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: 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, with KitProg2 installed

Note: The PSoC 6 BLE Pioneer kit is shipped with KitProg2, and PSoC Creator only works with KitProg2. If your kit was upgraded to KitProg3 for use with ModusToolbox™, 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

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 PSoC Creator 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 KitProg2 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-based DFU. For more information, see the KitProg2 User Guide.

Software Setup

To customize the DFU operation and enable DFU software development kit (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 I^{2}C- and SPI-based DFU, see Reusing This Example.

Note: If you are using a version of the PDL that is different from that specified in the Requirements, 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*.

1. Using PSoC Creator, build the App0 project for the communication channel that you want to use: UART, I²C, or SPI. For more information on building projects, see PSoC Creator Help.

   Note: For the MDK compiler, there is a known issue documented in PDL 3.1.0 Release Notes. To handle this issue, 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. Build the App1 project. First, select **Build** > **Generate Application**. The installed linker script files are by default set up for App0. For the App1 project, edit the files to change the application number:

   - For the GCC compiler, the following example shows edits for App1 in *dfu_cm0p.ld* and *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* and *dfu_cm4.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)
     ```

   Then complete the project build by selecting **Build** > **Build <project name>**.

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

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 red LED on the kit blinks once every two seconds. This indicates that App0 is running.
3. 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.
4. Run PSoC Creator Bootloader Host Program (BHP). In PSoC Creator, select **Tools** > **Bootloader Host...**.

   If you are using the UART communication channel, establish a connection with your computer's COM port corresponding with the KitProg2 USB-UART bridge.

   Configure BHP for the KitProg2 USB-UART, USB-I^{2}C, or USB-SPI bridge connection, depending on the communication channel that you are using.

   For more information on using BHP, see BHP Help.
5. Using BHP, download App1. After download is complete, confirm that the red LED on the kit blinks twice per second, indicating that App1 is running.
6. Press the SW2 kit button for at least half a second, and release it. Confirm that the other application is now running; the kit LED blinks at a different frequency. Repeat.
7. Repeat the two previous steps to test the process of reinstalling App1.

Note: Make sure that App0 is running before attempting to use BHP. App1 is not designed to do a DFU operation.

Note: In addition to a 'Program' option, BHP 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. 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 most of the tasks are done by CM4, as Table 2 shows. 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.

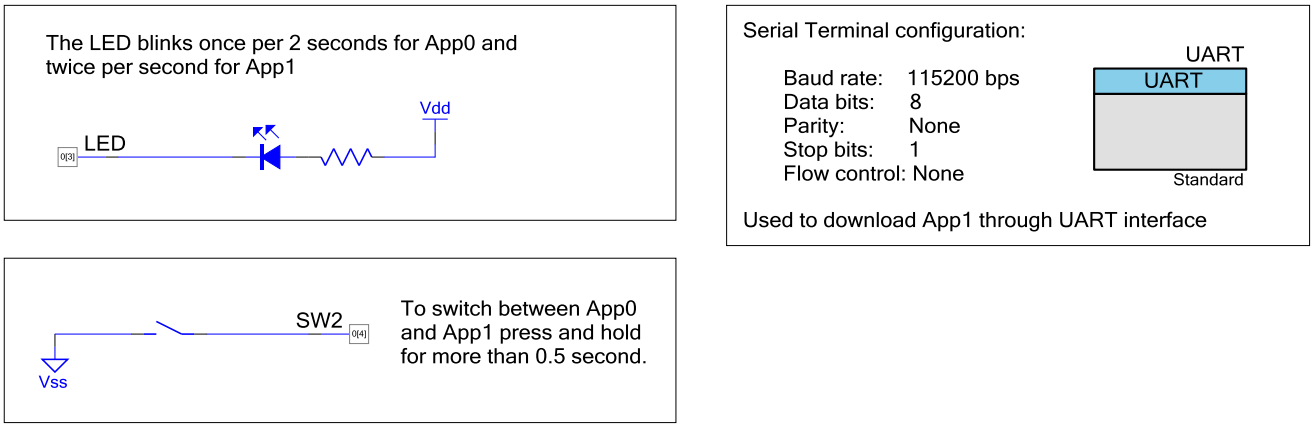
Design and Implementation

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

- App0 does the DFU operation; it downloads and installs App1.
- App0 blinks a red LED on 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 half a second and then released, the currently running application transfers control to the other application. The blinking LED changes frequency as described previously.

Figure 1 shows the PSoC Creator project schematic for both App0 and App1. App0 has the host communication Component; App1 does not.

Figure 1. PSoC Creator Schematic for App0 and App1, with UART as Host Communication Component



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 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 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 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_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 a “common” section that must be the same in both files.
dfu_mdk_common.h *dfu_mdk_symbols.c*	Similar in function to the GCC and IAR common linker scripts, for MDK. The MDK linker does not support includes in *.scat* files, so these files exist to create the necessary defines. 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 of the CPU cores 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 of the CPU cores 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 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 address 0x1000 0000. For more information on the device memory map, see the device datasheet. App1 starts at the next 256 KB boundary. Each app 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 projects. Each app 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. The linker scripts can also be modified to define dedicated regions of memory for each application.

Figure 2. Memory Layout of Applications

RAM	0x0804 7FFF Empty 0x0800 4000	272 KB
	ram, core1 0x0800 2000	8 KB
	ram, core0 0x0800 0100	7.75 KB
	ram_common 0x0800 0000	256 B

Flash	Metadata copy row 0x100F FC00	512 B
	Metadata flash row 0x100F FA00	512 B
	Empty / reserved 0x1006 0000	639 KB
	App1, core1 0x1005 0000	64 KB
	App1, core0 0x1004 0000	64 KB
	Empty 0x1002 0000	128 KB
	App0, core1 0x1001 0000	64 KB
	App0, core0 0x1000 0000	64 KB

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*.

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.

Components and Settings

Table 3 lists the PSoC Creator Components used in this example, the hardware resources used by each, and parameter settings that are changed from the default values.

Table 3. PSoC Creator Components

Component	Instance Name	Purpose	Non-default Settings
UART	UART	Host communication	Oversample changed from 12 to 8
I2C	I2C	Host communication	Data rate 400 kbps; Use TX FIFO; Use RX FIFO
SPI	SPI	Host communication	RX Data Width 8; TX Data Width 8
Pin	LED	Drive an LED	No HW connection; External terminal; Initial drive state High (1); Max frequency 1 MHz
Pin	SW2	Read button state	No HW connection; External terminal; Drive mode Resistive Pull Up; Max frequency 1 MHz

Design-Wide Resources

Figure 3 to Figure 5 show the pin assignments for the PSoC 6 BLE Pioneer Kit, for the UART, I2C, SPI, LED, and button.

Figure 3. Pin Assignments for the PSoC 6 BLE Pioneer Kit for UART DFU

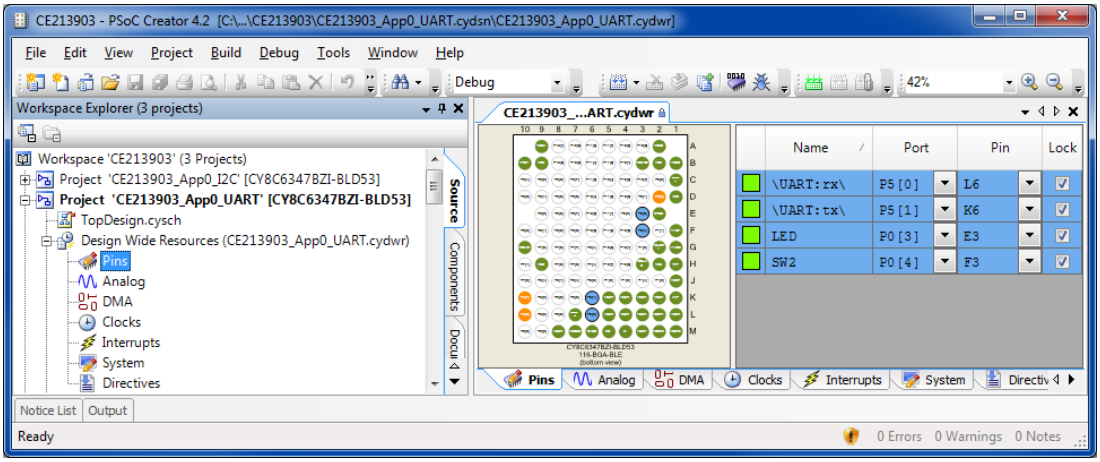


Figure 4. Pin Assignments for the PSoC 6 BLE Pioneer Kit for I²C DFU

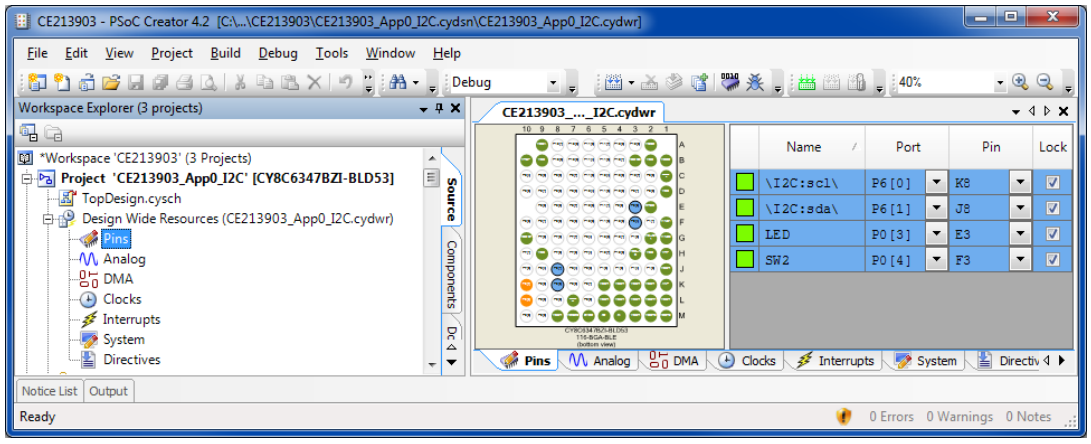
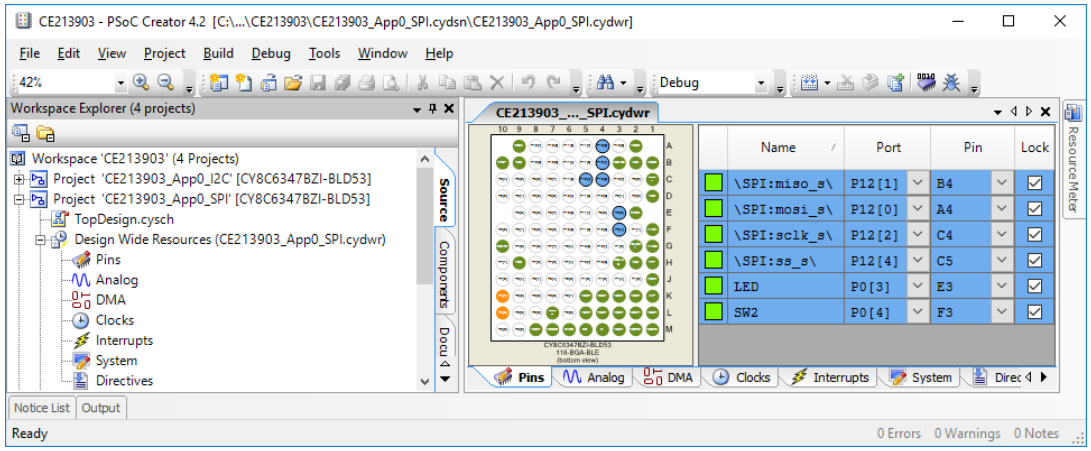


Figure 5. Pin Assignments for the PSoC 6 BLE Pioneer Kit for SPI DFU



Reusing This Example

The Operation instructions implement UART-based DFU. To implement I²C- or SPI-based DFU, first select **Build** > **Generate Application**, then edit *dfu_user.c* as follows:

- Replace the `#include`: "`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`".

Then complete the project build by selecting **Build** > **Build <project name>**.

This example is designed for the kit indicated in Related Hardware. To port the design to a different PSoC 6 MCU device and/or kit, change the target device using the Device Selector and update the pin assignments in the Design Wide Resources Pins settings as needed. For single-core PSoC 6 MCU devices, port the code from *main_cm4.c* 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
PSoC 6 DFU-Related Code Examples	
CE216767 – PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity Bootloader	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
CE221984 – PSoC 6 MCU Dual-Application I2C DFU	Similar to the basic I²C 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	
UART	Supports the serial communication block in UART mode
I2C	Supports the serial communication block in I²C mode
SPI	Supports the serial communication block in SPI mode
Pins	Supports connection of hardware resources to physical pins
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	
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 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 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 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_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 */
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 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_DFU_ValidateApp(1u)`, *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 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 App1 and, if success, stops host communication (`Cy_DFU_TransportStop()`, *dfu_user.c*) and transfers control to App1 (SRES; no return). If validation fails, then resets 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 host communication and restarts DFU.

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: 002-13903

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	5634506	MKEA	02/15/2017	New code example
*A	5654179	SHEA	03/08/2017	Updated logo and added the confidential disclaimer to the footer.
*B	5791097	MKEA	06/29/2017	Updated document and project for release versions of PSoC Creator 4.1 and PDL 3.0.0.
*C	5859753	MKEA	08/22/2017	Updated document and project for build versions of PSoC Creator 4.2 and PDL 3.0.1. Added support for I^2C bootloader. Added support for MDK compiler. Updated some links to other documents. Ported to new code example document template. Confidential tag removed.
*D	5933732	CFMM	10/27/2017	Updated document and project for Beta version of PSoC Creator 4.2 and PDL 3.0.1. Added support for SPI bootloader. Updated support for MDK and IAR compilers. Updated memory regions.
*E	6007229	MKEA	12/27/2017	Removed limitation that clocks between applications must be the same. Added information on manually editing bootloader files. Updated links in the Related Documents section. Added Appendix A, Code Theory of Operation. Updated projects for PSoC Creator 4.2 Beta 2. Updated document to latest code example template.
*F	6061562	MKEA	02/09/2018	Updated projects for Bootloader SDK 2.10. No document change.
*G	6457475	MKEA	01/22/2019	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, 2017-2019. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("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 does not assume any liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. 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. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, 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 or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

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.