CE221984 – PSoC 6 MCU Dual-Application Device Firmware Update (DFU)

Objective

This example demonstrates PSoC® 6 MCU device firmware update (DFU), also known as “bootloading”, with two applications. Either application can be downloaded from a host and installed in the device flash. The DFU system then transfers control to one of the applications, in either basic mode or factory default (“golden image”) mode.

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 6 MCU 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 the 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

For both applications, this example demonstrates several basic DFU operations:

- Downloading an application from a host, using the PSoC Creator I2C communication Component for host communication
- Installing the downloaded application into flash, and validating and transferring control to that downloaded application
- After device reset, managing two downloaded applications. That is, validating both applications and transferring control to one them, in either of two modes:
  - Basic mode: the first application (App1) is preferred
  - Factory default mode: the second application (App2) is preferred

In factory default mode, the DFU system (App0) does not overwrite an installed and valid App1. An attempt to do so results in an error message. In basic mode, either application can be overwritten. In either mode, after downloading the application, control is transferred to the application that was just downloaded.

This code example uses I^2C as a communication channel. Using PSoC Creator and the DFU Software Development Kit (SDK), it is easy to change the example to another communication channel such as UART, SPI, or Bluetooth Low Energy (BLE). These channels 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. For more information, see the KitProg2 User Guide.

Software Setup

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

To enable the basic mode of operation in this code example, set the macro `CY_DFU_OPT_GOLDEN_IMAGE` to zero (the default setting in *dfu_user.h*). To enable the factory default (“golden image”) mode of operation, set the macro to a non-zero value.

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, App1, and App2 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*:

```
/*
* 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 = 1;
```

   - 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*:

```
__cy_app_core1_start_addr   EQU __cpp(CY_APP1_CORE1_FLASH_ADDR)

/* Application number (ID) */
__cy_app_id                 EQU 1

/* 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

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 blue LED blinks 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. Run PSoC Creator Bootloader Host Program (BHP). Select PSoC Creator menu item **Tools** > **Bootloader Host...**.

   Configure BHP for the KitProg2 USB-I^{2}C bridge connection. See Figure 2 on page 4 for I^{2}C address and bit rate settings. For more information on using BHP, see BHP Help.
5. Using BHP, download App1, which is typically found in:

   <project file> \ PSoC6DfuDualApp1.cydsn \ CortexM4 \ ARM_GCC_541 \ Debug \ PSoC6DfuDualApp1.cyacd2

   Note: This file contains code for both CPUs in PSoC 6 MCU. See AN215656, PSoC 6 Dual-CPU System Design.

   During downloading, the kit RGB blue LED blinks at a rapid rate. After download is complete, confirm that the kit LED8 turns ON, indicating that App1 has been validated and is running.
6. Press the kit button SW2 for at least half a second, and then release it. Confirm that the kit RGB LED blinks blue once every two seconds, indicating that control has been transferred back to App0.
7. Using BHP, download App2. After download is complete, confirm that kit LED8 and LED9 turn ON, indicating that App2 has been validated and is running. Close BHP.
8. Press the kit button SW2 for at least half a second, and then release it. Confirm that the kit RGB LED blinks blue once per two seconds, indicating that control has been transferred back to App0.
9. Repeat steps 4 – 8 to confirm that App1 and App2 can be overwritten. Control that control is always transferred to the that most recently downloaded application.
10. Press the kit button SW2 for at least half a second, and then release it. Confirm that the kit LED8 turns ON, indicating that App1 has been validated and is running. This demonstrates the basic mode of operation, where App1 is the preferred application to which to transfer control.
11. Edit *dfu_user.h* in App0 to set the macro `CY_DFU_OPT_GOLDEN_IMAGE` to '1'. Rebuild App0. Then repeat steps 2 – 8.
12. Press the kit button SW2 for at least half a second, and then release it. Confirm that the kit LED8 and LED9 turn ON, indicating that App2 has been validated and is running. This demonstrates the factory default mode of operation, where App2 is the preferred application to which to transfer control.
13. Press the kit button SW2 for at least half a second, and then release it. Confirm that the kit RGB LED blinks blue once per two seconds, indicating that control has been transferred back to App0.
14. Repeat Step 5. Confirm that BHP displays an error and the kit RGB red LED turns ON for five seconds. This indicates an attempt to overwrite App1 failed because it is not allowed in this mode.
15. Repeat Step 7 to confirm that App2 can be overwritten.

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 2 on page 7 shows the tasks that are done by each of the CPUs. To debug App1 or App2, 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 uses the multiple-application architecture that is enabled by the Cypress DFU Software Development Kit (SDK). The SDK is part of the Peripheral Driver Library (PDL) provided with PSoC Creator. For more information on the DFU SDK, see the SDK Guide, AN213924.

This example has three applications, called “App0”, “App1”, and “App2”, as Figure 1 shows. Each application is a separate PSoC Creator project; all projects are in the same PSoC Creator workspace. The applications have the following features:

- App0 does the DFU operation; it downloads, installs, and transfers control to either App1 or App2.
- App0 blinks the kit RGB blue LED once every two seconds. App1 turns ON one kit LED, and App2 turns ON two kit LEDs. This makes it easy to see which application is currently running.
- All projects monitor the kit button SW2. If the button is pressed for more than half a second and then released, the currently running application transfers control to one of the other applications.

Figure 1. PSoC 6 MCU Dual-Application DFU

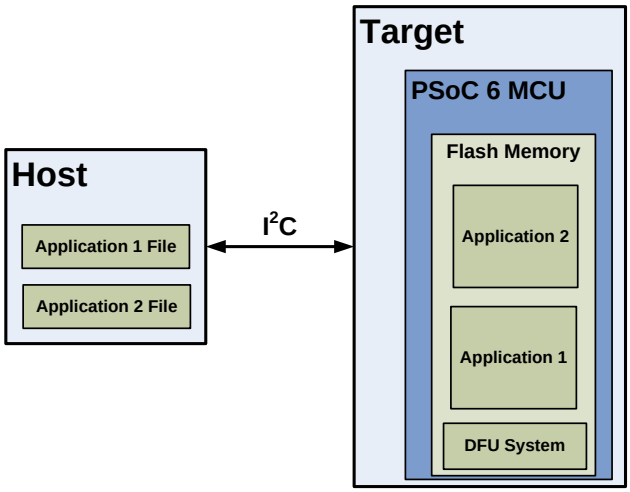


Figure 2 shows the PSoC Creator project schematic for App0. Figure 3 on page 5 shows the PSoC Creator project schematic for both App1 and App2.

Figure 2. PSoC Creator Schematic for App0

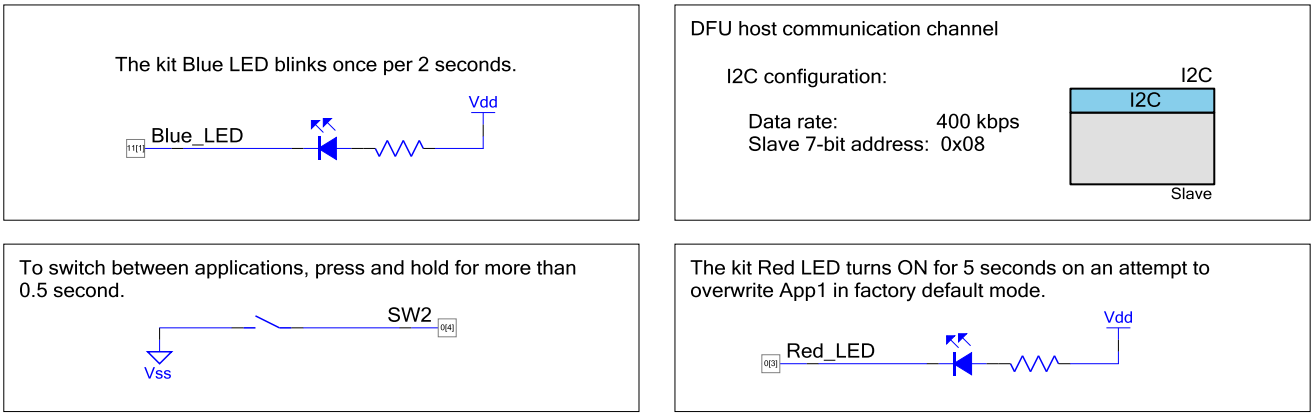
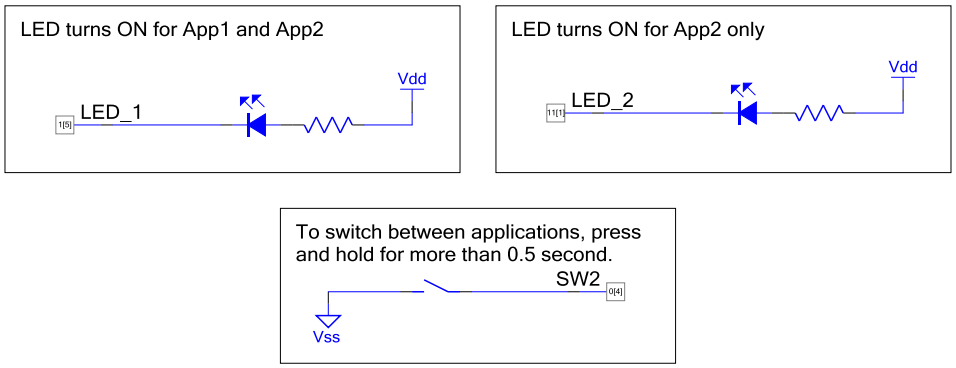


Figure 3. PSoC Creator Schematic for App1 and App2



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 core. PSoC 6 MCU has two CPUs: an Arm Cortex-M4 (CM4) and a Cortex-M0+ (CM0+). See Table 2 on page 7 for specific tasks for each core.
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 enabled 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
transport_i2c.h, .c	Contains transport functions for the host communications Component being used. 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 the locations in 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 images (*.cyacd2* files) for App1 and App2.

Memory Layout

Figure 4 shows the typical memory usage for each CPU in each application. This layout is for the DFU App0 using I²C as the communication channel in PSoC 6 MCU devices with 1 MB flash and 288 KB SRAM. Other DFU channels such as BLE have different layouts because the BLE API is much larger than the I²C API.

Note that App0 always starts at the beginning of the device user 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 5. The linker scripts can also be modified to define dedicated regions of memory for each application.

Figure 4. Memory Usage of App0, App1, and App2

APPLICATION FLASH

Address	Description	Size
0x100F FC00	Metadata copy row	512 B
0x100F FA00	Metadata row	512 B
0x100A 0000	Unused	383 KB
0x1009 0000	App2, CM4	64 KB
0x1008 0000	App2, CM0+	64 KB
0x1006 0000	Unused	128 KB
0x1005 0000	App1, CM4	64 KB
0x1004 0000	App1, CM0+	64 KB
0x1002 0000	Unused	128 KB
0x1001 0000	App0, CM4	64 KB
0x1000 0000	App0, CM0+	64 KB

RAM

Address	App0, App1, and App2	Size
0x0804 7FFF 0x0800 A000	Unused	248 KB
0x0800 2000	CM4 Application Data	32 KB
0x0800 0100	CM0+ Application Data	7.75 KB
0x0800 0000	Common RAM	256 B

Design Considerations

Note: All three projects – App0, App1, and App2 – 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 2 shows. For details, see Appendix A, Code Theory of Operation. This can easily be changed so that either core can run any of the tasks, including DFU.

Table 2. CPU Tasks in Each Application

Application	Cortex-M0+	Cortex-M4
App0	Executes first at device reset. The reset handler controls application transfer. Note that application transfer occurs before the Cortex-M4 is turned ON. Turns ON Cortex-M4. Does nothing else.	Blinks the RGB blue LED once per two seconds. Downloads and installs App1 or App2. Monitors the button. After the DFU operation, or when button pressed, initiates transfer of control to App1 or App2 based on mode setting, with software reset.
App1	Executes first, then turns ON CM4. Does nothing else.	Turns ON kit LED8. Monitors the button. When it is pressed, initiates transfer of control to App0, with software reset.
App2	Executes first, then turns ON CM4. Does nothing else.	Turns ON kit LED8 and LED9. Monitors the button, and when pressed, initiates transfer of control to App0, with software reset.

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.

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

Components and Settings

Table 3 lists the PSoC Creator Components used in this example, how they are used in the design, and the non-default settings required so they function as intended.

Table 3. PSoC Creator Components

Component	Instance Name	Purpose	Non-default Settings
I2C	I2C	Host communication	Data rate 400 kbps. Use TX FIFO and Use RX FIFO are checked.
Pin	LED	Drive an LED	No HW connection; External terminal; Initial drive state High (1); Max frequency 1 MHz
	SW2	Read button state	No HW connection; External terminal; Drive mode Resistive Pull Up; Max frequency 1 MHz

Design-Wide Resources

Figure 5 and Figure 6 shows the pin assignments for the PSoC 6 BLE Pioneer Kit, for the I²C, LEDs, and user button.

Figure 5. Pin Assignments for PSoC 6 BLE Pioneer Kit for I²C DFU App0

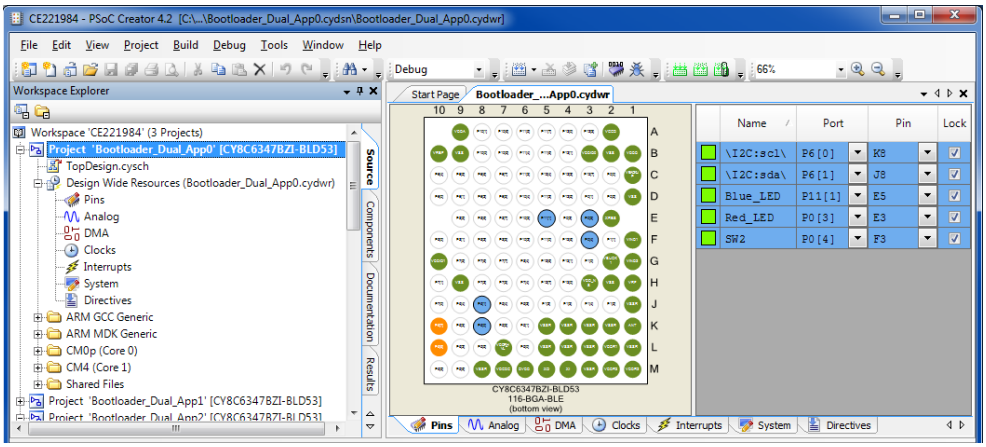
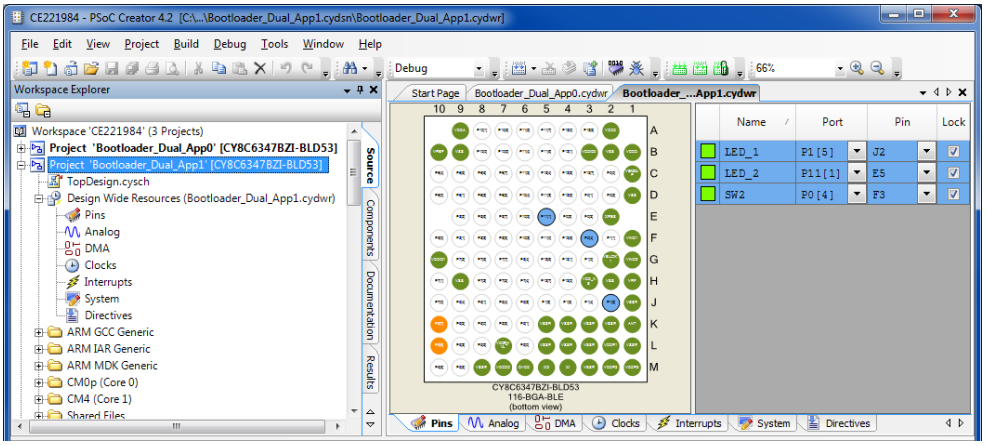


Figure 6. Pin Assignments for PSoC 6 BLE Pioneer Kit for App1 and App2



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.

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 and App2, note in Table 2 on page 7 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.

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.

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 DFU-Related Code Examples	
CE213903 – PSoC 6 MCU Basic DFU	Single-application DFU using UART, I^2C, or SPI
CE216767 – PSoC 6 MCU Bluetooth Low Energy (BLE) Device Firmware Update (DFU)	Describes a BLE DFU system for PSoC 6 MCU
CE220959 – PSoC 6 MCU BLE DFU with External Memory	Similar to the BLE DFU; the downloaded application is temporarily saved in external memory and then copied to its final destination
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
CE222802 – PSoC 6 MCU Encrypted DFU	Similar to the basic UART DFU; the application is digitally signed and encrypted
PSoC Creator Component Datasheets	
I2C	Supports the serial communication block in I^2C mode
Pins	Supports connection of hardware resources to physical pins
Device Documentation	
PSoC 6 MCU Datasheets	PSoC 6 MCU Architecture Technical Reference Manuals
Development Kit	
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
Development Tools	
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

Appendix A: Code Theory of Operation

This section describes in detail how the code example source code implements the functions listed in Table 2 on page 7. Due to their simplicity, the App1 and App2 projects are not described.

File: *main_cm0p.c*:

Function `main()`:

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

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

Then does nothing – empty for loop.

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 a device software reset.

File: *main_cm4.c*:

Has #defines for LED and button.

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 */
uint32_t state; /* NONE, UPDATING, FINISHED, or FAILED */
uint32_t count = 0; /* counts seconds */
const uint32_t PreferredAppID = 1u; /* app IDs are reversed in factory default mode */
const uint32_t NextAppID = 2u;
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 `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 the 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. Does a compare before writing, to avoid an unnecessary row write.

If the reset reason (`Cy_SysLib_GetResetReason()`, *cy_syslib.c*) was NOT a software reset (SRES), calls `LoadValidApp()`, which is part of the code example, not the SDK.

Initializes the host communication channel (`Cy_DFU_TransportStart()`, *dfu_user.c*).

Main loop:

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. May set the state to UPDATING or FINISHED.

If FINISHED, validates the downloaded application and, if success, stops host communication (`Cy_DFU_TransportStop()`, *dfu_user.c*) and transfers control to that application (SRES; no return). If validation fails, then resets the host communication and restarts DFU by calling `Cy_DFU_Init()`. User error handling can be placed here.

Else if FAILED, does the same as above.

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

Otherwise assumes that an attempt was made to overwrite a golden image. Flashes the kit RGB red LED for 5 seconds, and then resets the host communication and restarts the DFU.

If the 300-second timeout and state is NONE, calls `LoadValidApp()`. If the function returns then no valid application is installed, and `Cy_SysLib_Halt()` is called. User error handling can be placed here.

If 2-second timeout, inverts the kit RGB blue LED, for 2-second blinking.

If the kit button is pressed, waits for the button release and calls `LoadValidApp()`. If the function returns, then no valid application is installed, and the button press is ignored.

Function `LoadValidApp()`:

Receives a "preferred" and a "next" application to which to transfer control.

Validates the "preferred" application and, if success, stops the host communication (`Cy_DFU_TransportStop()`, *dfu_user.c*) and transfers control to that application (SRES; no return).

If validation fails, then validates the "next" application and, if success, stops the host communication (`Cy_DFU_TransportStop()`, *dfu_user.c*) and transfers control to that application (SRES; no return).

Otherwise returns with no return parameter. This is an indication to the caller that no valid application is installed.

Document History

Document Title: CE221984 – PSoC 6 MCU Dual-Application Device Firmware Update (DFU)

Document Number: 002-21984

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	6072561	MKEA	02/20/2018	New code example
*A	6545085	MKEA	04/22/2019	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 | 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.