Application Note

MSPM0 Live Firmware Update (LFU) Bootloader Implementation

Gary Gao and Luke Chen

ABSTRACT

This application note provides a method to do the firmware update without suspend the application code. It is based on MSPM0G3507 and using FreeRTOS to do the tasks handling. It also provides a PC GUI as the host and can help to generate using files for this demo.

Project collateral and source code discussed in this document can be downloaded using this URL: https://www.ti.com/lit/zip/slaaec9.

Table of Contents

List of Figures

List of Tables

Trademarks

FreeRTOS™ is a trademark of Benchmarq.
Code Composer Studio™ is a trademark of Texas Instruments.
All trademarks are the property of their respective owners.

1 Introduction

MSPM0 devices support ROM-based BSL (Boot-Strap Loader), flash-based bootloader and plug-in interface that can be used for firmware upgrade. However, these bootloaders will occupy CPU during the firmware upgrade period of time, means that the executing application code is suspended until the firmware upgrade process is completed.

In some applications, suspending application code during firmware upgrade process is not allowed. This application report provides a method not suspend the application code during firmware upgrade period. When firmware upgrade process is completed, power cycle the device and newer version of firmware will be executed.

2 LFU Bootloader Features Overview

Key features of the LFU Bootloader include:

- Communicates with Host via universal asynchronous receiver/transmitter (UART) port
- FreeRTOS™-based software example
- Software BSL invoke
- Password protection enabled
- Easy to use Windows Host GUI
- Automatic linker and header files generation, easy to migrate to other MSPM0 devices
- Non-main flash modification solution provided
- The code size of this bootloader is less than 16 KB in size.

3 Hardware and Software Setup

3.1 Hardware Requirement

- LP-MSPM0G3507 Launchpad
- PC with windows
- Miro USB type cable

3.2 Software Setup

- Code Composer Studio™ (CCS) 12.3 or later
- Uniflash 8.3 or later
- MSPM0 SDK

UART Interface setting

- Baud rate 9600 bps
- Data width - 8 bit
- One stop bit
- No parity

4 LFU Bootloader Implementation

4.1 LFU Bootloader and Application Projects

In this application note, three projects are used as described in Table 4-1.

Table 4-1. Projects Needed in This Demo

CCS Projects	Description
LFU Bootloader	This is a FreeRTOS based bootloader software and is around 16 KB in size, it is allocated to the starting address of main Flash memory. On a new device, this bootloader needs to be programmed via SWD interface first before starting firmware upgrade process.
Application code 1	This is application code 1 for firmware upgrade demo, it uses application space 1 (0x04000 - 0x11FFF) for function.
Application code 2	This is application code 2 for firmware upgrade demo, it uses application space 2 (0x12000 - 0x1FFFF) for function.

4.2 Memory Allocation

The main Flash memory is divided into three parts:

LFU bootloader:

The first 16 KB (0x0000 - 0x3FFF) of main flash memory is reserved for LFU bootloader, the other main flash space is assigned to application code.

Application code:

This memory space is divided into application space 1 and application space 2, only one version of application code is executed, the other application code space is used for new firmware upgrade purpose.

The first 4 KB (0x2020000 - 0x20200FFF) of RAM space is reserved for bootloader and the other memory space can be used for application code.

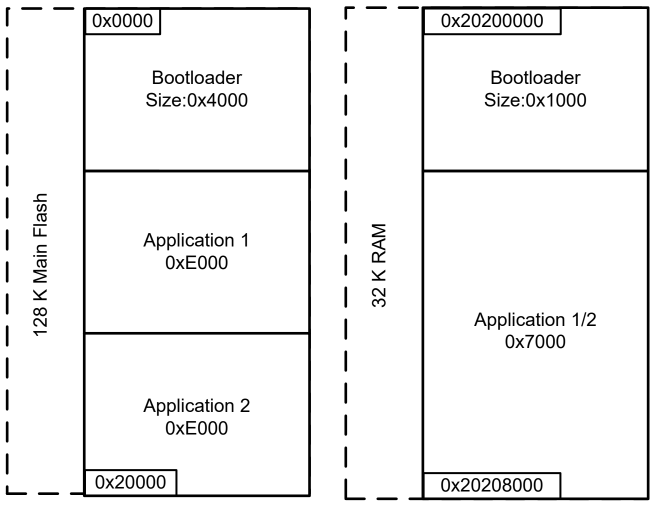


Figure 4-1. Memory Arrangement

4.3 LFU Bootloader Implemented

One task switch management tool is needed so that the application code keeps executing during firmware upgrade process. In this application note, FreeRTOS is used to handle this job and create four tasks for this demo.

Table 4-2. The Tasks in LFU Bootloader

Tasks Name	Priority	Description
Bootloader Task	2	Task to handle firmware upgrade. This task is suspended if application code is executing.
LED0 Toggle Task	1	Task to toggle LED every 500ms, this is used to show that FreeRTOS is working without problems.
Application task	1	Task to execute application code.
Idle	0	Default task when no any other task is pending for execution.

On a new device, the bootloader firmware needs to be programmed via SWD interface first, so that bootloader communicates with the host (PC GUI or the host MCU) and update the application code. The application code can also be programmed along with bootloader firmware via SWD.

When the device boots up and there is no application code, the bootloader waits for the firmware upgrade command from the host.

Figure 4-2 illustrates the bootloader flow diagram.

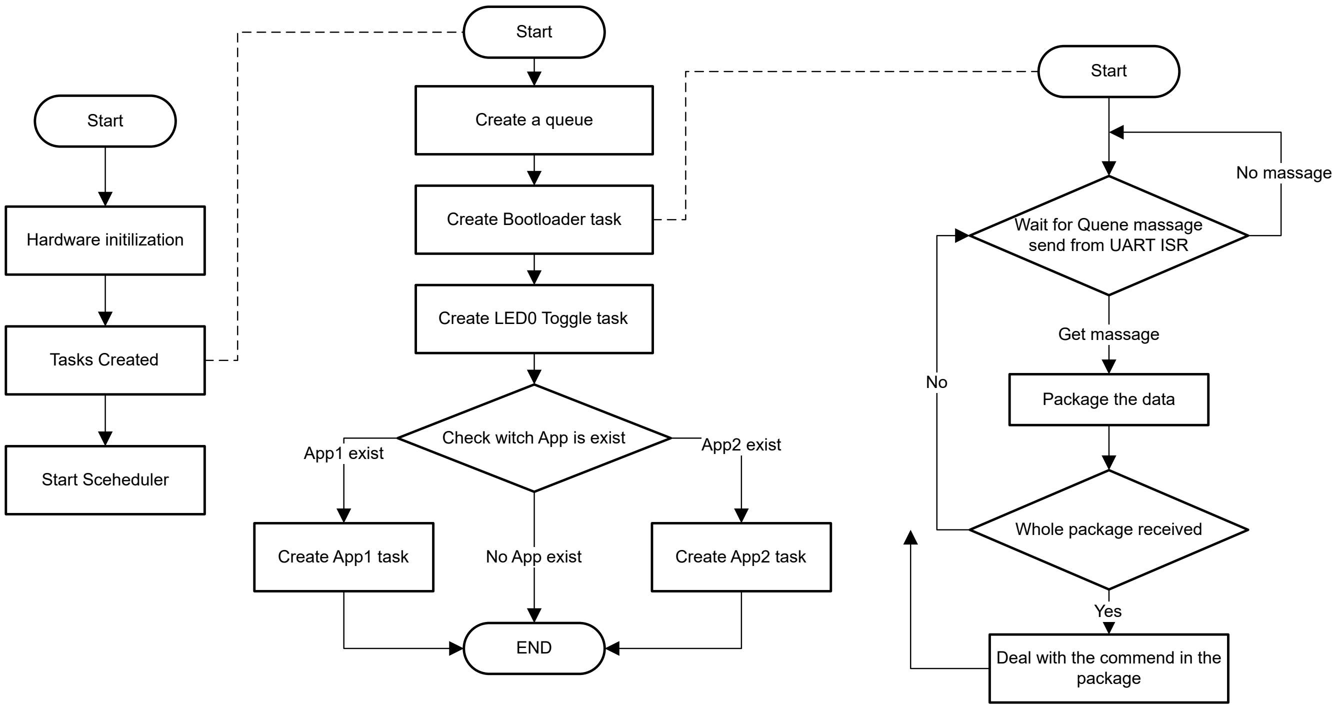


Figure 4-2. Bootloader Code Progress Diagram

The bootloader supports below commands. For detailed information, see Section 6.

- CMD_UNLOCK_BSL
- CMD_FLASH_RANGE_ERASE
- CMD_PROGRAM_DATA
- CMD_START_APPLICATION

4.4 LFU Application Code Implementation

There are two application projects: the application code 1 and application code 2 in different flash areas. These two projects toggle different LED. The application code can be called by the FreeRTOS in bootloader as a task thread. You must use the delay function defined in FreeRTOS for time delay purpose in the application code. In this demo example, the bootloader project put the FreeRTOS delay function's start address at fixed flash address 0x3FF0 to approximately 0x3FFF that is called shared area, the application project code can just to call the delay function just pick the start address in the shared area.

4.4.1 The Linker Command File for Application

Linker command file is used by linker during the project building process, the PC GUI tool can automatic generate this file, no need to take care of this linker command file for the projects.

4.4.2 Peripheral and Interrupt Initialization

The bootloader initializes some modules like clock module, UART0 and power modules. It is recommended not to reinitialize those modules already initialized by bootloader, otherwise, the settings by bootloader could lost.

The below function is used to register interrupt. It copies the interrupt table to SRAM and modify the ISR start address based on the input parameters.

```
void DL_Interrupt_registerInterrupt(uint32_t exceptionNumber, void (*intHandler)(void));
```

4.4.3 Debug for Application Project

The application projects cannot be downloaded and executed by CCS directly due to below reasons:

- The interrupt vector table in generated linker command (cmd) file is not assigned to default address 0x0000 for device bootup.
- No device module initialization function in application projects.
- No FreeRTOS delay function can be used in the flash shared area.

Here is the recommended workaround to debug application projects.

- Use the default cmd file in the SDK examples that interrupt vector table is assigned to address 0x0000 for bootup.
- Use sysconfig tool to generate peripheral initialization code for application projects.
- Replace FreeRTOS delay function with the function delay_cycles(); that is defined in SDK.

4.5 Invoke Firmware Upgrade Process

There are two cases when you want to start firmware upgrade process:

- If there is only bootloader code executed in the device, the bootloader task waits for the firmware upgrade command from host, so no extra action is required before starting firmware upgrade process in this case.
- If there are both bootloader and application code executed in the device, the bootloader task is suspended when application code is executed. In this case, you need to force bootloader task status from suspend to active before starting firmware upgrade process, you can send one byte invoke command 0xAA via UART to achieve this goal.

5 Host GUI Tool Introduction

The host can be one MCU, processor or PC that sending the commands via UART port. For detailed host commands information, see Section 6.

In this demo, use the on-board emulator XDS110 to handle the USB to UART function, so that you can use the PC as the host and send UART commands via USB port. The Python created PC GUI tool is used to send UART commands, there are three functions of this GUI tool.

- LFU firmware grade.
- Application project linker command file generation.
- Non-main flash configuration firmware generation.

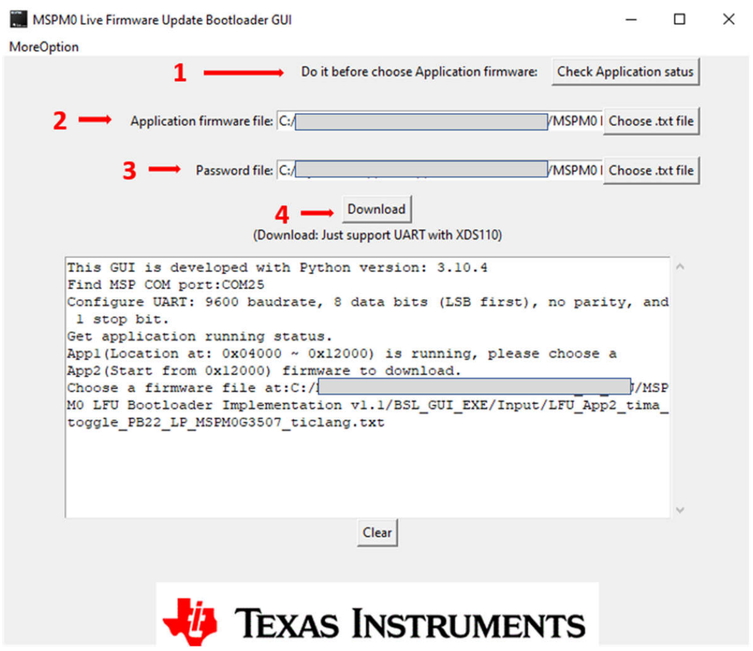
5.1 LFU Firmware Update

Confirm the items below before starting LFU firmware upgrade process:

- Make sure bootloader firmware already programmed into device via SWD interface.
- Build application project and generate the .txt file needed for firmware upgrade. Suggest generating both application code 1 and application code 2 for evaluation purpose.
- Prepare .txt format BSL password file.

Now you can launch PC GUI tool by double-click the file MSPM0_LFU_BSL_GUI.exe at the folder "…\MSPM0 LFU Bootloader Implementation v1.1\BSL_GUI_EXE". The following steps describe how to perform firmware upgrade process:

1. Check application code status, which application code is executed or no application code in the device.
2. Select application code for firmware upgrade evaluation.
   a. If no application code in the device, you can select either application code 1 or application code 2 for firmware upgrade.
   b. If application code 1 is executed in the device, select application code 2 for firmware upgrade.
   c. If application code 2 is executed in the device, select application code 1 for firmware upgrade.
3. Choose a password file based on the format of the default one in the input folder.
4. Click the download button to do the firmware update.



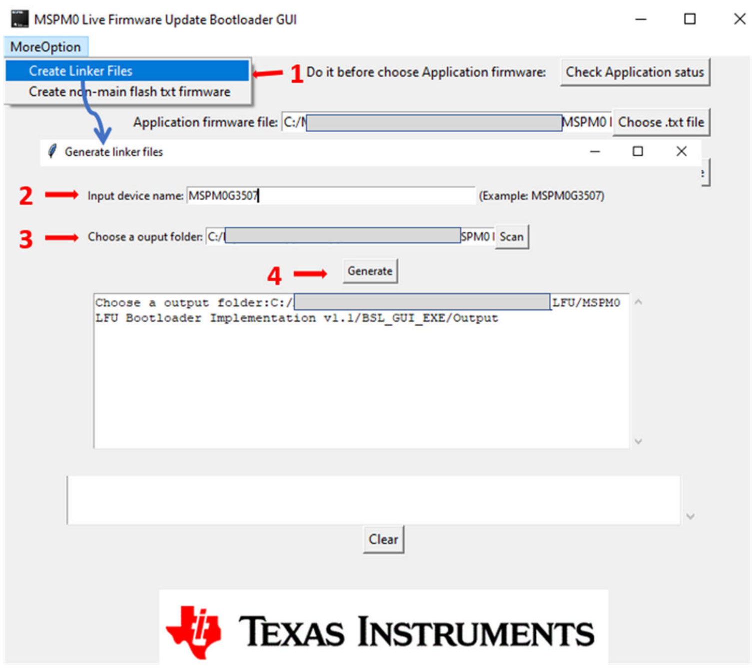
```
This GUI is developed with Python version: 3.10.4
Find MSP COM port:COM25
Configure UART: 9600 baudrate, 8 data bits (LSB first), no parity, and
 1 stop bit.
Get application running status.
App1(Location at: 0x04000 ~ 0x12000) is running, please choose a
App2(Start from 0x12000) firmware to download.
Choose a firmware file at:C:/[redacted]/MSP
M0 LFU Bootloader Implementation v1.1/BSL_GUI_EXE/Input/LFU_App2_tima_
toggle_PB22_LP_MSPM0G3507_ticlang.txt
```

Figure 5-1. Steps to Update Firmware With the GUI

5.2 Application Project Link Files Generation

If you would like to migrate this example project to any other MSPM0 devices, you need a different linker command file of application code for the specific device. The GUI tool can do this, automatic generate the cmd files according to the MSPM0 part number you input. Use MSPM0G3507 as an example and follow the steps below to generate the needed files:

1. In GUI tool, click the More Option menu and select the option - create linker files.
2. Enter MSPM0 part number, for example MSPM0G3507.
3. Select one folder to save the generated files.
4. Click the Generate button to generate the files.


MSPM0 Live Firmware Update Bootloader GUI
MoreOption
Create Linker Files
Create non-main flash txt firmware
1 Do it before choose Application firmware:
Check Application satus
Application firmware file: C:/
Choose .txt file
Generate linker files
2 Input device name: MSPM0G3507
(Example: MSPM0G3507)
3 Choose a ouput folder: C:/
Scan
4 Generate
Clear
TEXAS INSTRUMENTS

```
Choose a output folder:C:/ LFU/MSPM0
LFU Bootloader Implementation v1.1/BSL_GUI_EXE/Output
```

Figure 5-2. Steps to Generate Link Files With the GUI

5. The following three files are generated:
    a. mspm0g3507_App1.cmd, this is the cmd file for application code 1.
    b. mspm0g3507_App2.cmd, this is the cmd file for application code 2.
    c. device.h, this is the needed file for bootloader and both application codes.

5.3 Non-Main Flash Configuration Firmware Generation

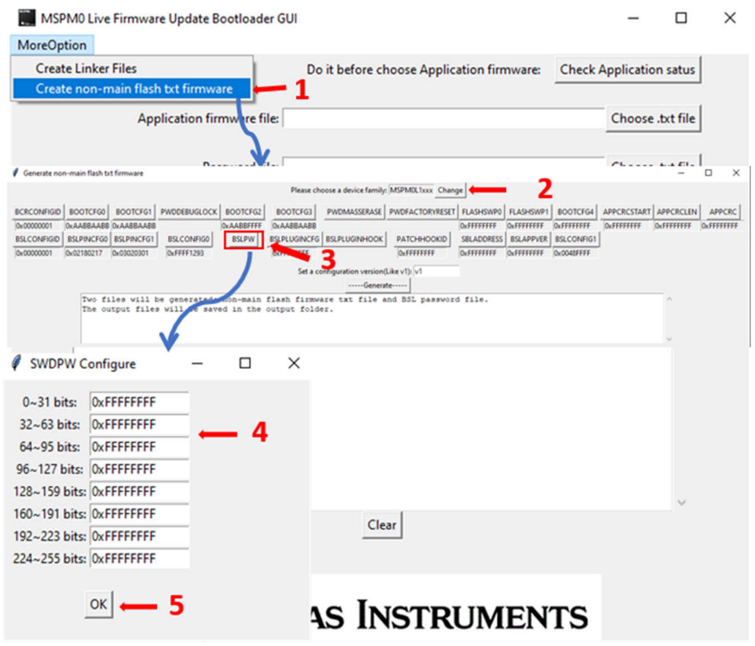
The NONMAIN is a dedicated region of flash memory which stores the configuration data used by the BCR(boot configuration routine) and BSL to boot the device. The password used in this example project can also be used for BSL, if you want to change the password, you need to modify NONMAIN flash configuration.

The GUI tool supports not only BSL password modification, but also all the NONMAIN flash configuration options.

5.3.1 Steps to Generate the Non-Main Flash Configuration Firmware

Below are the steps to modify the password with the GUI:

1. Click the More Option menu and select the option - Create non-main flash txt firmware.
2. Click change button if need change to other device family.
3. Click the BSLPW button.
4. Enter your new BSL password.
5. Click the OK button to save your new password, then close this dialog window.
6. Enter your new version number for this modification.
7. Click the Generate button and generate NONMAIN flash configuration data and password file.


MSPM0 Live Firmware Update Bootloader GUI
MoreOption
Create Linker Files
Create non-main flash txt firmware
1
Do it before choose Application firmware:
Check Application satus
Application firmware file:
Choose .txt file
Generate non-main flash txt firmware
2
3
4
5
SWDPW Configure
OK
Clear
-----Generate-----

Please choose a device family: MSPM0L1xxx Change

BCRCONFIGID 0x00000001
BOOTCFG0 0xAABBAABB
BOOTCFG1 0xAABBAABB
PWDDEBUGLOCK
BOOTCFG2 0xAABBFFFF
BOOTCFG3 0xAABBAABB
PWDMASSERASE
PWDFACTORYRESET
FLASHSWP0 0xFFFFFFFF
FLASHSWP1 0xFFFFFFFF
BOOTCFG4 0xFFFFFFFF
APPCRCSTART 0xFFFFFFFF
APPCRCLEN 0xFFFFFFFF
APPCRC 0xFFFFFFFF
BSLCONFIGID 0x00000001
BSLPINCFG0 0x02180217
BSLPINCFG1 0x03020301
BSLCONFIG0 0xFFFF1293
BSLPW
BSLPLUGINCFG
BSLPLUGINHOOK
PATCHHOOKID 0xFFFFFFFF
SBLADDRESS 0xFFFFFFFF
BSLAPPVER 0xFFFFFFFF
BSLCONFIG1 0x0048FFFF

Set a configuration version(Like v1): v1

```
Two files will be generated: non-main flash firmware txt file and BSL password file.
The output files will be saved in the output folder.
```

0~31 bits:	0xFFFFFFFF
32~63 bits:	0xFFFFFFFF
64~95 bits:	0xFFFFFFFF
96~127 bits:	0xFFFFFFFF
128~159 bits:	0xFFFFFFFF
160~191 bits:	0xFFFFFFFF
192~223 bits:	0xFFFFFFFF
224~255 bits:	0xFFFFFFFF

Figure 5-3. Steps to Modify Password With the GUI

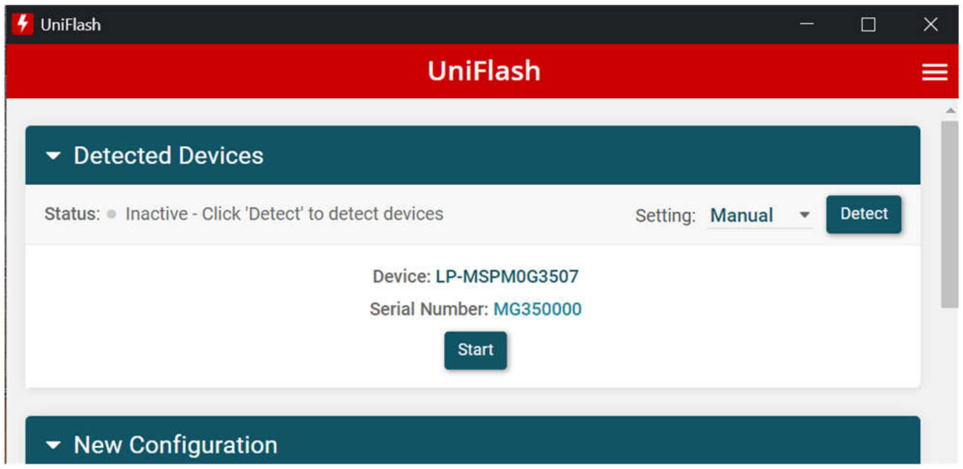
There are two generated files saved at the default output folder as follows:

- Non_main_flash_firmware_v1.txt
- BSL_Password_v1.txt

You can use UNIFLASH tool to program the Non_main_flash_firmware_v1.txt into device via SWD interface, configure the NONMAIN flash region. The BSL_Passward_v1.txt file is used for GUI tool firmware upgrade process.

5.3.2 UNIFLASH Tool to Program the NONMAIN Flash Configuration Data

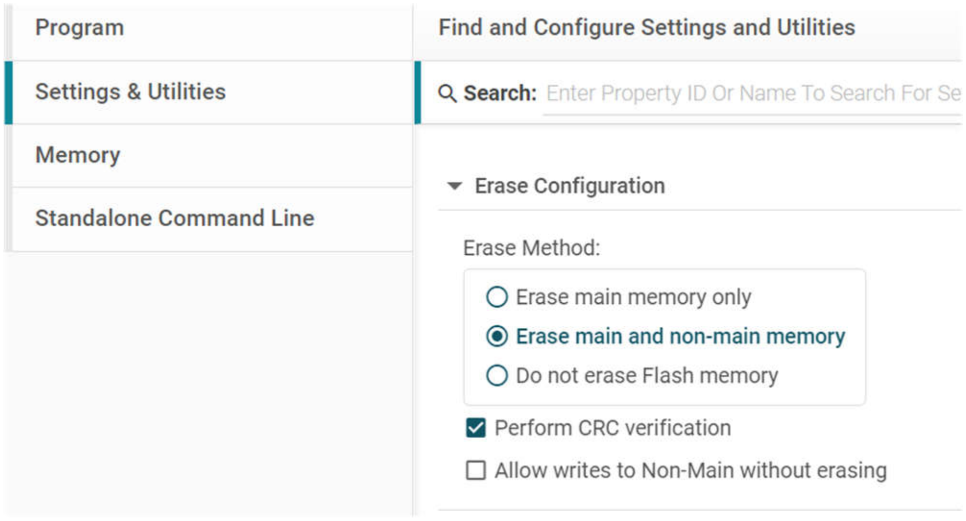
1. Connect LP-MSPM0G3507 to PC and launch UNIFLASH tool.
2. Manually select the EVM or leave UNIFLASH to detect the part number automatically.



Status: Inactive - Click 'Detect' to detect devices
Setting: Manual
Device: LP-MSPM0G3507
Serial Number: MG350000

Figure 5-4. Board Detected by the Uniflash

3. Select Settings and Utilities option, then check Erase main and non-main memory.


Program
Settings & Utilities
Memory
Standalone Command Line
Find and Configure Settings and Utilities
Search: Enter Property ID Or Name To Search For Se
Erase Configuration

Erase Method:

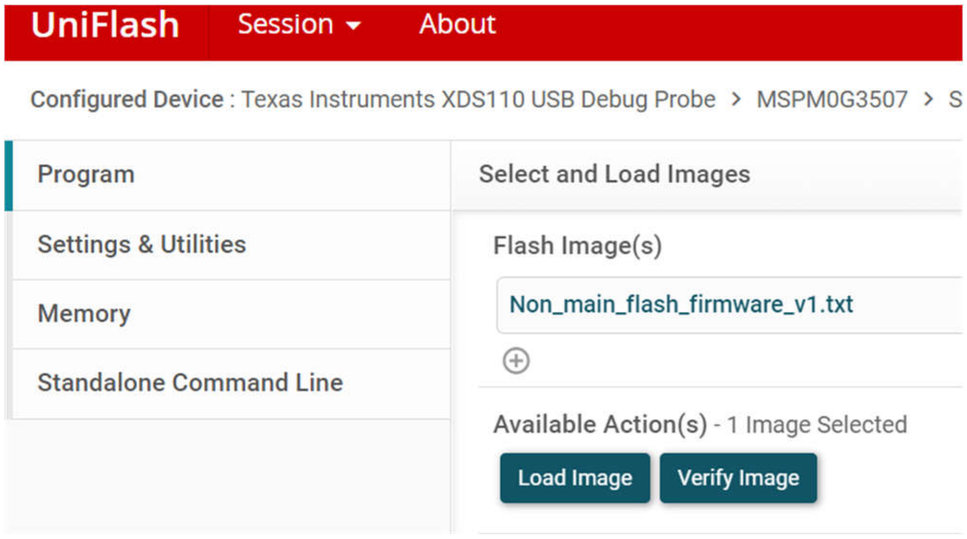
- ○ Erase main memory only
- ● Erase main and non-main memory
- ○ Do not erase Flash memory

☑ Perform CRC verification

☐ Allow writes to Non-Main without erasing

Figure 5-5. Change Erase Method

4. Select Program option, browse the generated file Non_main_flash_firmware_v1.txt and then click Load Image button.(If enabled the static flash protection, recommend to add the images need to download the area that be static protected).



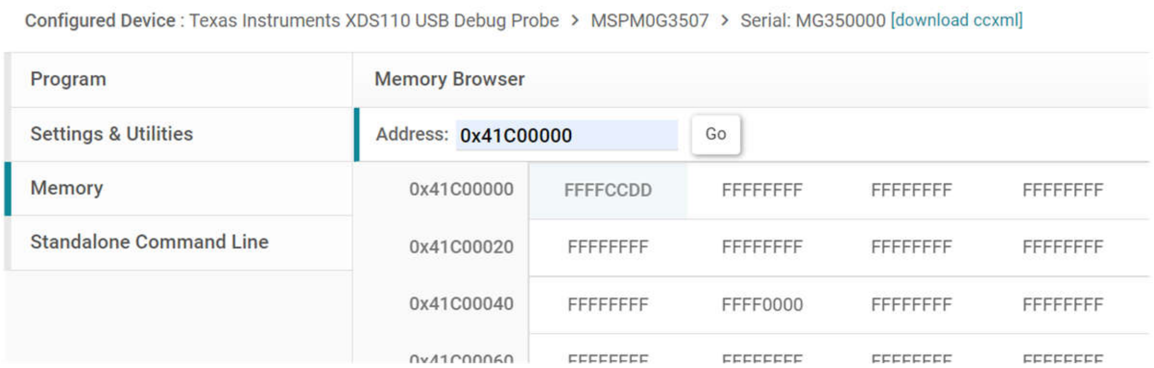
Configured Device : Texas Instruments XDS110 USB Debug Probe > MSPM0G3507 > S

Non_main_flash_firmware_v1.txt

Available Action(s) - 1 Image Selected

Figure 5-6. Download Non-Main Flash Firmware

5. Select Memory option, enter 0x41C00000 in Address field and read the data from device, check whether or not you successfully programmed the new configuration data.



Configured Device : Texas Instruments XDS110 USB Debug Probe > MSPM0G3507 > Serial: MG350000 [download ccxml]

Address: 0x41C00000

```
0x41C00000   FFFFCCDD   FFFFFFFF   FFFFFFFF   FFFFFFFF
0x41C00020   FFFFFFFF   FFFFFFFF   FFFFFFFF   FFFFFFFF
0x41C00040   FFFFFFFF   FFFF0000   FFFFFFFF   FFFFFFFF
0x41C00060   FFFFFFFF   FFFFFFFF   FFFFFFFF   FFFFFFFF
```

Figure 5-7. Verify the Firmware in the Memory Readback

6 LFU Bootloader Protocol

This section discusses bootloader protocol, how to uses UART commands and completes the firmware upgrade process.

6.1 Packet Format and Core Commands

The packet format of LFU bootloader is identical with ROM-based BSL. For more details, see the *MSPM0 Bootloader User's Guide*. LFU bootloader supports the core commands listed in Table 6-1.

Table 6-1. LFU BSL Core Commands

LFU BSL Command	Protected	CMD	Start Address	Data(bytes)	Core Response
CMD Unlock Bootloader	No	0x21	-	D1…D32(Password)	Yes
CMD Flash Range Erase	Yes	0x23	A1...A4	A1...A4 (End address)	Yes
CMD Program Data	Yes	0x20	A1...A4	D1…Dn	Yes
CMD Start application	No	0x40	-	-	No

6.2 Special Commands in LFU Bootloader

There are some special commands different from the ROM-based BSL commands, they are one-byte commands and are listed in Table 6-2.

Table 6-2. LFU BSL Special Commands

LFU BSL Special Commands	Packet	Response (package format)
Get App state	0x55	Yes (8 bytes: 0x51 + app state flag (1 byte) + app1 area start address (3 bytes) + app2 area start address (3 bytes))
Resume bootloader task	0xAA	Yes (1 byte: 0xBB)

- **Get App state** command:

  This command is used to get the application code execution status, to see if application code 1 or application code 2 is executed, or no application code in the device. Table 6-3 describes the application state flags.

  Table 6-3. Application State Flags

Variable	Value	Description
App state flag	0	No application running
	1	App1(application in app1 area) is running
	2	App2(application in app2 area) is running

  This command also returns the start address of application space 1 and space 2. This information can be used to do system integrity verification before starting firmware upgrade process
- **Resume bootloader task** command:

  Bootloader task is suspended when device is executing application code, this command is used to force bootloader task back to active status before starting firmware upgrade process.

6.3 Host Device Firmware Upgrade Flow

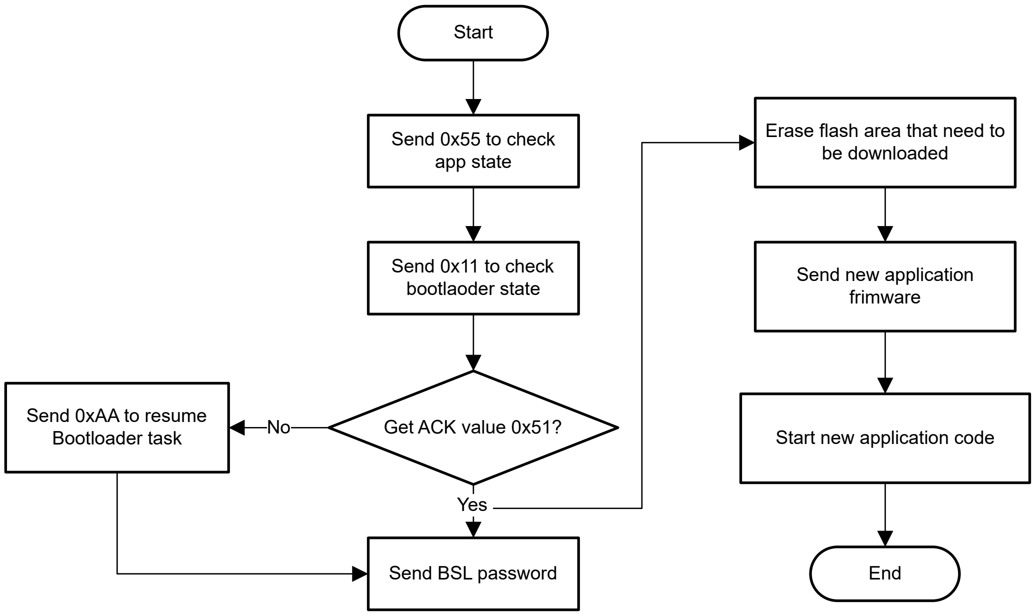

Start
Send 0x55 to check app state
Send 0x11 to check bootlaoder state
Get ACK value 0x51?
No
Yes
Send 0xAA to resume Bootloader task
Send BSL password
Erase flash area that need to be downloaded
Send new application frimware
Start new application code
End

Figure 6-1. LFU Host Operation Flow

Below are two steps need to be performed before starting firmware upgrade process:

1. Send one-byte command 0x55 to check application code execution status, so that you know which application code need to be upgraded.
2. Send one-byte command 0x11 to check bootloader status, if device responds 0x51, means that bootloader is in active status and is ready for firmware upgrade. Otherwise, you need to send one-byte command 0xAA to force bootloader from suspend to active status first.

When you are sure bootloader is in active status, send the password to unlock the device first, then erase the needed flash space and program new application code.

Here are tips to send the application code.

- Make sure the application code .txt file is 16-byte aligned. The host software is responsible to fill unused space with dummy data bytes, make sure the .txt file is 16-byte aligned, see Figure 6-2 and Figure 6-3.

```
@4130
C6 E0 70 47 C4 E0 00 BF 00 BF 00 BF 00 BF 00 BF
00 BF 00 BF
```

Figure 6-2. Unaligned Firmware

```
@4130
C6 E0 70 47 C4 E0 00 BF 00 BF 00 BF 00 BF 00 BF
00 BF 00 BF FF FF FF FF FF FF FF FF FF FF FF FF
```

Figure 6-3. Aligned Firmware

- Send the first line (first 16 bytes in the firmware) of the new application's firmware in the end. This is to avoid the case an uncompleted firmware be executed unexpected. For current bootloader just to check if the first 8 bytes are all 0xFF to check if the application firmware exist, it will not check if the firmware to be completed. If you send the first line of the firmware in the end, that makes sure that the whole firmware has been sent successfully.

After programming new application code, send the command CMD start application to execute new application code.

7 Migration to Other MSPM0 Devices

To migration the demos to other devices, both bootloader projects and application projects need to be modified. The GUI in this demo can help to generate the linker files and device.h header file for application and bootloader projects. Besides that, the low level peripheral configuration (ti_msp_dl_config.c and .h files) and the communication interface (uart.c and .h files) need to be modified based on the specific device.

8 References

- Texas Instruments: *MSPM0 Bootloader User's Guide*
- *MSPM0 G-Series 80-MHz Microcontrollers Technical Reference Manual*
- Texas Instruments: *MSPM0 Bootloader (BSL) Host Implementation*