Application Note

MSPM0 Bootloader (BSL) Implementation

Chao Gao

ABSTRACT

This application note provides application level description for bootloader(BSL) for MSPM0 devices. It summarizes the resources related with BSL about MSPM0 and provide the step by step usages for the BSL examples or tools in SDK. For more details about the ROM-based BSL, see the *MSPM0 Bootloader User's Guide*.

Table of Contents

List of Figures

List of Tables

Trademarks

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

1 Introduction

1.1 Bootloader Introduction

1.1.1 Bootloader Concept

A microcontroller bootloader can be used to program the internal memory of the MCU using common interfaces like universal asynchronous receiver/transmitter (UART) or inter-integrated circuit (I2C). A bootloader enables quick and easy programming of the device through the entire life cycle. The firmware update structure through BSL showed as Figure 1-1. Based on Figure 1-1, the new firmware can be downloaded into MSPM0 device by a BSL host that can be a PC or processor with the interface like UART, I2C and so on.

In this application note, the MCU being programmed is called the target, and the device or tool performing the update is called the host.

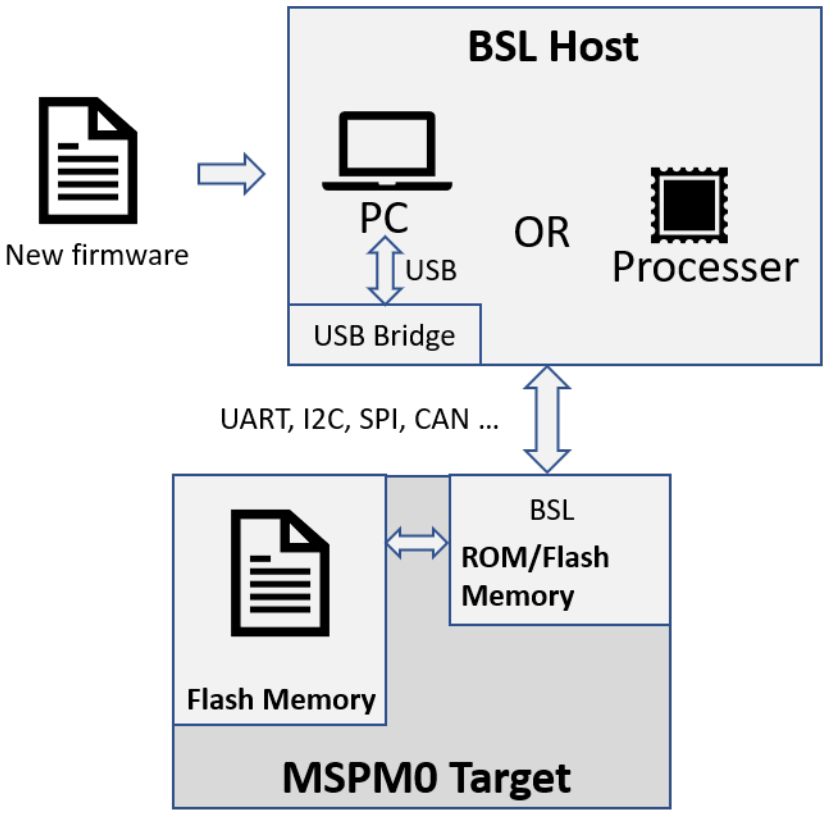

Figure 1-1. The Firmware Update Structure Through BSL

1.1.2 MSPM0 Bootloader Structure

There are three kind of Bootloader solutions provided with MSPM0 devices: ROM-based BSL, ROM-based BSL with flash based plug-in interface and flash-based secondary BSL. Just choose one of the three solutions based on the application's requirement. Both of the solutions use the same invoke mode (general-purpose input/output (GPIO) invoke, blank-device detection and software invoke). There are some parameters that need to be configured in the NON-MAIN flash. For more details, see Section 2.

Table 1-1. MSPM0L and MSPM0G BSL Solutions Summary

BSL Solutions	ROM Cost	Flash Cost (By default)	Interface	Pins Used With Hardware Invoke	Pins Used With Software Invoke	Using Case
ROM Based BSL	5K	N/A	UART	4	2	Need to follow TI's protocol and the setting with UART/I2C
			I2C	4	2	
ROM Based BSL with Plug-In interface	5K (just used the BSL Core section)	~ 1.6K	UART	4	2	Need to follow TI's protocol, for the interface level are totally open source.
		~ 1.3K	I2C	4	2	
		~ 1.6K	SPI	6	4	
		~ 5.8K	CAN	4	2	
Flash Based Secondary BSL	N/A	~ 4.9K	UART	4	2	Totally open source.
		~ 4.7K	I2C	4	2	
		~ 5K	SPI	6	4	
		~ 9K	CAN	4	2	

Note

Hardware invoke needs two more pins than software invoke, the pins are reset pin and GPIO invoke pin.

Table 1-2. MSPM0C Solutions Summary

BSL Solutions	Flash Cost	Interface	Pins Used With Hardware Invoke	Pins Used With Software Invoke	Using Case
Flash Based BSL	~ 3.8K	UART	4	2	Totally open source.
	~ 3.5K	I2C	4	2	

Note

For the flash cost is with limited features: Mass Erase , Get device Identity and Program. For other features can be enabled in the code.

Figure 1-2 shows the structure of BSL in MSPM0.

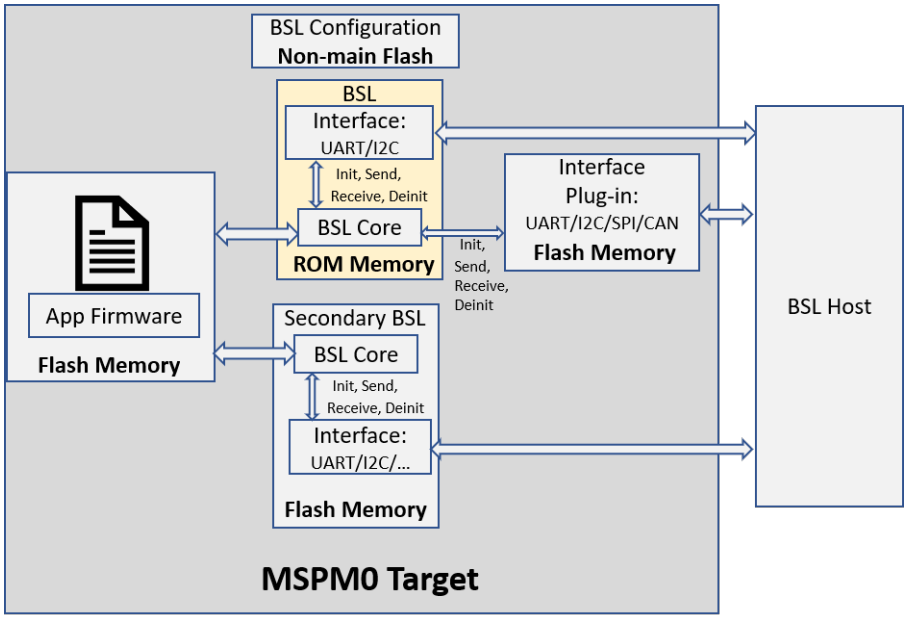

Figure 1-2. BSL Structure in MSPM0

1.1.2.1 ROM-Based BSL

MSPM0 L&G devices are shipped with a highly customizable ROM-based bootloader supporting UART and I2C.

The ROM-based BSL consist with BSL core and interface. The interface used to receive or send data packets between the host and target. The BSL core is used to interpret the packet data come from interface based on the protocol. Some of the parameters can be configured in non-main flash like BSL password or UART/I2C pins assignment.

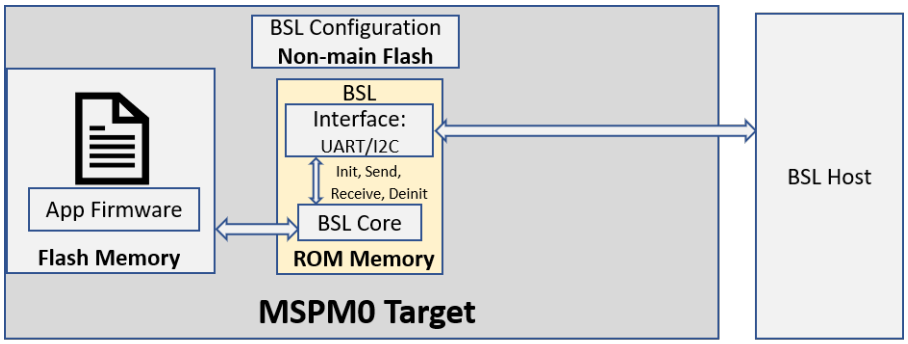

Figure 1-3. ROM-Based BSL Structure

1.1.2.2 ROM-Based BSL With Flash-Based Plug-In Interface

If the ROM-based communication interface (UART/I2C) cannot meet with the application, there are flash-based interface plug-in demos like UART, I2C,CAN and serial peripheral interface (SPI) that is fully open source can be modified as needed. The plug-in interface demos shared the ROM-based BSL core to interpret the packets that can save Flash memory to do that.

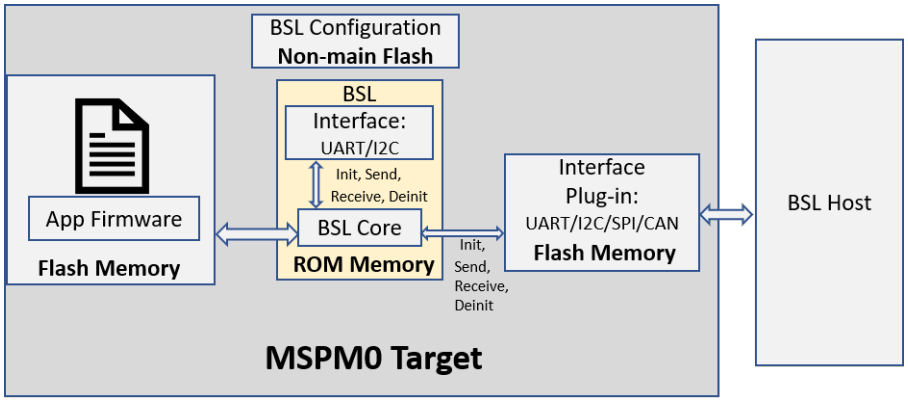

Figure 1-4. ROM-Based BSL With Flash-Based Plug-In Interface Structure

1.1.2.3 Flash-Based Secondary BSL

If the private protocol is needed, the ROM-based BSL core cannot be used any more and the secondary BSL demo can be referred. A totally open sourced secondary BSL demo is provided in the SDK that you can use to easily modify the protocol. The default protocol of the secondary BSL demo is the same with ROM-based BSL.

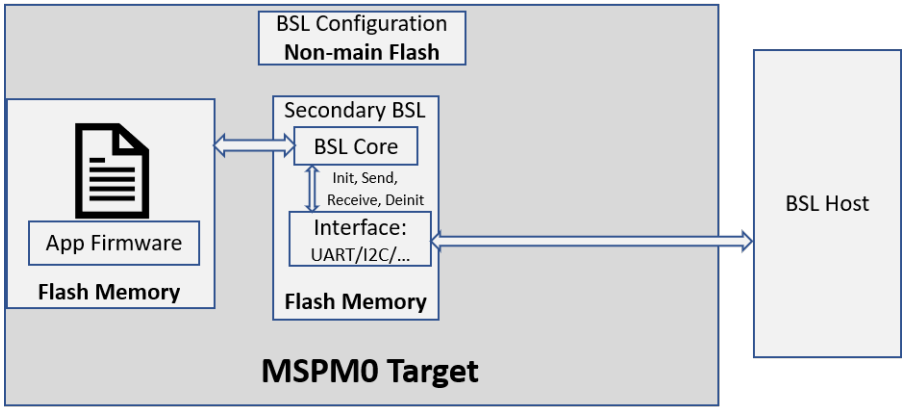

Figure 1-5. Flash-Based Secondary BSL Structure

There are also two kinds of secondary BSL demos mentioned, as shown in Figure 1-6. Use MSPM0G3507 as an example:

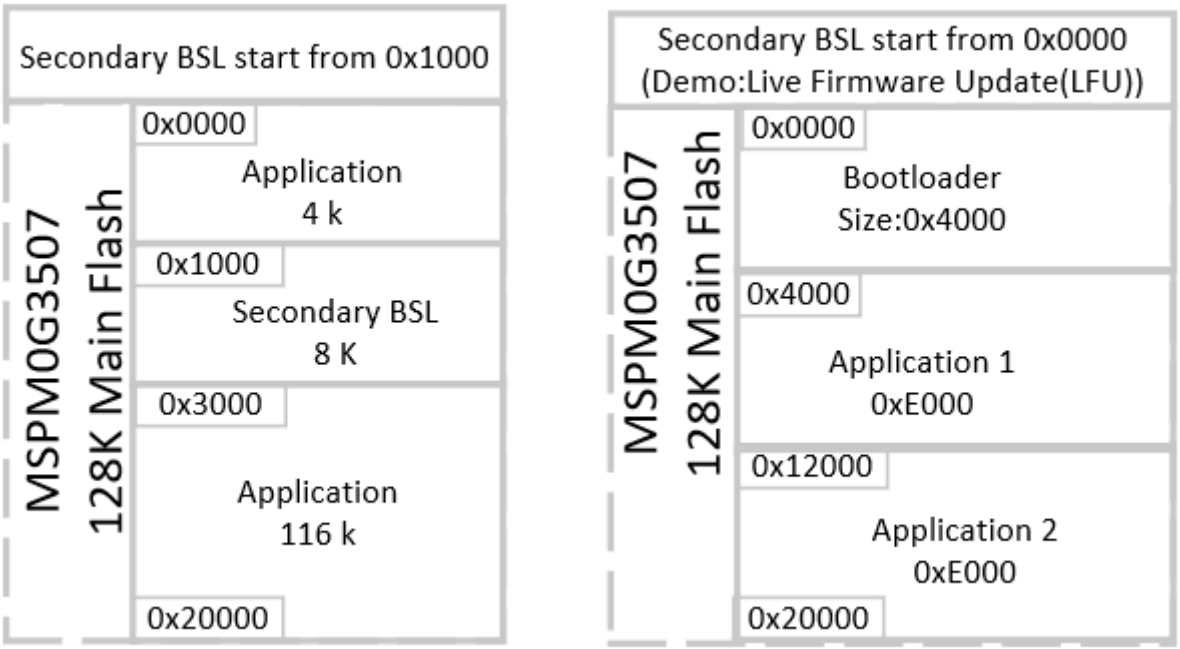

Figure 1-6. Secondary BSL Solutions

- For the secondary BSL start from 0x1000, you can put it anywhere in the flash except 0x0. Because the application code must start from the 0x0 address. In this condition when device power up or reseted, it checks the BSL invoke condition in the bootcode the decide run application code or BSL code.The demo reused the ROM-based BSL's trigger resources. (Both hardware, software and blank device detection, more information please refer to section 3.2 of *MSPM0 Bootloader User's Guide*). Figure 1-7 shows the secondary BSL demo executed flow.

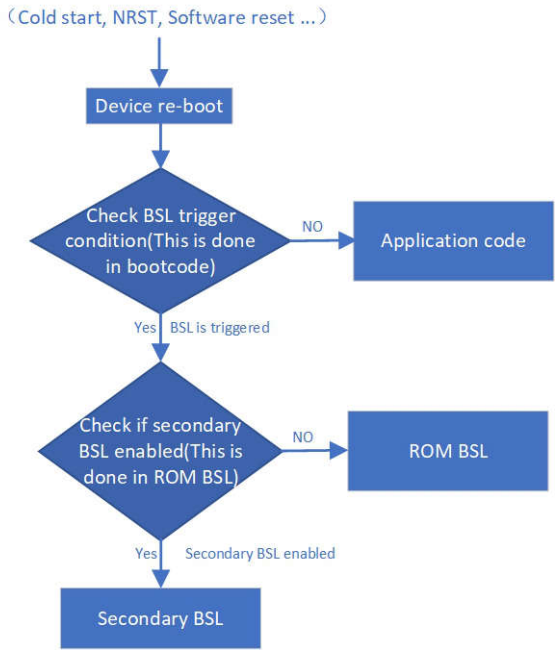

Figure 1-7. Secondary BSL Execute Flow

- For the secondary BSL that start from 0 address, the MCU runs into secondary BSL every time when power up or reset. In the secondary BSL use custom check judgment to decide if stay in BSL to do firmware update or go into application. For the advantage of this solution is that customer can used special judgment that not limited to GPIO, blank device detection. For example need to check CRC of application before jump to application code to make sure the integrity of the application code. The other using case is for some MSPM0 device without ROM BSL like MSPM0C and we do have demo code about it in the SDK. In this demo, after check the invoke condition, if need jump to application, it can set the PC to the start address of application. More information you can refer to Section 4.3.2. There is also a demo that using FreeRTOS in the secondary BSL that can realize live firmware update. That demo means the secondary BSL firmware update ongoing without stop application code. For more information, see *MSPM0 Live Firmware Update (LFU) Bootloader Implementation* .

1.1.3 MSPM0 BSL Features and Demos Summary

Table 1-3. MSPM0 BSL Features Summary

Device Family		MSPM0C	MSPM0L	MSPM0G
BSL General	BSL memory type	Flash	ROM	ROM
	BSL memory size	>3.5K	5K	5K
	User configuration in Non-main flash	✓	✓	✓
	UART	✓	✓	✓
	I2C	✓	✓	✓
Plug-In interface demos	UART		✓	✓
	I2C		✓	✓
	SPI		✓	✓
	CAN			✓
BSL Invoke	GPIO Invoke	✓	✓	✓
	Blank device detection	✓	✓	✓
	Software Invoke	✓	✓	✓
Hardware Tools	XDS110 with UART	✓	✓	✓
Software Tools	MSPM0_BSL_GUI in SDK		✓	✓
	Uniflash		✓	✓
Security	256 bits password protected	✓	✓	✓

There are some BSL code examples in the SDK and it can be summarized, as shown in Table 1-4.

Table 1-4. MSPM0 BSL Demos summary

	Demo Type	Demo Name	Using Case
Target side demos	**Plug-in interface demos**	bsl_spi_flash_interface	When ROM-based communication interface configuration or type not meet with requirement (need UART1 module as interface or need SPI) and TI's default BSL protocol can be used
		bsl_uart_flash_interface	
		bsl_i2c_flash_interface	
		bsl_can_flash_interface	
	Secondary BSL demo	secondary_bsl (uart/i2c/spi/can) flash_bsl(for MSPM0C)	When TI's default BSL protocol cannot meet with requirements, it re-used the same trigger condition of ROM-based BSL except the flash_bsl demo for MSPM0C.
	Application demo	bsl_software_invoke_app_demo (uart/i2c/spi/can)	Application example code can be co-work with ROM-based BSL or flash-based secondary BSL demo or flash-based interface plug-in demos, it also include software trigger function.
Host side demos	**MCU or processor as host**	bsl_host_mcu_to_m0x_target (uart/i2c/spi/can)	When MCU or processor as host and follow TI's default BSL protocol. It can be used with ROM BSL and default secondary BSL demos.
	PC as host	MSPM0_BSL_GUI/Uniflash	When PC as host with UART and follow TI's default BSL protocol. That means this can be used for ROM based UART BSL or default UART plug-in interface demo or default secondary BSL UART demo.

Table 1-5. MSPM0 BSL Demos Co-Work - MCU as host

Target side				Host side
	Memory location	**BSL code demo**	**Application code demo**	**MCU/Processor Host**
ROM BSL	ROM	/	bsl_software_invoke_app_demo (uart/i2c/spi/can)	bsl_host_mcu_to_m0x_target (uart/i2c/spi/can)
Plug-In interface demos	Main Flash (Need co-work with ROM BSL)	bsl_spi_flash_interface		
		bsl_uart_flash_interface		
		bsl_i2c_flash_interface		
		bsl_can_flash_interface		
Secondary BSL demo	Main Flash	secondary_bsl (uart/i2c/spi/can)		

Table 1-6. MSPM0 BSL Demos Co-Work - PC as host

Target side				Host side
	Memory location	**BSL code demo**	**Application code demo**	**PC Host**
ROM BSL	ROM	/	bsl_software_invoke_app_demo (uart/i2c/spi/can)	MSPM0_BSL_GUI/Uniflash
Plug-In interface demos	Main Flash (Need co-work with ROM BSL)	bsl_spi_flash_interface		N/A
		bsl_uart_flash_interface		MSPM0_BSL_GUI/Uniflash
		bsl_i2c_flash_interface		N/A
		bsl_can_flash_interface		N/A
Secondary BSL demo	Main Flash	secondary_bsl (uart/i2c/spi/can)		N/A

1.2 BSL Host Implementation Summary

This application note describes the implementation of two types of hosts: one is PC with an interface bridge like XDS110, the other is an MCU or processor. Figure 1-8 shows the signal connection diagram.

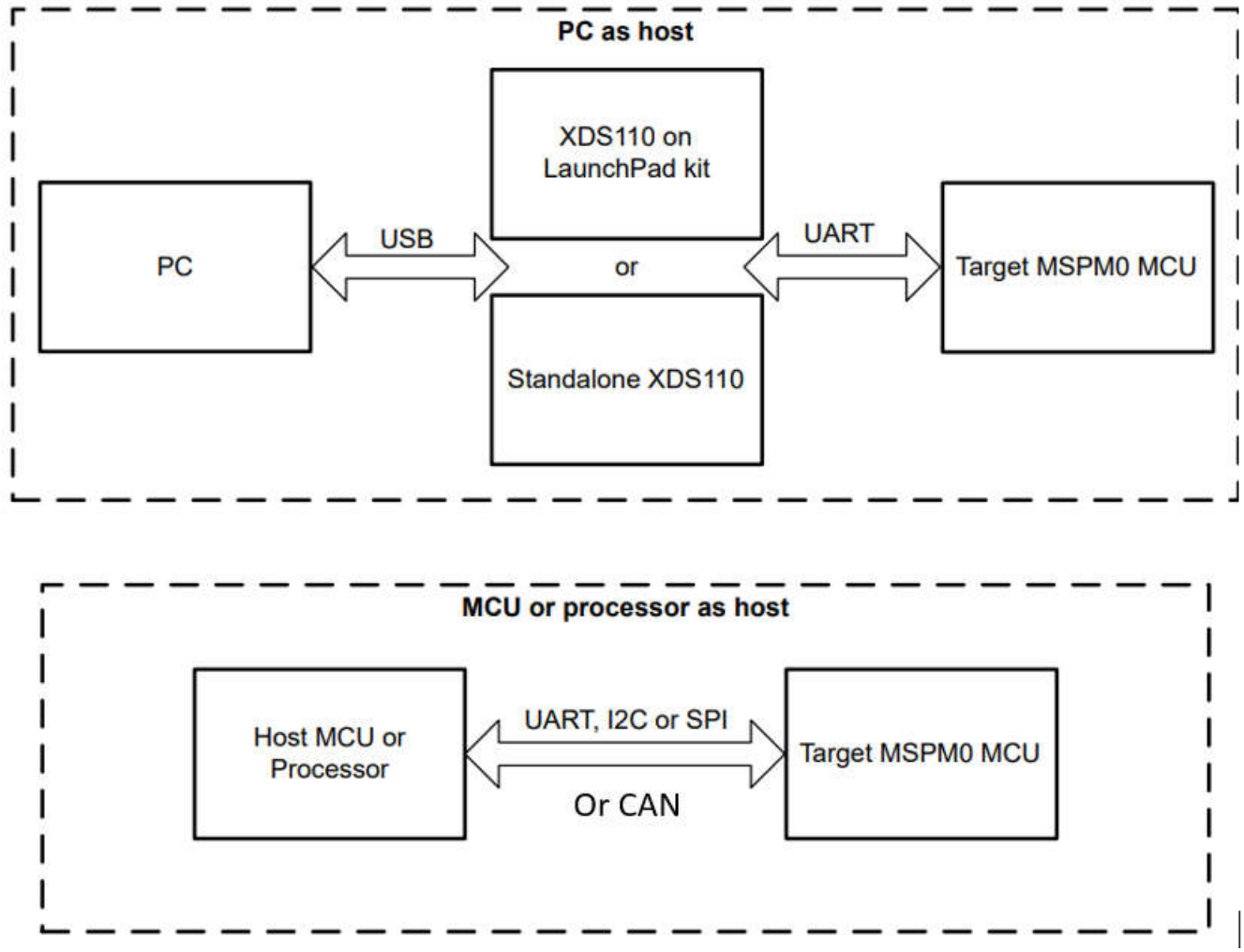

Figure 1-8. BSL Firmware Update System Block Diagram

When use PC as the host, there is a GUI that developed based on python 3 to handle the download operations. A pre-built Windows executable (tested on Win10 64-bits) is included, the source code of the GUI are also in the SDK. Uniflash can also be used on PC side.

When use MCU or processor as the host, there are some demos based on MSPM0 act as the host MCU to do firmware updated for another MSPM0 device.

2 BSL Configuration in Non-Main (Configuration NVM)

2.1 Non-Main Introduction

There are three different kind of Flash memories in MSPM0 devices.

Table 2-1. Flash Memory Regions

Flash Memory Region	Region Contents	Executable	Used by	Programmed by
FACTORY	Device ID and other parameters	No	Application	TI only(not modifiable)
NON-MAIN	Device boot configuration (BCR and BSL)	No	Boot ROM	TI, User
MAIN	Application code and data	Yes	Application	User

The NON-MAIN is a dedicated region of Flash memory that stores the configuration data used by the BCR and BSL to boot the device. The region is not used for any other purpose. The BCR and BSL both have configuration policies that can be left at their default values (as is typical during development and evaluation), or modified for specific purposes (as is typical during production programming) by altering the values programmed into the NON-MAIN flash region. Due to MSPM0C series do not have ROM based BSL, so there is no BSL related configuration part in MSPM0C series device's NON-MAIN.

Table 2-2. NON-MAIN Region Overview

NON-MAIN Section	Start Address	End Address
BCR Configuration	41C0.0000h	41C0.005Bh
BCR Configuration CRC	41C0.005Ch	41C0.005Fh
BSL Configuration	41C0.0100h	41C0.0153h
BSL Configuration CRC	41C0.0154h	41C0.0157h

The main BSL parameters can be configured in Table 2-3.

Table 2-3. NON-MAIN Flash BSL Configuration Main Parameters

Parameters Using Case	Parameters	Description
Common	BSLCONFIGID	BSL configuration ID
	BSLPW	256-bit BSL access password. (Optional for secondary BSL)
	BSLCONFIG0	BSL invoke pin configuration and memory read-out policy. (For memory read-out policy is optional for secondary BSL)
	BSLAPPVER	Address of the application version word.
	BSLCONFIG1	BSL security configuration.(Optional for secondary BSL)
	BSLCRC	CRC digest (CRC-32) of the BSL_CONFIG portion of the NON-MAIN memory.
ROM-Based BSL	BSLPINCFG0	BSL UART pin configuration
	BSLPINCFG1	BSL I2C pin configuration
ROM-Based BSL with Flash based Plug-in interface	BSLPLUGINCFG	Defines the presence and type of a BSL plug-in in MAIN Flash memory.
	BSLPLUGINHOOK	Function pointers for plug-in init, receive, transmit, and de-init functions
Flash-Based Secondary BSL	PATCHHOOKID	Alternate BSL configuration
	SBLADDRESS	Address of an alternate BSL.

For more details about the NON-MAIN flash, see the *MSPM0 L-Series 32-MHz Microcontrollers Technical Reference Manual* or *MSPM0 G-Series 80-MHz Microcontrollers Technical Reference Manual*

2.2 Example – Disable PA18 BSL Invoke Pin With Sysconfig

The NON-MAIN configuration can be done with Sysconfig. Here is an example how to disable PA18 BSL invoke function in NON-MAIN flash, for PA18 is used for BSL invoke by default settings in NON-MAIN. If the application do not use the PA18 as the BSL invoke, this pin must be pull down or disable its BSL invoke function in NON-MAIN to avoid the device to into BSL mode when power up or reset.

1. Open Sysconfig and add configuration NVM, it will show an error when you do this to remind you the risks to enable the NON-MAIN flash. Accept configuration risks at step 2 can remove the error.

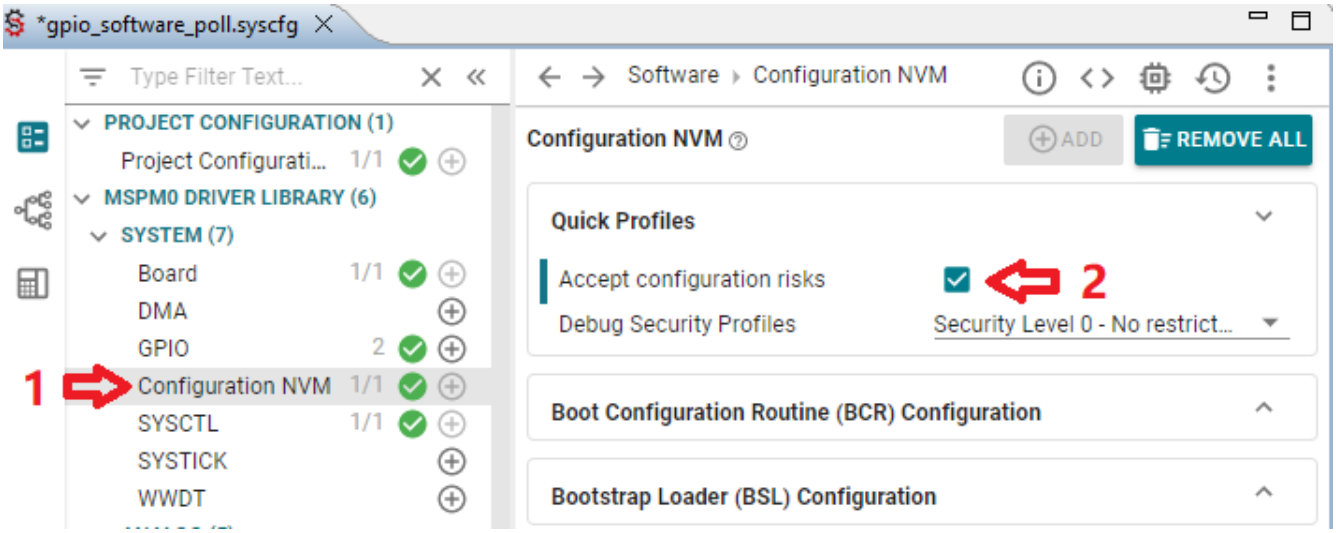

Figure 2-1. Disable PA18 BSL Invoke Pin Step One

2. Disable the PA18 BSL invoke function show in Figure 2-2 or choose another BSL invoke pin show in Figure 2-3.

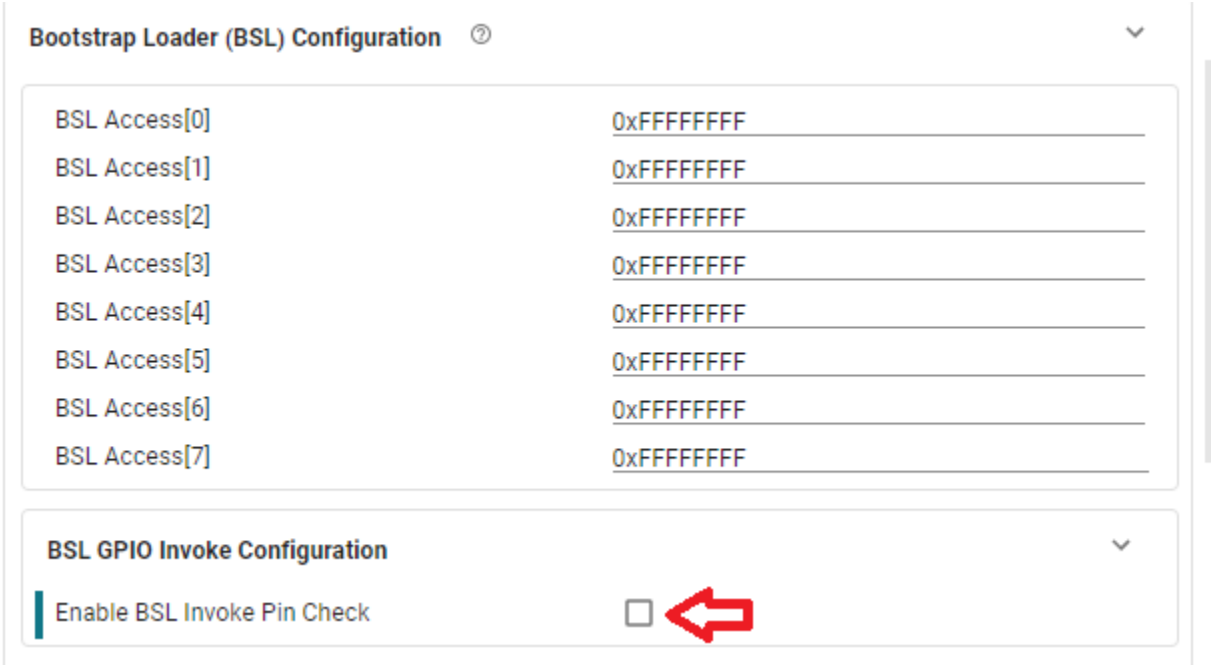

Figure 2-2. Disable PA18 BSL Invoke Function

Figure 2-3. Chose Other Pins as BSL Invoke

3. Build the project in Code Composer Studio™ (CCS) or IAR or Keil and then download the code into flash. The important thing to download the image is to enable NON-MAIN flash erase. For example in CCS to enable it in Figure 2-4.

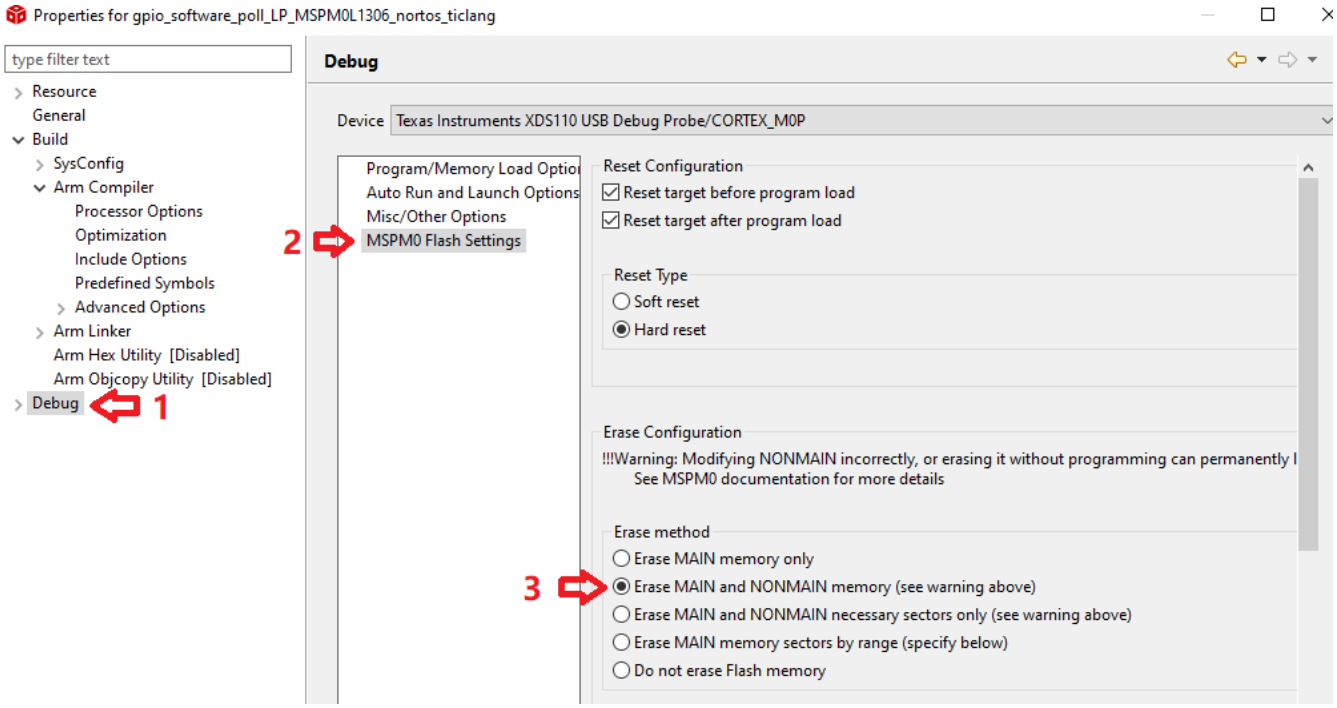

Figure 2-4. Enable NON-MAIN Flash Erase

3 Bootloader Host

3.1 MCU Host Code Introduction

The MCU host demos based on Code Composer Studio™ (CCS) are in the folder

< …\mspm0_sdk_xxxx\examples\nortos\LP_MSPM0xxxx\bsl >

These demos can update the target MSPM0 device through UART, I2C, SPI or CAN. The BSL host demo source code include a target device's firmware in application_image.h file that is converted from .txt image file by a GUI tool in SDK. For more details, see Section 3.1.2. It also includes the BSL password in the main.c file named BSL_PW_RESET array. The target side password is defined in the non-main flash, that is BSL configuration area BSLPW. Figure 3-1 shows a flow chart of the host BSL project.

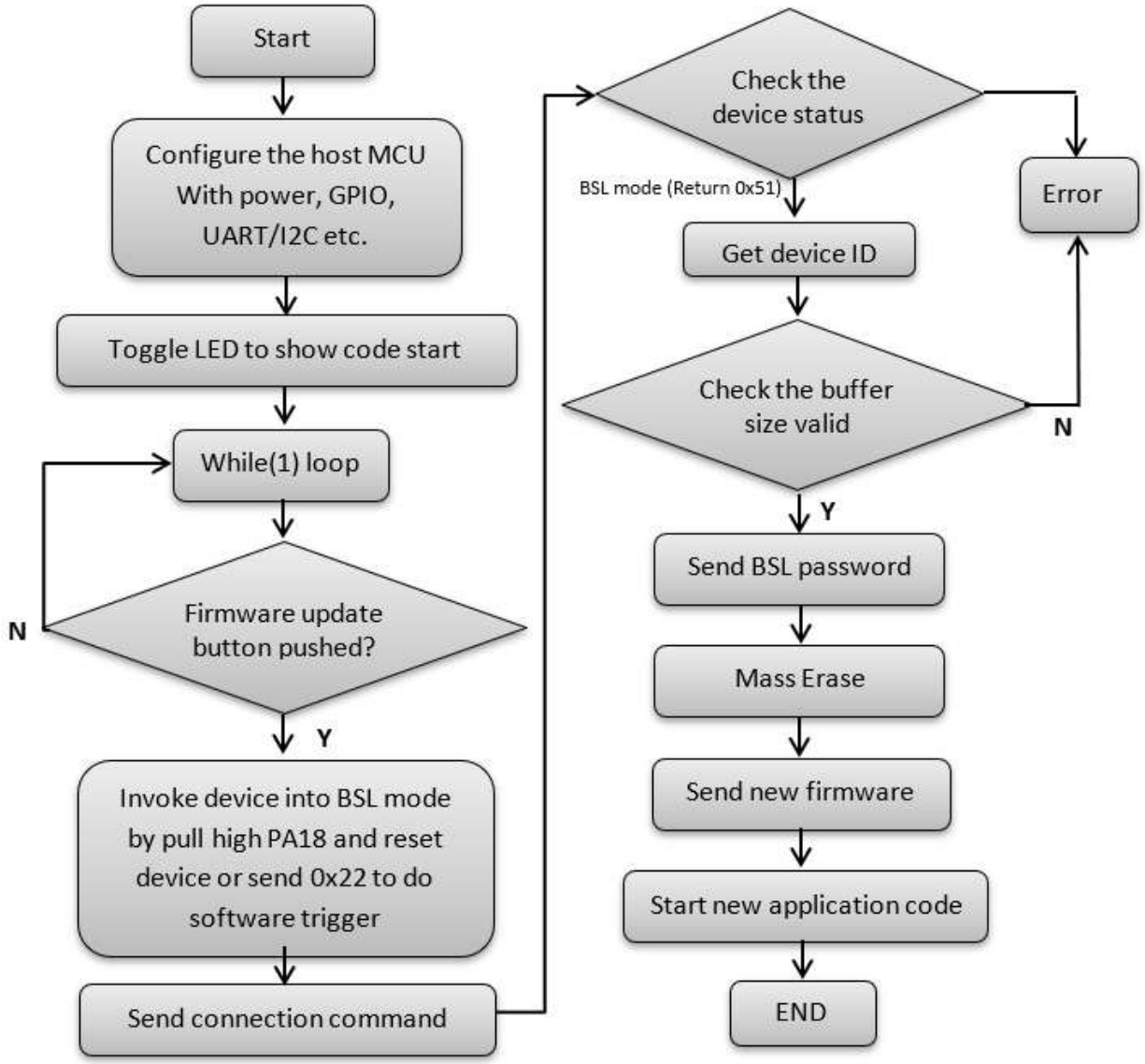

Figure 3-1. Flow Diagram of Host Project

The host demo can support hardware trigger that pull PA18 pin to high and then do a reset. Or the demo also can support software invoke that just need to send 0x22 command to trigger the BSL.

Note

When use software trigger, the application with software trigger function demo need to be downloaded first.

3.1.1 Hardware Connection

The host demo codes also use MSPM0 as the host MCU. The hardware signals connection between host and target is shown in Table 3-1

Table 3-1. Hardware Signal Connections

Signals	LP-MSPM0G3507		LP-MSPM0L1306	
	Host device	Target device	Host device	Target device
Reset	PB0	NRST pin	PA3	NRST pin
Invoke	PB16	PA18	PA7	PA18
UART	PB7/UART1_RX	PA10/UART0_TX	PA9/UART0_RX	PA23/UART0_TX
	PB6/UART1_TX	PA11/UART0_RX	PA8/UART0_TX	PA22/UART0_RX
I2C	PB2/I2C1_SCL	PA1/I2C0_SCL	PA11/I2C0_SCL	PA1/I2C0_SCL
	PB3/I2C1_SDA	PA0/I2C0_SDA	PA10/I2C0_SDA	PA0/I2C0_SDA
SPI	PB9/SPI1_SCLK	PB9/SPI1_SCLK	PA6/SPI0_SCLK	PA6/SPI0_SCLK
	PB8/SPI1_PICO	PB8/SPI1_PICO	PA5/SPI0_PICO	PA5/SPI0_PICO
	PB7/SPI1_POCI	PB7/SPI1_POCI	PA4/SPI0_POCI	PA4/SPI0_POCI
	PB6/SPI1_CS	PB6/SPI1_CS	PA8/SPI0_CS0	PA8/SPI0_CS
CANFD	PA12/CAN_TX	PA13/CAN_RX	\	\
	PA13/CAN_RX	PA12/CAN_TX	\	\

Note

Connect only one communication interface that UART or I2C or SPI. The target side pins are just default configuration pins that can be changed in the non-main flash.

Note

When use software invoke, the reset and invoke signals do not need to be connected.

Note

For CANFD, the transceiver are needed to connected with MSPM0 both host and target side.

Table 3-2. Hardware Signal Connections for MSPM0C

Signals	LP-MSPM0C1104	
	Host device	Target device
Reset	PA2	NRST pin
Invoke	PA4	PA18
UART	PA24/UART0_RX	PA27/UART0_TX
	PA27/UART0_TX	PA26/UART0_RX
I2C	PA11/I2C0_SCL	PA11/I2C0_SCL
	PA0/I2C0_SDA	PA0/I2C0_SDA

Note

When use software invoke, the reset and invoke signals do not need to be connected.

Note

When use UART interface, need to remove the jumpers connect to XDS110 UART back channel on J101.

3.1.2 TXT to Header File Conversion

The MCU host firmware contains an image of the target application as a header file(application_image.h). The header file is the new application firmware that need to programed into the MSPM0 target device. To get the header file, there is a conversion utility in the GUI MSPM0_BSL_GUI.exe in the following path:

< …\mspm0_sdk_xxxx\tools\bsl\BSL_GUI_EXE >.

1. Select TXT_to_H in the MoreOption menu.
2. Choose the TI-TXT format file to be converted. Few simple application demo files can be used that provided in the input folder.
3. Choose a folder for the output file (for example, choose the folder named Output).
4. Click the Convert button to start the conversion.

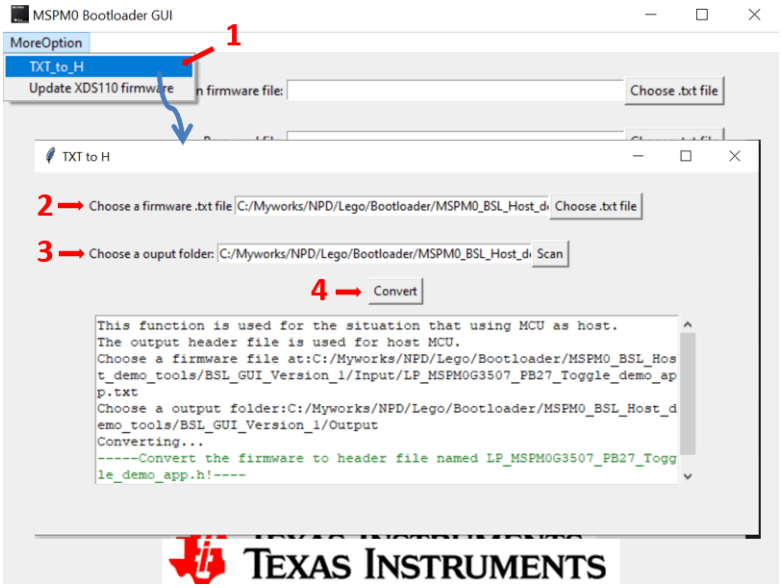

Figure 3-2. Steps to Convert TXT File to Header File

3.1.3 Step-by-Step Using the Demo

The following steps describe how to program an MSPM0 MCU using a LP-MSPM0G3507 as the host. A MSPM0G3507 is used as target device, hardware BSL invoke and UART communication are used in this demo. A similar process can be used to program other MSPM0 devices through either UART, I2C or SPI by using the proper hardware connections (seeTable 3-1).

1. Connect the hardware signals as shown in Figure 3-3. This example uses UART, so the I2C signals do not need to be connected.

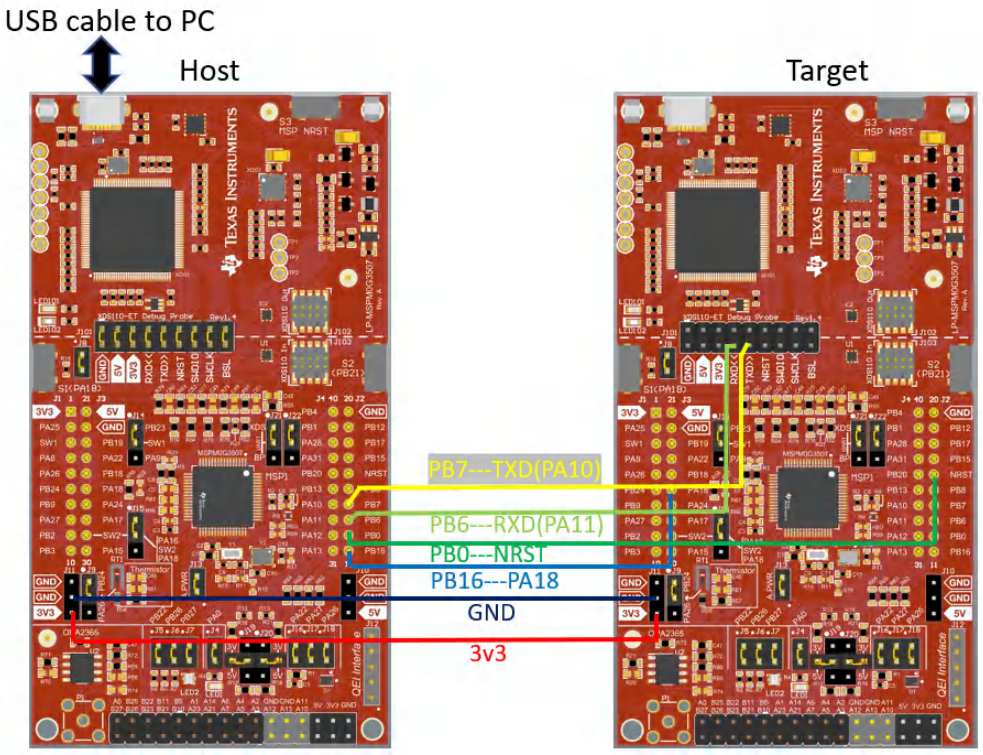

Figure 3-3. Hardware Signal Connections

2. Connect the jumpers as shown in Table 3-3.

Table 3-3. Jumper Connections

Board	Mode	Jumpers to Connect	Jumpers to Disconnect
LP-MSPM0G3507	Host	J101(Power and debug), J4,J7(LED)	None
LP-MSPM0G3507	Target	J7,(LED) J21, J22 (UART to J101 XDS110)	All in J101

Note

If use LP-MSPM0L1306 as target board, jumper on J6 must be removed.

3. Import the BSL host with UART demo that available in the folder < …\mspm0_sdk_xxxx\examples\nortos\LP_MSPM0G3507\bsl\bsl_host_mcu_to_mspm0g1x0x_g3x0x_target_uart> into CCS.

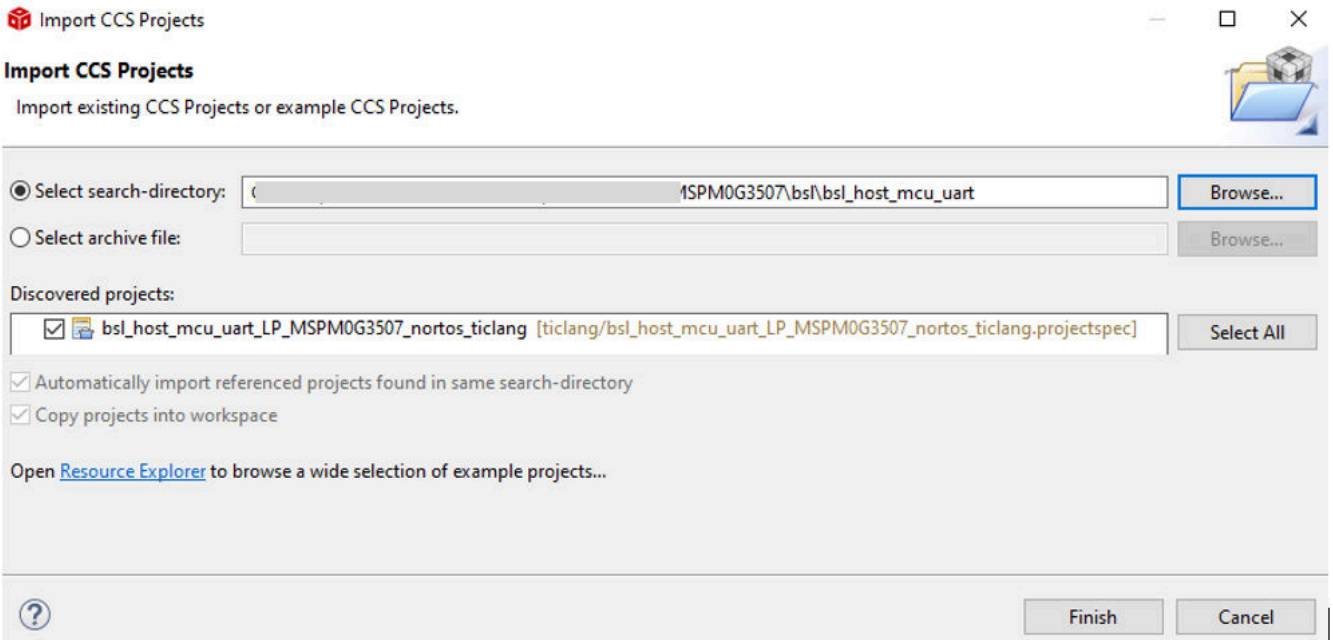

Figure 3-4. Import Host Project Into CCS

4. Modify the password in the bsl_password array in main.c, if needed. The default password are all 0xFF with 32 bytes. The target BSL password is defined in the NON-MAIN memory. For more information, see the technical reference manual [1] or [2] or the bootloader user's guide [3].
5. If just want to run the demo and no need to make any change of the application code, the BSL host demo include default firmware file application_image_uart.h that generate from demo named bsl_software_invoke_app_demo_uart) and the step 6 to 8 can be skip.
6. Import the application code(here can use the demo bsl_software_invoke_app_demo_uart) into CCS and generate the target device firmware in TI-TXT hex format (see Figure 3-5).

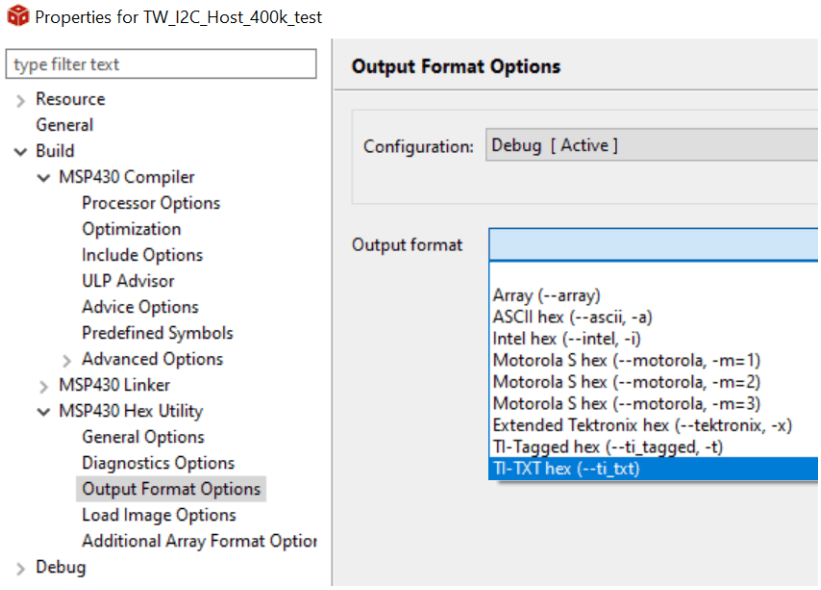

Figure 3-5. Generate TI-TXT Hex File in CCS

7. Run the GUI MSPM0_BSL_GUI.exe to convert the target device firmware .txt format to a header file. For more details, see Section 3.1.2.
8. Copy the contents of the output file xxx.h file by the GUI into the host project file application_image.h.
9. Build the host project and download to LP-MSPM0G3507.

10. Push button S2 on the host board to initiate the firmware update. If there is an error, LED1 turns on.

3.2 PC Host Example

The PC as host need a software GUI (MSPM0_BSL_GUI.exe or Uniflash) and a USB-to-UART bridge. Two hardware bridges are included (can be chosen in the MSPM0_BSL_GUI.exe): one is XDS110 on the MSPM0 LaunchPad kit and the other is a standalone XDS110. Both of the bridges support the backchannel UART that can be used as a USB-to-UART bridge. The XDS110 on the LaunchPad kit supports NRST pin and BSL invoke pin controlling that can be used by the GUI to start BSL mode on the MCU. For the standalone XDS110, two GPIO output pins (IOOUT0 and IOOUT1) in the AUX connection port can be used to control the NRST pin and BSL invoke pin on the target device and start BSL mode. (This is implemented with MSPM0_BSL_GUI.exe).

Note

Due to LP-MSPM0C1104 on boad XDS110 do not layout the BSL invoke pin, the GUI not support for it so far.

3.2.1 Prepare the Image File and Password File

Before downloading the firmware with the GUI, prepare two files: the application firmware file and the BSL password file.

The GUI (MSPM0_BSL_GUI.exe) supports only the TI-TXT format. For details on how to generate this format image file with CCS, see 6 in Section 3.1.3.

The format of the password file is similar to the TI-TXT format as shown in Figure 3-6. The BSL password is defined in the Non-Main memory. For more information, see the technical reference manual [1] [2] or the bootloader user's guide [3]. If the BSL password is not the default (all 0xFF), modify the password file. A default password file named BSL_Password32_Default.txt is available in this folder: < … \mspm0_sdk_xxxx\tools\bsl\BSL_GUI_EXE\Input >.

```
@00000000
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
q
```

Figure 3-6. BSL Default Password File (BSL_Password32_Default.txt)

3.2.2 Steps to Using the GUI

1. Connect the target device and the XDS110 to the PC. When using the XDS110 integrated in the LaunchPad kit, connect the micro USB cable to the PC as Figure 3-7.

   The ROM-based BSL UART pins for MSPM0G3507 are PA10 and PA11, and the pins are directly connected to the XDS110 backchannel UART, so all of the jumpers in J101 are required (see Table 3-5).

   On the LP-MSPM0L1306, the XDS110 backchannel UART pins are different from the BSL UART pins, so disconnect TXD and RXD in J101 and use jumper wires to connect PA22 and PA23 (see Table 3-5).

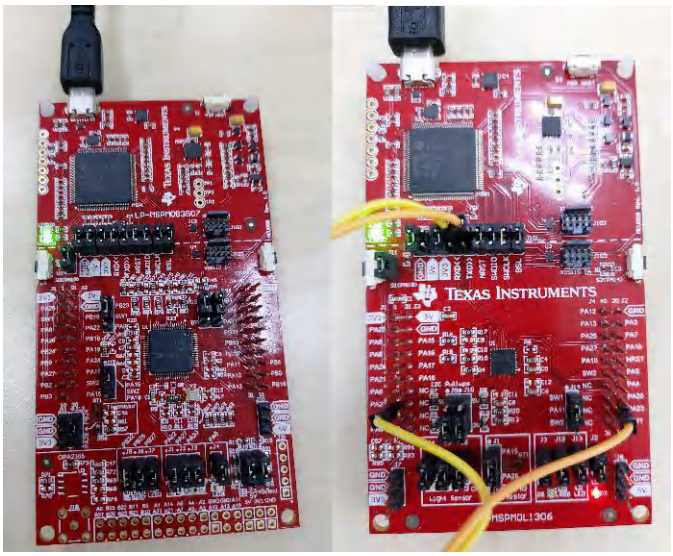

Figure 3-7. LaunchPad Kit Connection (Left: LP-MSPM0G3507, Right: LP-MSPM0L1306)

Table 3-4. Jumpers Connection

Boards	Mode	Jumpers Need Populated	Jumpers Need Unpopulated
LP-MSPM0G3507	Target	J101 (power, UART pins, Reset and BSL invoke pin), J4, J7(LED), J21, J22 (UART to J101 XDS)	NA
LP-MSPM0L1306	Target	J101 (GND, 3V3, NRST, BSL), J2, J3(LED)	J101 (TXD, RXD)

For standalone XDS110, the auxiliary interface (AUX) uses the signal connections in Table 3-5.

Table 3-5. Standalone Signal Connection

Signal	Standalone XDS110		Target Device		
	Signal	AUX Port	Signal	LP-MSPM0G3507	LP-MSPM0L1306
NRST	IO output	IOOUT0	NRST	NRST pin	NRST pin
Invoke	IO output	IOOUT1	Default: Invoke pin	PA18	PA18
UART	RXD	UARTRX	TXD	PA10/UART0_TX	PA23/UART0_TX
	TXD	UARTTX	RXD	PA11/UART0_RX	PA22/UART0_RX

2. Use the GUI to download the image to the target.
   a. Choose the TI-TXT format image file that need to be downloaded. (There are two demo images in the folder named input)
   b. Choose the TI-TXT format password file (a default file is in the *input* folder). For details on preparing this file, see Section 3.2.1.
   c. Choose hardware bridge.
   d. Click the download button.

The GUI automatically invokes the BSL so there is no need to manually invoke the BSL during this operation.

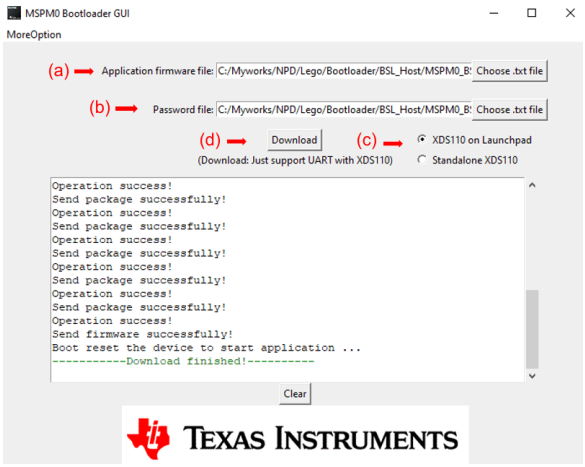

Figure 3-8. Steps to Download Image by GUI With UART

3. If using the XDS110, this GUI supports XDS110 firmware version firmware_3.0.0.20 or higher. If errors occur when download the image, update the XDS110 firmware.

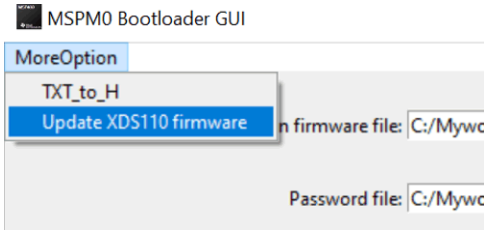

Figure 3-9. Update XDS110 Firmware

4 Bootloader Target

4.1 Default ROM-Based BSL

For some of the MSPM0 devices include the ROM-based BSL. The ROM-based BSL can just support UART and I2C interface. That cannot be changed but can be configured for some features in the NON-MAIN flash. For example, the UART/I2C pin assignment or the I2C address and so on. For more details, see the *MSPM0 Bootloader User's Guide*.

4.1.1 UART Interface

MSPM0 ROM-based BSL UART is enabled with following configuration:

- Baud rate: 9600bps (can be changed in NON-MAIN (just for some devices) or BSL core commands)
- Data width: 8 bit
- One stop bit
- No parity

The UART pins assignment and baud rate (just for some devices) can be configured in the NON-MAIN flash.

4.1.2 I2C Interface

MSPM0 ROM-based BSL I2C is enabled with following configuration:

- Address: 0x48 (can be changed in NON-MAIN)
- Address width: 7 bit

The I2C pins assignment and slave address can be configured in the NON-MAIN flash.

4.2 Flash-Based Plug-In Interface Demos

When the ROM-based BSL interface section settings can’s meet the application’s requirement. The flash-based plug-in interface demos can be used. Due to the code for the communication section are both open source, customer can change any settings in the code. Remember that the flash-based plug-in interface demos can only receive the BSL packets and not parse the packets. So, the plug-in interface demos need to co-work with BSL core located in the ROM BSL to parsing the commands.

4.2.1 UART Interface

MSPM0 flash-based UART is enabled with following configuration by default:

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

4.2.1.1 Step by Step Using the Demo

These are the steps on how to use the flash-based UART plug-in interface demo for MSPM0G3507:

1. Import the flash-based UART plug-in interface demo into CCS from the SDK.

   <…\mspm0_sdk_xxxx\examples\nortos\LP_ MSPM0G3507\bsl\ bsl_uart_flash_interface >
2. Make any needed changes and build the project.
3. Do a factory reset to clear any static flash write protection settings in NONMAIN. If the device is blank, this step can be skipped. For more information on the steps to perform this operation, see Section 5.2.
4. Download the UART plug-in code into flash. The important thing downloading the image is to enable NONMAIN flash erase shown in Figure 2-4. The plug-in interface demo cannot be debugged directly. For more details, see Section 4.2.1.2.
5. Prepare one LP-MSPM0G3507 launchpad and use the BSL host demos to do the firmware update. For more details, see Section 3.1.3 (MCU as host) or Section 3.2.2 (PC as host).

4.2.1.2 How to Debug the Plug-In Interface Code

When changing the plug-in interface demo code and doing a debug , here are some guide lines:

1. Make any changes needed and build the plug-in interface project.
2. Download it into the device with NON-MAIN erased as the same with step C in Section 2.2 and then do a power cycle.
3. Launch the device as shown below in Figure 4-1. Step 2 is to right click of the ccxml file.

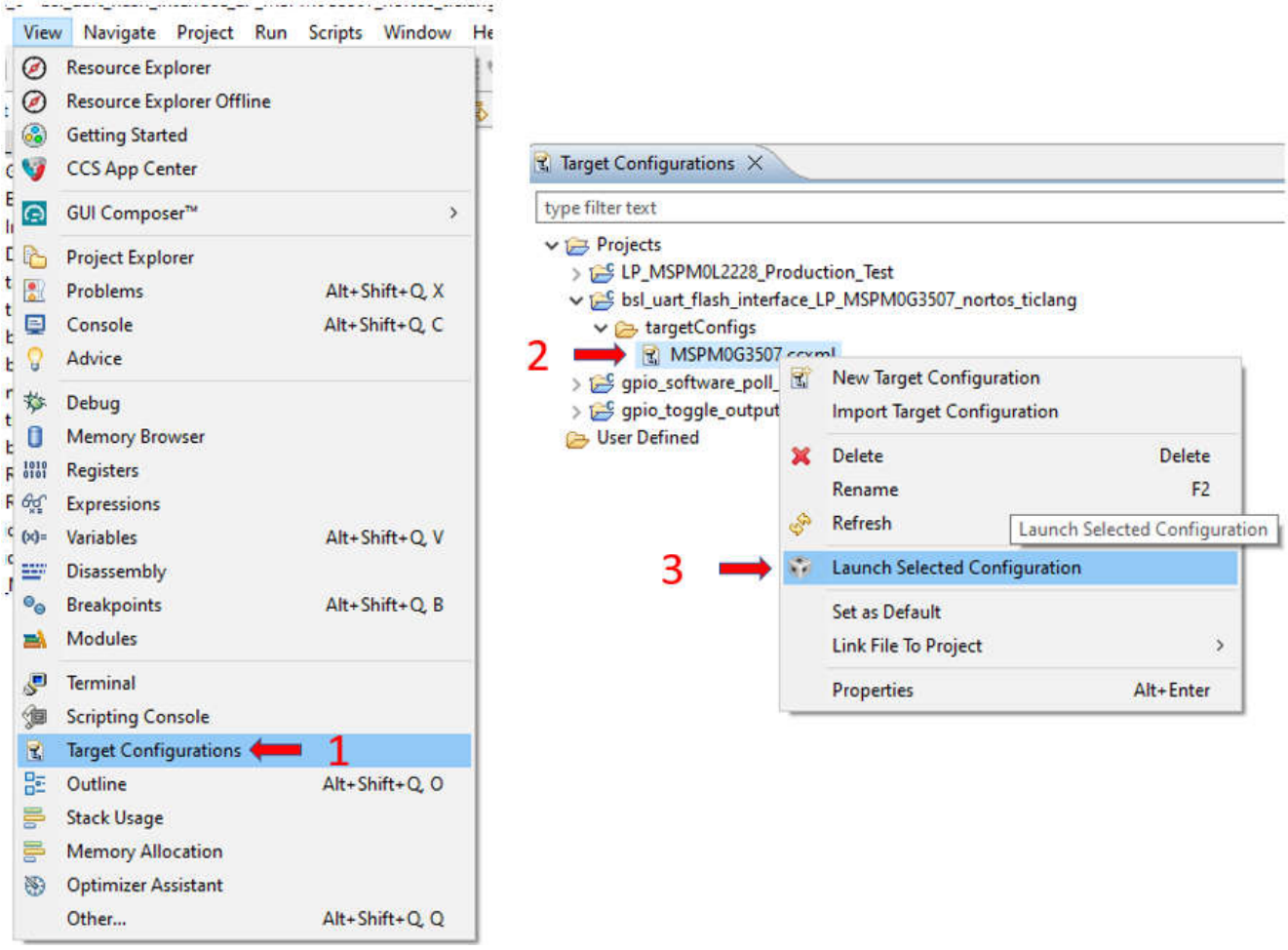

Figure 4-1. Launch the Device in CCS

4. Connect the target.

Figure 4-2. Connect the Device in CCS

5. Load symbols of the plug-in interface code and put the breakpoint needed.

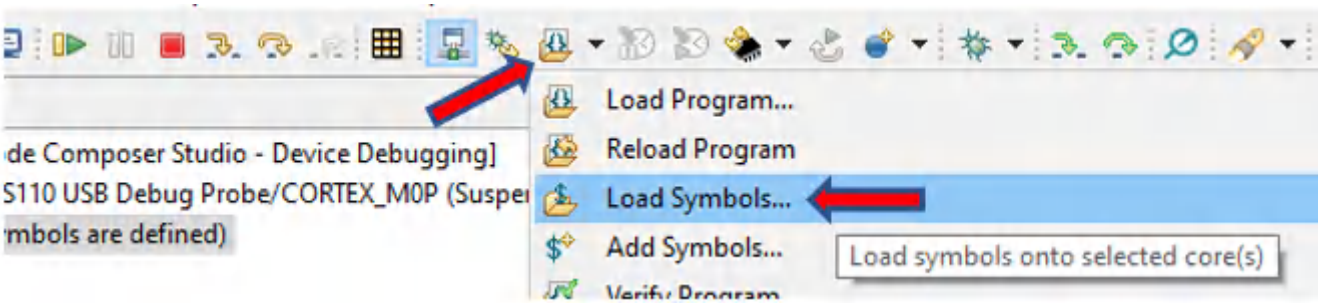

Figure 4-3. Load Symbols in CCS

6. Keep running the code to do the debug. The device go into BSL mode automatically due to the application area is empty.

4.2.2 I2C Interface

The I2C interface in the BSL acts as the I2C target or slave.

- I2C target address is 0x48 by default
- External pullup resistors are required for the SCL and SDA lines.

The demo's operation details is similar with UART plug-in interface. For the step by step operation and how to debug the demo, see Section 4.2.1.1 and Section 4.2.1.2

4.2.3 SPI Interface

The SPI plug-in demo is configured the SPI in target mode and other default settings as below:

- Motorola 4 wire connection
- Data captured on first clock edge
- Clock polarity low
- Bit order MSB

The demo's operation details is similar with UART plug-in interface. For the step by step operation and how to debug the demo, see Section 4.2.1.1 and Section 4.2.1.2

4.2.4 CAN Interface

The CAN plug-in demo is configured the CAN module as below by default:

- The Example is configured to work in CAN mode initially at 1Mbps.
- To change the bitrate of communication based on the configuration obtained from host through change baudrate command.

The data section in change baudrate command is expected to match the format shown in Figure 4-4.

Padding (5)	DRP (5)	DSJW (4)	DTSEG2 (4)	DTSEG1 (5)	NRP (9)	NSJW (7)	NSEG2 (7)	NTSEG1 (8)	BRS (1)	FD (1)

Figure 4-4. Data Section in Change Baudrate Command

- An arbitrary CAN frame is injected into CAN bus, on changing the CAN Mode to CAN FD to calibrate the transmission delay compensation attribute values. The Identity value can be modified as required.
- Message Identifier accepted by BSL Plug-in is 0x003
- Message Identifier sent from BSL Plug-in is 0x004

The demo's operation details is similar with UART plug-in interface. For the step by step operation and how to debug the demo, see Section 4.2.1.1 and Section 4.2.1.2

4.3 Secondary BSL Demo

If the private protocol is needed, the ROM-based BSL core cannot be used anymore or MSPM0C without ROM BSL, the secondary BSL demo can be referred. A totally open sourced secondary BSL demos is provided in the SDK that you can use to easily modify the protocol. The default protocol of the secondary BSL demo is the same with ROM-based BSL. There are some kinds of secondary BSL demos mentioned in Figure 1-6.

Table 4-1. MSPM0 Secondary BSL Demos Summary

Demos	Projects in SDK	Using Case
Seconday BSL start from 0x1000	<...\mspm0_sdk_xx\examples\nortos\LP_MSPM0L1306(or LP_MSPM0G3507)\bsl\secondary_bsl_uart/i2c/spi/can>	Just can be used with the device that can do BSL invoke detection in boot code(normally the device with ROM BSL) and need private protocol.
Flash based BSL start from 0x0 for MSPM0C	<...\mspm0_sdk_xx\examples\nortos\LP_MSPM0C1104\bsl\flash_bsl>	MSPM0 without ROM based BSL or need to change the judgment condition before jump to application like to do application's area CRC every time when power up or reset.
Live Firmware Update BSL	NA	Need live firmware update

In troditinal flash based BSL solution are most similar with the demo of Flash based BSL start from 0x0 for MSPM0C. For this kind of solution it will jump into the application code directly to set the PC with the start address of the application code. But it may cause stack conflict issue in some unexpected condition. For the solution of Seconday BSL start from 0x1000, it use reset to jump to applicaton code that will reset all registers or SRAM before go into applicaton code that will be more stable. So if the MSPM0 device with ROM based BSL and also need private protocol highly recommend to usde the demo of Seconday BSL start from 0x1000.

4.3.1 Flash-Based Secondary BSL Start From 0x1000

The secondary BSL start from 0x1000, it can be put anywhere in the flash area except start from 0x0. Because the application code must start from the 0x0. The secondary BSL demo executed flow showed in Figure 1-6. It can support UART or I2C or SPI or CAN interface if the device supported. The demo's step-by-step operation is the same as shown in Section 4.2.1.1.

When needed to debug the code after modification, follow the steps in Section 4.2.1.2.

In the secondary BSL, the interrupt vector table offset address has been moved start from 0x1000(due to the code start from 0x1000) in the reset handler that located in the file named startup_mspm0xxxx_ticlang.

When trying to move the secondary BSL to another flash area, this can be done in the cmd file. For example, move the secondary BSL start from 0x4000, make the modification of the cmd file as shown in Figure 4-5.

```
MEMORY
{
    FLASH_SBSL_INTVEC(RWX)  :   org = 0x00001000,   len = 0x00000040,
    FLASH_SBSL_START(RWX)   :   org = 0x00001040,   len = 0x00000040,
    FLASH_SBSL(RWX)         :   org = 0x00001080,   len = 0x00001F80,
```

```
MEMORY
{
    FLASH_SBSL_INTVEC(RWX)  :   org = 0x00004000,   len = 0x00000040,
    FLASH_SBSL_START(RWX)   :   org = 0x00004040,   len = 0x00000040,
    FLASH_SBSL(RWX)         :   org = 0x00004080,   len = 0x00001F80,
```

Figure 4-5. Move to 0x4000 cmd File Modification

The flash static write protection parameters and the start address of the alternate BSL also should be modified in the Sysconfig file, as shown in Figure 4-6.

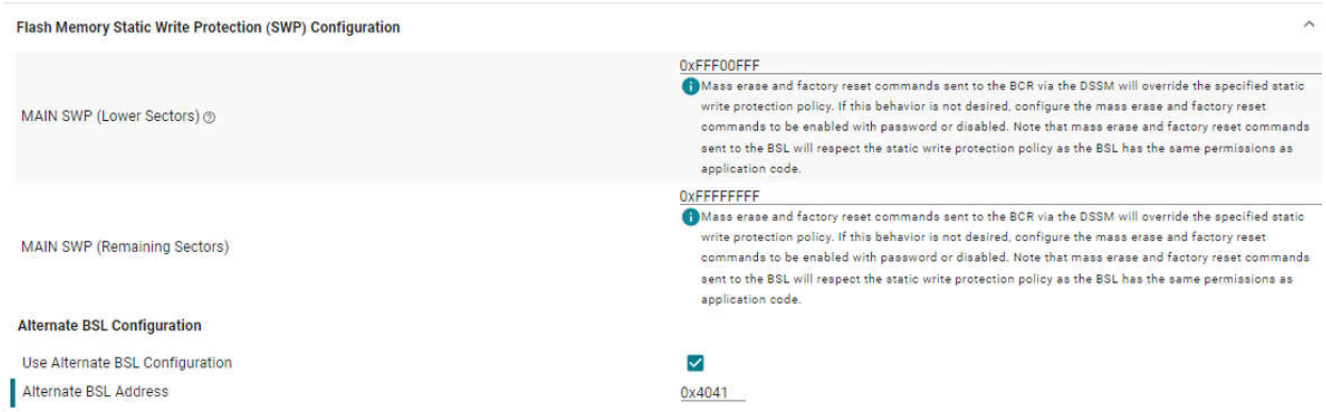

Figure 4-6. Move to 0x4000 Sysconfig File Modification

Except the modification in the secondary BSL, the application's cmd file also need to be modified that to avoid re-used the flash area that be used by secondary BSL.

4.3.2 Flash-Based Secondary BSL Start From 0x0000

For the secondary BSL that start from 0 address, the MCU will run into secondary BSL every time when power up or reset. In the secondary BSL use custom check judgement to decide if stay in BSL to do firmware update or go into application. For the advantage of this solution is that customer can used special judgement that not limited to GPIO, blank device detection. For example need to check CRC of application before jump to application code to make sure the integrity of the application code. The other using case is for some MSPM0 device without ROM BSL like MSPM0C and we do have demo code about it in the SDK.When jump to application can set the PC to the start address of application.

To use this kind of BSL, it is better to create two project one is for secondary BSL and the other is for the application. The flash area need to be seperated. So there will be two interrupt table that each project has one. It need to configurate the vector table offset registe(SCB->VTOR) to make the current interrupt table to be active when jump from BSL to applciaton code(Applicaton jump to BSL is using reset that will reset vector table offset registe automatically).

There is also a secondary BSL demos code that can support live firmware update. That demo means the secondary BSL firmware update ongoing without stop application code. For more information, see *MSPM0 Live Firmware Update (LFU) Bootloader Implementation* .

4.3.2.1 Flash-Based 0x0 Address BSL Demo for MSPM0C

Due to MSPMC device do not has the ROM based BSL, so it must to use flash based BSL and it must be start from 0x0 address that to run the invoke detection code every time when power up or reset.

In this demo, when device power up or reset, it will go into secondary BSL code first, in BSL reset handler it will detect the BSL invoke conditions (blank detectin, GPIO invoke or software invoke) to decide if need to stay in the BSL code to do firmware update or go into application code by set PC to application's start address. For this demo do not include application code CRC check, you can can refer to the *applicaton note of MSP430*.

Currently in this demo using the code below to set the PC of application's start address in reset handler ISR.

```
uint32_t *appResetHandler = (uint32_t *) (MAIN_APP_START_ADDR + VTOR_RESET_HANDLER_OFFSET);
appPointer FlashBSL_applicationStart = (appPointer) * (appResetHandler);
/* Before branch check if the address of reset handler is a valid Flash address */
if ((*((uint32_t *) MAIN_APP_RESET_VECTOR_ADDR) >= MAIN_APP_START_ADDR) &&
(*((uint32_t *) MAIN_APP_RESET_VECTOR_ADDR) < (MAIN_APP_START_ADDR + DEVICE_FLASH_SIZE))) {
FlashBSL_applicationStart(); }
```

There is another simple way to do the jump as below(APP_AREA_START_ADDR is the application's area start address that saved the interrupt vector table, shift 4 bytes to get the reset handler address)

```
/*! Jumps to application using its reset vector address */
#define TI_MSPBoot_APPMGR_JUMPTOAPP()     {((void (*)()) (*(uint32_t *)(APP_AREA_START_ADDR + 4)))
();}
```

If there are some jumping issue, please try the assembly code below that clear the RAM before jump to application's start address if the boot code and applicaton code are sharing some SRAM area.

```
 __asm(
#if defined(__GNUC__)
 ".syntax unified\n" /* Load SRAMFLASH register*/
#endif
 "ldr r4, = 0x41C40018\n" /* Load SRAMFLASH register*/
 "ldr r4, [r4]\n"
 "ldr r1, = 0x03FF0000\n" /* SRAMFLASH.SRAM_SZ mask */
 "ands r4, r1\n" /* Get SRAMFLASH.SRAM_SZ */
 "lsrs r4, r4, #6\n" /* SRAMFLASH.SRAM_SZ to kB */
#if defined ECC
 "ldr r1, = 0x20300000\n" /* Start of ECC-code */
 "adds r2, r4, r1\n" /* End of ECC-code */
 "movs r3, #0\n"
 "init_ecc_loop: \n" /* Loop to clear ECC-code */
 "str r3, [r1]\n"
 "adds r1, r1, #4\n"
 "cmp r1, r2\n"
 "blo init_ecc_loop\n"
#endif
 "ldr r1, = 0x20200000\n" /* Start of NON-ECC-data */
 "adds r2, r4, r1\n" /* End of NON-ECC-data */
 "movs r3, #0\n"
 "init_data_loop:\n" /* Loop to clear ECC-data */
 "str r3, [r1]\n"
 "adds r1, r1, #4\n"
 "cmp r1, r2\n"
 "blo init_data_loop\n"
 //Jump to Reset_Handler
 "ldr r0, = 0x7004\n" //FLASH_SBSL_INTVEC in .cmd file+ 4
 "ldr r0, [r0]\n"
 "blx r0\n"
 );
```

Add define of ECC if the device support ECC SRAM. For this demo the application start address is saved at address 0x7004, change it based on your application start address.

If you put the jump after peripheral initilization(not call the jump function before execute main() function), few points here need to take care:

- Do not jump in the ISR(execpt reset handler ISR) .
- Disable global interrupt(can use this function __disable_irq; and need to enable it in application's code by call __enable_irq;) → reset all peripherals that been used → clear all pending NVIC IRQs(call this API NVIC_ClearPendingIRQ(IRQn_Type IRQn)) → clear the RAM if needed → jump to application code start address.

4.3.2.2 Live Firmware Update (LFU) Solution

Live firmware update is used to run the application code during the firmware update. It uses FreeRTOS to handle the firmware update and the application code both running at the same time. For more information, see *MSPM0 Live Firmware Update (LFU) Bootloader Implementation*.

5 Common Questions

5.1 Linker File Modification

The demo provided currently are based on the CCS and most of the demos need to modify the linker files to do the memory arrangement. CCS is used in the cmd file to handle this work. For more information about the introduction of the cmd linker file, see this web page *TI Linker Command File Primer*.

5.2 Factory Reset by CCS to Recover Device

If the device cannot be accessed, try to do a factory reset of CCS to recover the device. The steps are as shown below:

1. Hardware connection: XDS110 with MSPM0 device.

   Signals needed: GND, SWDIO, SWCLK, NRST
2. Open Target Configurations.

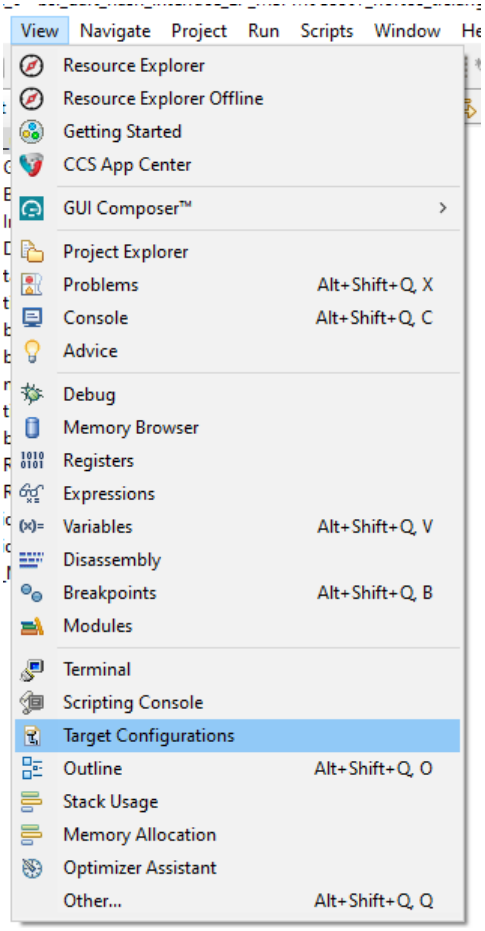

Figure 5-1. Open Target Configurations

3. In the Target Configurations window, find current MSPM0 project and expand the folders to find the .ccxml file:

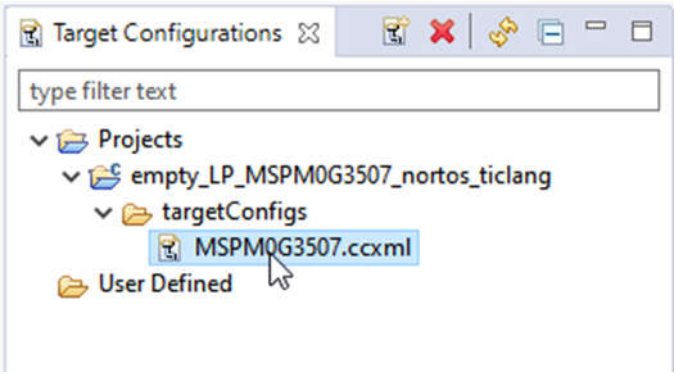

Figure 5-2. Find the ccxml File

4. Right-click the .ccxml file and click on Launch Selected Configuration.

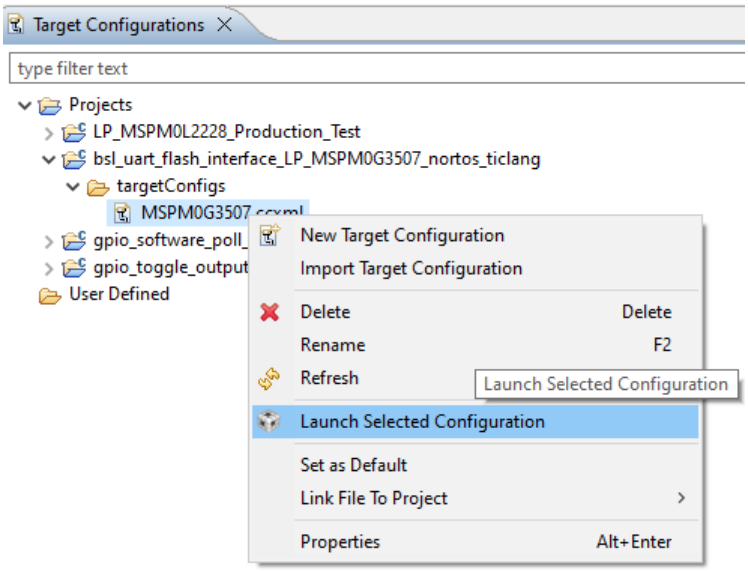

Figure 5-3. Launch Selected Configuration

5. Click on Scripts→ MSPM0G3507 Commands → MSPM0_Mailbox_FactoryReset_Auto.

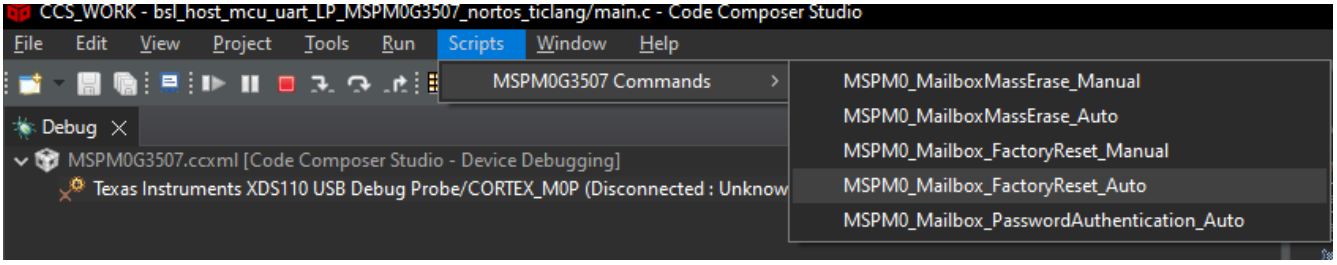

Figure 5-4. Do Factory Reset With the Script

6. The Console will show the following:

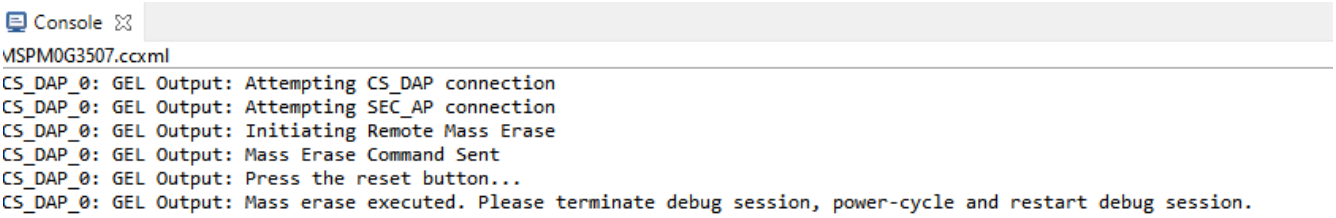

Figure 5-5. Log Information in Console

7. If that is not working, try force the device go into BSL and do the steps b to e above. To force the device go into BSL mode, if you have not modify the default BSL invoke pin that is PA18 in Non-main flash, you can pull high of PA18 before power up the device and keep it high. If you use the Launchpad you can just keep push the button connect with PA18 when connect the board to PC.

6 References

1. Texas Instruments: *MSPM0 G-Series 80-MHz Microcontrollers Technical Reference Manual*
2. Texas Instruments: *MSPM0 L-Series 32-MHz Microcontrollers Technical Reference Manual*
3. Texas Instruments: *MSPM0 Bootloader User's Guide*

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision B (March 2024) to Revision C (September 2024)	Page
• Updated Section 1.1.2	4
• Updated Section 1.1	6
• Updated Section 1.1.3	8
• Updated Section 1.2	15
• Updated Section 3	24

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265