



I2C Bootloader for PSoC™, 16-Byte Packet Transfer

Author: Ernie Buterbaugh

Associated Project: Yes

Associated Part Family: CY8C21xxx, CY8C24xxxA, CY8C27xxx, CY8C29xxx

PSoC Designer Version: 4.2 + SP1

Associated Application Notes: AN2273a

Abstract

This Application Note describes a bootloader for the PSoC™ device using I2C communication and 16-byte packet transfer.

Introduction

The PSoC device already provides an in-system programming interface that allows developers to download code into the device. However, it can be convenient for developers to update the code within the PSoC device using a standard system interface such as I2C. This Application Note describes the code that implements an I2C bootloader, which allows developers to load code into the PSoC device.

The I2C bootloader requires the use of the I2C Hardware User Module. It does not preclude the use of the I2C bus for other functions within the PSoC device. In fact, the I2C bootloader uses a separate I2C address for its associated functions. All of the code for the I2C bootloader is programmed in a protected area of EEPROM and cannot be accidentally overwritten.

Because the I2C bootloader requires the use of the I2C Hardware User Module, the PSoC device must be programmed with initial code containing the I2C bootloader functions prior to using the bootloader function.

Quick Start Guide

There are several steps that must be followed to place the I2C bootloader into a project. The following is a quick start guide for creating the project. The remaining sections of this Application Note discuss each of these steps in more detail.

1. Create a PSoC project using PSoC Designer and instantiate the I2C Hardware User Module.
2. Modify the *boot.tpl* file with the new interrupt vector map found in the I2C bootloader *boot.tpl* template. (Ensure the proper interrupt vector map is used for the selected PSoC device.)
3. Generate the application.
4. Modify the *flashsecurity.txt* file to allow writing to Flash blocks (e.g., unprotect blocks 2 through 223 for a 16K device). See example projects for a template.
5. Add the *i2cbootloader.asm* and *i2cbootloader.inc* files into the project.
6. Set the bootloader I2C address, Flash size, and other options in the *i2cbootloader.inc* file.
7. Under the Project >> Settings >> Linker menu set the transferable code start address at 150 or higher to accommodate the new ISR table. (If address conflicts occur during build, simply increase the address past the conflicting address.)
8. In the *main.c* or *main.asm* file, enable the PSoC global interrupt, I2C interrupt, and slave module.
9. Add any user code or user modules, compile, and calculate the 16-bit image checksum. (A bootloader utility that aids in the calculation of the checksums is described later.)
10. Add an absolute address for setting the image checksum in *main.c* or *main.asm* at location 3FFEh (for a 16K device).
11. Re-compile and test your project.

Bootloader Operation

The bootloader is located in the last 2K section of Flash memory (starting address 3800h for a 16K device). This memory space is write-protected to prevent any accidental modification or corruption. The reset vector is modified so that when the processor is reset, the bootloader is executed.

The following operations are carried out by the bootloader:

- Upon reset, the bootloader calculates checksum for the Flash user code and verifies it with a checksum written to the last two bytes of Flash memory. If the checksum matches, the previous programming attempt was successful and the bootloader branches to the beginning of the user code and the user code can execute.
- If the checksum does not match, the bootloader executes customizable user code to perform system critical tasks (such as turning on a fan) and then enters the bootloader mode, where it waits for a 10-byte bootloader key from the master. If the previous bootloading failed (such as a power transient), the program enters the bootloader mode due to a mismatch checksum.
- Upon receiving a valid bootloader key from the master, the bootloader responds with a status byte informing the master that it is ready to receive the Flash image.
- The master sends the updated user code in 16-byte packets with some encoding bytes (explained in a later section).
- The bootloader writes the user code to the Flash. When all the Flash pages are written successfully, the bootloader performs a Flash verify operation and then a software reset to start the user code. **Note:** The I2C master needs to wait 100 ms after each block write before reading the block status byte in order to allow the Flash block write operation.

The bootloader will be entered at power-on if the image checksum in the last location of Flash memory does not match the calculated Flash checksum. If the checksum matches, user code in *main.c* (or *main.asm*) is executed. Anytime during normal operation, the bootloader can be entered by addressing the I2C bootloader and sending the correct 10-byte authentication key. If successful, the bootloader will be entered. If the key is incorrect, the PSoC device will reset and execution will be transferred back to *main.c* (or *main.asm*). The I2C bootloader Read Status command can be executed at any time. While in the bootloader mode, only the bootloader I2C addresses will be serviced. All other addresses, including the user's I2C address, will be NAK'ed. **It is strongly recommended that a new application be tested to ensure bootloader compatibility before bootloading the code into the PSoC device.**

Memory Map

A memory map of a 16K Flash device is shown in Figure 1.

The bootloader resides in the last 2K section of Flash memory. This is done to so that user code can be developed with minimal modifications to the project settings.

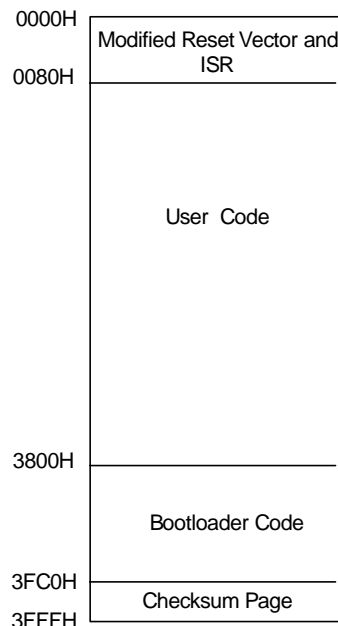


Figure 1. Bootloader Memory Map in 16K Device

```

; 0 40 80 C0 100 140 180 1C0 200 240 280 2C0 300 340 380 3C0 (+) Base Address
W W R R R R R R R R R R R R R R ; Base Address 0
R R R R R R R R R R R R R R R R ; Base Address 400
R R R R R R R R R R R R R R R R ; Base Address 800
R R R R R R R R R R R R R R R R ; Base Address C00
; End 4K parts
R R R R R R R R R R R R R R R R ; Base Address 1000
R R R R R R R R R R R R R R R R ; Base Address 1400
R R R R R R R R R R R R R R R R ; Base Address 1800
R R R R R R R R R R R R R R R R ; Base Address 1C00
; End 8K parts
R R R R R R R R R R R R R R R R ; Base Address 2000
R R R R R R R R R R R R R R R R ; Base Address 2400
R R R R R R R R R R R R R R R R ; Base Address 2800
R R R R R R R R R R R R R R R R ; Base Address 2C00
R R R R R R R R R R R R R R R R ; Base Address 3000
R R R R R R R R R R R R R R R R ; Base Address 3400
W W W W W W W W W W W W W W W W ; Base Address 3800
W W W W W W W W W W W W W W W R ; Base Address 3C00
; End 16K parts

```

Figure 2. flashsecurity.txt for I2C Bootloader

The following modifications are needed in the *flashsecurity.txt* file.

- All Flash memory blocks that contain the I2C bootloader are write protected and have a setting **W**. For example, a 16K device has memory blocks 224 to 254 protected. See Figure 2.
- The first two blocks that contain the reset vectors are also write protected with **W**. This is done to protect the reset vector and I2C interrupt vector from being corrupted.
- The last block contains the Flash image checksum. The 16-bit checksum is physically in the last two bytes of memory (block 255 and addresses 3FFEh and 3FFFh for a 16K device). This block is set to Field Upgrade mode **R** or Unprotect mode **U**.
- The protection level for all other blocks is set to Field Upgrade mode **R** or Unprotect mode **U**.

As the first two pages are write protected, the interrupt vectors in these pages are to be mapped to another unprotected location. Using the I2C bootloader *boot.tpl* template, the vector table can be replaced with the new table. The address of `__Start` is changed from 68h to C0h to accommodate the new ISR table. The section of the *boot.tpl* starting with the statement 'org 0' through the statement 'org C0h' must be copied into your project *boot.tpl*. Although the *boot.tpl* template in the I2C bootloader example project is a full and working *boot.tpl* file, you should not replace the entire file. Only replace the vector table (see the snip labels in the file).

Warning: Ensure that the ISR vector tables that you copy into the *boot.tpl* file have the same types of ISR entries. If you copy in a template with ISR vectors that are not in the device, indeterminate operation may occur at runtime.

Detailed Sequence of Operation

- Upon reset, the first operation performed is the `bootLoaderVerify`. Here, the checksum of the entire user code (from 0080h to 37FFh in the 16K device) is compared with the checksum stored in the last two locations of Flash memory (3FFEh and 3FFFh for a 16K device). If the checksum matches, the previous bootloading operation was successful and the program branches to `__Start`, from where the *main.c* (or *main.asm*) code executes.
- If the checksum does not match, it means either no user code is present, or the previous loading operation failed. If this condition occurs, the program enters bootloader mode, where it waits for a 10-byte bootloader key from the master.
- Upon receiving a valid 10-byte bootloader key, the program waits for the status byte to be read. Between every I2C transaction, a timeout variable is decremented. When the I2C master does not send a data packet or read status within the timeout period, the PSoC device is reset. There is no timeout while waiting for the bootloader key in bootloader mode caused by a failed boot.

- When a 14-byte packet is received from the master, the operation indicated by the bootloader command (byte number 2 of the 14-byte packet) is performed on the data. While the requested operation is being performed, the I2C clock is stalled to prevent the master from writing any further data. **Note:** The 14-byte packet can be padded with zeroes to create a 16-byte packet.
- For a write or verify, the I2C master sends a 14-byte packet indicating a write/verify command, the key sequence, the memory block, and the checksum. This is followed by four 16-byte data blocks, which is the data targeted for the Flash block. A read status occurs after the fourth byte write transfer. This sequence continues for every block including the last block (block 255 for a 16K device) with the image checksum.
- When the master sends the command 'Perform Flash Verify and Exit', the bootloader checks the Flash image checksum and compares it with the calculated checksum. The master reads the status byte one last time after which the bootloader resets the PSoC device. Note that the Flash Verify operation can take 750 ms on a 32K Flash device (12 MHz CPU). The I2C master should wait this amount of time before reading the last status byte.

10-Byte Bootloader Key

Table 1 shows the composition of the 10-byte bootloader authentication key (it can be 16 bytes in length by adding 6 bytes of 0s at the end after KEY8).

Table 1. 10-Byte Bootloader Authentication Key

Bootloader Mode (FFh)
Bootloader Command (38h, 3Bh)
KEY1
KEY2
KEY3
KEY4
KEY5
KEY6
KEY7
KEY8

14-Byte Bootloader Key

Table 2 shows the 14-byte bootloader command (it can be 16 bytes in length by adding 2 bytes of 0s at the end after the checksum).

Table 2. 14-Byte Bootloader Command

Bootloader Mode (FFh)
Bootloader Command (39h, 3Ah)
KEY1
KEY2
KEY3
KEY4
KEY5
KEY6
KEY7
KEY8
BlockNumber (msb)
BlockNumber (lsb)
Checksum of 64-Byte Data
Checksum (Preceding 13 Bytes)

The first byte is always FFh. The second byte is the bootloader command. Following are the commands available:

- 38h – Enter Bootloader Mode
- 39h – Perform Block Write Operation
- 3Ah – Perform Block Verify Operation
- 3Bh – Perform Image Verify and Exit

Commands 39h, 3Ah and 3Bh are only valid when in bootloader mode (set upon receiving a valid 38h command) as indicated by bit 5 in the status byte. Commands 38h and 3Bh are 10-byte bootloader key lengths. Commands 39h and 3Ah are 14 bytes in length and contain the 16-bit block number followed by the checksum of all 64 bytes to be written to the block. The last byte is the checksum of bytes 1 through 13 of the packet.

After the 39h or 3Ah command is sent, the I2C master writes four 16-byte packets containing the 64 bytes of data for the block. After the fourth 16-byte write, the data will be written to Flash. Note that it typically requires 64 ms to write the Flash and therefore the I2C master should wait about 100 ms before reading the status byte (see the device specification for details).

16-Byte Data Packet

Table 3 shows the composition of the 16-byte data packet sent by the master.

Table 3. 16-Byte Packet Sent by Master

1 to 16: 16-Byte Flash Data)

I2C Sequence Example

The full I2C sequence of events is shown in Code 1. Note that (w) is "I2C write" and (r) is "I2C read."

```
(w) CMD = 38h, with authentication key
(r) 1 byte status
(w) CMD = 39h, with authentication key,
blocknumber, block checksum and packet checksum
(w) 16 bytes data (bytes 1 through 16 of the
block)
(w) 16 bytes data (bytes 17 through 32 of the
block)
(w) 16 bytes data (bytes 33 through 48 of the
block)
(w) 16 bytes data (bytes 49 through 64 of the
block)
(r) 1 byte status
:
: --- continue for every block ---
:
(w) CMD = 3Bh, with authentication key
(r) 1 byte status
```

Code 1. Full I2C Sequence of Events

Status Byte

The 1-byte status that is read by the I2C master is shown in Table 4. The Read Status command (which is the only read function) must be performed after each I2C write command. The status byte can also be read without any conflicts during normal user code operation.

Allocation of RAM and Location of I2C Bootloader

1. The relocateable code start address in the Linker tab of Project >> Settings menu should be set to 150 in order to accommodate the new ISR table. Since *boot.tpl* can change with new releases of PSoC devices and PSoC Designer, the address of 150 may need to be increased to make room for additional code. If you receive an error during the link operation regarding a memory conflict at address 150, simply increase the setting higher than 150.
2. The I2C bootloader uses SRAM locations F0h through FFh for status, checksums, Flash write control, and a variety of other functions. The only bytes that are maintained during runtime are addresses FEh and FFh (Bank 0 for PSoC devices with more than 256 bytes of SRAM). These are the last locations on the block and should not be used for any other purpose.
3. The I2C bootloader code address is set in the *i2cbootloader.inc* file. This can be changed to accommodate different sized devices.

Table 4. 1-Byte Status

Bit Number	7	6	5	4	3	2	1	0
Bit Name	IVCERR	IVKERR	BM	CCERR	FPERR	FCERR	IVERR	BCOK

- | | | |
|---|--------|---|
| 0 | BCOK | Boot Completed OK. Set when boot operation has successfully completed. |
| 1 | IVERR | Image Verify Error. Set when Flash image verification from block 2 to 223 failed (16K device). |
| 2 | FCERR | Flash Checksum Error. Set when the verification of Flash block indicated by BlockID failed. |
| 3 | FPERR | Flash Protection Error. Set when the Flash block on which write operation was attempted is protected. |
| 4 | CCERR | Communication Checksum Error. Set when the checksum of the received packet does not match with the packet checksum. Also set when the first byte is not FFh or an I2C timeout occurs. |
| 5 | BM | Bootloader Mode. Set when the program is in bootloader mode. |
| 6 | IVKERR | Invalid Key Error. Set when the 5-byte bootloader key received is invalid. |
| 7 | IVCERR | Invalid Command Error. Set when the command byte or block sequence is invalid. |

User Options in *i2cbootload.inc*

There are several options the user must select in the *i2cbootload.inc* file in order for the I2C bootloader to operate correctly. The options are as follows:

- **Boot_Packet_Size16** – ‘1’ indicates the I2C packet size is 16 bytes. (Corresponds to the description in this Application Note.) Set to ‘1’.
- **Boot_Packet_Size78** – ‘1’ indicates the I2C packet size is 78 bytes. (Corresponds to the description in Application Note AN2273a “– Standard – I2C Bootloader for PSoC Device, 78-Byte Transfer.”) Set to ‘0’.
- **Boot_I2C_Addr** – This is the hexadecimal value of the I2C bootloader I2C address. Valid addresses are 00 through 7F.
- **KEYn** – These are the 10 authentication keys used to validate the bootloader command. Valid values for each are 00 through FF.
- **I2CpinSet** – These parameters select which I2C pinset is used. ‘0’ signifies P1[0] and P1[1] are the two I2C pins. ‘1’ signifies P1[5] and P1[7] are the two I2C pins.
- **Flash_32K** – Set this value to ‘1’ if the Flash size is 32K.
- **Flash_16K** – Set this value to ‘1’ if the Flash size is 16K.
- **Flash_8K** – Set this value to ‘1’ if the Flash size is 8K.
- **Flash_4K** – Set this value to ‘1’ if the Flash size is 4K.

Note: Only one Flash size is selected with a ‘1’. All others must be ‘0’.

- **WDT_Present** - Select ‘1’ if the code uses (or will use in the future) the Watchdog Timer. By selecting this option, Watchdog Timer resets will occur in the I2C bootloader code.
- **MAC** – Select ‘1’ if the PSoC device contains a MAC. Select ‘0’ if the device does not have a MAC. PSoC devices in the 22xxx, 24xxxA, 27xxx, 29xxx contain the MAC. The 21xxx does not.
- **I2CHW_1_THROTTLE_CLK_RATE** – Select ‘1’ for early versions of the CY8C27xxx family (for example, silicon revision A is leaded with no “X” in the package, CY7C27443-24PVI). All other devices select ‘0’. A selection of ‘1’ is okay for all devices. (This parameter throttles down the write clock when addressing the I2C control register.)

Code Size Considerations

After selecting the user options in *i2cbootload.inc* and compiling the project, the size of the bootloader code should be checked. With the address defaults, the I2C bootloader should not exceed 1984 bytes (the last 2K of Flash minus the last 64-byte block). For example, in a CY8C27xxx PSoC device with 16K of Flash, the bootloader default starts at address 3800h. The end address should be no greater than 3FBFh. This can be checked by viewing the *.lst* file. If this address is exceeded, you may need to alter the bootloader ROM start address in the *i2cbootloader.inc* file. The I2C Bootloader Calculator will provide a warning should the program overflow into the last, unprotected block.

Temperature Considerations

In the ‘Expert User’ Options of the *I2Cbootloader.inc* file, the temperature for the device during programming is set to 25 degrees Celsius. This guarantees the number of Flash erase/write cycles for a temperature range from 0 to 50 degrees Celsius. From 50 to 70 degrees Celsius, the Flash write endurance may slightly decrease; however, based on margins and testing, the Flash writes should still achieve 50,000 writes per block. If the operating temperature is outside the range of 0 to 70 degrees, the design must include a temperature module and pass the value to the bootloader.

Program Flow for Master Device to Download Data to User Module

1. Send a 10-byte bootloader key to the bootloader.
2. Read the error code from the bootloader. The error code indicates the status of the bootloader.
3. Check if the BM flag is set.
4. Send the 14-byte write command sequence, followed by four 16-byte data packets.
5. Read back the error code to determine the status of the completed operation.
6. Repeat steps 4 and 5 until all the Flash image data and Flash checksum data are sent.
7. Send the 10-byte bootloader complete command. A delay after this command is required for the Flash to be checksum-ed before issuing the read error code in step 8.
8. Read error code and check if BCOK flag is set.
9. If BCOK flag is set, the bootloader has successfully completed, it will reset, and the user code will execute.

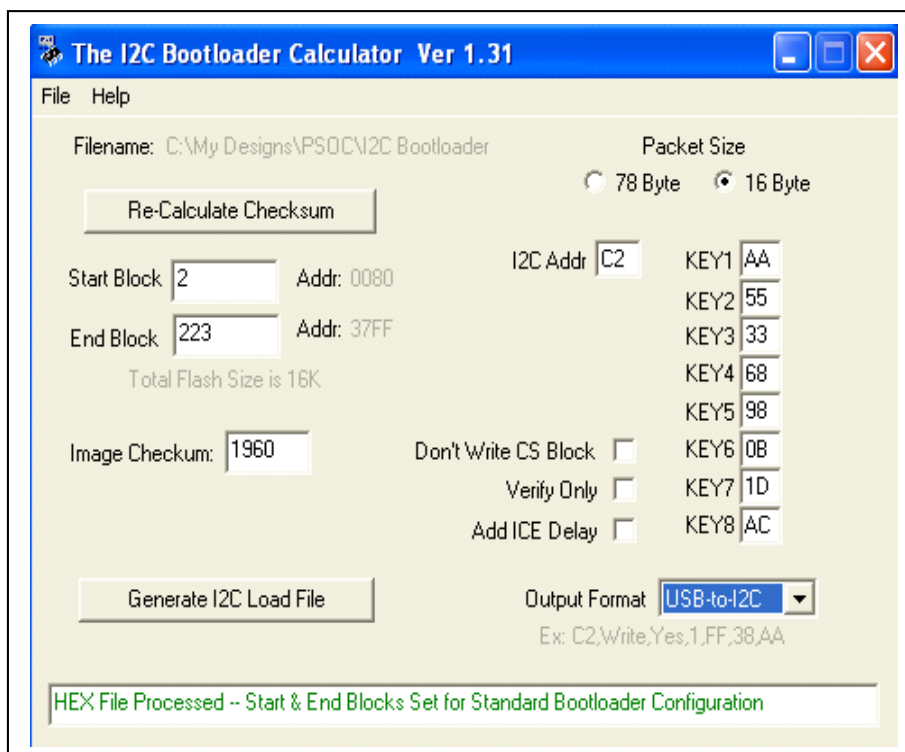


Figure 3. I2C Bootloader Calculator

I2C Bootloader Calculator

In order to calculate the image checksum, all of the bytes are added, starting from the first writable block through the last writable block (blocks 2 through 223 for the 16K device). Because this is a tedious task, it is useful to have an I2C 'Bootloader Calculator' that derives the image checksum from the *.hex* file generated by PSoC Designer. The I2C Bootloader Calculator software that is part of this project includes such a function. This program also creates the I2C file for the I2C master with all of the command sequences, data blocks and necessary checksums.

To use, start the I2C Bootloader Calculator. Open the *.hex* file that was generated by PSoC Designer. The calculator will detect the Flash size, set the block limits and generate the image checksum. You may change the block range to accommodate custom modules. Insert the calculated image checksum into your 'C' or assembly language code and re-compile.

To generate an I2C file, set the authentication key sequence if different than the default. Select the output file format from the Output Format drop-down menu. Then press 'Generate I2C Load File' and the file will be generated. (See Figure 3.)

Conclusion

There are multiple ways to load code into the PSoC device. The standard in-system programming pins are always available and program the PSoC device in any state. The I2C bootloader also allows the user to program the PSoC device within a system but uses a standard I2C system bus. This enables the system to easily upgrade firmware and functions using a system maintenance processor. The I2C bootloader project contains all of the files necessary for incorporating the function. The project also includes the I2C Bootloader Calculator for easy checksum calculation and file generation.

About the Author

Name: Ernie Buterbaugh
Title: Field Application Engineer
Principal

Background: BSEE from Pennsylvania State University. More than 25 years experience with embedded processors, board level design, ASIC, FPGA and CPLD design. Has authored a variety of Application Notes, articles, and the book *Perfect Timing: A Design Guide for Clock Generation and Distribution*.

Contact: ewb@cypress.com

Cypress Semiconductor
2700 162nd Street SW, Building D
Lynnwood, WA 98037
Phone: 800.669.0557
Fax: 425.787.4641

<http://www.cypress.com/>

Copyright © 2005 Cypress Semiconductor Corporation. All rights reserved.
PSoC™, Programmable System-on-Chip™, and PSoC Designer™ are PSoC-related trademarks of Cypress.
All other trademarks or registered trademarks referenced herein are the property of their respective owners.
The information contained herein is subject to change without notice. Made in the U.S.A.