CE213903 – PSoC 6 MCU Basic Bootloaders

Objective

These examples demonstrate basic 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.

Overview

These examples demonstrate several basic bootloading operations:

- Downloading an application from a host, using various PSoC Creator™ communication Components for host communication
- Installing the downloaded application into flash or external memory
- Validating an application, and then transferring control to that application

Multiple communication channels are supported, including UART, I^{2}C, and SPI.

Advanced communication channels such as BLE and USB are demonstrated in other code examples; see Related Documents.

Requirements

Tool: PSoC Creator 4.2; Peripheral Driver Library (PDL) 3.0.1

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

No special hardware setup need be done for the CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit. 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, as a USB-UART bridge for UART bootloading, as a USB-I^{2}C bridge for I^{2}C bootloading, and as a USB-SPI bridge for SPI bootloading. For more information, see the KitProg2 User Guide.

Software Setup

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

Operation

1. Using PSoC Creator, build the App0 project for the bootloader that you want to use: UART, I^{2}C, or SPI. For more information on building projects, see PSoC Creator Help.

   Note: When one of the App0 projects is built, the *bootload_user.c* file that is added contains default code for UART. For the App0_I2C and App0_SPI projects, first select **Build** > **Generate Application**, then edit *bootload_user.c* as follows:

   - Change `#include "transport_uart.h"` to `#include "transport_i2c.h"` or `#include "transport_spi.h"`
   - Change five instances of `"UART_Uart"` to `"I2C_I2c"` or `"SPI_Spi"`.

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

2. Connect CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit to your computer using the USB cable.

3. Program the App0 project into the kit. For more information on device programming, see PSoC Creator Help.

   Confirm that the kit red LED blinks once every two seconds. This indicates that App0 is running.

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. Build the App1 project. First, select **Build** > **Generate Application**. The installed linker script files are by default set up for application #0 (App0). For the App1 project, edit the files by changing the application number:

   - For the GCC compiler, the following example shows edits for App1 in *bootload_cm0p.ld* and *bootload_cm4.ld*:

```
/*
* Bootloader 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);

/* Bootloader SDK specific: sets app Id */
__cy_app_id = 1;
```

   - For the MDK compiler, the following example shows edits for App1 in *bootload_cm0p.scat* and *bootload_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 *bootload_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), or application transfer may fail. Check the **Build Settings** for each project.

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

   If you are using the UART bootloader, 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^2C, or USB-SPI bridge connection, depending on the bootloader that you are using.

   For more information on using BHP, see BHP Help or the Bootloader SDK User Guide.

7. Using BHP, download App1. After download is complete, confirm that the kit red LED blinks twice per second, indicating that App1 is running.
8. Press and hold the kit button for at least half a second, and then release it. Confirm that the kit red LED blinks once per two seconds, indicating that App0 is running.
9. Repeat steps 6 – 8 to test the process of reinstalling App1.
10. While in App0 and not updating App1, press and hold the kit button for at least half a second, and then release it. Confirm that the kit red LED blinks twice per second, indicating that App1 is running.

Design and Implementation

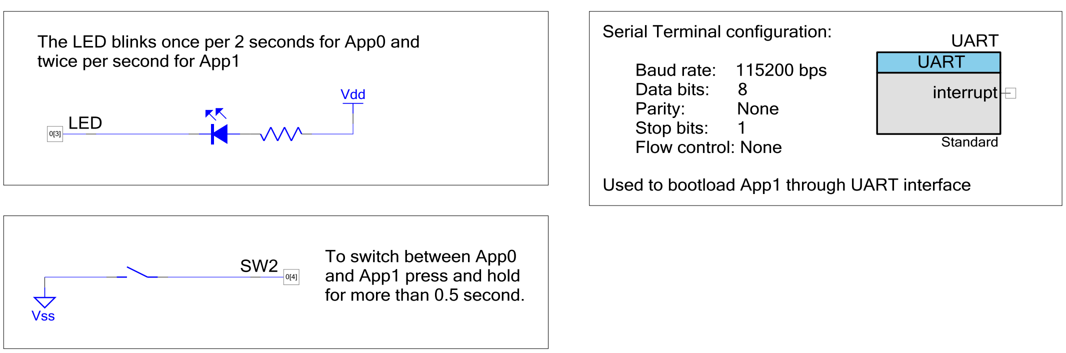
Each example has two applications, called "App0" and "App1". Each application has the following features:

- App0 does the bootloading; it downloads and installs App1.
- Each application blinks a kit LED at a different rate making it easy to see which application is currently running.
- Holding a kit button down for more than half a second, then releasing it, causes the application that is currently running to transfer control to the other application.

Each application is a separate PSoC Creator project; both projects are in the same PSoC Creator workspace.

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

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 application is running.

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 AN213924 - PSoC 6 MCU Bootloader Software Development Kit (SDK) 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_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 host communications Component being used. 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 image for App1.

Memory Layout

Figure 2 shows the typical memory usage for each core in each application. This layout is for the UART bootloader 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 UART 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 2. Memory Layout of Applications

SRAM	0x0804 7FFF Empty 0x0800 4810	~248 KB
	ram, core1 0x0800 2100	32 KB
	Core1 Vectors 0x0800 2100	1 KB
	ram, core0 0x0800 0100	8 KB
	Core0 Vectors 0x0800 0100	1 KB
	ram_common 0x0800 0000	256 B

Flash	TOC Must be the last row	512 B
	Metadata copy row 0x100F FC00	512 B
	Metadata flash row 0x100F FA00	512 B
	Empty 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), or application transfer may fail. Check the **Build Settings** for each project.

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.

Dual Core

PSoC 6 MCU has two CPU cores: Cortex-M4 and a Cortex-M0+. 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. 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 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 Cortex-M4. Does nothing else.	Blinks an LED once per two seconds Bootloads App1 Monitors the button After bootload 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 CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit, for the UART, I2C, SPI, LED, and button.

Figure 3. Pin Assignments for CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit for UART Bootloader

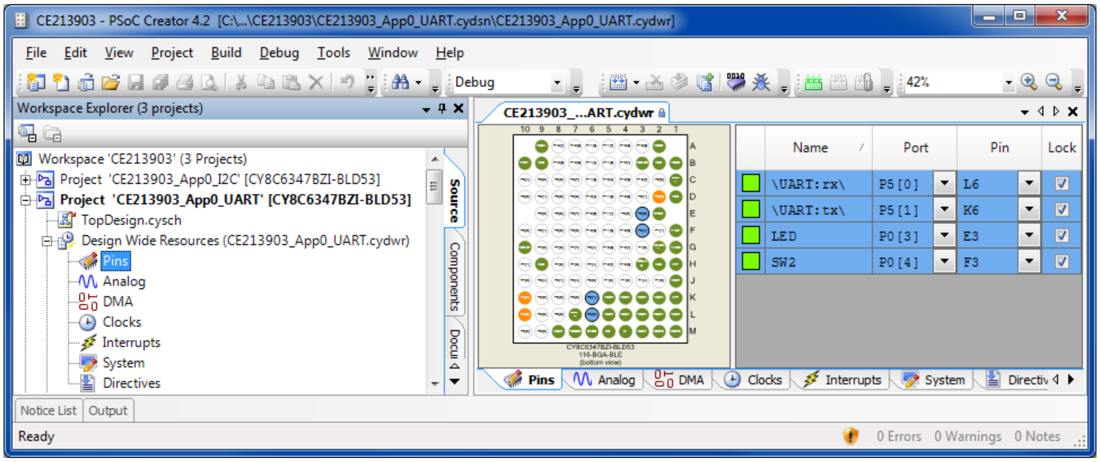


Figure 4. Pin Assignments for CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit for I^{2}C Bootloader

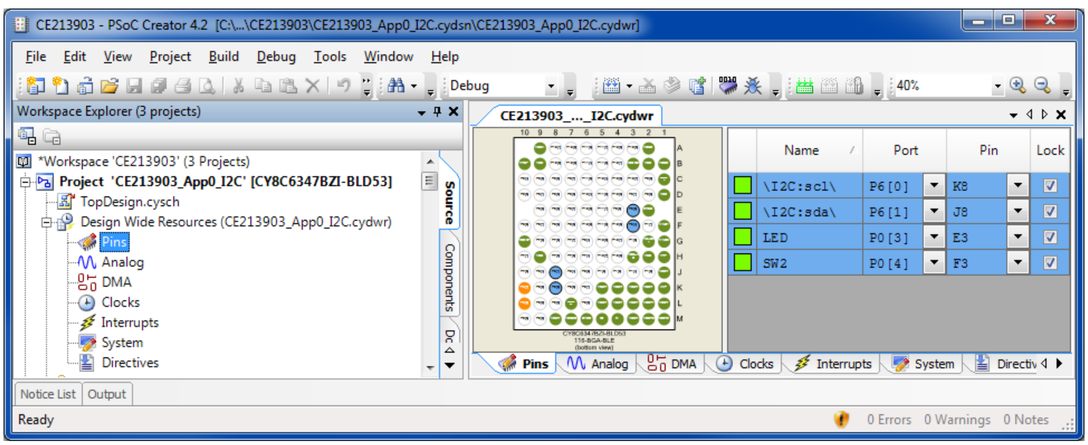
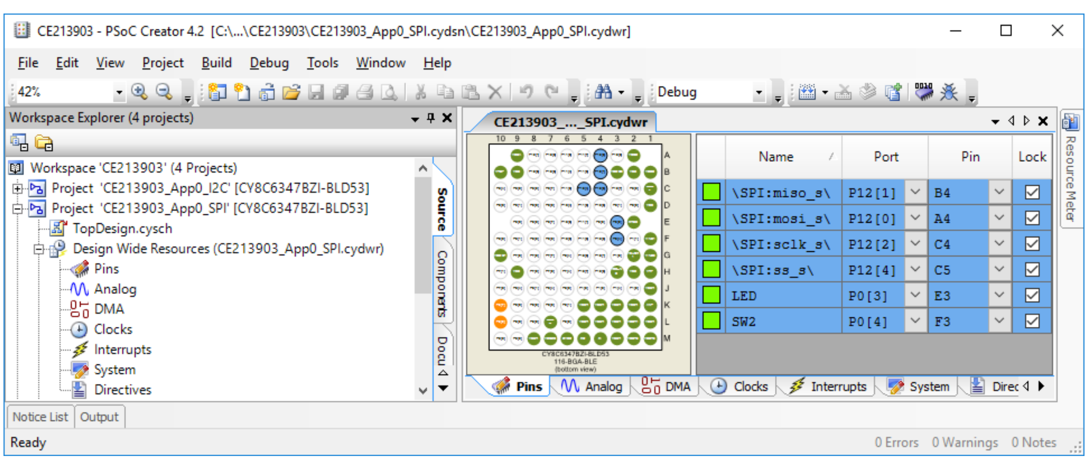


Figure 5. Pin Assignments for CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit for SPI Bootloader



Reusing This Example

This example is designed for the CY8CKIT-062-BLE pioneer kit. 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 (for example, an IP block) 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 what a particular device supports.

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 Application Notes	
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-Core CPU system Design	Describes the dual-core CPU architecture in PSoC 6 MCU, and shows how to build a simple dual-core design
AN219434 – Importing PSoC Creator Code into an IDE for a PSoC 6 MCU Project	Describes how to import the code generated by PSoC Creator into your preferred IDE
Bootloader-Related Code Examples	
CE216767 – PSoC 6 MCU with BLE Bootloader	Describes a BLE Bootloader for PSoC 6
CE220959 – PSoC 6 MCU with BLE Bootloader and External Memory	Describes a BLE Bootloader for PSoC 6 that uses SMIF external memory
PSoC Creator Component Datasheets	
UART	Supports the serial communication block in UART mode
I2C	Supports the serial communication block in I^2C 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 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 on page 6. The App0_UART project is described; the App0_I2C and App0_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 Bootloaders

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	02/090/2018	Updated projects for Bootloader SDK 2.10. No document change.

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

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