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

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

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

CE220959 – PSoC 6 MCU Bluetooth Low Energy (BLE) Device Firmware Update (DFU) with External Memory

Objective

This example demonstrates device firmware update (DFU), also known as "bootloading", over the air (OTA) with PSoC® 6 MCU with Bluetooth Low Energy (BLE) connectivity, with external memory. This includes downloading an application from a host, installing it in external memory, copying it to the PSoC 6 MCU device's flash memory, and then transferring control to that application. The downloaded application demonstrates some basic Bluetooth services. It is based on CE215121 BLE HID Keyboard.

Requirements

Tool: PSoC Creator™ 4.2; Peripheral Driver Library (PDL) 3.1.0

Programming Language: C (Arm® GCC 5.4.1 and Arm MDK 5.22)

Associated Parts: All PSoC 63 line parts

Related Hardware: CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit

Note: The PSoC 6 BLE Pioneer kit is shipped with KitProg2; PSoC Creator works only with KitProg2. If your kit was upgraded to KitProg3 for use with ModusToolbox™ IDE, revert the kit to KitProg2 before using this code example. See ModusToolbox **Help** > **ModusToolbox IDE Documentation** > **User Guide**; section **PSoC 6 MCU KitProg Firmware Loader**.

Overview

This example demonstrates several basic DFU operations:

- Communicating with a host via BLE, and downloading an application. The application is saved in a temporary location in a Quad SPI (QSPI) external memory device on the CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit board.
- Validating the downloaded application in the QSPI memory, and then copying it to the PSoC 6 MCU device flash.
- Validating an application in the PSoC 6 MCU device flash, and then transferring control to that application.

There are two PSoC Creator projects, generally called "App0" and "App1". The projects have the following features:

- App0 does the DFU operation; it downloads and installs App1 into the QSPI external memory. It also copies App1 from the external memory to the PSoC 6 MCU device flash.
- Each project blinks a kit LED at a different rate or color making it easy to see which one is currently running.
- Transfers control between applications using the BLE Immediate Alert Service (IAS).

Other communication channels such as UART, I^2C, and SPI are demonstrated in other code examples; see Related Documents.

Hardware Setup

This example uses the kit's default configuration. Refer to the kit guide to ensure the kit is configured correctly.

Install the BLE USB dongle (CY5677) in a USB port in your computer.

Software Setup

Install the latest CySmart™ tool in your computer to use the BLE USB dongle.

To customize the DFU operation and enable DFU software development kit (SDK) features, update the `#define` statements as needed in the *dfu_user.h* file. The default settings can be used for most designs.

Operation

Build the Projects

Note: If you are using a version of the PDL that is different from that specified in the Requirements section, PSoC Creator may have cleared the PDL software package import selections. Check the project **Build Settings** > **Peripheral Driver Library** > **DFU**. For more information, see PSoC Creator Help, or AN213924, *PSoC 6 MCU Device Firmware Update Software Development Kit Guide*.

Note: In some cases, you may be prompted to replace files from your project with files from the PDL. These files are templates. Do not replace the customized files for the project. Click **Cancel**.

1. Using PSoC Creator, build the App0 and App1 projects in that order. For more information on building projects, see PSoC Creator Help.

   Note: For the MDK compiler, there is a known compile-time defect documented in PDL 3.1.0 Release Notes. For a workaround, first select **Build** > **Generate Application**. Then, in the *dfu_mdk_common.h* generated file, comment out the line containing "`__asm void cy_DFU_mdkAsmDummy(void);`". Finally, complete the project build by selecting **Build** > **Build <project name>**.

2. In each project, confirm that the linker script files are set up for the corresponding application number, as the following examples show:

   - For the GCC compiler, the following example shows edits for App1 in *dfu_cm4.ld*. Note that the app ID is set to 2, not 1, to facilitate storage of the downloaded application in external memory.

     ```
     /*
     * DFU SDK specific: aliases regions, so the rest of code does not use
     * application specific memory region names
     */
     REGION_ALIAS("flash_core0", flash_app1_core0);
     REGION_ALIAS("flash",       flash_app1_core1);
     REGION_ALIAS("ram",           ram_app1_core1);

     /* DFU SDK specific: sets app Id */
     __cy_app_id = 2;
     ```

   - For the MDK compiler, the following example shows edits for App1 in *dfu_cm0p.scat*:

     ```
     ; Flash
     #define FLASH_START             CY_APP1_CORE0_FLASH_ADDR
     #define FLASH_SIZE              CY_APP1_CORE0_FLASH_LENGTH

     ; Emulated EEPROM Flash area
     #define EM_EEPROM_START         CY_APP1_CORE0_EM_EEPROM_ADDR
     #define EM_EEPROM_SIZE          CY_APP1_CORE0_EM_EEPROM_LENGTH

     ; External memory
     #define XIP_START               CY_APP1_CORE0_SMIF_ADDR
     #define XIP_SIZE                CY_APP1_CORE0_SMIF_LENGTH

     ; RAM
     #define RAM_START               CY_APP1_CORE0_RAM_ADDR
     #define RAM_SIZE                CY_APP1_CORE0_RAM_LENGTH
     ```

And edits for App1 in *dfu_mdk_symbols.c*. Note that the app ID is set to 2, not 1, to facilitate storage of the downloaded application in external memory.

```
__cy_app_core1_start_addr   EQU __cpp(CY_APP1_CORE1_FLASH_ADDR)

/* Application number (ID) */
__cy_app_id                 EQU 2

/* CyMCUElfTool uses these to generate an application signature */
__cy_app_verify_start     EQU __cpp(CY_APP1_CORE0_FLASH_ADDR)
__cy_app_verify_length    EQU __cpp(CY_APP1_CORE0_FLASH_LENGTH +
                                    CY_APP1_CORE1_FLASH_LENGTH -
                                    __CY_BOOT_SIGNATURE_SIZE)
```

If linker script files are edited, rebuild the corresponding project.

App0 can either do a DFU operation or transfer control to a previously downloaded application. The following steps explain how to download an application and how to transfer control between the two applications.

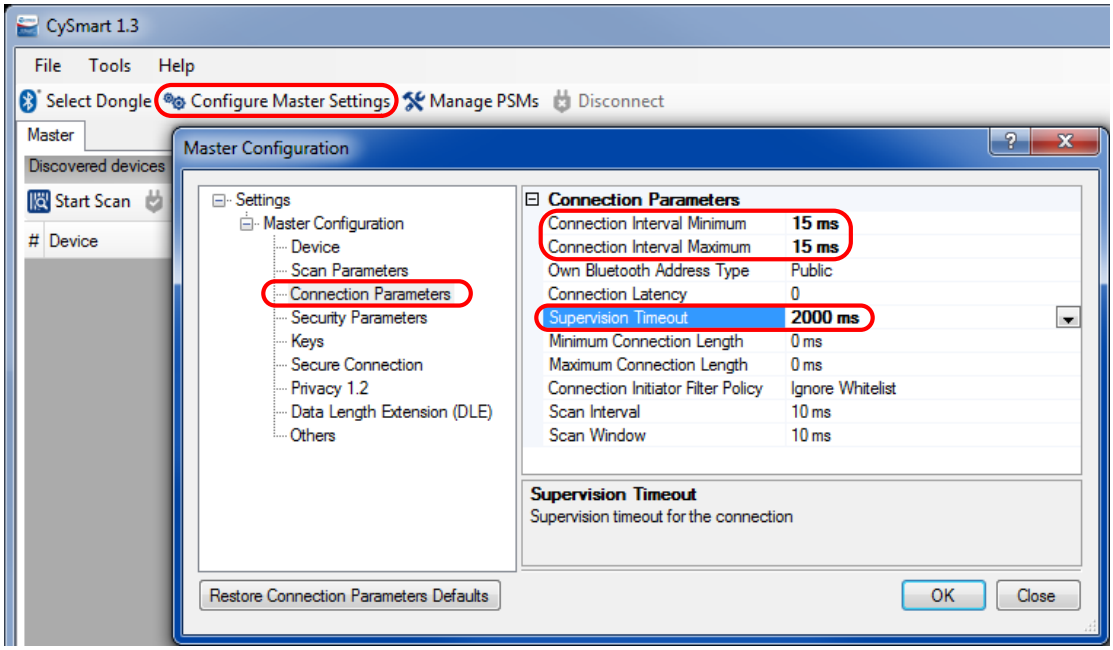
Test the Projects: Install App1

1. Connect the kit board to your PC using the provided USB cable.
2. Program the App0 project into the kit. For more information on device programming, see PSoC Creator Help.

   Confirm that the kit RGB LED blinks white once every two seconds. This indicates that App0 is running.
3. Press the kit button SW2 for at least half a second, and then release it. Confirm that nothing happens because App0 is the only application installed.
4. Connect the BLE USB dongle (CY5677) provided with the CY8CKIT-062-BLE kit to your computer.
5. Run the CySmart tool on your computer and connect to the BLE USB dongle.
6. Click **Configure Master Settings**. Go to **Connection Parameters**. Change the **Connection Interval Minimum**, **Connection Interval Maximum**, and **Supervision Timeout** to 15, 15, and 2000 ms respectively, as Figure 1 shows. Click **OK**.

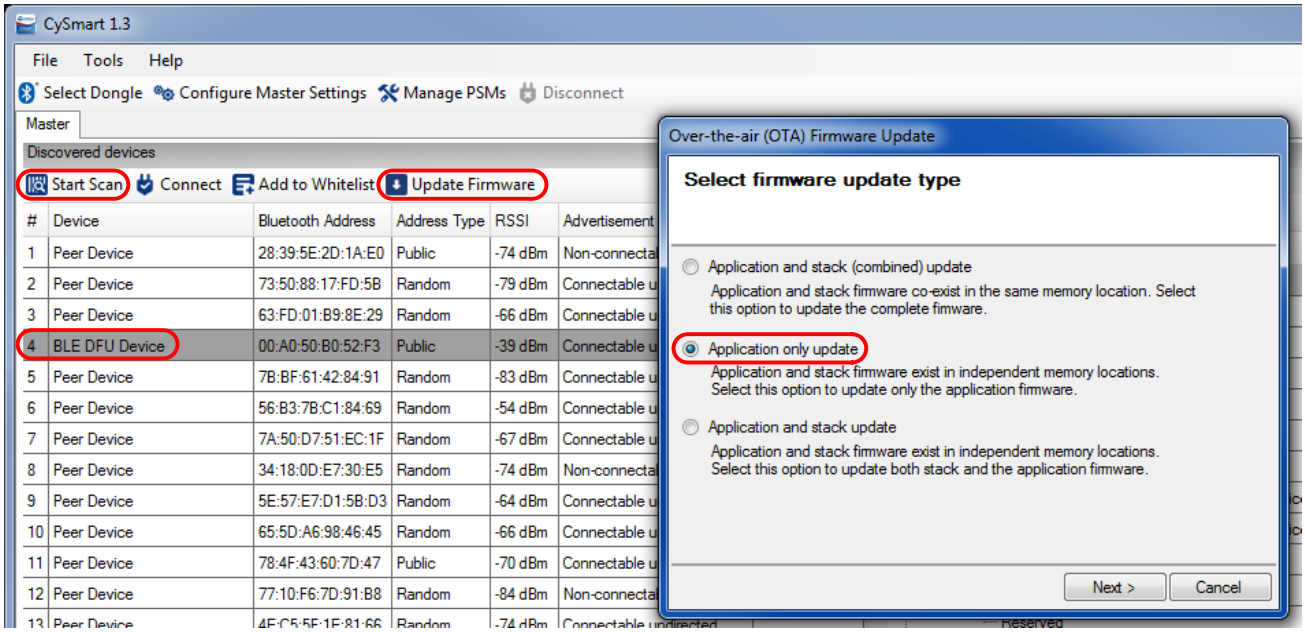
   Note: Using a lower connection interval speeds up application transmission at an increased risk of losing the connection.

Figure 1. CySmart Connection Parameters



7. In CySmart, click **Start Scan** to start scanning for the DFU device, as Figure 2 shows. When “BLE DFU Device” is listed, click **Stop Scan**. When scanning stops, select that device. Click **Update Firmware** and choose the **Application only update** option. Click **Next**.

Figure 2. CySmart Firmware Update



8. Browse to and select the App1 file (*PSoC6DfuBleApp1.cyacd2)* located in the project folder *PSoC6DfuBleApp1* > *CortexM4* > *[compiler name]* > *Debug*. This file is generated when App1 is built. Click **Update**.
9. Wait for the device firmware to be updated. While the firmware is downloading, the kit RGB LED blinks in white twice every two seconds.
10. After download is complete, confirm that the kit RGB LED blinks fast in white, indicating that App1 has been validated in external memory and is being copied to the PSoC 6 MCU device flash. This operation requires around 8 seconds.
11. Confirm that the RGB green LED blinks, indicating that App1 is running.
12. Switch to App0 using any of the methods shown below. Repeat steps 7 – 11 to test the process of reinstalling App1.

Test the Projects: Transfer Applications

When App1 has been installed into the PSoC 6 MCU device flash, control is automatically transferred to App1 whenever the device is powered ON or reset. The following steps show multiple ways to transfer control between the applications.

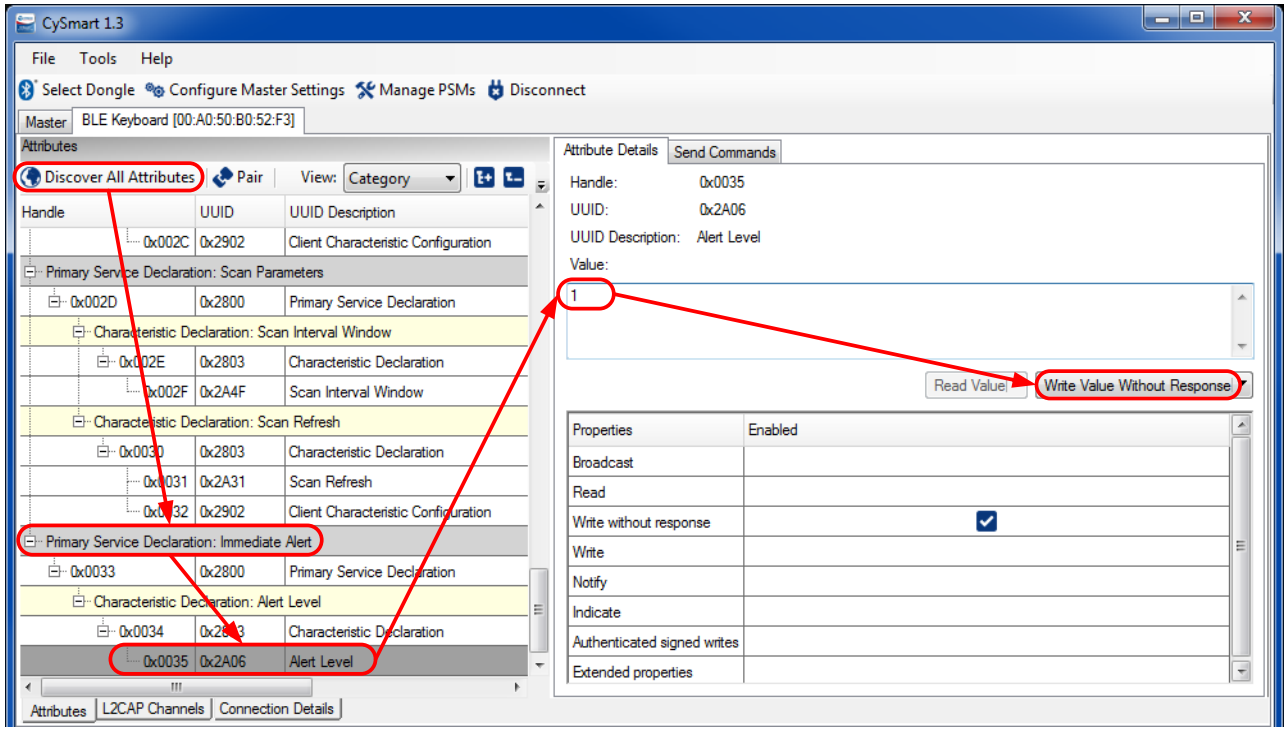
Transfer Using IAS

1. In CySmart, click **Start Scan**. When listed, select “BLE DFU Device” or “BLE Keyboard” for App0 and App1 respectively.
2. Click **Connect** to connect to the device.
3. Click **Pair** to pair with the device. The Pair button may be hidden if the window size is small. For App1, click **No** when prompted to add the device to the resolving list. For App0, just click **OK**.

   Note: If pairing fails, disconnect from the device and clear the device list in the CySmart tool. Go back to step 1.

4. Click **Discover All Attributes**, as Figure 3 shows. Navigate to the **Immediate Alert** service at the bottom and click **Alert Level**. Enter **1** in the **Value** textbox and click **Write Value Without Response**. Confirm that the other application is now running.

Figure 3. Transferring Control Between Applications Using IAS



Transfer Using the Kit Buttons

This method transfers control only from App1 to App0.

1. Confirm that App1 is running.
2. Press the kit reset and user buttons – SW1 and SW2 – at the same time.
3. Release the reset button (SW1).
4. Wait until the kit RGB LED starts blinking in white before releasing the user button. Confirm that App0 is now running.

Debugging

You can debug the example and step through the code. PSoC 6 MCU has two CPUs: an Arm Cortex®-M4 (CM4) and a Cortex-M0+ (CM0+). PSoC Creator supports debugging a single CPU (either CM4 or CM0+) at a time; in this example Table 5 on page 11 shows the tasks that are done by each of the CPUs. To debug App1, you may need to use the **Debug** > **Attach to Running Target...** option. For more information on debugging using PSoC Creator, see PSoC Creator Help.

Design and Implementation

This example has two applications: “App0” and “App1”. Each application is a separate PSoC Creator project; both projects are in the same PSoC Creator workspace. The applications have the following features:

- App0 does the DFU; it downloads, installs, and transfers control to the downloadable application (App1)
- A PSoC Creator SMIF Component is used to access the 64-MB Quad SPI memory device (“external memory”) mounted on the kit board. The application is stored and validated in this device and afterwards copied into the PSoC 6 MCU device flash.
- A warning sequence at App0 startup indicates the status of App1 in the PSoC 6 MCU device flash and external memory

- After downloading and copying App1, both applications reside in the PSoC 6 MCU device flash; see Memory Layout. Whenever the device is powered ON or reset, App0 runs first. It checks whether a valid App1 exists, and if so transfers control to it. The transfer operation may take longer if App1 is valid only in the external memory, because the application must first be copied into the PSoC 6 MCU device flash. For more information, see Design Firmware.
- You can transfer control between applications using the Bluetooth IAS, or using the kit buttons.
- After 300 seconds of inactivity within App0, it transfers control to App1. If there is no valid App1, App0 hibernates.
- App1 demonstrates several Bluetooth services. It is a lightly modified version of CE215121 BLE HID Keyboard with DFU SDK support added.

Figure 4 and Figure 5 on page 7 show the PSoC Creator project schematics for App0 and App1, respectively. For more information on App1, see CE215121.

Figure 4. App0 Schematic

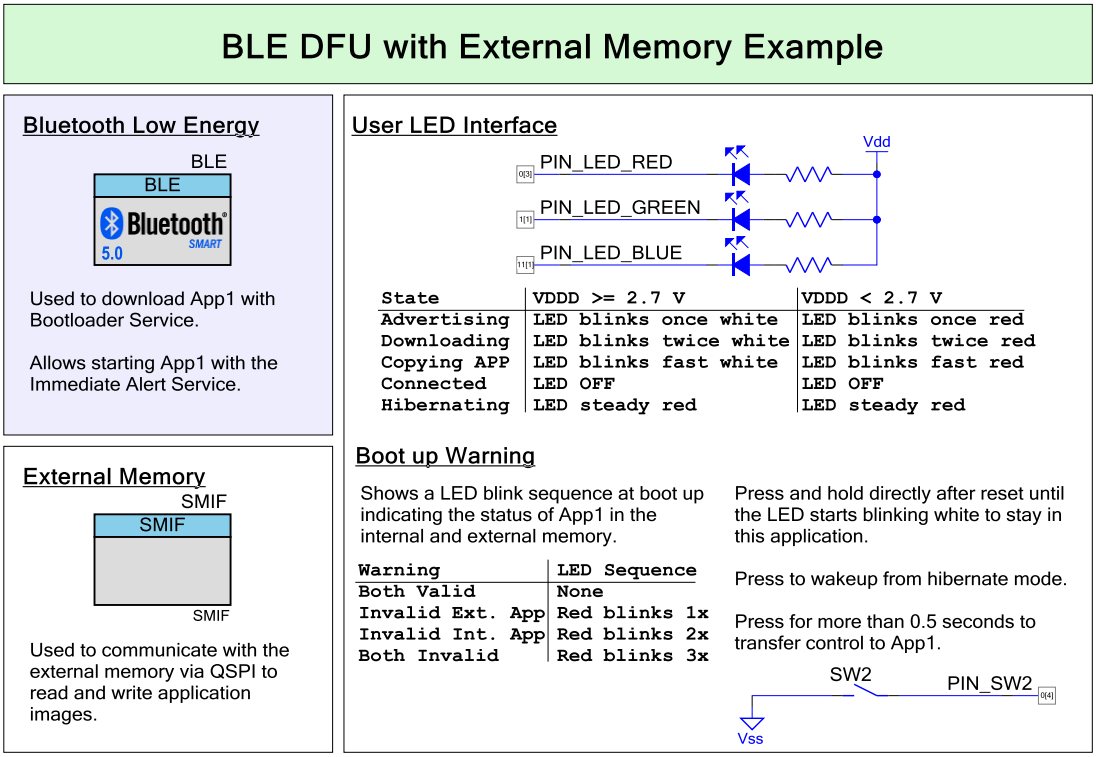
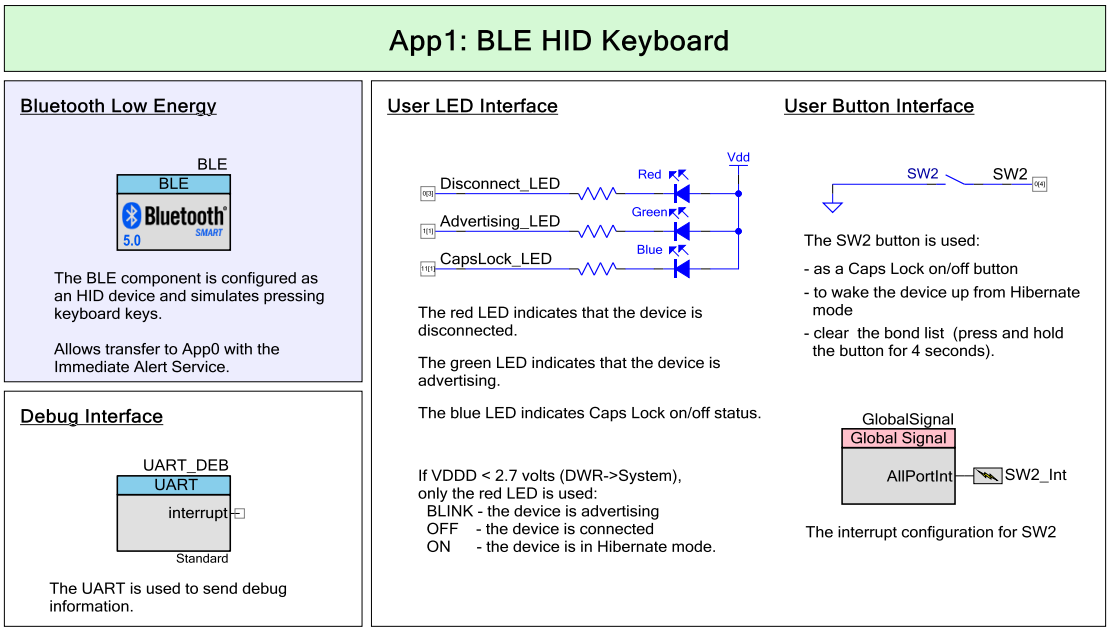


Figure 5. App1 Schematic



Design Firmware

The firmware portion of the design is implemented in the files listed in Table 1. Many of these files require custom settings in both the file and the related PSoC Creator projects. For more information on customizing DFU projects, see AN213924, *PSoC 6 MCU Device Firmware Update Software Development Kit Guide*.

Table 1. Design Firmware Files

File	Description
main_cm4.c *main_cm0p.c*	Contains the `main()` function for each CPU. Some PSoC 6 MCUs have two CPUs, an Arm Cortex-M4 (CM4) and a Cortex-M0+ (CM0+). See Table 5 on page 11 for specific tasks for each CPU.
ias.h / .c	BLE IAS files. Used to implement IAS between BLE GATT client and server. When the IAS receives a non-zero value, it transfers control to the other application.
debug.h / .c	LED status notification.
smif_mem.h / .c	Contains functions for initializing the SMIF Component, and read/write to the external memory.
cy_smif_memconfig.h / .c	Contains configuration parameters for the QSPI memory device (S25FL512S) on the kit board.
cy_dfu.h / .c	The DFU software development kit (SDK) files.
cy_dfu_bwc_macro.h	Contains macros for backward compatibility to facilitate porting of legacy bootloader projects.
dfu_user.h	Contains user-editable `#define` statements that control the operation and enable features in the SDK.
dfu_user.c	Contains user functions required by the SDK: ▪ Five functions that control communications with the DFU host. These are also called transport functions. ▪ Two functions – `ReadData()` and `WriteData()` – that control access to the PSoC 6 MCU device flash or external memory. Contains `EraseExternalApp()`. Erases the proper sectors of the external memory where the application is located. Contains the redefined function `Cy_DFU_ValidateApp()` to add support for external memory.
transport_ble.h / .c	Contains DFU transport functions for host communications using the BLE Component. These functions are typically called by the transport functions in *dfu_user.c.*
dfu_cm4.ld, *dfu_cm0p.ld*	Custom GCC linker scripts. In each application, these files replace the auto-generated linker script files. These files locate the code and data sections for each CPU core as well as other sections. These files include a “common” section that must be the same in both files.
dfu_mdk_common.h *dfu_mdk_symbols.c*	These files exist have defines needed by the MDK *.scat* files. These files are user-editable – they control the memory layout and locations in the memory for each application, and the code and data for each CPU core in each application. These files are common to all applications.
dfu_cm4.scat, *dfu_cm0p.scat*	Custom MDK linker scripts. In each application, these files replace the auto-generated linker script files. These files locate the code and data sections for each CPU core as well as the DFU code and other regions.
dfu_cm4.icf, *dfu_cm0p.icf*	Custom IAR linker scripts. In each application, these files replace the auto-generated linker script files. These files locate the code and data sections for each CPU core as well as the DFU code and other regions. These files include a “common” section that must be the same in both files.
post_build_core1.bat	Batch file to create the downloadable application image (*.cyacd2* file) for App1.

Memory Layout

Figure 6 shows the memory usage as the application is downloaded, stored, and copied. The DFU host sends an application (App1) to App0. App0 erases the application in the external memory when the DFU operation starts. As the application is downloaded, write operations are redirected to the external memory.

After the application has been completely downloaded, App0 verifies it in the external memory. If it is valid, it is copied into the proper memory space in the PSoC 6 MCU device flash, overwriting the previous application.

This method allows the application stored in the PSoC 6 MCU device flash to remain valid during the DFU process. If a corrupted application is received, the application in the device flash is not affected and can still be executed.

The downloaded application persists in the external memory after the copy operation. It may be used as a backup, or erased to use the memory for other purposes.

Figure 6. Application Storing Process

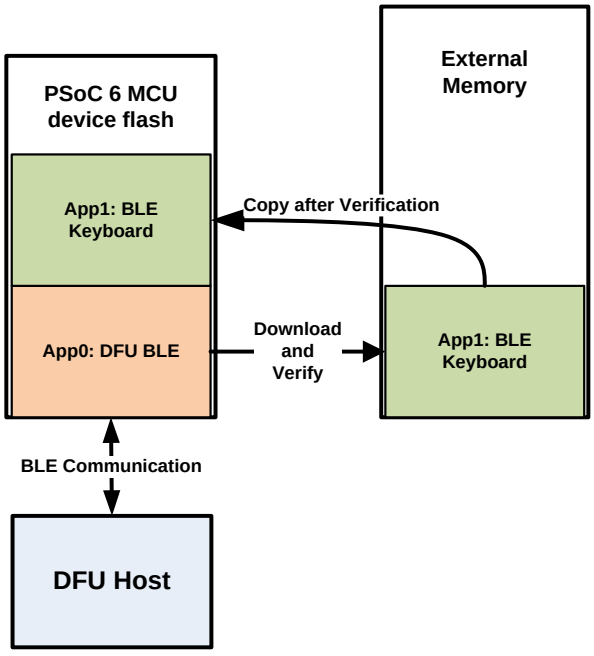


Figure 7 shows the PSoC 6 MCU device memory usage for each CPU in each application. This layout is for PSoC 6 MCU devices with 1 MB flash and 288 KB SRAM.

Approximately 170 KB of flash is required for the BLE API. Note that in App0, the BLE stack is executed by the Cortex-M4 (CM4) CPU; in App1, the BLE stack is executed by the Cortex-M0+ (CM0+) CPU. For more information on the memory layout, see AN213924, *PSoC 6 MCU Device Firmware Update Software Development Kit Guide*.

App0 always starts at the beginning of the device flash at address 0x1000 0000. For more information on the device memory map, see the device datasheet. App1 starts at the next 256 KB boundary. Each application has defined flash areas for each CPU core: core0 (CM0+) and core1 (CM4).

The RAM is shared by App0 and App1, with a common area used by both applications. Each application has defined RAM areas for each CPU core: core0 (CM0+) and core1 (CM4).

To change the memory layout or usage, update the linker script files shown in Table 1 on page 8. The linker scripts can also be modified to define dedicated regions of memory for each application.

Figure 7. Memory Layout of Applications

RAM	0x0804 7FFF Empty 0x0801 0000	224 KB
	ram, core1 0x0800 8000	32 KB
	ram, core0 0x0800 0100	31.75 KB
	ram_common 0x0800 0000	256 B

Flash	Metadata copy row 0x100F FC00	512 B
	Metadata flash row 0x100F FA00	512 B
	Empty / reserved 0x1008 0000	511.5 KB
	App1, core1 0x1007 0000	64 KB
	App1, core0 0x1004 0000	192 KB
	App0, core1 0x1001 0000	192 KB
	App0, core0 0x1000 0000	64 KB

LED Status

App0 indicates the current DFU state on the kit's RGB LED, as Table 2 shows.

Table 2. LED Status for App0

Color	State	Description
White	Blinks once every 2 seconds	DFU system is advertising
None	OFF	DFU system is connected but not receiving an application
White	Blinks twice every 2 seconds	An application is being received
White	Blinking fast	An application is being copied from external memory into PSoC 6 MCU device flash
Red	Steady	Hibernating

Note: If the specified V_{DDD} within **Design Wide Resources** > **System** is less than 2.7 V, only the RGB red LED is used.

At startup, App0 shows the status of App1 in the PSoC 6 MCU device flash and external memory on the kit RGB red LED, as Table 3 shows.

This warning may be disabled by modifying a `#define` statement in App0 *main_cm4.c*.

An application must be downloaded if no valid App1 is present in either the PSoC 6 MCU device flash or external memory.

Table 3. Boot Up Warning Sequences

App1 Warning	LED Sequence
Valid in both memories	None
Invalid in external memory	Blinks once red
Invalid in PSoC 6 MCU device flash	Blinks twice red
Invalid both memories	Blinks three times red

If a valid App1 is present only in the external memory, you don't need to download it again. The application may be copied into the PSoC 6 MCU device flash by pressing the user button for more than 0.5 seconds; the application starts afterwards. This situation may arise if a problem occurs during the copy operation (such as a power loss) or if the device is reprogrammed.

The application in the external memory may become invalid if the download operation fails or if you use the external memory for other purposes. In both situations, the application in the PSoC 6 MCU device flash remains valid. Keeping App1 on the external memory as a backup or erasing it to free the memory is at your discretion.

App1 indicates the BLE keyboard status on the kit's RGB LED, as Table 4 shows. The LED blinks green indicating that the device is advertising. After the BLE keyboard connects to a device, the LED indicates the status of the Caps Lock key. If the device remains disconnected for 150 seconds, the LED turns red and the device hibernates.

Note: If the specified V_{DDD} within the **Design Wide Resources** > **System** is less than 2.7 V, only the red LED is used. Caps Lock status is not shown.

Table 4. LED Status for App1

Color	State	Description
Green	Blinking	BLE Keyboard is advertising
None	OFF	BLE Keyboard is connected
Blue	Steady	Caps Lock is ON
Red	Steady	Hibernating

Design Considerations

Note: App0 and App1 projects must be built with the same toolchain (GCC or MDK); application transfer may fail otherwise. Check the **Build Settings** for each project.

Dual CPU

PSoC 6 MCU has two CPU cores: Cortex-M4 and Cortex-M0+. An application can include code for one or both CPUs. For more information, see AN215656, *PSoC 6 MCU Dual-CPU System Design*.

The CPUs in each application do as Table 5 shows. For details, see Appendix A, Code Theory of Operation. This can easily be changed so that either CPU can run any of the tasks, including DFU.

Table 5. CPU Tasks in Each Application

Application	Cortex-M0+	Cortex-M4
App0	Executes first at device reset. The reset handler controls the application transfer. Note that application transfer occurs before the Cortex-M4 CPU is turned ON. Turns ON Cortex-M4. Does nothing else.	Blinks an LED once per two seconds. Downloads and installs App1. Monitors the button. After the DFU operation, or when the IAS alert level is greater than zero, initiates transfer to App1, through a software reset.
App1	Demonstrates BLE services, as documented in CE215121, BLE HID Keyboard. If IAS alert level is greater than zero, switches to App0 through a software reset.	Does nothing.

Software Reset

When transferring control from one application to another, the recommended method is through a device software reset. This enables each application to initialize device hardware blocks and signal routing from a known state.

You can freeze the state of I/O pins so that they are maintained through a software reset. Defined portions of the SRAM can also be maintained through a software reset. For more information, see the PSoC 6 MCU: PSoC 63 with BLE Architecture Technical Reference Manual.

DFU Modifications

The default DFU linker scripts allocate an equal amount of flash and RAM to each CPU core in each application. In this code example, these allocations are modified because of the size of the BLE stack, and to minimize the size of App1 to reduce downloading time. The modifications were done to linker script files for both App0 and App1.

The metadata row defined in *dfu_user.c* is extended to include a virtual App2. The virtual App2 contains the metadata information of the received application. After the application has been completely received and copied into the PSoC 6 MCU device flash, the App1 metadata information is updated with the virtual App2 metadata. The App1 linker scripts have '`__cy_app_id`' set to 2, to use the virtual App2 metadata. For more information on the usage of the application metadata, see Appendix D of AN213924, PSoC 6 MCU DFU SDK Guide.

The default `Cy_DFU_WriteData()` and `Cy_DFU_ReadData()` functions in *dfu_user.c* are modified to add support for accessing the QSPI external memory. When writing to the first row of the external memory, `Cy_DFU_WriteData()` calls `EraseExternalApp()` to erase the needed sectors before they are written. The default `Cy_DFU_ValidateApp()` function in *cy_dfu.c* is redefined for validating applications in external memory.

To add support for external memory to your DFU system, copy *dfu_user.c* functions along with the file *dfu_user.h* and the SMIF files (*cy_smif_memconfig.h /.c* and *smif_mem.h / .c*) to your project. In the downloadable application, make sure that '`__cy_app_id`' is set to '2'. If a different memory interface, memory device, or kit is used, see the Reusing This Example section.

Components and Settings

Table 6 lists the PSoC Creator Components used in this example for App0, how they are used in the design, and the non-default settings required so they function as intended. For information on the Components used in App1, see CE215121.

Table 6. PSoC Creator Components

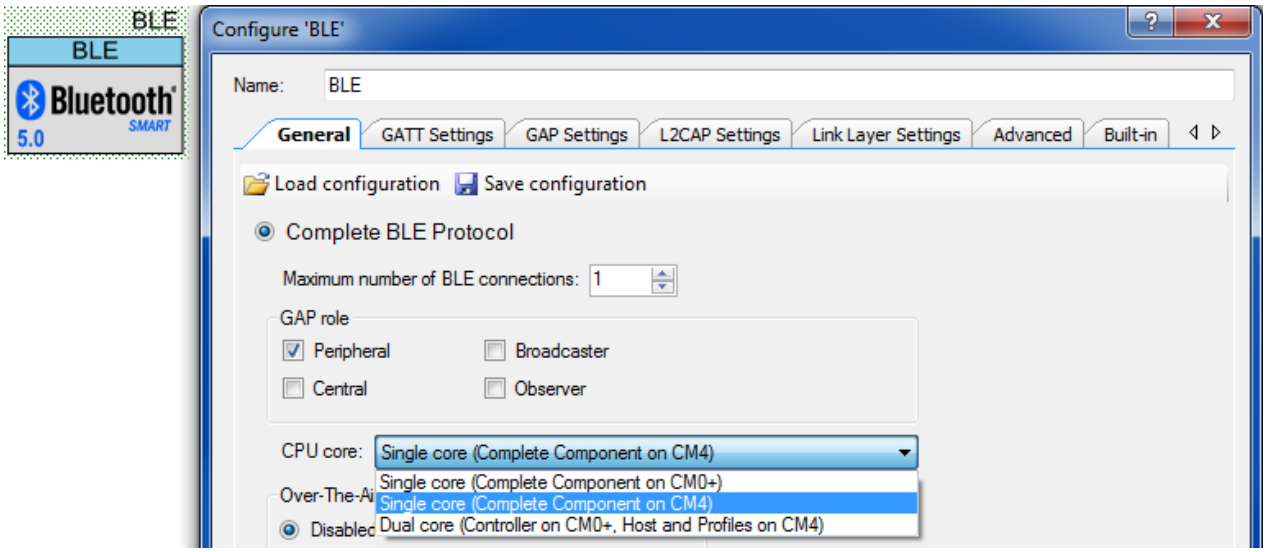
Component	Instance Name	Purpose	Non-default Settings
Bluetooth Low Energy	BLE	Provides communication between the PSoC 6 MCU device and the BLE host.	See BLE Component Configuration.
Serial Memory Interface (SMIF)	SMIF	Supports communication to external QSPI memory, to store the application.	SMIF dataline [2:3]: Checked. Generate code from cysmif file: Unchecked.
UART	UART_DEB	Debug UART for App1	(none)
Pin	PIN_LED_RED	Status notification on LEDs	No HW connection; External terminal; Initial drive state High (1); Max frequency 1 MHz
	PIN_LED_GREEN		
	PIN_LED_BLUE		
	PIN_SW2	Read button state	No HW connection; External terminal; Drive mode Resistive Pull Up; Max frequency 1 MHz

For information on the hardware resources used by a Component, see the Component datasheet.

BLE Component Configuration

- **General** tab (see Figure 8): The CPU core is set to **CPU core: Single core (Complete Component on CM4)** for App0, **CPU core: Single core (Complete Component on CM0+)** for App1.

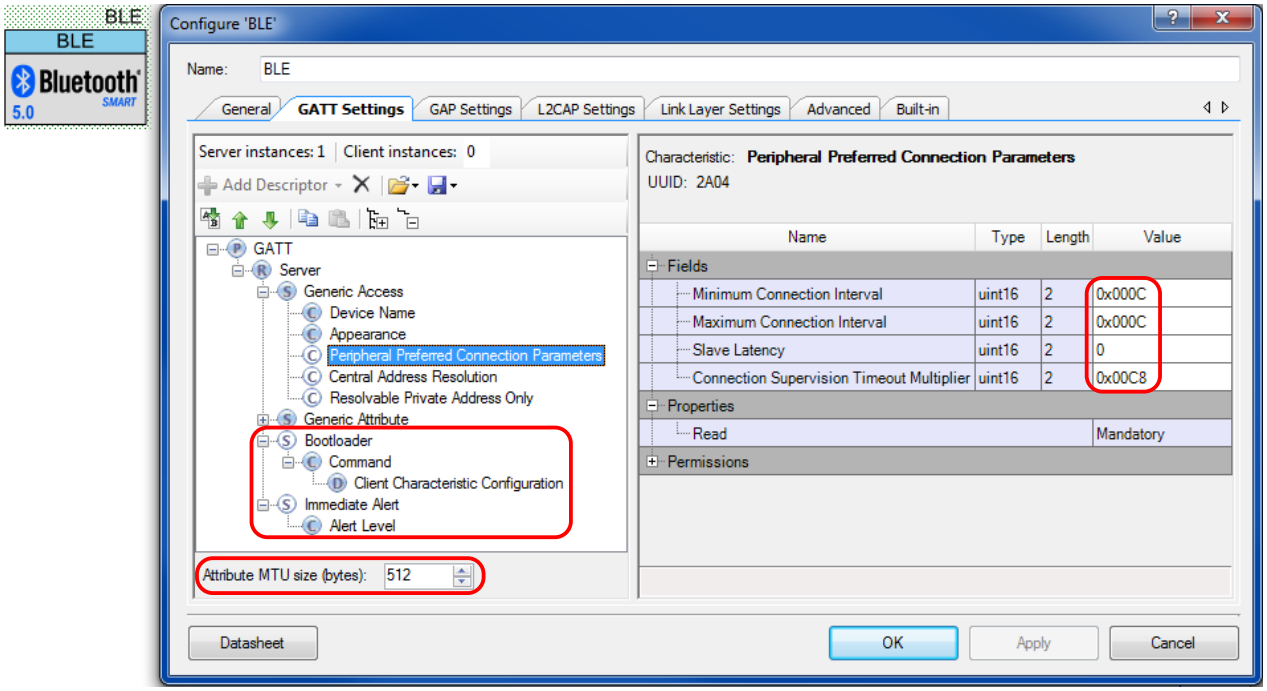
Figure 8. BLE Component, General Tab Configuration



- **GATT Settings** tab (see Figure 9):
  - Generic Access, Peripheral Preferred Connection Parameters:
    - **Minimum Connection Interval**: 0x000C
    - **Maximum Connection Interval**: 0x000C
    - **Connection Supervision Timeout Multiplier**: 0x00C8

    The above intervals are selected to minimize DFU time.
  - Add **Bootloader** service
  - Add **Immediate Alert** service
  - **Attribute MTU size (bytes)**: 512

Figure 9. BLE Component, GATT Settings Tab Configuration



- **GAP Settings** tab:
  - Device Name: “BLE DFU Device”
  - Peripheral Configuration 0, Advertisement packet: Local Name checked and set to Complete
  - Security configuration 0, Security level: Unauthenticated pairing with encryption
  - Security configuration 0, I/O capabilities: No Input No Output
  - Security configuration 0, Bonding requirement: No Bonding
- **Link Layer Settings** tab:
  - Link layer max TX and RX payload size (bytes): 251

Design-Wide Resources

Figure 10 and Figure 11 shows the pin assignments for the PSoC 6 BLE Pioneer Kit, for the SMIF, debug UART, LEDs, and user button.

Figure 10. Pin Assignments for PSoC 6 BLE Pioneer Kit for DFU App0

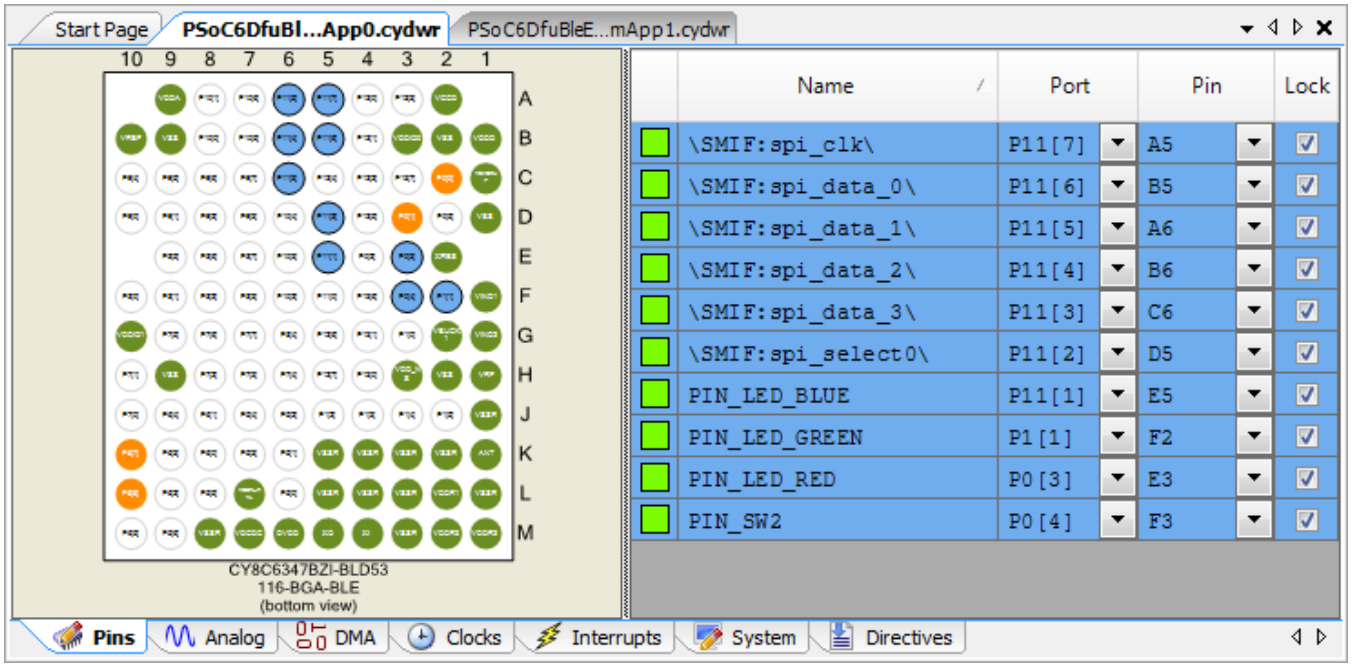
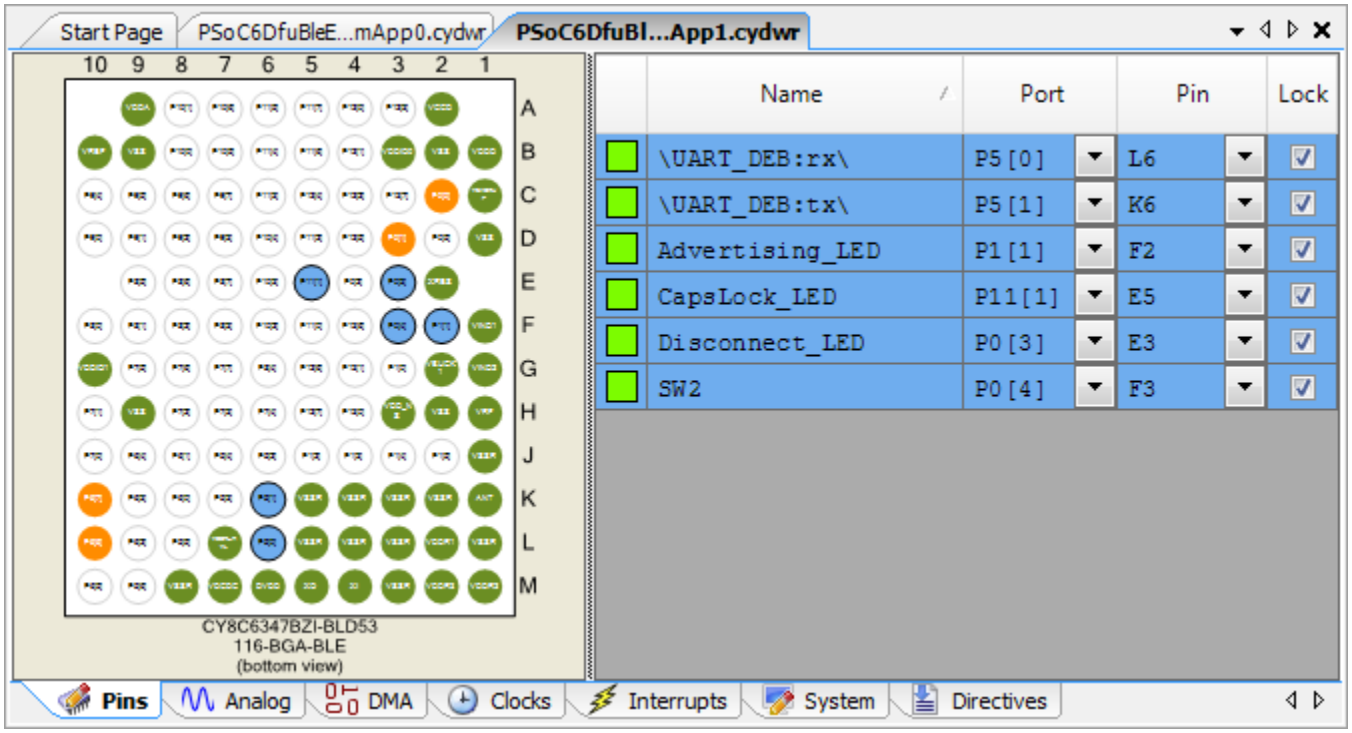


Figure 11. Pin Assignments for PSoC 6 BLE Pioneer Kit for App1



Reusing This Example

This example is designed for the kit indicated in Related Hardware. To port the design to a different PSoC 6 MCU device, kit, or both, change the target device using the Device Selector and update the pin assignments in the Design Wide Resources Pins settings as needed.

To use a different dual/quad/octal SPI external memory device, select the required datalines and check the **Generate code from cy_smif.cysmif file** option in the SMIF Component configuration tool. Generate new memory configuration files (*cy_smif_memconfig.h* / *.c*) using the SMIF configuration tool. See the SMIF Component datasheet for more information. The variable '`SectorSize`' in the `EraseExternalApp()` function in *dfu_user.c* must be updated. The functions in the file *smif_mem.c* may require modification depending on the addressing and initialization of the memory device.

For different memory interfaces, such as SPI and I^{2}C, delete the SMIF Component from the schematic and the files within the SMIF folder. Add the desired interface Component to the schematic. The functions `ReadMemory()`, `WriteMemory()`, `EraseExternalApp()`, and `Cy_DFU_ValidateApp()`in *dfu_user.c* must be updated to use the new memory interface.

The code to handle external memory for DFU purposes is located in *dfu_user.h* / *.c*. This code can be copied into other DFU systems to add external memory functionality. For more information on the DFU SDK files, see AN213924, PSoC 6 MCU DFU SDK Guide.

For single-CPU PSoC 6 MCU devices, port the code from *main_cm4.c* to *main.c,* and copy the `Cy_OnResetUser` function from *main_cm0p.c* to *main.c*. For App1, note in Table 5 on page 11 that the CM0+ performs all the tasks; port all the code from *main_cm0p* to *main.c*.

In some cases, a resource used by a code example is not supported on another device. In that case, the example will not work. If you build the code targeted at such a device, you will get errors. See the device datasheet for information on which resources a device supports.

Related Documents

PSoC 6 DFU-Related Application Notes	
AN213924 – PSoC 6 MCU Device Firmware Update Software Development Kit Guide	Provides comprehensive information on how to use the Device Firmware Update (DFU) Software Development Kit (SDK)
Other Application Notes	
AN221774 – Getting Started with PSoC 6 MCU	Describes PSoC 6 MCU devices and how to build your first ModusToolbox or PSoC Creator project
AN210781 – Getting Started with PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity	Describes PSoC 6 MCU with BLE Connectivity devices and how to build your first PSoC Creator project.
AN215656 – PSoC 6 MCU: Dual-CPU System Design	Describes the dual-CPU architecture in PSoC 6 MCU, and shows how to build a simple dual-CPU design
PSoC 6 MCU DFU-Related Code Examples	
CE213903 – PSoC 6 MCU Basic DFU	Single-application DFU using UART, I^{2}C, or SPI
CE216767 – PSoC 6 MCU Bluetooth Low Energy (BLE) Device Firmware Update (DFU)	Describes a BLE DFU system for PSoC 6 MCU
CE220960 – PSoC 6 MCU BLE DFU with Upgradeable Stack	Similar to the BLE DFU; the BLE stack can be updated in addition to the application
CE221984 – PSoC 6 MCU Dual-Application I2C DFU	Similar to the basic I^{2}C DFU; manages two downloaded applications instead of one
CE222802 – PSoC 6 MCU Encrypted DFU	Similar to the basic UART DFU; the application is digitally signed and encrypted

PSoC Creator Component Datasheets	
BLE	Supports the BLE subsystem
UART	Supports the serial communication block in UART mode
Pins	Supports connection of hardware resources to physical pins
Device Documentation	
PSoC 6 MCU Datasheets	PSoC 6 MCU Architecture Technical Reference Manuals
Development Kits	
CY8CKIT-062-BLE	PSoC 6 BLE Pioneer Kit
CY8CKIT-062-WiFi-BT	PSoC 6 WiFi-BT Pioneer Kit
CY8CPROTO-063-BLE	PSoC 6 BLE Prototyping Kit
CY8CPROTO-062-4343W	PSoC 6 Wi-Fi Prototyping Kit
Tool Documentation	
PSoC Creator	PSoC Creator enables concurrent hardware and firmware editing, compiling and debugging of PSoC devices. Applications are created using schematic capture and over 150 pre-verified, production-ready peripheral Components. Look in the downloads tab for Quick Start and User Guides.
ModusToolbox	ModusToolbox simplifies development for IoT designers. It delivers easy-to-use tools and a familiar microcontroller (MCU) integrated development environment (IDE) for Windows, macOS, and Linux.
Peripheral Driver Library (PDL)	Installed by PSoC Creator 4.2. Look in the <PDL install folder>/doc for the User Guide and the API Reference

Cypress Resources

Cypress provides a wealth of data at www.cypress.com to help you to select the right device, and quickly and effectively integrate the device into your design.

For the PSoC 6 MCU devices, see KBA223067 in the Cypress community for a comprehensive list of PSoC 6 MCU resources.

Appendix A: Code Theory of Operation

This section describes in detail how the code example source code implements the functions listed in Table 5 on page 11. App1 is described in detail in CE215121 BLE HID Keyboard.

File: *main_cm0p.c*:

Function `main()`:

Wakes up from device Hibernate mode by calling functions in the power management (`SysPm`) API.

Calls `Cy_SysEnableCM4((uint32_t)(&__cy_app_core1_start_addr))`

`__cy_app_core1_start_addr` is defined in *dfu_cm0p.ld*.

Then, goes into device Deep Sleep mode, by calling a function in the `SysPm` API.

Function `Cy_OnResetUser()`:

Called by the startup reset handler. Calls `Cy_DFU_OnResetApp0()`, which is defined in *cy_dfu.c*. This is the mechanism by which control is transferred to another application after device software reset.

File: *main_cm4.c*:

Has `#defines` for BLE parameters, and for verifying an application in either external memory or PSoC 6 MCU device flash ("internal").

Function `main()`:

Has local variables:

```
const uint32_t paramsTimeout = 20u; /* timeout, in milliseconds */
cy_stc_dfu_params_t dfuParams; /* configures DFU */
cy_en_dfu_status_t status; /* Status codes from DFU SDK API */
bool buttonReleased = false; /* user button management */
uint32_t state; /* NONE, UPDATING, FINISHED, or FAILED */

uint32_t count = 0; /* counts seconds */
uint32_t ledTimer = 0; /* for pulsing the LEDs */

CY_ALIGN(4) static uint8_t buffer[CY_DFU_SIZEOF_DATA_BUFFER]; /* flash row data */
CY_ALIGN(4) static uint8_t packet[CY_DFU_SIZEOF_CMD_BUFFER]; /* host packet */
```

Initializes LEDs.

Configures the SMIF Component for accessing external memory.

Initializes `dfuParams` with timeout, and two buffer addresses. Calls `Cy_DFU_Init()` (in *cy_dfu.c*), which sets the `state` to NONE.

Calls `HandleMetadata()`, which is part of the code example, not the SDK. It updates metadata (MD) and MD copy rows of flash, or initializes the MD row.

Calls `CopyRow()`, which is part of the code example, not the SDK. Reads a source row and writes it to a destination row, all in PSoC 6 MCU device flash. Does a compare before writing, to avoid an unnecessary row write.

Tests application validity in the external memory and PSoC 6 MCU device flash ("internal" memory). Flashes an LED the corresponding number of times, see Table 3 on page 10.

If the reset reason (`Cy_SysLib_GetResetReason()`, *cy_syslib.c*) was NOT a software reset (SRES), and the kit user button has NOT been pressed for two seconds (`IsButtonPressed()`, in this file):

Validates App1 in PSoC 6 MCU device flash (`Cy_DFU_ValidateApp(VERIFY_INT_APP)`, *cy_dfu.c*). If OK, clears the reset reason and transfers control to App1 (`Cy_DFU_ExecuteApp(1u)`, *cy_dfu.c*). This function does an SRES and does not return.

Initializes the host communication channel (`Cy_DFU_TransportStart()`, *dfu_user.c*), which in turn calls `CyBLE_CyBtldrCommStart()`, *transport_ble.c*.

Initializes BLE Immediate Alert Service (`IasInit()`, *ias.c*, which is part of this project). This function registers the `IasEventHandler()` function, also in *ias.c*, for BLE IAS events.

Main loop:

Calls `Cy_BLE_ProcessEvents()`. This may result in a call to `AppCallBack()`.

Calls `Cy_DFU_Continue()` (*cy_dfu.c*), which, depending on the state, may read one command packet from the host, process the command, and write one response packet to the host. This function has calls to functions in *dfu_user.c*. This function may set the state to UPDATING or FINISHED.

Increments the '`count`' variable, for timing purposes.

If FINISHED, validates App1 in external memory and, if success, copies it to the PSoC 6 MCU device flash (`CopyApp()`, in this file). Then, stops the host communication (`Cy_DFU_TransportStop()`, *dfu_user.c*) and transfers control to App1 (SRES; no return). If validation fails, then resets the host communication and turns OFF LEDs, and restarts DFU by calling `Cy_DFU_Init()`. User error handling can be placed here.

Else if still UPDATING, checks for 5-second timeout. If so, resets the host communication and restarts DFU.

Manages the LEDs per Table 2 on page 10.

Checks the button and the IAS alert level, and transfers control to App1 accordingly. Validation is done first for the PSoC 6 MCU device flash; if valid, then control is transferred. Otherwise, validation is done for the external memory; if valid, then the application is copied to the PSoC 6 MCU device flash and control is transferred.

Function `CopyAPP()`:

Establishes source and destination addresses in the external memory and PSoC 6 MCU device flash, respectively. Calls `Cy_DFU_ReadData()` and `Cy_DFU_WriteData()`, both in *dfu_user.c*, to transfer data row by row. Updates App1 metadata.

Function `AppCallBack()`:

Handles BLE events for BLE communication starts, device connected and disconnected, advertisement state change, and authorization and encryption.

If an advertising stopped event occurs, it is due to either download complete or advertising timeout. In this event, the host communication is stopped. Control is transferred to App1 in PSoC 6 MCU device flash if it is valid. Otherwise, the device enters hibernate mode with the corresponding LED display.

File: *transport_ble.c*:

This file implements DFU host communication over BLE. It is auto-generated from the PDL.

Function `CyBLE_CyBtldrCommStart()`:

Registers the `AppCallBack()` function in *main_cm4.c*, for general BLE GAP and GATT events, and the `DFUCallBack()` function in *transport_ble.c*, for BLE Bootloader Service (BTS) events.

Function `CyBLE_CyBtldrCommStop()`:

Initiates a GAP disconnect, and waits for the disconnect event. Then stops the BLE Component.

Function `CyBLE_CyBtldrCommWrite()`:

Sends a BTSS notification to send data to the DFU host.

Function `CyBLE_CyBtldrCommRead()`:

Processes BLE events while waiting for the required number of bytes to be received from the host. Processing BLE events results in a call to the callback function `DFUCallBack()`.

Function `DFUCallBack()`:

Handles BTSS write events by reading data from the DFU host.

File: *ias..h, c*:

This file implements the BLE Immediate Alert Service (IAS), to transfer control to the downloaded application.

Function `IasInit()`:

Registers the `IasEventHandler()` function for BLE IAS events.

Function `IasEventHandler()`:

Handles IAS write events by updating the global variable ‘alertLevel', which is read in main_cm4.c main loop.

File: *dfu_user.h*:

Note that the macro `CY_DFU_MAX_APPS` is set to 3, to accommodate the app stored in external memory.

File: *dfu_user.c*:

This file implements transport function redirects to *transport_ble.c*.

It also provides functions to override the DFU SDK weak functions `Cy_DFU_WriteData()` and `Cy_DFU_ReadData()`. These functions implement flash and external memory updates reads respectively. They call SMIF functions `ReadMemory()` and `WriteMemory()`.

Function `EraseExternalApp()`:

Erases that portion of external memory that contains the downloaded application. It calls `EraseSMIFSector()` in *smif_mem.c*.

Function `Cy_DFU_ValidateApp()`:

Overrides the DFU SDK weak function. Gets the start address and size of the application from the DFU metadata, for a given application ID.

If the address is in the external memory range, calls `SwitchSMIFMemory()` in *smif_mem.c* to map external memory into the CPU address range.

Validates the address range by doing a checksum on the range (`Cy_DFU_DataChecksum()` in the DFU SDK), reading the checksum from the application footer, and comparing the two for equality.

If the address is in the external memory range, calls `SwitchSMIFNormal()` in *smif_mem.c to* un-map external memory from the CPU address range.

File: *smif_mem.h, c*:

This file contains functions to access the external memory, for read, write, erase, and control operations.

Document History

Document Title: CE220959 – PSoC 6 MCU Bluetooth Low Energy (BLE) Device Firmware Update (DFU) with External Memory

Document Number: 002-20959

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	6002398	CFMM	02/23/2018	New code example
*A	6508755	MKEA	04/30/2019	Updated projects for PDL 3.1.0. Changed "bootloader" to "DFU".

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 MCU

Cypress Developer Community

Community Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

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

© Cypress Semiconductor Corporation, 2018-2019. This document is the property of Cypress Semiconductor Corporation and its subsidiaries ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.
TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress shall have no liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. CYPRESS DOES NOT REPRESENT, WARRANT, OR GUARANTEE THAT CYPRESS PRODUCTS, OR SYSTEMS CREATED USING CYPRESS PRODUCTS, WILL BE FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (collectively, "Security Breach"). Cypress disclaims any liability relating to any Security Breach, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from any Security Breach. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. "High-Risk Device" means any device or system whose failure could cause personal injury, death, or property damage. Examples of High-Risk Devices are weapons, nuclear installations, surgical implants, and other medical devices. "Critical Component" means any component of a High-Risk Device whose failure to perform can be reasonably expected to cause, directly or indirectly, the failure of the High-Risk Device, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from any use of a Cypress product as a Critical Component in a High-Risk Device. You shall indemnify and hold Cypress, its directors, officers, employees, agents, affiliates, distributors, and assigns harmless from and against all claims, costs, damages, and expenses, arising out of any claim, including claims for product liability, personal injury or death, or property damage arising from any use of a Cypress product as a Critical Component in a High-Risk Device. Cypress products are not intended or authorized for use as a Critical Component in any High-Risk Device except to the limited extent that (i) Cypress's published data sheet for the product explicitly states Cypress has qualified the product for use in a specific High-Risk Device, or (ii) Cypress has given you advance written authorization to use the product as a Critical Component in the specific High-Risk Device and you have signed a separate indemnification agreement.
Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.