PSoC® Creator™ Example Project

BLE OTA External Memory Bootloader and Bootloadable Example Projects

Features

- Over The Air (OTA) firmware update
- Hidden BLE service to receive bootloadable images
- Storage of received images in external memory
- Device Information Service (DIS)
- LED status indication

General Description

This example project demonstrates the implementation of an OTA firmware update using the BLE Bootloader Service that is hidden by default and can be activated by pressing button **SW2** while the device is running. By default, this is a regular bootloadable project that contains the BLE component with Device Information Service. Once bootloader mode is enabled, this example project is ready for receiving a new image of the bootloadable project and storing it to the external memory.

Development Kit Configuration

This example project is designed to run on CY8CKIT-042-BLE from Cypress Semiconductor. A full description of the kit, along with more example programs and ordering information, can be found at http://www.cypress.com/go/cy8ckit-042-ble.

Make sure that kit is powered by 3.3 V (J16 is set to 1 and 2). No connection on the kit board is required to use this example project.

Please, refer to section Functional Description for instructions on this example project usage.

Bootloader Project Configuration

The Bootloader project consists of the Bootloader component and SCB (I^2C master).

The purpose of this project is to replace a bootloadable image that is stored in the internal memory with a bootloadable image that is stored in the external memory.

After a bootloadable image in the internal memory is replaced, the bootloader project invalidates the image in the external memory and resets the device with a scheduled bootloadable launch after startup.

SCB

This project uses the SCB component in the I^2C Master mode for communication with the external memory that is located at the CY8CKIT-042-BLE kit board.

Bootloader

This project also uses the Bootloader component configured to work with the custom interface that is based on the SCB component.

Bootloadable Project Configuration

This project consists of the following components:

- Bootloadable
- SCB (I^2C in Master mode)
- BLE (central role, DIS and Bootloader services)
- Pins

BLE

The BLE component used to implement BLE Device Information Service (DIS) and a hidden Bootloader Service. The purpose of the Bootloader Service is to receive other bootloadable images for further storing in the external memory. The Bootloader service is enabled when SW2 is pressed.

Figure 1. BLE GATT Settings

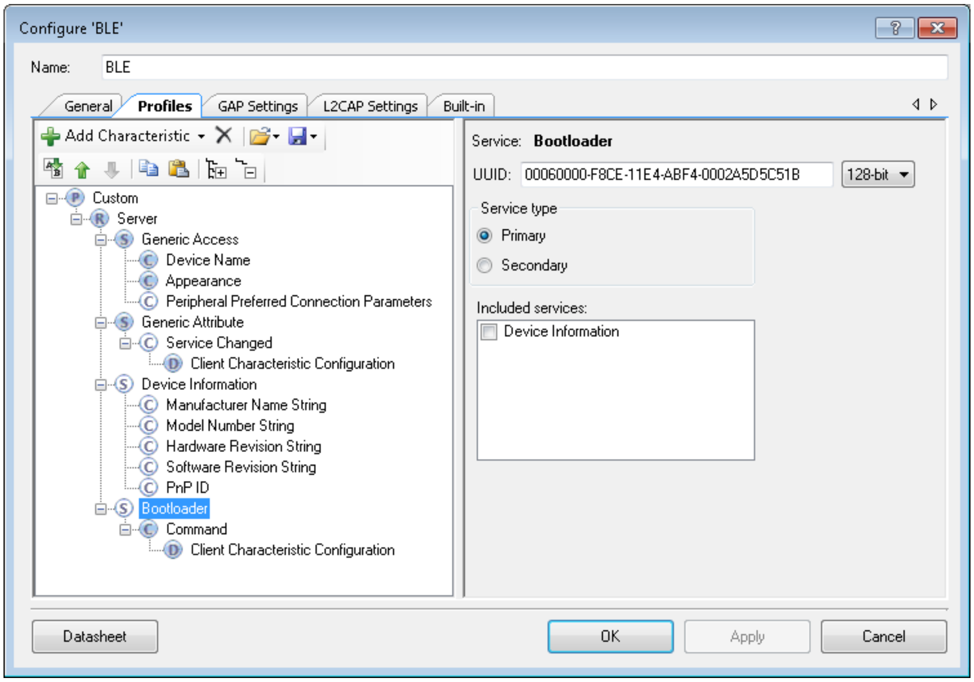


Figure 2. BLE GATT Settings

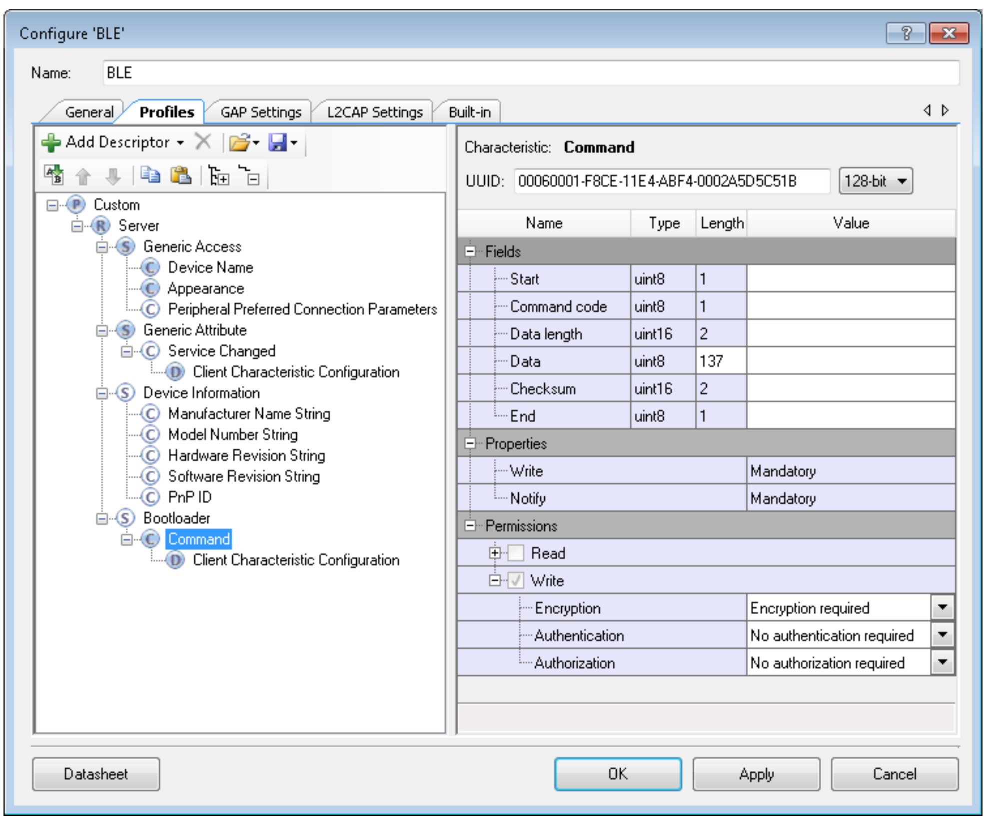
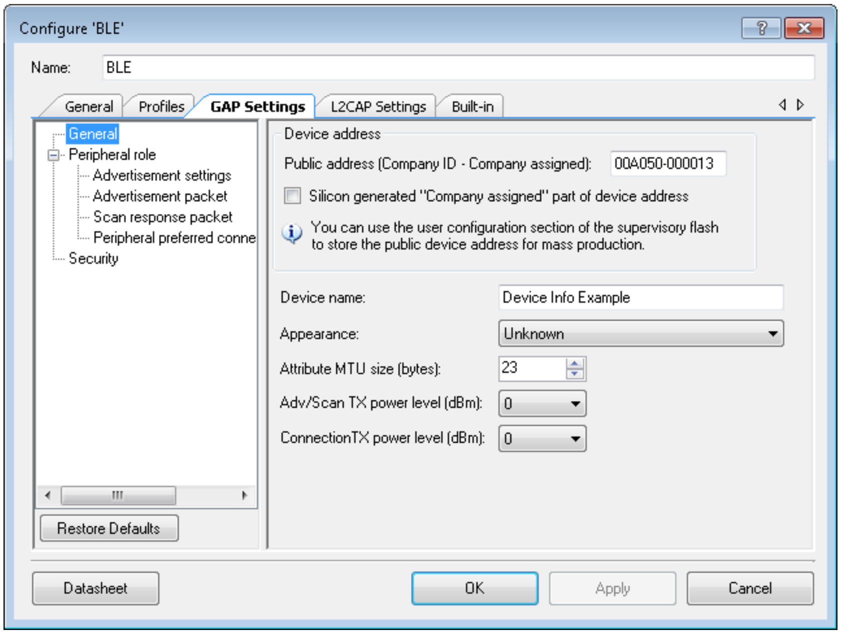


Figure 3. BLE GAP Settings



The BLE component has two services: Device Information Service that is always available, and a Bootloader Service that is hidden by default. Operation of the DIS service is well described in a separate dedicated example project.

The Bootloader Service can be enabled if button SW2 is pressed for longer than 0.1 second. The purpose for this service is to emulate a bootloader component interface during communication with Bootloader Host Tool. To implement this, the following configuration is used.

The Bootloader Service has one characteristic that supports the 'Write' procedure and notifications and it also has a characteristic descriptor – Client Characteristic Configuration.

For communication Bootloader Host Tool writes a command to the Command Characteristic and enables notifications in the characteristic descriptor.

The bootloader emulator reads the command from the characteristic, processes it, and if notifications are enabled in the characteristic descriptor, writes a response to the notification that is sent to Bootloader Host Tool.

Bootloader Host Tool receives the notification and depending on its content (a bootloader emulator response) either sends the next command or reports an error if any.

If commands contain a valid flash row, the bootloader emulator writes it to the external memory. After transfer of a bootloadable image is completed, the bootloader emulator writes metadata to the external flash as well as a flag for the bootloader to replace the bootloadable image.

At the next device reset bootloader is expected to replace the bootloadable image with an image from the external flash if a flag in the external flash is set and if the bootloadable image from the external flash is valid.

The BLE component is configured to have MTU of 23 bytes and the Bootloader Service UUID which is 00060000-F8CE-11E4-ABF4-0002A5D5C51B.

After pressing SW2 onboard the LED changes its indication from green to red, this can be applied to both ‘advertising’ and ‘connected’ modes.

Note The bootloadable project is performing FRAM writes. Due to this maximum connection interval value is set to 1000 ms. Changing this parameter value to smaller values might result in disconnect during bootloading in case when external memory write will take longer.

SCB

This project uses the SCB component in the I^2C Master mode for communication with the external memory that is located on the CY8CKIT-042-BLE kit board. More detail on the external FRAM memory is available in the datasheet – http://www.cypress.com/?mpn=FM24V10-G.

Bootloadable

The Bootloadable component is used to create an image with bootloadable firmware that can be updated without affecting the bootloader.

External Memory

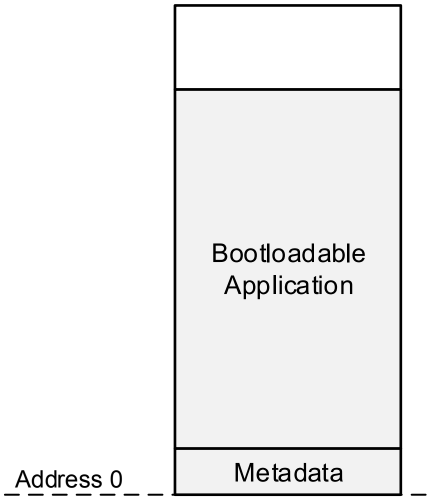
External Memory Interface

The SCB component in the I^2C Master mode is used for communication with the external memory. This project is designed to work with 1-Mbit ferroelectric random access memory (F-RAM) that is present on the CY8CKIT-042-BLE kit board. More details on the used memory can be found in the device datasheet http://www.cypress.com/?mpn=FM24V10-G.

The external memory interface API designed in a generic way, so used F-RAM device can be easily adopted for other external memory devices (I2C EEPROM, SPI EEPROM, etc). The external memory API is available in the OTAMandatory.c file.

Memory Map

The external memory contains metadata and application sections.



Metadata Section

The metadata section is a 128 bytes block of memory that is used as a common area for both bootloader and bootloadable applications. The metadata section is placed at the beginning of the external memory.

Address	Size, Bytes	Description
EMI_MD_APP_STATUS_ADDR	1	Status of the application image located in the external memory. The following values are recognized: • EMI_MD_APP_STATUS_VALID – the application is valid. • EMI_MD_APP_STATUS_LOADED – the application is valid and copied to the internal flash. • EMI_MD_APP_STATUS_INVALID – the application is invalid (it was not copied there or its checksum does not match checksum value stored in the external memory metadata section).
EMI_MD_ENCRYPTION_STATUS_ADDR	1	Stores encryption status.
EMI_MD_APP_EM_CHECKSUM_ADDR	2	Stores the application image checksum.
EMI_MD_APP_SIZE_IN_ROWS_ADDR	2	Stores application image size in flash rows.
EMI_MD_APP_FIRST_ROW_NUM_ADDR	2	Stores the number of the first flash row of the application.
EMI_MD_EXTERNAL_MEMORY_PAGE_SIZE_ADDR	1	External memory page size.

Application Section

The application section starts immediately after the metadata section. The size of the application section is stored in the metadata section (EMI_MD_APP_SIZE_IN_ROWS_ADDR).

Functional Description

Encryption

This example project contains encryption APIs. If the image that is stored on the external FRAM memory has to be encrypted user can enable encryption algorithm. Encryption APIs use same algorithms as BLE stack. In this case all information that is stored on the external FRAM will be encrypted before it is written to it and then decrypted for writing to the internal memory. Metadata part is not encrypted.

APIs are located in the OTAOptional.c files for both bootloader and bootloadable projects.

The encryption algorithm is enabled by setting define value ENCRYPT_ENABLED to YES in files Options.h for bootloadable example projects. ENCRYPT_ENABLED value has to be always set to YES for bootloader project. Enabling of encryption in bootloader project is controlled by EMI_MD_ENCRYPTION_STATUS_ADDR in metadata.

Using UART for debugging

In these example projects UART component is used for printing various debug information (disabled by default).

File Options.h contains define “DEBUG_UART_ENABLED” that is set to “NO”. If extra debugging information have to be provided this define should be set to “YES” in each of bootloader or bootloadable or for both of them. This will decrease the project’s performance, but it will provide extra debugging output to the UART.

A HyperTerminal program is required in the PC to receive debugging information. If you don’t have a HyperTerminal program installed, download and install any serial port communication program. Freeware such as HyperTerminal, Bray’s Terminal, Putty etc. is available on the web.

1. Connect the PC and kit with a USB cable.
2. Open the device manager program in your PC, find the COM port in which the kit is connected, and note the port number.
3. Open the HyperTerminal program and select the COM port in which the kit is connected.
4. Configure Baud rate, Parity, Stop bits and Flow control information in the HyperTerminal configuration window. By default, settings are following: Baud rate – 115200, Parity – None, Stop bits – 1 and Flow control – XON/XOFF. These settings have to match the configuration of the PSoC Creator UART component in the project
5. Start communicating with the device as explained in the project description.

File debug.h contain macros that used for printing various types of data:

- DBG_PRINT_TEXT(a) - prints text string
- DBG_PRINT_DEC(a) – prints decimal number
- DBG_PRINT_HEX(a) – prints hexadecimal number
- DBG_PRINT_ARRAY(a,b) – prints 'b' first elements of array 'a'
- DBG_PRINTF(...) – printf function macro

These macros are used for printing information to UART only if DEBUG_UART_ENABLED define is set to YES.

Checksum type option

The bootloader component allows choosing the type of checksum. The type of checksum in the bootloader project is chosen by the CI_PACKET_CHECKSUM_CRC value. If CI_PACKET_CHECKSUM_CRC is set to YES, the CRC-16 CCITT checksum algorithm will be used for the checksum calculation. If CI_PACKET_CHECKSUM_CRC is set to NO, the basic summation checksum algorithm will be used for the checksum calculation. You have to choose the same checksum type in the bootloader component customizer and in bootloadable project options (Option.h) in order for projects to work.

Project options summary

Option	Value	Explanation
ENCRYPT_ENABLED	YES	Enable encryption of the external memory bootloader image. Always set YES to bootloader project.
	NO	Disable encryption of the external memory bootloader image (Default)
DEBUG_UART_ENABLED	YES	Enable output of debug messages to UART
	NO	Disable output of debug messages to UART (Default)
CI_PACKET_CHECKSUM_CRC	YES	Set checksum type to CRC-16 CCITT checksum algorithm
	NO	Set checksum type to basic summation checksum algorithm (Default)

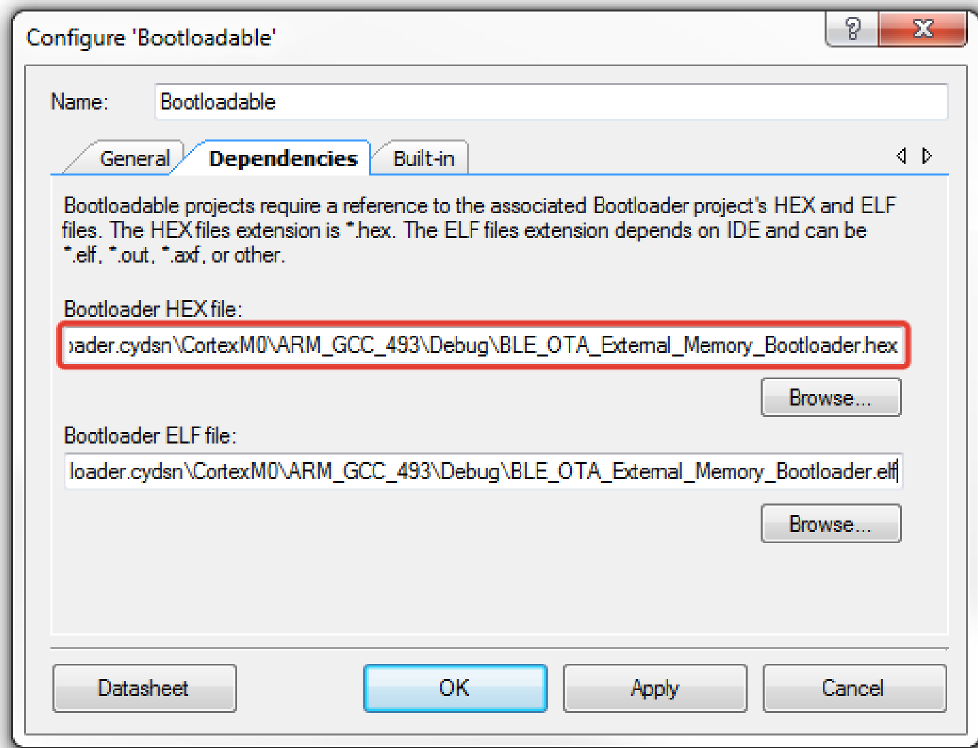
Setup and Run Example Project

Building Example Project

1. Build the *BLE_OTA_External_Memory_Bootloader* example project.
2. Add the *BLE_OTA_External_Memory_Bootloadable* example project to the workspace.

3. Open the top design schematic of the *BLE_OTA_External_Memory_Bootloadable* project. Specify the path to the bootloader project HEX and ELF files by double-clicking on the Bootloadable component and going to the Dependencies tab and link Bootloadable to the BLE_OTA_External_Memory_Bootloader.hex file, as **Figure 4** shows.

Figure 4. Bootloadable Component Configuration



4. Build and Program the *BLE_OTA_External_Memory_Bootloadable* project.

   At this point CYC8CKIT-042-BLE contains firmware that can receive new updates over-the-air. The LED3 flashes with the green color.

   In the *BLE_OTA_External_Memory_Bootloadable* example project's file main.h there is #define LED_ADV_COLOR with value LED_GREEN. Change its value to LED_BLUE and build the project without flashing it.

Upgrading Project images

There are two software tools that can send Example project image updates to device:

- Bootloader Host Tool (BHT), which is provided with PSoC Creator.
- CySmart, external software that needs to be downloaded and installed separately.

There are two hardware Dongles that communicate with BLE devices and can be used to update firmware:

- CY5670, with 128KB Flash chip, supports BLE 4.1. It can be used with both BHT and CySmart.
- CY5677, with 256KB Flash chip, supports BLE 4.2. It can be used only with CySmart. It also supports higher transfer rates, up to 1Mbps.

If BHT is used for the update, ensure that the dongle used is CY5670. You can refer to the documentation that came with the dongle to get this information. You can also get this information from CySmart.

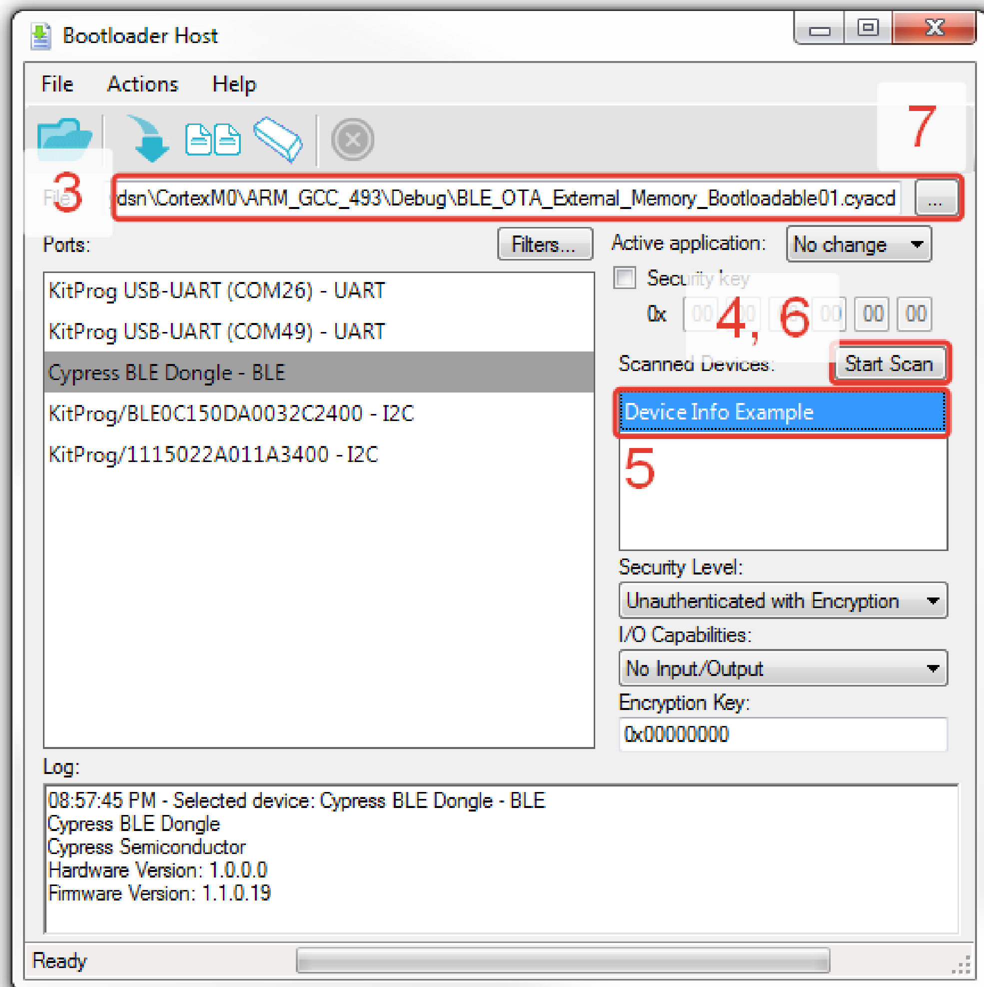
1. Connect your dongle, and install its drivers if you have not done it already.
2. Open CySmart.
3. In the “Select BLE Dongle Target” dialog that appears, look for the dongle name under “Supported devices” or “Unsupported devices”. If the dongle name contains **4.2**, for e.g. "CySmart BLE **4.2** USB Dongle (COM3)", then the dongle supports BLE 4.2 and can be used only with CySmart.
4. If your dongle is not listed by CySmart, click on the **Refresh** button in the “Select BLE Dongle Target” dialog window.

Upgrade Project images with BHT

Follow the steps mentioned below to update the kit firmware Over-The-Air (OTA):

1. Open the Bootloader Host tool (BHT) by navigating to Tools > Bootloader Host in PSoC Creator.
2. Press the **SW2** button on the kit board. The LED indication changes color to red and prepares the device for receiving new application image.
3. In the Bootloader Host Tool select “Cypress BLE Dongle”.
4. Press the **Start** button next to the “Scanned Devices” field.
5. Wait until the list “Device Info Example” in “Scanned Devices” appears.
6. Now press the **Stop** button and select “Device Info Example” from the list.
7. Through File > Open menu point Bootloader Host Tool to the *.cyacd file from bootloadable project folder. It is located in the project folder ([*project folder*]\CortexM0\[*compiler name*])

Figure 5. Bootloader Host Tool



8. Press the **Program** button in Bootloader Host Tool and wait while new application image uploads.
9. As soon as the new application is uploaded, the device will start updating its firmware. Once it is finished the device will reset and newly programmed application will start.

Note If the update fails at authentication, then check that **Security Level** options are equal to the ones selected in Stack project's BLE component.

Upgrade Project images with CySmart

More detailed information about CySmart usage can be found in CySmart User Guide (section 2.7 Updating Peripheral Device Firmware). You can download the CySmart tool and its user guide from here http://cypress.com/cysmart.

To upgrade the device firmware OTA follow the next steps:

1. Make sure OTA device is running Stack project image, and is ready to receive updates.
2. Connect BLE Dongle.
3. Open CySmart software, and select BLE Dongle.
4. Press **Start Scan** button to start scanning for the peripheral device.
5. When the device is listed in the Discovered Device list, stop scanning by clicking the **Stop Scan** button.
6. Select the device from the Discovered Device list.
7. Click **Update Firmware** button.
8. Select the firmware update type on OTA firmware update window.

Firmware Update Type	Description
Application and Stack (Combined) update	Application and Stack firmware co-exist in the same memory location. This option to update complete firmware.
Application only update	Application and stack firmware exist in independent memory locations. This option updates only application firmware.
Application and Stack update	Application and Stack firmware exist in independent memory locations. Select this option to update first the stack firmware followed by application update.

9. Click **Next**. In the next page, browse and select the new firmware image (*.cyacd) file.
10. Click on **Update** button.
11. Wait for the device firmware to be updated.

Note If the update fails at authentication, then check if the security settings in CySmart matches with the settings used in the BLE component of the stack project. To change the security settings in CySmart, click on the **Configure Master Settings** button in the tool and go to the **Security parameters** page.

Expected Results

After the firmware update the device should work as it did before, except the LED indication – now the color used for advertisement indication is blue instead of green.

If there are more changes to the *BLE_OTA_External_Memory_Bootloadable* project, repeat all the steps described for an OTA firmware update in the Project Description section.

© Cypress Semiconductor Corporation, 2016. 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.