AN97060

PSoC® 4 BLE and PRoC™ BLE – Over-the-Air (OTA) Device Firmware Upgrade (DFU) Guide

Author: Deepak John

Associated Part Family: All PSoC 4 BLE and PRoC BLE

Associated Code Examples: CE95351

Related Application Notes: For a complete list, click here.

To get the latest version of this application note, or the associated project file, please visit http://www.cypress.com/go/AN97060.

AN97060 provides guidelines on implementing the over-the-air (OTA) firmware upgrade feature for applications based on PSoC® 4 and PRoC™ BLE devices.

Contents

1 Introduction

The over-the-air (OTA) device firmware upgrade is essentially a bootload mechanism that uses a wireless link to update the firmware on a target device. Even though OTA can be performed over any wireless link, in the context of this application note, OTA means over a Bluetooth® Low Energy (BLE) link. The OTA feature in a BLE device can help to upgrade the device functionality or to fix firmware issues on devices that are already deployed in the field.

This application note briefly explains various OTA upgrade options and how you can select the right option for your product. It also provides testing and troubleshooting details to help with integration and deployment of the feature in the end product.

Before you read this application note, read AN73854, which gives a brief introduction to bootloader theory and technology and then shows how bootloaders are quickly and easily implemented in PSoC 3, PSoC 4, and PSoC 5LP devices using PSoC Creator™.

In addition to the OTA feature described in this application note, you can update the firmware for PSoC 4 BLE and PRoC BLE devices (see PSoC and PRoC Resources for more details) through other interfaces such as UART, I^2C, and SPI by using the PSoC Bootloader; see Related Application Notes. In the context of this application note, you will be going through OTA over BLE.

You can also access example projects related to the bootloader from PSoC Creator using the menu option **File** > **Code Example…**. Search for "bootloader" in the pop-up window.

2 PSoC and PRoC Resources

Cypress provides a wealth of data at www.cypress.com to help you to select the right PSoC (Programmable System on Chip) and PRoC (Programmable Radio on Chip) device for your design and quickly and effectively integrate the device into your design. 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 and PRoC BLE:

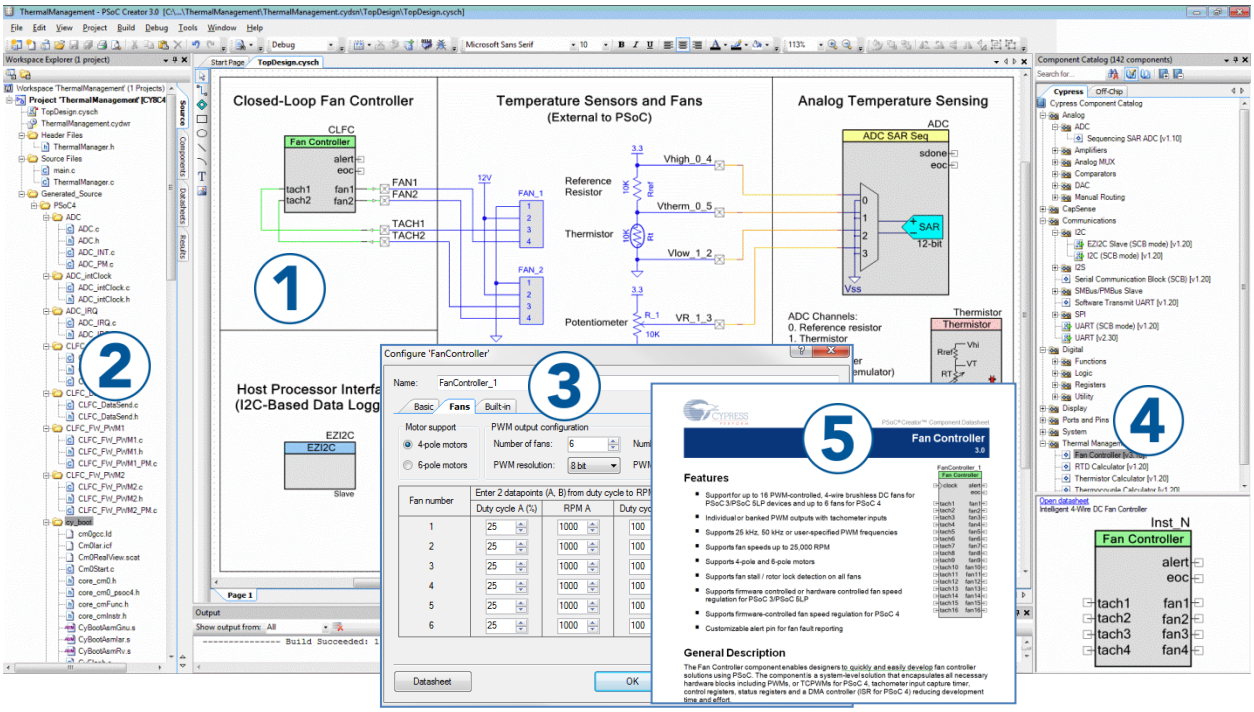
- **Overview:** Bluetooth Low Energy Portfolio, Cypress Wireless/RF Roadmap
- **Product Selectors:** PSoC 4 BLE or PRoC BLE. In addition, PSoC Creator includes a device selection tool.
- **Datasheets** describe and provide electrical specifications for PSoC 4 BLE and PRoC BLE device families.
- **CapSense® Design Guides:** Learn how to design capacitive touch-sensing applications with the PSoC 4, PSoC 4 BLE, and PRoC BLE families of devices.
- **Application Notes and Code Examples** cover a broad range of topics, from basic to advanced level. Many of the application notes include code examples.
- **Technical Reference Manuals (TRM)** provide detailed descriptions of the architecture and registers in each of the PSoC 4 BLE and PRoC BLE device families.
- **Development Kits:**
  - CY8CKIT-042-BLE and CY8CKIT-042-BLE-A BLE Pioneer Kits, enables customers to evaluate and develop BLE applications using the PSoC 4 BLE and PRoC BLE devices.
  - CY5682, PRoC BLE Touch Mouse RDK provides a production-ready implementation of a BLE or Bluetooth Smart touch mouse.
  - CY5672, PRoC BLE Remote Control RDK provides a production-ready implementation of a BLE or Bluetooth Smart remote control.
  - CY8CKIT-042 and CY8CKIT-040, PSoC 4 Pioneer Kits are easy-to-use and inexpensive development platforms. These kits include connectors for Arduino™ compatible shields and Digilent® Pmod™ daughter cards.
  - CY8CKIT-049 is a series of very low-cost prototyping platforms for sampling PSoC 4 devices.
  - CY8CKIT-001 is a common development platform for all PSoC family devices.
- The MiniProg3 device provides an interface for flash programming and debug.
- **CySmart™** is a BLE host emulation tool for Windows PCs. The tool provides an easy-to-use GUI to enable you to test and debug your BLE peripheral applications.
- **CySmart Mobile App** is a BLE or Bluetooth Smart utility developed by Cypress. CySmart can be used to connect to various BLE products and along with BLE development kits from Cypress, including the CY8CKIT-042-BLE PSoC 4 BLE Pioneer Kit, the CY5672 PRoC BLE Remote Control RDK, and CY5682 PRoC BLE Touch Mouse RDK.
- **Cypress's Custom BLE Profiles and Services:** Cypress's BLE Component (available as part of PSoC Creator) is regularly updated to include the GATT-based BLE profiles and services adopted by the Bluetooth Special Interest Group (SIG). Apart from the Bluetooth SIG-defined profiles and services, Cypress has defined several custom BLE profiles and services. These enable you to send data over BLE for features that are not supported by the Bluetooth SIG-specified standard BLE profiles and services. The profile and services can be utilized by devices communicating with a Cypress BLE device or by Cypress BLE devices communicating with each other.

3 PSoC Creator

PSoC Creator is a free Windows-based integrated design environment (IDE). It enables concurrent hardware and firmware design of systems based on PSoC 3, PSoC 4, PSoC 4 BLE, PRoC BLE, and PSoC 5LP. See Figure 1. With PSoC Creator, you can:

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

Figure 1. PSoC Creator Features



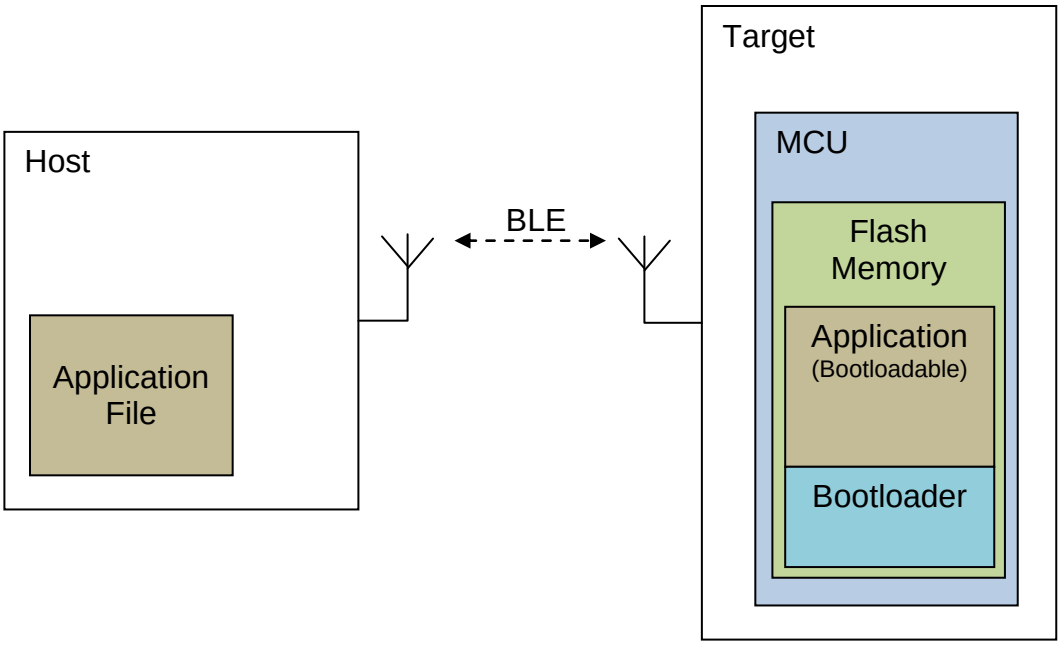
4 BLE OTA Bootloaders

The following terms are used frequently throughout this document. Knowing their definitions is important to the understanding of this application note.

- **Bootloader:** The portion of firmware that knows how to update the flash memory and is responsible for doing so
- **Bootloadable:** The portion of firmware that contains the application that is received over the air and is updated in the flash memory of the target device
- **Launcher:** A bootloader that is defined through the incorporation of a Bootloader Component without a communication Component. A Launcher can also be configured as a Launcher + Copier. The Copier is an additional functionality (built into the Launcher) that copies a previously saved Stack Application (PSoC Creator project containing the BLE stack) image from a temporary location to the Stack Application flash space (overwriting the old Stack Application).

The flash memory on the target MCU is split into two sections, as shown in Figure 2: the application (bootloadable image) and the bootloader. Other variations of this architecture are available to serve specific needs. To learn more about the implementation of bootloaders and their functional flow, see AN73854 and the Bootloader and Bootloadable Component datasheet.

Figure 2. BLE Bootloader System



The process of transferring the data (the bootloadable part) from a host (a PC or a smartphone) to the target device flash is called “bootloading” (also called a “bootload operation” or simply “bootload”), “firmware upgrade,” or “device firmware upgrade (DFU).” Another common term for bootloading is “in-system programming (ISP).”

Cypress provides three kinds of BLE bootloaders that you can add to any BLE project to enable OTA upgrades:

- External Memory OTA Bootloader
- Fixed Stack OTA Bootloader
- Upgradable Stack OTA Bootloader

Each bootloader has its own advantages and disadvantages. This section explains their architectures in brief and other design considerations. Based on the information presented here, you can select the best bootloader for your design.

4.1 External Memory OTA Bootloader

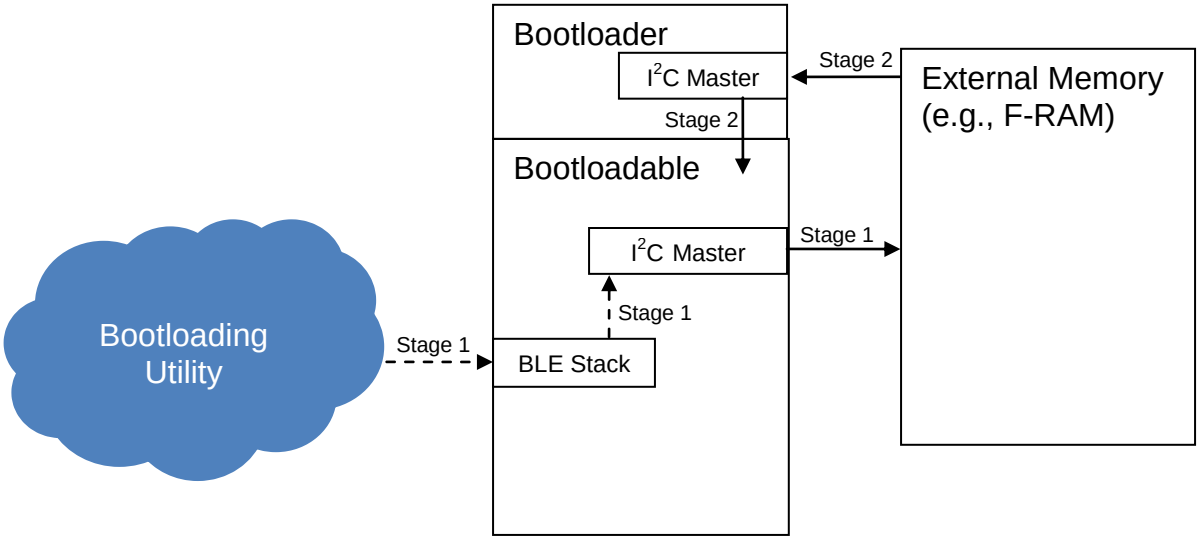
The External Memory OTA Bootloader uses the bootloader and single bootloadable image architecture available in PSoC Creator. It employs an external memory to temporarily store the bootloadable image. The BLE stack is located in the bootloadable section of the firmware; see Figure 3. Thus, a firmware upgrade can upgrade both the application and the BLE stack.

For example, an external memory is connected to the PSoC/PRoC BLE device via the I^{2}C bus in the BLE Pioneer Kit. The PSoC/PRoC BLE device is the I^{2}C master, and the external memory device (F-RAM™) is the I^{2}C slave.

In the External Memory OTA Bootloader implementation, bootload occurs in two stages. In stage 1, the bootloadable application currently present in the device flash memory receives the new bootloadable image over BLE. As each chunk of the application image is received, it is validated and written into the external memory through I^{2}C. On successful reception of the bootloadable image, the firmware passes control to the bootloader section.

In stage 2, the bootloader reprograms the device flash with the new application image (received in stage 1). The entire process is shown in Figure 3. Arrows indicate the direction of the data flow.

Figure 3. External Memory OTA Bootload Process



Advantages

- BLE stack and application can be upgraded together.
- Firmware is easy to implement because the bootloader and bootloadable are two separate projects that do not share memory.

Disadvantages

- Requires an external memory: This is an issue (increases BOM) if the external memory's only purpose is to enable OTA.
- A firmware upgrade takes longer than other OTA bootloaders. This is because the image has to be saved and retrieved from an external memory device using I^{2}C.
- Code related to I^{2}C is present in both the bootloader and bootloadable area, thereby increasing flash requirements.

An External Memory OTA Bootloader can be used with Cypress's programmable BLE parts.

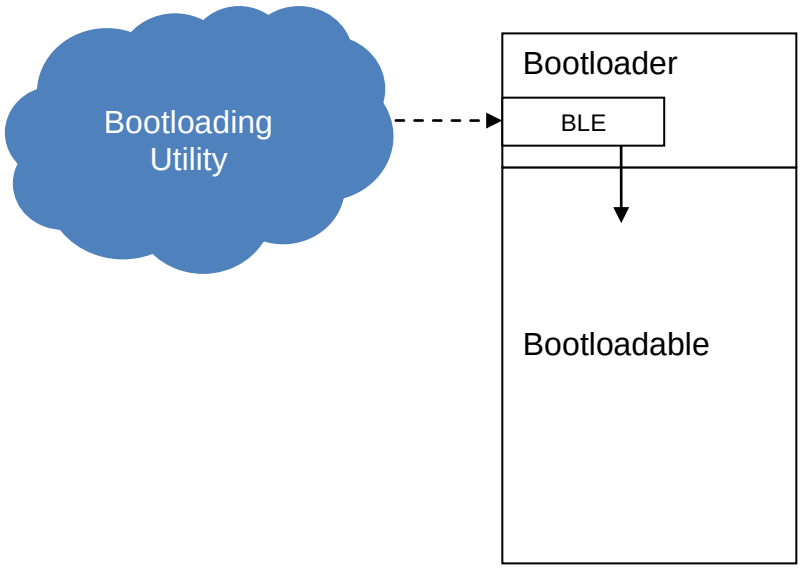
4.2 Fixed Stack OTA Bootloader

Similar to the External Memory Bootloader approach, the Fixed Stack OTA Bootloader also uses the bootloader and single bootloadable image architecture available in PSoC Creator. However, because the BLE stack is located in the bootloader memory, it cannot be upgraded via an OTA upgrade. The bootloadable application must link to the BLE APIs (refer to the BLE Component datasheet) located in the bootloader memory.

Ideally, the bootloader and the bootloadable images (implemented in two PSoC Creator projects) should have their own copy of the BLE stack; however, this increases memory consumption. As a practical approach, sharing the stack enables you to have the bootloader and bootloadable projects reuse the code related to the BLE Component. This helps to save a considerable amount of flash, which can be used by the application part of the firmware. This architecture also simplifies the firmware upgrade process as follows.

In the Fixed Stack OTA Bootloader implementation, bootload occurs in one stage. The firmware directly enters the bootloader mode and waits for the application image to be received over the BLE link. The received application image is directly written to the bootloadable area after successful validation. Figure 4 shows the Fixed Stack OTA bootload process with arrows indicating the data flow.

Figure 4. Fixed Stack OTA Bootload Process



Advantages

- BLE stack is reused across the bootloader and bootloadable projects, thereby saving on flash memory requirements.
- Upgrade time is faster because the received application image is directly written to the flash memory.
- An external memory or storage is not required.
- Application image upgrade is possible even if the current image is invalid.

Disadvantages

- BLE stack is part of the bootloader and cannot be upgraded (including the BLE profiles) via OTA.

The BLE stack is located in the bootloader area within the flash memory. It is unmodified during an OTA firmware upgrade. Therefore, the bootloader always recovers and is ready for an OTA upgrade even though a valid application image is absent. The Fixed Stack OTA Bootloader can be used with Cypress's programmable BLE parts.

4.3 Upgradable Stack OTA Bootloader

The Upgradable Stack OTA Bootloader uses a dual-application-image architecture. In this architecture, the available flash is divided into three sections: the Launcher + Copier image (from now on referred to as “Launcher”), the Stack Application image, and the User Application image (see Figure 5).

The Launcher image starts either the Stack Application or the User Application, depending on flags and the validity of the images. The Launcher also copies the latest Stack Application image (downloaded over the BLE link and stored at a temporary [User Application] location) to the Stack Application location of the flash while updating the Stack Application.

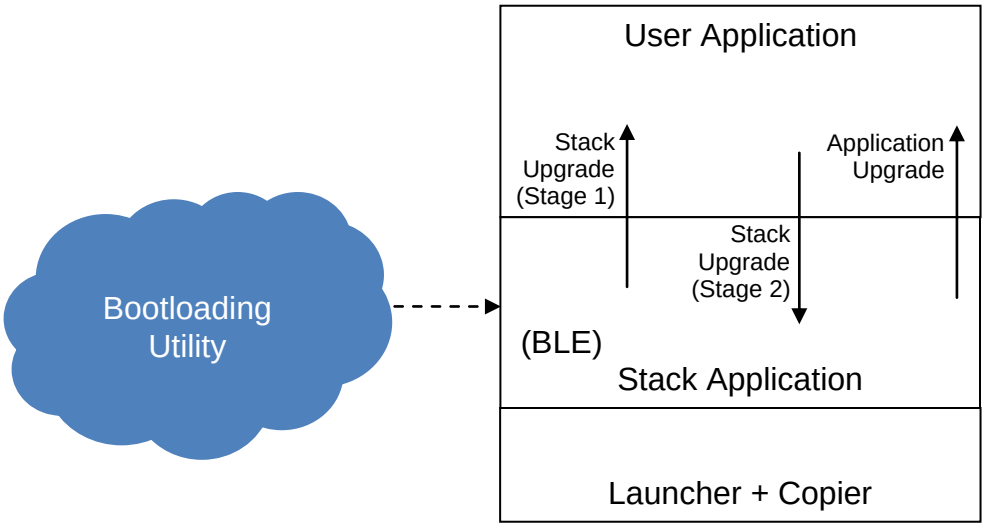
The Stack Application image contains the BLE stack or any other code that must be shared between the Stack Application image and User Application image. The Stack Application is responsible for upgrading the User Application image. It also downloads and temporarily stores the new version of the Stack Application image in the User Application region of the flash.

The User Application implements the functionality of the end application, such as a heart rate sensor, remote control, or a mouse. It links to the BLE APIs (see BLE Component datasheet) located in the Stack Application image. This allows the stack (and/or any other code in this region) to be shared by the Stack Application and User Application.

Ideally, the Stack Application and User Application images (implemented in two PSoC Creator projects) have their own copy of the BLE stack; however, this increases memory consumption. By sharing the stack (and/or any other code), you can have the Stack Application and User Application reuse the code related to the BLE stack. This helps to save a considerable amount of flash, which can be used by the application image. This architecture also provides two firmware upgrade options:

- **Application Upgrade:** Only the User Application image is upgraded. The Application Upgrade happens in a single stage. To enter the OTA upgrade mode, the firmware passes the control to the Stack Application, which receives the new User Application image. The Stack Application then directly writes the new User Application image to the corresponding region of the flash (see Figure 5).
- **Stack Upgrade:** Both the Stack Application and User Application are upgraded.
  - Stage 1: The firmware passes control to the Stack Application, which receives the new Stack Application image and writes it to a temporary location (User Application region) in the flash memory. The User Application becomes corrupted in this process (the new Stack Application image overwrites the existing User Application image).
  - Stage 2: After the download is complete, a software reset is initiated by the Stack Application, and the control passes to the Launcher image. It detects the image located in the temporary location (User Application region) and copies it to the Stack Application region (refer to Figure 5). Because at this point the User Application image is corrupted, an Application Upgrade has to be performed as well.

Figure 5. Upgradable Stack OTA Bootload Process



Advantages

- Flexibility of updating both BLE stack and application image
- The BLE stack is reused, so this approach results in reducing flash memory consumption.
- Faster upgrade time because the received BLE stack/application image is directly written to flash
- An external memory or storage is not required.

Disadvantages

- A higher capacity (256 KB) flash device must be used to store the new copy of the BLE stack when the BLE stack is being upgraded.

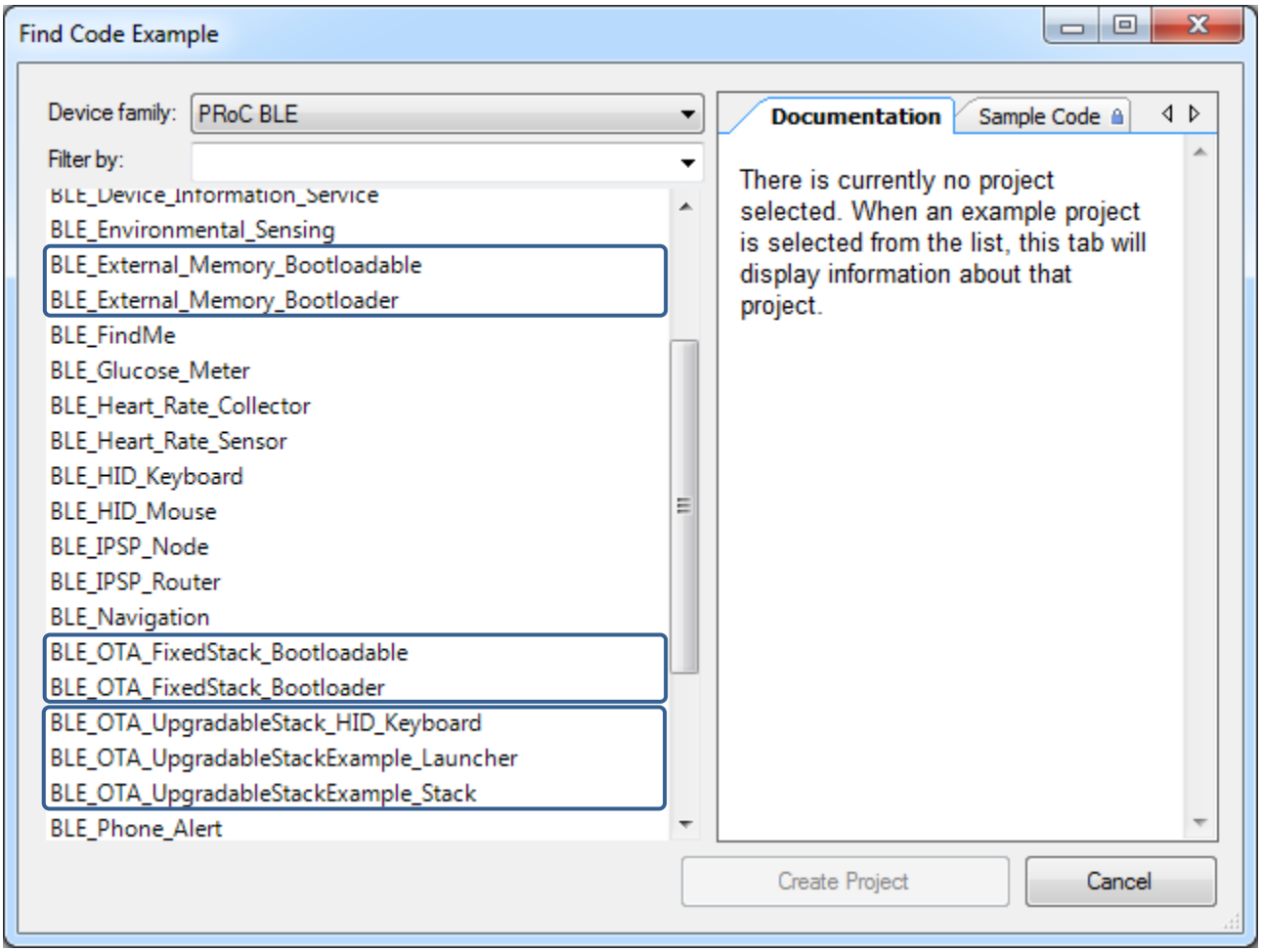
The Upgradable Stack OTA allows both the BLE stack and bootloadable/application image to be upgraded. Additionally, the BLE stack image must be temporarily stored in flash (while downloading), requiring a higher capacity flash device. Therefore, the Upgradable Stack OTA Bootloader option can be used only with Cypress's 256-KB BLE parts.

5 Adding Firmware OTA Bootloader Support to a Target Project

This section explains how to add an OTA bootloader to a target project. The achieve this, you will create an example target project and then add an OTA bootloader (steps to add all three kinds of OTA bootloader are explored individually) to this target project.

PSoC Creator ships with OTA-enabled example projects that can be accessed by navigating to **File** > **Example Project…** and choosing the appropriate **Device family** and **Filter by** keywords, as shown in Figure 6. Each example project comes with its own documentation. Review the documentation for specific implementation information. This section explains how to add the OTA feature to non-OTA application firmware. See BLE OTA Bootloaders to learn more about the different kinds of bootloaders. .

Figure 6. BLE Example Projects



5.1 Creating a Basic Example Target Project

You need to have a basic example project to which an OTA bootloader can be added. To do so, you will use the PWMExample project as the starting point. This project implements LED brightness control using the PWM Component in PRoC BLE/PSoC 4 BLE. Follow these steps to create the PWMExample project for PSoC 4 BLE/PRoC BLE devices. Skip this section if you already have a project to which you want to add one of the OTA bootloaders.

1. In PSoC Creator, choose **File** > **Code Example …**. This will launch the **Find Example Project** dialog, as shown in Figure 7.

2. In the **Find Example Project** dialog, set the **Device Family** filter to **PSoC 4200 BLE** and the **Filter by** keyword to **PWMExample,** as shown in Figure 7.
3. Select the **PWMExample** project from the list and then click **Create New Project**.
4. Select the **Create New Workspace** option for **Workspace**. Select a location for the new example project workspace (see Figure 8) and click **Finish**.

Figure 7. Find Example Project Dialog

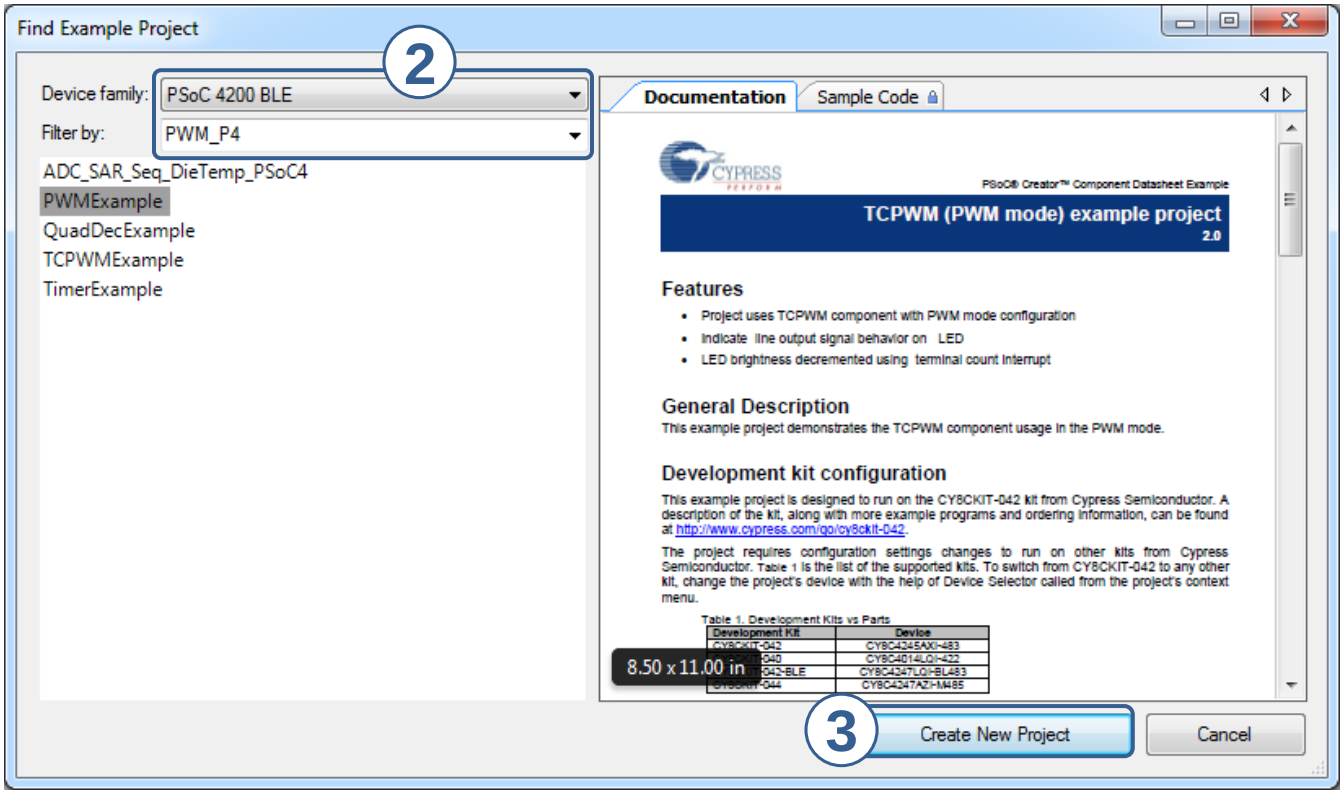
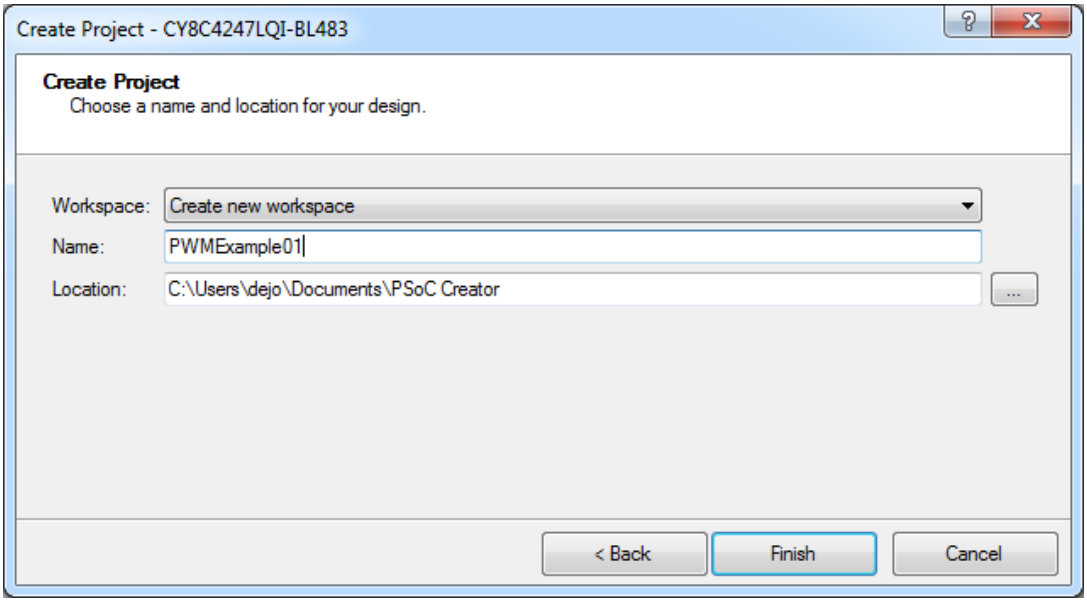


Figure 8. Create Project Dialog



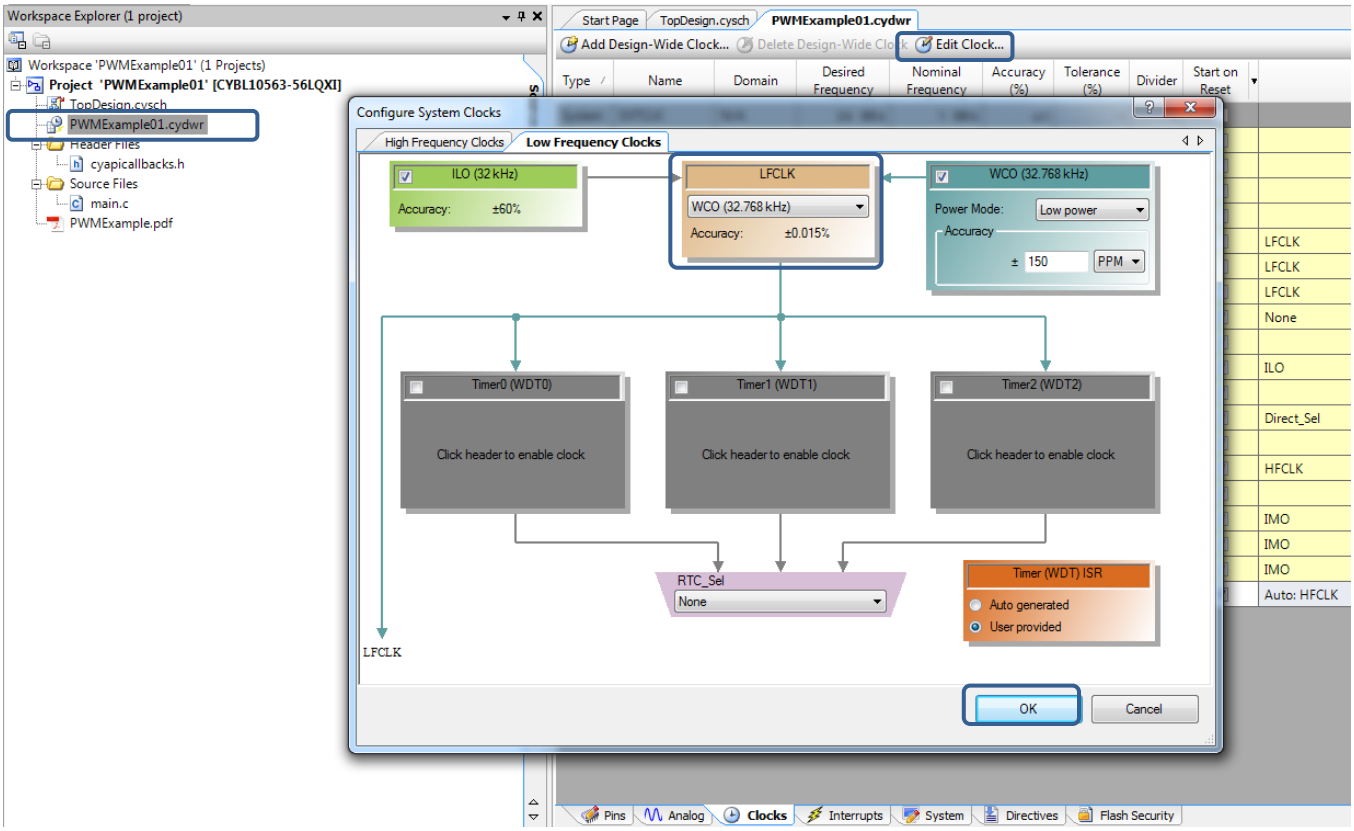
5. Open the **PWMExample01.cydwr** window by double-clicking the file from **Workspace Explorer**. Change the LED_GREEN default port assignment to P3[6] (see Table 1). This change is required to port the project and make it work with the CY8CKIT-042-BLE Pioneer Kit. Port P3[6] is connected to the green LED on the kit.

   Follow the instructions in the Selecting Another Device section to change/select the correct device for the target application.
6. Change the **LFCLK** source to **WCO (32.768 kHz)**.
   a. Open the **PWMExample01.cydwr** window, navigate to the **Clocks** tab, and then click **Edit Clock…** to open the **Configure System Clocks** dialog.
   b. In the **Configure System Clocks** dialog, navigate to the **Low Frequency Clocks** tab to change the LFCLK source (see Figure 9).
7. Save the project.

Table 1. Pin Mapping for PWMExample01

Pin Name	Port Assignment
LED_GREEN	P3[6]

Figure 9. LFCLK Selection



At this point, you have set up a basic PWM example project that can run on your target BLE device. You can build and program (to build and program at once, choose **Debug** > **Program**) the target device to see how this example project works. To learn more about PSoC Creator and programming the CY8CKIT-042-BLE Pioneer Kit, see AN91267 and AN94020.

The behavior of the LED can be controlled by changing the value of the `BRIGHTNESS_DECREASE` macro located in *main.c*. This macro value can be defined between 0 and 63000. Smaller values will result in slower brightness dimming cycle rates, while higher values will result in faster brightness dimming cycle rates.

5.2 Adding an External Memory OTA Bootloader

This section explains how to add an External Memory OTA Bootloader to the PWMExample project that you prepared in the Creating a Basic Example Target Project section. You should also review the BLE External Memory Bootloader and Bootloadable example project datasheets and source code for reference.

On the CY8CKIT-042-BLE Pioneer Kit, an I^2C-based F-RAM is used as the external memory. However, the External Memory OTA Bootloader can be made to work with other kinds of memory such as flash, which uses interfaces such as I^2C or SPI. An example of the SPI-based External Memory OTA Bootloader can be found here.

The following steps will help you set up an I^2C-based External Memory Bootloader in the PWMExample project for the PSoC 4 BLE Pioneer Kit.

1. Open the PWMExample01 project created in PSoC Creator (see Creating a Basic Example Target Project).
2. Add the BLE External Memory Bootloader example to the PWMExample01 workspace (see Adding an Example Project to an Existing Workspace).
3. Set the BLE_External_Memory_Bootloader01 project as the active project by right-clicking the project in **Workspace Explorer** and selecting **Set As Active Project**.
4. Follow the instructions in the Selecting Another Device section to change/select the correct device for the target application.
5. Build the BLE_External_Memory_Bootloader01 project by choosing **Build** > **Build BLE_External_Memory_Bootloader01**. The project should build without any errors.
6. Open another instance of PSoC Creator and create a new workspace for the BLE External Memory Bootloadable example project (see Creating an Example Project Workspace). You will be using the necessary files and code snippets from this project to enable the External Memory Bootloader.
7. Set the PWMExample01 project as the active project.
8. Copy the following Components from the *TopDesign.cysch* of the BLE External Memory Bootloadable project (created in step 6) to the *TopDesign.cysch* of the PWMExample01 project created in the Creating a Basic Example Target Project section. Configuration of the Components added in this step is covered in the BLE External Memory Bootloadable example project datasheet.

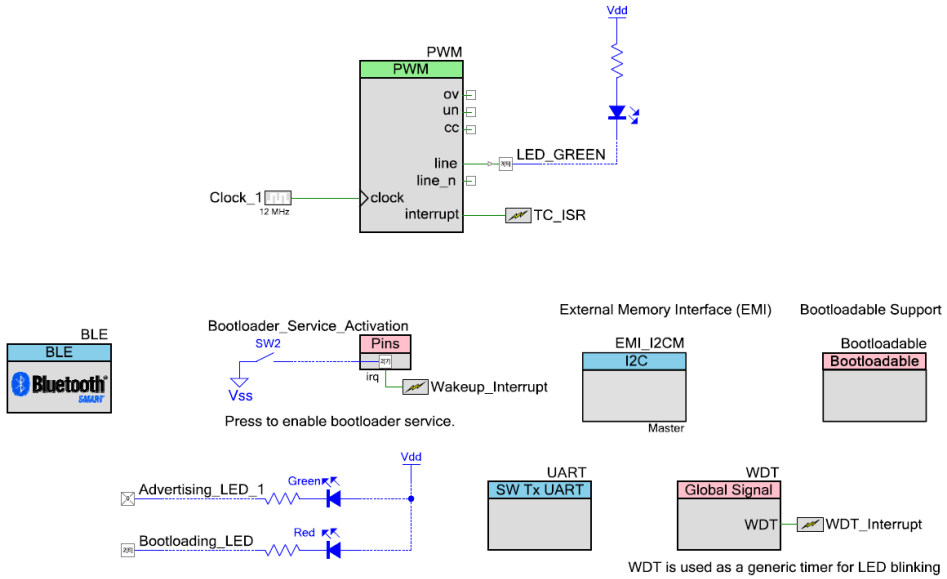
   - BLE
   - Bootloader_Service_Activation
   - Bootloading_LED
   - Advertising_LED_1
   - Bootloadable
   - EMI_I2CM
   - UART
   - WDT
   - WDT_Interrupt
   - Wake_Interrupt

   If the project of your choice already contains a BLE Component, omit the BLE Component in this step and instead follow the instructions in the Adding Bootloader Service section to add and configure a bootloader service in the existing BLE Component.

   The BLE, EMI_I2CM, and Bootloadable Components are required to implement the External Memory OTA. Bootloader_Service_Activation and Wakeup_Interrupt are used as a trigger to enter bootloader mode. If your project has some other mechanism to enter bootloader mode, avoid copying the Bootloader_Service_Activation and Wakeup_Interrupt Components. The UART Component is used to print debug messages. All other Components are not critical for implementing the External Memory OTA and can be omitted in this step. They are required to prevent compile time errors.

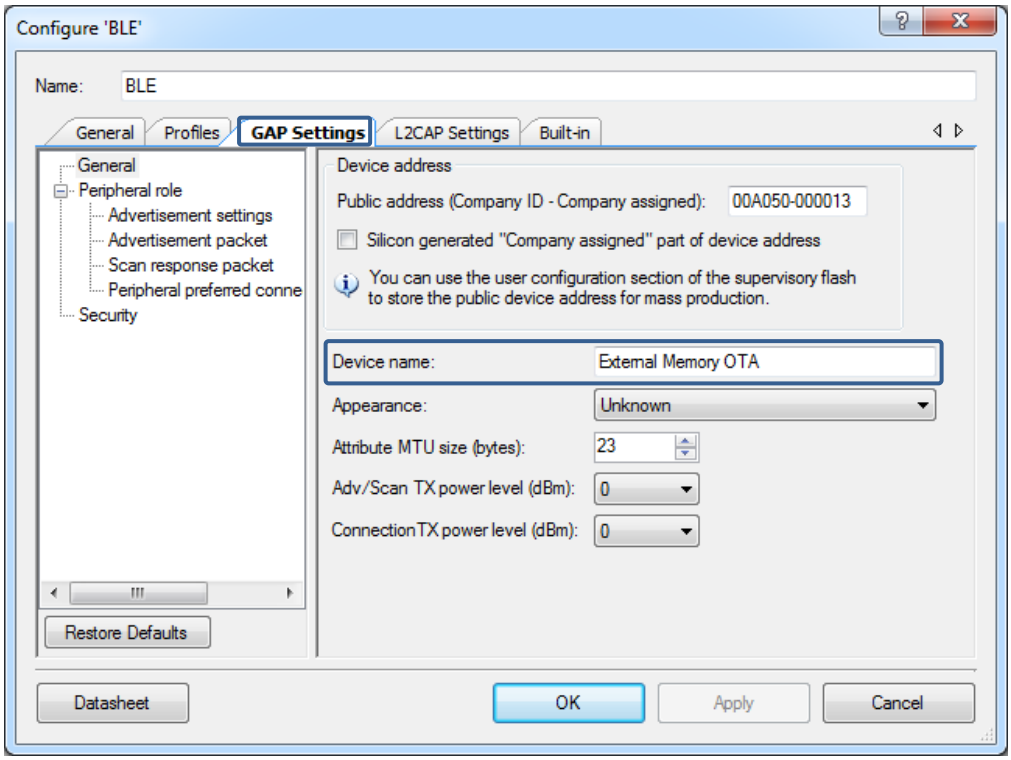
   At the end of this step, your schematic should look similar to Figure 10.

Figure 10. TopDesign.cysch View After Adding Necessary Components



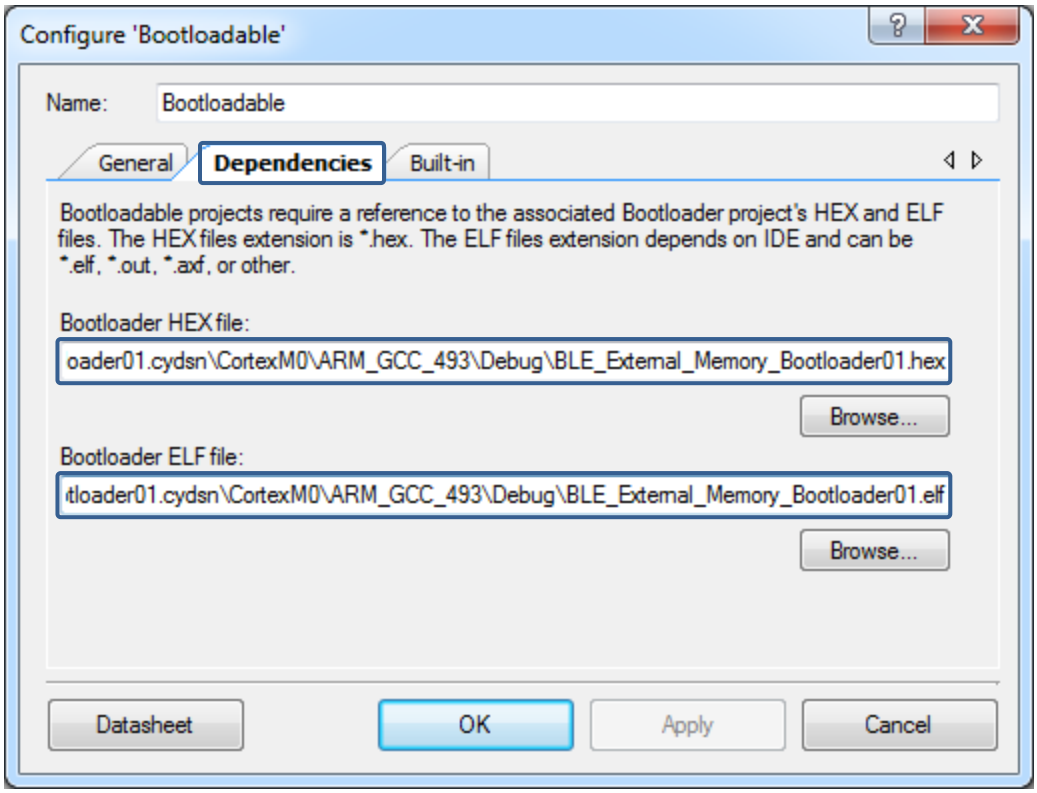
9. Change the **Device name** to “External Memory OTA” in the **GAP Settings** tab in the BLE Component configuration window (Double-click the BLE Component. See Figure 11.

Figure 11. BLE Component Configuration Showing GAP Settings



10. Specify the paths to the bootloader project HEX and ELF files:
    a. Double-click on the Bootloadable Component.
    b. Navigate to the **Dependencies** tab and link the **Bootloader HEX file** to the *BLE_External_Memory_Bootloader01.hex* file (located at …*BLE_External_Memory_Bootloader01.cydsn**CortexM0*\<*compiler version*>\<*build configuration*>\), as shown in Figure 12.
    After you have selected the HEX file, the corresponding ELF file will be automatically selected for you.
    c. Click **OK** to close the Bootloadable Component configuration dialog.

Figure 12. Bootloadable Component Configuration



11. To assign the correct pins for the Components added in step 8, open *PWMExample01.cydwr* and navigate to the **Pins** tab. Configure the pins as described in Table 2.

Table 2. Pin Mapping for PWMExample01

Pin Name	Port Assignment
EMI_I2CM:scl	P5[1]
EMI_I2CM:sda	P5[0]
UART:tx	P1[5]
Advertising_LED_1	P3[7]
Bootloader_Service_Activation	P2[7]
Bootloading_LED	P2[6]
LED_GREEN	P3[6]

12. Copy the following files from the bootloadable project workspace directory to the PWMExample01 project workspace directory and add them to the PWMExample01 project. These files implement a part of the OTA and debug functionality.

a. To add header files, right-click the **Header Files** folder in **Workspace Explorer** and choose **Add** > **Existing Item…**. Browse and select the necessary files and click **Open**.

b. To add source files, right-click the **Source Files** folder in **Workspace Explorer** and choose **Add** > **Existing Item…**. Browse and select the necessary files and click **Open**.

- *Common.h*
- *debug.h*
- *main.h*
- *Options.h*
- *OTAMandatory.h*
- *OTAOptional.h*
- *Common.c*
- *debug.c*
- *OTAMandatory.c*
- *OTAOptional.c*

The *OTAMandatory.c/h* files implement all the required functionality for enabling External Memory OTA. All other files are not mandatory and are copied over to prevent compile time errors. The *debug.h/c* files implement UART-based debug message printing. The *Common.c/h* files implement helper functions for watchdog timer (WDT), LEDs, debug message printing, and setting bootloader service visibility. The *main.h* file contains defines for LED states and enabling/disabling bootloader service. The *Options.h* file contains defines to enable/disable debugging and encryption. The *OTAOptional.c/h* files implement encryption and decryption of information being stored in external memory. This can be enabled by setting the `ENCRYPT_ENABLED` macro to `YES` in *Options.h*.

13. Additional code (from the BLE External Memory Bootloadable project) must be added to *main.c* of the PWMExample01 project to enable the bootload or OTA functionality. The changes are numerous and cannot be individually listed; instead, you can download and use the modified file. Replace the PWMExample01 project *main.c* file with *main.c* from *…\Code\External Memory OTA*.
14. After adding a Bootloader/Bootloadable Component, debug support is disabled in PSoC Creator. However, once the firmware begins execution, you can attach to the target device (**Debug** > **Attach to Running Target…**) or use UART messaging to debug the firmware. Make sure that an adequate heap (0x400 bytes) and stack (0x800 bytes) size has been set for the project to work correctly when UART is enabled. Failure to do so will result in unpredictable behavior of the firmware; see the Debugging section for more details.
15. Build the PWMExample01 project by choosing **Build** > **Build PWMExample01**. The PWMExample01 project should build without any errors.
16. Program the PSoC BLE Pioneer Kit by choosing **Debug** > **Program**.

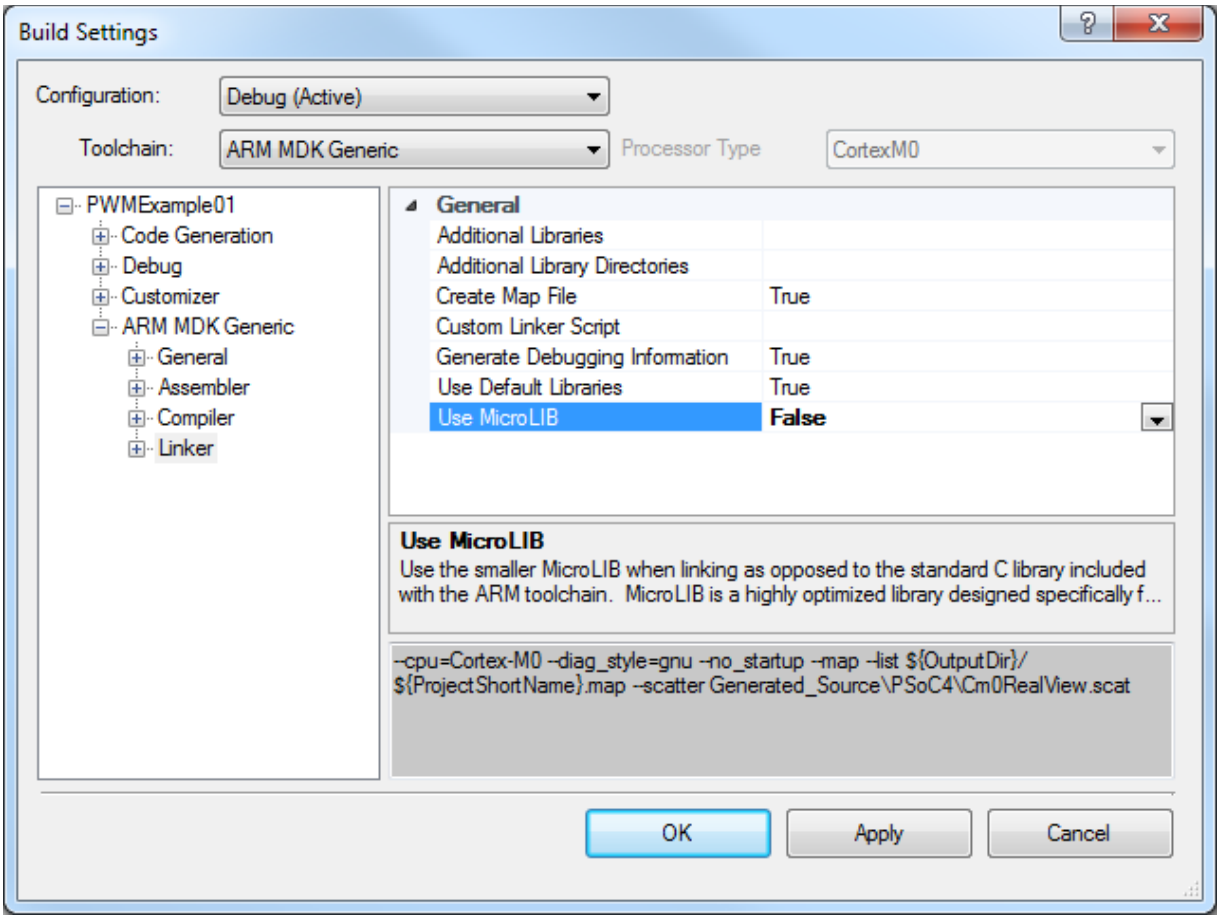
After programming is complete, the green LED will cycle through high to low brightness levels. At this point, you can perform a device firmware upgrade by following one of the methods described in the Performing an OTA Upgrade section. In addition, test the OTA feature by following the steps listed in the Testing the OTA Feature section.

5.3 Adding a Fixed Stack OTA Bootloader

This section explains how to add a Fixed Stack OTA Bootloader to the PWMExample project that you prepared in the Creating a Basic Example Target Project section. You should also review the BLE Fixed Stack Bootloader and Bootloadable example project datasheets and source code for reference. The following steps will help you set up the fixed stack bootloader in the PWMExample project.

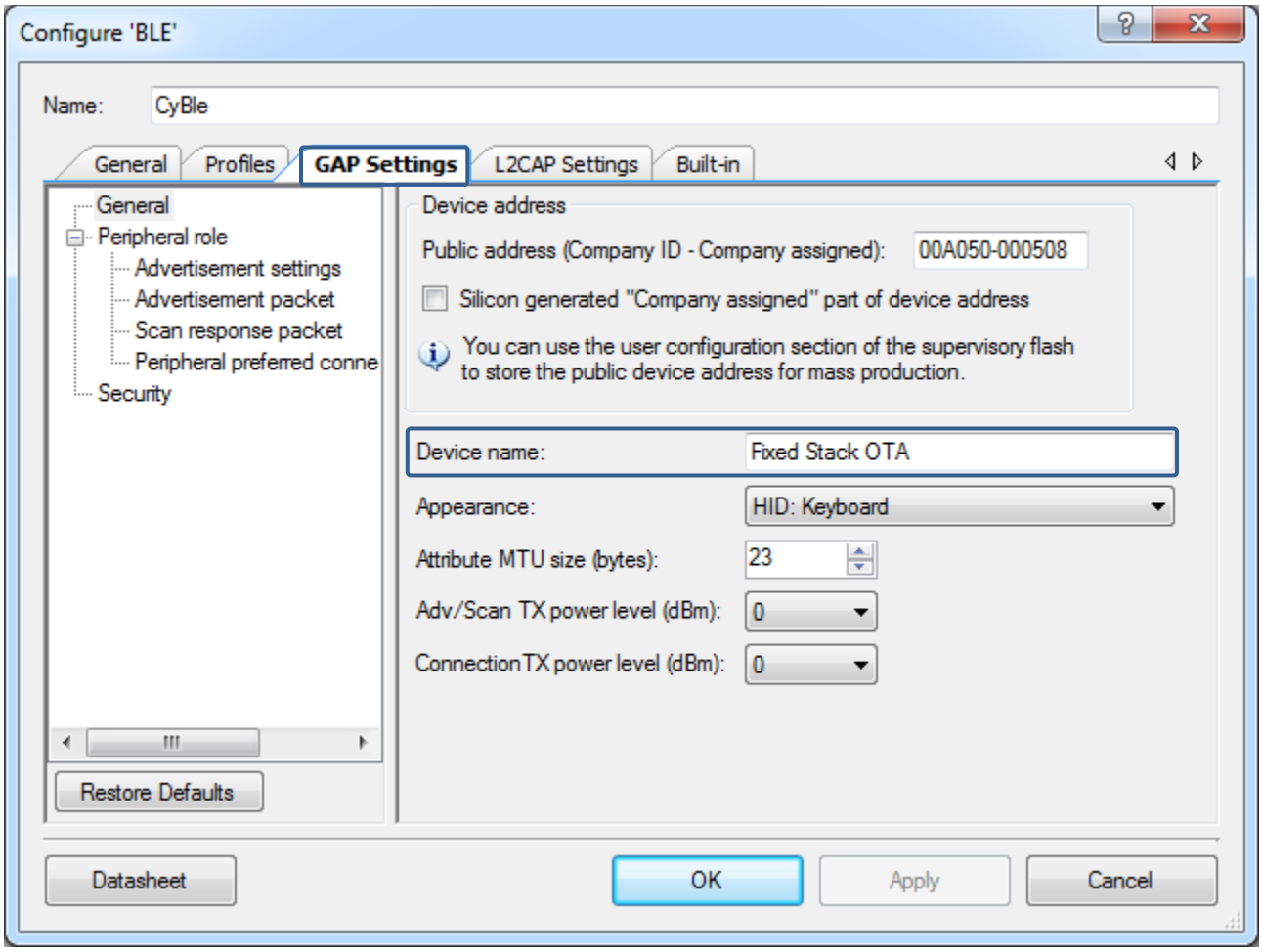
Note: The code sharing feature of the Fixed Stack OTA Bootloader is not supported when the MDK MicroLIB linker option is selected. So **Use MicroLIB** must be set to **False** in **Project** > **Build Settings** (see Figure 13).

Figure 13. Build Settings for MDK Linker



1. Open the PWMExample01 project created in PSoC Creator (see Creating a Basic Example Target Project).
2. Add the BLE_OTA_FixedStack_Bootloader example to the PWMExample01 workspace (see Adding an Example Project to an Existing Workspace).
3. Set the BLE_OTA_FixedStack_Bootloader01 project as the active project by right-clicking the project in **Workspace Explorer** and selecting **Set As Active Project**.
4. Follow the instructions in the Selecting Another Device section to change/select the correct device for the target application.
5. Change the **Device Name** to “Fixed Stack OTA” in the **GAP Settings** tab in the BLE Component configuration window (see Figure 14). To open the BLE Component configuration dialog, double-click the BLE Component.

Figure 14. BLE Component Configuration Showing GAP Settings



If the bootloadable project of your choice already contains a BLE Component, then:

a. Replace the BLE Component present in the BLE_OTA_FixedStack_Bootloader01 project with the BLE Component from your bootloadable project.
b. Add and configure the bootloader service in the existing BLE Component (see Adding Bootloader Service).
c. Remove the BLE Component from the bootloadable project schematic.

By replacing the BLE component, some of the services/profiles related code might have to be removed or added, to prevent compile time errors, depending on the new BLE component configuration.

6. BLE OTA Fixed Stack example projects use custom linker scripts and must be configured for the selected device. Follow the Configuring Fixed Stack OTA Projects for Other Cypress BLE Devices section to do so.
7. Build the BLE_OTA_FixedStack_Bootloader01 project. The project should build without any errors.
8. Create a new folder named “LinkerScripts” in the *PWMExample01.cydsn* project folder.
9. In PSoC Creator, double-click the *mk.bat* file (**Workspace Explorer** > **BLE_OTA_FixedStack_Bootloader01** > **Scripts** > **mk.bat**) to open it.
10. Edit and save the file per Table 3. `LOADABLE_PRJ_NAME` must be assigned with the bootloadable application name (in this case, it is “PWMExample01”).

Table 3. Changes in *mk.bat*

Line Number	Variable	Value
28	LOADER_PRJ_NAME	BLE_OTA_FixedStack_Bootloader01
30	LOADABLE_PRJ_NAME	PWMExample01

11. Run the *mk.bat* file from **Windows Explorer**. This file is located at *…\PWMExample01\BLE_OTA_FixedStack_Bootloader01.cydsn\Scripts*. The batch file should run without errors. After the batch file has finished running, press any key to dismiss the window. This step creates a *BootloaderSymbolsGcc.ld* file under *…\PWMExample01\PWMExample01.cydsn\LinkerScripts*.
12. Open another instance of PSoC Creator and create a new workspace for the BLE_OTA_FixedStack_Bootloadable example project (see Creating an Example Project Workspace). You will be using the necessary files and code snippets from this project to enable the fixed stack bootloader.
13. Set the PWMExample01 project as the active project.
14. Create a new folder named “LinkerScripts” in the **Workspace Explorer** of PSoC Creator for the PWMExample01 project. To create a new folder, right-click on the project name in **Workspace Explorer** and choose **Add** > **New Folder**.
15. The bootloadable example project created in step 12 has a LinkerScripts folder. This folder contains linker scripts (*cm0gcc.ld*, *Cm0Iar.icf*, and *Cm0Mdk.scat*) for all three compilers supported by PSoC Creator. Copy all three files to the LinkerScripts folder created in the PWMExample01 project in step 8.
16. Add the following files to the LinkerScripts folder (created in step 14) in the **Workspace Explorer** of the PSoC Creator PWMExample01 project. In the file system, these files are located in the LinkerScripts folder created in step 8.
    - *BootloaderSymbolsGcc.ld*
    - *Cm0Iar.icf*
    - *cm0gcc.ld*
    - *Cm0Mdk.scat*
17. BLE OTA fixed stack example projects use custom linker scripts and must be configured for the selected device. Follow the Configuring Fixed Stack OTA Projects for Other Cypress BLE Devices section to do so.
18. Copy the following Components from *TopDesign.cysch* of the BLE_OTA_FixedStack_Bootloadable example project (created in step 12) to the *TopDesign.cysch* of PWMExample01 project created in the Creating a Basic Example Target Project section. The configuration for Components added in this step is covered in the BLE Fixed Stack Bootloadable example project datasheet. Components may be spread across multiple schematic pages.
    - Bootloadable
    - WDT
    - UART_DEB
    - WDT_Interrupt

    Make sure that the final bootloadable project schematic does not contain the BLE Component. If the bootloadable project already contains a BLE Component, then:
    a. Move it to BLE_OTA_FixedStack_Bootloader01 project.
    b. Add and configure the bootloader service into the existing BLE Component (see Adding Bootloader Service).

    After this step, repeat steps from step 5 to update the bootloader and generate a new linker script.

    The Bootloadable Component is required to implement the fixed stack OTA. All other Components are not critical and can be omitted in this step. However, they are required to prevent compile time errors. The UART Component is used to print debug messages. WDT and WDT_Interrupt implement timing in the project wherever required.

At the end of this step, your schematic should look similar to Figure 15. Specify paths to the bootloader project HEX and ELF files.

a. Double-click on the Bootloadable Component.

b. Navigate to the **Dependencies** tab and link the **Bootloader HEX file** to the *BLE_OTA_FixedStack_Bootloader01.hex* file (located at …*\BLE_OTA_FixedStack_Bootloader01.cydsn \CortexM0\\<compiler version>\<build configuration>*), as shown in Figure 16.

c. Click **OK** to close the Bootloadable Component configuration dialog.

After you have selected the HEX file, the corresponding ELF file will be automatically selected for you.

19. To assign the correct pins for Components added in step 17, open the *PWMExample01.cydwr* and go to the **Pins** tab. Configure pins as shown in Table 4.

Figure 15. TopDesign.cysch View After Adding Necessary Components

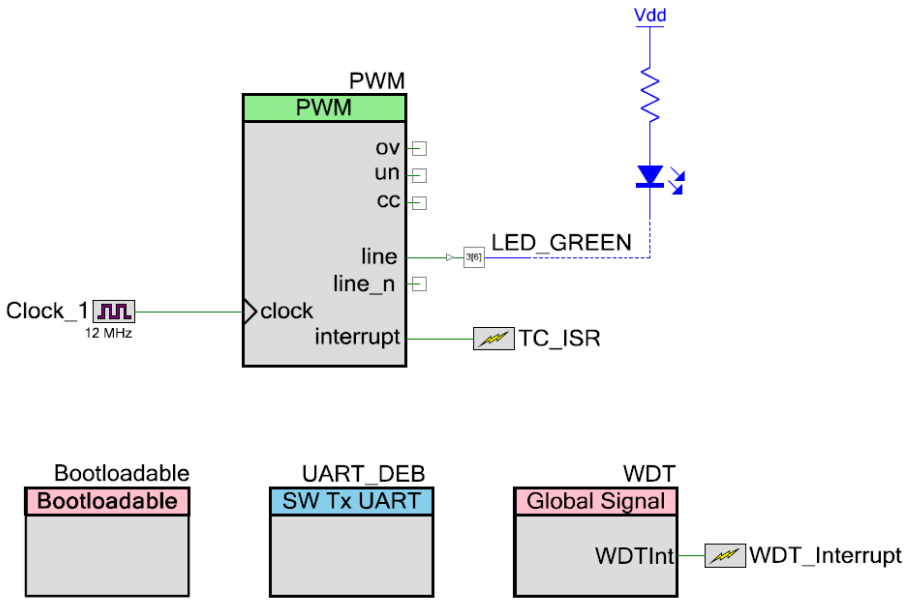


Figure 16. Bootloadable Component Configuration

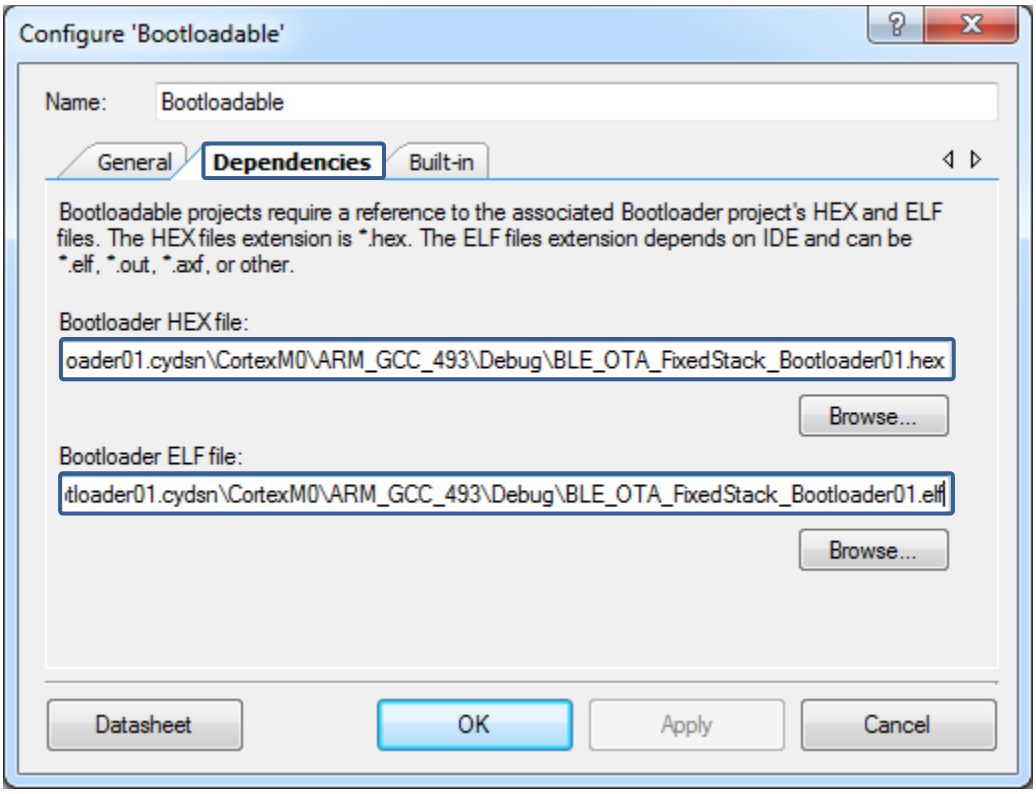


Table 4. Pin Mapping for PWMExample01

Pin Name	Port Assignment
UART_DEB:tx	P1[5]
LED_GREEN	P3[6]

20. Copy the following files from the BLE Fixed Stack Bootloadable example project workspace directory to the PWMExample01 project workspace directory and add them to the PWMExample01 project. These files implement a part of the OTA and debug functionality.

    a. To add header files, right-click the **Header Files** folder in **Workspace Explorer** and choose **Add** > **Existing Item…**. Browse and select the necessary files and click **Open**.

    b. To add source files, right-click the **Source Files** folder in **Workspace Explorer** and choose **Add** > **Existing Item…**. Browse and select the necessary files and click **Open**.

    - *common.h*
    - *main.h*
    - *OTAMandatory.h*
    - *OTAOptional.h*
    - *debug.h*
    - *Options.h*
    - *OTAMandatory.c*
    - *OTAOptional.c*
    - *debug.c*

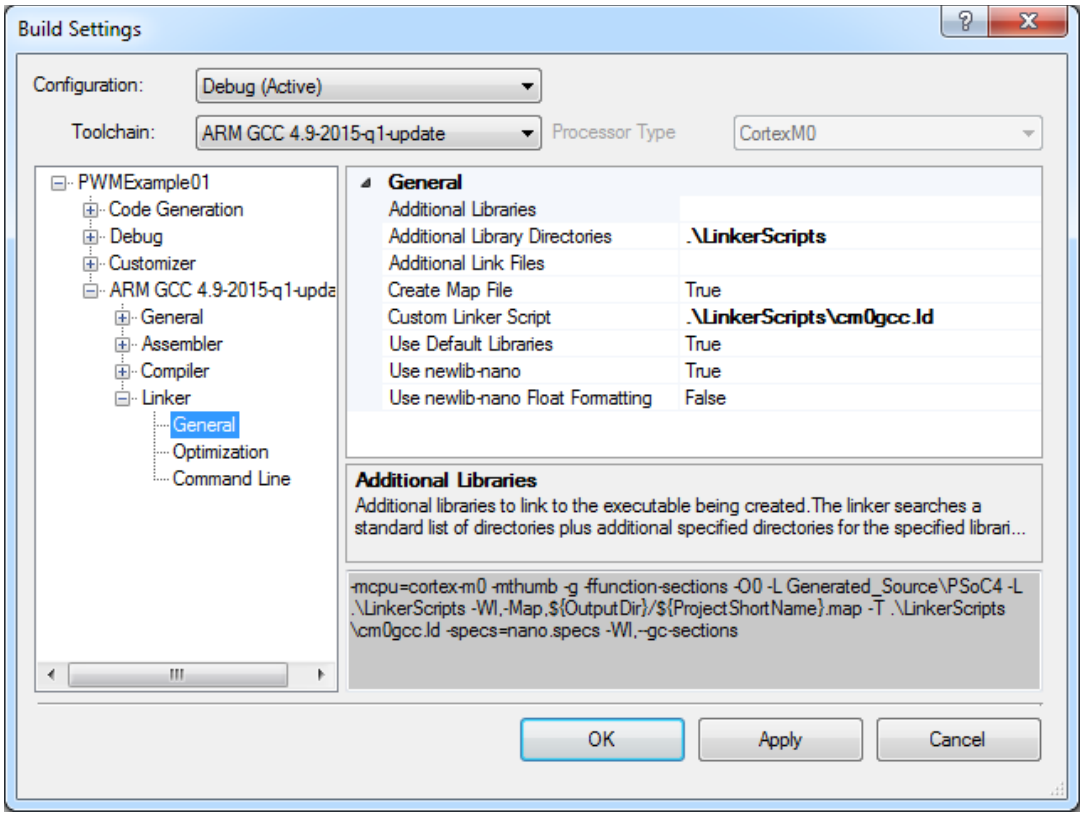
    The *OTAMandatory.c/h* files implement all the required functionality for enabling external memory OTA. All other files are not mandatory and are copied over to prevent compile time errors. The *debug.h/c* files implement UART-based debug message printing. The *common.h* file contains defines for LED states and WDT options. The *Options.h* file contains defines to enable/disable debugging. The *OTAOptional.c/h* files implement helper functions for WDT, LEDs, and debug message printing.

21. Additional code (borrowed from the BLE_OTA_FixedStack_Bootloadable project) must be added to *main.c* of the PWMExample01 project to enable the bootload or OTA functionality. The changes are numerous and cannot be individually listed; instead, you can download and use the modified file. Replace the PWMExample01 project *main.c* file with *main.c* from *…\Code\Fixed Stack OTA*.
22. Make sure the settings listed in Table 5 are applied to the build settings for PWMExample01. These changes tell the linker to use the new custom linker script. To change the build settings, choose **Project** > **Build Settings…** and then **PWMExample01** > **ARM GCC 4.9-2015-q1-update** > **Linker** > **General** on the tree view, as shown in Figure 17.

Table 5. Build Setting Changes

Field	Value
Additional Library Directories	.\LinkerScripts
Custom Linker Script	.\LinkerScripts\cm0gcc.ld

Figure 17. Build Settings Dialog Showing Linker Settings



23. After adding a Bootloader/Bootloadable Component, debug support is disabled in PSoC Creator. However, once the firmware begins execution, you can attach to the target device (**Debug** > **Attach to Running Target…**) or use UART messaging to debug the firmware. To enable or disable UART debugging, see the Debugging section. Make sure that an adequate heap (0x400 bytes) and stack (0x800 bytes) size has been set for the project to work correctly when UART is enabled. Failure to do so will result in unpredictable behavior of the firmware; see the Debugging section for more details.
24. Build the PWMExample01 project. The project should build without any errors.
25. Program the PSoC BLE Pioneer Kit by choosing **Debug** > **Program**.

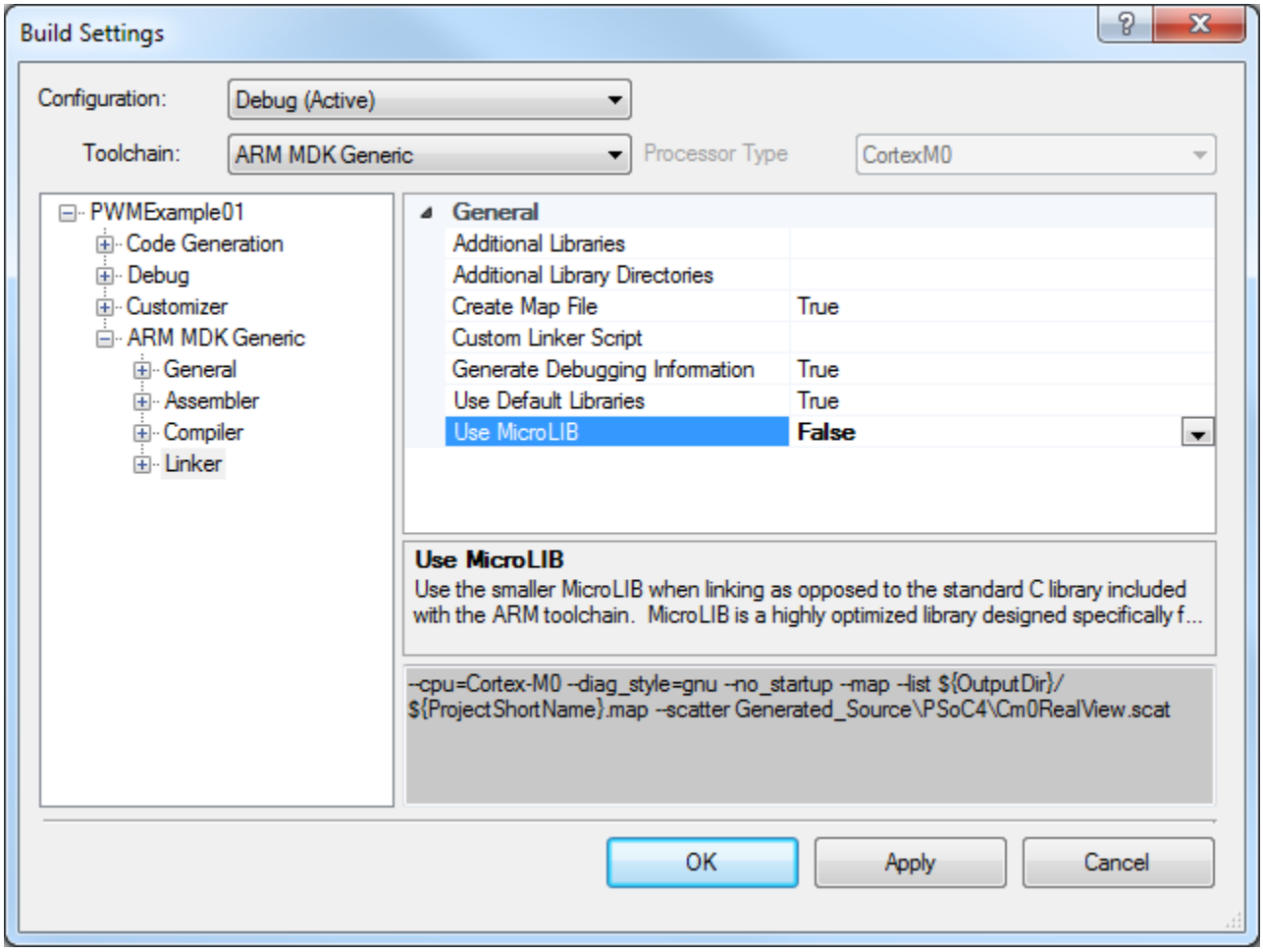
After programming is complete, the green LED will cycle through high to low brightness levels. At this point, you can perform a device firmware upgrade by following one of the methods described in the Performing an OTA Upgrade section. In addition, you can test the OTA feature by following the steps listed in the Testing the OTA Feature section.

5.4 Adding an Upgradable Stack OTA Bootloader

This section explains how to add an Upgradable Stack OTA Bootloader to the PWMExample project that you prepared in the Creating a Basic Example Target Project section. You should also review the BLE OTA Upgradable Stack Launcher, Stack and Keyboard example project datasheets, and the source code for reference. The following steps will help you set up the Upgradable Stack Memory Bootloader in the PWMExample project.

Note: The code sharing feature of the Upgradable Stack OTA Bootloader is not supported when the MDK MicroLIB linker option is selected. So **Use MicroLIB** must be set to **False** in **Project** > **Build Settings** (see Figure 18).

Figure 18. Build Settings for MDK Linker

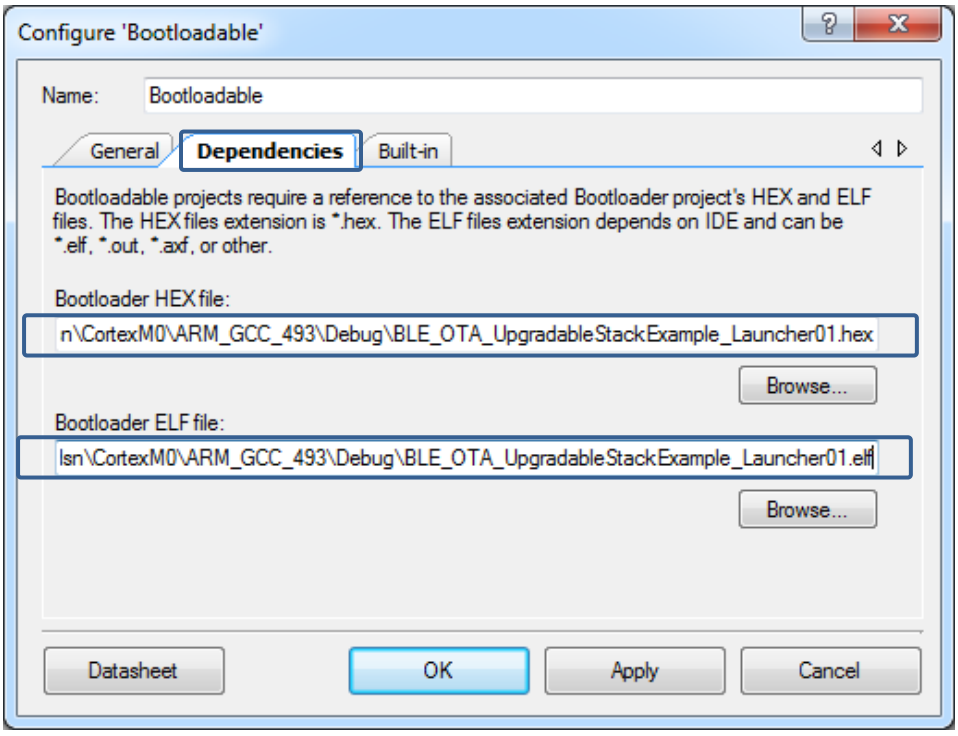


1. Open the PWMExample01 project created in PSoC Creator (see Creating a Basic Example Target Project).
2. Add the BLE_OTA_UpgradableStackExample_Launcher example to the PWMExample01 workspace (see Adding an Example Project to an Existing Workspace).
3. Set the BLE_OTA_UpgradableStackExample_Launcher01 project as the active project. To do so, right-click the project in **Workspace Explorer** and select **Set As Active Project**.
4. Follow the instructions in the Selecting Another Device section to change/select the correct device for the target application.
5. Build the BLE_OTA_UpgradableStackExample_Launcher01 project. The project should build without any errors.
6. Add the BLE_OTA_UpgradableStackExample_Stack example to the PWMExample01 workspace (see Adding an Example Project to an Existing Workspace).
7. Set the BLE_OTA_UpgradableStackExample_Stack01 project as the active project.

8. Follow the instructions in the Selecting Another Device section to change/select the correct device for the target application.
9. Specify the paths to the launcher project HEX and ELF files.
   a. Double-click on the Bootloadable Component.
   b. Navigate to the **Dependencies** tab and link **Bootloader HEX file** to the *BLE_OTA_UpgradableStackExample_Launcher01.hex* file (located at …\ *BLE_OTA_UpgradableStackExample_Launcher01.cydsn \CortexM0\\<compiler version>\<build configuration>*\), as shown in Figure 19.
   c. Click **OK** to close the Bootloadable Component configuration dialog.

   After you have selected the HEX file, the corresponding ELF file will be automatically selected for you.

Figure 19. Bootloadable Component Configuration

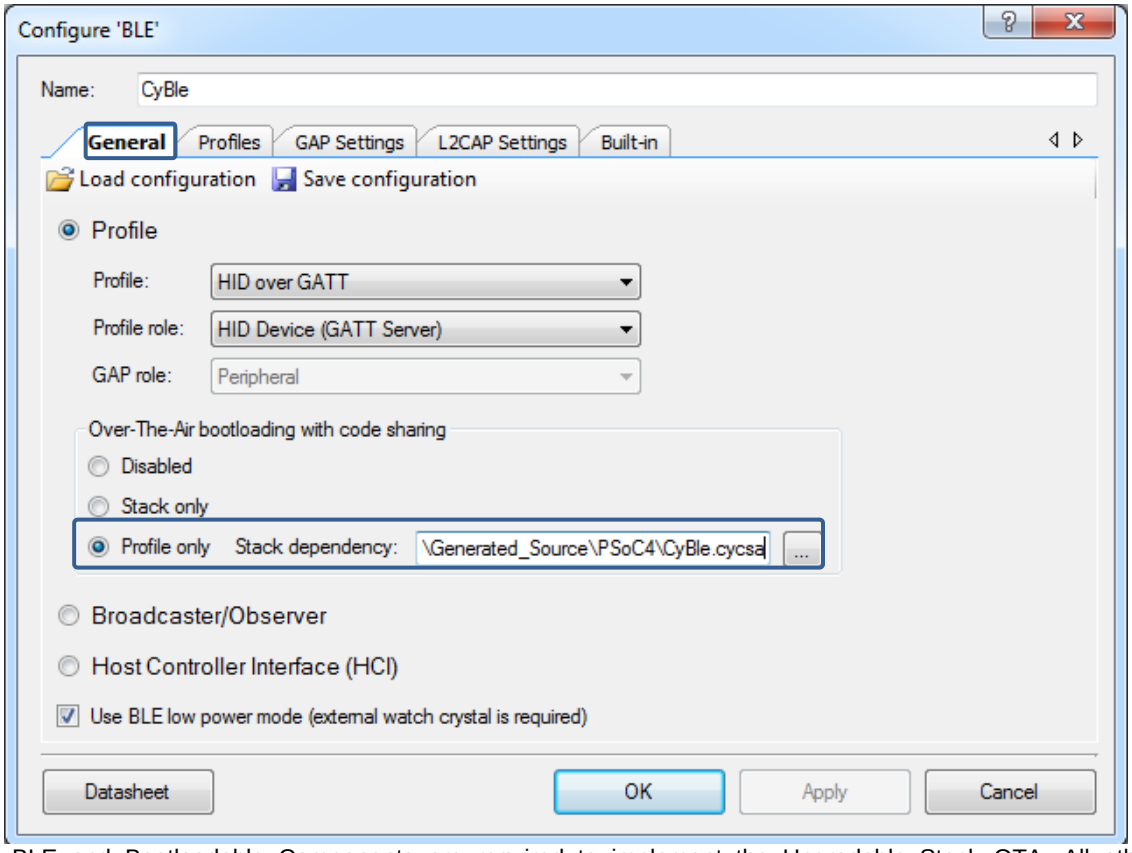


10. Build the BLE_OTA_UpgradableStackExample_Stack01 project. The project should build without any errors.
11. Open another instance of PSoC Creator and create a new workspace for the BLE_OTA_UpgradableStack_HID_Keyboard example project (see Creating an Example Project Workspace). You will be using the necessary files and code snippets from this project to enable the Upgradable Stack OTA Bootloader.
12. Set the PWMExample01 project as the active project.
13. Copy the following components from the *TopDesign.cysch* file from the BLE_OTA_UpgradableStack_HID_Keyboard example project (created in step 23) to the *TopDesign.cysch* file of the PWMExample01 project (created in the section Creating a Basic Example Target Project). The configuration for the Components added in this step is described in the BLE_OTA_UpgradableStack_HID_Keyboard example project datasheet.

    - CyBle
    - Bootloadable
    - UART
    - SW2
    - Wakeup_Interrupt

If the project of your choice already contains a BLE Component, ensure that the BLE Component is updated to the latest version and is configured in the **Profile only** mode (refer to Figure 20).

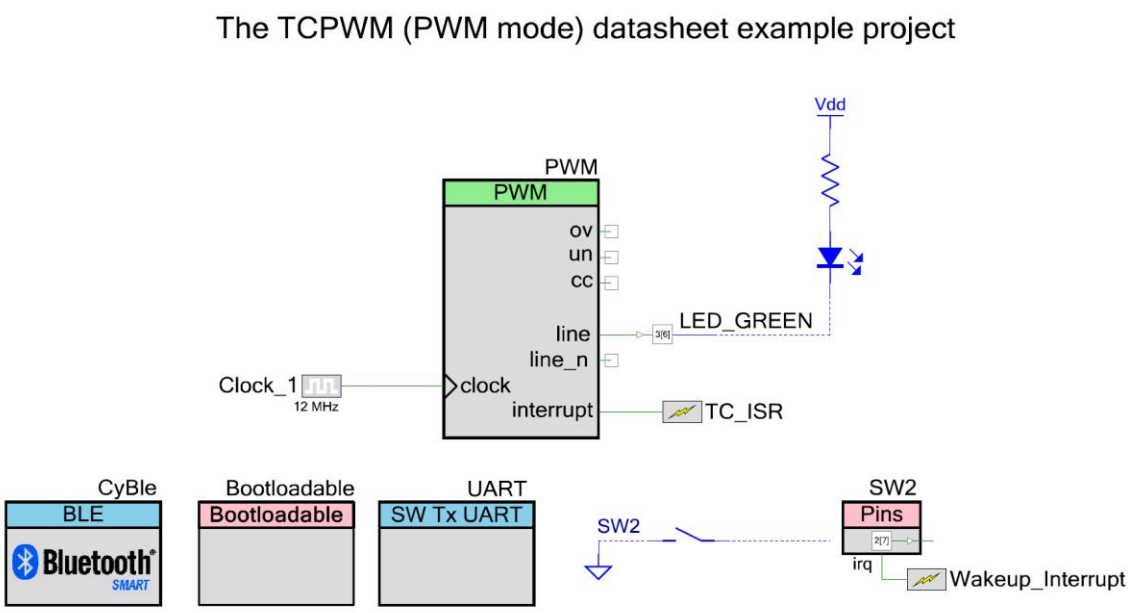
Figure 20. BLE Component Configuration Showing Profile Only Option Selection



The BLE and Bootloadable Components are required to implement the Upgradable Stack OTA. All other Components are not critical for implementing the Upgradable Stack OTA and can be omitted in this step. However, they are required to prevent compile time errors. The UART Component is used to print debug messages. SW2 and Wakeup_Interrupt are used as a trigger to enter bootloader mode. If your project has some other mechanism to enter bootloader mode, avoid copying the SW2 and Wakeup_Interrupt Components.

At the end of this step, your schematic should look similar to Figure 21.

Figure 21. TopDesign.cysch View after Adding Necessary Components

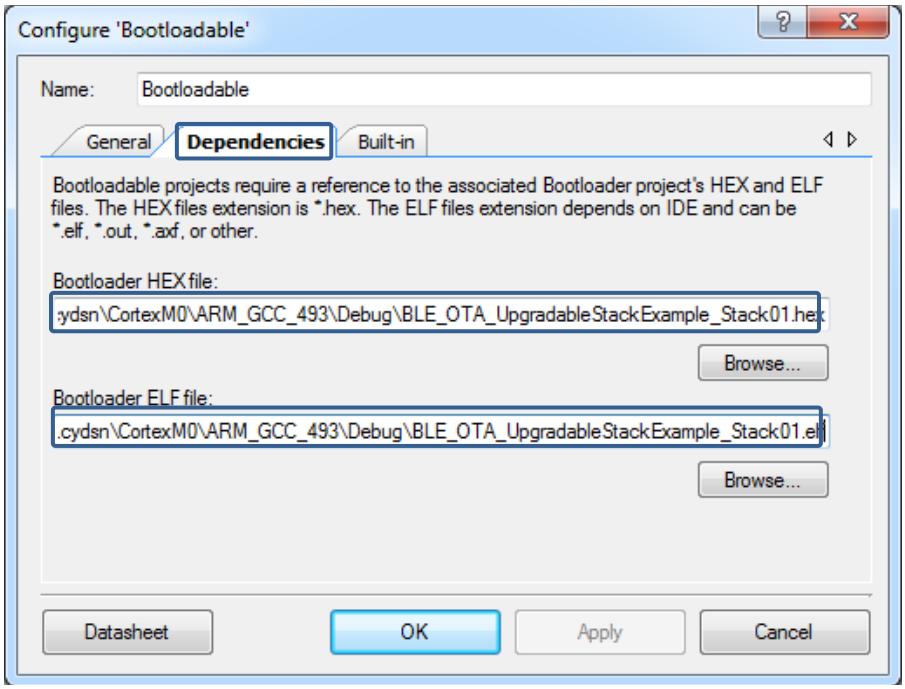


14. Specify the path to the *CyBle.cycsa* file of BLE_OTA_UpgradableStackExample_Stack01 in the BLE Component.
    a. Double-click on the BLE Component.
    b. Navigate to the **General** tab; in the **Over-The-Air bootloading with code sharing** section, select the **Profile only** option.
    c. Select the *CyBle.cycsa* file in the **Stack dependency** field, as shown in Figure 20
    d. .

    The *CyBle.cycsa* file is located in the …*Generated_Source**PSoC4*\ directory under the BLE_OTA_UpgradableStackExample_Stack01 project directory.
15. Specify paths to the launcher project HEX and ELF files.
    a. Double-click on the Bootloadable Component.
    b. Navigate to the **Dependencies** tab and link **Bootloader HEX file** to the *BLE_OTA_UpgradableStackExample_Stack01.hex* file (located at …\ *BLE_OTA_UpgradableStackExample_Stack01.cydsn* *CortexM0*\\<*compiler version*>\<*build configuration*>\), as shown in Figure 22.
    c. Click **OK** to close the Bootloadable Component configuration dialog.

    After you have selected the HEX file, the corresponding ELF file will be automatically selected for you.

Figure 22. Bootloadable Component Configuration



16. To assign the correct pins for the Components added in step 17, open *PWMExample01.cydwr* and go to the **Pins** tab. Configure the pins as described in Table 6.

Table 6. Pin Mapping for PWMExample01

Pin Name	Port Assignment
UART:tx	P1[5]
LED_GREEN	P3[6]
SW2	P2[7]

17. Copy the following files from the BLE_OTA_UpgradableStack_HID_Keyboard example project workspace directory to the PWMExample01 project workspace directory and add them to the PWMExample01 project. These files implement a part of the OTA and debug functionality.

    a. To add header files, right-click the **Header Files** folder in **Workspace Explorer** and choose **Add** > **Existing Item…**. Browse and select the necessary file and click **Open**.

    b. To add source files, right-click the **Source Files** folder in **Workspace Explorer** and choose **Add** > **Existing Item…**. Browse and select the necessary file and click **Open**.

    - *OTAMandatory.h*
    - *debug.h*
    - *common.h*
    - *options.h*
    - *OTAMandatory.c*
    - *debug.c*

    The *OTAMandatory.c/*h files implement all the required functionality for enabling Upgradable Stack OTA. All other files are not mandatory and are copied over to prevent compile time errors. The *debug.h/c* files implement UART-based debug message printing. The *common.h* file contains defines for LED states and WDT options. The *options.h* file contains defines to enable/disable debugging.

18. Additional code (borrowed from the BLE_OTA_UpgradableStack_HID_Keyboard project) must be added to *main.c* of the PWMExample01 project to enable bootload or OTA functionality. The modified file can be downloaded. Replace the PWMExample01 project *main.c* file with *main.c* from *…\Code\Upgradable Stack OTA*. Following is a walkthrough of the relevant (for OTA) portions of the code.

The `AfterImageUpdate()` function checks if the application image has been updated and is running for the first time. If it is running for the first time and the Bonding requirement option is set to Bonding in the BLE Component, it verifies the bonding data and erases it if it is not valid. It also sets up the update detection flag in the unused metadata area.

The `InitializeBootloaderSRAM()` function is used to initialize the BLE Stack SRAM, which is required for code sharing. This function has to be called at the very beginning of the main function. Failure to do so can result in unexpected behavior of the firmware.

To switch from the application image to the Stack Application, first set Stack as the active application by calling `Bootloadable_SetActiveApplication()` with input parameter `0`. Then call the `Bootloadable_Load()` function followed by a software reset using `CySoftwareReset()`. This entire sequence is shown in Code 1. The example uses SW2 being pressed and released as a trigger to initiate the bootloading process.

Code 1. Application-Level Bonding Information Write

```
/* For GCC compiler use separate API to initialize BLE Stack SRAM.
 * This is needed for code sharing.
 */
#if !defined(__ARMCC_VERSION)
    InitializeBootloaderSRAM();
#endif


/* Checks if Self Project Image is updated and Runs for the First time */
AfterImageUpdate();

/* Start CYBLE component and register generic event handler */
CyBle_Start(AppCallBack);

while (1)
{
    /* If key press event was detected - debounce it and switch to
    bootloader emulator mode */
    if (SW2_Read() == 0u)
    {
        CyDelay(500u);
        if (SW2_Read() == 0u)
        {
            CyDelay(500u);
            while (SW2_Read() == 0u)
            {
                /* Wait for button to be released */
            }

            //Switch to the Stack project, which enables OTA service
            Bootloadable_SetActiveApplication(0);
            Bootloadable_Load();
            CySoftwareReset();
        }
    }

}
```

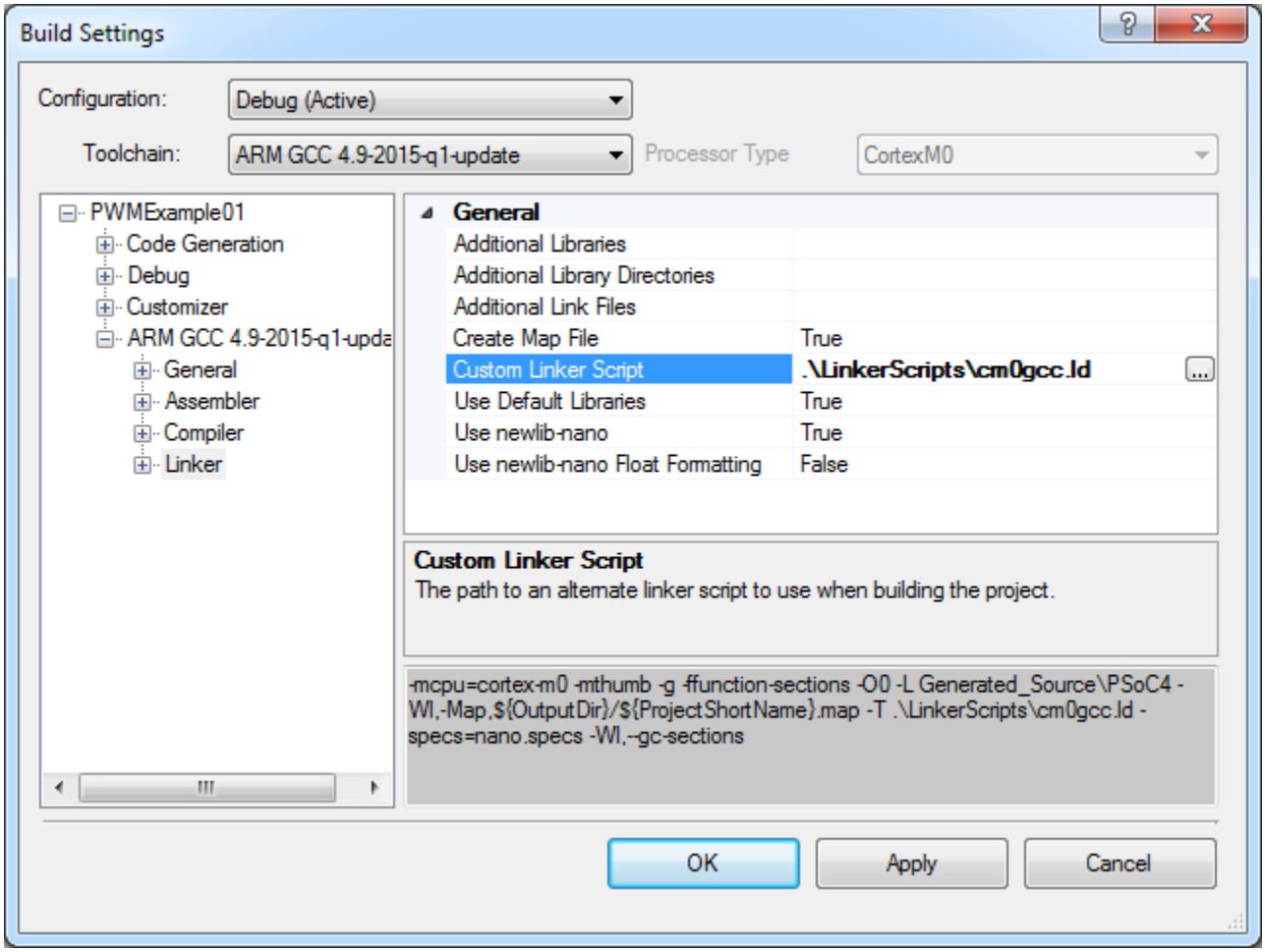
19. Create a new folder named “LinkerScripts” in the *PWMExample01.cydsn* project.

20. The bootloadable example project created in step 11 has a LinkerScripts folder. This folder contains linker scripts (*cm0gcc.ld* and *Cm0Mdk.scat*) for all three compilers supported by PSoC Creator. Copy all three files to the LinkerScripts folder created in the PWMExample01 project in step 19.
21. Create a new folder named “LinkerScripts” in the **Workspace Explorer** of PSoC Creator for the PWMExample01 project.
22. Add the following files to the LinkerScripts folder (created in step 21), in the **Workspace Explorer** of the PSoC Creator PWMExample01 project. In the file system, these files are located in the LinkerScripts folder created in step 19.
    - *cm0gcc.ld*
    - *Cm0Mdk.scat*
23. Make sure that the settings listed in Table 7 are applied to the build settings for PWMExample01. These changes tell the linker to use the new custom linker script. To change the build settings, choose **Project** > **Build Settings…** and then select **PWMExample01** > **ARM GCC 4.9-2015-q1-update** > **Linker** > **General** in the tree view, as shown in Figure 23.

Table 7. Build Setting Changes

Field	Value
Custom Linker Script	.\LinkerScripts\cm0gcc.ld

Figure 23. Build Settings Dialog Showing Linker Settings



24. After adding a Bootloader/Bootloadable Component, debug support is disabled in PSoC Creator. However, once the firmware begins execution, you can attach to the target device (**Debug** > **Attach to Running Target…**) or use UART messaging to debug the firmware. To enable or disable UART debugging, see the Debugging section. Make sure that an adequate heap (0x400 bytes) and stack (0x800 bytes) size has been set for the project to work correctly when UART is enabled. Failure to do so will result in unpredictable behavior of the firmware; see the Debugging section for more details.
25. Build the PWMExample01 project. The project should build without any errors.
26. Program the PSoC BLE Pioneer Kit by choosing the **Debug** > **Program** menu item.

After programming is complete, the green LED will cycle through high to low brightness levels. At this point, you can perform a device firmware upgrade by following one of the methods described in the Performing an OTA Upgrade section. In addition, you can test the OTA feature by following the steps listed in the Testing the OTA Feature section.

Two bootloadable images are available: *BLE_OTA_UpgradableStackExample_Stack01.cyacd* (stack image) and *PWMExample01.cyacd* (application image). To upgrade the application, use the application image file and perform the upgrade. To upgrade the stack, use the stack image file and perform the upgrade. Re-establish connection with the target device and then perform the application upgrade.

6 Performing an OTA Upgrade

Cypress provides three types of host applications through which the firmware on a target device can be upgraded. All three hosts can be interchangeably used to perform an OTA upgrade regardless of the OTA method implemented on the target device.

To perform an OTA upgrade, the target platform must be preprogrammed with the HEX file generated at the end of the processes detailed in the Adding an External Memory OTA Bootloader, Adding a Fixed Stack OTA Bootloader, or Adding an Upgradable Stack OTA Bootloader section. The build process also generates a *.cyacd* file (along with the HEX file, in the bootloadable project output directory), which is the bootloadable/application image.

The *.cyacd* file obtained from a project using a particular kind of OTA bootloader cannot be used to upgrade a device programmed with another OTA bootloader. For example, the *.cyacd* file from a project with an External Memory OTA Bootloader does not work with a device flashed with a Fixed Stack OTA Bootloader.

A CySmart USB dongle (see Figure 24) is required for the PC-based firmware upgrade methods to work correctly. It comes with the PSoC 4 BLE Pioneer Kit.

Figure 24. CySmart USB Dongle

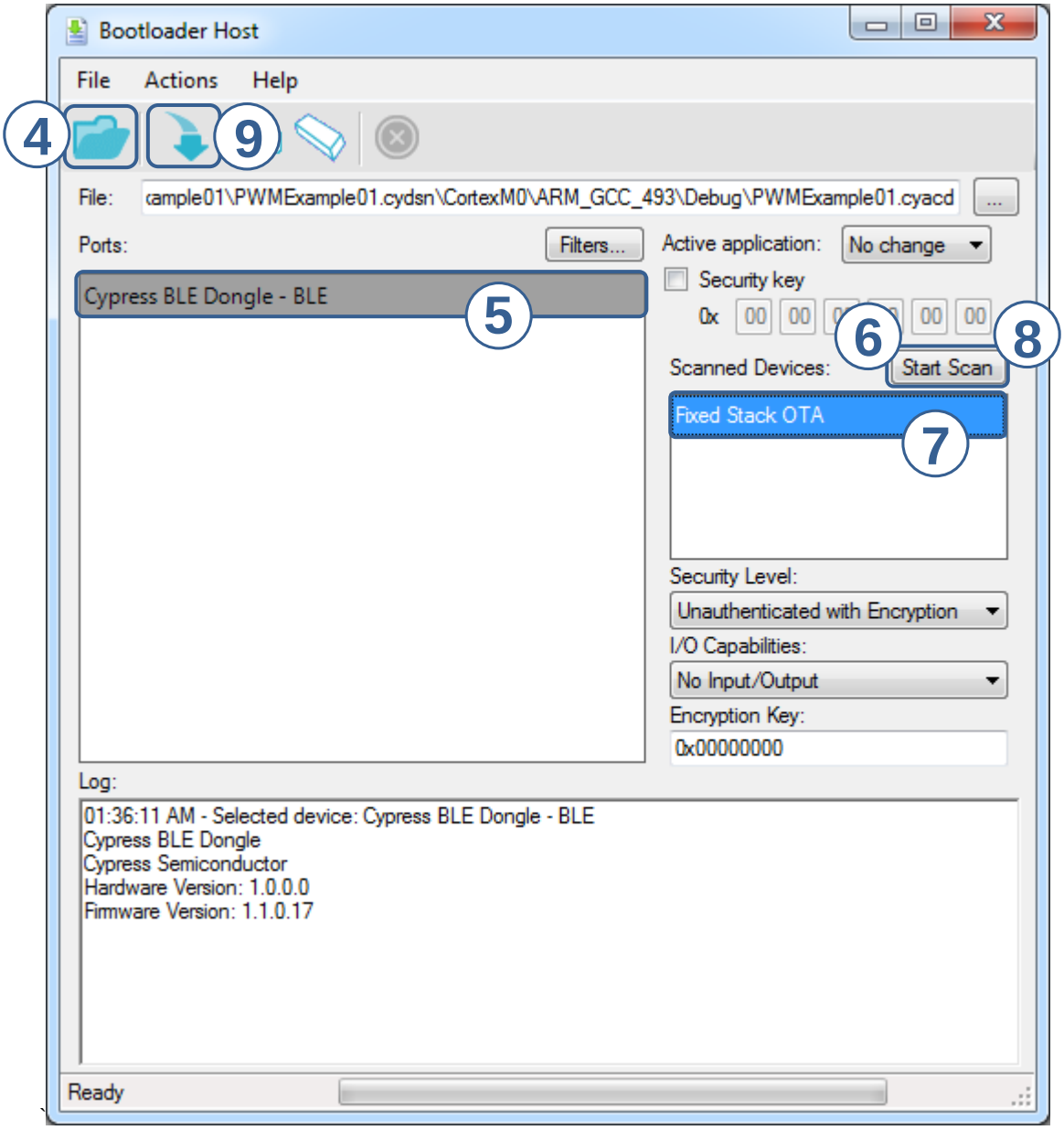
6.1 Upgrading Through Bootloader Host Tool

The Bootloader Host tool (see Figure 25) ships with PSoC Creator and can be used for a variety of device firmware upgrade operations including a BLE OTA upgrade. Follow these steps to use the Bootloader Host tool to perform an OTA-based device firmware upgrade. Make sure that the CySmart USB dongle is plugged into the PC before proceeding.

Note: Bootloader Host tool support for OTA is broken with the release of the CySmart 1.2 dongle firmware.

1. Connect the CySmart USB dongle (shown in Figure 24) to a USB port on a PC.

Figure 25. Bootloader Host Tool



2. Press the **SW2** switch on the PSoC 4 BLE Pioneer Kit to put the device into the bootloader mode indicated by the red LED. The method used to enter the bootloader can be re-implemented to suit specific requirements (for example, the bootload process can be triggered on a specific BLE characteristic write command).

For the Upgradable Stack OTA implementation, the bootload mode times out in 40 seconds if the OTA process has not yet started. For the External Memory OTA and Fixed Stack OTA implementation, there is no timeout; the firmware will wait indefinitely in the bootload mode.

3. Open the Bootloader Host tool by choosing **Tools** > **Bootloader Host** in PSoC Creator.
4. Click **Open** (see Figure 25) and point the path to the **.cyacd* file. It is located in the project folder *([project folder]\CortexM0\[compiler name])*.
5. In the Bootloader Host tool, select **Cypress BLE Dongle** listed under **Ports**. See Figure 25.
6. Click the **Start Scan** button next to the **Scanned Devices** field, as shown in Figure 25.
7. Wait until the expected device (External Memory OTA/Fixed Stack OTA) appears in **Scanned Devices,** and then select it (see Figure 25).
8. Click the **Stop Scan** button (see Figure 25).
9. Click **Program** (see Figure 25) in the Bootloader Host tool and wait until the new application image upload is complete.

After the firmware upgrade is complete, the device will reset automatically and you will see the green LED blink. See the PWMExample Project datasheet for details.

6.2 Upgrading Through CySmart PC Tool

CySmart is a BLE host emulation tool for Windows PCs. With the easy-to-use GUI, you can test and debug your BLE peripheral applications. The CySmart PC tool can be downloaded. See the user guide, which is available at the same location, to get more information about CySmart.

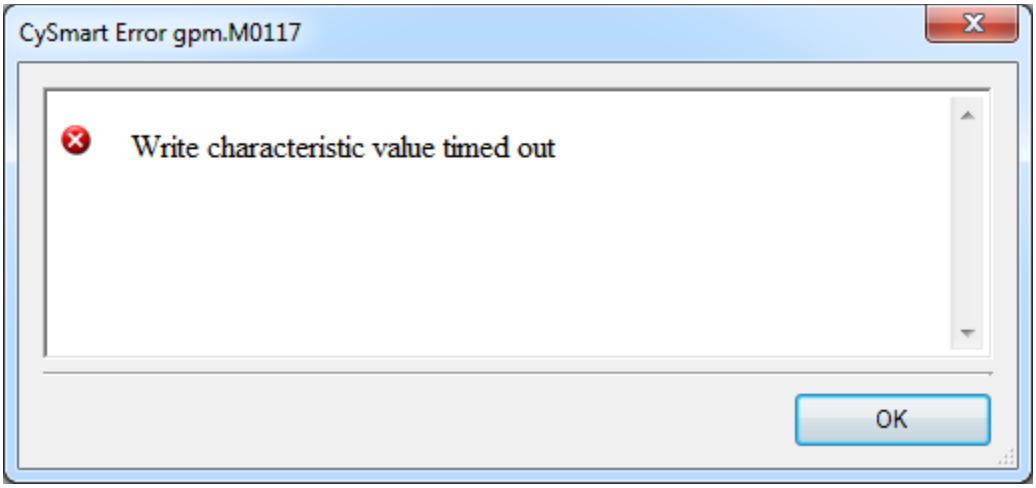
Follow these steps to use the CySmart PC tool to perform an OTA-based device firmware upgrade. Make sure that the CySmart USB dongle is plugged into the PC before proceeding.

1. Connect the CySmart USB dongle (shown in Figure 24) to a USB port on a PC.
2. Press the **SW2** switch on the PSoC 4 BLE Pioneer Kit to put the device in the bootloader mode indicated by the red LED. The method used to enter the bootloader can be re-implemented to suit specific requirements (for example, the bootload process can be triggered on a specific BLE characteristic write command).
3. Open the CySmart tool and follow the instructions in section 2.7, “Updating Peripheral Device Firmware,” in the CySmart User Guide to download the image.

After the firmware upgrade is complete, the device will reset automatically and you will see the green LED blink. See the PWMExample Project datasheet for details.

While performing an OTA upgrade with the CySmart PC tool, you may encounter a **Write characteristic value timed out** error towards the end of the bootload process (see Figure 26). Note that at this point the bootload process itself may not have failed, and you may dismiss this error.

Figure 26. Write Characteristic Value Timed Out Error Dialog at End of Bootload Process



6.3 Upgrading Through CySmart Mobile Apps

CySmart mobile apps are BLE or Bluetooth Smart utilities developed by Cypress Semiconductor. CySmart can be used to connect to various BLE products and can be used with BLE development kits from Cypress, including CY8CKIT-042-BLE PSoC 4 BLE Pioneer Kit, CY5672 PRoC BLE Remote Control RDK, and CY5682 PRoC BLE Touch Mouse RDK. CySmart mobile apps are available for iOS® and Android™. They can be downloaded from www.cypress.com/cysmartmobile. See the user guide, which is available at the same location, for more information.

6.3.1 Upgrading from iOS

Follow these steps to use the CySmart iOS app to perform an OTA-based device firmware upgrade. Make sure that the CySmart iOS app is installed on your mobile/tablet device before proceeding. In addition, make sure that the new application image (*PWMExample01.cyacd*) is present on the mobile device.

1. Press the **SW2** switch on the PSoC 4 BLE Pioneer Kit to put the device in the bootloader mode indicated by the red LED.
2. Launch the CySmart app on the iOS device and follow the instructions in section 2.1.2.3, “Cypress Bootloader Service,” of the CySmart iOS App User Guide to download the image.

6.3.2 Upgrading from Android

Follow these steps to use the CySmart Android app to perform an OTA-based device firmware upgrade. Make sure that the CySmart Android app is installed on your mobile/tablet device before proceeding. In addition, make sure that the new application image (*PWMExample01.cyacd*) is present on the mobile device.

1. Press the **SW2** switch on the PSoC 4 BLE Pioneer Kit to put the device in the bootloader mode indicated by the red LED.
2. Launch the CySmart app on the Android device and follow the instructions in section 2.1.2.3, “Cypress Bootloader Service,” of the CySmart Android App User Guide to download the image.

After the firmware upgrade is complete, the device will reset automatically and you will see the green LED blink. See the PWMExample Project datasheet for details.

7 Testing the OTA Feature

Follow these steps to test the OTA feature:

1. In the PWMExample project *main.c*, change the `BRIGHTNESS_DECREASE` macro value to 1000. See the PWMExample datasheet for more details.
2. Rebuild the PWMExample project (**Build** > **Build PWMExample01**). This will generate the new *.cyacd* file.
3. Follow one of the OTA upgrade methods described in Performing an OTA Upgrade. Be sure to select the *PWMExample.cyacd* file output from step 2.

After the firmware upgrade is complete, the device will reset automatically and you will see the green LED brightness dimming cycling at a faster rate.

Try changing the `BRIGHTNESS_DECREASE` macro to any value between 0 and 63000 and follow the previous steps to see different results. Smaller values will result in slower brightness dimming cycle rates, and higher values will result in faster brightness dimming cycle rates.

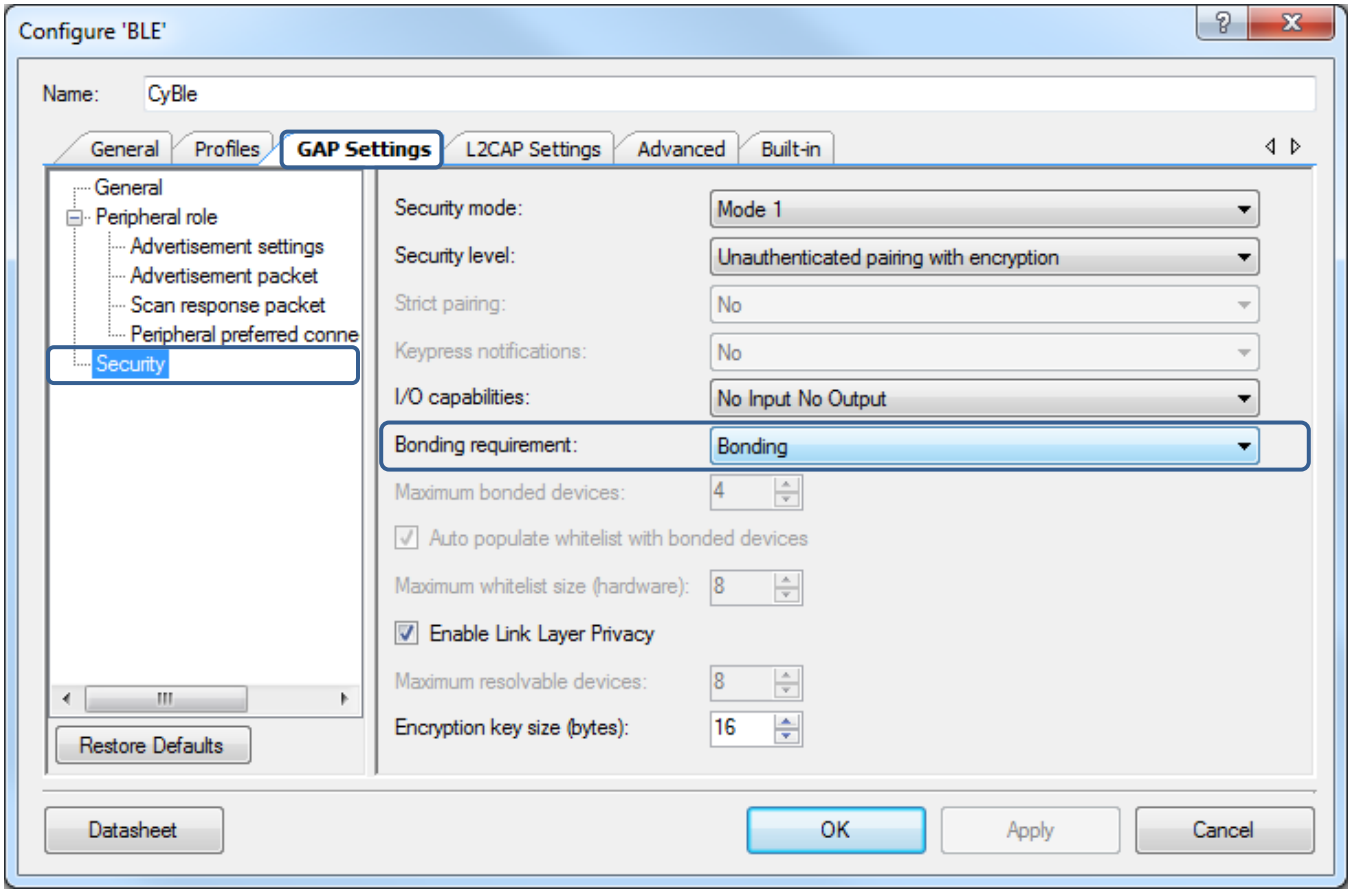
8 Other Considerations

8.1 Bonding/Pairing Information

This section provides information on how to enable bonding in the BLE Component. It also discusses the effects of the three different kinds of OTA upgrades on bonding/pairing information.

To enable bonding, in **Security** under **GAP Settings** in the **Configure 'BLE'** dialog, select the **Bonding** option for the **Bonding requirement** setting (see Figure 27).

Figure 27. Configuring BLE Component for Retaining Bonding Information



The flash write has to be handled by the application at a convenient time, since interrupts cannot be allowed or handled during flash write. The `cyBle_pendingFlashWrite` variable is set by the stack whenever a flash write event is pending. The application can write the bonding information by checking the status of this flag in conjunction with the `CyBle_StoreBondingData` API. See Code 2.

Code 2. Application-Level Bonding Information Write

```
if((cyBle_pendingFlashWrite != 0u))
{
    #if (DEBUG_UART_ENABLED == YES)
        CYBLE_API_RESULT_T apiResult;
        apiResult = CyBle_StoreBondingData(0u);
        DBG_PRINTF("Store bonding data, status: %x \r\n", apiResult);
    #else
         (void)CyBle_StoreBondingData(0u);
   #endif /* (DEBUG_UART_ENABLED == YES) */
}
```

The following sections provide bonding/pairing information specific to each kind of OTA bootloader.

8.1.1 External Memory OTA Bootloader

In the External Memory OTA Bootloader, the bonding information is handled by the application (bootloadable) project. Therefore, after an upgrade, the entire bonding information is lost.

8.1.2 Fixed Stack OTA Bootloader

For the Fixed Stack OTA Bootloader, the bonding information is allocated in the scope of the bootloader project. Therefore, even after the bootloadable project is upgraded, bonding information is intact and will be erased only if the device is reprogrammed using a serial wire debug (SWD) programmer.

8.1.3 Upgradable Stack OTA Bootloader

For the Upgradable Stack OTA Bootloader, the bonding information is stored by both stack (optional) and application image. By default, the bonding information is not stored by the stack. Application upgrade does not have any impact on the application's bonding information as long as the Client Characteristic Configuration Descriptors (CCCDs) remain unchanged. The application's bonding information will be lost when upgrading the stack.

8.2 Debugging

After the addition of an OTA bootloader to a project, the option to debug the project from PSoC Creator will be disabled. However, once the firmware begins execution, you can attach to the target (**Debug** > **Attach to Running Target)**. Using this method, only one project can be debugged at a time. Attaching to a running target can reprogram the device if it is done from the wrong project. Also, the SWD interface has to be enabled to attach the debugger to the target device. The SWD interface for a project can be enabled by changing **Debug Select** to **SWD** in the *.cydwr* file (for example, *PWMExample01.cydwr*) of your project (see Figure 28). The *.cydwr* file can be accessed from the **Workspace Explorer** of the project.

If the SWD interface is not available for debugging, use GPIO toggles or any serial communication interface or protocol such as UART. The OTA example projects available in PSoC Creator utilize a software UART (TX only) Component to achieve this. Because you have borrowed the *debug.c* and *debug.h* files from the PSoC Creator example projects to create your bootloadable projects (see Adding Firmware OTA Bootloader Support to a Target Project), you can use this feature to debug these projects.

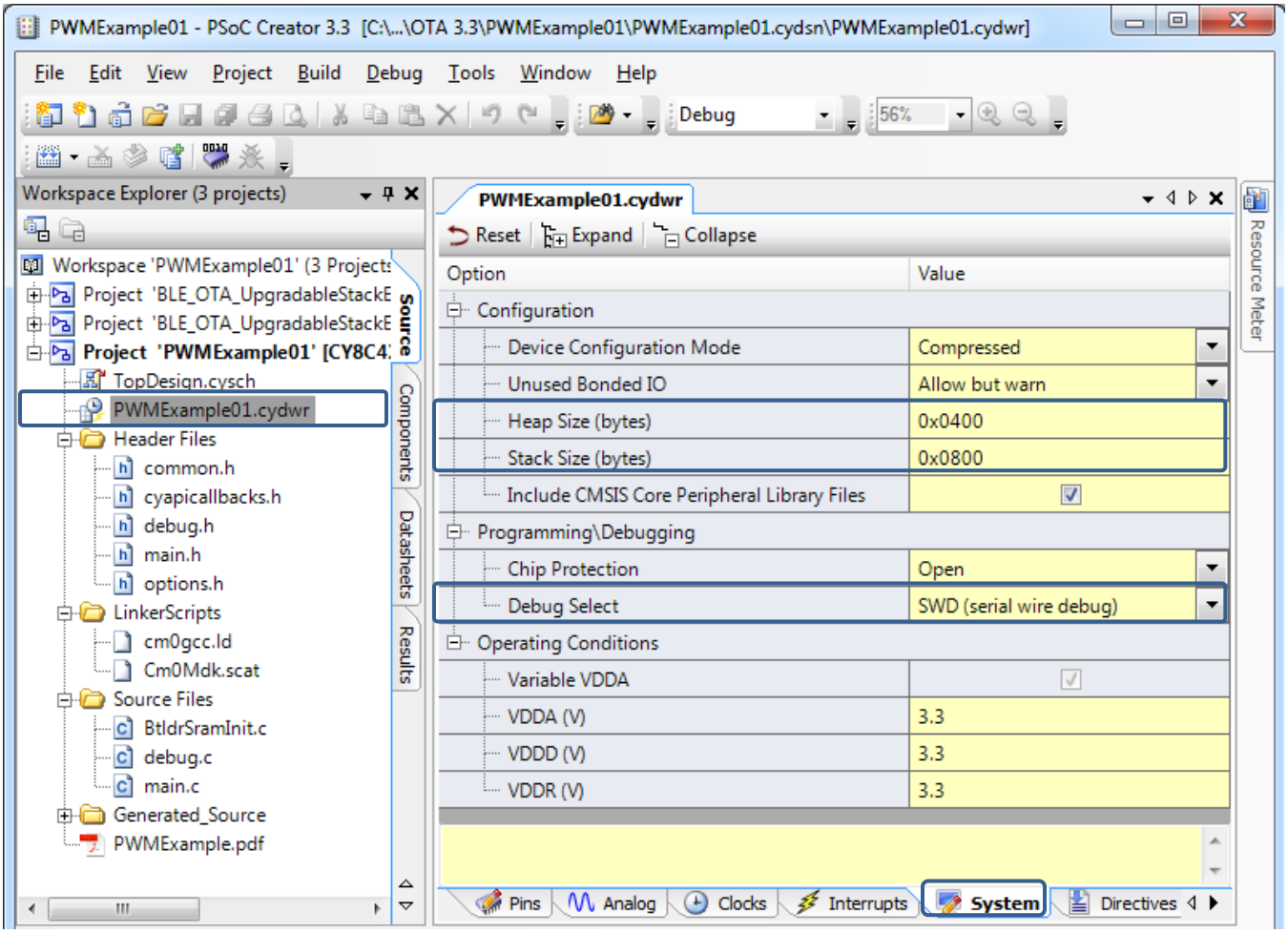
UART debugging can be enabled on any OTA project by setting the `DEBUG_UART_ENABLED` macro, located in the *Options.h* header file, to `YES` as shown in Code 3.

Code 3. `DEBUG_UART_ENABLED` Macro Located in Options.h

```
#define DEBUG_UART_ENABLED      (YES)
```

The OTA example uses `printf` statements (directly or defined in macros) to send messages through UART. `printf` requires a significant amount of heap and stack space for proper execution. So enough heap and stack memory must be allocated for proper functioning of the application while the `DEBUG_UART_ENABLED` macro is enabled. To do so, open the *.cydwr* file (for example, *PWMExample01.cydwr*) of your project by double-clicking it from **Workspace Explorer,** and then navigate to the **System** tab (see Figure 28). Set the appropriate **Heap Size** and **Stack Size** (**0x400** bytes of **Heap Size** and **0x800** bytes of **Stack Size** should be a good starting point and works for the example projects).

Figure 28. Changing Heap Size and Stack Size in PSoC Creator



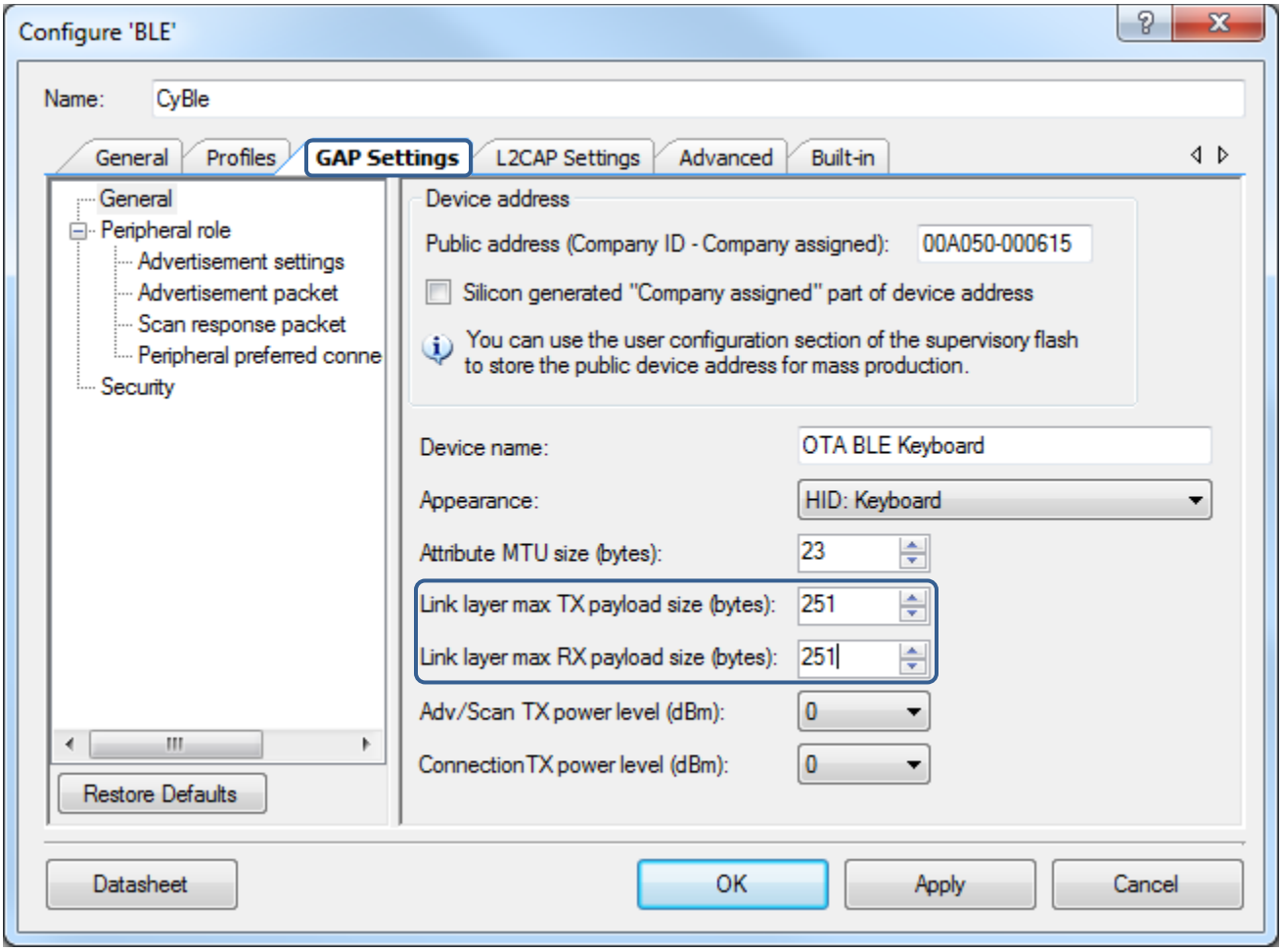
8.3 Data Length Extension (DLE)

The Bluetooth SIG introduced LE Data Packet Length Extension or Data Length Extension (DLE) in Bluetooth Core Specification 4.2. With this feature, the maximum data channel payload length in the link layer is increased from 27 bytes (in Bluetooth Core Specification 4.1 or earlier) to 251 bytes. This increases the capacity of a link layer data packet by approximately 10 times and the throughput of a link by approximately 2.6 times. More details about DLE and other features of Bluetooth Core Specification 4.2 as supported by Cypress devices can be found in AN99209.

The DLE feature is available in BLE Component version 3.0 or higher and is supported by all Bluetooth Core Specification 4.2 compliant Cypress devices. DLE can be enabled for any BLE project (including OTA projects) by following these steps.

1. Open the BLE Component configuration dialog by double-clicking the BLE Component.
2. Change the values for **Link layer max TX payload size (bytes)** and **Link layer max RX payload size (bytes)**, in the **GAP Settings** tab, to any value between (and including) **27** and **251** (see Figure 29).

Figure 29. GAP Settings Tab of BLE Component



8.4 Data Persistence

8.4.1 Using SFlash

Both PSoC and PRoC BLE devices have user SFlash regions. SFlash is used to store information such as flash protection settings, trim settings, and so on (these cannot be accessed by the user). There are also four rows of user-configurable SFlash that can be used to store Bluetooth or product-specific information such as device addresses, manufacturing/serial numbers, sensor calibration data, and so on. These user-configurable rows do not get erased during a programming cycle or during OTA upgrade, hence providing data persistence. Furthermore, SFlash can be written to during the manufacturing cycle, which is separate from the firmware programming. The user-configurable rows of the SFlash can be accessed through the firmware or the SWD programming interface.

An example project for user SFlash read/writes is available here. You can refer to the project user guide for more details on how to implement this feature in your projects. This example project is based on the 128-KB BLE devices. Table 8 provides parameter information for all BLE devices.

Table 8. Device-Specific Parameters for SFlash Example Project

Parameter	128-KB Device	256-KB Device
USER_SFLASH_ROW_SIZE	128	256
USER_SFLASH_ROWS	4	4
USER_SFLASH_BASE_ADDRESS	0x0FFFF200u	0x0FFFF400u

8.4.2 Using Checksum Exclusion

You can exclude a portion of flash from bootloadable application checksum calculations. This flash region can be used to store user or Component data (for instance, BLE stores pairing data using this method). More detail about checksum exclusion can be found in the Bootloadable Component datasheet. To enable this feature, follow these steps:

1. Relocate the target data to `cy_checksum_exclude` area. See the following code examples.

Code 4. `cy_checksum_exclude` Example for GCC Compiler

```
const uint8 byteArray[10] CY_SECTION(".cy_checksum_exclude") =
            {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
```

Code 5. `cy_checksum_exclude` Example for MDK Compiler

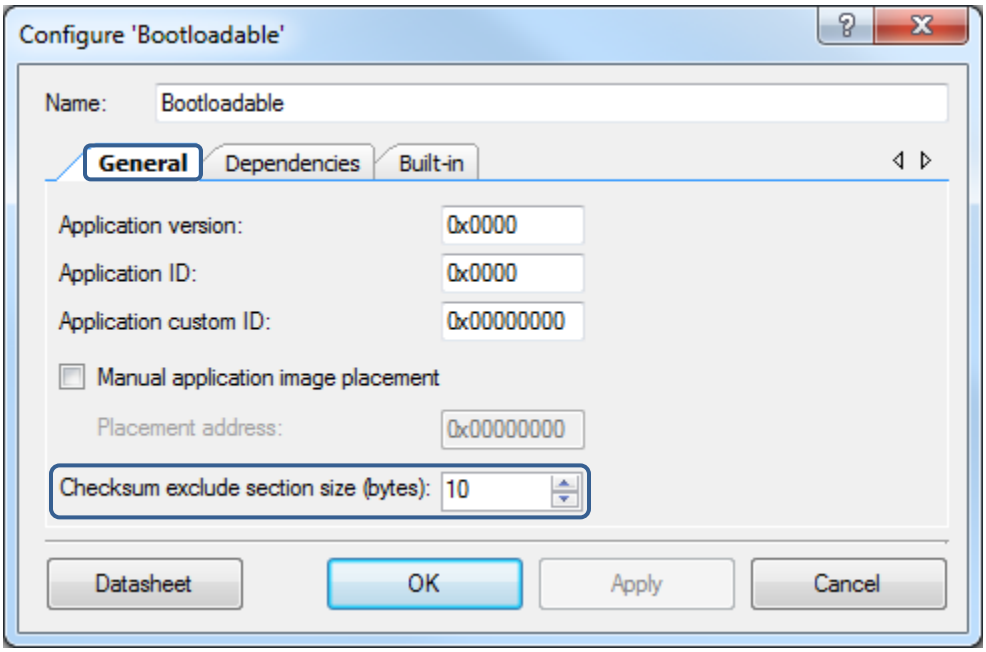
```
const uint8 byteArray[10] CY_SECTION(".cy_checksum_exclude") =
            {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
```

Code 6. `cy_checksum_exclude` Example for IAR Compiler

```
#pragma location=".cy_checksum_exclude"
const uint8 byteArray[10] =
            {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
```

2. In the Bootloadable Component in the **General** tab, set the **Checksum exclude section size (bytes)** to the required value (for Code 4, 5, and 6, this should be 10 bytes). See Figure 30.

Figure 30. Bootloadable Component Dialog



3. For the Fixed Stack OTA Bootloader, where a custom linker script (*cm0gcc.ld* present in LinkerScripts folder) is being used, the checksum exclusion information has to be manually added. To do so, follow these steps:

   a. Once the **Checksum exclude section size (bytes)** is set in the Bootloadable Component, clean (**Build** > **Clean PWMExample01**) and generate the project (**Build** > **Generate Application**). This will create new linker script files in *….\PWMExample01.cydsn\Generated_Source\PSoC4*. Or they can be accessed from **Generated Source** > **PSoC 4** > **cy_boot** in PSoC Creator **Workspace Explorer**.

b. Open the custom linker script file. In the Fixed Stack OTA example project, this file can be found in *…\PWMExample01.cydsn\LinkerScripts*. Or it can be accessed from **PWMExample01** > **LinkerScripts** in PSoC Creator **Workspace Explorer**.

c. Find the `CY_CHECKSUM_EXCLUDE_SIZE` linker script variable in the PSoC Creator generated linker script. Make the same changes to the custom linker script. For example, in this case, you need to exclude 10 bytes as follows.

Code 7. `CY_CHECKSUM_EXCLUDE_SIZE` variable in *cm0gcc.ld* linker script for GCC

```
CY_CHECKSUM_EXCLUDE_SIZE        = ALIGN(10, CY_FLASH_ROW_SIZE);
```

Code 8. `CY_CHECKSUM_EXCLUDE_SIZE` variable in *Cm0Iar.icf* linker script for IAR

```
define symbol CY_CHECKSUM_EXCLUDE_SIZE = 10;
```

Code 9. `CY_CHECKSUM_EXCLUDE_SIZE` variable in *`Cm0RealView.scat`* linker script for MDK

```
#define CY_CHECKSUM_EXCLUDE_SIZE        AlignExpr(10, CY_FLASH_ROW_SIZE)
```

d. After the changes are made, save the linker script file and build the project (**Build** > **PWMExample01**).

9 Summary

This application explained different kinds of OTA and how they can be implemented in an application. It also explained how to use the OTA upgrade feature and Cypress-provided tools to update firmware on a target device.

10 Related Application Notes

AN86526 - PSoC 4 I^2C Bootloader

AN73854 - PSoC 3, PSoC 4, and PSoC 5LP Introduction to Bootloaders

AN68272 - PSoC 3, PSoC 4, and PSoC 5LP UART Bootloader

AN91267 - Getting Started with PSoC 4 BLE

AN94020 - Getting Started with PRoC BLE

About the Author

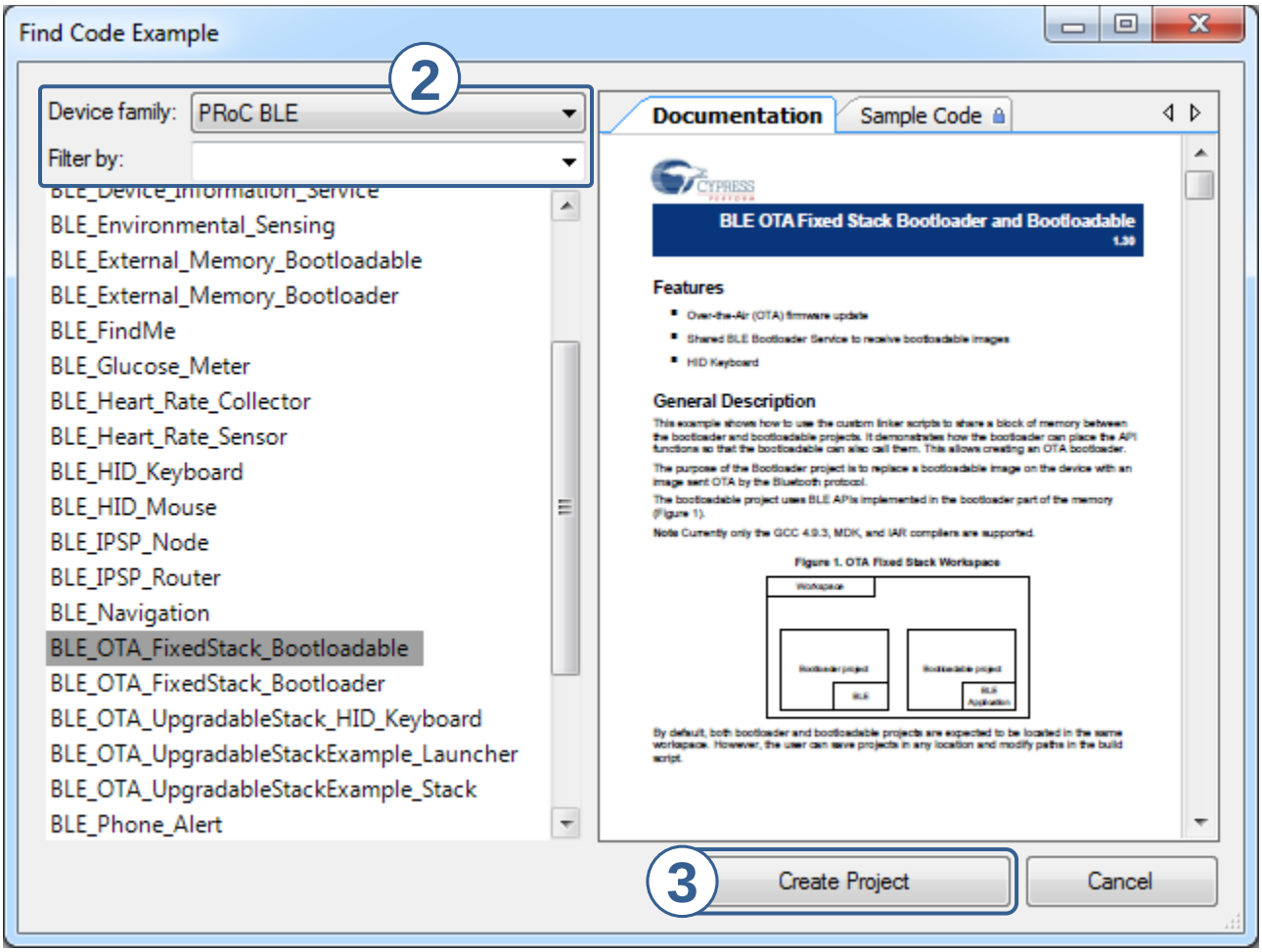
Name:	Deepak John
Title:	Staff Applications Engineer
Background:	Deepak John has a B.Tech in Electronics and Communication Engineering from Cochin University of Science and Technology and an MSEE from University of Bridgeport, as well as many years of experience designing and building embedded systems.

A Appendix A

A.1 Creating an Example Project Workspace

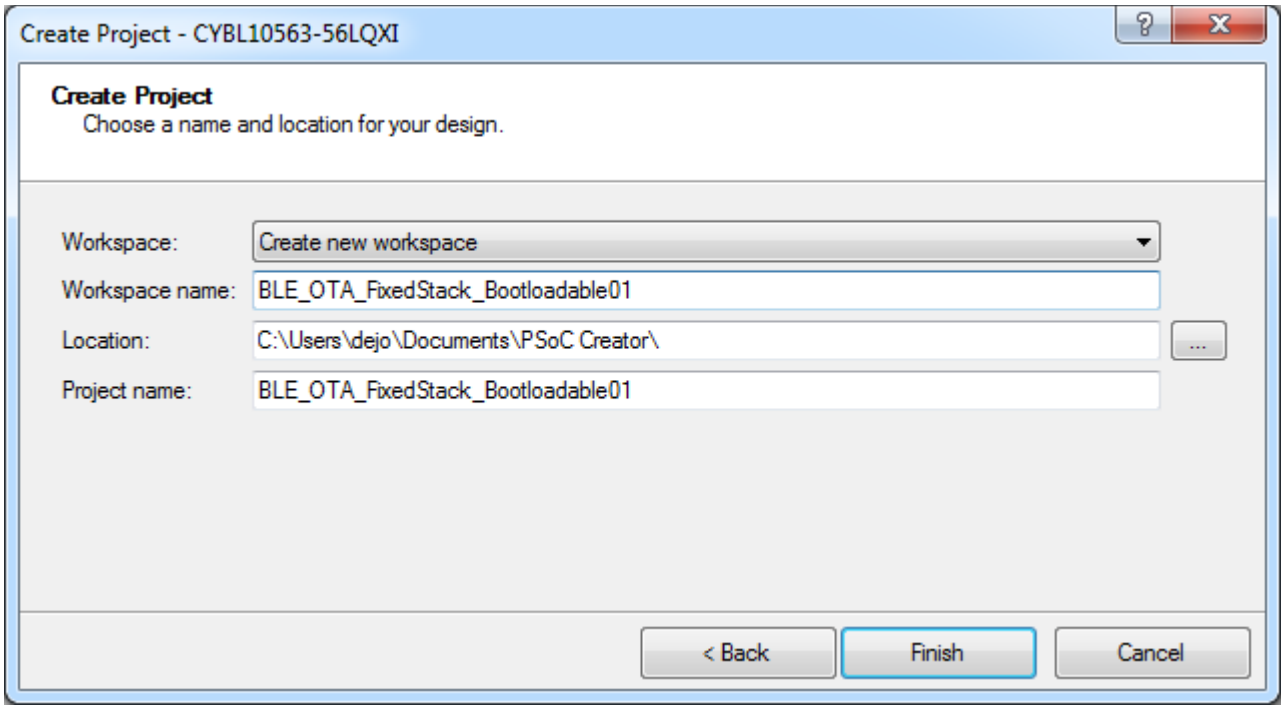
1. In PSoC Creator, choose **File** > **Code Example…** to open the **Find Example Project** dialog, as shown in Figure 31.
2. Apply the necessary **Device Family** and **Filter by** keywords to narrow your search, as shown in Figure 31.
3. Select the example project of your choice and click the **Create Project** button.

Figure 31. Find Code Example Project Dialog – Creating an Example Project Workspace



4. Select the **Create new workspace** option for **Workspace**. Select a **Location** for the new example project workspace (see Figure 32) and click **Finish**.
5. Select the required path and click **Finish**.

Figure 32. Create Project Dialog – Creating an Example Project Workspace



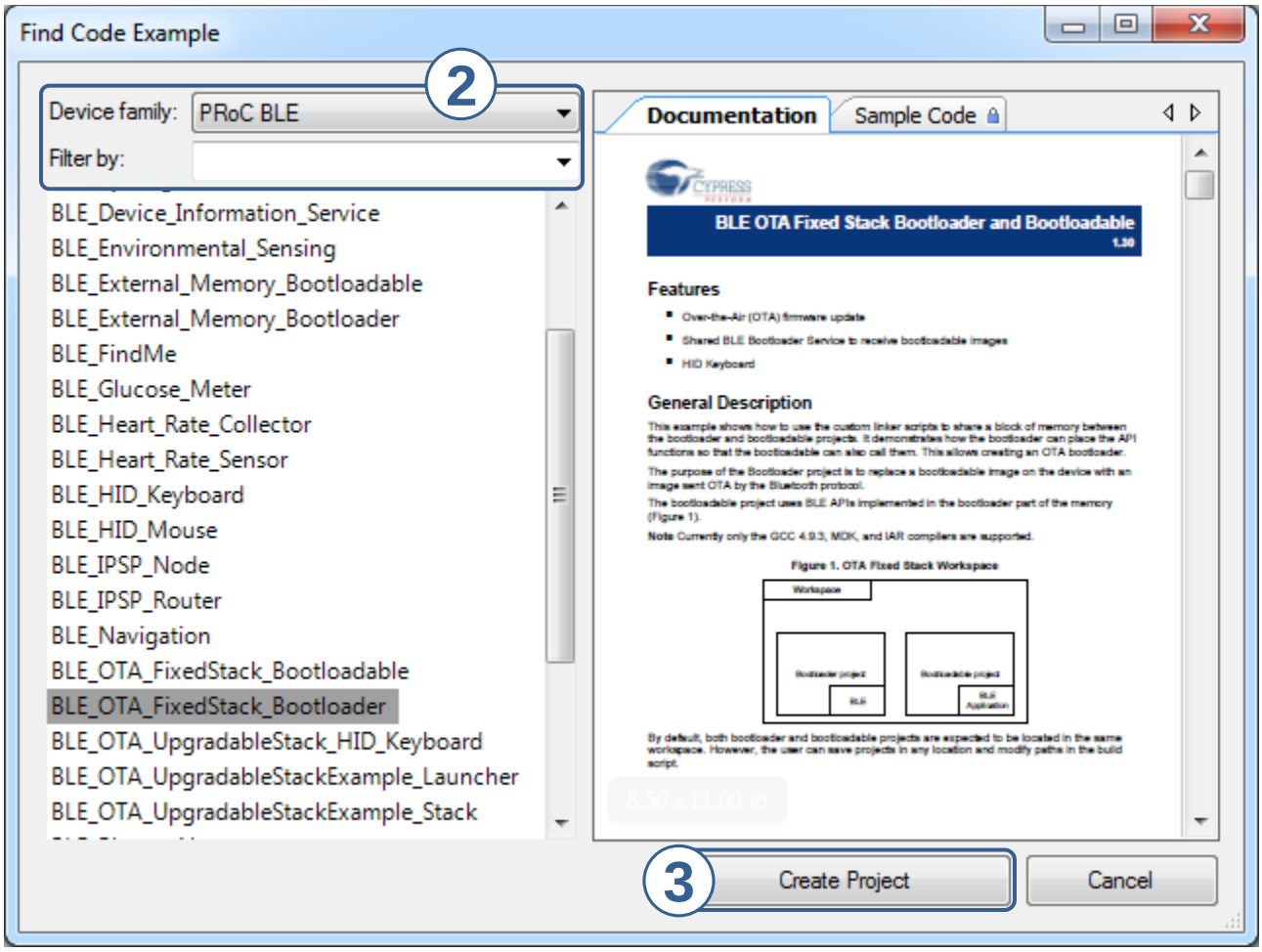
On completing these steps, a workspace is created in the chosen location. The selected example project will be added to this workspace. A datasheet *(<project_name>.pdf*) is present under the newly created project. You can review this document for details about the example project.

A.2 Adding an Example Project to an Existing Workspace

Before you follow the steps in this section, make sure a Project/Workspace is already open in a PSoC Creator instance. To open an existing Project/Workspace, choose **File** > **Open** > **Project/Workspace**.

1. Choose **File** > **Code Example…** to open the **Find Example Project** dialog, as shown in Figure 33.

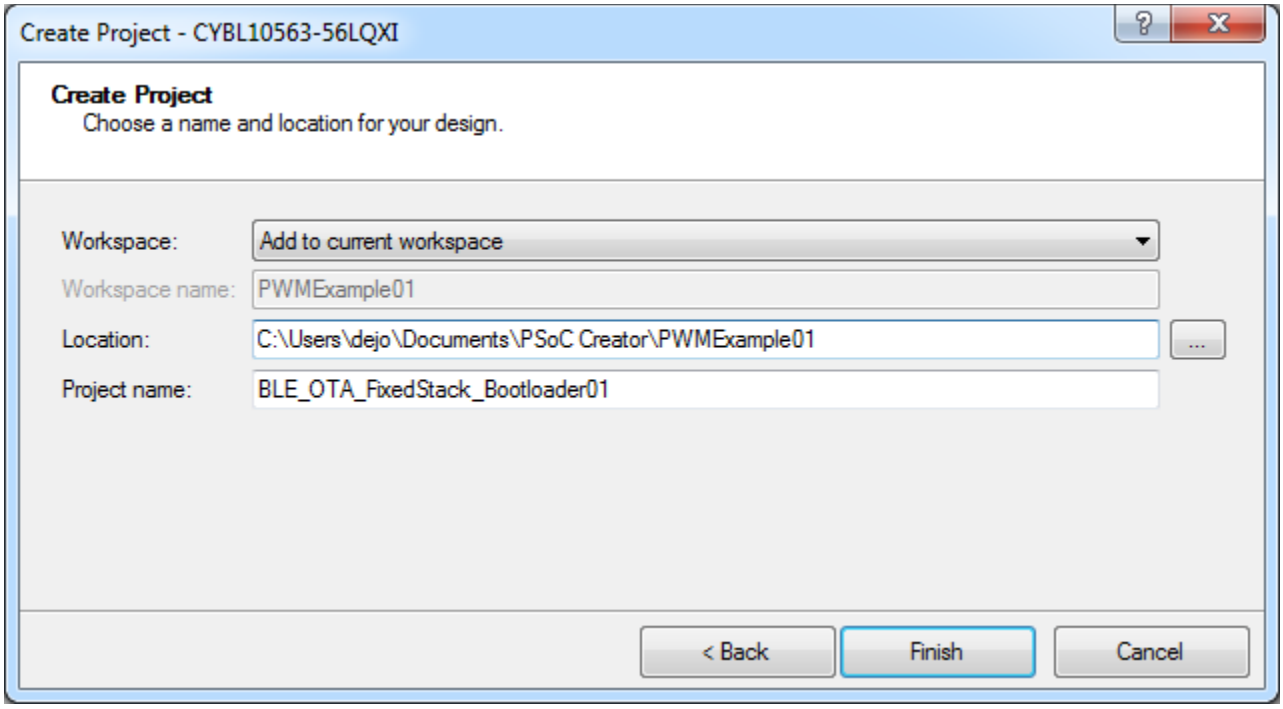
Figure 33. Find Example Project Dialog – Adding Example Project to Existing Workspace



2. Apply the necessary **Device family** and **Filter by** keywords to narrow your search, as shown in Figure 33.
3. Select the example project of your choice and click the **Create Project** button.
4. Select the **Add to current workspace** option for **Workspace**. Select a **Location** for the new example project workspace (see Figure 34) and click **Finish**.
5. Select the required path and click **Finish**.

Once you have completed these steps, the selected example project will be added to the workspace already open in that instance of PSoC Creator. A datasheet (*<project_name>.pdf*) will be present under the newly added project. You can review this document for details about the example project.

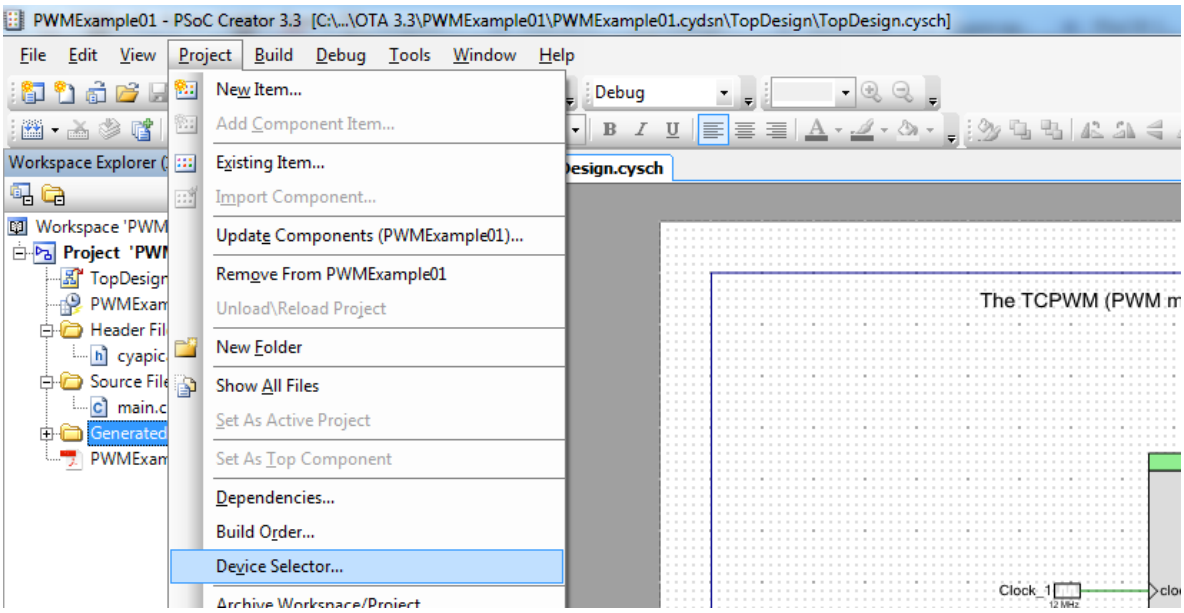
Figure 34. Create Project Dialog – Adding Example Project to Existing Workspace



A.3 Selecting Another Device

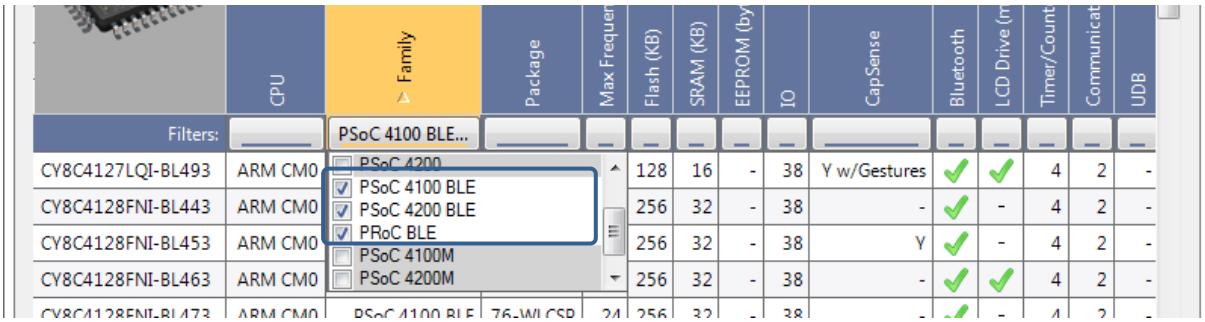
1. In PSoC Creator, choose **Project** > **Device Selector…** (see Figure 35).

Figure 35. Launching Device Selector



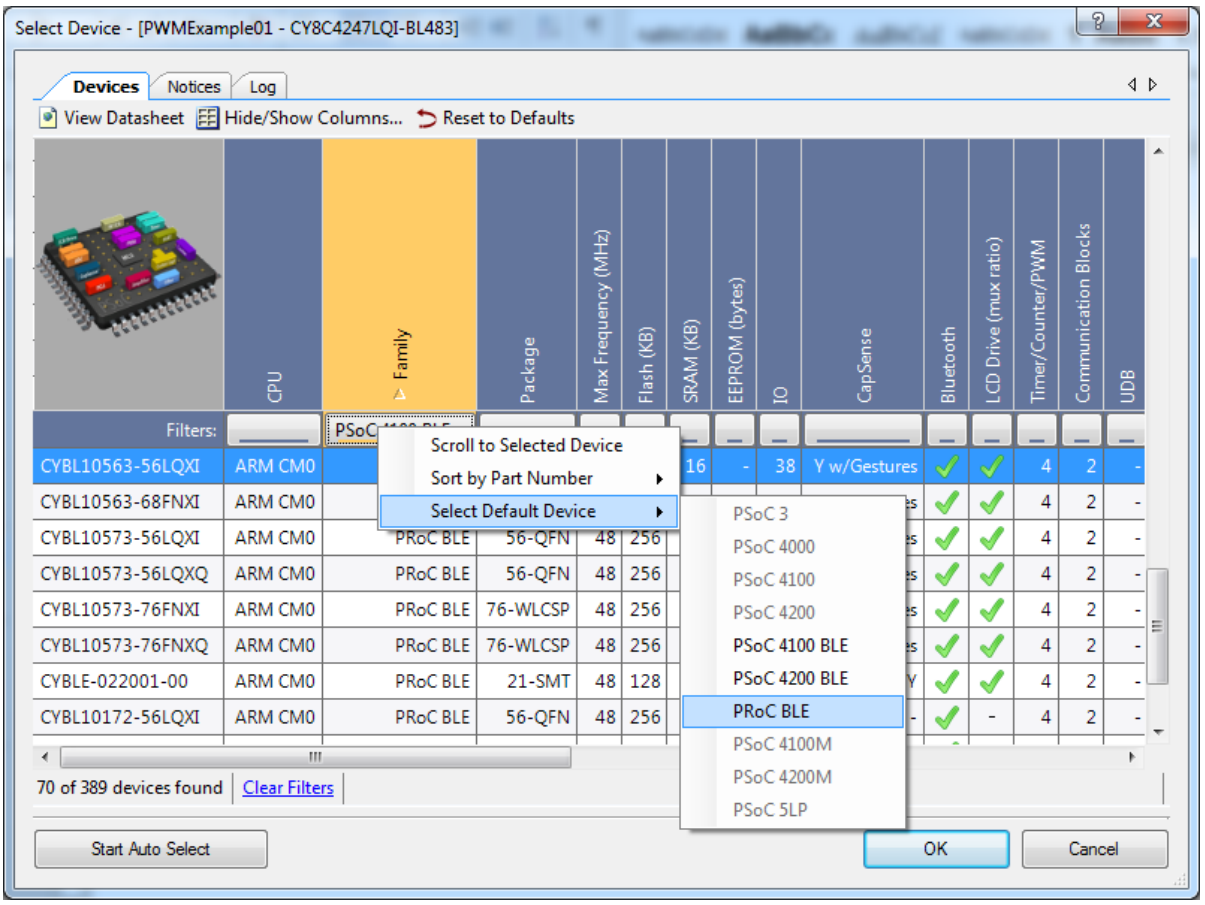
2. Enable the **PSoC 4200 BLE, PSoC 4100 BLE**, and **PRoC BLE** filter items in the **Family** filter category, as shown in Figure 36.

Figure 36. Select Device Family



3. Select the appropriate BLE device for your application from the list. If your target is the PRoC BLE/PSoC 4 BLE module that comes with the CY8CKIT-042-BLE Pioneer Kit, right-click the **Family** filter, select **Select Default Device,** and then select the appropriate device family, as shown in Figure 37.

Figure 37. Select Default PRoC BLE Device for CY8CKIT-042-BLE Pioneer Kit



4. Click **OK** to close the Device Selector.

A.4 Adding Bootloader Service

Follow the steps in this section to add a bootloader service to an existing BLE Component. See the BLE Component datasheet to learn more about the bootloader service.

1. Double-click the BLE Component to open the configuration dialog (see Figure 38).
2. Go to the **Profiles** tab and select a service/profile (like, **HID Device** in Figure 38).
3. Click **Add Service** (see Figure 38) to bring up the available services, and then select **Bootloader** (see Figure 39).
4. Set the **Data** field at **Bootloader** > **Command** > **Fields** to 137 (see Figure 40).
5. Set the **Security level** (at the **GAP Settings** > **Security** tab) to **Unauthenticated pairing with encryption** (see Figure 41). Example projects supplied with PSoC Creator use this setting, and it is required for procedures mentioned in this application note to work without issues. Alternatively, use the security option (selected in the Component) for the update tool (CySmart) while performing updates. This step is not mandatory.

Figure 38. BLE Component Configuration Dialog

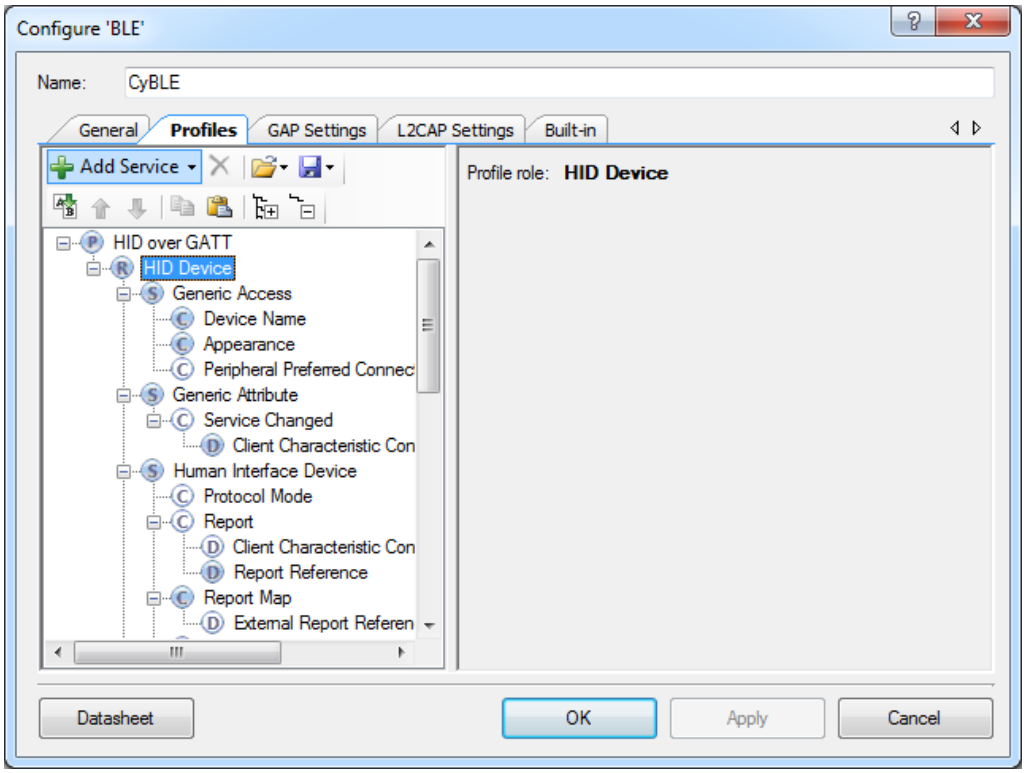


Figure 39. Select Bootloader Service

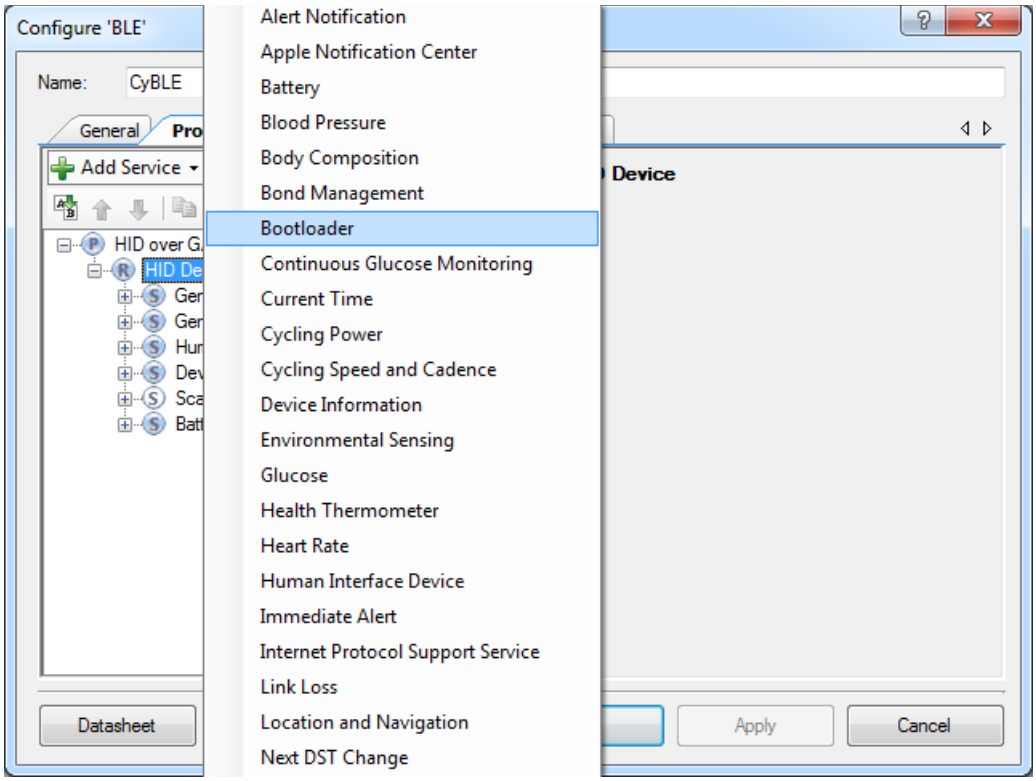


Figure 40. Set Bootloader Service Data Field to 137

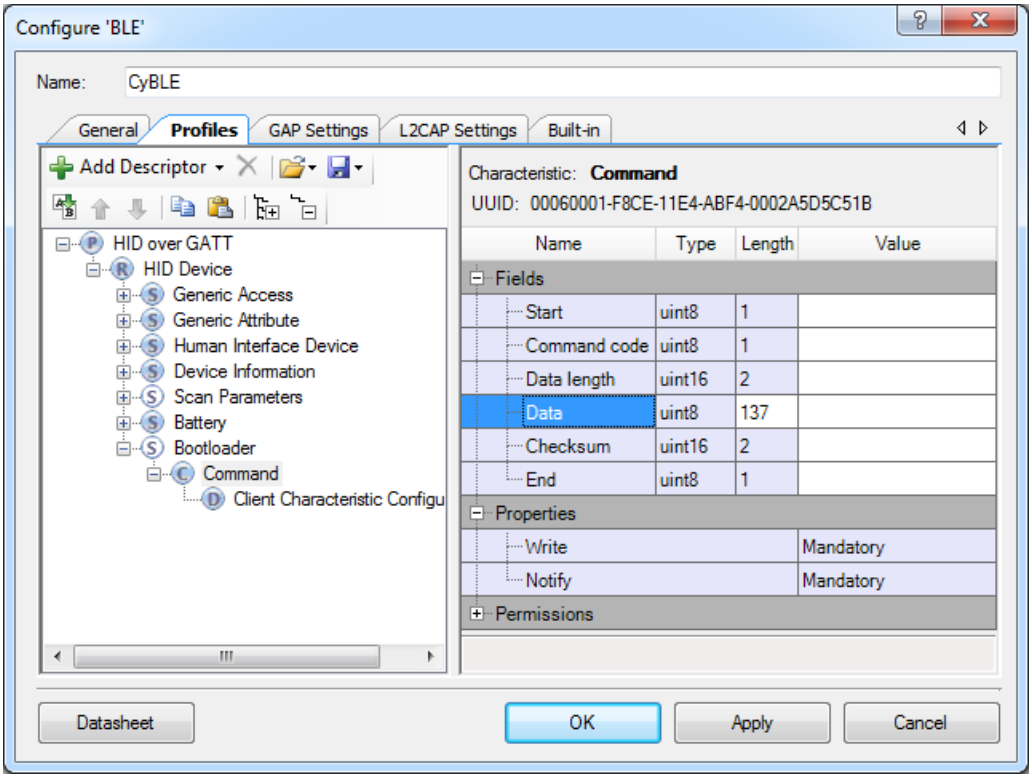
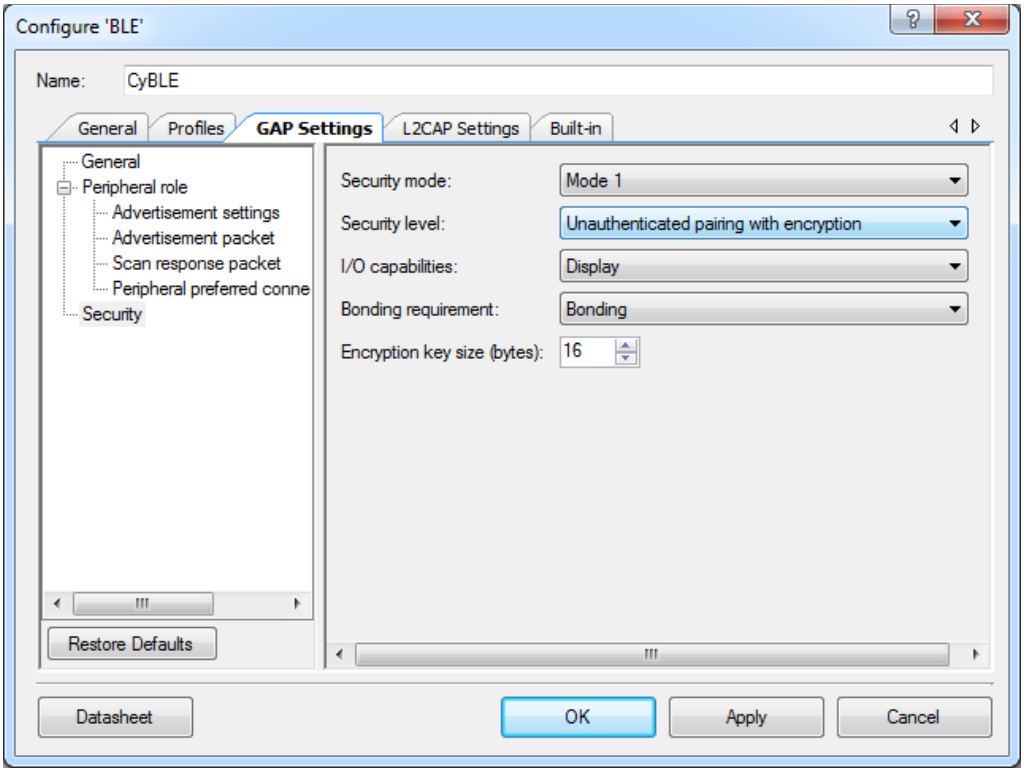


Figure 41. Change Security Option



A.5 Configuring Fixed Stack OTA Projects for Other Cypress BLE Devices

BLE OTA Fixed Stack example projects use custom linker scripts, so PSoC Creator will not be able to configure linker scripts to provide the correct linking and placement. This section describes steps to follow if you are not using the CY8C4247LQI-BL483 or CYBL10563-56LQXI devices or are using devices that have a different size of RAM or ROM memory. By default, the linker scripts include settings for a 128-KB device.

The following sections describe linker script modification after you have changed the device in PSoC Creator.

A.5.1 GCC Compiler

For the GCC compiler, changes are required for both the bootloader and bootloadable linker scripts as follows.

1. Change the device flash memory size to 262144 for 256-KB devices (see Figure 42).
2. Change the device RAM size to 32768 for 256-KB devices (see Figure 42).
3. Change the device row size to 256 for 256-KB devices (see Figure 42).

Figure 42. cm0gcc.ld File Contents Showing Configuration for 128-KB Devices

```
26
27  MEMORY
28  {
29      rom (rx) : ORIGIN = 0x0, LENGTH = 131072
30      ram (rwx) : ORIGIN = 0x20000000, LENGTH = 16384
31  }
32
33
34  CY_APPL_ORIGIN                  = 0;
35  CY_FLASH_ROW_SIZE               = 128;
36  CY_APPL_NUM                     = 1;
37  CY_APPL_MAX                     = 1;
38  CY_METADATA_SIZE                = 64;
39  CY_APPL_LOADABLE                = 0;
40  CY_CHECKSUM_EXCLUDE_SIZE        = ALIGN(0, CY_FLASH_ROW_SIZE);
41  CY_APP_FOR_STACK_AND_COPIER     = 0;
42
```

Linker scripts are located in *…\cm0gcc.ld* for the bootloader project and in *…\LinkerScripts\cm0gcc.ld* for the bootloadable project.

If you are not sure about the values that are to be entered for your device, you can refer to the values in the linker script generated by PSoC Creator, even though it is not used. It is located in the folder *%PROJECT_DIR%\Generated_Source\PSoC4\cy_boot* and has the name *cm0gcc.ld*.

A.5.2 MDK Compiler

For the MDK compiler, only the linker script of bootloadable project needs to be changed. The linker script is located in folder *…\LinkerScripts\Cm0Mdk.scat.*

1. Change the device flash memory size to 262144 for 256-KB devices (see Figure 43).
2. Change the device row size to 256 for 256-KB devices (see Figure 43).

Figure 43. Cm0Mdk.scat File Contents Showing Memory and Row Size for 128-KB Devices

```
36
37  #define CY_FLASH_SIZE           131072
38  #define CY_APPL_ORIGIN          0
39  #define CY_FLASH_ROW_SIZE       128
40  #define CY_METADATA_SIZE        64
41
```

3. Change the device RAM size (both values) to `32768` for 256-KB devices (see Figure 44).

Figure 44. Cm0Mdk.scat File Contents Showing RAM Size for 128-KB Devices

```
125     ARM_LIB_HEAP (0x20000000 + 16384 - 0x400 - 0x0800) EMPTY 0x400
126     {
127     }
128
129     ARM_LIB_STACK (0x20000000 + 16384) EMPTY -0x0800
130     {
131     }
```

If you are not sure about the values to be entered for your device, you can refer to the values in the linker script that is generated by PSoC Creator, even though it is not used after a project rebuild. It is located in the folder *%PROJECT_DIR%\Generated_Source\PSoC4\cy_boot* and has the name *Cm0Mdk.scat*.

A.5.3 IAR Compiler

If you use IAR, for the linker script of the bootloader project, it is safe to use the one that is generated by PSoC Creator after the device is changed in PSoC Creator. However, the bootloadable project linker script must be modified. It is located in folder …*LinkerScripts**Cm0Iar.icf.*

1. Change the device flash memory size to 262144 for 256-KB devices (see Figure 45).
2. Change the device RAM size to 32768 for 256-KB devices (see Figure 45).
3. Change the device row size to 256 for 256-KB devices (see Figure 45).

Figure 45. Cm0Iar.icf File Contents Showing Configuration for 128-KB Devices

```
7  define symbol __ICFEDIT_region_ROM_start__  = 0x0;
8  define symbol __ICFEDIT_region_ROM_end__    = 131072 - 1;
9  define symbol __ICFEDIT_region_RAM_start__  = 0x20000000;
10 define symbol __ICFEDIT_region_RAM_end__    = 0x20000000 + 16384 - 1;
11 /*-Sizes-*/
12 define symbol __ICFEDIT_size_cstack__ = 0x0800;
13 define symbol __ICFEDIT_size_heap__   = 0x400;
14 /**** End of ICF editor section. ###ICF###*/
15
16
17 /******** Definitions ********/
18 define symbol CY_FLASH_SIZE     = 131072;
19 define symbol CY_APPL_ORIGIN    = 0;
20 define symbol CY_FLASH_ROW_SIZE = 128;
21 define symbol CY_APPL_LOADABLE  = 1;
22 define symbol CY_APPL_LOADER    = 0;
23 define symbol CY_APPL_NUM       = 1;
24 define symbol CY_METADATA_SIZE  = 64;
25 define symbol CY_APPL_MAX       = 1;
26 define symbol CY_CHECKSUM_EXCLUDE_SIZE = 0;
27 define symbol CY_APPL_FOR_STACK_AND_COPIER  = 0;
28 define symbol CY_FIRST_AVAILABLE_META_ROW   = 1;
```

If you are not sure about the values to be entered for your device, you can refer to the values in the linker script that is generated by PSoC Creator, even though it is not used after a project rebuild. It is located in the folder *%PROJECT_DIR%\Generated_Source\PSoC4\cy_boot* and has the name *Cm0Iar.icf*.

Document History

Document Title: AN97060 - PSoC® 4 BLE and PRoC™ BLE – Over-The-Air (OTA) Device Firmware Upgrade (DFU) Guide

Document Number: 001-97060

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	4938472	DEJO	10/01/2015	New application note.
*A	4988612	DEJO	10/26/2015	Added Configuring Fixed Stack OTA Projects for Other Cypress BLE Devices Added write characteristic value timed Out error details in Upgrading Through CySmart PC Tool Added steps to configure linker scripts for different devices in section Adding a Fixed Stack OTA Bootloader Replaced EEPROM with external memory Updated figures, formatting and table of contents
*B	5279419	DEJO	05/27/2016	Added steps to enable bonding in section 8.1 Added Data Length Extension (DLE) section Added Data Persistence section Updated section 5.2, Adding an External Memory OTA Bootloader Updated section 5.3, Adding a Fixed Stack OTA Bootloader Updated section 5.4, Adding an Upgradable Stack OTA Bootloader Updated section 6, Performing an OTA Upgrade Updated section 8.1, Bonding / Pairing Information Updated section 8.2, Debugging Updated sections A.1, A.2, A.3, A.4, and A.5 Updated subheading formatting in sections 6.3 and 8.1 Updated figures, formatting, and table of contents
*C	5687926	BENV	04/19/2017	Updated template

Worldwide Sales and Design Support

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

Products

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

PSoC® Solutions

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

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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