AN86526

PSoC 4 I²C Bootloader

Author: Charles Cheng

Associated Project: Yes

Associated Part Family: PSoC 4000, 4100, 4200, 4000S, 4100S, 4100S Plus, 4100PS

Software Version: PSoC Creator™ 4.2 SP2 and higher

Related Application Notes: Click here.

To get the latest version of this application note, or the associated project file, please visit http://www.cypress.com/AN86526.

More code examples? We heard you.

To access an ever-growing list of hundreds of PSoC code examples, please visit our code examples web page. You can also explore the PSoC video library here.

AN86526 describes an I²C-based bootloader for PSoC® 4. In this application note, you will learn how to use PSoC Creator™ to quickly and easily build an I²C-based bootloader project and bootloadable project. You will also see how to build an I²C-based embedded bootloader host program.

Contents

1 Introduction

Bootloaders are a common part of the MCU system design. A bootloader makes it possible for a product's firmware to be updated in the field. At the factory, initial programming of the firmware into a product is typically done through the MCU's Joint Test Action Group (JTAG) or Arm® serial wire debug (SWD) interface. However, these interfaces are usually not accessible in the field.

This is where bootloading comes in. Bootloading is a process that allows you to upgrade your system firmware over a standard communication interface such as USB, I²C, UART, or SPI. A bootloader communicates with a host to get new application code or data and writes it into the device's flash memory.

This application note covers the following topics:

- How to create an I²C bootloader using PSoC Creator
- Bootloader host topics:
  - How to use the Bootloader Host tool
  - The basic building blocks and functionality of a bootloader host system
  - How to create an embedded I²C bootloader host using PSoC

This application note assumes that you are familiar with PSoC 4, and the PSoC Creator IDE. If you are new to PSoC 4, see AN79953 - Getting Started with PSoC 4. If you are new to PSoC Creator, see the PSoC Creator home page.

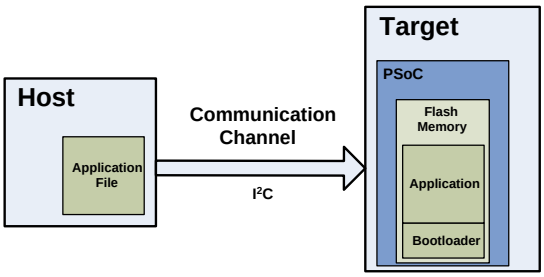
This application note also assumes that you are familiar with bootloader concepts. If you are new to these concepts, see AN73854 -PSoC 3, PSoC 4 and PSoC 5LP Introduction to Bootloaders. For a complete list of other application notes on bootloading, click here.

Finally, this application note assumes that you are familiar with the I²C protocol and the PSoC Creator I²C (SCB mode) Component. If you are new to this Component, see the PSoC 4 Serial Communication Block (SCB) Component datasheet. You can also get the datasheet by right-clicking on an I²C (SCB mode) Component in PSoC Creator. Refer to the following sections for information on PSoC Creator.

1.1 Terms and Definitions

Figure 1 illustrates the main elements in a bootloader system. It shows that the product's embedded firmware must be able to use the communication port for two different purposes: normal operation and updating flash memory. The portion of the embedded firmware that knows how to update the flash is called the **bootloader**.

Figure 1. Bootloading System Diagram



The system that provides the data to update the flash is called the **host**, and the system being updated is called the **target**. The host can be an external PC (PC Host) or another MCU (Embedded Host).

The act of transferring data from the host to the target flash is called **bootloading**, a **bootload operation**, or a **bootload** for short. The firmware that is placed in flash is called the **application** or the **bootloadable**.

Another common term in bootloading is in-system programming (ISP). Cypress has a product with a similar name but a different function called "In-System Serial Programmer (ISSP)" and an operation called "Host-Sourced Serial Programming (HSSP)." For more information, see AN84858 - PSoC 4 Programming Using an External Microcontroller (HSSP).

1.2 Using a Bootloader

A bootloader communication port is typically shared between the bootloader and the actual application. The first step to use a bootloader is to manipulate the target so that the bootloader, and not the application, is executing.

Once the bootloader is running, the host can send a “start bootload” command over the communication channel. If the bootloader sends an “OK” response, bootloading can begin.

During bootloading, the host reads the file for the new application, parses it into flash write commands, and sends those commands to the bootloader. After the entire file is sent, the bootloader can pass control to the new application.

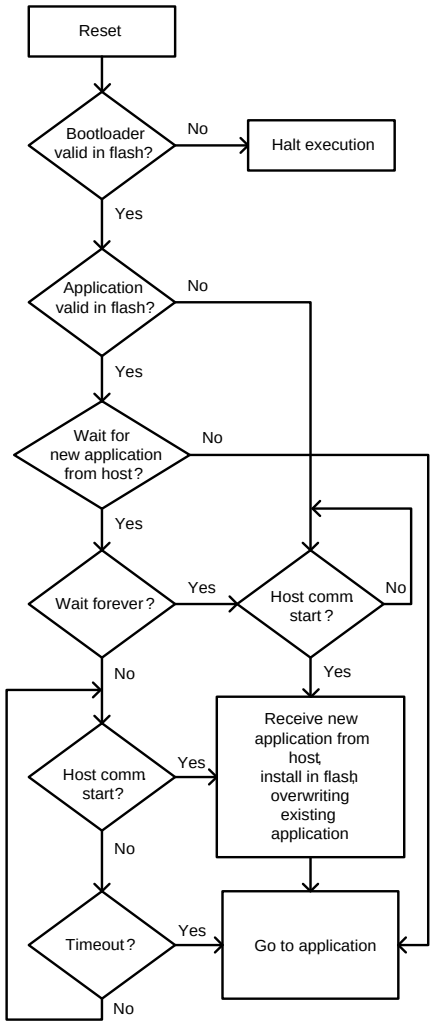
1.3 Bootloader Function Flow

Typically when the device resets, the bootloader is the first function to execute. It then performs the following actions:

- Checks the application's validity before letting it run
- Manages the timing to start host communication
- Does the bootload/flash update operation
- Passes control to the application

Figure 2 is a flow diagram that shows how this works.

Figure 2. Bootload Process Flowchart



1.4 Techniques to Enter Bootloader

As mentioned previously, the bootloader is the first function to run at reset. As Figure 2 shows, the bootloader code waits for the host for a short period of time before passing control to the application. This may cause the host to miss an opportunity to start the bootload operation. However, another way to start bootloading is to pass control from the application, or bootloadable, back to the bootloader.

1.4.1 Bootloadable API

The Bootloadable Component in PSoC Creator has an API function, `Bootloadable_Load()`, to start the bootloader. This allows the host to start a bootload operation at any time.

The problem with this method is that you must depend on the application code to perform an application upgrade. What happens if the application has a defect that prevents transfer of control to the bootloader?

1.4.2 Customize Bootloader

Instead, it may be better to have the bootloader wait an infinite amount of time for the host. To do that, you can customize the bootloader project to check for some user input before calling `Bootloader_Start()` and running through its normal routine.

For example, the bootloader may monitor the UART and wait forever for a user command before calling `Bootloader_Start()`. For more information, see AN73854 - PSoC 3, PSoC 4 and PSoC 5LP Introduction to Bootloaders.

2 Projects

This section lists the steps to create the following PSoC Creator projects:

- I²C Bootloader
- Bootloadable
- Embedded bootloader host

The projects are designed to be used with Cypress development kits. Kit selection is based on the target device; see Appendix F- Kit Selection for details. You may have to change the pin connection based on the kit. Review the specific kit documentation for connections. However, the projects can be easily adapted for other custom boards.

2.1 I²C Bootloader

In this section, you create and build an I²C-based bootloader project. One feature of this project is that while bootloading takes place, the kit's red LED blinks. The following examples demonstrate the I²C bootloader projects using the PSoC 4200 device, but the process is exactly the same for all other PSoC devices.

1. Create a new PSoC Creator project and name it "I2C_Bootloader_Red." Select the target PSoC device and create a new workspace for the project.

Note: For PSoC Creator 3.1 and lower versions, the "Application Type" should be specified while creating the project. For specifying the application type, click the '**+**' button next to the **Advanced** tab to expand the configuration option. Select Bootloader as the application type.

2. Add an I²C (SCB mode) and a Bootloader Component to the top design schematic. To blink an LED, add a PWM (TCPWM mode), a Clock, and a Digital Output Pin Component. Rename the Components as Table 1 shows.

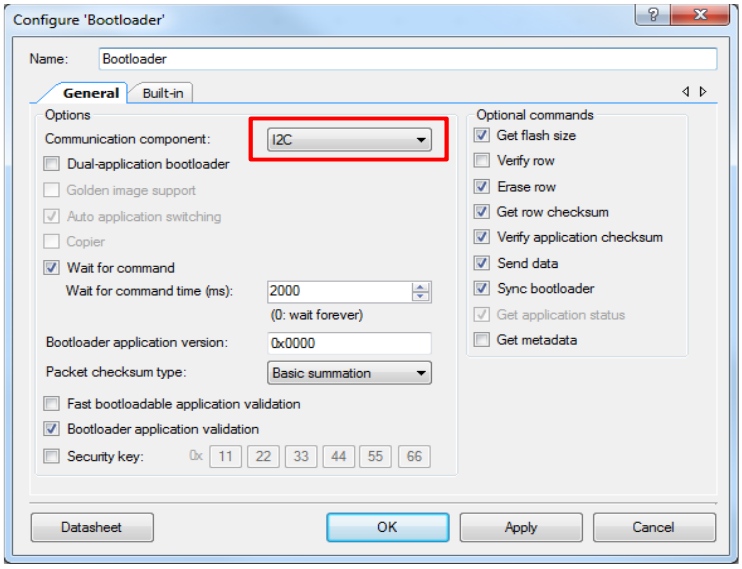
Table 1. I2C_Bootloader_Red Project Component Names

Component	Name
Bootloader_1	Bootloader
I2C_1	I2C
PWM_1	PWM
Clock_1	Clock
Pin_1	Pin_LED

3. To configure the Bootloader, double-click on the Component. Select **I2C** as the **Communication component**, as Figure 3 shows. Leave the other parameters at their default settings. For more information on these configuration parameters, refer to the Bootloader Component datasheet.

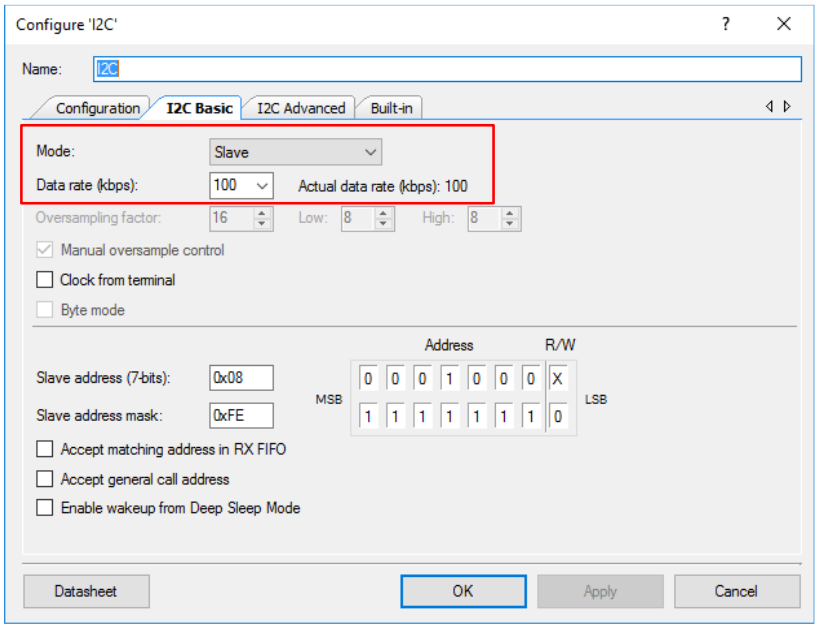
Note: The parameter **Wait for command time** is set to 2 seconds. For details on bootloader wait time, see "Wait for command time" in the Bootloader Component datasheet.

Figure 3. Bootloader Configuration



4. To configure the I²C (SCB mode) Component, double-click on it. By default, the mode is set to Slave, the data rate is set to 100 kbps, and the slave address is set to 8. If you want to use a different address, enter it in the **Slave address** box. Leave the other parameters at their default settings. Figure 4 shows the basic configuration tab of the I²C Component.

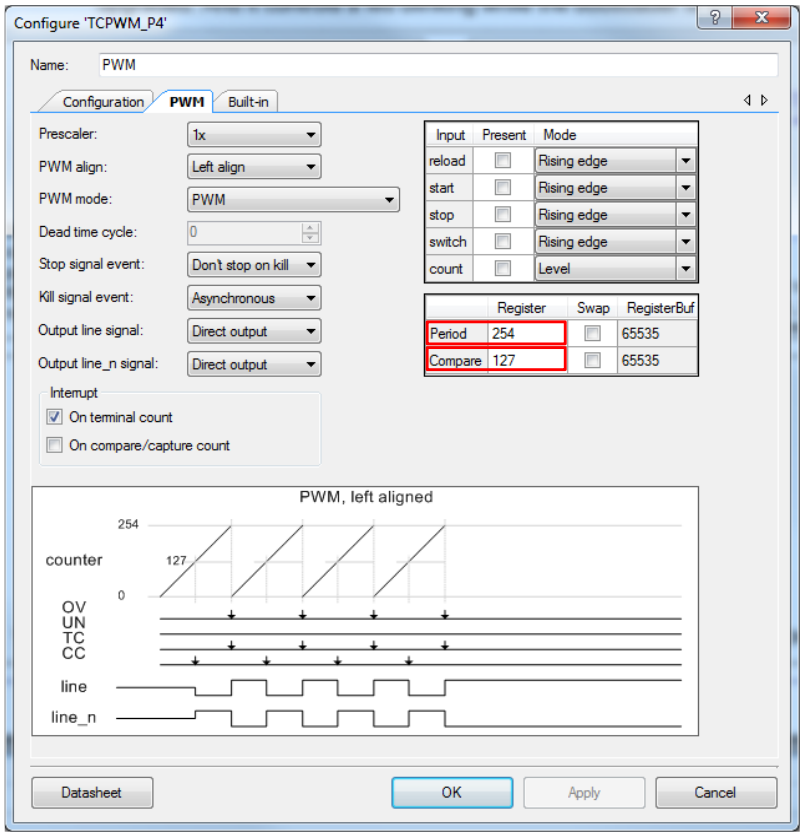
Figure 4. Basic I²C (SCB mode) Configuration



The I²C (SCB mode) Component contains Pin Components that are automatically configured for use with an I²C bus (open drain, drives low). You must add external pull-up resistors, as Figure 8 on page 7 shows, and assign them to physical pins (step 8 on page 7).

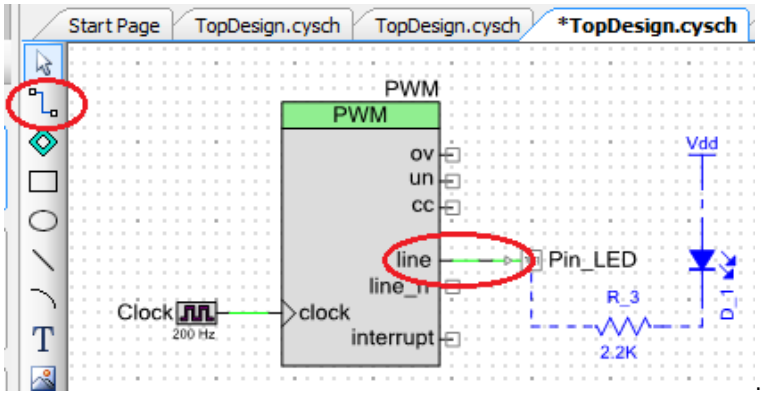
5. To configure the PWM (TCPWM mode) Component, double-click the component. Set **Period** to 254 and **Compare** to 127. Leave the other parameters at their default settings. Figure 5 shows the basic configuration tab of the PWM (TCPWM mode) Component.

Figure 5. Basic PWM (TCPWM mode) Configuration



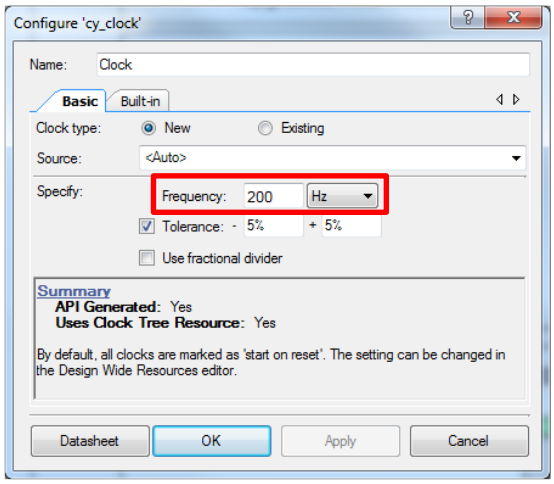
6. For the Pin Component, leave the parameters at their default settings. Connect the Pin to the “line” terminal of the PWM Component using a wire, as Figure 6 shows.

Figure 6. Pin Connection



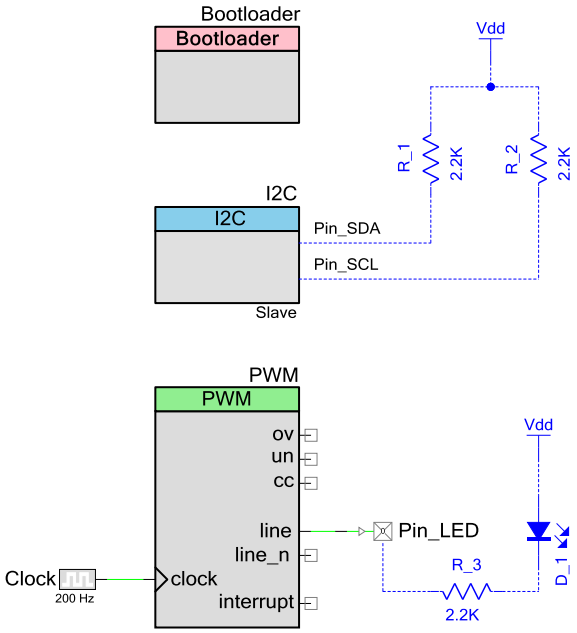
7. To configure the Clock Component, double-click the Component. Set the **Frequency** to 200 Hz, as Figure 7 shows. Leave the other parameters at their default settings.

Figure 7. Clock Configuration



With resistors and an LED added as Annotation Components, the top design of the project looks similar to Figure 8.

Figure 8. Top Design of I2C_Bootloader_Red Project

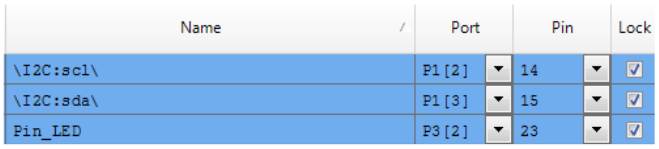


8. Assign the I²C Pin Components to physical pins. In the **Workspace Explorer** window, double-click the *I2C_Bootloader_Red.cydwr* file, and click the **Pins** tab. Pin assignments depend on the device; see the device datasheet for information. For example, for the PSoC 4200 device on CY8CKIT-042, assign the Pins as Figure 9 shows, and for PSoC 4000 on CY8CKIT-040, assign the pins as Figure 10 shows.

Figure 9. I²C Pin Assignments for CY8CKIT-042

Name	Port	Pin	Lock
\I2C:scl\	P3[0]	11	☑
\I2C:sda\	P3[1]	12	☑
Pin_LED	P1[6]	43	☑

Figure 10. I²C Pin Assignments for CY8CKIT-040



Name	Port	Pin	Lock
\I2C:scl\	P1[2]	14	☑
\I2C:sda\	P1[3]	15	☑
Pin_LED	P3[2]	23	☑

9. Add the `Bootloader_Start()` function in the `main()` function. This API function does the entire bootloading operation. It does not return; it ends with a software device reset. Therefore, the code after this function call is never executed. Add two API functions to initialize the PWM in `main()`, as Code 1 shows. For more information on this Component API, refer to the TCPWM Component datasheet.

Code 1. PWM Initialization in Bootloader

```
Intmain()
{
  /* Initialize PWM */
  PWM_Start();
  PWM_TriggerCommand(PWM_MASK,
                     PWM_CMD_START);

Bootloader_Start();

for(;;)
  {
    /* Place your code here. */
  }
}
```

10. Build the project and program it into the specific kit based on your target device selection.

2.2 Bootloadables

You will now create two bootloadable projects. The first project blinks a green LED on the PSoC development kit. The second project demonstrates how to enter the bootloader project from a bootloadable project, and it blinks the blue LED on the kit.

Note There are RGB LEDs on the CY8CKIT-042 and CY8CKIT-040, as Figure 31 and Figure 32 on page 20 show. Here the green LED and the blue LED are used to indicate which project is running.

2.2.1 Bootloadable_Green Project – Example 1

This section describes the steps to create the first bootloadable project.

1. Create a new PSoC Creator project of application type Bootloadable. Name the project **Bootloadable_Green**. The devices for this project and the I2C_Bootloader_Redproject must be the same.

2. For this project, you need the Bootloadable and Digital Output Pin Components. Add the two Components to your top design schematic. Name the Components according to Table 2.

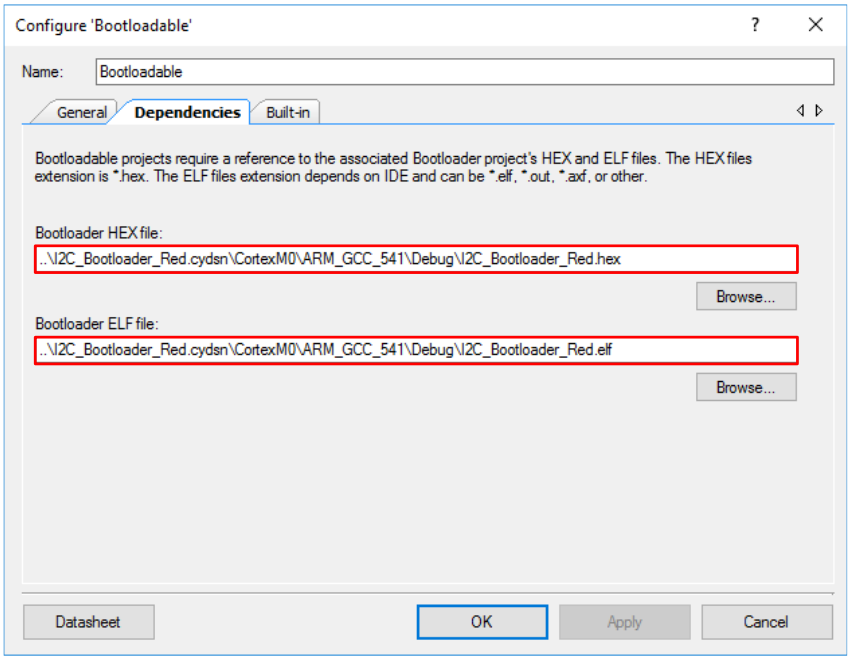
Table 2. Bootloadable_Green Project Component Names

Component	Name
Bootloadable_1	Bootloadable
Pin_1	Pin_LED

3. To configure the Bootloadable Component, double-click the Component.

A bootloadable project is always linked to the *.hex* file of a bootloader project. To link the project, go to the **Dependencies** tab and then link the bootloadable to the *I2C_Bootloader_Red.hex* file, as Figure 11 shows. For more information on Bootloadable Component configuration, see the Bootloadable Component datasheet.

Figure 11. Bootloadable Component Configuration



You can find the *I2C_Bootloader_Red.hex* file in the bootloader project's *Debug* or *Release* folder:

1. For PSoC 4000, 4100, 4200

   ..\I2C_Bootloader_Red.cydsn\CortexM0\ARM_GCC_541\Debug\I2C_Bootloader_Red.hex
2. For PSoC 4000S, 4100S, 4100S Plus, and 4100PS:

   ..\I2C_Bootloader_Red.cydsn\CortexM0p\ARM_GCC_541\Debug\I2C_Bootloader_Red.hex
3. To configure the Pin Component, double-click the Component. Disable the **HW Connection**, as Figure 12 shows. Set the drive mode to **Strong Drive**, as Figure 13 shows.

Figure 12. Pin Component Configuration – Type

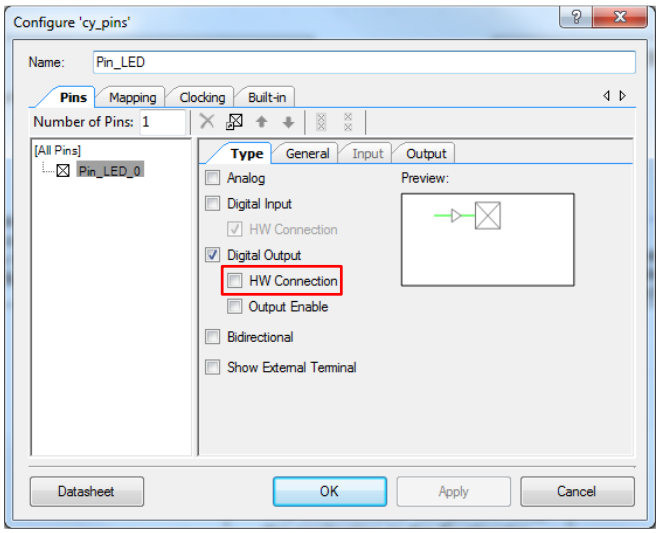
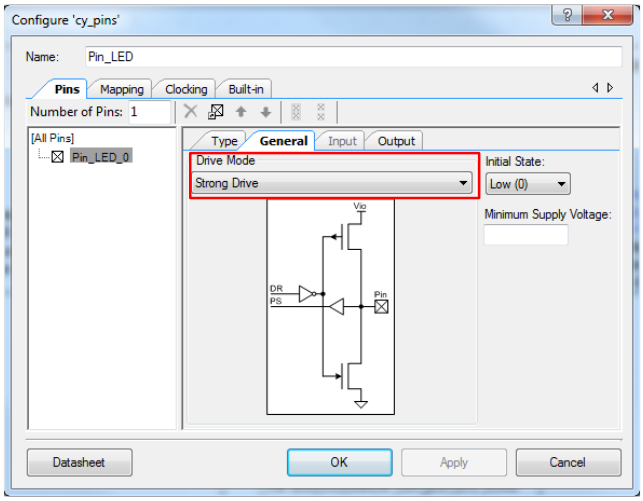
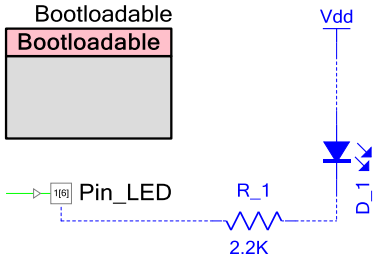


Figure 13. Pin Component Configuration – General



With the addition of Annotation Components for the LED, the top design is complete; it should be similar to Figure 14.

Figure 14. Top Design of Bootloadable_Green Project



4. Add the following code to *main.c* to blink the LED:

```
Voidmain()
{
for(;;)
  {
/* Toggle the LED */
Pin_LED_Write(~Pin_LED_Read());

/* Delay 1 second */
CyDelay(1000u);
  }
}
```

5. Assign the Pin Component to a physical pin. In the **Workspace Explorer** window, double-click the *Bootloadable_Green.cydwr* file and assign the pins. For pin assignments, see the kit user guide. For example, Figure 15 shows pin assignments for the CY8CKIT-042 kit board and Figure 16 shows pin assignments for the CY8CKIT-040 kit board.

Figure 15. Pin Assignment of Bootloadable_Green Project for CY8CKIT-042

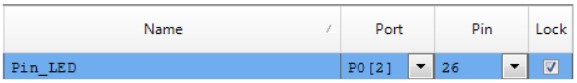


Figure 16. Pin Assignment of Bootloadable_Green Project for CY8CKIT-040

6. Build the project. When a bootloadable project is built, PSoC Creator generates a *.cyacd* file. This is the file that is bootloaded onto the target. For more information on this file and its contents, see Appendix B.

Now you will bootload this project into PSoC using PSoC Creator.

2.2.2 Bootloading Using a PC Host

A bootloader host executable is provided with PSoC Creator for bootloading an application from a PC host. The PSoC 4 kits implement a USB-I^{2}C bridge using the onboard PSoC 5LP. The bridge can be used to interface the PC to the bootloader, as shown for the CY8CKIT-042 kit in Figure 17 and for the CY8CKIT-040 in Figure 18.

Figure 17. Bootloading Using a PC Host for CY8CKIT-042

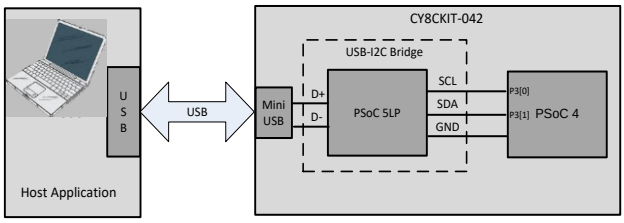
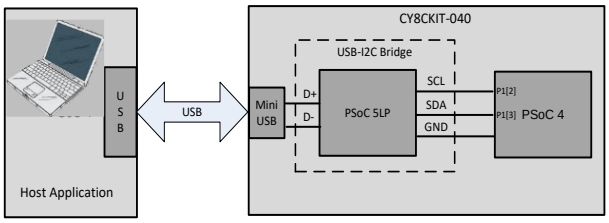


Figure 18. Bootloading Using a PC Host for CY8CKIT-040



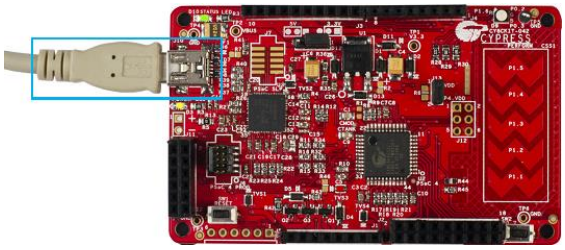
Note: In all PSoC 4 kits, pins connected with on-board PSoC 5LP SCL and SDA pins have internal pull-up resistors. Therefore, you need not use the external pull-ups. For information on the I²C pin connections, see the kit user guide.

Note: You can also use the CY8CKIT-002 MiniProg3 as a USB-I²C bridge to bootload. For more information, see the MiniProg3 User Guide and the knowledge base article MiniProg3 Connections for Bootloading Over I²C.

Follow these steps to bootload an application using the bootloader host program for the PSoC 4200 device. The procedure is the same for other parts of the PSoC family. As noted previously, you must program the bootloader project into the PSoC device before starting a bootload operation.

1. To start, connect the CY8CKIT-042 to the PC through the USB cable (which provides power), as Figure 19 shows.

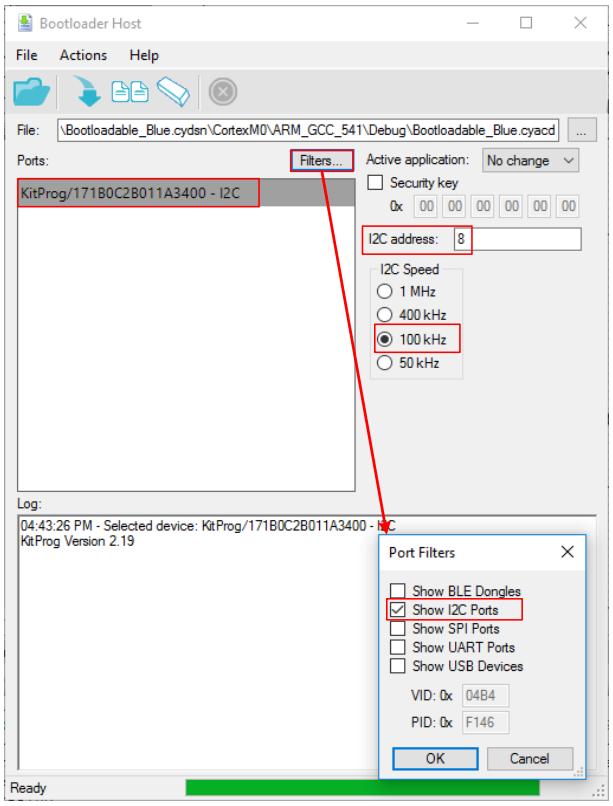
Figure 19. Connect USB Cable to J10 in CY8CKIT-042



2. Open the Bootloader Host tool by navigating to **Tools** > **Bootloader Host** in PSoC Creator.
3. Make sure that the bootloader host application's I²C configuration, shown in Figure 20, is the same as the bootloader project's I²C (SCB mode) Component configuration (Figure 4 on page 5).

Note: If there is no KitProg information in the Bootloader Host GUI after connecting the kit through the USB cable, click **Filters** and make sure that **Show I2C Devices** is enabled.

Figure 20. Bootloader Host Application

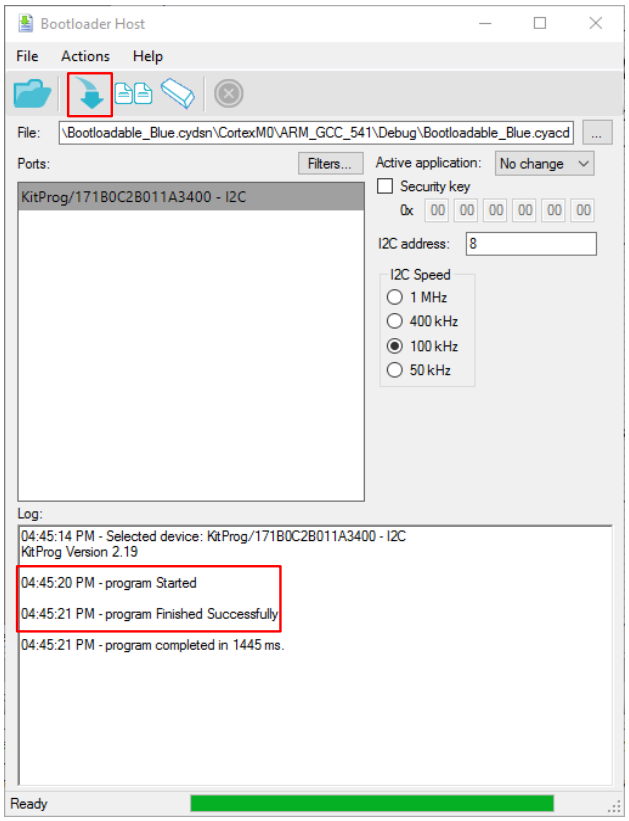


4. Press the **File** button and select the bootloadable file *Bootloadable_Green.cyacd* in the bootloadable project's *Debug* or *Release* folder:
   - For PSoC 4000, 4100, 4200

     ..\Bootloadable_Green.cydsn\CortexM0\ARM_GCC_541\ Debug\Bootloadable_Green.cyacd
   - For PSoC 4000S, 4100S, 4100S Plus, and 4100PS

     ..\Bootloadable_Green.cydsn\CortexM0p\ARM_GCC_541\ Debug\Bootloadable_Green.cyacd
5. To bootload the device, click the **Program** button. You should see a screen similar to Figure 21.

Figure 21. Downloading Bootloadable Project



6. After the bootloadable project is bootloaded successfully, a software reset occurs, and the device starts executing the new application. The kit RGB LED blinks green.

To bootload another application, reset the device (press and release the reset button of the PSoC development kit) to activate the bootloader. Then press the program button of the bootloader host within **Wait for command time**, which is set in Figure 3 on page 5. For details on bootloader wait time, see “Wait for Command Time” in the Bootloader and Bootloadable Component datasheet.

2.2.3 Bootloadable_Blue Project – Example 2

If you want to bootload a new project (application upgrade) when the current bootloadable project is running, the bootloadable project can invoke the bootloader by calling the API function `Bootloadable_Load()`.

In this example, the `Bootloadable_Load()` function is called when a button is pressed. This section describes the steps for creating this bootloadable project.

1. Create a new PSoC Creator project of application type Bootloadable, similar to Example 1. Name the project “Bootloadable_Blue.” The PSoC devices for this project and the I2C_Bootloader_Red project must be the same.
2. For this project, you need a Bootloadable Component, two Pin Components, and an Interrupt Component. Add these Components to your top design schematic and name them according to Table 3.

Table 3. Bootloadable Component Names

Component	Name
Bootloadable_1	Bootloadable
Pin_1 (Digital Input Pin)	Pin_StartBootloader
Pin_2 (Digital Output Pin)	Pin_LED
isr_1	isr_EnterBootloader

3. Link the Bootloadable Component to the bootloader project, as Figure 11 on page 9 shows.
4. The digital input pin Pin_StartBootloader is used to switch from the application back to the bootloader. When the button connected to this pin is pressed, the application enters the bootloader by calling the API function `Bootloadable_Load()`. The bootloader waits indefinitely for the host to start the bootload operation.

   When the DVK button is pressed, it shorts to ground, so configure the drive mode of the pin to be **Resistive Pull Up**, as Figure 22 shows. Also, configure it to generate an interrupt on its **Falling Edge**, as Figure 23 shows. The interrupt is generated when the button is pressed (and not when it is released).

   Finally, click the **Type** tab and disable the **HW Connection**.

Figure 22. Digital Input Pin Configuration

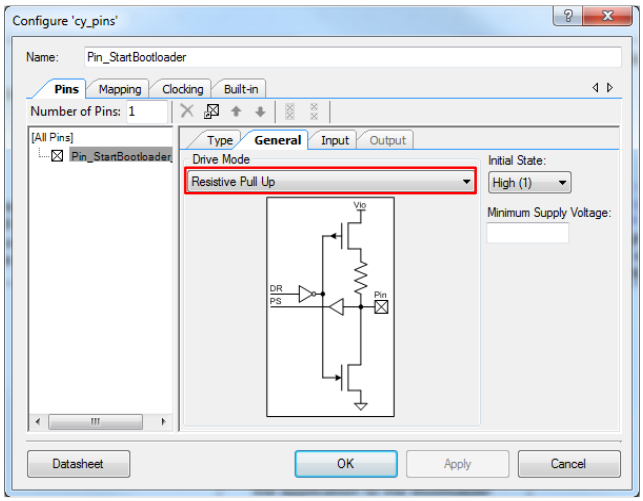
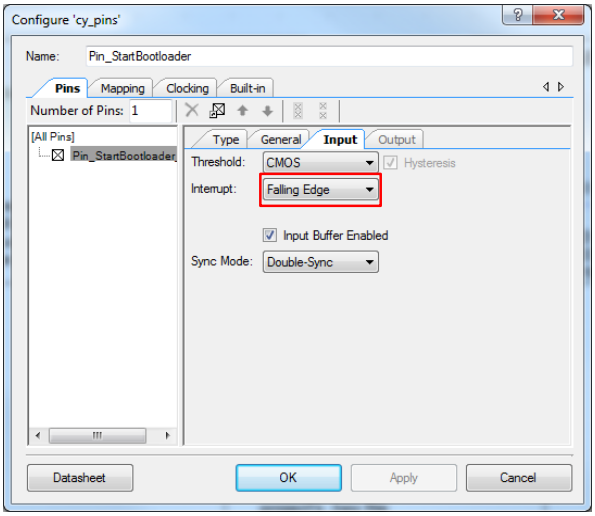
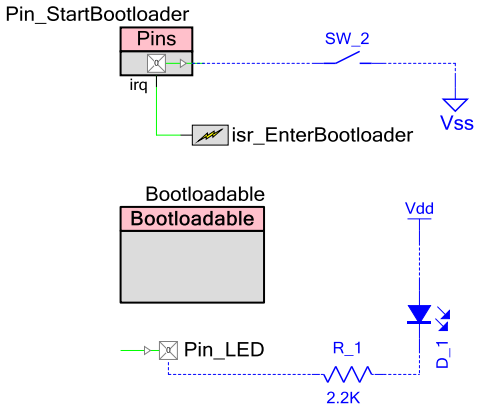


Figure 23. Digital Input Pin Configuration



5. The digital output pin Pin_LED is used to control the LED. Its setting is the same as that of the Pin_LED in Example 1; see Figure 12 and Figure 13.
6. Connect the ISR Component isr_EnterBootloader to the interrupt terminal (irq) of the Pin_StartBootloader. With the addition of Annotation Components for the button and LED, the top design is complete; it should be similar to Figure 24.

Figure 24. Top Design of the Bootloadable_Blue Project



7. Assign the Pin Components to physical pins. In the **Workspace Explorer** window, double-click the *Bootloadable_Blue.cydwr* file and assign the pins. For pin assignments, refer to the kit user guide for Cypress kits. Figure 15 shows pin assignments for the CY8CKIT-042 kit board and Figure 16 shows pin assignments for the CY8CKIT-040 kit board.

Figure 25. Pin Assignment of Bootloadable_Blue Project for CY8CKIT-042

Name	Port	Pin	Lock
Pin_LED	P0[3]	27	☑
Pin_StartBootloader	P0[7]	31	☑

Figure 26. Pin Assignment of Bootloadable_Blue Project for CY8CKIT-040

Name	Port	Pin	Lock
Pin_LED	P0[2]	3	☑
Pin_StartBootloader	P0[7]	11	☑

8. Build the project; this generates the ISR Component API files. Then add the code to the interrupt service routine to set the variable `bootload_flag`. The code is as follows.

```
CY_ISR(isr_EnterBootloader_Interrupt)
{
/* Place your Interrupt code here.
*/
/* `#START
isr_EnterBootloader_Interrupt`
*/
  bootload_flag=1u;

Pin_StartBootloader_ClearInterrupt();
  /* `#END` */
}
```

Note: The variable `bootload_flag` is defined in *main.c* and therefore must be declared as an extern variable in the *isr_EnterBootloader.c* file. Also, to avoid a build warning, `#include<device.h>` in the *isr_EnterBootloader.c* file.

9. A completed Bootloadable_Blue project is associated with this application note. Insert the code listing from the *main.c* file of this associated project into the *main.c* file of your project.

   The `main()` function continuously checks the `bootload_flag` variable. If the variable is set, `main()` turns OFF the LED and calls the API function `Bootloadable_Load()` to invoke the bootloader.

10. Build the project again. Download the bootloadable project *Bootloadable_Blue.cyacd* using the steps described in the section Bootloading Using a PC Host. The kit RGBLED on the target blinks blue.

11. Press the **SW2** switch to enter the bootloader and the kit RGB LED starts blinking red.

12. Start the bootloader host program and select a different bootloadable file such as *Bootloadable_Green.cyacd* and press the **Program** button. After the new application is successfully bootloaded, the kit RGB LED blinks green.

    Note: To bootload again, you must reset the device and quickly download the new *.cyacd* file. This is because the Bootloadable_Green application does not have the `Bootloadable_Load()` function call to invoke the bootloader, and hence the bootloader can be invoked only on reset.

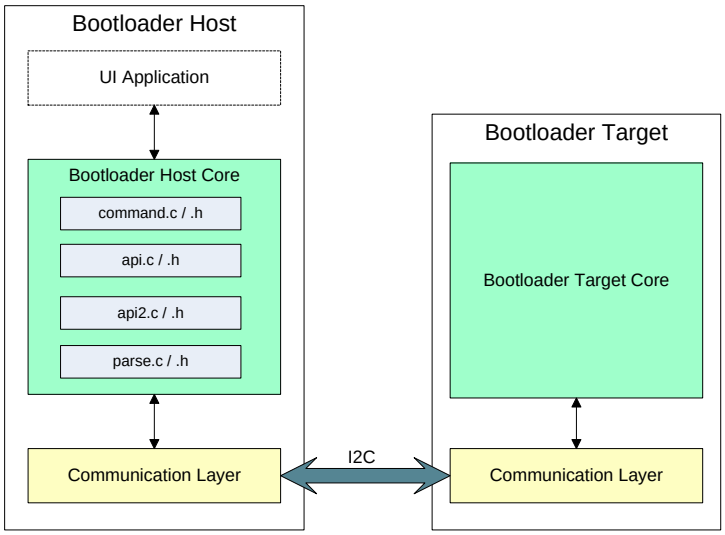
2.3 I²C Bootloader Host

In addition to studying the example projects, understanding the general structure of a bootloader host program can help you to build your own bootloader host system.

2.3.1 Bootloader Host Program

Figure 27 illustrates a protocol-level diagram of a bootloader system. The bootloader host and target each have two blocks—a core and a communication layer.

Figure 27. Protocol-Level Diagram of Bootloading



The bootloader host core performs all bootloading operations. It sends command packets and flash data to the target. Based on the response from the target, it decides whether to continue bootloading.

The bootloader target core decodes the commands from the host; executes them by calling flash routines such as erase row, program row, and verify row; and forms response packets.

The communication layer on both the host and the target provides physical layer support to the bootloading protocol. They contain communication protocol (I²C) specific APIs to perform this function. This layer is responsible for sending and receiving protocol packets between the host and the target.

2.3.2 Bootloader System APIs

PSoC Creator automatically generates all APIs for the bootloader target core and communication layer when you build a bootloader project.

PSoC Creator also provides the host-side APIs for the core, which you can find at this location:

<install folder> \ PSoC Creator \ 4.2 \ PSoC Creator \ cybootloaderutils

For more information on these API files, see Appendix D.

The only code that you need to write is the host-side API functions for the communication layer, which are in a file pair *communication_api.c / .h*. There are four functions: `OpenConnection()`, `CloseConnection()`, `ReadData()`, and `WriteData()`. Function pointers point to them within the "CyBtldr_CommunicationsData" structure defined in *cybtldr_api.h*.

This project uses the Bootloadable_Blue Project – Example 2 to generate the *.cyacd* files needed to invoke the bootloader on a switch press. Generate a *.cyacd* file for this project and rename it to *Bootloadable_BlueLED.cyacd*. Similarly, generate a *Bootloadable_GreenLED.cyacd* by assigning Pin_LED to P0[2]. These two files are provided with the application note for quick reference.

2.3.3 Steps to Create an I²C Bootloader Host Project

This section shows you how to create an embedded I²C bootloader host project using PSoC, which can bootload another PSoC device. With this project, the host can bootload two different bootloadable files (*.cyacd* files) on alternate switch presses.

1. Create a new PSoC Creator project. Select the PSoC device, name the project as **I2C_Bootloader_Host** and create a new workspace for the project.
2. Since the bootloader project has an I²C slave, the host project must have an I²C master. So, add an I²C (SCB mode) Component to the top design schematic. Also, add Digital Input Pin, Interrupt, and UART (SCB mode) Components to the top design. Name the Components according to Table 4.

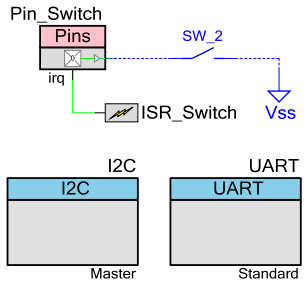
Table 4. Component List for I2C_Bootloader_Host Project

Component	Name
I2C_1	I2C
Pin_1	Pin_Switch
isr_1	ISR_Switch
UART_1	UART

3. To configure the I2C Component, double-click the component. Set it to Master mode. By default, the data rate is 100 Kbps.
4. The digital input pin Pin_Switch is used to initiate the bootloading operation in the host. When the kit button is pressed, it shorts to ground, so you need to configure this pin to have a resistive pull-up and generate an interrupt on its falling edge. Connect the ISR_Switch Component to the interrupt output (irq) of this pin.
5. The UART is used to transfer the bootloading status or error code to the PC. Leave the parameters at their default settings (baud rate: 115200 kbps, data bits: 8 bits, parity: none, stop bits: 1 bit).

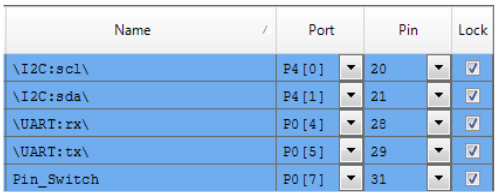
   With the addition of an Annotation Component for the button, the top design of this project should be similar to Figure 28.

Figure 28. Top Design of the I2C_Bootloader_Host Project



6. Assign the input and output pins. In the **Workspace Explorer** window, double-click the *I2C_Bootloader_Host.cydwr* file and assign the pins. For pin assignments, refer to the device datasheet or kit user guide for Cypress kits. Figure 29 shows pin assignment for CY8CKIT-042.

Figure 29. Pin Assignment for the I2C_Bootloader_Host Project



7. Build the project to generate the ISR Component API files. Add code to the interrupt service routine to set the variable `switch_flag`. The code is as follows.

```
CY_ISR(ISR_Switch_Interrupt)
{
/* Place your Interrupt code here.
 */
/* `#START ISR_Switch_Interrupt` */
  switch_flag = 1u;
Pin_Switch_ClearInterrupt();
/* `#END` */
       }
```

Note: `switch_flag` is defined in *main.c* and therefore must be declared as an extern variable in *ISR_Switch.c*. Also, to avoid a build warning, `#include<device.h>` in the *ISR_Switch.c* file.

8. Add firmware to this project. The I2C_Bootloader_Host project is attached to this application note. Insert the code listing from the *main.c* file of this associated project into the *main.c* file of your project.

   The `main()` function in *main.c* continuously checks the `switch_flag` variable. When it is set, bootloading is initiated. The file *main.c* has a function call to `BootloadStringImage()` that is defined in *device.h*. This function bootloads the *.cyacd* file using the Bootloader Host API files (host core; see Figure 27).

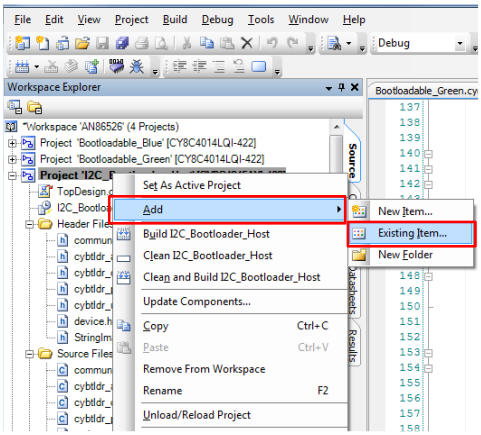
   The `main()` function has another variable called `toggle_appcode`. It alternates between '0' and '1' on each button press. This makes the host select alternate bootloadable files.

9. As explained previously, a bootloader host core is built upon four API files. These files do all the host bootloading operations. You must include these files in this project. Find these API files at the following location:

   <install folder> \ PSoC Creator \ 4.2 \ PSoC Creator \ cybootloaderutils

   To add these files, go to the **Workspace Explorer** window, right-click the project name, and select **Add > Existing Item**, as Figure 30 shows. Add the following files provided by PSoC Creator: *cybtldr_api.c / .h*, *cybtldr_command.c / .h*, and *cybtldr_parse.c / .h*, and *cybtldr_utils.h*.

Figure 30. Adding API Files



10. In addition to the bootloading API files, the host also requires communication layer support. This support is provided by adding the *communication_api.c / .h* files. You may include the contents of these files from the I2C_Bootloader_Host project associated with this application note (follow the previous step to add these files to the project). Update these files by copying from the project attached to this application note.
11. Include the bootloadable files in the host system. When a bootloadable file is built, a *.cyacd* file is generated; the file is similar to a *.hex* output file. For more information on the *cyacd* file, see Appendix B.

    Copy the contents of this file in the form of an array of strings such that each line is an element of the array. Since there are two bootloadable files, you must define two such arrays, named “StringImage_0” and “StringImage_1”. For each array, define a macro to store the number of lines in that array. Define these arrays in a separate file named *StringImage.h* (this file must be added to the project before defining the strings).

    Refer to the *StringImage.h* file in the I2C_Bootloader_Host project associated with this application note.

    Build the project and program it into the PSoC on the Cypress kit.

3 Testing the Projects

Note: The LEDs on the CY8CKIT-042 kit and CY8CKIT-040 are RGB LEDs, as Figure 31 and Figure 32 show. Here, the green LED and the blue LED are used respectively to indicate which bootloadable is currently running.

Figure 31. RGB LED Schematic Diagram for CY8CKIT-042

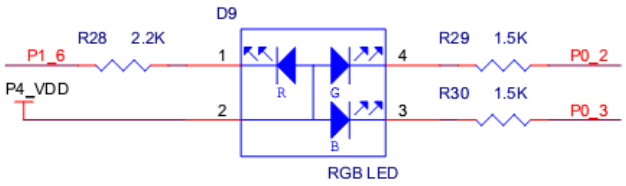
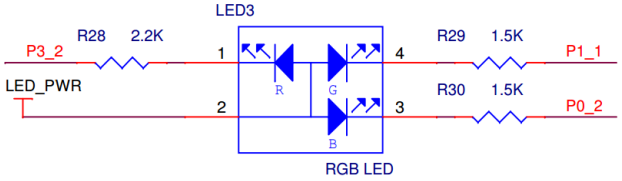


Figure 32. RGB LED Schematic Diagram for CY8CKIT-040



3.1 Configuring the Kits

To test the projects, configure the kits as follows.

For the target PSoC kits, follow these steps:

1. Connect the target to the PC with the USB cable (which provides power).
2. Program the target PSoC with the I2C_Bootloader_Red project.

For the host PSoC kits, follow these additional steps:

1. Connect the host to the PC with the USB cable (which provides power).
2. Program the host PSoC with the I2C_Bootloader_Host project.
3. Open any serial port viewer such as Tera Term on the PC to display the bootloading information.

For example, for the host CY8CKIT-042 kit, connect the Rx (P0[4]) and Tx (P0[5]) pins assigned in PSoC 4 to pin 10 and pin 9 on the expansion header J8. For more information, see the CY8CKIT-042 PSoC 4 Pioneer Kit Guide. Program it with the I2C_Bootloader_Host project and open serial port viewer to display the bootloading information.

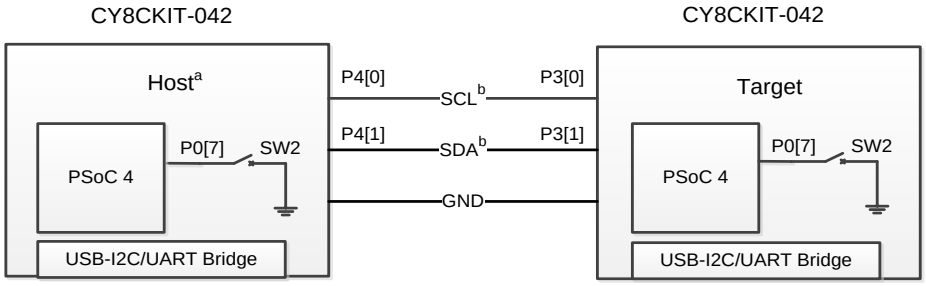
For making connections between the host kit and the target, follow these steps:

1. Connect SCL and SDA pins of the host kit to the respective SCL and SDA pins of the target kit.
2. Short together the ground pins of the kits.

For example, for the CY8CKIT-042 host and CY8CKIT-042 target, connect P4[0](SCL) and P4[1] (SDA) of the host to the respective P3[0] (SCL) and P3[1] (SDA) pins of the target as shown in Figure 33. For the CY8CKIT-042 host and CY8CKIT-040 target, connect P4[0] (SCL) and P4[1] (SDA) of the host to the respective P1[2] (SCL) and P1[3] (SDA) pins of the target as shown in Figure 34. Short together the ground pins of the kits.

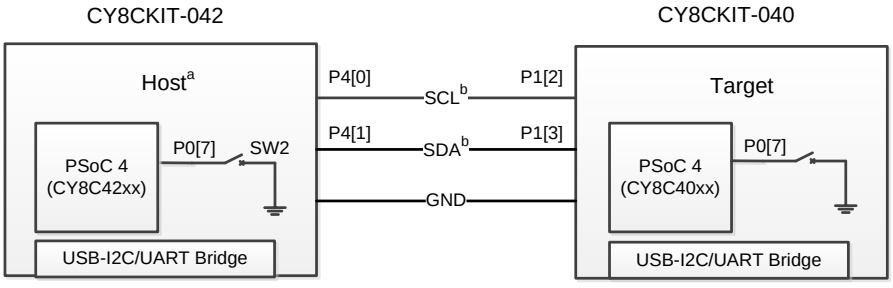
Figure 33 and Figure 34 illustrate these connections.

Figure 33. Host/Target Connections (CY8CKIT-042 target)



a. The Bootloading Status / Error Code is transferred to PC via USB-UART Bridge in CY8CKIT-042
b. P3[0]/P3[1] and P4[0]/P4[1] are pulled up with 2.2 KΩ resistors in CY8CKIT -042

Figure 34. Host/Target Connections (CY8CKIT-040 target)



a. The Bootloading Status / Error Code is transferred to PC via USB-UART Bridge in CY8CKIT-042
b. P1[2]/P1[3] and P4[0]/P4[1] are pulled up with 2.2 KΩ resistors in CY8CKIT-040 and CY8CKIT -042 respectively

3.2 Verifying the Results

After the kits are configured, you can test the example projects as follows:

- On the first button press (P0[7] for CY8CKIT-042) on the host kit, the *Bootloadable_GreenLED.cyacd* file is bootloaded to the target PSoC. On successful completion, the message “Bootloaded. LED blinks with green color on Target” is displayed on the serial port viewer, and the target green LED blinks.
- For subsequent bootloading operations, press the button (P0[7] for CY8CKIT-042) on the target kit. PSoC enters the bootloader when it detects the button press. The red LED blinks when the PSoC 4 is executing the bootloader.
- On the next button press on the host kit, the *Bootloadable_BlueLED.cyacd* file is bootloaded to the target PSoC. On successful bootloading, the message “Bootloaded. “Blue LED blinks on the target” is displayed on the serial port viewer, and the target blue LED blinks.

4 Summary

This application note explained how to bootload PSoC using I^2C as the communication interface. It also introduced the basic building blocks of a bootloader host and showed how to build an embedded I^2C bootloader host.

Bootloaders are a standard method for doing field upgrades. With PSoC Creator doing the entire configuration for you, it is easy to make a bootloader for PSoC.

For more advanced information, see the Appendix sections and the PSoC 4 Technical Reference Manual.

5 Related Application Notes

To better understand bootloaders and flash programming, refer to the following application notes:

- AN73854 – PSoC 3, PSoC 4, and PSoC 5LP Introduction to Bootloaders
- AN60317 – PSoC 3 and PSoC 5LP I^2C Bootloader
- AN68272 – PSoC 3, PSoC 4, and PSoC 5LP UART Bootloader
- AN84858 – PSoC 4 Programming Using an External Microcontroller (HSSP)
- AN61290 – PSoC 3 and PSoC 5LP Hardware Design Considerations
- AN79953 – Getting Started with PSoC 4

To learn more about the many other features and capabilities of PSoC 4, click here for a complete list of application notes.

6 Related Projects

The projects attached to this application note are organized as Table 5 shows.

Table 5. Projects Attached to This Application Note

Design Project Name	Description
I2C_Bootloader_Red	This project demonstrates how to create an I2C bootloader project for PSoC using PSoC Creator. This bootloader blinks the red LED on the CY8CKIT-042* kit.
Bootloadable_Green	This project demonstrates how to create bootloadable projects for PSoC using PSoC Creator. This application blinks the green LED on the CY8CKIT-042* kit.
Bootloadable_Blue	This project demonstrates how to create bootloadable projects and how to enter the bootloader from a bootloadable project. This application blinks the blue LED on the CY8CKIT-042* kit.
I2C_Bootloader_Host	This is an example bootloader host program demonstrating a PSoC bootloading another PSoC.

*The projects can be easily adapted to work with other kits.

7 PSoC Resources

Cypress provides a wealth of data at www.cypress.com to help you to select the right PSoC device for your design, and to help you to quickly and effectively integrate the device into your design. For a comprehensive list of resources, see KBA86521, How to Design with PSoC 3, PSoC 4, and PSoC 5LP. The following is an abbreviated list for PSoC 4:

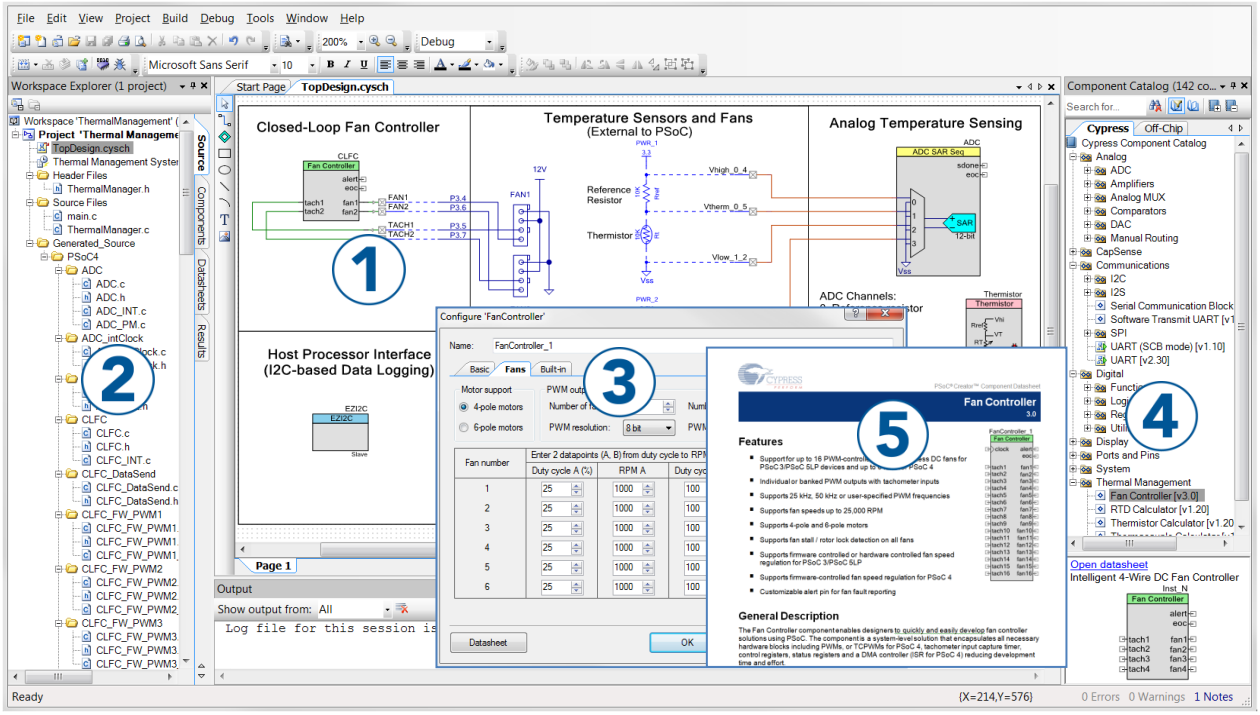
- **Overview:** PSoC Portfolio, PSoC Roadmap
- **Product Selectors:** PSoC 1, PSoC 3, PSoC 4, or PSoC 5LP. In addition, PSoC Creator includes a device selection tool.
- Datasheets describe and provide electrical specifications for the PSoC 4000, PSoC 4000S, PSoC 4100, PSoC 4100S, PSoC 4100PS, PSoC 4100S Plus, PSoC 4200, PSoC 4xx7 BLE, PSoC 4200-M, PSoC 4200-L device families.
- **CapSense Design Guide:** Learn how to design capacitive touch-sensing applications with the PSoC 4 family of devices.
- **Application Notes and Code Examples** cover a broad range of topics, from basic to advanced level. Many of the application notes include code examples. PSoC Creator provides additional code examples – see Code Examples.
- **Technical Reference Manuals (TRM)** provide detailed descriptions of the architecture and registers in each PSoC 4 device family.
- **Development Kits:**
  - CY8CKIT-040, CY8CKIT-041, CY8CKIT-042, CY8CKIT-042-BLE, CY8CKIT-044, and CY8CKIT-046 PSoC 4 Pioneer Kits are easy-to-use and inexpensive development platforms. These kits include connectors for Arduino™ compatible shields and Digilent® Pmod™ daughter cards.
  - CY8CKIT-001 is a common development platform for all PSoC family devices.
  - CY8CKIT-147 and CY8CKIT-149 are very low-cost prototyping platforms for sampling PSoC 4 devices.
- The MiniProg3 device provides an interface for flash programming and debug.

7.1 PSoC Creator

PSoC Creator is a free Windows-based Integrated Design Environment (IDE). It enables concurrent hardware and firmware design of systems based on PSoC 3, PSoC 4, and PSoC 5LP (see Figure 35). With PSoC Creator, you can:

1. Drag and drop Components to build your hardware system design in the main design workspace
2. Codesign your application firmware with the PSoC hardware
3. Configure Components using configuration tools
4. Explore the library of 100+ Components
5. Review Component datasheets

Figure 35. PSoC Creator Features



7.2 Code Examples

PSoC Creator includes a large number of code example projects. These projects are available from the PSoC Creator Start Page, as Figure 36 shows.

Example projects can speed up your design process by starting you off with a complete design, instead of a blank page. The example projects also show how PSoC Creator Components can be used for various applications. Code examples and datasheets are included, as Figure 36 shows.

In the Find Example Project dialog shown in Figure 37, you have several options:

- Filter for examples based on architecture or device family, i.e., PSoC 3, PSoC 4, or PSoC 5LP; category; or keyword
- Select from the menu of examples offered based on the **Filter Options**
- Review the datasheet for the selection (on the **Documentation** tab)
- Review the code example for the selection. You can copy and paste code from this window to your project, which can help speed up code development, or
- Create a new project (and a new workspace if needed) based on the selection. This can speed up your design process by starting you off with a complete, basic design. You can then adapt that design to your application.

Figure 36. Code Examples in PSoC Creator

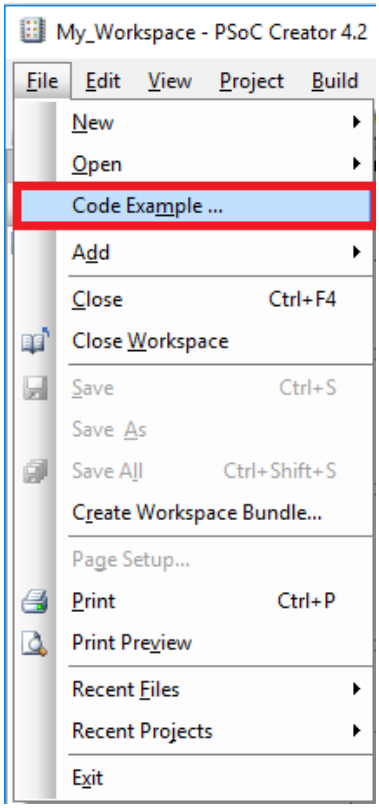
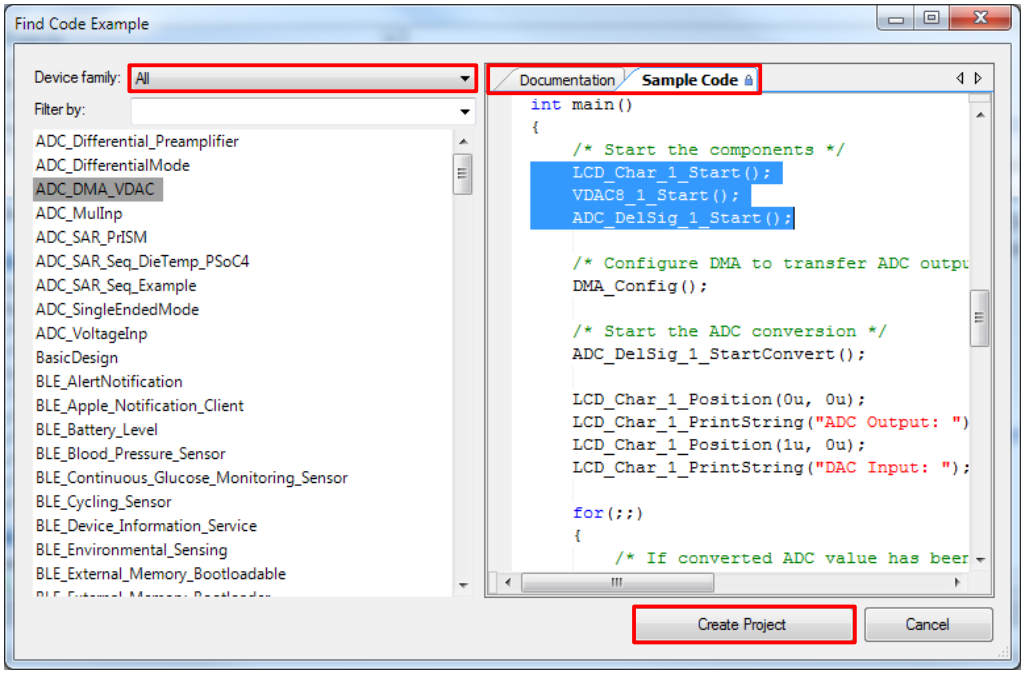


Figure 37. Code Example Projects, with Sample Code



7.3 PSoC Creator Help

Visit the PSoC Creator home page to download the latest version of PSoC Creator. Then, launch PSoC Creator and navigate to the following items:

- **Quick Start Guide**: Choose **Help** > **Documentation** > **Quick Start Guide**. This guide gives you the basics for developing PSoC Creator projects.
- **Simple Component example projects**: Choose **File** > **Open** > **Example projects**. These example projects demonstrate how to configure and use PSoC Creator Components.
- **Starter designs**: Choose **File > New** > **Project** > **PSoC 4 Starter Designs**. These starter designs demonstrate the unique features of PSoC 4.
- **System Reference Guide**: Choose **Help** > **System Reference** > **System Reference Guide**. This guide lists and describes the system functions provided by PSoC Creator.
- **Component datasheets**: Right-click a Component and select “Open Datasheet.” Visit the PSoC 4 Component Datasheets page for a list of all PSoC 4 Component datasheets.
- **Document Manager**: PSoC Creator provides a document manager to help you to easily find and review document resources. To open the document manager, choose the menu item **Help** > **Document Manager**.

7.4 Technical Support

If you have any questions, our technical support team is happy to assist you. You can create a support request on the Cypress Technical Support page.

If you are in the United States, you can talk to our technical support team by calling our toll-free number: +1-800-541-4736. Select option 8 at the prompt.

You can also use the following support resources if you need quick assistance.

- Self-help
- Local Sales Office Locations

About the Author

Name:	Charles Cheng.
Title:	Sr. Applications Engineer
Background:	Charles is an application engineer in the Cypress Semiconductor Programmable Systems Division focused on PSoC Applications.

A Appendix A: Memory

A.1 Flash Memory Details

Flash memory provides storage for firmware, bulk data, device configuration data, factory configuration data, and user-defined flash protection data. Figure 38 shows the physical organization of flash memory in PSoC.

PSoC flash memory is divided into blocks called “arrays.” Arrays are uniquely identified by array IDs. Each array has 128 or 256 rows of flash memory. Each row has 128 data bytes for the PSoC 4100, 4200, 4000S, and 4100PS family of devices and 256 data bytes for the 4100S family of device. Each row has 64 data bytes for the 4000 family of devices. So, an array can have 8 KB, 16 KB, 32 KB, or 64 KB for instruction and data storage.

The PSoC 4100, 4200, 4000S, and 4100PS family of devices have a maximum flash of 32 KB, so it has only one array, and the only valid array ID is 0. The PSoC 4100S devices have a maximum flash of 64 KB, also have only one array, and the only valid array ID is 0. The PSoC 4100S Plus devices have a maximum flash of 128 KB, and the only valid array ID is 0. The PSoC 4000 devices have a maximum flash of 16 KB, also have only one array, and the only valid array ID is 0.

Flash memory is programmed one row at a time. It can be erased a row at a time, or the entire flash can be erased at once. Rows are identified by a unique combination of the array ID and the row number.

Figure 38 also shows that the first X rows of flash are occupied by the bootloader. X is set such that there is enough space for the following:

- The vector table for the bootloader, starting at address 0
- The bootloader project configuration bytes
- The bootloader project code and data
- The checksum for the bootloader portion of the flash

For PSoC, the vector table contains the initial stack pointer (SP) value for the bootloader project and the address of the start of the bootloader project code. It also contains vectors for the exceptions and interrupts to be used by the bootloader.

The bootloadable project occupies the flash starting at the first 128-byte boundary after the bootloader. This region of the flash includes the following:

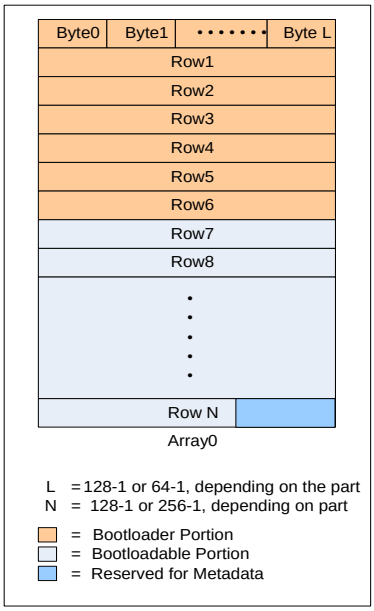
- The vector table for the bootloadable project
- The bootloadable project code and data

The highest 64-byte block of flash is used as a common area for both projects. Parameters saved in this block include:

- The entry address in flash of the bootloadable project (4-byte address)
- The amount of flash occupied by the bootloadable project (number of flash rows)
- The checksum for the bootloadable portion of flash (one byte)
- The size in bytes of the bootloadable portion of flash (4 bytes).

For more information on the location of metadata in the flash memory, see Metadata Layout in Flash.

Figure 38. Physical Organization of Flash Memory in PSoC

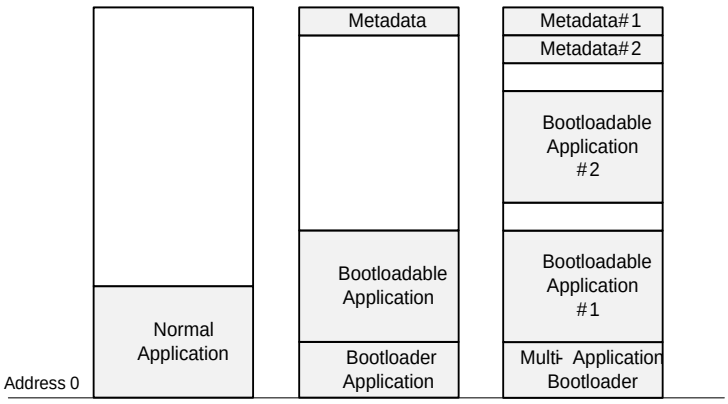


A.2 Memory Use in PSoC

There are two types of bootloader project types: standard bootloader and multi-application bootloader. The multi-application bootloader is useful for designs that require a guarantee that there is always a valid application that can be run. But this guarantee comes with a limitation that each application has only one-half of the flash available.

Figure 39 shows the flash memory use for each type of PSoC Creator project.

Figure 39. Flash Memory Use



A.3 Metadata Layout in Flash

The metadata section is the highest 64-byte block of flash and is used as a common area for both bootloader and bootloadable projects, as Figure 39 shows. Various parameters are stored in this block, as Table 6 shows. For the multi-application bootloader, there are two sets of metadata.

Table 6. Metadata Layout

Address	PSoC
0x00	App Checksum
0x01	Application Address
0x02	
0x03	
0x04	
0x05	Last Bootloader Row
0x06	
0x07	
0x08	
0x09	Application Length
0x0A	
0x0B	
0x0C	
0x0D	NA
0x0E	NA
0x0F	NA
0x10	Application Active
0x11	Application Verified
0x12	Bootloader Application Version
0x13	
0x14	Bootloadable Application ID
0x15	
0x16	Bootloadable Application Version
0x17	
0x18	Bootloadable Application Custom ID
0x19	
0x1A	
0x1B	
0x1C-0x3F	NA

Note: For the multi-application bootloader, the Last Bootloader Row for metadata (image 2) signifies the last row of bootloadable 1 in the flash section and not the bootloader row.

A.3.1 Flash Protection

If the bootloader code is invalid, it makes the product unusable. So, it is important to protect the bootloader portion of the flash from accidental overwrites.

PSoC protection includes chip-level protection (Open, Protected, and Kill) and flash-level protection. For more information, see the device datasheet or Technical Reference Manual (TRM). The flash protection feature is designed to prevent duplication and reverse engineering of proprietary code. But it can also be used to protect against inadvertent writes to the bootloader portion of flash.

Two protection levels are provided for flash-level protection, as Table 7 shows. Each row of flash can be configured to have a different protection level, which can be set using PSoC Creator (the Flash Security tab of the *.cydwr* file).

Table 7. Levels of Flash Protection for PSoC

Protection level	Allowed	Not Allowed
Unprotected	External read and write Internal read and write	
Full protection	Internal read External read	External write Internal write

Once the bootloader portion of the flash is configured to have a protection level of Full protection, it cannot be changed. The only way to alter the protection level or to change the bootloader code is to completely erase the flash and reprogram it using the SWD interface.

An example for protecting bootloader flash follows.

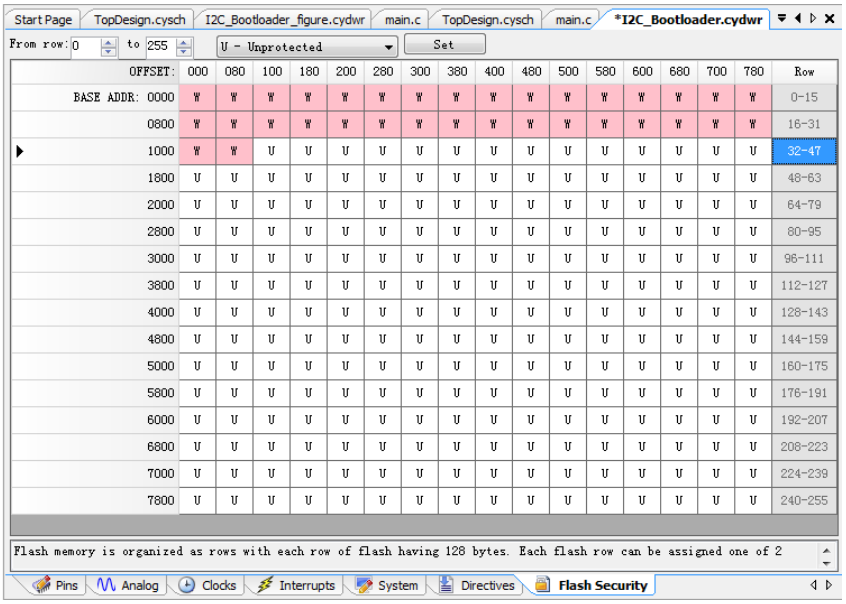
A.3.2 Example for Flash Protection

When the bootloader project is built, the PSoC Creator Output window shows the amount of flash used. For example, if the flash occupied by the I2C_Bootloader project is 4352 bytes, then the output is (for PSoC with 32 KB flash) as follows:

```
Flash used: 4352 of 32768 bytes (13.3%).
```

The bootloader thus occupies 34 rows of flash (4352 / 128), that is, flash locations 0x0000 to 0x10FF. Set the flash protection level to Full protection for these rows (under the Flash Security tab of the *.cydwr* file in PSoC Creator). The protection level for the remaining rows can be Unprotected (the default), as Figure 40 shows. For more information on how to use the Flash Protection dialog, see the PSoC Creator help article Flash Security Editor.

Figure 40. Flash Protection in PSoC Creator



B Appendix B: Project Files

B.1 Bootloadable Output Files

When any PSoC Creator project is built, an output file of type *.hex* is generated. This is the file that is downloaded to PSoC while programming using the SWD interface.

For a bootloadable project, the *.hex* file is a combined *.hex* file of both the bootloadable and the related bootloader project. This file is typically used to download both projects via SWD in a production environment.

B.1.1 **.cyacd* File Format

When a bootloadable project is built, an additional file of type *.cyacd* (application code and data) is also generated. This file contains a header followed by lines of flash data. Excluding the header, each line in the file represents an entire row of flash data. The data is stored as ASCII data in big-endian format. Hence, while bootloading, the contents of this file must be parsed (converted from ASCII to hex). Parsing is not required for programming a file of type *.hex*.

The header of this file has the following format:

```
[4 bytes Silicon ID] [1 byte Silicon rev] [1 byte checksum type]
```

The flash lines have the following format:

```
[1 byte array ID] [2 bytes row number] [2 bytes data length] [N bytes of data] [1 byte checksum]
```

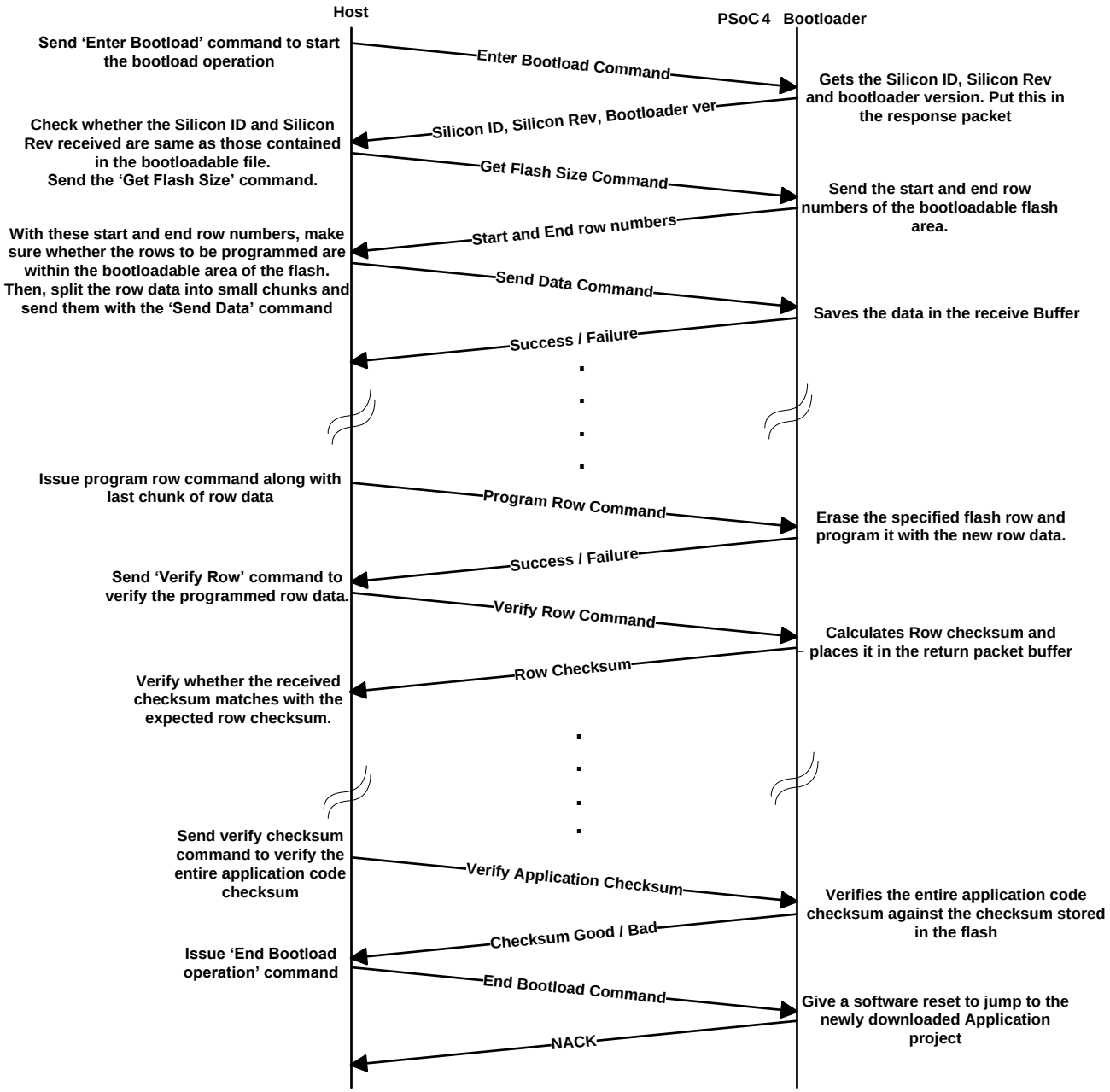
The checksum type in the header indicates the type of checksum used in the packets sent between the bootloader and the bootloader host during the bootloading operation. If this byte is 0, the checksum is a basic summation. If it is 1, the checksum is CRC-16.

C Appendix C: Host/Target Communications

C.1 Communication Flow

The Bootloader Function Flow section described the operation of a bootloader in PSoC, and the I2C Bootloader Host section introduced the building blocks of a bootloader host. With this background, Figure 41 explains the flow of communication between the host and the target during a bootloading operation. It gives the order in which commands are issued to the target and responses are received. See Command and Status/Error Codes for a complete list of bootload commands, their codes, and their expected responses.

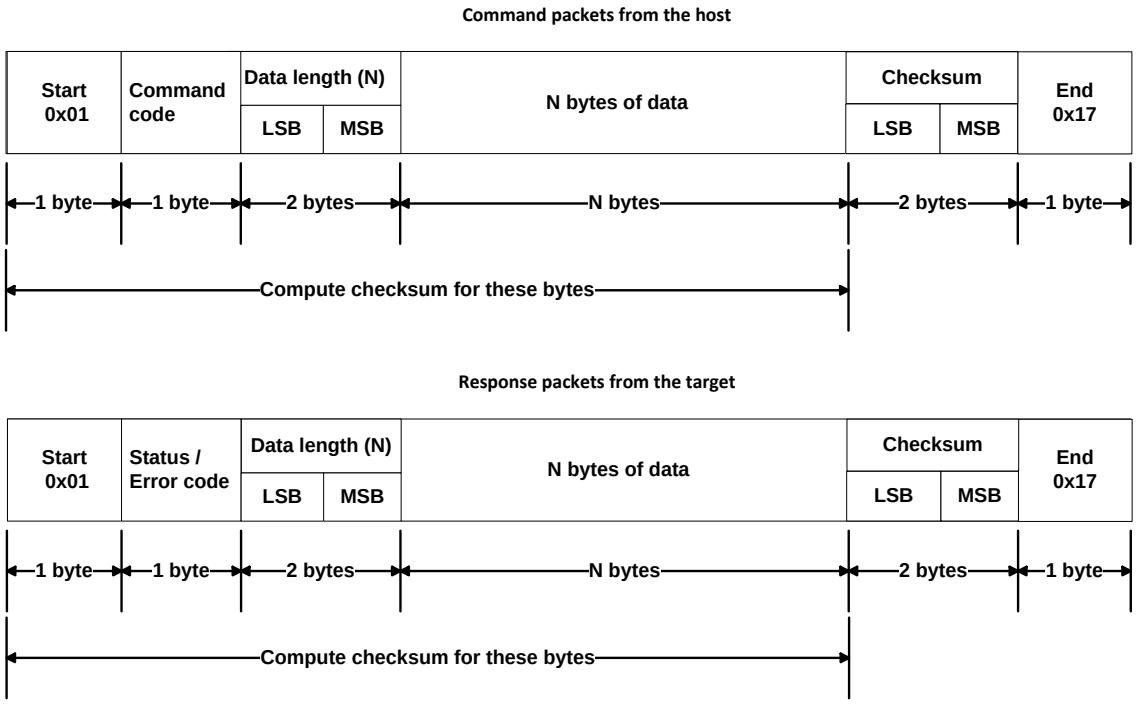
Figure 41. Communication Flow During Bootloading



C.2 Protocol Packet Format

The bootloading operation involves an exchange of command and response packets between the host and the target. These packets have specific formats, as Figure 42 shows.

Figure 42. Bootloading Packet Format



Each packet includes checksum bytes. The checksum can be a basic summation (2's complement) or CRC-16 depending on the bootloader project setting. When sending multibyte data such as Data Length and Checksum, the least significant byte is sent first.

The bootloader responds to each command from the host with a response packet. The format of the response packet is similar to the command packet except that there is status/error code instead of command code. Table 8 on page 35 gives the important commands and data bytes and the bootloader response packet data.

C.3 I²C Transaction Information for Bootloader Host

The I²C transaction is a lower level of communication between the two devices. Figure 43 and Figure 44 give the format for reading and writing to PSoC from the host.

Figure 43. Transaction Sent from Host to Bootloader Through I²C Interface (Write)

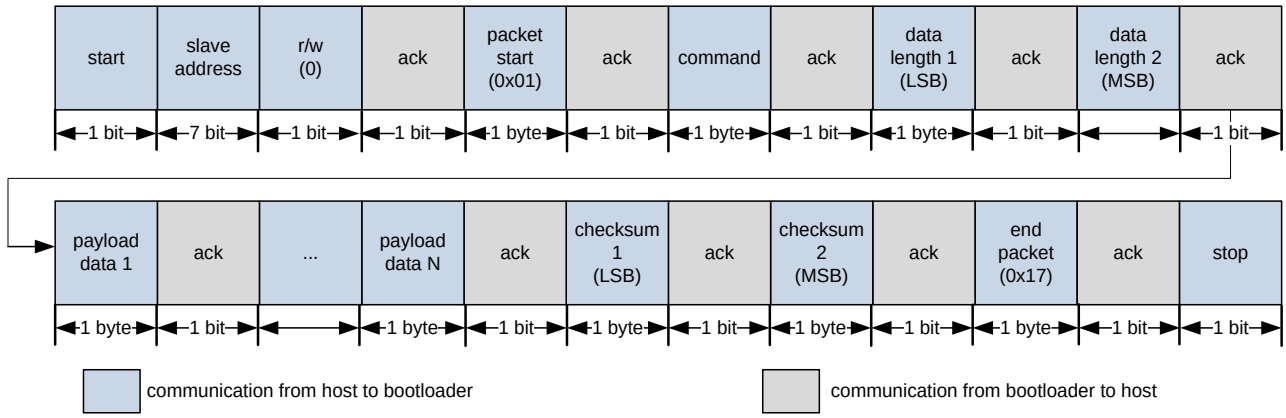
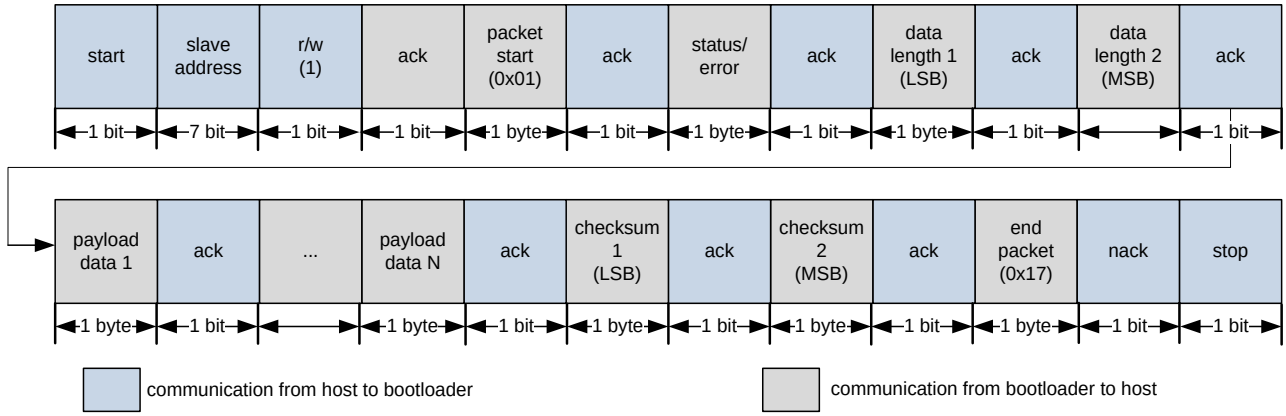


Figure 44. Transaction Received from Bootloader Through I²C Interface (Read)



C.3.1 Command and Status/Error Codes

As the previous section explains, the command and response packet structures are similar. The only difference is that the second byte contains a command code or a status/error code.

Table 8 provides a list of commands and their expected responses. Table 9 on page 36 provides a list of status and error codes.

Table 8. Bootloading Commands

Command Byte	Command	Data Byte in the Command Packet	Expected Response Data Bytes
0x31	Verify Checksum	N/A	1 byte: Non-zero or '0'. If it is a non-zero byte, then the application checksum matches and it is a valid application. If it is a zero byte, then the checksum is bad and the application is invalid.
0x32	Get Flash Size	Flash array ID, 1 byte	First row number of the bootloadable flash, 2 bytes. Last row number of the bootloadable flash, 2 bytes. These numbers are for requested array ID.
0x33	Get Application Status (valid only for multi-application bootloader)	Application number, 1 byte	Valid application number, 1 byte. Active application number, 1 byte. Checks whether the specified application is valid and if it is active.
0x34	Erase Row	Flash array ID, 1 byte Flash row number, 2 bytes	N/A. Erases the contents of the specified flash row.
0x35	Sync Bootloader	N/A	N/A. Resets the bootloader to a clean state. Any data that was buffered in will be thrown out. This command is needed only if the bootloader and the host become out of sync with each other.
0x36	Set Active Application (valid only for multi-application bootloader)	Application number, 1 byte	N/A. Sets the specified application as active.
0x37	Send Data	N bytes of data to be sent	N/A. The received data bytes will be buffered by the bootloader in anticipation of the Program Row command.
0x38	Enter Bootloader	N/A	Silicon ID, 4 bytes. Silicon Rev, 1 byte. Bootloader version, 3 bytes. All the commands are ignored until this command is received.
0x39	Program Row	Flash array ID, 1 byte Flash row number, 2 bytes N bytes of data to be sent	N/A. After sending multiple bytes of data to the bootloader using the send data command, the last chunk of data is sent along with this command.
0x3A	Verify Row	Flash array ID, 1 byte Flash row number, 2 bytes	Row checksum, 1 byte. Returns the checksum of the specified row.
0x3B	Exit Bootloader	N/A	N/A. This command is not acknowledged.

Table 9. Bootloading Status/Error Codes – Possible Responses to Commands

Status/Error Codes	Label	Description
0x00	CYRET_SUCCESS	The command was successfully received and executed.
0x02	BOOTLOADER_ERR_VERIFY	The verification of flash failed.
0x03	BOOTLOADER_ERR_LENGTH	The amount of data available is outside the expected range.
0x04	BOOTLOADER_ERR_DATA	The data is not of the proper form.
0x05	BOOTLOADER_ERR_CMD	The command is not recognized.
0x06	BOOTLOADER_ERR_DEVICE	The expected device does not match the detected device.
0x07	BOOTLOADER_ERR_VERSION	The bootloader version detected is not supported.
0x08	BOOTLOADER_ERR_CHECKSUM	The checksum does not match the expected value.
0x09	BOOTLOADER_ERR_ARRAY	The flash array ID is not valid.
0x0A	BOOTLOADER_ERR_ROW	The flash row number is not valid.
0x0C	BOOTLOADER_ERR_APP	The application is not valid and cannot be set as active.
0x0D	BOOTLOADER_ERR_ACTIVE	The application is currently marked as active.
0x0F	BOOTLOADER_ERR_UNK	An unknown error occurred.

D Appendix D: Host Core APIs

D.1 cybtldr_api2.c / .h

This is a higher-level API that handles the entire bootload operation. It has functions to open and close files. It invokes the functions of the *cybtldr_api.c / .h* API for the bootload operations. This API can be used when building a GUI-based bootloader host.

D.2 cybtldr_parse.c / .h

This module handles the parsing of the *.cyacd* file that contains the bootloadable image to send to the device. It also has functions for setting up access to the file, reading the header, reading the row data, and closing the file.

D.3 cybtldr_api.c / .h

This is a row-level API file for sending a single row of data at a time to the bootloader target. It has functions for setting up the bootload operation, erasing a row, programming a row, verifying a row, and ending the bootload operation. Table 10 describes in detail the functions of this API file.

Table 10. Functions of *cybtldr_api.c /.h*

Function	Description
`CyBtldr_StartBootloadOperation`	Enables the communication interface and sends an Enter Bootloader command to the target. From the response packet received, verifies the silicon ID, silicon revision of the target device, and bootloader version.
`CyBtldr_ProgramRow`	First validates a row, that is, sends a Get Flash Size command to the target for a particular array ID of the target flash. In response to this, the target returns the start and end row numbers of the bootloadable flash portion in that array. The host reads this response and checks whether the specified row is in the bootloadable area of the flash. If row validation is a success, the host breaks the row data into smaller pieces and sends them to the target using Send Data commands. Along with the last portion of row data, sends a Program Row command to the target.
`CyBtldr_VerifyRow`	This function also first validates a row for a particular array ID and row number. If row validation is successful, sends a Verify Row command for the validated flash row. In response to this command, the target returns the checksum of the row. The returned checksum is verified against the expected checksum value.
`CyBtldr_EraseRow`	This function also first validates a row for a particular array ID and row number. If row validation is successful, sends an Erase Row command for the validated flash row.
`CyBtldr_EndBootloadOperation`	Sends an Exit Bootload command and disables the communication interface.

D.4 cybtldr_command.c / .h

This API handles the construction of command packets to the target and parsing of the response packets received from the target. The *cybtldr_api.c / .h* invokes the functions of this API. For example, to send an Enter Bootload command, `CyBtldr_StartBootloadOperation()` calls the `CyBtldr_CreateEnterBootloadCmd()` function of this API. It also has a function for calculating the checksum of the command packets before sending to the target.

E Appendix E: Miscellaneous Topics

E.1 Bootloader Versus HSSP

The bootloader allows your system firmware to be upgraded over a communication interface. But for a complete flash upgrade, including the bootloader flash area, you must use the SWD programmer (Host Sourced Serial Programming). To create HSSP for PSoC 4, see AN84858, PSoC 4 Programming Using an External Microcontroller (HSSP).

E.2 What Happens If Power Fails During the Bootload Operation?

If power fails during the bootload operation, then at the next reset the checksum of the bootloadable project does not match the expected value (the bootloadable project's checksum stored in the last row of flash), and the bootloadable project is considered to be invalid. Program execution remains in the bootloader until a successful bootload happens. The bootloader host must send a Start Bootload command to restart the bootload operations.

E.3 Why Do I Need a Reset to Jump Between the Bootloader and the Bootloadable Projects?

PSoC is an enormously configurable device. The bootloader allows you to change on-chip hardware resources as well as firmware. Due to its highly configurable architecture, hardware reconfiguration (placement, routing, functional) is possible only from a reset state. Therefore, the bootloader requires a reset to jump between the bootloader and bootloadable projects.

E.4 Converting a Normal Application Project to a Bootloadable Project

If you have already created a standard (Normal) project and want to convert it to a bootloadable project, add a Bootloadable Component to the top design and add the bootloader project's *.hex* file as a dependency, as Figure 11 on page 9 shows.

If a project is created as a normal project and then later changed to a bootloader project by changing the application type to **Bootloader**, you should insert the `Bootloader_Start()` function call in *main.c* for the bootloader project to work as expected.

Note: For PSoC creator 3.1, if you want to convert a standard (Normal) project to a bootloadable/bootloader project, change the application type of the project to Bootloadable/Bootloader. To do this, right-click on **Project** > **Build setting** > **Code Generation** > **General tab** and change Application Type. In addition, add the Bootloadable/Bootloader Component to the top design.

E.5 Debugging Bootloadable Projects

In the PSoC Creator bootloader system, the bootloader project executes first followed by the bootloadable project. The jump from the bootloader to the bootloadable project is done through a software-controlled device reset. This resets the debugger interface, which means that the bootloadable project cannot be run in debugger mode.

To debug a bootloadable project, convert it to Application Type Normal, debug it, and then convert it back to Bootloadable after debugging is done.

Another option is to program the bootloadable project *.hex* file onto the device and then use the "Attach to running target" option for debugging while the bootloadable project is running. In this case, you can debug the bootloadable project only from the point where debugger is attached to the device.

E.6 Multi-Application Bootloader

A Multi-Application Bootloader (MABL) is used to put two bootloadable applications in flash simultaneously. The two applications can be the same to ensure that there is always a valid application in the device's flash. Or, the two applications can be different so that they can be switched using bootloader commands. This functionality comes with the obvious limitation that each application has one-half of the available flash memory. Figure 39 shows the project placement in flash memory for a MABL.

You can implement a MABL by following these steps, which are different from the standard bootloader application:

1. Create a new MABL bootloader project. Check the Multi-App Bootloader checkbox in the Bootloader configuration window.

Note: For PSoC Creator 3.1, set the application type as Multi-App Bootloader.

2. Add two bootloadable projects to the workspace, say, Project_A and Project_B. For each project, add a dependency to the MABL project. Two *.cyacd* files are generated for each project—one for the lower part of flash and one for the upper part of flash (see Figure 39):
   - *Project_A_1.cyacd* and *Project_A_2.cyacd*
   - *Project_B_1.cyacd* and *Project_B_2.cyacd*
3. The *.cyacd* file with suffix 1 always occupies the first half of flash, and the *.cyacd* file with suffix 2 occupies the second half. Thus only certain combinations of the *.cyacd* file can be used. These combinations are as follows:
   - *Project_A_1.cyacd* and *Project_A_2.cyacd*
   - *Project_B_1.cyacd* and *Project_B_2.cyacd*
   - *Project_A_1.cyacd* and *Project_B_2.cyacd*
   - *Project_B_1.cyacd* and *Project_A_2.cyacd*
4. Program the device with the multi-application bootloader project and bootload the applications (*.cyacd* files) sequentially, in one of the above combinations, using the Bootloader Host application.
5. To switch between applications, follow these steps:
   - Connect the PSoC kit to the PC through the USB cable. Make sure that the bootloader is active.
   - Open the Bridge Control Panel by navigating to **Start** > **All Programs** > **Cypress** > **Bridge Control Panel**. Select the KitProg for I^{2}C Protocol.
   - Send command 0x38 to enter the bootloader:
     w 08 01 38 00 00 C7 FF 17
     r 08 x x x x x x x x x x x x x x x x
   - To switch from application_1 to application_2, send the set_active_application command (0x36):
     w 08 01 36 01 00 01 C7 FF 17
     r 08 x x x x x x x
   - To switch from application_2 to application_1, send the set_active_application command (0x36):
     w 08 01 36 01 00 00 C8 FF 17
     r 08 x x x x x xx
   - Send the exit_bootloader command (0x3B) to launch the application:
     w 08 01 3B 00 00 C4 FF 17
     r 08 x x x x x x x

Table 11 explains these commands using the example of the set_active_application command (application_1 to application_2).

Table 11. Command Bytes

Bytes	1	1	1	2	N	2	1
Value	08	01	36	01 00	01	C7FF	17
Description	slave address	packet start byte	command	number of bytes to follow (LSB first)	data bytes	checksum (LSB first)	packet end byte

7.4.1 Memory Requirement for Bootloader

A typical I²C bootloader project with all the optional commands included occupies approximately 4.3 KB of PSoC flash with Arm GCC compiler optimization set to "size". You can find the memory used by the bootloader project in the output window when you build the project. RAM memory used by the bootloader project can be reused by the bootloadable project.

The memory use of a bootloader project can be reduced a small amount by removing the optional commands supported by the Bootloader Component, as Figure 45 shows.

Set the **Device Configuration Mode** to **Compressed** in the *.cydwr* **> System** tab, as Figure 46 shows, to minimize flash memory use.

Figure 45. Deselecting Optional Commands in Bootloader Component

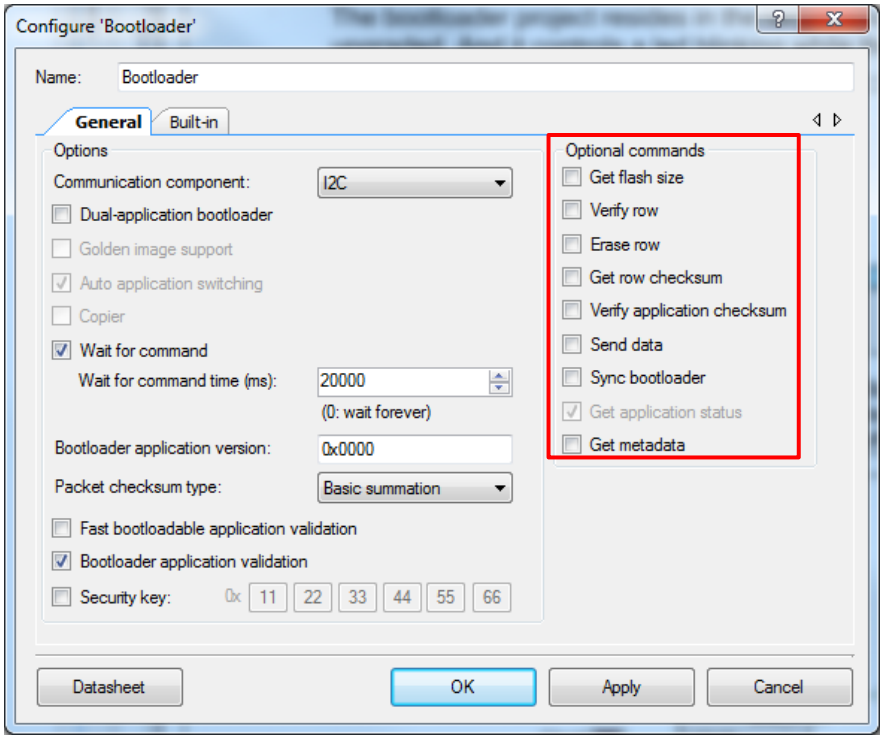
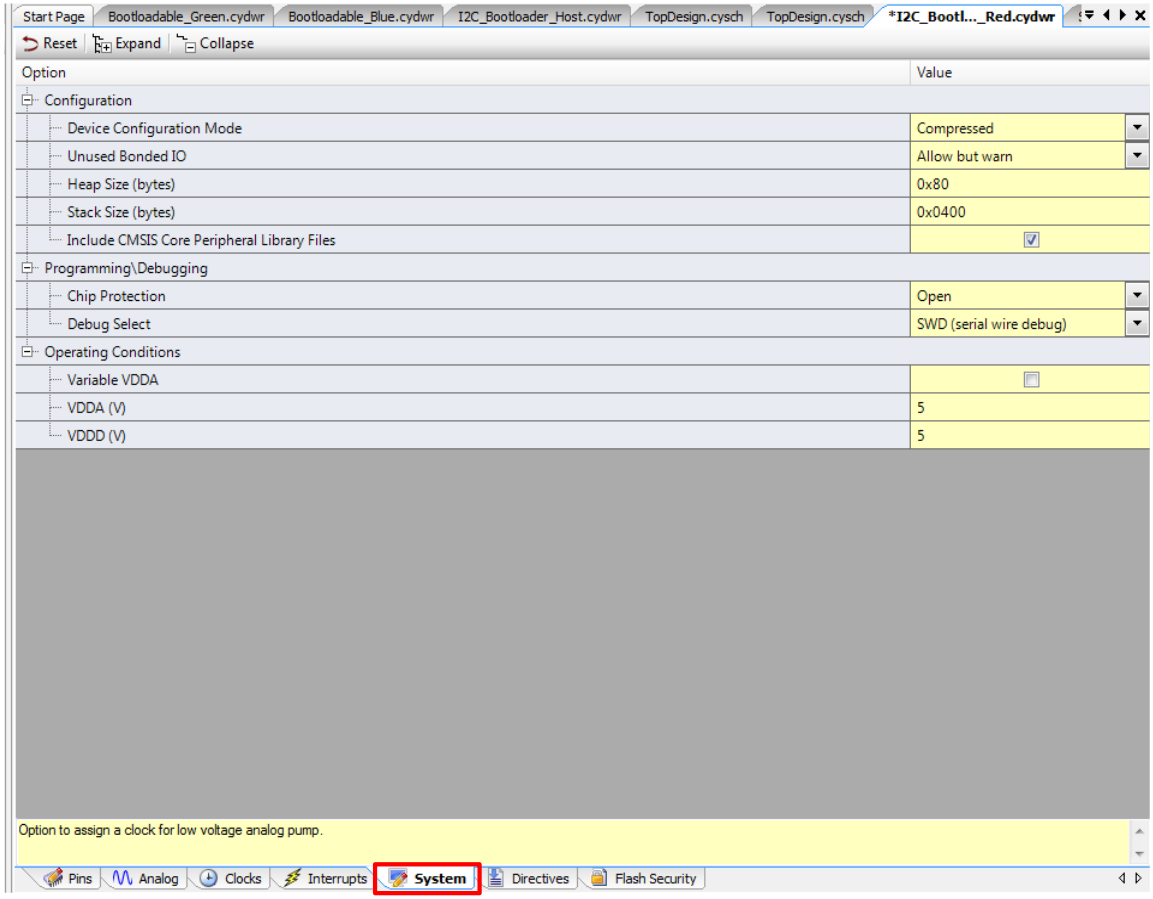


Figure 46. Device Configuration Mode



7.4.2 PSoC Kits versus MiniProg3 for PSoC

PSoC 4 kits and MiniProg3 support debug/program and the USB-I^{2}C bridge for PSoC. In addition, PSoC also supports a USB-UART bridge, which MiniProg3 does not support. Therefore, it is easy to debug/program and bootload using the PSoC kits. For more information, see the CY8CKIT-040 Kit Guide and CY8CKIT-042 PSoC Pioneer Kit Guide.

F Appendix F- Kit Selection

Target Device	Kit Name	User Guide
CY8C42xx	CY8CKIT-042 PSoC 4 Pioneer Kit	CY8CKIT-042
CY8C40xx-S	CY8CKIT-041-40XX PSoC 4 S-Series Pioneer Kit	CY8CKIT-041
CY8C41xx-S	CY8CKIT-041-41XX PSoC 4100S CapSense Pioneer Kit	CY8CKIT-041
CY8C40xx	CY8CKIT-040 Pioneer Kit	CY8CKIT-040
CY8C41xx-PS	CY8CKIT-147 PSoC 4100PS Prototyping Kit	CY8CKIT-147
CY8C4100S Plus	CY8CKIT-149 PSoC 4100S Plus Prototyping Kit	CY8CKIT-149

Document History

Document Title: AN86526 - PSoC 4 I2C Bootloader

Document Number: 001-86526

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	4027459	CLSC	06/13/2013	New application note.
*A	4290409	RNJT	02/24/2014	Updated for CY8C40xx family of devices.
*B	4339775	RNJT	04/10/2014	Projects updated for PSoC Creator 3.0 SP1.
*C	4586160	CLSC	12/10/2014	Updated Table 6. Metadata Layout. Updated projects for PSoC Creator 3.0 SP2
*D	5155612	AKSM	03/21/2016	Updated for PSoC 4000S/4100S and PSoC Analog Coprocess or family of devices. Updated figures and projects for PSoC Creator 3.3 SP2.
*E	5366705	RLIU	07/22/2016	Updated template
*F	5701881	BENV	04/19/2017	Updated logo and copyright
*G	5955192	TAVA/JSLN	04/17/2018	Updated template. Updated for PSoC 4100S Plus devices. Updated figures and information for PSoC Creator 4.2. Added the PSoC 4100S Plus information in Flash Memory Details. Changed ARM_GCC_493 to ARM_GCC_541. Updated Figure 4, Figure 11, Figure 20, and Figure 21. Added PSoC 4100S Plus in Appendix F- Kit Selection. Updated for PSoC 4100PS and removed PSoC Analog Coprocessor. Updated projects to PSoC Creator 4.2.

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

Forums | WICED IOT Forums| Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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