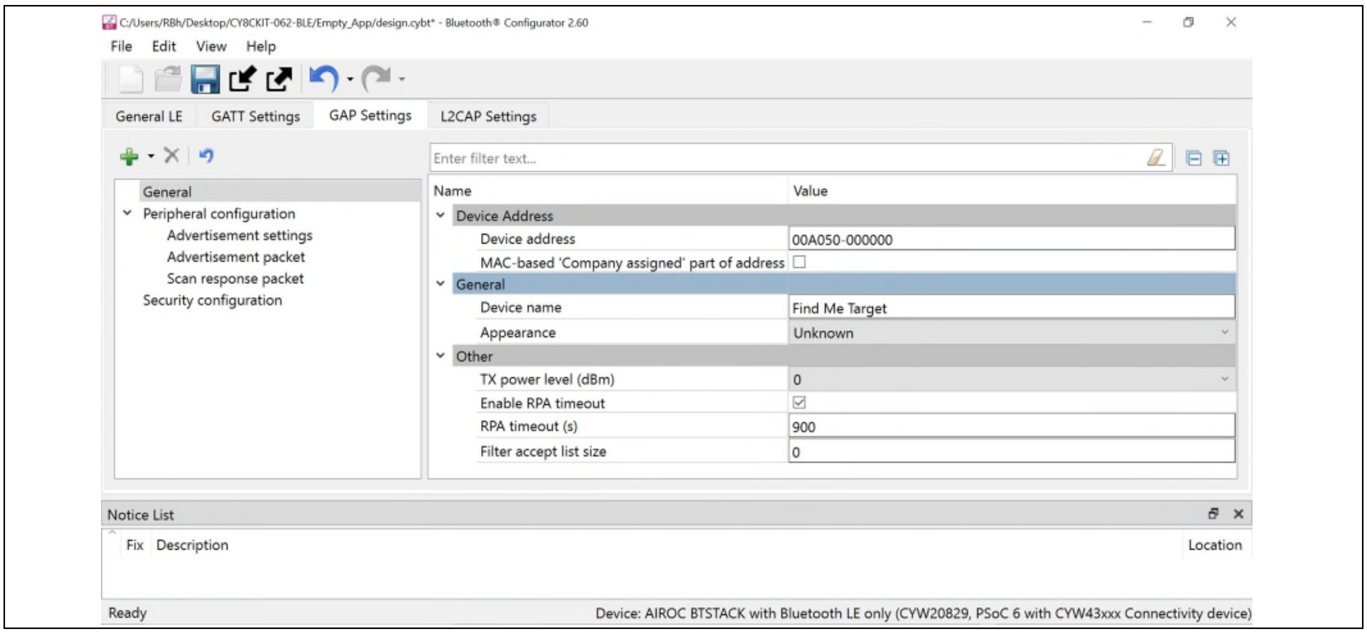
Name	Value
Service properties	
UUID	1802
Description	This service exposes a control point to allow a peer device to cause the device to immediately alert.
General	
Service declaration	Primary

Figure 22 **Final GATT database view**

There is a series of panels to cover GAP settings. The left menu provides access to all the panels (see Figure 23).

- Click the **GAP Settings** tab to display GAP options. The **General** panel appears by default.
- Enter Find Me Target as the **device name**

All other general settings use default values. This configures the device name that appears when a Host device attempts to discover your device.



C:/Users/RBh/Desktop/CY8CKIT-062-BLE/Empty_App/design.cybt* - Bluetooth® Configurator 2.60

General LE | GATT Settings | GAP Settings | L2CAP Settings

- General
- Peripheral configuration
  - Advertisement settings
  - Advertisement packet
  - Scan response packet
- Security configuration

Name	Value
Device Address	
Device address	00A050-000000
MAC-based 'Company assigned' part of address	☐
General	
Device name	Find Me Target
Appearance	Unknown
Other	
TX power level (dBm)	0
Enable RPA timeout	✓
RPA timeout (s)	900
Filter accept list size	0

Notice List

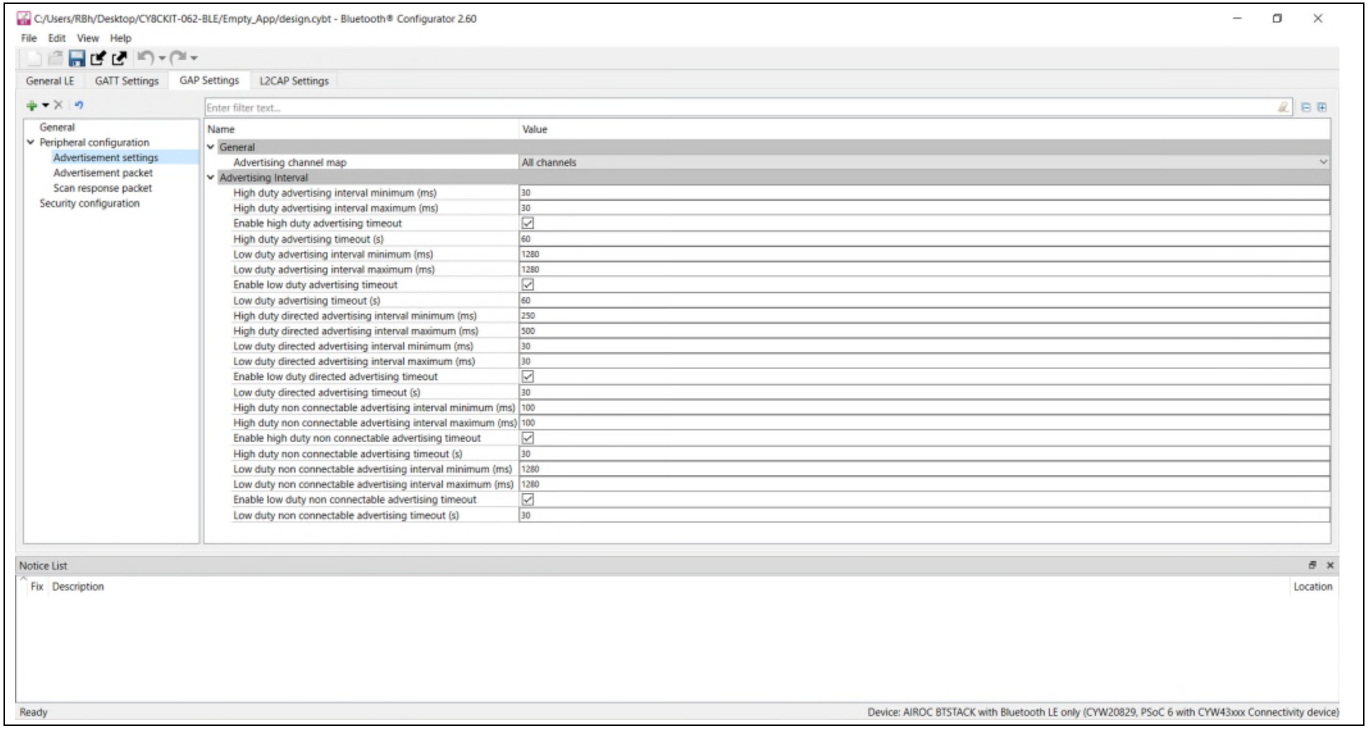
Fix | Description | Location

Ready | Device: AIROC BTSTACK with Bluetooth LE only (CYW20829, PSoC 6 with CYW43xxx Connectivity device)

Figure 23 General GAP Settings

Specify the GAP advertisement settings

- Click the **Advertisement settings** item in the left menu. Default values work for this application (see Figure 24). It uses a high-duty advertising interval of 30 ms and a low-duty advertising interval of 1280 ms. High-duty advertising allows quick discovery and connection but consumes more power due to increased RF advertisement packets.



C:/Users/RBh/Desktop/CY8CKIT-062-BLE/Empty_App/design.cybt - Bluetooth® Configurator 2.60

General LE | GATT Settings | GAP Settings | L2CAP Settings

- General
- Peripheral configuration
  - Advertisement settings
  - Advertisement packet
  - Scan response packet
- Security configuration

Name	Value
General	
Advertising channel map	All channels
Advertising Interval	
High duty advertising interval minimum (ms)	30
High duty advertising interval maximum (ms)	30
Enable high duty advertising timeout	✓
High duty advertising timeout (s)	60
Low duty advertising interval minimum (ms)	1280
Low duty advertising interval maximum (ms)	1280
Enable low duty advertising timeout	✓
Low duty advertising timeout (s)	60
High duty directed advertising interval minimum (ms)	250
High duty directed advertising interval maximum (ms)	500
Low duty directed advertising interval minimum (ms)	30
Low duty directed advertising interval maximum (ms)	30
Enable low duty directed advertising timeout	✓
Low duty directed advertising timeout (s)	30
High duty non connectable advertising interval minimum (ms)	100
High duty non connectable advertising interval maximum (ms)	100
Enable high duty non connectable advertising timeout	✓
High duty non connectable advertising timeout (s)	30
Low duty non connectable advertising interval minimum (ms)	1280
Low duty non connectable advertising interval maximum (ms)	1280
Enable low duty non connectable advertising timeout	✓
Low duty non connectable advertising timeout (s)	30

Notice List

Fix | Description | Location

Ready | Device: AIROC BTSTACK with Bluetooth LE only (CYW20829, PSoC 6 with CYW43xxx Connectivity device)

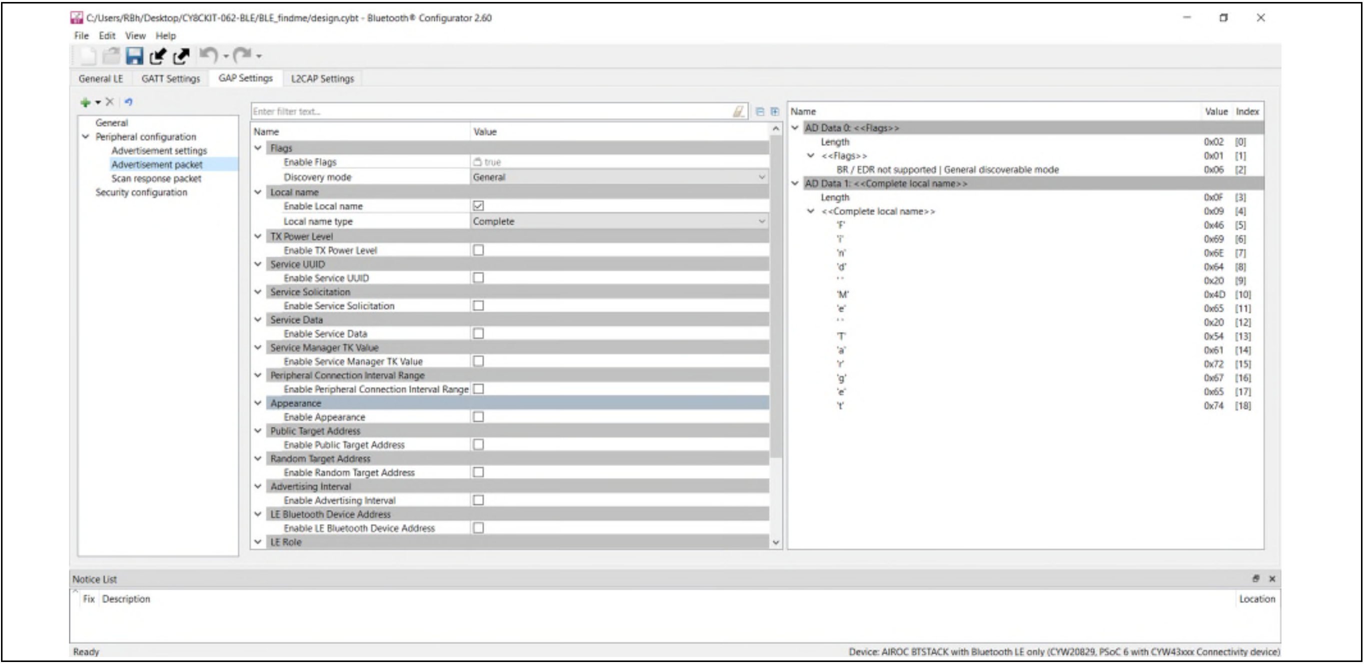
Figure 24 GAP advertisement settings

Specify the GAP advertisement packet settings

In this step, you specify the data for the advertisement packet(see Figure 25).

- Click the **Advertisement packet** item in the left menu, displays the panel
- The application uses a **General** discovery mode
- Select **Enable Local name** to include it in the advertisement packet, and **local name type** as **Complete**

This configures the advertisement packet of the device. As you add items, the structure and content of the advertisement packet appear to the right of the configuration panel.



Name	Value
Flags	
Enable Flags	true
Discovery mode	General
Local name	
Enable Local name	☑
Local name type	Complete
TX Power Level	
Enable TX Power Level	☐
Service UUID	
Enable Service UUID	☐
Service Solicitation	
Enable Service Solicitation	☐
Service Data	
Enable Service Data	☐
Service Manager TK Value	
Enable Service Manager TK Value	☐
Peripheral Connection Interval Range	
Enable Peripheral Connection Interval Range	☐
Appearance	
Enable Appearance	☐
Public Target Address	
Enable Public Target Address	☐
Random Target Address	
Enable Random Target Address	☐
Advertising Interval	
Enable Advertising Interval	☐
LE Bluetooth Device Address	
Enable LE Bluetooth Device Address	☐
LE Role	

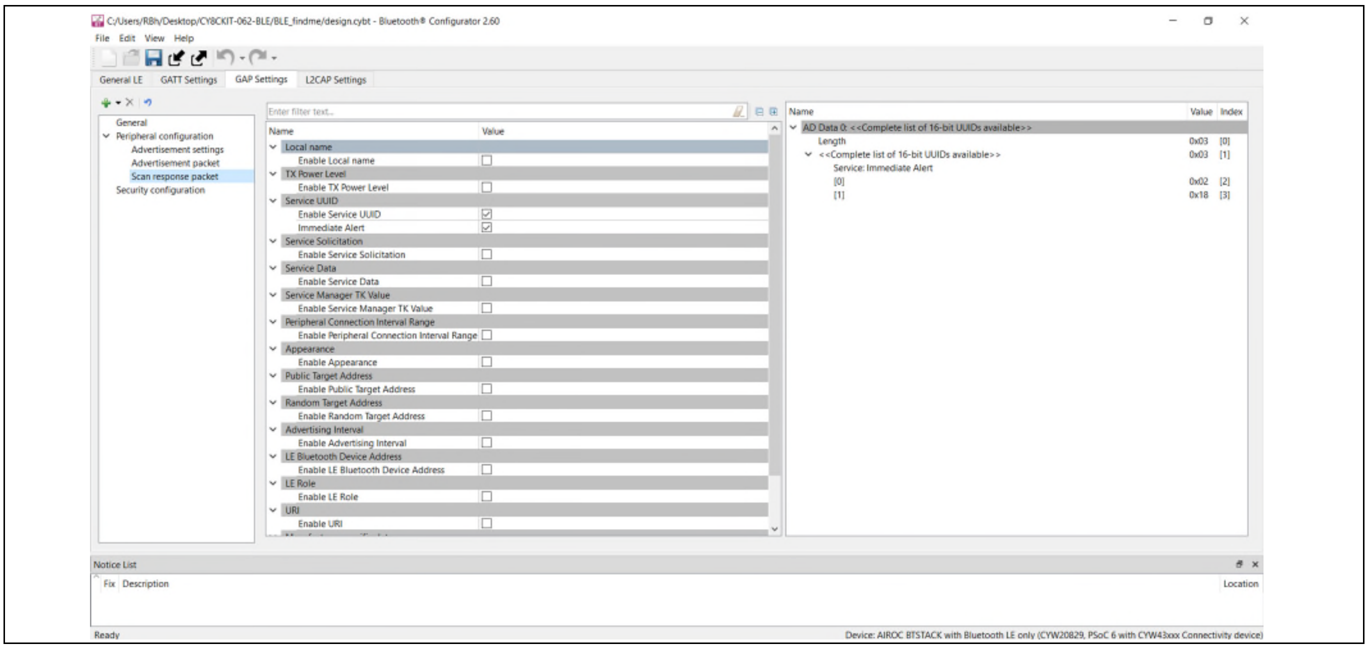
Name	Value	Index
AD Data 0: <<Flags>>		
Length	0x02	[0]
<<Flags>>	0x01	[1]
BR / EDR not supported | General discoverable mode	0x06	[2]
AD Data 1: <<Complete local name>>		
Length	0x0F	[3]
<<Complete local name>>	0x09	[4]
'F'	0x46	[5]
'i'	0x69	[6]
'n'	0x6E	[7]
'd'	0x64	[8]
' '	0x20	[9]
'M'	0x4D	[10]
'e'	0x65	[11]
' '	0x20	[12]
'T'	0x54	[13]
'a'	0x61	[14]
'r'	0x72	[15]
'g'	0x67	[16]
'e'	0x65	[17]
't'	0x74	[18]

Figure 25 **GAP advertisement packet settings**

Specify the scan response packet settings

In this step, you specify the data for the scan response packet (see Figure 26). Note that as you add values, the structure and content of the scan response packet appear to the right of the configuration panel.

- Click the Scan response packet item in the left menu. The panel appears
- Select **Enable Service UUID** to include that item in the response. It displays the available UUIDs. Select **Immediate Alert**.



Name	Value
Local name	
Enable Local name	☐
TX Power Level	
Enable TX Power Level	☐
Service UUID	
Enable Service UUID	✓
Immediate Alert	✓
Service Solicitation	
Enable Service Solicitation	☐
Service Data	
Enable Service Data	☐
Service Manager TK Value	
Enable Service Manager TK Value	☐
Peripheral Connection Interval Range	
Enable Peripheral Connection Interval Range	☐
Appearance	
Enable Appearance	☐
Public Target Address	
Enable Public Target Address	☐
Random Target Address	
Enable Random Target Address	☐
Advertising Interval	
Enable Advertising Interval	☐
LE Bluetooth Device Address	
Enable LE Bluetooth Device Address	☐
LE Role	
Enable LE Role	☐
URI	
Enable URI	☐

Name	Value	Index
AD Data 0: <<Complete list of 16-bit UUIDs available>>		
Length	0x03	[0]
<<Complete list of 16-bit UUIDs available>>	0x03	[1]
Service: Immediate Alert		
[0]	0x02	[2]
[1]	0x18	[3]

Figure 26 GAP scan response packet settings

You have now completed the required Bluetooth® settings for the application. Save your settings and close the bt-configurator.

4.5 Part 3: Write the application code

At this point in the development process, you have created an application, configured the hardware resources, and generated the configuration code, including the Bluetooth® Low Energy GATT database. This part examines the application code that implements the Find Me Target functionality.

Path	"Using CE directly" path (Evaluate existing code example (CE) directly)	"Working from scratch" path (Use existing code example (CE) as reference only)
Actions	Ignore Step 1 - the CE already has all the necessary source files added. Read through the Firmware description section to understand the firmware design.	Perform Step 1. Read through the Firmware description section to understand the firmware design.

The application code must do the significant tasks as follows:

- Perform system initialization, including the Bluetooth® stack.
- Implement Bluetooth® stack event handler functions for different events, such as an advertisement, connection, and attribute read/write requests.
- Implement user interface logic to update the LED state on the kit based on the events triggered.

1. Add files to your project (required only for the "Working from Scratch" flow)

   a) Download the mtb-example-btstack-freertos-findme code example from GitHub using the **Download ZIP** option. Unzip the downloaded folder and you will get the files for the Bluetooth® LE Findme code example.

b) Copy the following files/folder from the downloaded *mtb-example-btstack-freertos-findme* code example top-level folder to your *Empty_App* folder inside the ModusToolbox™ workspace folder.

- *main.c*
- *app_bt_utils.c*
- *app_bt_utils.h*
- *configs folder*

2. Add values to the variables in the **Makefile** of Empty_app as shown below:

   a) **COMPONENTS=FREERTOS WICED_BLE**
   The two components FREERTOS and WICED_BLE are required to include the files from FreeRTOS and btstack libraries for compilation.

   b) **DEFINES=CY_RETARGET_IO_CONVERT_LF_TO_CRLF CY_RTOS_AWARE**
   CY_RETARGET_IO_CONVERT_LF_TO_CRLF is provided by the retarget-io library to enable conversion of line feed (LF) into a carriage return followed by the line feed (CR & LF) on the output direction (STDOUT). CY_RTOS_AWARE must be defined to inform the HAL that an RTOS environment is being used.

4.6 Firmware description

This section explains the application firmware of the Find Me application. The important source files relevant to the user application-level code to this code example are listed in Table 3.

Table 3 Important user application-related source files

File name	Comments
GeneratedSource/cycfg_gatt_db.c, *GeneratedSource/cycfg_gatt_db.h*	These files reside in the *GeneratedSource* folder under the application folder. They contain the GATT database information generated using the Bluetooth® Configurator tool.
app_bt_utils.c *app_bt_utils.h*	These files consist of the utility functions that will help to debug and develop the applications easier with much more meaningful information.
main.c	Contains the `int main ()` function which is the entry point for execution of the user application code after device startup, also contains the code for the Bluetooth® stack event handler functions and code for the application user interface (in this case, the LED) functionality.
design.cybt	This file is used by the application to specify Bluetooth® configurations and the GATT database using the GUI tool bt-configurator.
Configs/COMPONENT_CM4/FreeRTOSConfig.h	This file is provided by the FreeRTOS library and copied into the application directory. This file has settings for the FreeRTOS kernel. The application can modify the settings based on the use case.
Makefile	This file contains settings for application build.

4.6.1 Bluetooth® Low Energy GATT database

The *cycfg_gatt_db.c* and *cycfg_gatt_db.h* files contain the Bluetooth® Low Energy GATT database definitions for the Find Me Target profile generated in the previous step using the Bluetooth® Configurator tool. The GATT database is accessed by both the Bluetooth® stack and the application code. The stack will directly access the attribute handles, UUIDs, and attribute permissions to process some of the Bluetooth® events. The application code will access the GATT database to perform attribute read/write operations. The relevant database structures are listed below.

- `gatt_database []`: This array contains the attribute handles, types and permissions. Note that this array does not have the actual attribute values, it is maintained as separate arrays as explained below.
- `GATT Value Arrays`: The actual GATT database containing the attribute values are declared as a series of `uint8_t` arrays under the section `GATT Initial Value Arrays` in *cycfg_gatt_db.c*. These arrays are also exposed as extern variables for application code access in the *cycfg_gatt_db.h* file. The FMP target application has these arrays defined by the name's `app_gap_device_name []`, `app_gap_appearance []`, and `app_ias_alert_level []`. `app_ias_alert_level []` is the Alert Level characteristic corresponding to the IAS service that the client will write to set the alert level. The application code performs the actual write to this attribute.
- `app_gatt_db_ext_attr_tbl []`: This array of structures is a GATT lookup table that conveys the mapping of the attribute handles defined in `gatt_database []` to the GATT value arrays. The application code uses this lookup table to perform the attribute read/write operations on the actual GATT arrays.

4.6.2 Bluetooth® stack configuration parameters

The *cycfg_bt_settings.c* and *cycfg_bt_settings.h* files contain the runtime Bluetooth® stack configuration parameters such as device name (`BT_LOCAL_NAME`) and core stack configuration parameters (`wiced_bt_cfg_settings []`). In the scope of this application note, will not be covering these parameters. However, you can see the comments in the source files to learn about these parameters. Note that the device name defined in the `BT_LOCAL_NAME` variable is the one that will be used on the peer device side to identify the device to establish a connection ("Find Me Target" in this case).

4.6.3 User application code entry

The *main.c* file contains the `int main ()` function. This function is the entry point for executing the user application code after device initialization is complete. In this code example, this function does the following:

- Initializes the BSP which includes initializing the target hardware. For example, it initializes system power management and device configuration. It performs other platform-specific initialization. If the BSP initialization fails, the app enters CY_ASSERT. If you are debugging your application then CY_ASSERT acts as a breakpoint.
- Initializes retarget-io to use debug UART port to view the trace messages and prints a startup message on the debug UART using the `printf` function.
- Initializes porting layer required for Bluetooth® communication between host and controller. The application needs to pass the HCI transport settings defined in the structure of type `cybt_platform_config_t`.
- Registers a Bluetooth® stack management callback function by calling `wiced_bt_stack_init ()`. The stack management callback function then typically controls the rest of the application based on Bluetooth® events. Typically, only a minimal application initialization is done in the `int main ()` function. Most application initialization is done in the stack callback function once the Bluetooth® stack has been enabled.

The stack callback function `app_bt_management_callback` is defined in *main.c*. A callback function is a function that is called by another function when a particular event happens. If the stack initialization fails, the application enters CY_ASSERT.

- Once all the above are initialized successfully, the application starts the FreeRTOS scheduler.

4.6.4 Bluetooth® stack events

The *main.c* file contains the application code logic to handle the different types of events generated by the stack. At a high level, two categories of events need to be handled:

- Bluetooth® stack management events
- GATT events

4.6.4.1 Bluetooth® stack management events

The callback function `app_bt_management_callback` handles events like Stack Enabled, Advertisement State Change, and Security-related events such as Pairing, and Key Exchange. This callback function is registered as a part of the `int main ()` function. See the `wiced_bt_management_evt_t` definition in *wiced_bt_dev.h* for the list of management events. It is not required for the application code to handle all the management events. The events handled depend on the application requirements. Figure 27 shows the execution logic for the stack management event handler in this code example.

The figure shows that only two management events (`BTM_ENABLED_EVT` and `BTM_BLE_ADVERT_STATE_CHANGED_EVT`) are handled in the stack management callback function.

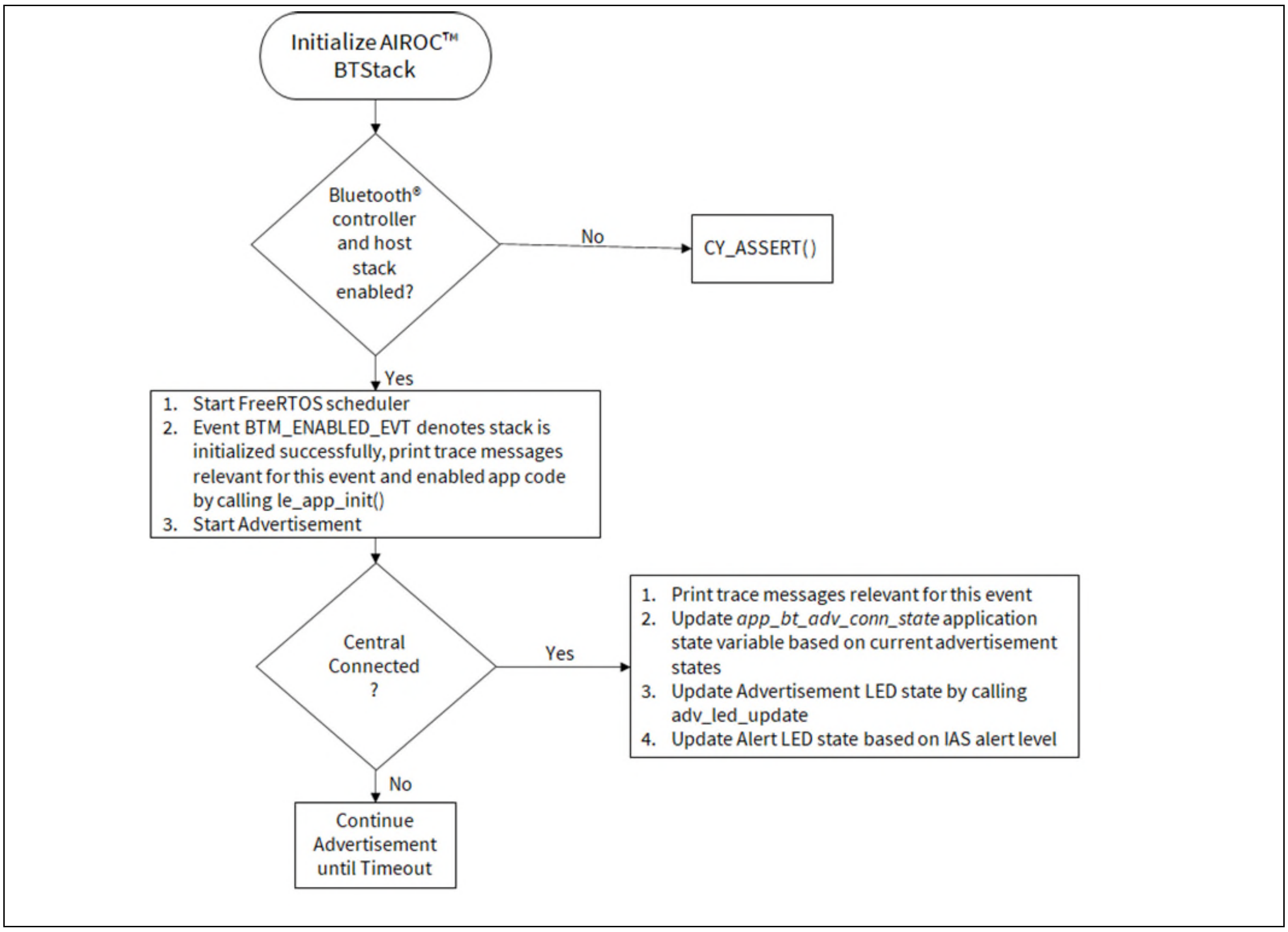

Initialize AIROC™ BTStack
Bluetooth® controller and host stack enabled?
No
CY_ASSERT()
Yes
1. Start FreeRTOS scheduler
2. Event BTM_ENABLED_EVT denotes stack is initialized successfully, print trace messages relevant for this event and enabled app code by calling le_app_init()
3. Start Advertisement
Central Connected ?
Yes
1. Print trace messages relevant for this event
2. Update *app_bt_adv_conn_state* application state variable based on current advertisement states
3. Update Advertisement LED state by calling adv_led_update
4. Update Alert LED state based on IAS alert level
No
Continue Advertisement until Timeout

Figure 27 Bluetooth® stack management event handler function flow

At this point, it is pertinent to discuss the `BTM_ENABLED_EVT` event, which is an essential management event that must be handled in all PSoC™ 6 Bluetooth® LE-based applications. It signifies that the Bluetooth® stack has been enabled. All the application code initialization is done only after the Bluetooth® stack has been enabled successfully by calling the `le_app_init()` function as Figure 27 shows.

The `le_app_init ()` function, defined in *main.c* performs the initialization tasks listed below. For any PSoC™ 6 Bluetooth® LE-based application that you create, you should add the required initialization code in this function.

- Initializes two or one PWM blocks used to control IAS LED and Advertisement LED. On BSP CY8CKIT-062-BLE, two LEDs are available on the board, therefore, two PWMs are initialized.
- Disables pairing by calling `wiced_bt_set_pairable_mode ()`. For this application, the pairing feature is not used.
- Configures the advertisement packet data by calling `wiced_bt_ble_set_raw_advertisement_data ()`. Look at this function definition in the code example to understand how to configure the elements of an advertisement packet.
- Registers the callback function to handle GATT events (`le_app_gatt_event_callback ()`) by calling `wiced_bt_gatt_register ()`.

- Initializes the GATT database (`gatt_database`) defined in *cycfg_gatt_db.c* by calling `wiced_bt_gatt_db_init ()`.
- As the final step of the initialization process, the device starts advertising by calling `wiced_bt_start_advertisements ()`.

4.6.4.2 GATT events

The `le_app_gatt_event_callback ()` function handles GATT events such as connection and attribute request events. This function is registered with a call to `wiced_bt_gatt_register ()` from the `le_app_init ()` function. Refer to the `wiced_bt_gatt_evt_t` definition in *wiced_bt_gatt.h* for the list of GATT events. It is not required for the application code to handle all the GATT events. The events handled depend on the application requirements. Figure 28 shows the execution logic for the GATT event handler in this code example. The figure shows that only two GATT events (`GATT_CONNECTION_STATUS_EVT` and `GATT_ATTRIBUTE_REQUEST_EVT`) are handled in the function.

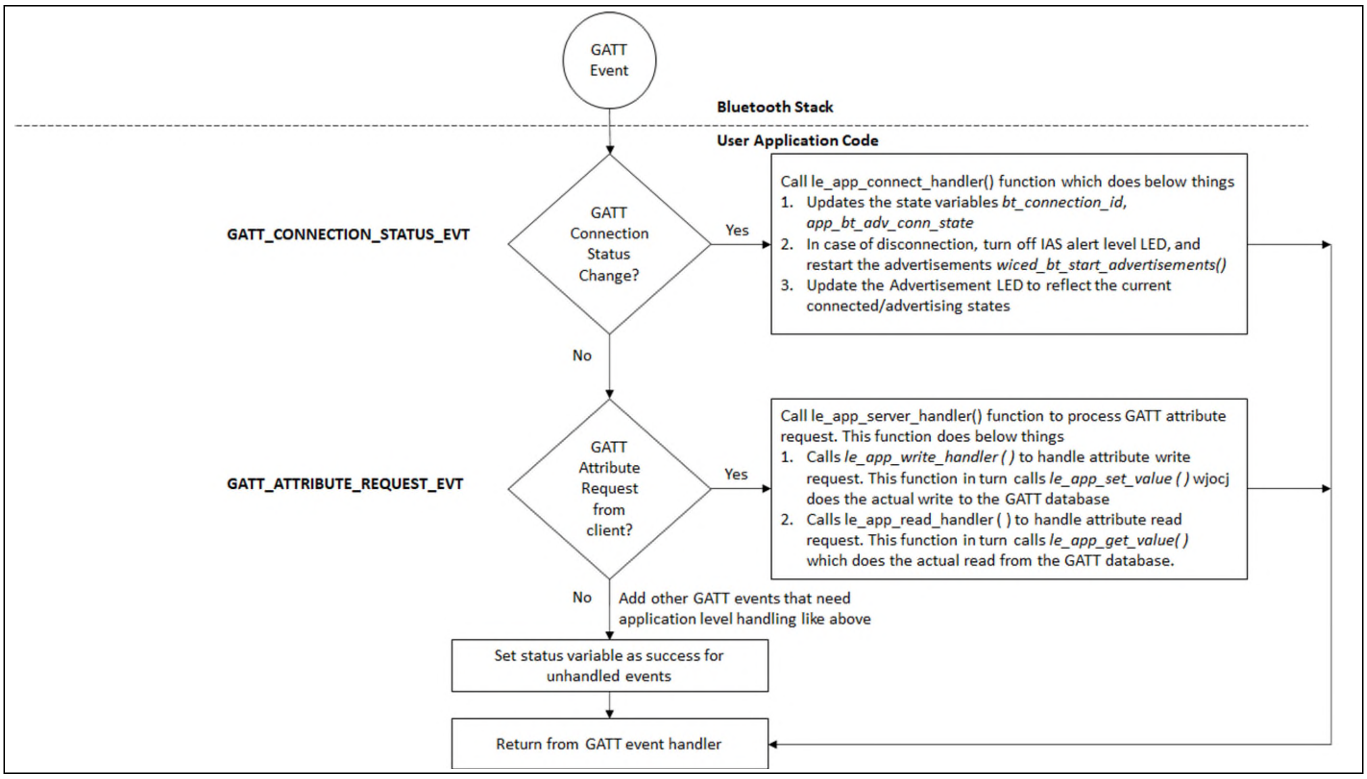

GATT Event
Bluetooth Stack
User Application Code
GATT_CONNECTION_STATUS_EVT
GATT Connection Status Change?
Yes
Call le_app_connect_handler() function which does below things
1. Updates the state variables bt_connection_id, app_bt_adv_conn_state
2. In case of disconnection, turn off IAS alert level LED, and restart the advertisements wiced_bt_start_advertisements()
3. Update the Advertisement LED to reflect the current connected/advertising states
No
GATT_ATTRIBUTE_REQUEST_EVT
GATT Attribute Request from client?
Yes
Call le_app_server_handler() function to process GATT attribute request. This function does below things
1. Calls le_app_write_handler ( ) to handle attribute write request. This function in turn calls le_app_set_value ( ) wjocj does the actual write to the GATT database
2. Calls le_app_read_handler ( ) to handle attribute read request. This function in turn calls le_app_get_value( ) which does the actual read from the GATT database.
No
Add other GATT events that need application level handling like above
Set status variable as success for unhandled events
Return from GATT event handler

Figure 28 GATT event handler

At this point, it is pertinent to discuss the `GATT_ATTRIBUTE_REQUEST_EVT` event, which is used to process the GATT Attribute read/write operations. Figure 29 gives information on the functions called in the case of a read or write operation. In this code example, when the Find Me Locator updates the IAS Alert Level characteristic on the PSoC™ 6 Bluetooth® LE device, `GATT_ATTRIBUTE_REQUEST_EVT` is triggered, which in turn calls the series of functions related to the attribute write request. At the end of the write operation, the `app_ias_alert_level []` function in the GATT database in *cycfg_gatt_db.c* gets updated with the alert level set by the Find Me Locator, and the LED is set appropriately to the alert level.

Figure 29 shows the function call chart summarizing the sequence of function calls for different stack events for this application. All these functions (except `adv_led_update ()`) are defined in *app_bt_event_handler.c*. Refer to the source code to understand the implementation details of these functions.

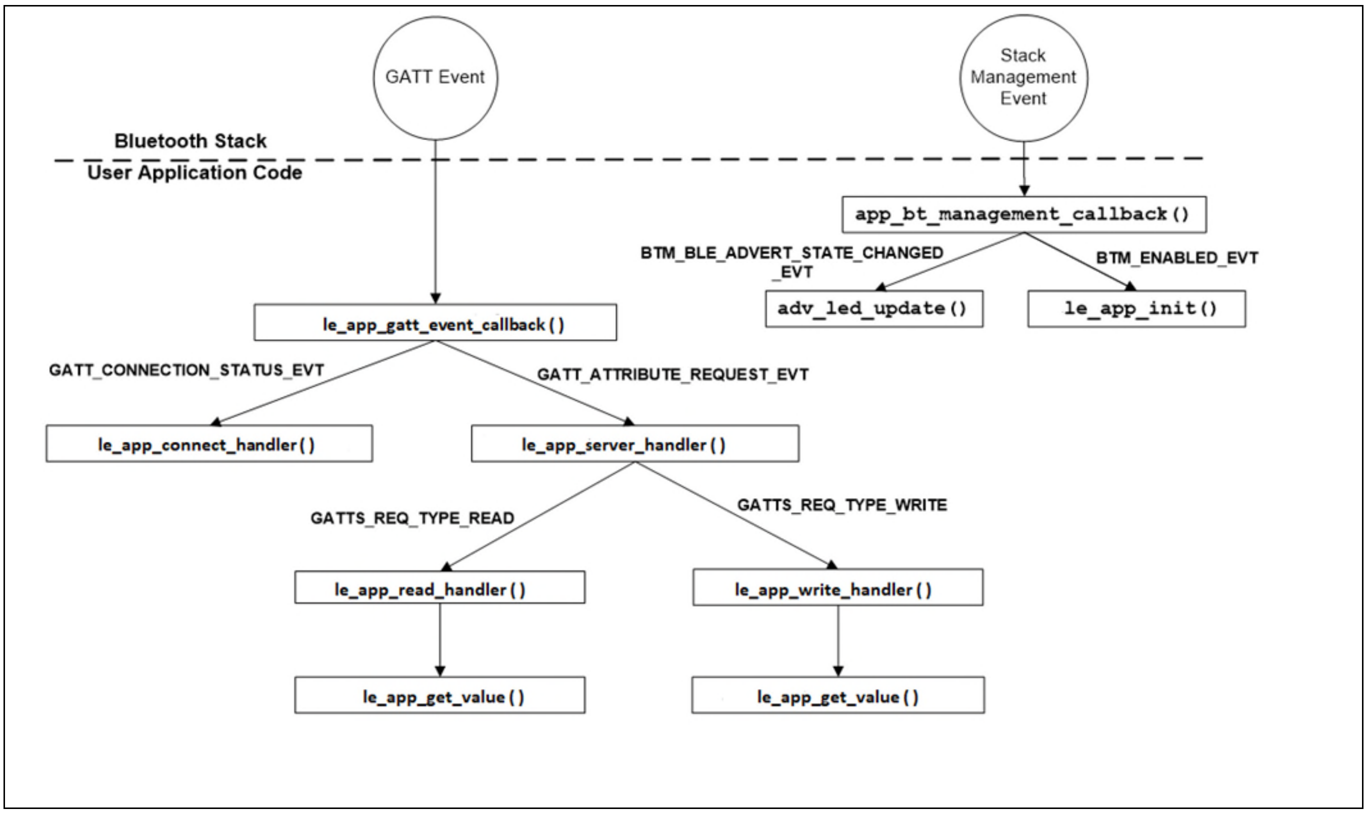


Figure 29 Bluetooth® stack events function call chart

4.6.5 User interface logic

The *app_user_interface.c* and *app_user_interface.h* files contain the application code to handle the user interface logic. The design uses two LEDs for the user interface, whose details are as follows:

- LED9 (red LED) on the kit indicates the advertising/connected state of the Bluetooth® Low Energy peripheral device. LED9 is in the OFF state when the device is not advertising, blinking state while advertising, and always in the ON state when connected to the peer device. See the `adv_led_update ()` function for implementation details. A global state variable `app_bt_adv_conn_state` is used to update the LED state. The `adv_led_update ()` function is called from two places in the application code:
  - The `app_bt_management_callback ()` function updates LED9 when the advertisement state changes (stack management event `BTM_BLE_ADVERT_STATE_CHANGED_EVT`)
  - The `ble_app_connect_callback ()` function updates LED9 when the connection state changes (GATT event `GATT_CONNECTION_STATUS_EVT`)
- LED8 (orange LED) on the kit indicates the IAS alert level characteristic when the device is connected to a peer device. When connected to a peer device, LED8 is in the OFF state for low alert, blinking state for mid alert, and ON state for high alert. When the device is not connected to any peer device, LED8 is in the ON state. See the `ias_led_update ()` function for implementation details. The `ias_led_update ()` function is called from two places in the application code:

- The `ble_app_set_value ()` function updates LED8 when an attribute write request to the IAS Alert Level characteristic is done from the client side.
- The `ble_app_connect_callback ()` function drives LED8 to the OFF state when a disconnection occurs (GATT event `GATT_CONNECTION_STATUS_EVT`)

4.7 Part 4: Build, program, and test your design

This section shows how to build the application and program the PSoC™ 63 MCU on the CY8CKIT-062-BLE kit. It also explains how to test the Find Me Profile Bluetooth® Low Energy design using the AIROC™ Bluetooth® Connect mobile app, and the USB – UART serial interface to view the Bluetooth® stack and application trace messages.

At this point, it is assumed that you have followed the previous steps in this application note to develop the Find Me Profile application.

Path	**"Using CE directly" path (Evaluate existing Code Example (CE) directly)**	**"Working from Scratch" path (Use existing Code Example (CE) as reference only)**
Actions	Perform all the steps in this section	Perform all the steps in this section

Connect the kit to your PC using the provided USB cable.

1. The USB – UART serial interface on the kit provides access to the UART interface of the CY8CKIT-062-BLE device. Use your favorite serial terminal application and connect to the USB – UART serial port. Configure the terminal application to access the serial port using the following settings:

   Baud rate: 115200 bps; Data: 8 bits; Parity: None; Stop: 1 bit; Flow control – None; New line for receiving data: Line Feed (LF) or auto setting
2. Build and Program the Application: In the project explorer, select the **<App Name>** project. In the Quick Panel, scroll to the **Launches** section, and click the **<App Name> Program (KitProg3_MiniProg4)** configuration as shown in Figure 30.

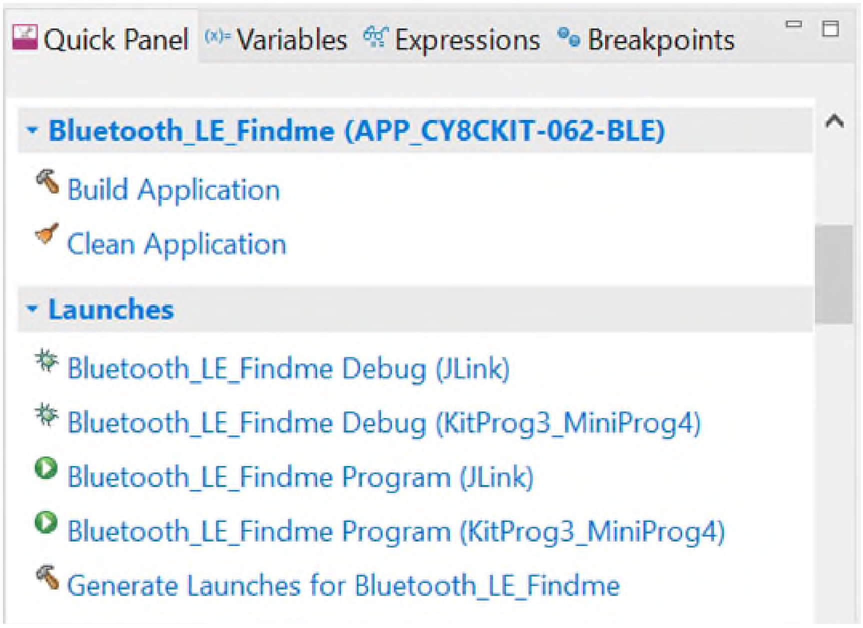

Quick Panel
Variables
Expressions
Breakpoints
Bluetooth_LE_Findme (APP_CY8CKIT-062-BLE)
Build Application
Clean Application
Launches
Bluetooth_LE_Findme Debug (JLink)
Bluetooth_LE_Findme Debug (KitProg3_MiniProg4)
Bluetooth_LE_Findme Program (JLink)
Bluetooth_LE_Findme Program (KitProg3_MiniProg4)
Generate Launches for Bluetooth_LE_Findme

Figure 30 Programming the CY8CKIT-062-BLE device from ModusToolbox™

Note: *If you encounter errors in the application build process, read the error messages in the IDE console window, and revisit the relevant previous steps in this document to check if you have missed or incorrectly done any of those steps.*

3. To test using the AIROC™ Bluetooth® Connect mobile app, follow these steps (see equivalent AIROC™ Bluetooth® Connect app screenshots in Figure 31 for iOS and Figure 32 for Android.
   a) Turn ON Bluetooth® on your Android or iOS device.
   b) Launch the AIROC™ Bluetooth® Connect app.
3. Press the reset switch on the CY8CKIT-062-BLE kit to start sending advertisements. The Red LED (LED9) starts blinking to indicate that advertising has begun. Advertising will stop after 120 seconds if a connection has not been established.
4. Swipe down on the AIROC™ Bluetooth® Connect app home screen to start scanning for Bluetooth® Low Energy peripherals; your device appears on the AIROC™ Bluetooth® Connect app home screen. Select your device to establish a Bluetooth® Low Energy connection. Once the connection is established, the Red LED (LED9) changes from a blinking state to an always ON state.
5. Select the 'Find Me' Profile from the carousel view. (Swipe left or right to rotate the carousel).
6. Select an Alert Level value on the Find Me Profile screen. Observe that the state of the Orange LED (LED8) on the device changes based on the alert level.

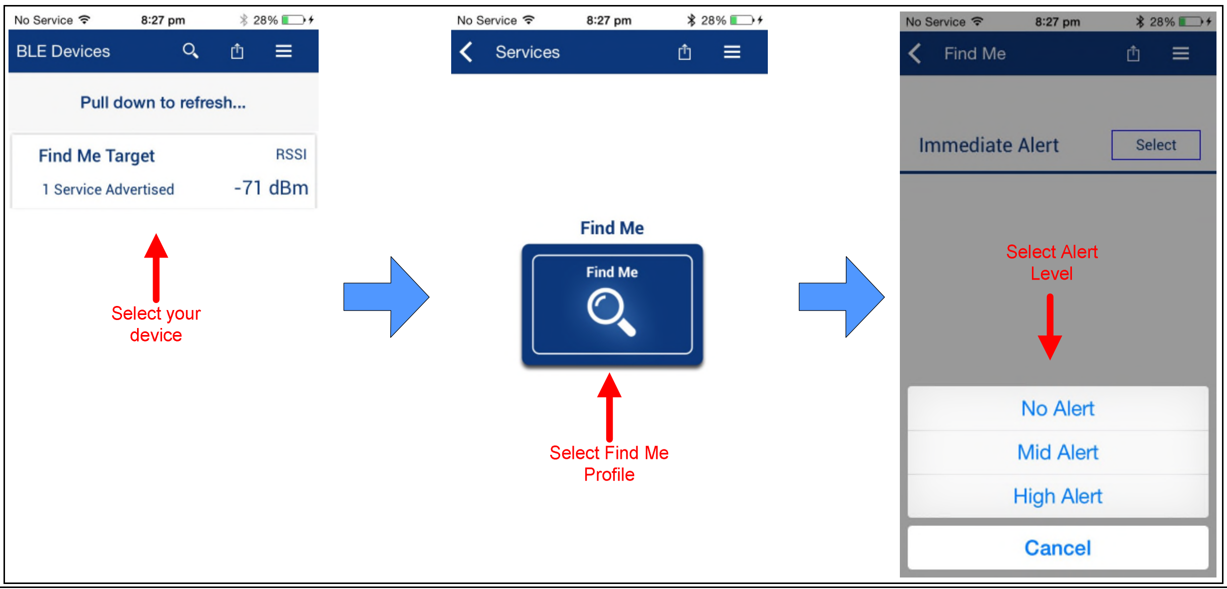


Figure 31 Testing with the AIROC™ Bluetooth® Connect app on iOS

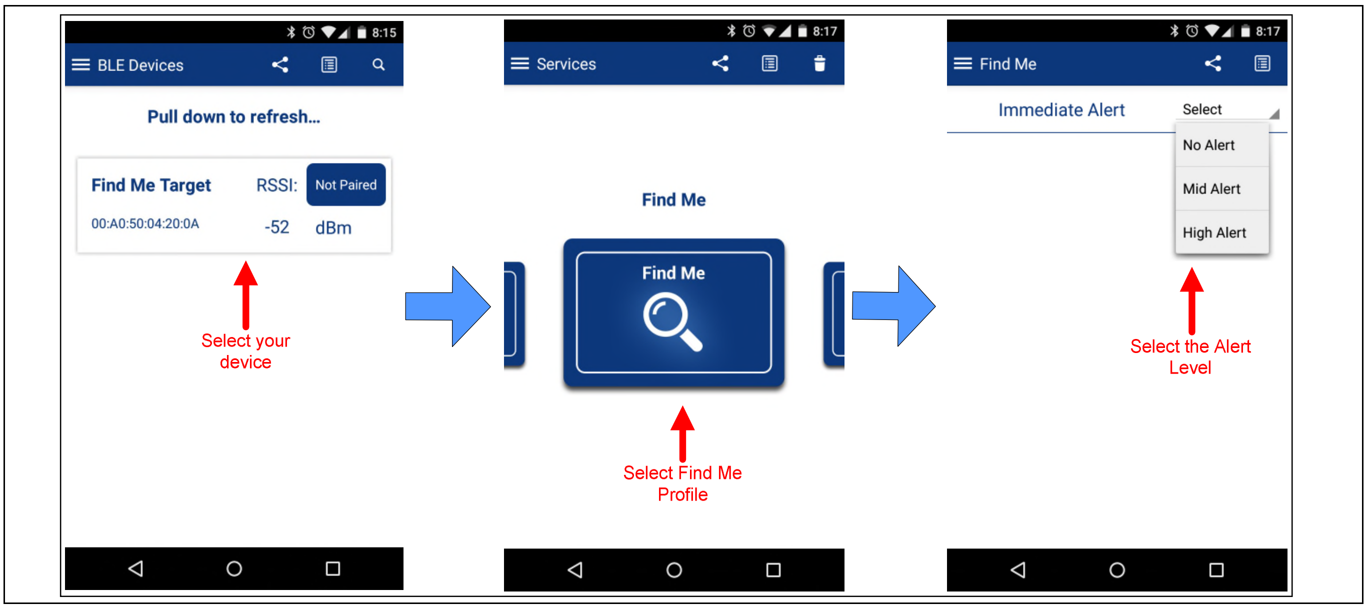
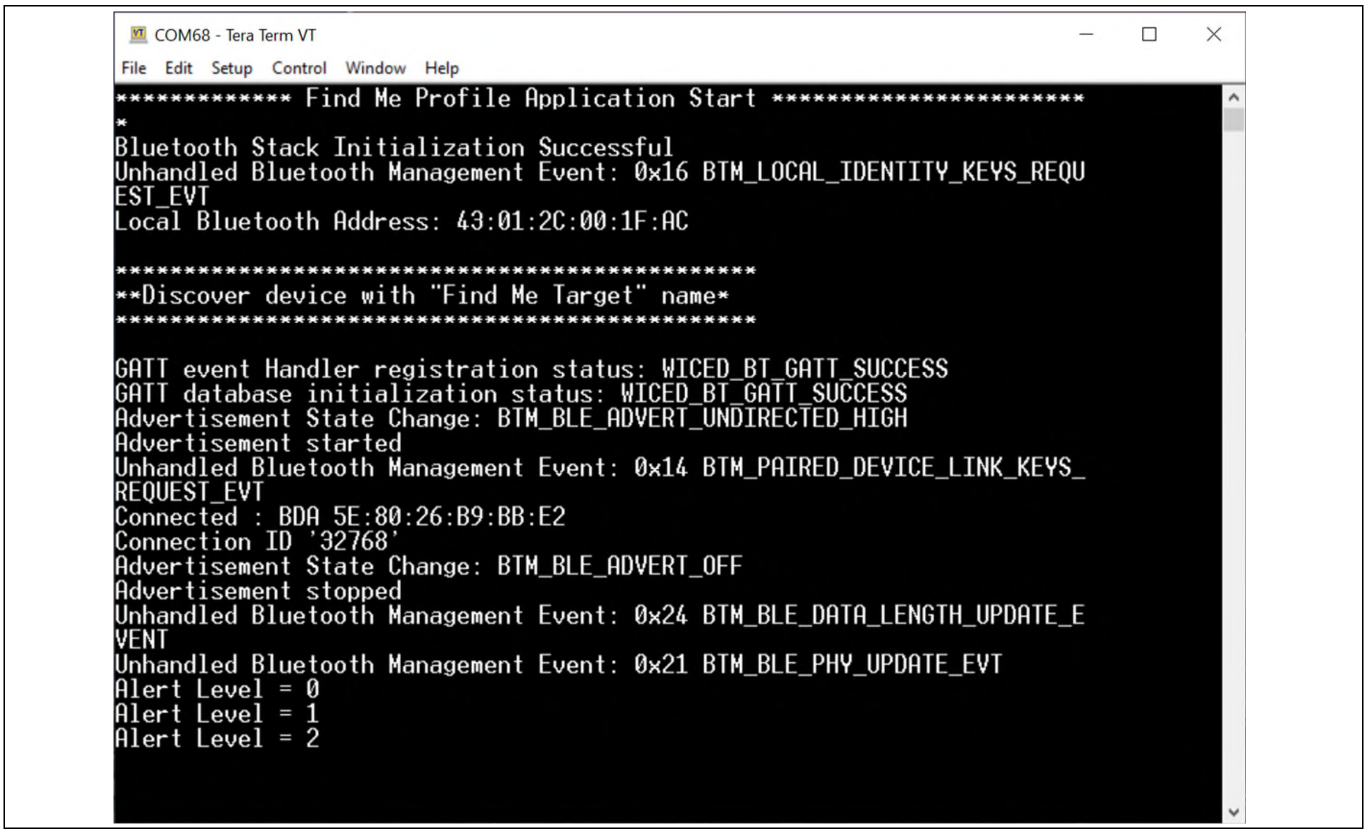


Figure 32 Testing with the AIROC™ Bluetooth® Connect app on Android

7. Use the USB – UART serial port to view the Bluetooth® stack and application trace messages in the terminal window as shown in Figure 33.



```
************* Find Me Profile Application Start ***********************
*
Bluetooth Stack Initialization Successful
Unhandled Bluetooth Management Event: 0x16 BTM_LOCAL_IDENTITY_KEYS_REQU
EST_EVT
Local Bluetooth Address: 43:01:2C:00:1F:AC

***********************************************
**Discover device with "Find Me Target" name*
***********************************************

GATT event Handler registration status: WICED_BT_GATT_SUCCESS
GATT database initialization status: WICED_BT_GATT_SUCCESS
Advertisement State Change: BTM_BLE_ADVERT_UNDIRECTED_HIGH
Advertisement started
Unhandled Bluetooth Management Event: 0x14 BTM_PAIRED_DEVICE_LINK_KEYS_
REQUEST_EVT
Connected : BDA 5E:80:26:B9:BB:E2
Connection ID '32768'
Advertisement State Change: BTM_BLE_ADVERT_OFF
Advertisement stopped
Unhandled Bluetooth Management Event: 0x24 BTM_BLE_DATA_LENGTH_UPDATE_E
VENT
Unhandled Bluetooth Management Event: 0x21 BTM_BLE_PHY_UPDATE_EVT
Alert Level = 0
Alert Level = 1
Alert Level = 2
```

Figure 33 Log messages on USB – UART serial port

My first PSoC™ 6 Bluetooth® Low Energy application

Now, you have successfully developed a simple Bluetooth® Low Energy application for the PSoC™ 6 device using Eclipse IDE for ModusToolbox™. For further learning about PSoC™ 6 device including technical documents, and additional code examples, see the References section.

5 Summary

This application note explored the PSoC™ 63 Bluetooth® MCU device architecture, the associated development tools, and the steps to create a simple Bluetooth® Low Energy application for CY8CKIT-062-BLE using ModusToolbox™. PSoC™ 6 Bluetooth LE is a Bluetooth® 5.2-compliant, standalone baseband processor with an integrated 2.4-GHz transceiver with support for Bluetooth® Low Energy. The device is intended for use in audio (source), sensors (medical, home, security), HID, and remote-control functionality as well as a host of other IoT applications.

A wealth of code examples, application notes, and other technical documents are available to help you quickly develop PSoC™ 6-based Bluetooth® applications that meet your end application requirements. See the References section to continue learning more about the PSoC™ 6 device and develop Bluetooth® applications.

References

[1] Application notes

- ModusToolbox™ 3.0 user guide

[2] Code examples

- Code examples for ModusToolbox™ software – Visit this code example for a comprehensive collection of code examples using ModusToolbox™ IDE.

[3] Device documentation

- PSoC™ 63 MCU datasheet

[4] Development kits

- CY8CKIT-062-BLE pioneer kit
- CY8CPROTO-063-BLE prototyping kit
- CYBLE-416045-EVAL Arduino evaluation board

[5] Tool documentation

- Eclipse IDE for ModusToolbox™ - The Infineon IDE for IoT designers

Revision history

Document revision	Date	Description of changes
**	2023-02-07	Initial release

Edition 2023-02-07

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

002-37038 Rev. **