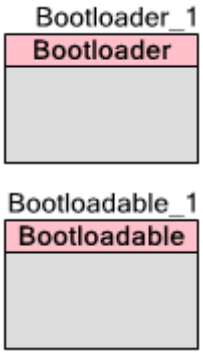
Bootloader and Bootloadable

1.10



Features

- Separate Bootloader and Bootloadable components
- Configurable set of supported commands
- Flexible component configuration

General Description

The bootloader system manages the process of updating the device flash memory with new application code and/or data. To make the process work we use these components:

- Bootloader project - project with Bootloader and Communication components
- Bootloadable project - project with a Bootloadable component, which creates the code

Bootloader Component

The Bootloader component allows you to update the device flash memory with new code. The bootloader accepts and executes commands, then passes responses to those commands back to the communications component. The bootloader collects and arranges the received data and manages the actual writing of flash through a simple command/status register interface.

The project application type needs to match the component placed onto the schematic. For example, for bootloader project set the application type to Bootloader and Bootloader component and placed onto the schematic. For the information about application types see the PSoC Creator Help.

Communications Component

The communications component manages the communications protocol to receive commands from an external system, and passes those commands to the bootloader. It also passes command responses from the bootloader back to the off-chip system.

The USB and I^2C are the only officially supported communication methods for the bootloader. See the USBFS or I^2C component datasheet for more details about the appropriate communication method. There is also a custom interface option to add bootloader support to any existing communications component.

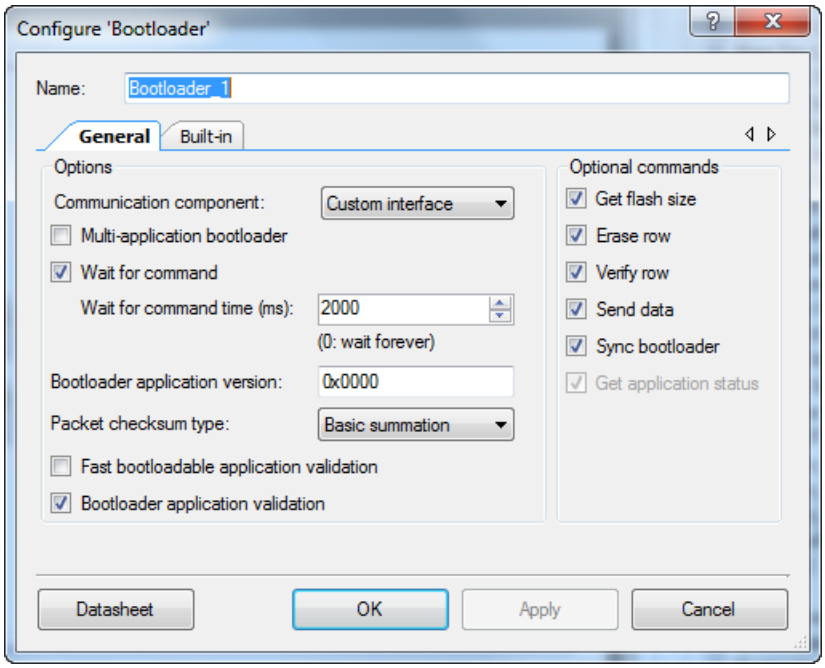
You can also create your own bootloader component for any number of communication methods. For information and instructions on how to do this, see the Component Author Guide.

Bootloadable Component

When you use the Bootloadable component you can specify additional parameters for the bootloadable project.

Bootloader Component Parameters

Drag a Bootloader component onto your design and double-click it to open the **Configure** dialog box.



The Bootloader component has these parameters:

Communication Component

This is the communications component that the bootloader uses to receive commands and send responses. Select one, and only one, communications component. This property is a list of the available communications protocols on the schematic that have bootloader support. In all cases, independent of what is on the schematic, there is also a custom interface option available that allows for implementing the bootloader functions directly.

If there is no communications component on the schematic, then the Custom Interface option is selected. This allows for implementing the communication in any way.

Multi-application bootloader

This option allows two bootloadable applications to reside in flash. It is useful for designs that require a guarantee that there is always a valid application that can be run. This guarantee comes with the limitation that each application has one half of the flash available from what would have been available for a "standard" bootloader project.

Wait for command

On device reset, the bootloader can wait for a command from the bootloader host or jump to the application code immediately. If this option is enabled, the bootloader waits for a command from the host until the timeout period specified by **Wait for command time** parameter occurs. If the bootloader does not receive this command within the specified timeout interval, the active bootloadable project in the flash is executed after the timeout.

Wait for command time

If the bootloader waits for the command to start loading a new bootloadable application after a reset, this is the amount of time it waits before starting the existing bootloadable application. This option is valid only if **Wait for command** is enabled, otherwise it is ignored and grayed out. The zero value is interpreted as wait forever. The default value is a 2 second time out.

Bootloader application version

This parameter provides a 2 byte number to represent the version of the Bootloader application. Default value is 0x0000.

Packet checksum type

This parameter has a couple of options for the type of checksum to use when transferring packets of data between the host and the bootloader. The default value is **Basic summation**.

The basic summation checksum is computed by adding all the bytes (excluding the checksum) and then taking the 2’s complement. The other option is CRC-16CCITT – the 16 bit CRC using the CCITT algorithm.

The checksum is computed for the entire packet with the exception of the Checksum and End of Packet fields.

Fast bootloadable application validation

This option controls how the bootloader verifies the application data. If it is disabled, the bootloader computes the bootloadable application checksum every time before starting it. If enabled, the bootloader only computes the checksum the first time and assumes that it remains valid in each future startup.

Bootloader application validation

If this option is enabled, the bootloader validates itself by calculating the checksum and comparing it with the saved one that resides in metadata. If the validation is not passed, the device is halted. If this option is disabled, the bootloader is executed even if it is corrupted. This could lead to unpredictable results.

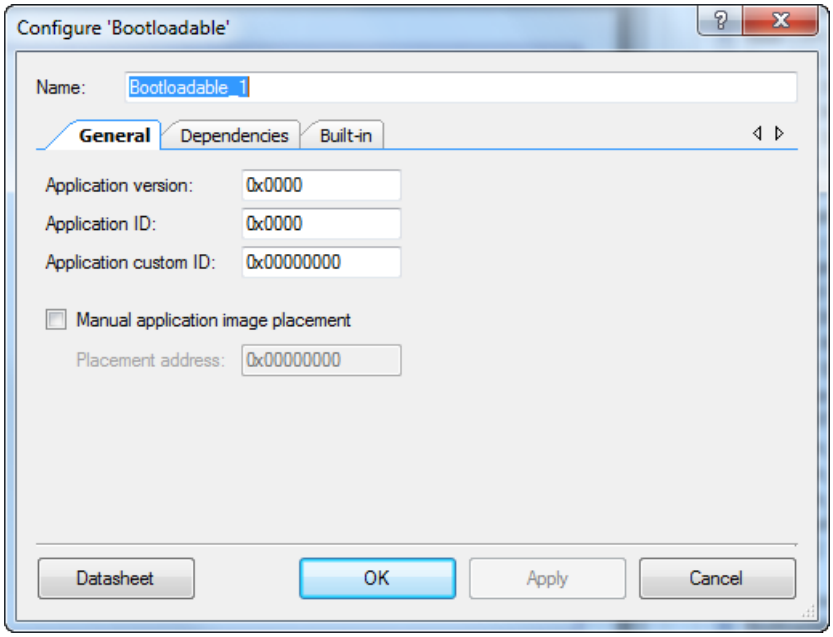
Optional Commands

This group of options determines whether or not a corresponding command is supported by the bootloader. If it is enabled, then the corresponding command is supported. By default all optional commands are supported.

The **Get flash size**, **Send data**, and **Verify row** commands are required by the Cypress Bootloader Host tool. These commands might not be used by custom bootloader host tools.

Bootloadable Component Parameters

Drag a bootloadable component onto your design and double-click it to open the **Configure** dialog.



The General tab of the Bootloadable component contains these parameters:

Application version

This parameter provides a 2 byte number to represent the version of the bootloadable application. Default value is 0x0000.

Application ID

This parameter provides a 2 byte number to represent the ID of the bootloadable application. The default value is 0x0000.

Application custom ID

This parameter provides a 4 byte custom ID number to represent anything in the bootloadable application. The default value is 0x00000000.

Manual application image placement

If this option is enabled, PSoC Creator places the bootloadable application image(s) at the location specified by **Placement address** option. It is also placed according to the rules outlined in section **Bootloadable Project** below.

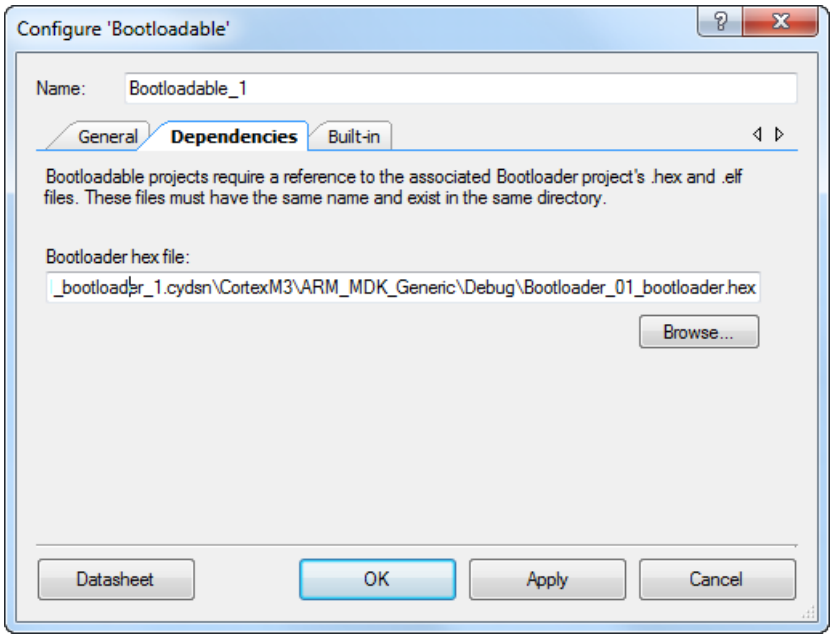
Use this option independently for each of two bootloadable applications, if both of them are referenced to the **Multiapplication bootloader** application.

Placement Address

This option allows you to specify the address where the bootloadable application is placed in memory. This option is only valid if you enable the **Manual application image placement** option; otherwise it is grayed out. You need to specify the address above the bootloader image and below the metadata area.

You calculate the placement address by multiplying the number of the flash row (starting from which the image is placed) by the flash row size and adding result to the flash base address. Align the placement address to the flash row size. See the *Flash and EEPROM* chapter of the *System Reference Guide* for details about flash memory organization.

You get the first available row for the bootloadable application from the associated cyacd file when the **Manual application image placement** option is disabled or can be reported by the Get Flash Size command.



The Dependencies tab of the Bootloadable component contains these parameters:

Bootloader hex file

This option allows you to associate a bootloadable project with bootloader one. This is necessary so that the build of the bootloadable project gets the information about the bootloader project (for example, properly calculate where it belongs in memory).

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. This table lists and describes the interface to each function. The following sections cover each function in more detail.

By default, PSoC Creator assigns the instance name “Bootloader_1” to the first instance of a Bootloader component and “Bootloadable_1” to the first instance of a Bootloadable component in a given design. You can rename the instance to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance names used in the following tables are “Bootloader” and “Bootloadable.”

Bootloader and Bootloadable Functions

Function	Description
Bootloader_Start()	Once called, a software reset is executed, and then the Bootloader application takes over the CPU.

Function	Description
Bootloadable_Load()	Updates the meta data area for Bootloader to be started on device reset and resets device.

void Bootloader_Start(void)

Description: Once called, a software reset is executed, and then the bootloader application takes over the CPU. The associated communication component is started as part of the bootloader application initialization. Bootloadable application code, including interrupt handlers, is not executed.

Depending on the Bootloader component configuration, application waits for command from the bootloader host or jump to the application code.

Parameters: None

Return Value: None. The processor is reset when the transfer is complete.

Side Effects: None

void Bootloadable_Load(void)

Description: Updates the meta data area for Bootloader to be started on device reset and resets device.

Parameters: None

Return Value: None. The processor is reset upon function execution.

Side Effects: None

MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for the component. There are two types of deviations defined:

- project deviations – deviations that are applicable for all PSoC Creator components
- specific deviations – deviations that are applicable only for this component

This section provides information on component-specific deviations. Project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment.

The Bootloader/Bootloadable components have not been verified for MISRA-C:2004 coding guidelines compliance.

Sample Firmware Source Code

PSoC Creator provides many example projects that include schematics and example code in the Find Example Project dialog. For component-specific examples, open the dialog from the Component Catalog or an instance of the component in a schematic. For general examples, open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

See the “Find Example Project” topic in the PSoC Creator Help for more information.

Functional Description

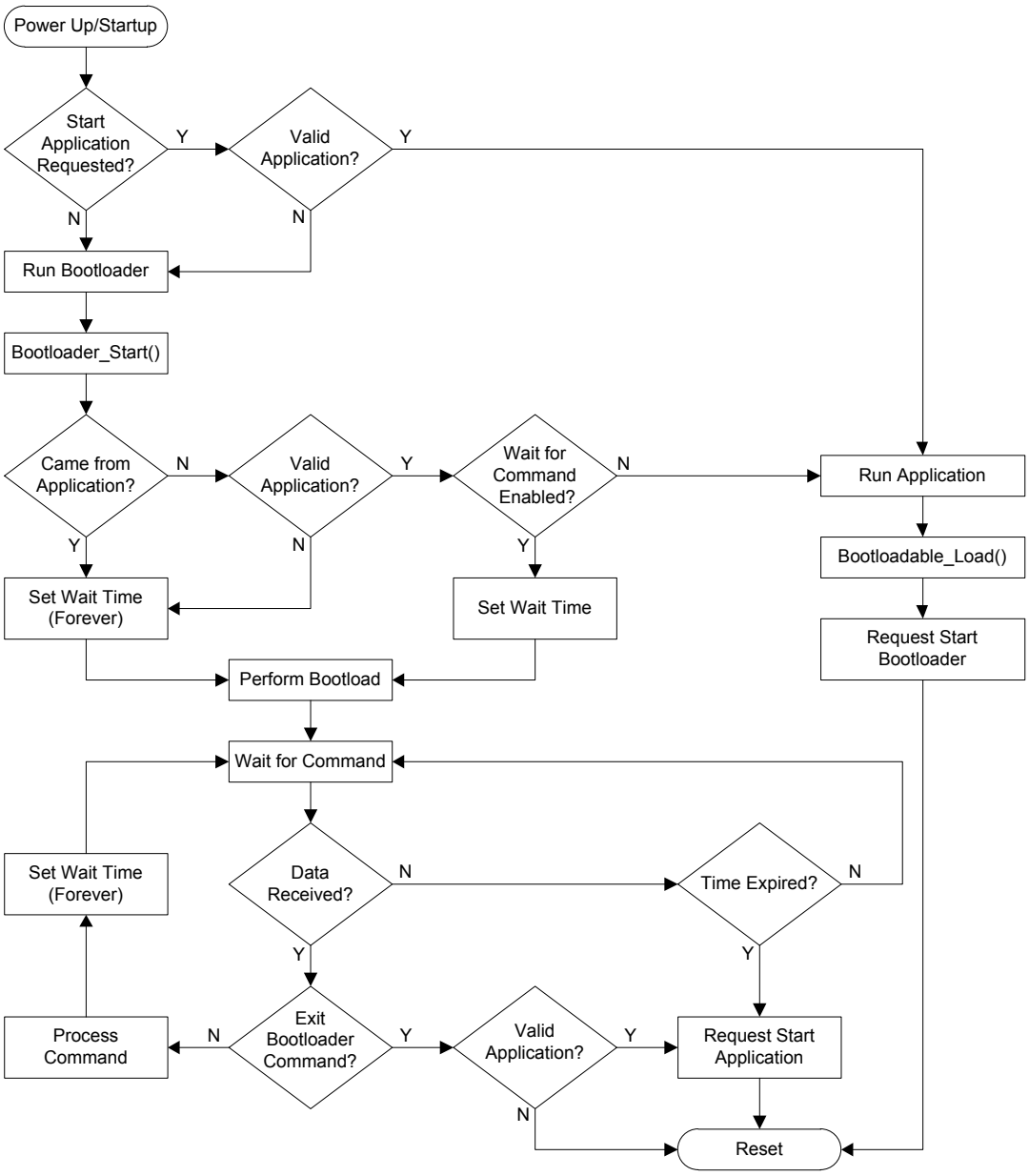
Bootloader and Bootloadable Project Functions

The bootloader project performs overall transfer of a bootloadable project, or new code, to the flash via the bootloader project’s communications component. After the transfer, the processor is always reset. The bootloader project is also responsible at reset time for testing certain conditions and possibly auto-initiating a transfer if the bootloadable project is non-existent or is corrupt.

At startup, the bootloader code loads configuration bytes for its own configuration. It must also initialize the stack and other resources as well as peripherals to do the transfer. When the transfer is complete, control is passed to the bootloadable project with a software reset.

The bootloadable project then loads configuration bytes for its own configuration; and reinitializes the stack and other resources and peripherals for its functions. The bootloadable project may call the Bootloadable_Load() function in the bootloadable project to switch to the bootloader application (this results in another software reset).

This diagram shows how the bootloader works.



Bootloader Application

You typically complete a bootloader design project by dragging a Bootloader component and communication component onto the schematic, routing the I/O to pins, setting up clocks, and so on. A project with Bootloader and communication components implements the basic bootloader application function of receiving new code and writing it to flash. You add custom functions to a basic bootloader project by dragging other components onto the schematic or by adding source code.

Bootloadable Application

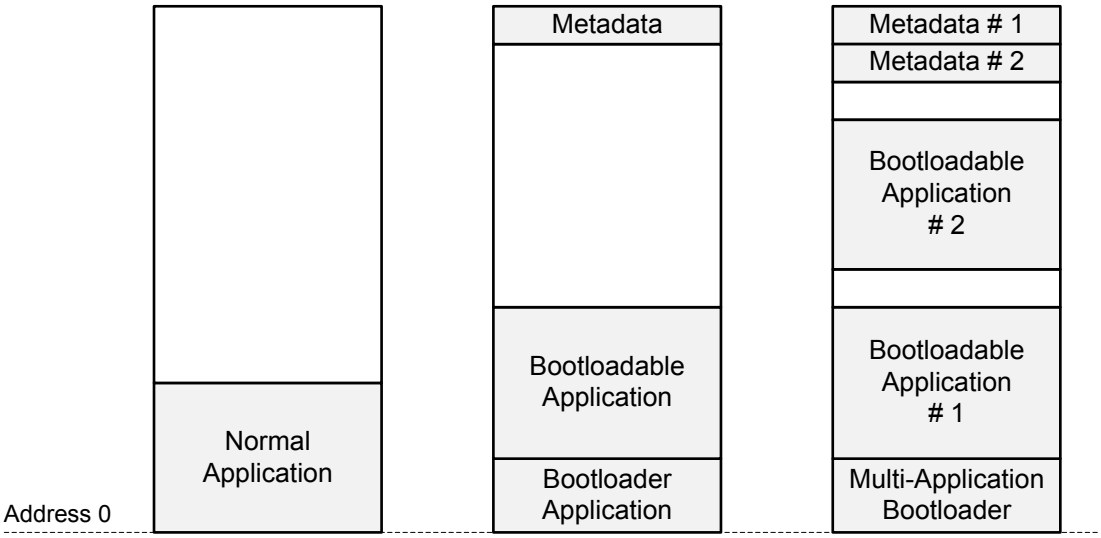
The bootloadable application is actually the code. It is very similar to a normal application type. The main differences are that a bootloadable application is always associated with a bootloader application, while a normal project is never associated with a bootloader application.

Memory Usage

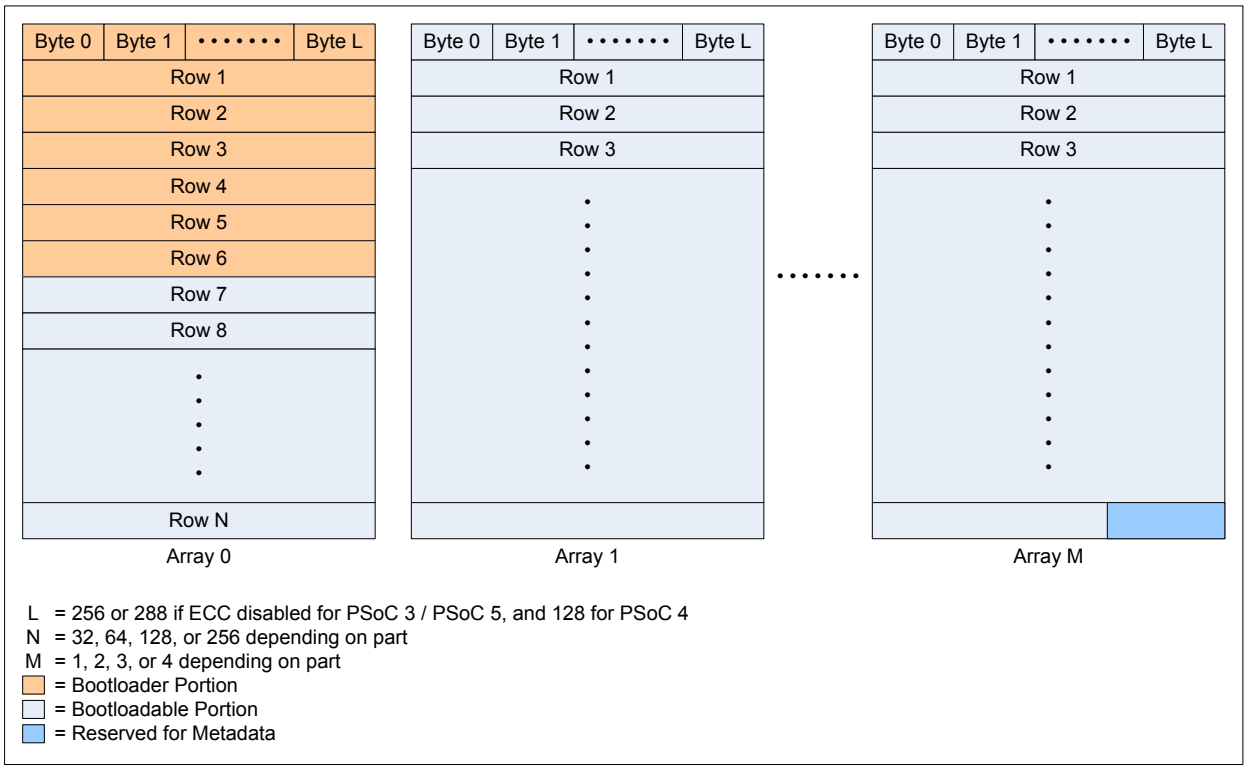
Normal and bootloader applications reside in flash starting at address zero. A bootloadable application occupies flash starting from the next empty flash row to the bootloader application. In case of a multiapplication bootloader, the first bootloadable application resides above the bootloader application. The second bootloadable application occupies flash starting at the row that is halfway between the start of the first bootloadable application and the end of flash.

If the **Manual application image placement** option in the Bootloadable component customizer is enabled, the bootloadable application is placed at an address specified by the **Placement address** option.

This diagram shows (from left to right) the memory usage of normal application, bootloader and bootloadable applications, and the multiapplication bootloader two bootloadable applications:



This diagram shows the device's flash memory layout.



The bootloader project always occupies the bottom X flash rows. X is set so that there is enough flash for:

- The vector table for this project, starting at address 0 (except PSoC 3), and
- The bootloader project configuration bytes, and
- The bootloader project code and data, and
- The checksum for the bootloader portion of flash

The bootloader project configuration bytes are always stored in main flash, never in ECC flash. The relevant option is removed from the bootloader project design-wide resource file.

The bootloader application portion of flash should be protected in the Flash Protection tab of the design-wide resource file to make it only be overwritten by downloading via JTAG / SWD.

The bootloadable project occupies flash starting at the first flash row size boundary after the bootloader, and includes:

- The vector table for the project (except PSoC 3),
- The bootloadable project code and data, and

- 64 bytes of data reserved at the very end of the last flash array to store metadata used by both the bootloader and bootloadable.

The bootloadable project's configuration bytes may be stored in the same manner as in a standard project, i.e. in either main flash or in ECC flash, per settings in the design-wide resource file.

PSoC 3 Details

In the PSoC 3, the only "exception vector" is the 3-byte instruction at address 0, which is executed at processor reset. (The interrupt vectors are not in flash – they are supplied by the Interrupt Controller [IC]). So at reset the PSoC 3 bootloader code simply starts executing from flash address 0.

PSoC 5 and PSoC 4 Details

In the PSoC 5 / PSoC 4 devices, a table of exception vectors must exist at address 0. (The table is pointed to by the Vector Table Offset Register, at address 0xE000ED08, whose value is set to 0 at reset.) The bootloader code starts immediately after this table.

The 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 also has its own vector table, which contains that project's starting SP value and first instruction address. When the transfer is complete, as part of passing control to the bootloadable project the value in the Vector Table Offset Register is changed to the address of the bootloadable project's table.

Metadata

The metadata section is a 64-byte block of flash that is used as a common area for both bootloader and bootloadable applications. In the bootloader application, the metadata is placed at row N-1; in case of multiapplication bootloader, the bootloadable application number 1 uses row N-1, and application number 2 uses row N-2 to store its metadata, where N is the total number of rows for the selected device. Various parameters are saved in this block, which may include:

- The bootloader application version
- The bootloadable application Id
- The bootloadable application version
- The bootloadable custom Id.

PSoC Creator Project Output Files

When either project type – bootloader or bootloadable - is built, an output file is created for that project.

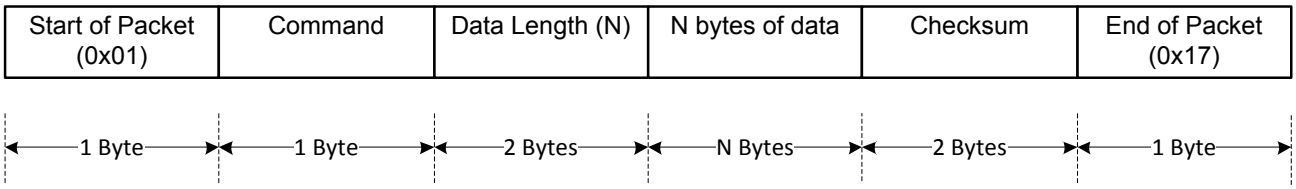
In addition, an output file for both projects – a "combination" file – is created when the bootloadable project is build. This file includes both the bootloader and bootloadable projects. This file is typically used to facilitate downloading both projects (via JTAG / SWD) to device flash in a production environment.

Configuration bytes for bootloadable projects may be stored in either main flash or in ECC flash. The format of the bootloadable project output file is such that when the device has ECC bytes which are disabled, transfer operations are executed in less time. This is done by interleaving records in the bootloadable main flash address space with records in the ECC flash address space. The bootloader takes advantage of this interleaved structure by programming the associated flash row once – the row contains bytes for both main flash and ECC flash.

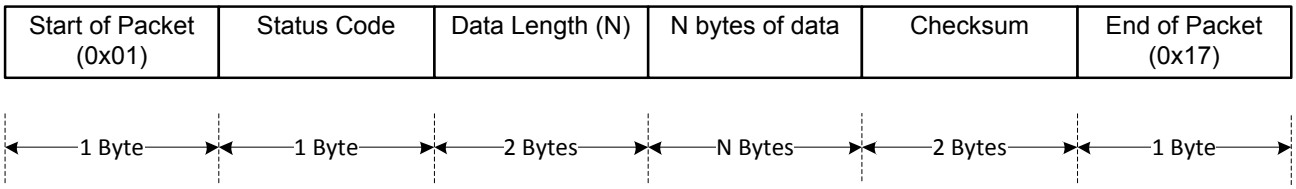
Each project has its own checksum. The checksums is included in the output files at project build time.

Bootloader Packet Structure

Communication packets sent from the Host to the Bootloader have this structure:



Response packets read from the Bootloader have this structure:



Status/Error Codes

The possible status/error codes output from the bootloader are:

Status/Error Code	Value	Description
CYRET_SUCCESS	0x00	The command was successfully received and executed
BOOTLOADER_ERR_VERIFY	0x02	The verification of flash failed
BOOTLOADER_ERR_LENGTH	0x03	The amount of data available is outside the expected range

Status/Error Code	Value	Description
BOOTLOADER_ERR_DATA	0x04	The data is not of the proper form
BOOTLOADER_ERR_CMD	0x05	The command is not recognized
BOOTLOADER_ERR_DEVICE	0x06	The expected device does not match the detected device.
BOOTLOADER_ERR_VERSION	0x07	The bootloader version detected is not supported.
BOOTLOADER_ERR_CHECKSUM	0x08	Packet checksum does not match the expected value
BOOTLOADER_ERR_ARRAY	0x09	Flash array ID is not valid
BOOTLOADER_ERR_ROW	0x0A	The flash row number is not valid
BOOTLOADER_ERR_APP	0x0C	The application is not valid and cannot be set as active
BOOTLOADER_ERR_ACTIVE	0x0D	The application is currently marked as active
BOOTLOADER_ERR_UNK	0x0F	An unknown error occurred

Bootloader Commands

The bootloader supports these commands. All received bytes that do not start with one of the set of command bytes is discarded with no response generated. All multi-byte fields are output LSB first.

Note The time required for the bootloader to execute any command is based on the configuration of the device. Some of the factors that affect the timing include:

- Clock speed at which the part is running
- Toolchain used to build the project
- Optimization settings used during the build
- Number of interrupts running in the background

Bootloader Command Name (Command Code)			
Data Byte (bytes number)	**Response Packet Status Code**	**Response Packet Data (bytes number)**	**Description**
Enter Bootloader (0x38)			
N/A	Success Error Command Error Data Error Length Error Checksum	Silicon ID (4) Silicon Rev (1) Version (3)	The bootloader responds to this command with the device information and version of the Bootloader component. Version means version of the Bootloader component.
Get Flash Size (0x32) (optional)			
Flash Array ID (1)	Success Error Command Error Data Error Length Error Checksum	First available row (2) Last available row (2)	The bootloader responds to this command with the first full row after the bootloader application (first row of the bootloadable application) and last flash row in the selected flash array.
Program Row (0x39)			
Flash Array ID (1) Flash Row Number (2) Data to write (n)	Success Error Command Error Data Error Length Error Checksum Error Flash Row Error Active	N/A	Writes one row of flash data to the device. The data to be written to the flash can be sent in multiple packets using the Send Data command. This command may be sent along with the last block of data, to program the row.
Erase Row (0x34) (optional)			
Flash Array ID (1) Flash Row Number (2)	Success Error Command Error Data Error Length Error Checksum Error Flash Row Error Active	N/A	Erases the contents of the provided flash row.
Verify Row (0x3A) (optional)			
Flash Array ID (1) Flash Row Number (2)	Success Error Command Error Data Error Length Error Checksum	Row checksum (1)	Gets a 1 byte checksum for the contents of the provided row of flash.

Bootloader Command Name (Command Code)			
Data Byte (bytes number)	**Response Packet Status Code**	**Response Packet Data (bytes number)**	**Description**
Verify Checksum (0x31)			
N/A	Success Error Command Error Data Error Length Error Checksum	Checksum valid (1)	A non-zero return value indicates that the application code flash checksum matches the expected value stored in flash and therefore the application is valid. A return value of 0 indicates that the checksums do not match, and therefore the application is not valid.
Send Data (0x37) (optional)			
Data for Device (n)	Success Error Command Error Data Error Length Error Checksum	N/A	Sends a block of data to the device. This data is buffered up in anticipation of another command that will inform the bootloader what to do with the data. If multiple send data commands are issued back-to-back, the data is appended to the previous block. This command is used to breakup large transfers into smaller pieces to prevent bus starvation in some protocols.
Sync bootloader (0x35) (optional)			
N/A	N/A	N/A	Resets the bootloader to a clean state, ready to accept a new command. Any data that was buffered is thrown out. This command is only needed if the host and client get out of sync with each other.

Bootloader Command Name (Command Code)			
Data Byte (bytes number)	**Response Packet Status Code**	**Response Packet Data (bytes number)**	**Description**
Exit Bootloader (0x3B)			
N/A	N/A	N/A	Exits from the bootloader by triggering software reset of the device. Before the software reset is executed, the bootloadable application is verified. If the application passes verification, the application will be executed after the software reset. If the application fails verification, then execution will begin again with the bootloader after the software reset.
Get Application Status (Multi-application bootloader Only) (0x33) (optional)			
Application # (1)	Success Error Length Error Checksum Error Data	App # Valid (1) App # Active (1)	Returns the status of the specified application.
Set Active Application (Multi-application bootloader Only) (0x36)			
Application # (1)	Success Error Application Error Length Error Data Error Checksum	N/A	The specified bootloadable application is set as active. This command is used to switch between two bootloadable applications.

Bootloader Application and Code Data File Format

The bootloader application and code data (.cyacd) file format stores the bootloadable portion of a design. The file is a header followed by lines of flash data. Excluding the header, each line in the .cyacd file represents an entire row of flash data. The data is stored as ASCII data in big endian format.

The header record has this format:

[4-byte SiliconID][1-byte SiliconRev][1-byte Checksum Type]

The data records have this format:

[1-byte ArrayID][2-byte RowNumber][2-byte DataLength][N-byte Data][1-byte Checksum]

The checksum type in the header indicates the type of checksum used for packets sent between the bootloader host and the bootloader itself. The checksum in the data records is a basic summation, computed by summing all bytes (excluding the checksum itself) and then taking the 2's complement.

Bootloader Host Tool

PSoC Creator ships with a bootloader host tool (bootloader_host.exe) that you can use to test the bootloader running on a PSoC chip. The bootloader host tool is the application that communicates directly with the bootloader to send new bootloadable images. The bootloader host tool provided is only a development and testing tool.

Source Code

In addition to the host executable itself, much of the source code used is also provided. Use this source code to create your own bootloader host applications. The source code is located in this directory:

<Install Dir>\cybootloaderutils

By default, this directory is:

C:\Program Files\Cypress\PSoC Creator\<Release Version>\PSoC Creator\cybootloaderutils

This source code is broken up into four different modules. These modules provide implementations for the various pieces of functionality required for a bootloader host. Depending on the desired level of control, some or all of these modules can be used in developing a custom bootloader host application.

cybtldr_command.c/h

This module handles construction of packets to send to the bootloader, and the parsing of packets received from the bootloader. It has a single function for constructing each type of packet that the bootloader understands, and a single function for parsing the results for each packet the bootloader can send back.

cybtldr_parse.c/h

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

cybtldr_api.c/h

This is a row level API that allows for sending a single row of data at a time to the bootloader using a supplied communication mechanism. It has functions for setting up the bootload operation, programming a row, erasing a row, verifying a row, and ending the bootload operation.

cybtldr_api2.c/h

This is a higher level API that handles the entire bootload process. It has functions for programming the device, erasing the device, verifying the device, and aborting the current operation.

Resources

The Bootloader and Bootloadable projects use these device resources:

- The Bootloader component uses both general purpose bits of the reset status (RESET_SR0) register. These bits are necessary to communicate bootloader intents across the software reset boundaries.
- The resources used by communication component are in the corresponding component datasheet.

API Memory Usage

The component memory usage varies significantly, depending on the compiler, device, number of APIs used and component configuration. The following table provides the memory usage for all APIs available in the given component configuration.

The measurements were done with the associated compiler configured in Release mode with optimization set for Size. For a specific design, the map file generated by the compiler can be analyzed to determine the memory usage.

Configuration	PSoC 3 (Keil_PK51)		PSoC 4 (GCC)		PSoC 5LP (GCC)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
Bootloader	1768	294	1090	8	1032	296
Bootloader (full app)[1]	5841	1560	4280	408	4384	580
Bootloadable (full app)[2]	1747	94	960	232	1544	276

Notes:

1. The measurements for this configuration were done for the entire bootloader project, with the fixed-function based I^2C used as communication component and Bootloader component configured for the minimal flash consumption.
2. The measurements for this configuration were done for entire bootloadable project.

Component Changes

This section lists the major changes in the component from the previous version.

Version	Description of Changes	Reason for Changes / Impact
1.10	Added MISRA Compliance section.	
	Added PSoC 4 device support.	New device support
	Minor datasheet edits	
1.0.a	Datasheet corrections	
1.0	Initial component version	

© Cypress Semiconductor Corporation, 2013. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC® is a registered trademark, and PSoC Creator™ and Programmable System-on-Chip™ are trademarks of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.