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.

CE213903 – PSoC 6 MCU Basic Device Firmware Update (DFU)

Objective

These examples demonstrate basic device firmware update (DFU), also known as "bootloading", with PSoC® 6 MCU. This includes downloading an application from a host and installing it in device flash, and then transferring control to that application.

Requirements

Tool: ModusToolbox™ IDE 1.0

Programming Language: C

Associated Parts: All PSoC 6 MCU parts

Related Hardware: PSoC 6 BLE Pioneer Kit

Note: The PSoC 6 BLE Pioneer kit is shipped with KitProg2, and ModusToolbox only works with KitProg3. Before testing this code example, make sure that the kit is upgraded to KtiProg3. See ModusToolbox Help > ModusToolbox IDE Documentation > User Guide; section **PSoC 6 MCU KitProg Firmware Loader**.

Overview

These examples demonstrate several basic DFU operations:

- Communicating with a host, and downloading an application,
- Installing the downloaded application into flash or external memory
- Validating an application, and then transferring control to that application

There are two project types, generally called "App0" and "App1". There are three App0 projects, one for each communication channel: UART, I^{2}C, and SPI. The projects have the following features:

- App0 does the DFU operation; it downloads and installs App1 into device flash
- Each project blinks a kit LED at a different rate making it easy to see which one is currently running
- Pressing a kit button causes the project that is currently running to transfer control to the other project

Advanced communication channels such as BLE and USB 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. The KitProg3 system on the kit acts as a programmer for direct programming, a USB-UART bridge for UART-based DFU, a USB-I^{2}C bridge for I^{2}C-based DFU, and as a USB-SPI-based bridge for SPI DFU. For more information, see the KitProg3 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.

Operation

Build the Projects

Note: These instructions implement the UART-based DFU system. For I2C- and SPI-based DFU, see Reusing This Example.

1. In the ModusToolbox IDE, Quick Panel window, click **New Application**. Click **Dev/Eval Kit**, select the kit indicated in Related Hardware, and click **Next**.

2. In the Starter Application dialog, click **Browse**, and navigate to and select *<this code example folder>* \ *PSoC6DfuBasicApp0Uart* \ *modus.mk*. Confirm that the Name and description change accordingly, and click **Next**, followed by **Finish** in the next dialog.
3. Build the Application. Select the *PSoC6DfuApp0Uart_mainapp* folder, then on the Quick Panel click **Build PSoC6DfuApp0Uart Application**. Confirm that the application builds without errors.
4. Now, create the downloadable application using the same **New Application** process. Select the same kit selected in Step 1. Select *<this code example folder>* \ *PSoC6DfuBasicApp1* \ *modus.mk*.
5. After the application is created, select the *PSoC6DfuApp1_mainapp* project folder, then on the Quick Panel click **Project Build Settings**. Select **C/C++ Build** > **Settings** > **Build Steps**, and replace the **Post-build steps Command** entry with the following:

   ```
   ${cy_sdk_install_dir}/tools/modus-shell-1.0/bin/bash --norc --noprofile
   ${workspace_loc:/PSoC6DfuApp1_mainapp}/dfu_postbuild.bash
   ${cy_sdk_install_dir}/tools/cymcuelftool-1.0/bin/cymcuelftool
   ${workspace_loc:/PSoC6DfuApp1_mainapp_cm0p}/${config_name:PSoC6DfuApp1_mainapp_cm0p}/PSoC6DfuA
   pp1_mainapp_cm0p.elf
   ${workspace_loc:/PSoC6DfuApp1_mainapp}/${config_name:PSoC6DfuApp1_mainapp}/PSoC6DfuApp1_mainap
   p.elf ARM_CM4
   ```

   This creates the output *.cyacd2* file, which is used to download the DFU application. Click **Apply and Close**.
6. Build the application. Select the *PSoC6DfuApp1_mainapp* folder, then on the Quick Panel click **Build PSoC6DfuApp1 Application**. Confirm that the application builds without errors.

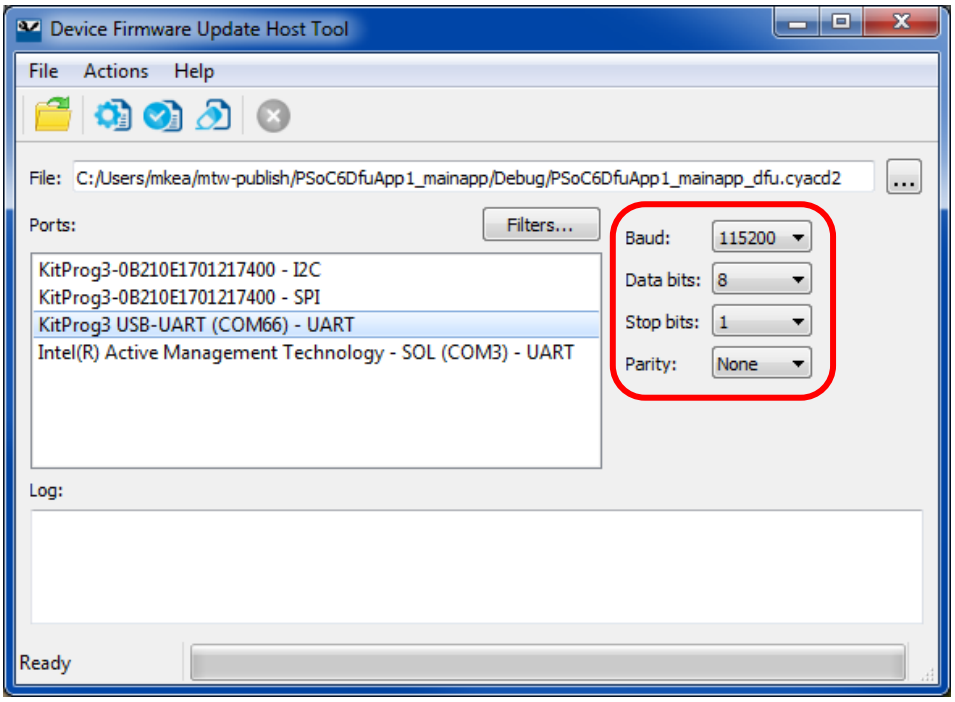
Test the Projects

1. Connect the kit board to your PC using the provided USB cable.
2. (Optional) To remove any artifacts from device flash that may affect DFU operation, select **Run** > **Run Configurations ...**. In the Run Configurations dialog, select **PSoC6DfuApp0Uart Erase (KitProg3)**. Click **Run**. This erases all of device flash.
3. Program the PSoC 6 MCU device with App0. In the Project Explorer window, select the **PSoC6DfuApp0Uart_mainapp** project. In the Quick Panel window, in the **Launches** section, click **PSoC6DfuApp0Uart Program (KitProg3)**. After programming is complete, confirm that the kit red LED toggles every 2 seconds. This indicates that the DFU project is running.

   Note: If you are reinstalling / overwriting App0, and App1 is already installed, you may see that control has been transferred to App1, and the LED is blinking at a different rate – see Step 6. Press SW2 to transfer control back to App0; see Step 7.
4. Press and hold the kit button for at least half a second, and then release it. Confirm that nothing happens because App0 is the only application installed.
5. Run the Device Firmware Update Host Tool: *dfuh-tool.exe* in the*...*\ *ModusToolbox_1.0* \ *tools* \ *dfuh-tool-1.0* folder. Click the open file icon and select the PSoC6DfuApp1 *.cyacd2* file. It's in the *...* \ *<workspace name>* \ *PSoC6DfuApp1_mainapp* \ *Debug* workspace folder.

6. Select and configure the correct COM port, as Figure 1 shows. Click the **Program** icon (**F5** hot key). Confirm that download is successful, and that the kit red LED blinks at a 2-Hz rate (twice per second), indicating that the downloadable installable application App1 is running.

Figure 1. Select and Configure COM Port in Device Firmware Update Host Tool



7. Press the SW2 kit button for at least 0.5 second, and release it. Confirm that the other application is now running; the kit LED blinks at a different frequency. Repeat.

Note: Make sure that App0 is running before attempting to use the Device Firmware Update Host Tool. App1 is not designed to do a DFU operation.

Note: In addition to a Program option, the Device Firmware Update Host Tool has a Verify option and an Erase All option. The Erase All option erases only App1; it does not erase App0.

Debugging

You can debug the example to step through the code. Use a **Debug (KitProg3)** configuration. See KBA224621 in the Cypress community to learn how to start a debug session with ModusToolbox IDE.

Design and Implementation

App0 blinks a red LED on CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit once every two seconds, and App1 blinks it twice every second, both using firmware delays. The LED blink frequency makes it easy to see which project is running.

Both projects monitor the kit button SW2. If the button is pressed for more than 0.5 second, then released, the currently running application transfers control to the other application. The blinking LED changes frequency as described previously.

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 related projects. For more information on customizing ModusToolbox DFU projects, see AN213924, PSoC 6 MCU Bootloader Software Development Kit (SDK) Guide.

Table 1. Design Firmware Files

File	Description
main.c (two instances)	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 for specific tasks for each core.
cy_dfu.h, *.c*	The DFU software development kit (SDK) files.
cy_dfu_bwc_macro.h	Facilitates 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 internal or external memory
transport_xxx.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_common.ld	GCC linker script. It is user-editable – it controls the memory layout and the locations in memory for each project, and the code and data for each CPU core in each project. This file is included in the custom GCC linker scripts described next. This file is common to all projects.
dfu_cm4.ld, *dfu_cm0p.ld*	Custom GCC linker scripts. In each project, these files replace the auto-generated linker script files. These files locate the code and data sections for each of the CPU cores as well as other sections. These files include the memory layout described in *dfu_common.ld*.
dfu_postbuild.bash	Script file to create the downloadable application image (*.cyacd2* file) for App1.

Memory Layout

Figure 2 shows the typical memory usage for each CPU core in each project. This layout is for 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 UART API.

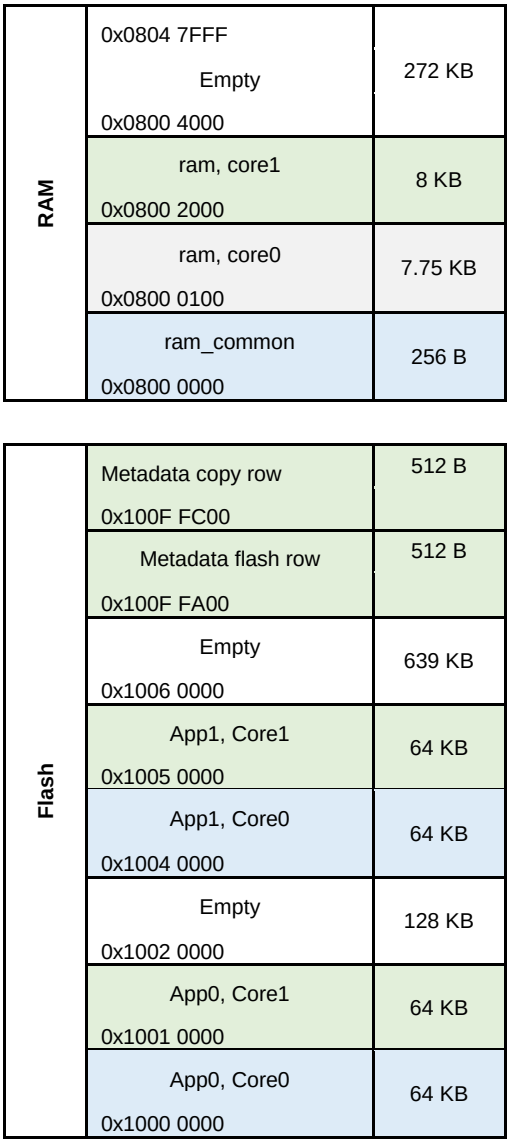
App0 always starts at the beginning of device user flash at the address 0x1000 0000. For more information on the device memory map, see the device datasheet.

App1 starts in the upper half of flash. This facilitates the flash read-while-write (RWW) operation. For more information on RWW, see the device datasheet.

The RAM is shared by App0 and App1, with a common area used by both projects. The linker scripts can be modified to define dedicated regions of memory for each project.

To change the memory layout or usage, update the linker script files shown in Table 1.

Figure 2. Memory Layout of Applications



Design Considerations

Dual CPU

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

In these projects, 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. Reset handler controls application transfer. Note that if application transfer does occur, it occurs before the Cortex-M4 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 button pressed, initiates transfer of control to App1, with software reset.
App1	Executes first, then turns ON Cortex-M4. Does nothing else.	Blinks an LED twice per second. Monitors the button. When button is 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.

Resources and Settings

Table 3 lists the ModusToolbox resources used in this example, and how they are used in the design. For pin usage and configuration, open the **Pins** tab of the *design.modus* file.

Table 3. ModusToolbox Resources

Resource	Alias	Purpose	Non-default Settings
SCB5	KIT_UART	Host communication	Oversample changed to 12. Pins P5[1:0] used.
SCB6	KIT_I2C	Host communication	None. Pins P6[1:0] used.
SCB6	KIT_SPI	Host communication	None. Pins P12[4, 2:0] used
Pin	KIT_RBG_R	Drive an LED	Strong Drive, Input buffer off
Pin	KIT_BTN1	Read button state	Resistive Pull-Up, Input buffer on

Reusing This Example

The Operation instructions implement UART-based DFU. To implement I²C- or SPI-based DFU, select *modus.mk* from the *...I2c* or *...Spi* folder instead of *...Uart*. After the application is created, make sure that the following edits are in the file *dfu_user.c*, which is in the folder *mainapp* > *Source*:

- Replace "transport_uart.h" with either "transport_i2c.h" or "transport_spi.h"
- Replace five instances of "UART_Uart" with either "I2C_I2c" or "SPI_Spi".

This example is designed for the kit indicated in Related Hardware. It is easily portable to the CY8CKIT-062 PSoC 6 Pioneer Kit, which has the same pin assignments for the LEDs, button, and communication channels as CY8CKIT-062-BLE. Change the device to CY8C6247BZI-D54.

To port this code example to a different platform or device, right-click the ..._mainapp project and click **Change ModusToolbox Platform...** or **Change ModusToolbox Device...**. If changing to a different platform, you may need to reassign pins.

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

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)
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
Device Documentation	
PSoC 6 MCU: PSoC 63 with BLE Datasheet	PSoC 6 MCU: PSoC 63 with BLE Architecture Technical Reference Manual
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	
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.

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 2 on page 6. The App0 UART project is described; the I²C and SPI projects are similar. Due to its simplicity, the App1 project is 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 bootload_cm0p.ld.

Then does nothing – empty for loop.

Function `Cy_OnResetUser()`;

Called by the startup reset handler. Calls `Cy_Bootload_OnResetApp0()`, which is defined in *cy_bootload.c*. This is the mechanism by which control is transferred to another application after device software reset.

File: *main_cm4.c*:

Has GPIO #defines for LED and button.

Function `main()`:

Has local variables:

```
const uint32_t paramsTimeout = 20u; /* timeout, in milliseconds */
cy_stc_bootload_params_t bootParams; /* configures bootloader */
cy_en_bootload_status_t status; /* Status codes from Bootloader SDK API */
uint32_t state; /* NONE, BOOTLOADING, FINISHED, or FAILED */
uint32_t count = 0; /* counts seconds */
CY_ALIGN(4) static uint8_t buffer[CY_BOOTLOAD_SIZEOF_DATA_BUFFER]; /* flash row data */
CY_ALIGN(4) static uint8_t packet[CY_BOOTLOAD_SIZEOF_CMD_BUFFER]; /* host packet */
```

Initializes `bootParams` with timeout, and two buffer addresses.

Calls `Cy_Bootload_Init()` (in *cy_bootload.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. 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), validates App1 (`Cy_Bootload_ValidateApp(1u)`, *cy_bootload.c*). If OK, clears the reset reason and transfers control to App1 (`Cy_Bootload_ExecuteApp(1u)`, *cy_bootload.c*). This function does an SRES and does not return.

Initializes host communication channel (`Cy_Bootload_TransportStart()`, *bootload_user.c*).

Main loop:

Calls `Cy_Bootload_Continue()` (*cy_bootload.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 BOOTLOADING or FINISHED.

If FINISHED, validates App1 and, if success, stops host communication (`Cy_Bootload_TransportStop()`, *bootload_user.c*) and transfers control to App1 (SRES; no return). If validation fails, then resets host communication and restarts bootloading by calling `Cy_Bootload_Init()`. User error handling can be placed here.

Else if FAILED, does the same as above.

Else if still BOOTLOADING, checks for 5-second timeout. If so, resets host communication and restarts bootloading.

If 300-second timeout and state is NONE, transfers control to App1 if it is valid, otherwise `Cy_SysLib_Halt()`, with the kit red LED ON.

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

If the kit button is pressed, wait for button release and transfer control to App1 if it is valid. Otherwise ignores the button press.

Document History

Document Title: CE213903 – PSoC 6 MCU Basic Device Firmware Update (DFU)

Document Number: 002-25604

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	6379280	MKEA	11/08/2017	New version of CE213903, updated for ModusToolbox IDE ES-100 revision. Term "bootload" changed 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

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