CE221984 – PSoC 6 MCU Dual-Application Bootloader

Objective

This code example demonstrates a PSoC® 6 MCU Bootloader with two applications. Either application can be downloaded from a host through an I^{2}C communication channel and installed in device flash. The bootloader then transfers control to one of the applications, in either basic mode or factory default ("golden image") mode.

Overview

For both applications, this example demonstrates several basic bootloading 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 bootloader (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 bootloading control is transferred to the application that was just downloaded.

This code example uses I^{2}C as a communication channel. Using PSoC Creator and the Bootloader Software Development Kit (SDK), it is easy to change the example to another communication channel such as UART, SPI, Bluetooth Low Energy (BLE), or USB. These channels are demonstrated in other code examples; see Related Documents.

Requirements

Tool: PSoC Creator 4.2; Peripheral Driver Library (PDL) 3.0.1 with Bootloader SDK 2.10

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

Hardware Setup

This example uses the kit's default configuration. See the kit guide to ensure the kit is configured correctly. Connect the kit's USB port to your computer's USB port. The KitProg2 system on the kit acts as both a programmer for direct programming and as a USB-I^{2}C bridge for I^{2}C bootloading. For more information, see the KitProg2 User Guide.

Software Setup

This code example uses the Cypress Bootloader SDK. To customize bootload operation and enable Bootloader SDK features, update the `#define` statements as needed in the file *bootload_user.h*. For more information on the SDK, see the Bootloader SDK Guide, AN213924.

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

Operation

1. Plug the CY8CKIT-062-BLE kit board into your computer's USB port.
2. Using PSoC Creator, build all three code example projects: App0, App1, and App2. For more information on building projects, see PSoC Creator Help.

   Note: During a project build, you may be prompted by a popup dialog to replace bootloader files in a project with files from the PDL. The PDL files are templates. Do not replace the customized files that were shipped with the project. Click **Cancel** to close the dialog box and continue the build without replacing the files.
3. Program the App0 project into the PSoC 6 MCU device. Select PSoC Creator menu item **Debug** > **Program**. For more information on device programming, see PSoC Creator Help. Code for both CPUs is programmed in a single program operation.

   After programming is complete, confirm that the kit blue LED blinks once per 2 seconds. This indicates that App0 is running.
4. The kit button SW2 is used by all three applications as a command to transfer control to another application. Press and hold SW2 for at least half a second, and then release it. Confirm that the kit blue LED continues to blink, indicating that App0 is still running. Control cannot be transferred because App0 is the only application installed.
5. 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 for I^{2}C address and bit rate settings. For more information on using BHP, see BHP Help or the Bootloader SDK Guide, AN213924.
6. Using BHP, download App1, which is typically found in:

   <project file> \ Bootloader_Dual_App1.cydsn \ CortexM4 \ ARM_GCC_541 \ Debug \ Bootloader_Dual_App1.cyacd2

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

   During bootloading, the kit 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.
7. Press and hold the kit button SW2 for at least half a second, and then release it. Confirm that the kit blue LED blinks once every two seconds, indicating that App0 is running.
8. Using BHP, download App2. After download is complete, confirm that kit LED8 and LED 9 turn ON, indicating that App2 has been validated and is running. Close BHP.
9. Press and hold the kit button for at least half a second, and then release it. Confirm that the kit blue LED blinks once per two seconds, indicating that App0 is running.
10. Repeat steps 5 – 9 to confirm that App1 and App2 can be overwritten. Download App2 first (Step 8) to confirm that control is transferred to the application that was just downloaded.
11. Press and hold the kit button 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.
12. Edit *bootload_user.h* in App0 to set the macro `CY_BOOTLOAD_OPT_GOLDEN_IMAGE` to 1. Rebuild App0. Then repeat steps 3 – 9.
13. Press and hold the kit button 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.
14. Press and hold the kit button for at least half a second, and then release it. Confirm that the kit blue LED blinks once per two seconds, indicating that App0 is running.
15. Repeat Step 6. Confirm that BHP displays an error and the kit red LED turns ON for five seconds. This indicates an attempt to overwrite App1 failed because it is not allowed in this mode.
16. Repeat Step 8 to confirm that App2 can be overwritten.

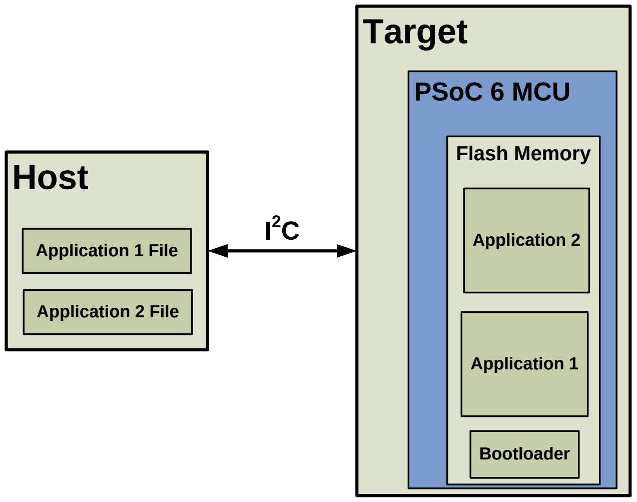
Design and Implementation

This code example uses the multiple-application architecture that is enabled by the Cypress Bootloader Software Development Kit (SDK). The SDK is part of the Peripheral Driver Library (PDL) provided with PSoC Creator. For more information on the Bootloader SDK, see the Bootloader SDK Guide, AN213924.

In this architecture, the PSoC 6 MCU stores multiple applications in separate sections of flash, as Figure 1 shows. The applications behave as follows:

- App0 does the bootloading; it downloads and installs App1 or App2. App0 executes first at device reset. It either transfers control to one of the other applications or waits for communication from the host.
- App0 blinks the kit 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.
- The kit button SW2 is used to transfer control between applications, as described in Operation.

Figure 1. PSoC 6 Dual-Application Bootloader System



Each application is a separate PSoC Creator project; all projects are in the same PSoC Creator workspace. Figure 2 shows the PSoC Creator project schematic for App0. Figure 3 shows the PSoC Creator project schematic for both App1 and App2.

Figure 2. PSoC Creator Schematic for App0

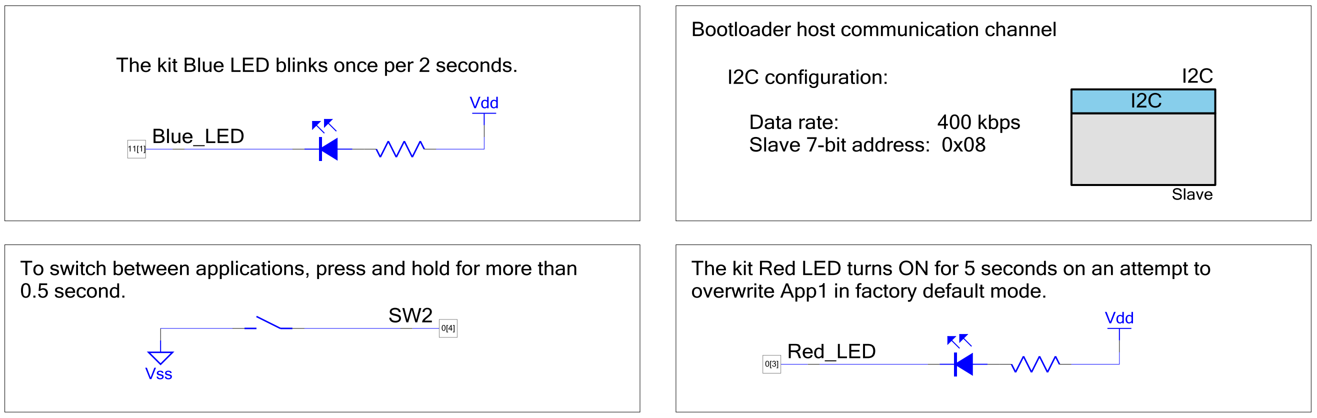
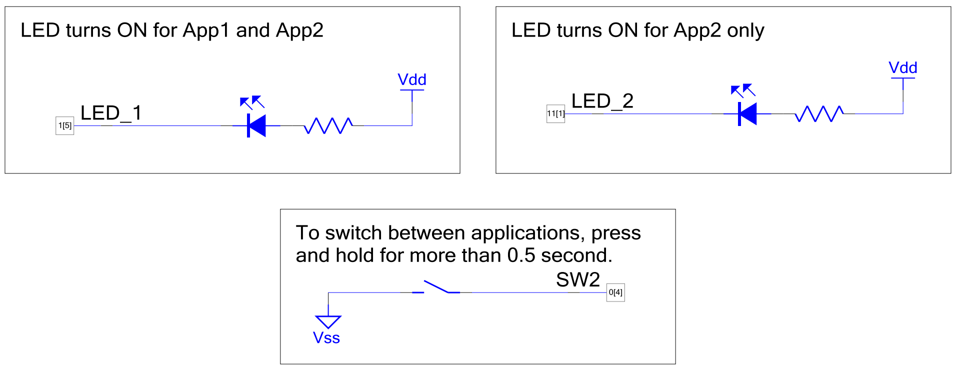

The kit Blue LED blinks once per 2 seconds.
Vdd
Blue_LED
11[1]
Bootloader host communication channel
I2C configuration:
Data rate: 400 kbps
Slave 7-bit address: 0x08
I2C
I2C
Slave
To switch between applications, press and hold for more than 0.5 second.
SW2
0[4]
Vss
The kit Red LED turns ON for 5 seconds on an attempt to overwrite App1 in factory default mode.
Vdd
Red_LED
0[3]

Figure 3. PSoC Creator Schematic for App1and App2


LED turns ON for App1 and App2
Vdd
LED_1
1[5]
LED turns ON for App2 only
Vdd
LED_2
11[1]
To switch between applications, press and hold for more than 0.5 second.
SW2
0[4]
Vss

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 PSoC Creator projects for the Bootloader SDK, see the Bootloader SDK Guide, AN213924.

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 an Arm Cortex-M0+ (CM0+). See Table 2 for specific tasks for each core.
cy_bootload.h, *.c*	The bootloader software development kit (SDK) files.
bootload_user.h	Contains user-editable `#define` statements that control the operation and enabled features in the SDK.
bootload_user.c	Contains user functions required by the SDK: • Five functions that control communications with the bootloader 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 bootloader transport functions for the PSoC Creator Component being used for host communications. These functions are typically called by the transport functions in *bootload_user.c*.
bootload_common.ld	GCC linker script. It is user-editable – it controls the memory layout and the locations in memory for each application, and the code and data for each CPU core in each application. This file is included in the custom GCC linker scripts described next. This file is common to all applications.
bootload_cm4.ld, *bootload_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 of the CPU cores as well as the bootloader and other regions. These files include the memory layout described in *bootload_common.ld*.
bootload_mdk_common.h *bootload_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.
bootload_cm4.scat, *bootload_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 bootloader and other regions.
bootload_common.icf	IAR linker script. It is user-editable – it controls the memory layout and the locations in memory for each application, and the code and data for each CPU core in each application. This file is included in the custom IAR linker scripts described below. This file is common to all applications.
bootload_cm4.icf, *bootload_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 bootloader and other regions. These files include the memory layout described in *bootload_common.icf*.
post_build_core1.bat	Batch file to create the downloadable application images for App1 and App2.

Memory Layout

Figure 4 shows the typical memory usage for each core in each application. This layout is for an I^{2}C bootloader and applications in PSoC 6 MCU devices with 1 MB flash and 288 KB SRAM. Other bootloaders such as BLE have different layouts because the BLE API is much larger than the I^{2}C API.

Note that 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.

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

Figure 4a. SRAM Memory Layout of Applications

SRAM	0x0804 7FFF Empty 0x0800 4810	~248 KB
	ram, core1 0x0800 2000	32 KB
	Core1 Vectors 0x0800 2000	1 KB
	ram, core0 0x0800 0100	8 KB
	Core0 Vectors 0x0800 0100	1 KB
	ram_common 0x0800 0000	256 B

Figure 4b. Flash Memory Layout of Applications

	Metadata copy row 0x100F FC00	512 B
	Metadata flash row 0x100F FA00	512 B
	Empty 0x100A 0000	383 KB
	App2, Core1 0x1009 0000	64 KB
	App2, Core0 0x1008 0000	64 KB
	Empty 0x1006 0000	128 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: All three projects – App0, App1, and App2 – must be built with the same toolchain (GCC or MDK), or application transfer may fail. Check the **Build Settings** for each project, for both Debug and Release builds.

Dual Core

PSoC 6 MCU has two CPU cores: An Arm Cortex-M4 (CM4) and a Cortex-M0+ (CM0+). An application can include code for one or both cores. For more information, see AN215656, PSoC 6 MCU Dual-Core CPU System Design.

In these examples, CPUs in each application do as Table 2 shows. This can easily be changed so that either core can run any of the tasks, including bootloading.

Table 2. CPU Tasks in Each Application

Application	Cortex-M0+	Cortex-M4
App0	Executes first at device reset. Reset handler controls application transfer. Turns ON CM4. Does nothing else	Blinks an LED once per two seconds Bootloads App1 or App2 Monitors the button After bootload or when button pressed, validates and 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 a PSoC 6 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.

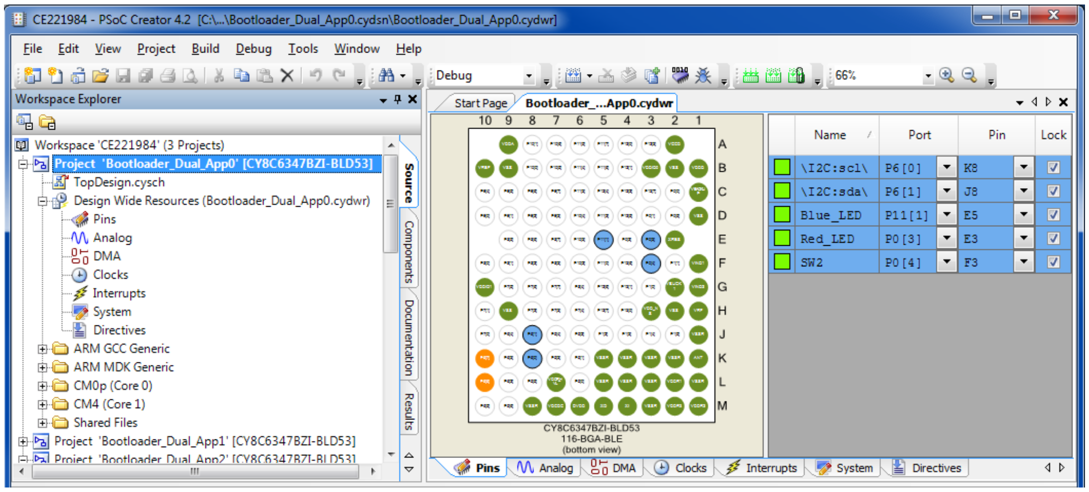
Table 3. PSoC Creator Components

Component	Instance Name	Purpose	Non-default Settings
SCB_I2C_PDL_v2_0	I2C	Bootloader-host communication	Data rate 400 kbps. Use TX FIFO and Use RX FIFO are checked.
GPIO_PDL_v1_0	LEDs	Drive kit LEDs	HW connection unchecked. External terminal checked.
	SW2	Read kit button state	HW connection unchecked. External terminal checked. Drive mode Resistive Pull Up Initial drive state High (1)

Design-Wide Resources

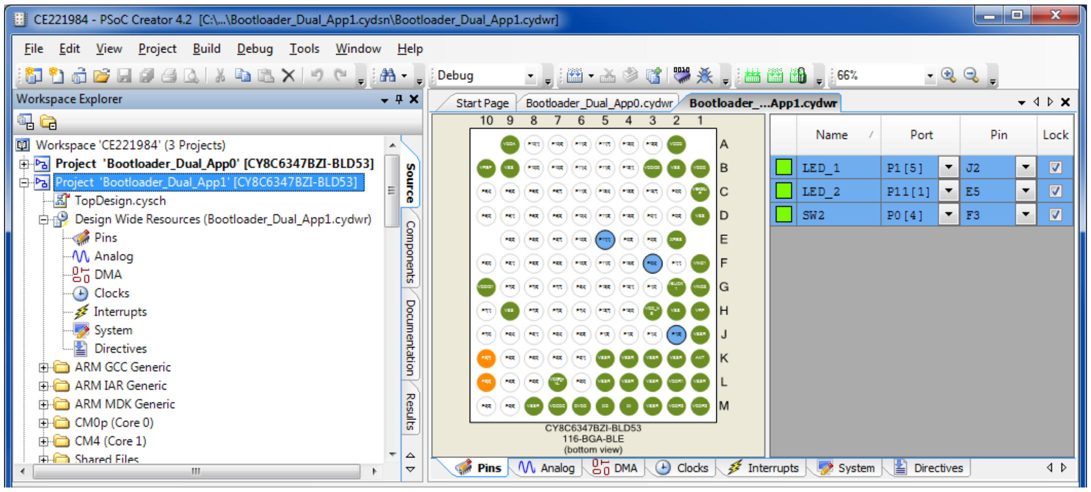
Figure 5 shows the pin assignments for App0. Figure 6 shows the pin assignments for App1 and App2.

Figure 5. Pin Assignments for CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit for I²C Bootloader


CE221984 - PSoC Creator 4.2 [C:\...\Bootloader_Dual_App0.cydsn\Bootloader_Dual_App0.cydwr]
File Edit View Project Build Debug Tools Window Help
Debug
66%
Workspace Explorer
Workspace 'CE221984' (3 Projects)
Project 'Bootloader_Dual_App0' [CY8C6347BZI-BLD53]
TopDesign.cysch
Design Wide Resources (Bootloader_Dual_App0.cydwr)
Pins
Analog
DMA
Clocks
Interrupts
System
Directives
ARM GCC Generic
ARM MDK Generic
CM0p (Core 0)
CM4 (Core 1)
Shared Files
Project 'Bootloader_Dual_App1' [CY8C6347BZI-BLD53]
Project 'Bootloader_Dual_App2' [CY8C6347BZI-BLD53]
Start Page
Bootloader_...App0.cydwr
CY8C6347BZI-BLD53
116-BGA-BLE
(bottom view)
Pins Analog DMA Clocks Interrupts System Directives

	Name	Port	Pin	Lock
🟩	\I2C:scl\	P6[0]	K8	✓
🟩	\I2C:sda\	P6[1]	J8	✓
🟩	Blue_LED	P11[1]	E5	✓
🟩	Red_LED	P0[3]	E3	✓
🟩	SW2	P0[4]	F3	✓

Figure 6. Pin Assignments for CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit Applications, App1 and App2


CE221984 - PSoC Creator 4.2 [C:\...\Bootloader_Dual_App1.cydsn\Bootloader_Dual_App1.cydwr]
File Edit View Project Build Debug Tools Window Help
Debug
66%
Workspace Explorer
Workspace 'CE221984' (3 Projects)
Project 'Bootloader_Dual_App0' [CY8C6347BZI-BLD53]
Project 'Bootloader_Dual_App1' [CY8C6347BZI-BLD53]
TopDesign.cysch
Design Wide Resources (Bootloader_Dual_App1.cydwr)
Pins
Analog
DMA
Clocks
Interrupts
System
Directives
ARM GCC Generic
ARM IAR Generic
ARM MDK Generic
CM0p (Core 0)
CM4 (Core 1)
Shared Files
Start Page
Bootloader_Dual_App0.cydwr
Bootloader_...App1.cydwr
CY8C6347BZI-BLD53
116-BGA-BLE
(bottom view)
Pins Analog DMA Clocks Interrupts System Directives

	Name	Port	Pin	Lock
🟩	LED_1	P1[5]	J2	✓
🟩	LED_2	P11[1]	E5	✓
🟩	SW2	P0[4]	F3	✓

Reusing This Example

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

It is also easy to change the host communication channel to UART, SPI, or BLE. For details, see the SDK Guide or other bootloader code examples in Related Documents.

Related Documents

PSoC 6 Bootloader-Related Application Notes	
AN213924 – PSoC 6 MCU Bootloader Software Development Kit (SDK) Guide	Provides information on how to use the Bootloader SDK, as well as information on bootloading in general.
Other PSoC 6 Application Notes	
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-core design.
PSoC 6 Bootloader-Related Code Examples	
CE213903 – Basic Bootloaders	Describes UART, I^{2}C, and SPI single-application bootloaders
CE216767 – BLE Bootloader	Describes a BLE bootloader
CE220959 – BLE Bootloader with External Memory	Describes a BLE Bootloader that uses SMIF external memory
PSoC Creator Component Datasheets	
I2C	Supports I^{2}C communication
Pins	Supports connection of hardware resources to physical pins
Device Documentation	
PSoC 63 with BLE Datasheet	PSoC 63 with BLE Architecture Technical Reference Manual
Development Kit Documentation	
CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit	

Appendix A: Code Theory of Operation

This section describes in detail how the code example source code implements the functions listed in Table 2.

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 */
PreferredAppID, NextAppID; /* conditionally compiled to 1 or 2 */
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 and transfers control to the preferred app, otherwise the other app.

Calls `LoadValidApp()`, which is part of the code example, not the SDK. This function returns if neither app is valid, otherwise does an SRES and does not return.

If the function returns, that means that there is no valid app to transfer control to; no action is taken.

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 the app just downloaded and, if success, stops host communication (`Cy_Bootload_TransportStop()`, *bootload_user.c*) and transfers control to that app (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 BOOTLOAD_ERROR_TIMEOUT and 5-second timeout. If so, resets host communication and restarts bootloading.

For other bootload errors, assume that it's an attempt to overwrite App1 while in factory default mode; turn ON the red LED for 5 seconds.

If 300-second timeout and state is NONE, validates and transfers control to the preferred app, otherwise the other app.

Calls `LoadValidApp()`, which is part of the code example, not the SDK. This function returns if neither app is valid, otherwise does an SRES and does not return.

If the function returns, that's an error condition; call `Cy_SysLib_Halt()`.

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

If the kit button is pressed, waits for button release. Then validates and transfers control to the preferred app, otherwise the other app.

Calls `LoadValidApp()`, which is part of the code example, not the SDK. This function returns if neither app is valid, otherwise does an SRES and does not return.

If the function returns, that means that there is no valid app to transfer control to; the button press is ignored.

Document History

Document Title: CE221984 – PSoC 6 MCU Dual-Application Bootloader

Document Number: 002-21984

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	6072561	MKEA	02/20/2018	New code example

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