



# **MPLAB Harmony Bootloader Help**

MPLAB Harmony Integrated Software Framework

# Bootloader Library Help

This section describes the Bootloader library.

## Introduction

This library provides software Bootloader Library that is available on the Microchip family of Micro-controllers

## Description

The Bootloader Library can be used to upgrade firmware on a target device without the need for an external programmer or debugger.

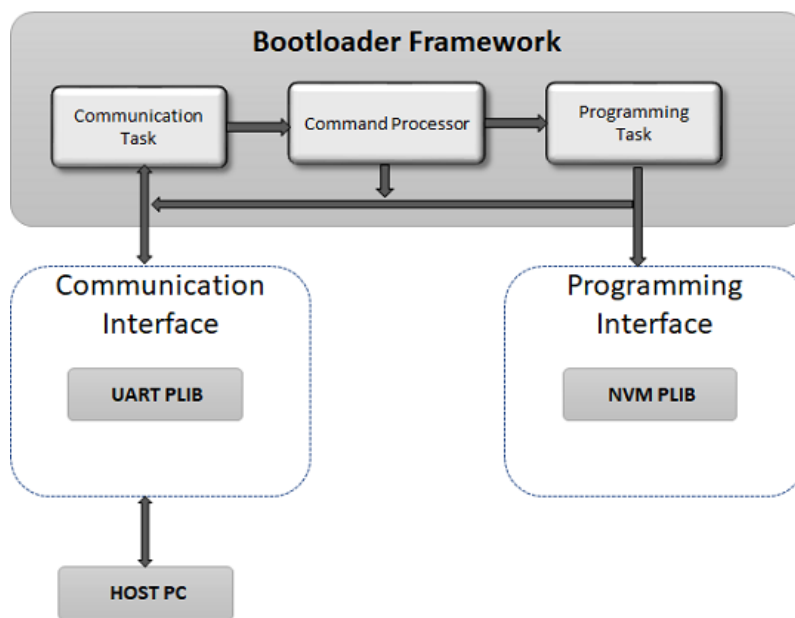
A Bootloader is a small application that starts the operation of the device. A Bootloader does not fully operate the device, but can perform various functions prior to starting the main application. Such functions can include:

- Firmware upgrades
- Application integrity
- Starting the application

## Bootloader Framework

This section describes the basic architecture of the Bootloader library

## Description



**Bootloader Block Diagram**

The Bootloader framework is divided into 3 sub-systems

### 1. Communication Task:

- This task is responsible for receiving data from host PC through the communication interface
- The communication interface here is UART and is accessed using the UART peripheral library
- The task keeps polling for data to be received when bootloader is in idle mode
- The task also validates the incoming packet from host with expected header information
- Once the packet reception is completed communication task gives control to command processor

### 2. Command Processor:

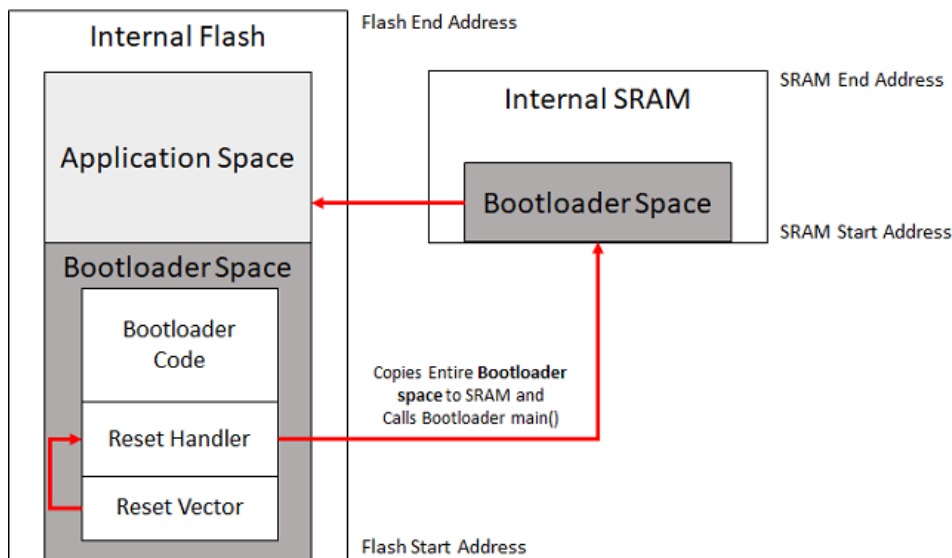
- This task processes the commands received from communication tasks and provides response back to host accordingly
- If the command received is a program command it gives control to the programming task

### 3. Programming Task:

- This task is responsible to program the internal flash memory with data packet received
- The task uses the NVM peripheral library to perform the Unlock/Erase/Write Operations
- The task also invokes communication task in parallel to receive next packet while waiting for the flash operation to complete

### Key Features

- Simple and Optimized Bootloader
- Supports simultaneous Flash memory write and reception of the next block of data, Achieved by loading bootloader into flash and running from SRAM
- Has capability to self update as it is running from SRAM



- At reset the bootloader Reset handler copies the entire bootloader firmware into SRAM from Start location and start executing from SRAM
- Once the application is called from bootloader, applications startup code takes control over SRAM and starts executing

## How The Library Works

This section describes how the Bootloader Library Works.

### Description

The Bootloader Library is implemented using a framework. The Bootloader firmware communicates with the personal computer host application by using a predefined communication protocol. The Bootloader Framework works in two different modes

#### Basic Mode

- This mode is supported for all the devices
- Resides from the starting location of the flash memory region
- The Bootloader performs flash erase/program/verify operations with the binary sent from host while in the firmware upgrade mode
  - The binary sent is only of the application to be programmed
  - Bootloader always performs flash operation from the offset address for application sent from host
  - The application can use the entire flash memory region starting from the end of bootloader space
- Jumps to the application once verification is completed

## Fail Safe Update Mode

- This mode is supported for the devices which have a Dual Bank flash memory
- Resides from the starting location of the flash memory region of both the banks
- The Bootloader performs flash erase/program/verify operations with the binary sent from host while in the firmware upgrade mode
  - The binary sent should contain bootloader and application to be programmed, As both the banks need to have bootloader code at starting location to run the application residing in them at reset
  - Bootloader always performs flash operation from the start of opposite flash memory bank
  - The application can use only the flash memory region of one bank starting from the end of bootloader space
- Performs a bank swap and reset to run the application programmed in opposite bank once verification is completed

## Bootloader Protocol

This section describes the Bootloader protocol used and supported commands

### Description

#### UART Bootloader Protocol

The uart bootloader protocol as shown in below figure is common for all the supported commands.



- **GUARD**
  - The Guard value must be a constant value of **0x5048434D**
  - This value provides protection against spurious commands
  - Bootloader always checks for the Guard value at start of packet reception and proceeds further accordingly
- **Data Size**
  - This field indicates the number of data bytes to be received
  - This value varies for different commands
- **Command**
  - Indicates the command to be processed. Each command is of 1 Byte width
  - Below are the supported commands
    - Unlock (0xA0)
    - Data (0xA1)
    - Verify (0xA2)
    - Reset (A3)
- **Data**
  - Contains the actual Data to be processed based on the command.
  - Length of the data to be received is indicated by **Data Size** field
  - Bootloader receives data in size of words
  - All data words must be sent in a little-endian (LSB first) format

## Response Codes

Bootloader will send a single character response code in response to each command. Sequential commands can only be send after the response code is received for a previous command, or after 100 ms timeout without a response.

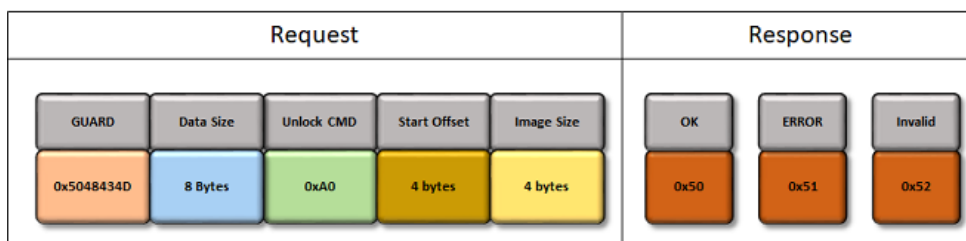
Possible response codes are:

- OK (0x50) – command was received and processed successfully
- Error (0x51) – there were errors during the processing of the command
- Invalid (0x52) – invalid command is received
- CRC OK (0x53) – CRC verification was successful
- CRC Fail (0x54) – CRC verification failed

## Command Description

### Unlock Command

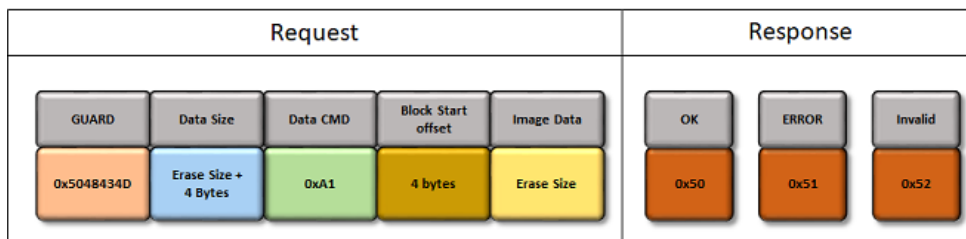
The Unlock Command sequence is as shown in below figure with corresponding responses.



- Unlock command must be issued before the first Data command
  - It is used to calculate application start address and end address
  - This information will be used to validate if the addresses sent are within the range of flash memory
  - It will also be used to validate the address coming with data packet to be programmed are within the region for which the unlock command is invoked
- Number of bytes of data to be received is 8 Bytes (start Offset + Image Size)
- Start Offset
  - Its the application offset from the beginning of the flash memory
  - It is device dependent and should be always greater than or equal to the bootloader end offset
  - It must be aligned at an Erase Unit Size boundary, which is also device dependent
  - To upgrade the bootloader itself this value must be set to 0
- Image size must be in increments of Erase Unit bytes which is also device dependent

### Data Command

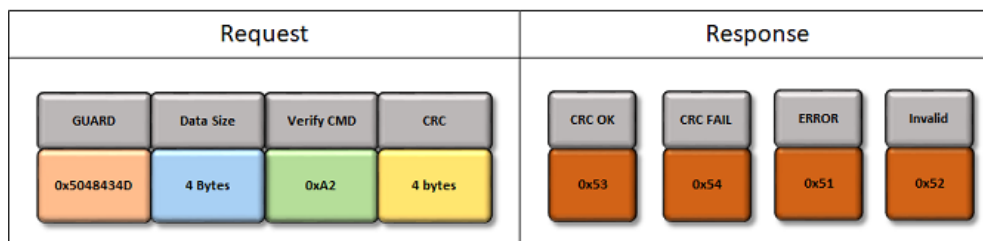
The Data Command sequence is as shown in below figure with corresponding responses.



- Data command is used to send the image data
- Data size is equal to sum of block start offset (4 Bytes) and Erase Unit Size which is device dependent
- Block start offset must be located inside the region previously unlocked via the Unlock command
- Attempts to request the write outside of the unlocked region will result in error and supplied data will be discarded

## Verify Command

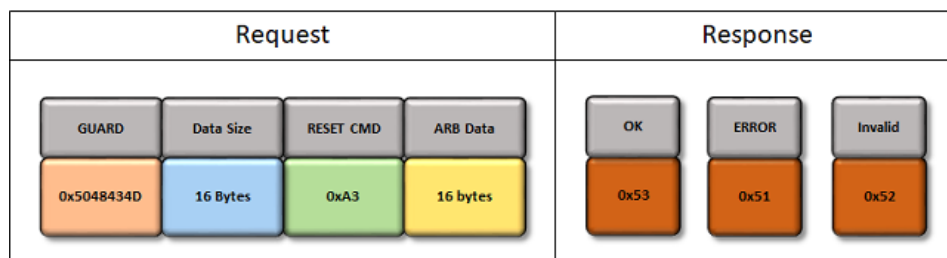
The Verify Command sequence is as shown in below figure with corresponding responses.



- Verify command is used to verify the image data sent and programmed
- Image CRC is a standard IEEE CRC32 with a polynomial of 0xEDB88320
- Internal CRC is calculated based on the values actually read from the Flash memory after programming, so it verifies the whole chain.
- Image CRC is calculated over the previously unlocked region.

## Reset Command

The Reset Command sequence is as shown in below figure with corresponding responses.



- Reset command is used to exit the bootloader and run the application
- It is necessary if the host has no control over the reset pin. It can also be useful even if host has control over the Reset
- This command allows host to communicate up to 4 words of arbitrary information to the application
- Supplied arbitrary values are passed to the application in the first 4 locations in the SRAM
- Supplied values must not be all equal to GUARD Bytes, as this will request the bootloader execution upon reset instead of running the application

## Note

As this bootloader supports simultaneous Flash memory write and reception of the next block of data, The next block of data may be transmitted as soon as the status code is returned for the first one.

Because of this behavior, the status code for the last block will be sent before this block is written into the Flash memory. To ensure that this block is written, host must send another command and wait for the response. So either Verify or Reset command must be sent after the last block of data.

## Bootloader Trigger Methods

This Section describes different methods used to enter bootloader

### Description

Bootloader can be invoked in number of ways:

1. Bootloader will run automatically if there is no valid application firmware.

- Firmware is considered valid if the first word at application start address is not **0xFFFFFFFF**
  - Normally this word contains initial stack pointer value, so it will never be **0xFFFFFFFF** unless device is erased.
2. Bootloader will run on external request if the state of the Bootloader entry pin is low on `bootloader_Trigger()` check.
  3. Bootloader will run on application (internal) request if the first 4 locations in the SRAM are equal to GUARD Bytes (**0x5048434D**).
    - The following code can be used by the application to request the bootloader execution:

```
void invoke_bootloader(void)
{
    uint32_t *sram = (uint32_t *)RAM_START_ADDRESS;

    sram[0] = 0x5048434D;
    sram[1] = 0x5048434D;
    sram[2] = 0x5048434D;
    sram[3] = 0x5048434D;

    NVIC_SystemReset();
}
```

## NOTE

External reset takes priority over any other method of entry.

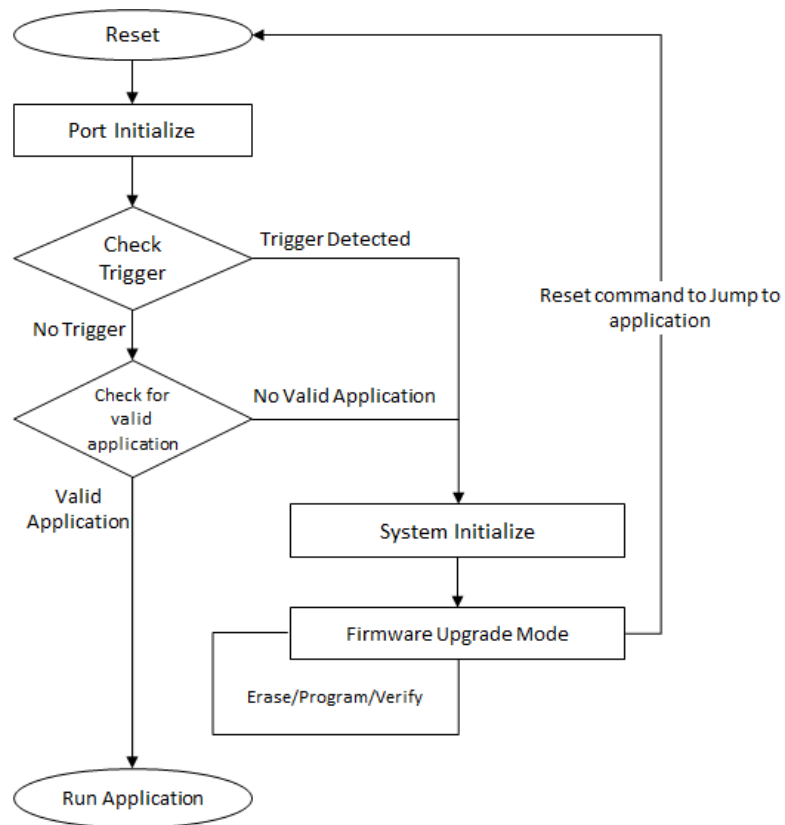
## Bootloader Execution Flow

This section describes the bootloader execution flow.

### Description

#### System Level Execution Flow

- The Bootloader code starts executing on a device Reset
- If there are no conditions to enter the firmware upgrade mode, the Bootloader starts executing the user application
- The Bootloader performs Flash erase/program operations while in the firmware upgrade mode

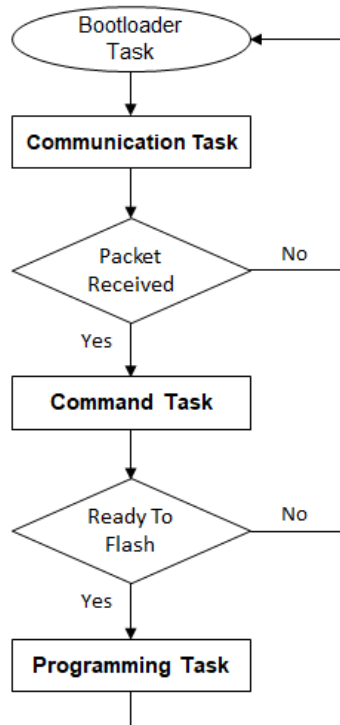


## Firmware Upgrade Mode Execution Flow

### Bootloader Task Flow

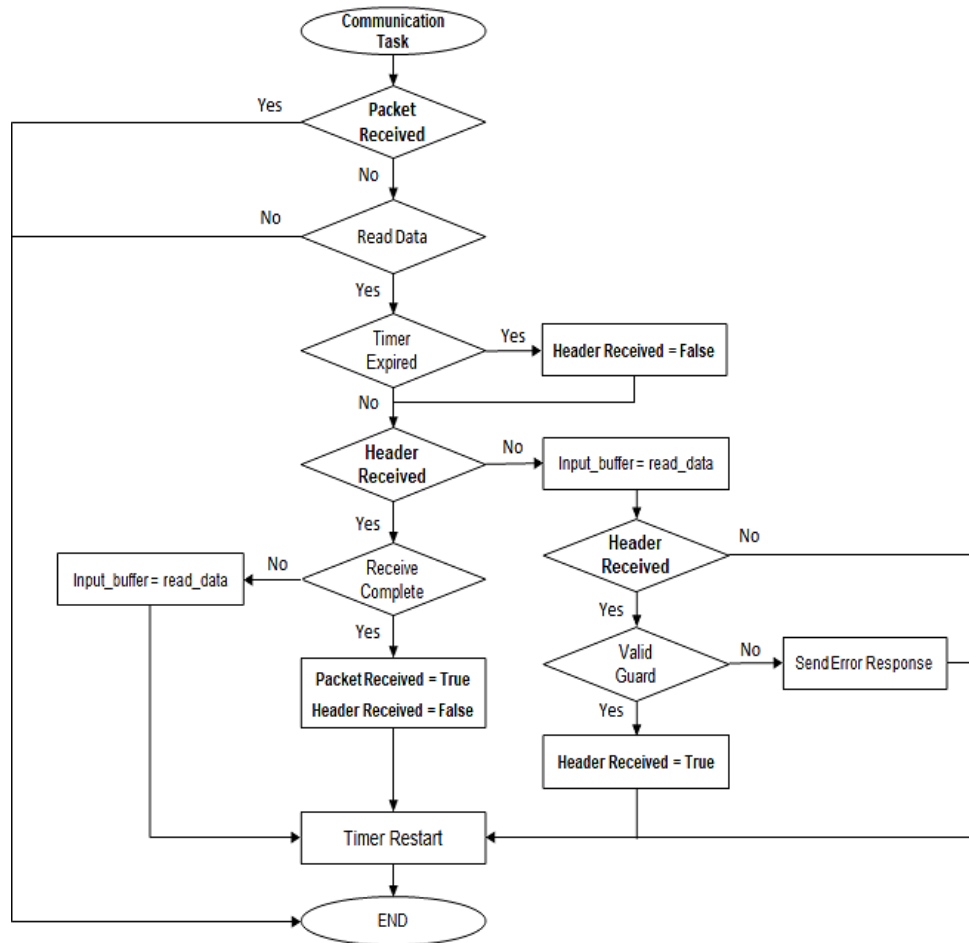
- Bootloader task is the main task which calls the 3 sub-tasks in a forever loop.
- It always calls the communication task to poll for command packets from host
- Once complete packet is received it calls Command task to process the received command
- If the command received was a data command it calls programming task to flash the application





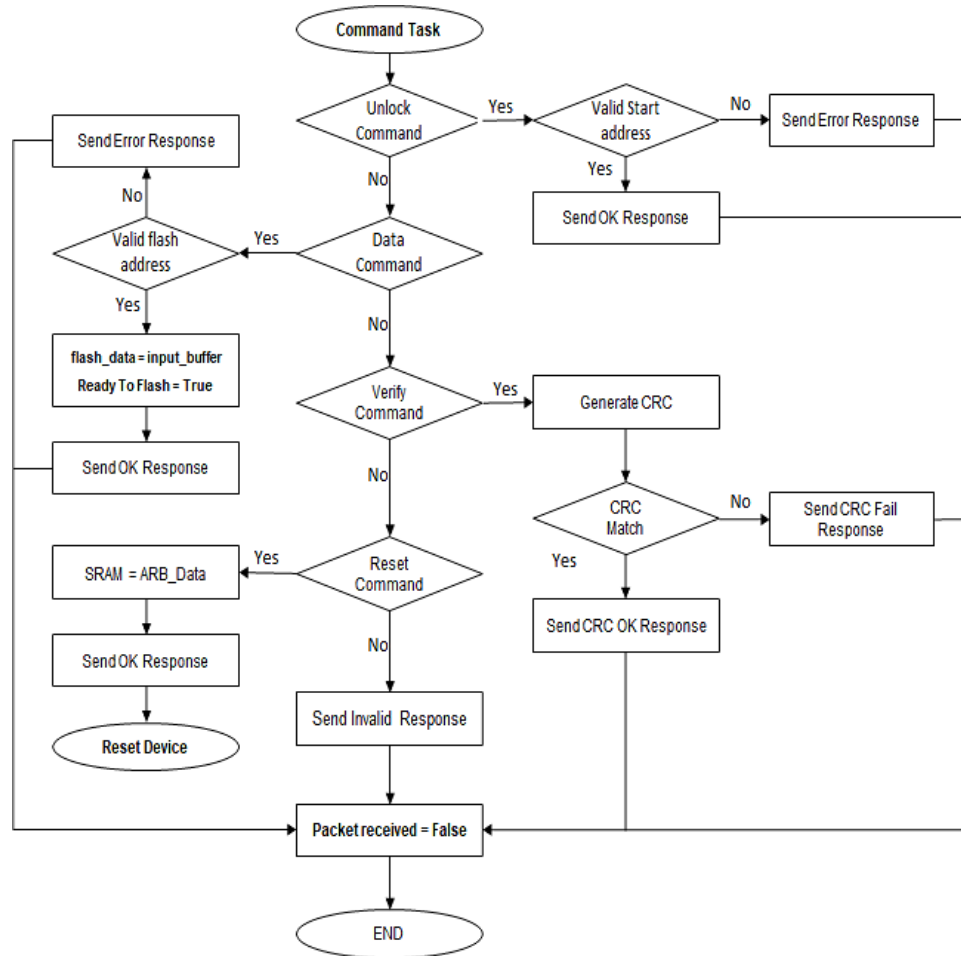
### Communication Task Flow

- This task is used to receive the data bytes from host PC
- If there are valid GUARD bytes received at start of packet it proceeds further to receive the whole packet or else reports error and waits for next command
- All bytes of the command frame must be sent within 100 ms of each other. After 100 ms of idle time, incomplete command is discarded and bootloader goes back to waiting for a new Command.
- This behavior allows host to re-synchronize in the case of synchronization loss.



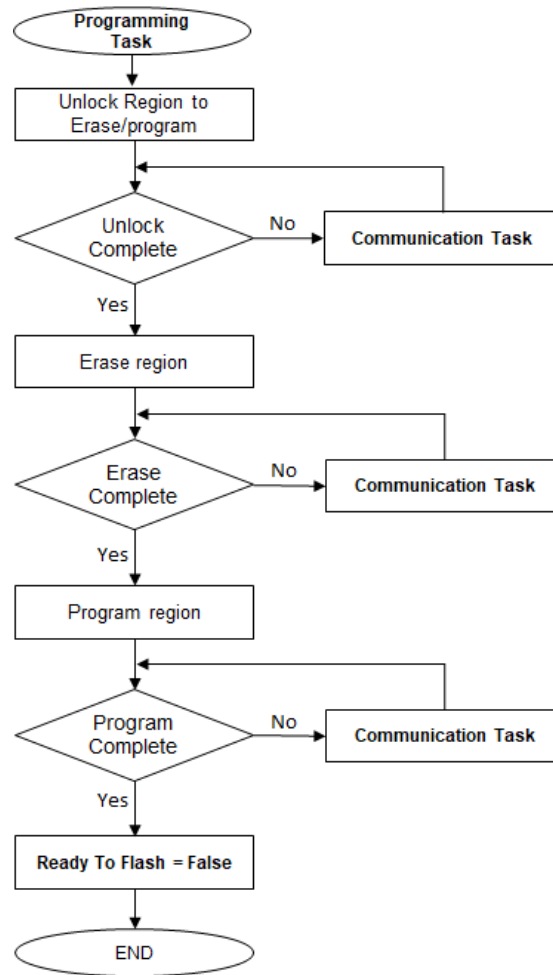
### Command Task Flow

- This task processes the packet received for supported commands
- If the received command is a DATA command, it sets **ready\_to\_flash** flag so that the bootloader task can call programming task



### Programming Task Flow

- This task performs flash operations on the received data
- As the bootloader is running from RAM, While waiting for flash operations to complete it calls **communication task** to receive the next command in parallel



## Bootloader Memory Layout

This Section describes the memory layout for bootloader

### Description

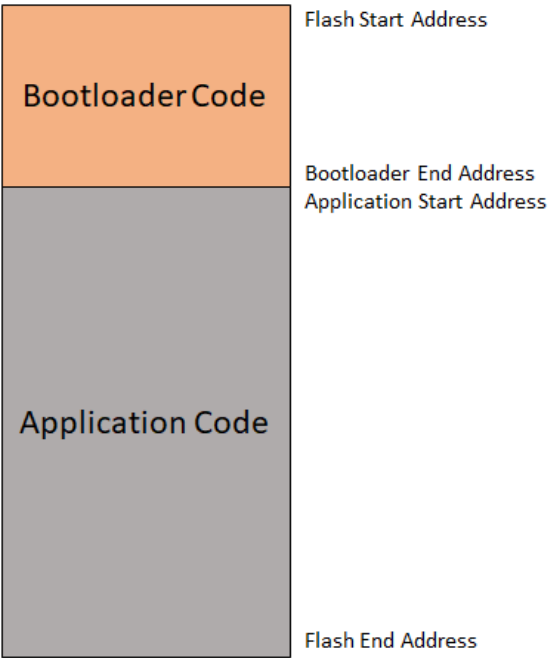
The placement of the Bootloader and the application in flash memory should be such that the application will not overwrite the Bootloader, and the Bootloader can properly program the application when it is downloaded.

- Bootloader code is always placed at start of the flash address
- The application code can be placed anywhere after the bootloader end address
- As the Bootloader code requires the application start address to be mentioned at compile time, it should match in both the application and bootloader

### Note

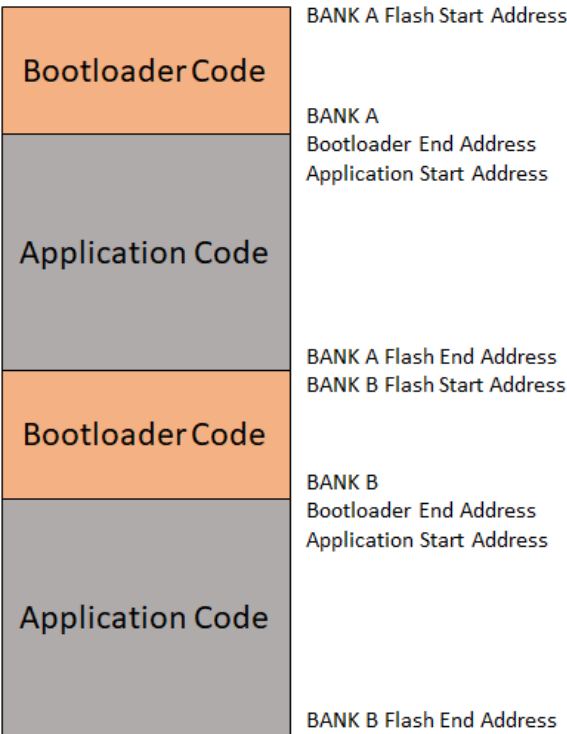
- The start address and the end address of the Bootloader and the application will vary for different devices. Refer to respective Data sheets for details of Flash memory layout.

### Basic Memory Layout



### Fail Safe Update Memory layout

- Fail Safe update mode is only supported for devices which have dual bank support in flash.
- Bootloader code is always placed at start of both the flash banks as upon reset the device runs from either of bank as per configuration. Refer to device data sheet for more details on flash banks
- Application start address should always be the offset from actual physical address being mapped after reset. Bootloader takes care of placing the application in opposite bank.



## Configuring the Library

This section provides information on how to configure Bootloader library and application.

### Description

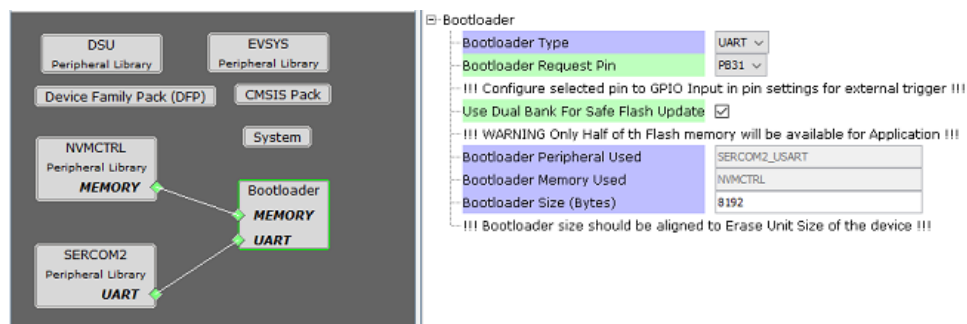
The Bootloader library and application to be Boot-Loaded should be configured through the MHC. The following sections show the MHC configuration window for both Bootloader library and Application.

## Bootloader Configurations

This section provides information on how to configure Bootloader library.

### Description

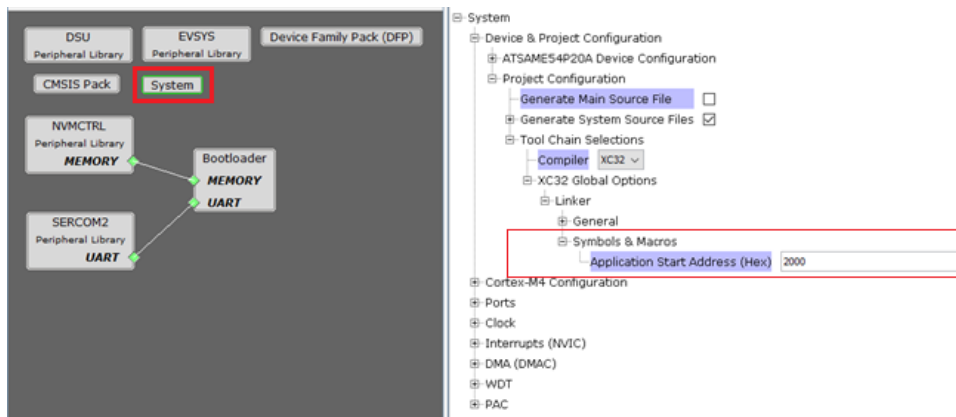
#### Bootloader Specific User Configurations



Bootloader Component Configuration

- **Bootloader Type:**
  - Specifies the type of bootloader to be used
- **Bootloader Request Pin:**
  - Specifies the gpio pin to be used to check for external trigger to enter into bootloader upon reset
- **Use Dual Bank For Safe Flash Update:**
  - Used to configure bootloader to use Dual banks of device to upload the application
  - This option is visible only for devices supporting Dual flash banks
- **Bootloader Peripheral Used:**
  - Specifies the communication peripheral used by bootloader to receive the application
  - The name of the peripheral will vary from device to device
- **Bootloader Memory Used:**
  - Specifies the memory peripheral used by bootloader to perform flash operations
  - The name of the peripheral will vary from device to device
- **Bootloader Size (Bytes):**
  - Specifies the maximum size of flash required by the bootloader
  - This size is calculated based on Bootloader type and Memory used
  - This size will vary from device to device and will always be aligned to device erase unit size

#### Bootloader System Configuration



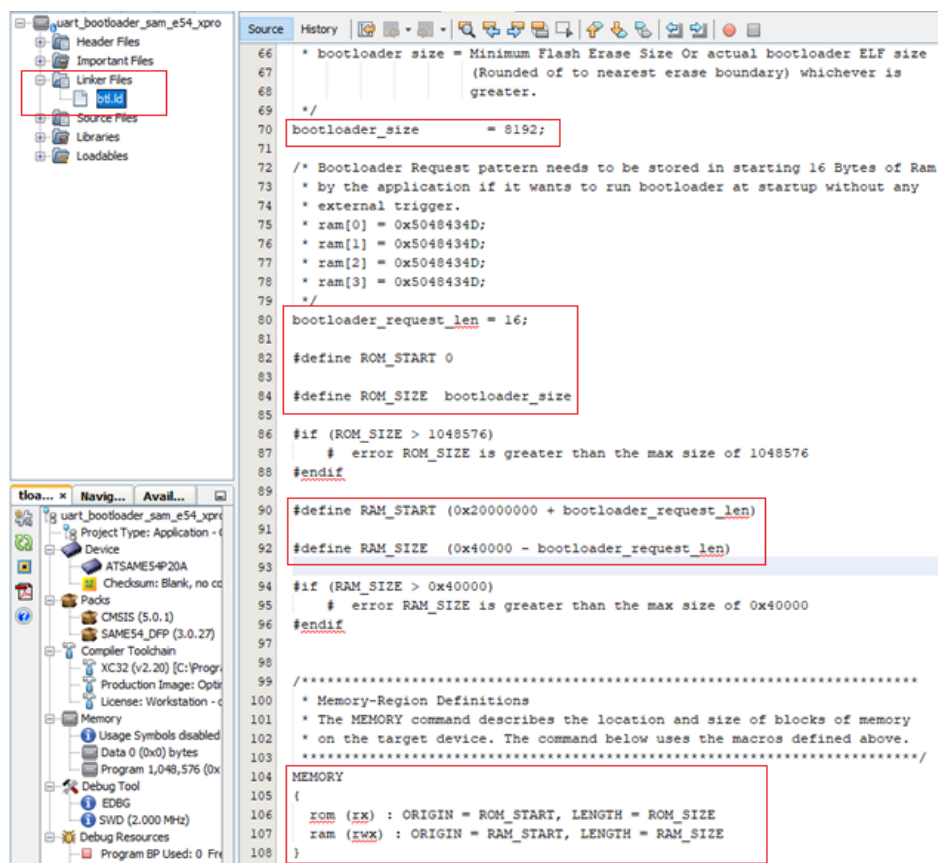
System Configuration for Bootloader

### • Application Start Address (Hex):

- Start address of the application which will be programmed by the bootloader
- This value is filled by the bootloader when it is loaded, which is equal to the bootloader size. It can be modified as per user need
- This value will be used by the bootloader to jump to the application at device reset

## Bootloader Linker Script

- Bootloader library uses a custom linker script generated through MHC
- The values populated in the linker script are based on the above MHC configurations
- Bootloader is configured to run from RAM in this linker script to achieve simultaneous Flash memory write and reception of the next block of data.



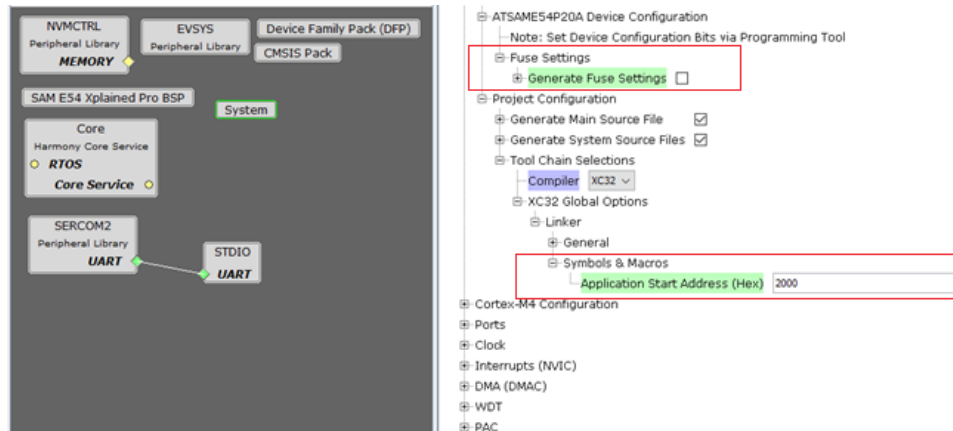
Bootloader Linker Script

## Application Configurations

This section provides information on how to configure the application to be Boot-loaded

### Description

#### Application System Configuration

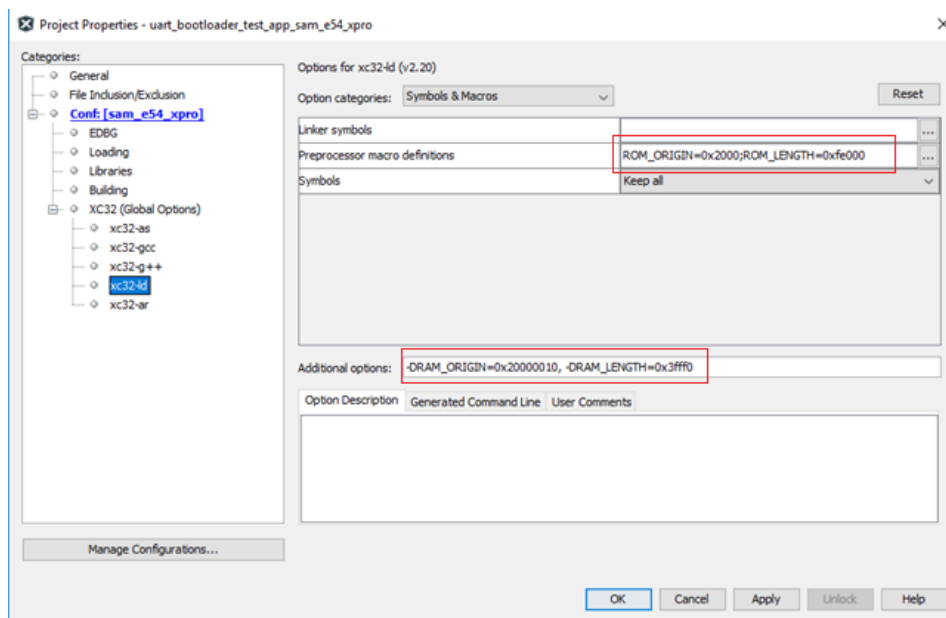


**System Configuration for Application**

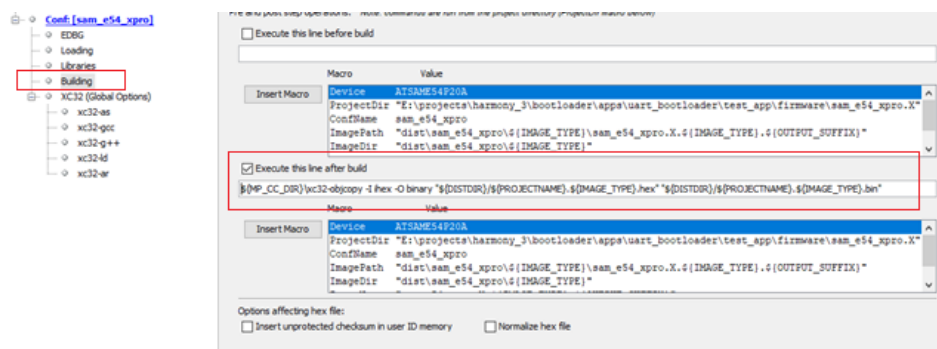
- **Generate Fuse Settings:**
  - Fuse settings needs to be disabled for the application which will be boot-loaded as the fuse settings are supposed to be programmed through programming tool from bootloader code.
  - Also the fuse settings are not programmable through firmware
  - Enabling the fuse settings also increases the size of the binary when generated through the hex file
- **Application Start Address (Hex):**
  - Start address of the application from where the it will be run
  - This value should be equal to or greater than the bootloader size
  - As this value will be used by bootloader to Jump to application at device reset it should match the value provided to bootloader code during generation
  - This value will also be used to generate XC32 compiler settings to place the code at intended address as shown in below snapshots

#### Application Settings in MPLAB-X IDE





Linker Settings in MPLAB-X



Build Settings for generating Binary

- **Preprocessor macro definitions:**
    - ROM-ORIGIN and ROM\_LENGTH are the XC32 linker variables which will be overridden with values provided here
    - These values are auto populated with value of Application start address provided in MHC after regeneration
  - **Additional Options:**
    - RAM\_ORIGIN and RAM\_LENGTH values should be provided for reserving 16 bytes of start of RAM to trigger bootloader from firmware
    - This is optional and can be ignored if not required
  - **Execute the line after Build:**
    - This option can be used to automatically generate the binary file from hex file once the build is complete
- ```

${MP_CC_DIR}\xc32-objcopy -I ihex -O binary "${DISTDIR}/${PROJECTNAME}.${IMAGE_TYPE}.hex"
"${DISTDIR}/${PROJECTNAME}.${IMAGE_TYPE}.bin"

```

## Bootloader Sizing and Considerations

This Section provides information on Bootloader sizes and considerations while changing the size

### Description

#### Bootloader Sizes

The example Bootloaders provided in the **apps/uart\_bootloader/bootloader/** folder have optimization settings for use by most MPLAB XC32 C/C++ Compiler users, which is `-O1`.

However, in terms of size, this option does not produce the most optimal code. Turning on the `-Os` will reduce the size of the Bootloader.

Following table provides the maximum Bootloader size based on Optimization in Bytes. The actual size of the bootloader will be less than the values mentioned as it is rounded off to nearest Erase Unit Size.

| Device                                | -O1  | -Os  |
|---------------------------------------|------|------|
| SAM C21N Xplained Pro Evaluation Kit  | 1792 | 1536 |
| SAM D20 Xplained Pro Evaluation Kit   | 1792 | 1536 |
| SAM D21 Xplained Pro Evaluation Kit   | 1792 | 1536 |
| SAM E54 Xplained Pro Evaluation Kit   | 8192 | 8192 |
| SAM E70 Xplained Ultra Evaluation Kit | 8192 | 8192 |
| SAM L22 Xplained Pro Evaluation Kit   | 1792 | 1536 |

## Size change considerations

It must be ensured that the user application's memory region does not overlap with the memory region reserved for the Bootloader.

The Bootloader generated by MHC should be considered a starting point for your product's Bootloader. As such, adding new features may cause the Bootloader to exceed the default size calculated.

If the size of the Bootloader changes, the following steps should be performed to adjust both the Bootloader and the application in order to make sure that both fit and make best use of the device memory:




- Increase the **Bootloader Size** in Bootloader MHC config menu to some approximate value and regenerate the code
- Determine the new ending address of the Bootloader. This can be done by using either the `.map` file generated by MPLAB X IDE with the respective Compiler, or by using the ELFViewer plug-in for MPLAB X IDE
- Round the size from the map file to the nearest ERASE unit size
- Enter the new value again in **Bootloader Size** in Bootloader MHC config menu
- Change the Application start Address from system settings in both Bootloader and application projects accordingly if it is falling inside bootloader region
- Recompile both the Bootloader and the application, as well as test operations.

If only the Application start address needs to be modified for application then perform following steps:

- Change the Application start Address from system settings in both Bootloader and application projects accordingly
- Recompile both the Bootloader and the application, as well as test operations

## Library Interface

### a) System Functions

|                                                                                     | Name                               | Description                                         |
|-------------------------------------------------------------------------------------|------------------------------------|-----------------------------------------------------|
|  | <a href="#">bootloader_Trigger</a> | Checks if Bootloader has to be executed at startup. |
|  | <a href="#">run_Application</a>    | Runs the programmed application at startup.         |
|  | <a href="#">bootloader_Start</a>   | Starts bootloader execution.                        |

### Description

This section describes the Application Programming Interface (API) functions of the Bootloader library.

Refer to each section for a detailed description.

### a) System Functions

## ***bootloader\_Trigger Function***

### **C**

```
bool bootloader_Trigger();
```

### **Description**

This function can be used to check for a External HW trigger or Internal firmware trigger to execute bootloader at startup.

This check should happen before any system resources are initialized apart for PORT as the same system resource can be Re-initialized by the application if bootloader jumps to it and may cause issues.

- **External Trigger:** Is achieved by triggering the selected GPIO\_PIN in bootloader configuration in MHC.
- **Firmware Trigger:** Application firmware which wants to execute bootloader at startup needs to fill first 16 bytes of ram location with bootloader request pattern.

```
uint32_t *sram = (uint32_t *)RAM_START_ADDRESS;  
  
sram[0] = 0x5048434D;  
sram[1] = 0x5048434D;  
sram[2] = 0x5048434D;  
sram[3] = 0x5048434D;
```

### **Preconditions**

PORT/PIO Initialize must have been called.

### **Returns**

- True : If any of trigger is detected.
- False : If no trigger is detected..

### **Example**

```
NVMCTRL_Initialize();  
  
PORT_Initialize();  
  
if (bootloader_Trigger() == false)  
{  
    run_Application();  
}  
  
CLOCK_Initialize();
```

## ***run\_Application Function***

### **C**

```
void run_Application();
```

### **Description**

This function can be used to run programmed application through bootloader at startup.

If the first 4Bytes of Application Memory is not 0xFFFFFFFF then it jumps to the application start address to run the application programmed through bootloader and never returns.

If the first 4Bytes of Application Memory is 0xFFFFFFFF then it returns from function and executes bootloader for accepting a new application firmware.

### **Preconditions**

`bootloader_Trigger()` must be called to check for bootloader triggers at startup.

## Returns

None

## Example

```
NVMCTRL_Initialize();

PORT_Initialize();

if (bootloader_Trigger() == false)
{
    run_Application();
}

CLOCK_Initialize();
```

## **bootloader\_Start Function**

### C

```
void bootloader_Start();
```

## Description

This function can be used to start bootloader execution.

The function continuously waits for application firmware from the HOST-PC via selected communication protocol to program into internal flash memory.

Once the complete application is received, programmed and verified successfully, It resets the device to jump into programmed application.

Before Jumping into application it clears the initial 16 bytes of ram memory so that the bootloader is not triggered at reset in case application has previously filled it to have a internal firmware trigger.

## Preconditions

`bootloader_Trigger()` must be called to check for bootloader triggers at startup.

## Returns

None

## Remarks

This function never returns.

## Example

```
SYS_Initialize( NULL );

bootloader_Start();
```

## Bootloader Tools Help

This section describes the usage of bootloader host tools

## Description

| Host Script             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Preconditions                                                                                                                                                                                                                            |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>btl_host.py</b>      | <ul style="list-style-type: none"> <li>This host script should be used to communicate with the Bootloader running on the device via UART interface</li> <li>It is a command line interface and implements the bootloader protocol required to communicate from host PC</li> <li>If size of the input binary file is not aligned to device erase boundary it appends 0xFF to the binary to make it aligned and then sends the binary to device</li> <li>It should be used with -s (swap banks) option when using the merged binary generated by <b>btl_app_merge.py</b> for fail safe update and the offset going to device will be 0x0 as the binary has both bootloader and application</li> </ul> | <ul style="list-style-type: none"> <li>Compatible with Python 2.7.x</li> <li>Requires pyserial package to communicate with device over UART <ul style="list-style-type: none"> <li>python -m pip install pyserial</li> </ul> </li> </ul> |
| <b>btl_app_merge.py</b> | <ul style="list-style-type: none"> <li>This script should be used to merge the bootloader binary and application binary</li> <li>It creates a merged binary output where bootloader is placed from start and the application will be placed at the offset passed as parameter</li> <li>If the application offset is not equal to end of bootloader offset it fills the gap with 0xFF until the application offset</li> <li>The merged binary will be used by <b>btl_host.py</b> as input for fail safe update</li> </ul>                                                                                                                                                                            | <ul style="list-style-type: none"> <li>Compatible with Python 2.7.x</li> </ul>                                                                                                                                                           |

## Basic Mode Usage Example

```
python <harmony3_path>\bootloader\tools\btl_host.py --help
```

```
Usage: btl_host.py [options]

Options:
  -h, --help            show this help message and exit
  -v, --verbose          enable verbose output
  -t, --tune             auto-tune UART baudrate
  -i PATH, --interface=PATH
                        communication interface
  -f FILE, --file=FILE  binary file to program
  -o OFFS, --offset=OFFS
                        destination offset (default 0x600)
  -b, --boot            enable write to the bootloader area
  -s, --swap            swap banks after programming
  -d DEV, --device=DEV  target device
                        (same7x/same5x/samd5x/samc2x/samd2x/saml2x)
```

```
python <harmony3_path>\bootloader\tools\btl_host.py -v -i COM18 -d same5x -o 0x2000 -f
<harmony3_path>\bootloader\apps\uart_bootloader\test_app\firmware\sam_e54_xpro.X\dist\sam_e54_xp
ro\production\sam_e54_xpro.X.production.bin
```

```
Unlocking
Uploading 3 blocks at offset 8192 (0x2000)
... block 1 of 3
... block 2 of 3
... block 3 of 3
Verification
... success
Rebooting
Reboot Done|
```

## Fail Safe Update Mode Usage Example

```
python <harmony3_path>\bootloader\tools\btl_app_merge.py --help
```

```
Usage: btl_app_merge_bin.py [options]

Options:
  -h, --help            show this help message and exit
  -v, --verbose          enable verbose output
  -b BTL_FILE, --btl_file=BTL_FILE
                        bootloader binary file to program
  -a APP_FILE, --app_file=APP_FILE
                        application binary file to program
  -o OFFS, --offset=OFFS
                        application start offset (default 0x2000)
  -d DEV, --device=DEV  target device (same5x/samd5x)
```

```
python <harmony3_path>\bootloader\tools\btl_app_merge.py -o 0x2000 -b
<harmony3_path>\bootloader\apps\uart_fail_safe_bootloader\bootloader\firmware\sam_e54_xpro.X\dis
t\sam_e54_xpro\production\sam_e54_xpro.X.production.bin
-a
<harmony3_path>\bootloader\apps\uart_fail_safe_bootloader\test_app\firmware\sam_e54_xpro.X\dist\
sam_e54_xpro\production\sam_e54_xpro.X.production.bin

python <harmony3_path>\bootloader\tools\btl_host.py -v -s -i COM18 -d same5x -f
<harmony3_path>\bootloader\tools\btl_app_merged.bin
```

```
Unlocking
Uploading 4 blocks at offset 0 (0x0)
... block 1 of 4
... block 2 of 4
... block 3 of 4
... block 4 of 4
Verification
... success
Swapping Bank And Rebooting
Reboot Done
```

## Bootloader Applications Help

### uart\_bootloader

| Name                       | Description                                                                                                               |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------|
| <a href="#">bootloader</a> | This example application shows how to use the Bootloader Library to bootload an application using UART protocol           |
| <a href="#">test_app</a>   | This example application is used to demonstrate how it can be loaded from bootloader and trigger bootloader from firmware |

### uart\_fail\_safe\_bootloader

| Name                       | Description                                                                                                                                              |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">bootloader</a> | This example application shows how to use the Bootloader Library to bootload an application on device having dual flash bank feature using UART protocol |
| <a href="#">test_app</a>   | This example application is used to demonstrate how it can be loaded from bootloader and trigger bootloader from firmware                                |

### Description

This section provides help for the Bootloader applications.

### uart\_bootloader

#### **bootloader**

This example application shows how to use the Bootloader Library to bootload an application using UART protocol

## Description

- This application is a bootloader application which resides in the starting location of the device flash memory
- The application calls the `bootloader_Task()` which receives application to be programmed into flash memory over UART channel
- The application uses the Virtual Com port of the device (EDBG port) to receive application binary from host PC
- This application uses the On board Switch as bootloader trigger pin to force enter the bootloader at reset of device.

## Building The Application

This section provides information on how to build an application using the MPLAB X IDE.

## Description

The parent folder for all the MPLAB X projects for this application is given below:

| Application Path | bootloader/apps/uart_bootloader/bootloader/firmware |
|------------------|-----------------------------------------------------|
|------------------|-----------------------------------------------------|

To build the application, refer the following table and open the appropriate project file in the MPLAB X IDE.

| Project Name    | Description                           |
|-----------------|---------------------------------------|
| sam_c21n_xpro.X | SAM C21N Xplained Pro Evaluation Kit  |
| sam_d20_xpro.X  | SAM D20 Xplained Pro Evaluation Kit   |
| sam_d21_xpro.X  | SAM D21 Xplained Pro Evaluation Kit   |
| sam_e54_xpro.X  | SAM E54 Xplained Pro Evaluation Kit   |
| sam_e70_xult.X  | SAM E70 Xplained Ultra Evaluation Kit |
| sam_l21_xpro.X  | SAM L21 Xplained Pro Evaluation Kit   |
| sam_l22_xpro.X  | SAM L22 Xplained Pro Evaluation Kit   |

## MPLAB Harmony Configurations

This section provides information on the MHC configurations.

## Description

### Components Used

- Bootloader
- SysTick peripheral library

The following components used may vary based on the project configuration selected:

- EFC or NVMCTRL peripheral library
- USART or SERCOM peripheral library
- DSU

### Other MHC Settings:

- Provide Application Start Address in XC32-LD settings

## Hardware Setup

This section describes how to configure the supported hardware.

## Description

1. **Project sam\_c21n\_xpro.X.**
  - **Hardware Used**
    - [SAM C21N Xplained Pro Evaluation Kit](#)
  - **Hardware Setup**
    - No Special hardware Setup Required
2. **Project sam\_d20\_xpro.X.**
  - **Hardware Used**
    - [SAM D20 Xplained Pro Evaluation Kit](#)
  - **Hardware Setup**
    - No Special hardware Setup Required
3. **Project sam\_d21\_xpro.X.**
  - **Hardware Used**
    - [SAM D21 Xplained Pro Evaluation Kit](#)
  - **Hardware Setup**
    - No Special hardware Setup Required
4. **Project sam\_e54\_xpro.X.**
  - **Hardware Used:**
    - [SAM E54 Xplained Pro Evaluation Kit](#)
  - **Hardware Setup**
    - No Special hardware Setup Required
5. **Project sam\_e70\_xult.X.**
  - **Hardware Used**
    - [SAM E70 Xplained Ultra Evaluation Kit](#)
  - **Hardware Setup**
    - No Special hardware Setup Required
6. **Project sam\_l21\_xpro.X**
  - **Hardware Used**
    - [SAM L21 Xplained Pro Evaluation Kit](#)
  - **Hardware Setup**
    - No Special hardware Setup Required
7. **Project sam\_l22\_xpro.X**
  - **Hardware Used**
    - [SAM L22 Xplained Pro Evaluation Kit](#)
  - **Hardware Setup**
    - No Special hardware Setup Required

## Running The Application

This section provides information on how to run an application using the MPLAB X IDE.

## Description

1. Connect a micro USB cable to the DEBUG port.
2. Build and program the application using the MPLAB X IDE.



3. Build the Application [test\\_app](#) using MPLAB X IDE
4. Run the **btl\_host.py** from command prompt to program the application binary. Refer to [Bootloader Tools Help](#) for setting up the host script

```
python <harmony3_path>\bootloader\tools\btl_host.py -v -i <COM PORT> -d <Device Name> -o
<offset> -f <harmony3_path>\bootloader\apps\uart_bootloader\test_app\firmware\<Project
Name>\dist\<Config Name>\production\<Project Name>.production.bin
```

The following table provides the Details required to run the above command:

| Kit Name                              | Offset | Device Name | Project Name    | Config Name   |
|---------------------------------------|--------|-------------|-----------------|---------------|
| SAM C21N Xplained Pro Evaluation Kit  | 0x700  | samc2x      | sam_c21n_xpro.X | sam_c21n_xpro |
| SAM D20 Xplained Pro Evaluation Kit   | 0x700  | samd2x      | sam_d20_xpro.X  | sam_d20_xpro  |
| SAM D21 Xplained Pro Evaluation Kit   | 0x700  | samd2x      | sam_d21_xpro.X  | sam_d21_xpro  |
| SAM E54 Xplained Pro Evaluation Kit   | 0x2000 | same5x      | sam_e54_xpro.X  | sam_e54_xpro  |
| SAM E70 Xplained Ultra Evaluation Kit | 0x2000 | same7x      | sam_e70_xult.X  | sam_e70_xult  |
| SAM L21 Xplained Pro Evaluation Kit   | 0x700  | saml2x      | sam_l21_xpro.X  | sam_l21_xpro  |
| SAM L22 Xplained Pro Evaluation Kit   | 0x700  | saml2x      | sam_l22_xpro.X  | sam_l22_xpro  |

5. Following snapshot shows example output of successfully programming the test application. It may vary from device to device.

- **Rebooting** and **Reboot Done** messages in below output signifies that bootloading is successful

```
Unlocking
Uploading 3 blocks at offset 8192 (0x2000)
... block 1 of 3
... block 2 of 3
... block 3 of 3
Verification
... success
Rebooting
Reboot Done|
```

6. Refer to [Running The Application](#) section of [test\\_app](#) to observe further output and next steps
7. Repeat Steps 4-6 once and jump to Step-8.
  - This step is to verify that bootloader is running after triggering bootloader from test\_app
8. Reset the device by holding the On-Board Switch mentioned in below table to force trigger bootloader at startup.
9. Repeat Steps 4-6 once.
  - This step is to verify whether bootloader is triggered by switch press at reset

| Kit Name                              | Switch Name |
|---------------------------------------|-------------|
| SAM C21N Xplained Pro Evaluation Kit  | SW0         |
| SAM D20 Xplained Pro Evaluation Kit   | SW0         |
| SAM D21 Xplained Pro Evaluation Kit   | SW0         |
| SAM E54 Xplained Pro Evaluation Kit   | SW0         |
| SAM E70 Xplained Ultra Evaluation Kit | SW0         |
| SAM L21 Xplained Pro Evaluation Kit   | SW0         |
| SAM L22 Xplained Pro Evaluation Kit   | SW0         |

## test\_app

This example application is used to demonstrate how it can be loaded from bootloader and trigger bootloader from firmware

### Description

- This application is a test application which resides from the end of bootloader size in device flash memory
- This application will be loaded into flash memory by bootloader application
- The application Blinks an LED and provides console output
- This application uses the On board Switch to trigger the bootloader from firmware
- Once the switch is pressed it loads first 16 bytes of RAM with bootloader request pattern and resets the device

## Building The Application

This section provides information on how to build an application using the MPLAB X IDE.

### Description

The parent folder for all the MPLAB X projects for this application is given below:

|                         |                                                          |
|-------------------------|----------------------------------------------------------|
| <b>Application Path</b> | <b>bootloader/apps/uart_bootloader/test_app/firmware</b> |
|-------------------------|----------------------------------------------------------|

To build the application, refer the following table and open the appropriate project file in the MPLAB X IDE.

| Project Name    | Description                           |
|-----------------|---------------------------------------|
| sam_c21n_xpro.X | SAM C21N Xplained Pro Evaluation Kit  |
| sam_d20_xpro.X  | SAM D20 Xplained Pro Evaluation Kit   |
| sam_d21_xpro.X  | SAM D21 Xplained Pro Evaluation Kit   |
| sam_e54_xpro.X  | SAM E54 Xplained Pro Evaluation Kit   |
| sam_e70_xult.X  | SAM E70 Xplained Ultra Evaluation Kit |
| sam_l21_xpro.X  | SAM L21 Xplained Pro Evaluation Kit   |
| sam_l22_xpro.X  | SAM L22 Xplained Pro Evaluation Kit   |

## MPLAB Harmony Configurations

This section provides information on the MHC configurations.

### Description

#### Components Used

- SysTick peripheral library
- STDIO Library

The following components used may vary based on the project configuration selected:

- USART or SERCOM peripheral library

#### Other MHC Settings:

- Provide Application Start Address in XC32-LD settings matching the bootloader application value
- Disable Fuse settings to generate binary out of the hex file

## Hardware Setup

This section describes how to configure the supported hardware.

### Description

#### 1. Project sam\_c21n\_xpro.X.

- **Hardware Used**
  - [SAM C21N Xplained Pro Evaluation Kit](#)
- **Hardware Setup**
  - No Special hardware Setup Required

#### 2. Project sam\_d20\_xpro.X.

- **Hardware Used**
  - [SAM D20 Xplained Pro Evaluation Kit](#)
- **Hardware Setup**
  - No Special hardware Setup Required

#### 3. Project sam\_d21\_xpro.X.

- **Hardware Used**
  - [SAM D21 Xplained Pro Evaluation Kit](#)
- **Hardware Setup**
  - No Special hardware Setup Required

#### 4. Project sam\_e54\_xpro.X.

- **Hardware Used:**
  - [SAM E54 Xplained Pro Evaluation Kit](#)
- **Hardware Setup**
  - No Special hardware Setup Required

#### 5. Project sam\_e70\_xult.X.

- **Hardware Used**
  - [SAM E70 Xplained Ultra Evaluation Kit](#)
- **Hardware Setup**
  - No Special hardware Setup Required

#### 6. Project sam\_l21\_xpro.X

- **Hardware Used**
  - [SAM L21 Xplained Pro Evaluation Kit](#)
- **Hardware Setup**
  - No Special hardware Setup Required

#### 7. Project sam\_l22\_xpro.X

- **Hardware Used**
  - [SAM L22 Xplained Pro Evaluation Kit](#)
- **Hardware Setup**
  - No Special hardware Setup Required

## Running The Application

This section provides information on how to run an application using the MPLAB X IDE.

### Description

1. Perform [Running The Application](#) steps for bootloader application if not done already.
2. If above step is successful then **LED should start blinking**
3. Open the Terminal application (Ex.:Tera Term) on the computer.
4. Configure the serial port settings as follows:
  - Baud : 115200
  - Data : 8 Bits
  - Parity : None
  - Stop : 1 Bit
  - Flow Control : None
5. Reset the device
6. **LED should start blinking** and you should see below output on the console

```
COM18 - Tera Term VT
File Edit Setup Control Window Help
##### Press and Hold the Switch to trigger Bootloader #####
█
```

The following table provides the LED and Switch names:

| Kit Name                              | LED Name | Switch Name |
|---------------------------------------|----------|-------------|
| SAM C21N Xplained Pro Evaluation Kit  | LED 0    | SW0         |
| SAM D20 Xplained Pro Evaluation Kit   | LED 0    | SW0         |
| SAM D21 Xplained Pro Evaluation Kit   | LED 0    | SW0         |
| SAM E54 Xplained Pro Evaluation Kit   | LED 0    | SW0         |
| SAM E70 Xplained Ultra Evaluation Kit | LED 1    | SW0         |
| SAM L21 Xplained Pro Evaluation Kit   | LED 0    | SW0         |
| SAM L22 Xplained Pro Evaluation Kit   | LED 0    | SW0         |

7. Press and hold the Switch to trigger Bootloader and you should see below output

```
COM18 - Tera Term VT
File Edit Setup Control Window Help
##### Press and Hold the Switch to trigger Bootloader #####
##### Bootloader Triggered #####
##### Disconnect console to program new firmware from Bootloader #####
█
```

## uart\_fail\_safe\_bootloader

## bootloader

This example application shows how to use the Bootloader Library to bootload an application on device having dual flash bank feature using UART protocol

### Description

- This application is a fail safe bootloader application which resides in the starting location of the both the banks of device flash memory
- The application calls the `bootloader_Task()` which receives application to be programmed into opposite bank of flash memory over UART channel
- The application uses the Virtual Com port of the device (EDBG port) to receive application binary from host PC
- This application uses the On board Switch as bootloader trigger pin to force enter the bootloader at reset of device.

## Building The Application

This section provides information on how to build an application using the MPLAB X IDE.

### Description

The parent folder for all the MPLAB X projects for this application is given below:

| Application Path | bootloader/apps/uart_fail_safe_bootloader/bootloader/firmware |
|------------------|---------------------------------------------------------------|
|------------------|---------------------------------------------------------------|

To build the application, refer the following table and open the appropriate project file in the MPLAB X IDE.

| Project Name   | Description                         |
|----------------|-------------------------------------|
| sam_e54_xpro.X | SAM E54 Xplained Pro Evaluation Kit |

## MPLAB Harmony Configurations

This section provides information on the MHC configurations.

### Description

#### Components Used

- Bootloader
- SysTick peripheral library
- NVMCTRL peripheral library
- SERCOM peripheral library
- DSU

#### Other MHC Settings:

- Provide Application Start Address in XC32-LD settings

## Hardware Setup

This section describes how to configure the supported hardware.

### Description

1. **Project sam\_e54\_xpro.X.**

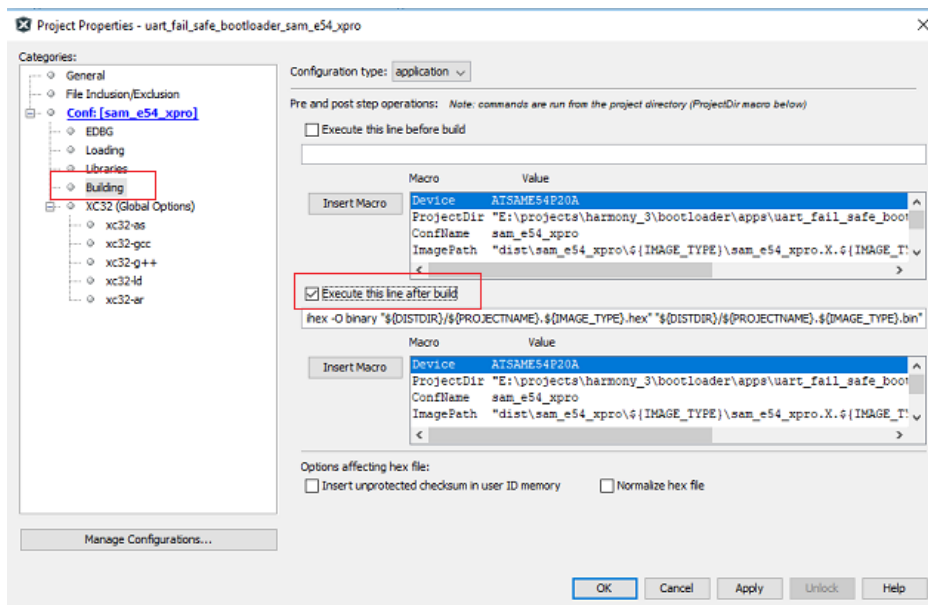
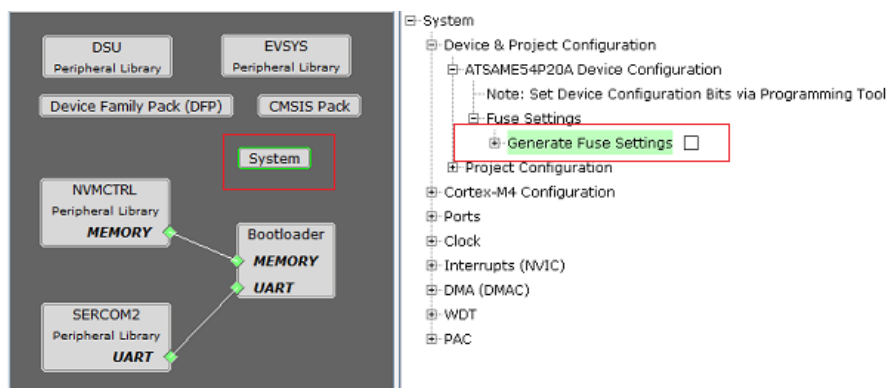
- **Hardware Used:**
  - [SAM E54 Xplained Pro Evaluation Kit](#)
- **Hardware Setup**
  - No Special hardware Setup Required

## Running The Application

This section provides information on how to run an application using the MPLAB X IDE.

### Description

1. Connect a micro USB cable to the DEBUG port.
2. Build and program the application using the MPLAB X IDE.
3. Launch MHC for the bootloader application
  - **Disable Fuse Settings**
  - Enable **Execute this line After Build** option in MPLAB X Project properties->Building option



4. Build the bootloader application again using MPLAB X IDE
  - This is required to generate the binary file for bootloader application
5. Build the Application [test\\_app](#) using MPLAB X IDE
6. Run the [btl\\_app\\_merge.py](#) from command prompt to merge the generated Bootloader binary and application binary. Refer to [Bootloader Tools Help](#) for setting up the host script

```
python <harmony3_path>\bootloader\tools\bt1_app_merge.py -o <Offset> -b
<harmony3_path>\bootloader\apps\uart_fail_safe_bootloader\bootloader\firmware\<Project
Name>\dist\<Config Name>\production\<Project Name>.production.bin -a
<harmony3_path>\bootloader\apps\uart_fail_safe_bootloader\test_app\firmware\<Project
Name>\dist\<Config Name>\production\<Project Name>.production.bin
```

7. Run the **bt1\_host.py** from command prompt to program the merged binary to opposite panel. Refer to [Bootloader Tools Help](#) for setting up the host script

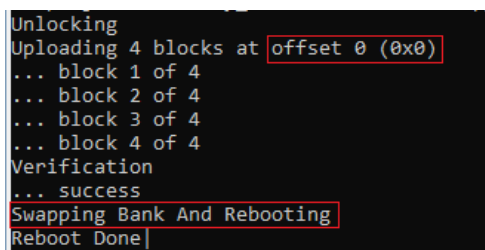
```
python <harmony3_path>\bootloader\tools\bt1_host.py -v -s -i <COM PORT> -d <Device Name> -f
<harmony3_path>\bootloader\tools\bt1_app_merged.bin
```

The following table provides the Details required to run the above command:

| Kit Name                            | Offset | Device Name | Project Name   | Config Name  |
|-------------------------------------|--------|-------------|----------------|--------------|
| SAM E54 Xplained Pro Evaluation Kit | 0x2000 | same5x      | sam_e54_xpro.X | sam_e54_xpro |

8. Following snapshot shows example output of successfully programming the merged binary.

- **Swapping Bank And Rebooting** and **Reboot Done** messages in below output signifies that bootloading is successful



```
Unlocking
Uploading 4 blocks at offset 0 (0x0)
... block 1 of 4
... block 2 of 4
... block 3 of 4
... block 4 of 4
Verification
... success
Swapping Bank And Rebooting
Reboot Done
```

9. Refer to [Running The Application](#) section of **test\_app** to observe further output and next steps

10. Repeat Steps 7-9 once

- This step is to verify that bootloader is running after triggering bootloader from test\_app
- Also to program the new firmware in opposite bank
- You should see other Bank in test\_app console displayed compared to first run

## test\_app

This example application is used to demonstrate how it can be loaded from bootloader and trigger bootloader from firmware

### Description

- This application is a test application which resides from the end of bootloader size in device flash memory
- This application will be loaded into flash memory by bootloader application
- The application Blinks an LED and provides console output
- This application uses the On board Switch to trigger the bootloader from firmware
- Once the switch is pressed it loads first 16 bytes of RAM with bootloader request pattern and resets the device

## Building The Application

This section provides information on how to build an application using the MPLAB X IDE.

### Description

The parent folder for all the MPLAB X projects for this application is given below:

|                  |                                                             |
|------------------|-------------------------------------------------------------|
| Application Path | bootloader/apps/uart_fail_safe_bootloader/test_app/firmware |
|------------------|-------------------------------------------------------------|

To build the application, refer the following table and open the appropriate project file in the MPLAB X IDE.

| Project Name   | Description                         |
|----------------|-------------------------------------|
| sam_e54_xpro.X | SAM E54 Xplained Pro Evaluation Kit |

## MPLAB Harmony Configurations

This section provides information on the MHC configurations.

### Description

#### Components Used

- SysTick peripheral library
- STDIO Library
- SERCOM peripheral library

#### Other MHC Settings:

- Provide Application Start Address in XC32-LD settings matching the bootloader application value
- Disable Fuse settings to generate binary out of the hex file

## Hardware Setup

This section describes how to configure the supported hardware.

### Description

1. **Project sam\_e54\_xpro.X.**
  - **Hardware Used:**
    - [SAM E54 Xplained Pro Evaluation Kit](#)
  - **Hardware Setup**
    - No Special hardware Setup Required

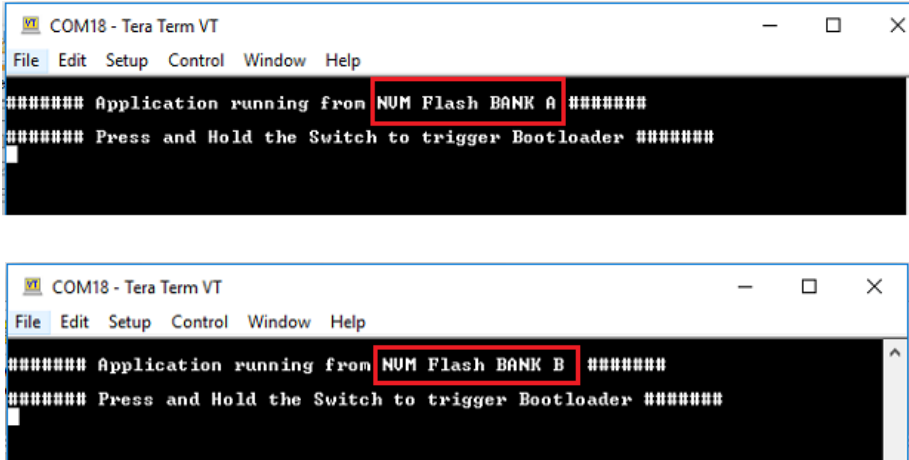
## Running The Application

This section provides information on how to run an application using the MPLAB X IDE.

### Description

1. Perform [Running The Application](#) steps for bootloader application if not done already.
2. If above step is successful then **LED should start blinking**
3. Open the Terminal application (Ex.:Tera Term) on the computer.
4. Configure the serial port settings as follows:
  - Baud : 115200
  - Data : 8 Bits
  - Parity : None
  - Stop : 1 Bit
  - Flow Control : None
5. Reset the device
6. **LED should start blinking** and you should see below output on the console
  - The NVM Flash Bank Can be BANK A or BANK B based on from where the program is running

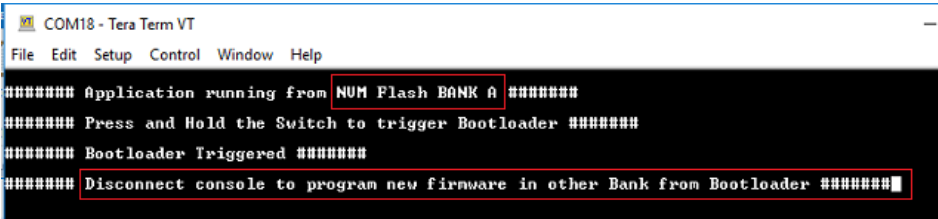




The following table provides the LED and Switch names:

| Kit Name                            | LED Name | Switch Name |
|-------------------------------------|----------|-------------|
| SAM E54 Xplained Pro Evaluation Kit | LED 0    | SW0         |

7. Press and hold the Switch to trigger Bootloader to program firmware in other bank and you should see below output



## Index

### A

Application Configurations 16

### B

bootloader 22, 29

Bootloader Applications Help 22

Bootloader Configurations 14

Bootloader Execution Flow 7

Bootloader Framework 2

Bootloader Library Help 2

Bootloader Memory Layout 12

Bootloader Protocol 4

Bootloader Sizing and Considerations 17

Bootloader Tools Help 20

Bootloader Trigger Methods 6

bootloader\_Start function 20

bootloader\_Trigger function 19

Building The Application 23, 26, 29, 31

### C

Configuring the Library 14

### H

Hardware Setup 23, 27, 29, 32

How The Library Works 3

### I

Introduction 2

### L

Library Interface 18

### M

MPLAB Harmony Configurations 23, 26, 29, 32

### R

run\_Application function 19

Running The Application 24, 28, 30, 32

### T

test\_app 26, 31