User's Guide

MSPM0 NONMAIN Flash Memory Configuration Guide

Sal Ye, Wei Gao

Abstract

MSPM0 NONMAIN is a dedicated flash memory region which configures chip start-up related parameters and extended function selection. This document introduces the detailed configuration of NONMAIN, and provides guidance when configuring NONMAIN section for use in applications.

Table of Contents

Trademarks

1 Introduction

Figure 1-1 shows the common platform memory map that all MSPM0 devices share. The map has four different memory regions.

- MAIN Flash
- NONMAIN Flash
- FACTORY Region
- ROM

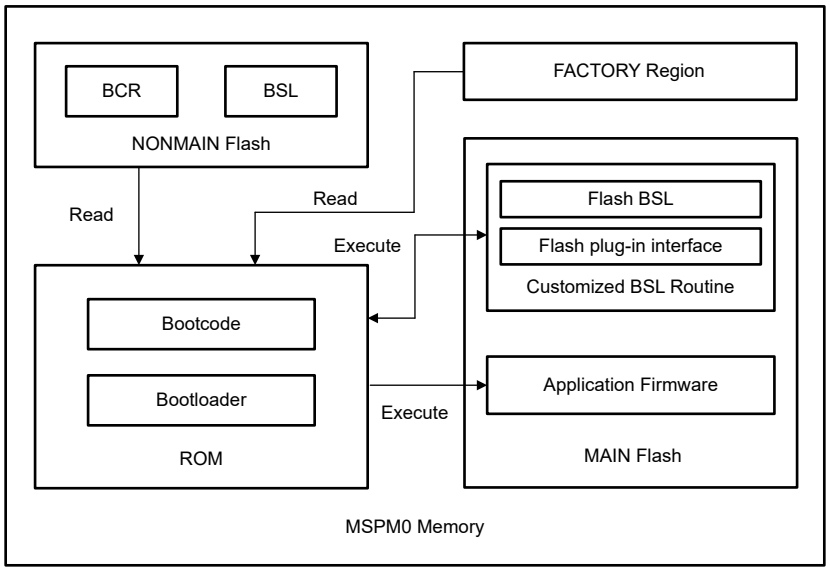


Figure 1-1. MSPM0 Memory Map

MAIN flash is the memory used to store executable code and data. The MSPM0 device supports single bank or dual bank MAIN flash, see the device-specific data sheet for more details.

NONMAIN flash is a dedicated region of flash memory which stores the configuration data used by the boot configuration routine (BCR) configuration and bootstrap loader (BSL) configuration to boot the device. The BCR and BSL have configuration policies which can remain at the 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 NONMAIN flash region.

Note

After erasing the NONMAIN flash, verify that the proper value loads into the NONMAIN flash. Failure to verify the load results in permanently locking the device after the next BOOTRST.

FACTORY region is a memory-mapped flash region which provides read-only data describing the capabilities of a device, as well as any factory-provided trim information for use by application software.

The read-only memory (ROM) consists of an immutable root-of-trust boot configuration routine (BCR) and bootstrap loader (BSL). The BCR is always the first code to run on the Cortex-M0+ processor following a BOOTRST of the device before the main application starts. The BCR also runs using hardware or software invocation of the bootstrap loader (BSL) and authorizes the BSL entry. Figure 1-2 shows the high level boot flow for MSPM0 devices. For more details, refer to the *Cybersecurity Enablers in MSPM0 MCUs* application note.

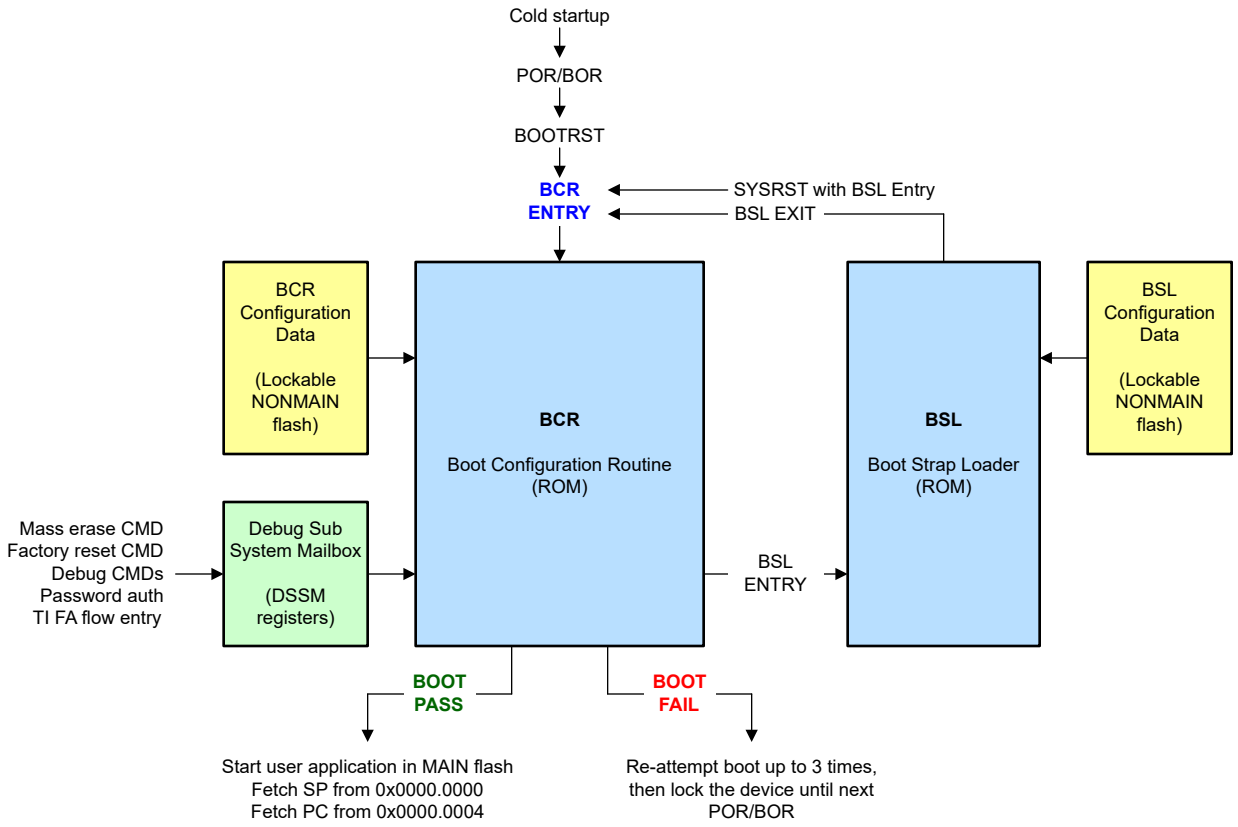


Figure 1-2. MSPM0 Boot Flow

The BCR sets up the device security policies, configures the device for operation, and optionally starts the BSL if ROM or Flash BSL presents. If the BCR starts the BSL, use the BSL to program or verify the device memory (flash and SRAM) using a standard serial interface (UART or I2C).

1.1 Terminology

Bootcode (BCR)	Start-up routine that runs after BOOTRST, configuring the device to execute an application.
BCR configuration	Configuration structure that contains all user-configurable parameters for Bootcode, which resides in NONMAIN flash memory.
Bootstrap loader (BSL)	Boot routine used to load data to the device memory.
BSL configuration	Configuration structure that contains all user configurable parameters for Bootstrap loader, which resides in NONMAIN flash memory.
Customer Secure Code (CSC)	The code runs after TI boot code and executes from flash memory to further configure security elements. For more details, see the Secure Booting User's Guide

2 NONMAIN Architecture

This section provides a detailed introduction to the NONMAIN structure, supported features, and applications of various NONMAIN configurations.

2.1 MSPM0 Family Overview

MSPM0 family supports different types of NONMAIN layout that limit the available features supported by the NONMAIN configuration.

Table 2-1 shows which NONMAIN layout type to use with which devices. For more details, see the *Boot Configuration* section of the device-specific technical reference manual, linked in Section 10.

Table 2-1. NONMAIN Layout Types

NONMAIN Layout Type	Supported Devices	Overview
Type A	MSPM0L110x MSPM0L130x MSPM0L134x MSPM0Gx10x MSPM0Gx50x	• No CSC support • Passwords stored in a plain text format • Only CRC32 support for application integrity checks • Supports ROM and Flash BSL(1)
Type B	MSPM0C1103 MSPM0C1104 MSPS003Fx	• No CSC support • No application integrity check • No passwords-based configuration • No BSL support
Type C	MSPM0Lx22x	• Includes CSC support • Passwords are stored in a SHA256 hashed format • Application integrity check has CRC32 or SHA256 hash option • Supports ROM and Flash BSL
Type D	MSPM0C1105 MSPM0C1106 MSPM0H321x	• Includes CSC support • Passwords are stored in SHA256 hashed format • Application integrity check supports CRC32 or SHA256 • Supports Flash BSL
Type E	MSPM0L111x MSPM0G511x MSPM0G5187	• Includes CSC support • Passwords stored in SHA256 hashed format • Application integrity check has CRC32 or SHA256 hash option • UART default Baud rate configuration • Disable NRST in BSL • Support ROM and Flash BSL
Type F	MSPM0Gx51x MSPM0G352x	• Includes CSC support • Passwords storage in a SHA256 hashed format • Application integrity check uses CRC32 or SHA256 hash options • UART default Baud rate configuration • Supports ROM and Flash BSL

(1) Flash BSL is the alternative BSL interface that the user defines in MAIN flash memory.

2.2 NONMAIN Configuration Overview

MSPM0 NONMAIN flash memory has different types of structure layouts. In certain layout types, some features are reduced or not supported.

Table 2-2. NONMAIN Configuration Parameters

Configuration Parameters			Register Field	NONMAIN Layout Type					
				A	B	C	D	E	F
BCR	BCR Configuration ID		BCRCONFIGID	✓	✓	✓	✓	✓	✓
	Serial Wire Debug (SWD) Policy Configuration	Access Policy	BOOTCFG0	✓	✓	✓	✓	✓	✓
		Debug Policy							
		Debug Password	PWDDEBUGLOCK	✓		✓	✓	✓	✓
		TI Failure Analysis	BOOTCFG1	✓		✓	✓	✓	✓
	SWD Factory Reset Configuration	Factory Reset Access Property	BOOTCFG3	✓	✓	✓	✓	✓	✓
		Factory Reset Password	PWDFACTORYRESET	✓		✓	✓	✓	✓
	SWD Mass Erase Configuration	Mass Erase Access Property	BOOTCFG3	✓		✓	✓	✓	✓
		Mass Erase Password	PWDMASSERASE	✓		✓	✓	✓	✓
	Flash Memory Static Write Protection (SWP) Configuration	MAIN Static Write Protection	FLASHSWP0 FLASHSWP1 FLASHSWP2	✓	✓	✓	✓	✓	✓
		NONMAIN Static Write Protection	BOOTCFG4						
	Customer Secure Code (CSC)	CSC Policy	BOOTCFG5			✓	✓	✓	✓
		Flash Bank Swap Policy				✓		✓	✓
		Debug Hold	BOOTCFG4			✓	✓	✓	✓
	Fast Boot Mode		BOOTCFG2	✓		✓	✓	✓	✓
	Application Digest Check Configuration	Application Digest Check Policy	BOOTCFG6	✓		✓	✓	✓	✓
		Application Start Address	APPDIGESTSTART						
		Length of the Application	APPDIGESTLENGTH						
		Application Checksum	APPDIGEST						
	BSL Policy	Enable Invoke Pin Check	BOOTCFG1	✓		✓	✓	✓	✓
		Enable BSL Mode	BOOTCFG2						
	BCR Checksum	CRC16-CCITT	BCRCRC				✓		
		CRC32-ISO3309		✓		✓		✓	✓

Table 2-2. NONMAIN Configuration Parameters (continued)

Configuration Parameters			Register Field	NONMAIN Layout Type					
				A	B	C	D	E	F
BSL	BSL Configuration ID		BSLCONFIGID	✓		✓	✓	✓	✓
	BSL Pin Configuration	Invoke Pin Configuration	BSLCONFIG0	✓		✓	✓	✓	✓
		NRST Pin Configuration	BSLCONFIG3						✓
	UART Interface Configuration	Pin Configuration	BSLPINCFG0	✓		✓		✓	✓
		Baud Rate Configuration	BSLCONFIG1					✓	✓
	I2C Interface Configuration	Pin Configuration	BSLPINCFG1	✓		✓		✓	✓
		Target Address Configuration	BSLCONFIG2						
	Plug-in Interface Configuration	Enable Plug-in Interface	BSLPLUGINCFG	✓		✓		✓	✓
		Function Pointer Address Assignment	BSLPLUGINHOOK						
	Alternate BSL Configuration	Enable Alternate BSL Interface	BSLCONFIG1	✓		✓	✓	✓	✓
		Alternate BSL Interface Address Assignment	SBLADDRESS						
	BSL Security Configuration	BSL Access Password	PWDBSL	✓		✓		✓	✓
		BSL Read Out Feature	BSLCONFIG0						
		BSL Alert Configuration	BSLCONFIG1						
		App Integrity Check	BSLAPPVER						
	BSL Checksum	CRC16-CCITT	BSLCRC				✓		
		CRC32-ISO3309		✓		✓		✓	✓

Note

For different types of NONMAIN layout, the register address offset can be different. See the device-specific technical reference manual for details, linked in Section 10.

2.3 NONMAIN Memory

Table 2-3 shows the overall memory sections for NONMAIN flash. The register memory map is different per the NONMAIN layout type. Refer to the device-specific technical reference manual for further details, linked in Section 10.

Table 2-3. NONMAIN Memory Map

NONMAIN Section	Start Address	End Address
BCR Configuration	41C0.0000h	41C0.00FFh
BSL Configuration	41C0.0100h	41C0.01FFh
Reserved Memory	41C0.0200h	41C0.03FFh

Some devices provision a second NONMAIN sector (1KB) that is accessed with the following restrictions:

- It can never be erased (factory reset, mass erase operations included)
- It can be written only until CSC calls INITDONE, and subsequently it can only be read.

This protection serves to implement hardware-based monotone counter feature. Because the sector can only be programmed, it effectively works as a nonvolatile up-counter (or down-counter depending on how the counter value is interpreted by software). The CSC can maintain revision or roll-back protection information in this sector, assuring that a version lower than the counter value cannot be activated.

Refer to the device-specific data sheet to determine if the device supports a second NONMAIN sector.

3 NONMAIN Configuration

This section provides a detailed introduction to the NONMAIN configuration, including BCR configuration and BSL configuration.

3.1 BCR Configuration

The boot configuration routine (BCR) is the first firmware to run on the device after a BOOTRST. Users can set different properties for device at the start-up stage of boot.

TI recommends using the Sysconfig tool to customize BCR configuration, see BCR Configuration With SysConfig for more details.

3.1.1 BCR Configuration ID

TI defines a default BCR configuration ID for different types of NONMAIN layout. Users can overwrite the configuration ID for application usage. Perform a factory reset to recover the default BCR configuration ID.

Table 3-1 shows the default values of each device.

Table 3-1. Default BCR Configuration ID

NONMAIN Layout Type	Device Family	Default Value
Type A	MSPM0L110x	0x00000001
	MSPM0L130x	
	MSPM0L134x	
	MSPM0Gx10x	
	MSPM0Gx50x	
Type B	MSPM0C1103	0x00000003
	MSPM0C1104	
	MSPS003Fx	
Type C	MSPM0Lx22x	0x01000002
Type D	MSPM0C1105	0x05000000
	MSPM0C1106	
	MSPM0H321x	
Type E	MSPM0L111x	0x06000000
	MSPM0G511x	
	MSPM0G5187	
Type F	MSPM0Gx51x	0x04000000
	MSPM0G352x	

3.1.2 Serial Wire Debug (SWD) Policy

The serial wire debug (SWD) related policies configure the available functions in the physical debug interface of the device. MSPM0 devices support three generic security levels: no restrictions (Level 0), custom restrictions (Level 1), and fully restricted (Level 2). Table 3-2 shows the three generic security levels, from least restrictive to most restrictive, including the differences between each level.

Table 3-2. Generic Security Levels

Level	Scenario	SW-DP Access Policy	APP Debug Policy	Mass Erase Policy	Factory Reset Policy	TI FA Policy
0	No restrictions	EN	EN	EN,DIS(1)	EN	EN
1	Custom restrictions	EN	EN, EN with PW, DIS	EN, EN with PWDIS	EN, EN with PW,DIS	EN, DIS
2	Fully restricted	DIS	Do not care(2) (access in not possible when SW-DP is disabled)			

(1) MSPM0C1103/4 does not support mass erase operations.

(2) When the SW-DP policy is *SW-DP disabled*, the mass erase and factory reset policies are a don't care from the point of view of the SWD interface. However, if the bootstrap loader (BSL) is enabled, the mass erase and factory reset policies do impact the available BSL functions. See the BSL security section (Section 3.2.6) for details on securing the BSL.

By default, TI provides MSPM0 in an unrestricted state. The unrestricted state allows for easy production programming, evaluation, and development. However, this unrestricted state is not recommended for mass production, as the unrestricted states leaves a large attack surface present.

There are 4 main uses of the SWD interface to take into consideration when determining protection needs:

- Application debug access: Debugging in the IDE tool
- Mass erase access: Erase the MAIN memory region
- Factory reset access: Erase the MAIN memory region and reset the NONMAIN device configuration memory to TI factory defaults (Level 0)
- TI failure analysis access: Ability for TI to initiate a failure analysis

The SWD security policies are implemented as 16-bit pattern-match fields in the NONMAIN memory, with the following characteristics:

- An exact pattern match is required to enable lower security states
- Any value in the 16-bit field mismatching the exact defined patterns results in a maximally secure state for the respective parameter

SWD also supports customizing four sets of passwords (128 bits in total), used to unlock device using a debug subsystem mailbox (DSSM) command. See the *Hardware Programming and Debugger Guide for MSPM0* application note for detailed instructions.

3.1.2.1 Access Policy

BCR supports modifying the SWD access policy through the BOOTCFG0.SWDP_MODE field, which configures the device in a maximally restrictive state (level 2).

When level 2 is selected (SW-DP disabled), the physical debug port (SW-DP) is completely disabled, and all of the SWD-accessible functions (application debug, mass erase, factory reset, and TI failure analysis) are inaccessible through the SWD, regardless of individual configurations.

If the BSL is enabled, then the mass erase and factory reset configuration fields are still used by the BSL to authorize mass erase or factory reset commands originating from the BSL interface. See Section 3.2 for details.

3.1.2.2 Debug Policy

When SWD access policy remains enabled, the user can further modify SWD debug policy through the BOOTCFG0.DEBUGACCESS field.

MSPM0 supports SWD enable, disable, and encryption with a *128-bit* password. Device only verify the password during boot stage. The user generates a DSSM command for password authentication, which includes a reset behavior to unlock the SWD interface.

MSPM0 provide two types of password storage, plain text and SHA2-256 digest. Passwords are stored in the PWDDEBUGLOCK field. The plain text passwords are directly stored in NONMAIN, resulting in the risk of password leakage. The SHA2-256 digest storgae pattern does not directly store the *128-bit* password in NONMAIN memory region. Instead, the user generates a *256-bit* hash value from the *128-bit* password and stores the hash value in the NONMAIN, which enhances the security of the password storage.

Below is how the user generates and sets a SHA2-256 digest password:

1. Determine the *128-bit* password that unlocks the SWD interface. Set the password into 32-bit alignment to form 4 *32-bit* passwords.
2. Reverse the 4 *32-bit* password endianness as SHA uses *8-bit* addressing format.
3. When calculating hash value, combine 4 reversed *32-bit* password as one string.
4. Calculate the SHA256 value of the input string.
5. Break the output SHA256 value into 8 *32-bit* words.
6. Reverse the endianness of the output 8 *32-bit* words as SHA2-256 calculation uses *8-bit* addressing format.
7. Store the reversed passwords into the PWDDEBUGLOCK.DIGEST field in sequence.

Note

The *0* value in the password is important. Do not delete the *0* value during calculation.

The flow is designed for all *256-bit* SHA2-256 digest password generation if device supports, including:

- SWD access
- Factory reset
- Mass erase
- BSL access (*256-bit* password)

Section 4.2 shows how to set the password using the SysConfig tool. Section 6.3 shows how to unlock device using the CCS tool.

3.1.2.2.1 Plain Text Password Example

Figure 3-1 shows the case where the SWD password is stored in plain text, if the user sets the *128-bit* password as *0123456789ABCDEF67452301EFCDAB89*.

	32-bit Hex	*8-bit Hex*
PWDDEBUGLOCK[0].DIGEST :	0x01234567	67 45 23 01
PWDDEBUGLOCK[1].DIGEST :	0x89ABCDEF	EF CD AB 89
PWDDEBUGLOCK[2].DIGEST :	0x67452301	01 23 45 67
PWDDEBUGLOCK[3].DIGEST :	0xEFCDAB89	89 AB CD EF

Figure 3-1. Plain Text Password Example

3.1.2.2.2 SHA2-256 Password Example

Below is a calculation example for the SHA2-256 digest password, if the user sets the *128-bit* password as *0123456789ABCDEF67452301EFCDAB89*.

1. Set the password into 32-bit alignment: 0x01234567, 0x89ABCDEF, 0x67452301, 0xEFCDAB89
2. Reverse the password endianness as SHA2-256 is calculated in byte: 0x67452301, 0xEFCDAB89, 0x01234567, 0x89ABCDEF
3. Combine the 4 reversed *32-bit* password as one string: 67452301EFCDAB890123456789ABCDEF
4. Calculate the SHA2-256 value (select *HEX* as the input encoding): 8420347FCB0F019E15564A0F65B8E197ECDF9F92D1ECA2BBAB1B8CB314C763DA (SHA256 online tool)
5. Break up the SHA2-256 value into 8 *32-bit* words: 0x8420347F, 0xCB0F019E, 0x15564A0F, 0x65B8E197, 0xECDF9F92, 0xD1ECA2BB, 0xAB1B8CB3, 0x14C763DA
6. Reverse the output endianness: 0x7F342084, 0x9E010FCB, 0x0F4A5615, 0x97E1B865, 0x929FDFEC, 0xBBA2ECD1, 0xB38C1BAB, 0xDA63C714
7. Store the 8 *32-bit* passwords in the PWDDEBUGLOCK.DIGEST field in order

3.1.2.3 Mass Erase and Factory Reset Policy

A *SWD Mass Erase* is an erase of the MAIN flash regions only, which typically includes the user application. The BCR and BSL policies stored in the NONMAIN flash region are not affected by a mass erase. A mass erase is useful for erasing all application code and data while leaving the device NONMAIN configuration intact.

A *SWD Factory Reset* is an erase of the MAIN flash regions followed by a reset of the NONMAIN flash region to default values. Such an erase is useful for completely resetting the BCR and BSL device boot policies while also erasing the application code and data.

The BCR configuration provides mass erase and factory reset functions through commands sent to the device over the SWD interface from a debug probe using the DSSM. These commands are not available in SWD security level 2, but the commands are optionally available in security level 0 and level 1. When the device is not configured for SWD security level 2, there are three configure positions possible for the mass erase and factory reset commands: enabled, enabled with a unique 128-bit password, or disabled.

Note

When Factory Reset is individually configured as disable, then BSL can *not* perform factory reset command.

See device-specific data sheet to determine the supported *128-bit* password storage pattern (plain text or SHA2-256). Refer to Section 3.1.2.2 for instructions on calculating the SHA2-256 digest password, and Section 8.2 for the flow to generate a SWD Mass Erase or Factory Reset.

3.1.2.4 TI Failure Analysis

TI failure analysis access is the ability for TI to initiate a failure analysis (FA) return flow through SWD. The TI FA flow always forces a factory reset before FA access is given to TI. This establishes that TI does not have a mechanism to read proprietary customer information stored in the device flash memory when a failure analysis flow is initiated.

Users can disable TI failure analysis using the BOOTCFG1.TI_FA_MODE field. This access prevents TI from performing a factory reset and further FA activities on the device.

3.1.3 Flash Memory Static Write Protection

The flash memory protection and integrity policies specify which sectors of flash memory are locked from modification, as well as which sectors are require integrity checks during the boot process before the user application begins.

Two types of flash static protection are supported: Static Write Protection on MAIN Flash Memory and Static Write Protection on NONMAIN Flash Memory.

Note

The DSSM command when generates mass erase or factory reset overrides the specified static write protection policy. While the BSL command does *not* override, the corresponding protected flash area remains unchanged.

3.1.3.1 MAIN Flash Static Write Protection

For devices with up to 32KB flash memory, included is a FLASHSWP0.DATA field to protect the whole 32KB memory. A value of *0* indicates the write protection applies to the corresponding sector. For FLASHSWP0, the 1 bit represents 1 sector in flash memory (1KB).

For devices with up to 256KB flash memory, included are the FLASHSWP0 and FLASHSWP1 fields to protect the complete 256KB memory. A value of *0* indicates the write protection applies to the corresponding sectors. For FLASHSWP1, 1 bit represents 8 sectors in flash memory, and the lower 4 bits are ignored.

For the devices with up to 512KB flash memory, included are the FLASHSWP0, FLASHSWP1 and FLASHSWP2 fields to protect the whole 512KB memory. A value of *0* indicates the write protection applies to the corresponding sectors. For FLASHSWP2, 1 bit represents 8 sectors in flash memory, starting from 256K.

Note

When bank swap features are enabled and swapped in multibank devices, the static write protection address also swaps.

3.1.3.2 NONMAIN Flash Static Write Protection

When configures BOOTCFG4 to protect NONMAIN, the entire NONMAIN region is write-locked. This makes the region functionally immutable when the boot configuration routine transfers execution to either the bootstrap loader or the user application code in the MAIN flash. Any attempt to program or erase the NONMAIN by the

application code or the bootstrap loader results in a hardware flash operation error, and the sector remains unmodified.

3.1.4 Customer Secure Code (CSC)

Customer Secure Code (CSC) is the code that runs after the TI boot code. The CSC executes from flash memory to further configure security elements. The CSC is customer-owned secure software, verify that CSC is located at the 0x0 address. Figure 3-2 shows the simplified flow of CSC execution.

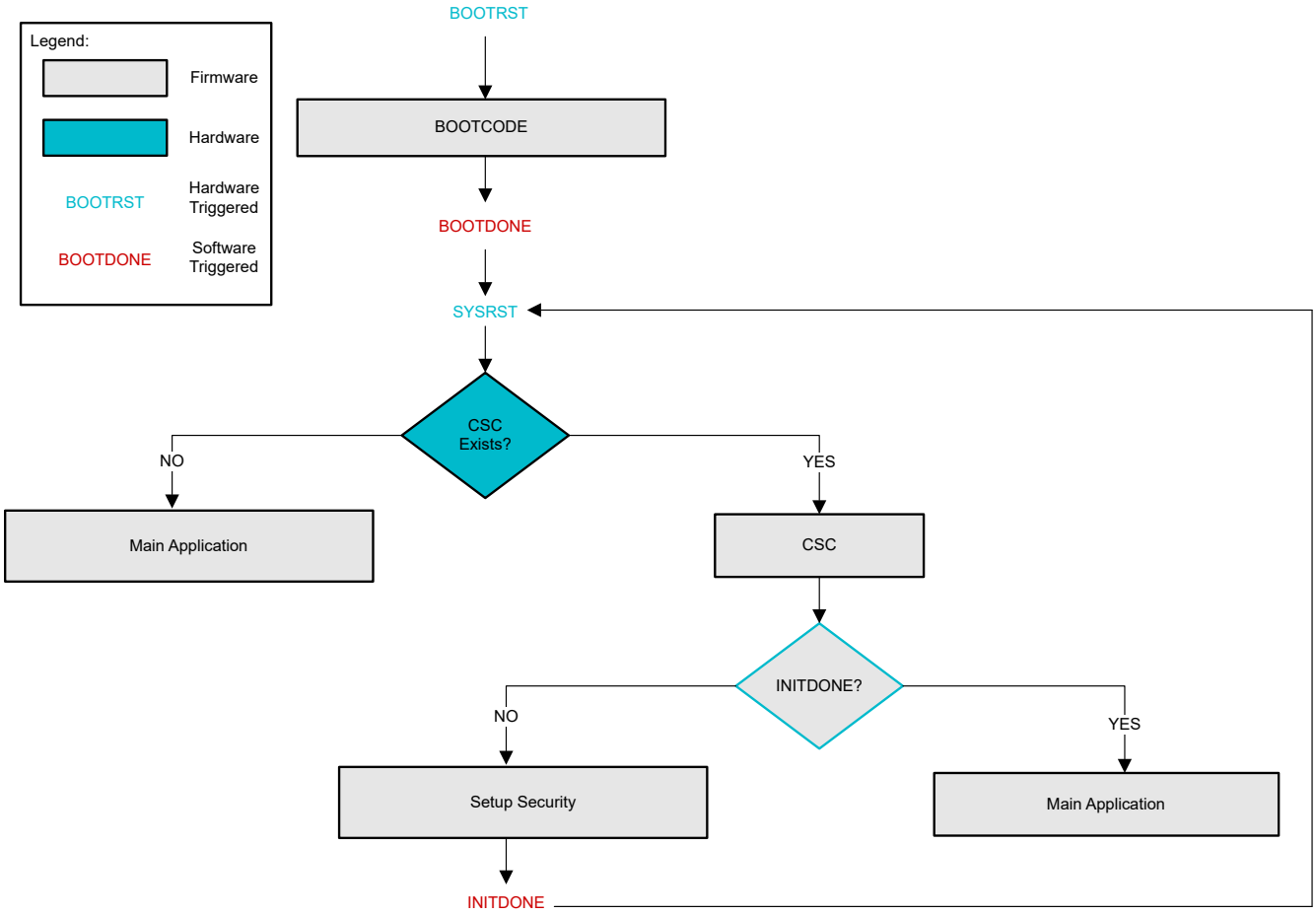


Figure 3-2. Secure Boot and Start Sequence

TI security model concept trusts both TI boot code and CSC. To trust the CSC, TI recommends enabling the Debug Hold and Flash Memory Static Write Protection.

3.1.4.1 CSC Policy

By default, CSC is disabled. The user enables CSC using the BOOTCFG5.CSCEXISTS field. When the CSC completes execution, the system undergoes a second SYSRST, followed by the main application starting.

Note

When CSC enables, boot code bypass BSL invoke pin check during the boot stage. The device can *not* enter BSL mode through a hardware invoke. In the meantime, the software invoke method keeps available.

3.1.4.2 Flash Bank Swap Policy

In multiple bank devices, the bank swap is disabled by default. Multiple banks have the same property (Write, Read, and Execute) with different memory addresses.

Users enable the bank swap using the BOOTCFG5.FLASHBANKSWAPPOLICY field. Boot-code read this configuration and writes to the SYSCTL.SECCFG.FLBANKSWPPOLICY with the appropriate KEY (0xCA). With the hardware default, the swappable configuration is ENABLED, and the lower bank is used as a logical bank 0. While the default value of BOOTCFG5.FLASHBANKSWAPPOLICY field disables bank swap policy, so that the complete main flash region has Read, Write, Execute access property.

The bank swap behavior happens in the CSC stage, while the behavior goes into effect after the device issues INITDONE. Based on which bank (or pair) is executable, that bank (pair) receives reads and executes privileges and loses write and erase privileges. The other bank (or pair) is readable and writable, but not executable. This mechanism enforces the policy where after INITDOWN issued, the only save location for firmware updates is the writable bank, and can never be executed.

Note

The complete main flash region has Read, Write, and Execute access property in the CSC stage, even if users enable the bank swap.

For more details on bank swap features, see the *Flash Multi Bank Feature in MSPM0 Family* application note.

3.1.4.3 Debug Hold

For the devices that support CSC features, BCR configuration further provides the property lock the access ability of the SWD interface during CSC execution. Device release debug access until INITDOWN is issued in CSC.

The Debug Hold feature is disabled by default, and TI recommends users keep it disabled during application debugging. When the product enters the mass production stage with CSC enabled, TI recommends enabling the Debug Hold using the BOOTCFG4.DEBUGHOLD field to protect CSC in an inaccessible state.

3.1.5 Fast Boot Mode

Reduce the execution time of the BCR by enabling fast boot mode. Fast boot mode speeds up the boot process using the following methods:

- Limiting the BSL entry: By enabling fast boot mode, enter the bootloader using only the SYSCTL register invoke method and the DSSM invoke method.
- Bypassing the Application Digest Check, even if the application digest check is enabled. See Section 3.1.6 for more information.

Set the fast boot mode to be enabled using the BOOTCFG2.FASTBOOTMODE field in the NONMAIN flash memory.

3.1.6 Application Digest Check

The BCR supports the execution of complete CRC32 or SHA2-256 integrity check of the application code and data contained in the MAIN flash regions, during the boot process, before starting the user application. The Application Digest Check is useful to verify the integrity of some or all of the application code and data before executing it.

Follow the steps below to enable the application digest check:

1. Select the application digest check policy, which can be set as CRC32, or SHA2-256 (BOOTCFG6.APPDIGESTMODE field)
2. Set 32-bit starting address of the application digest check (APPDIGESTSTART.ADDRESS field)
3. Set the length of the application (specified in bytes) for which the CRC or SHA digest check applies (APPDIGESTLENGTH.LENGTH field).
4. Calculate the CRC32/SHA2-256 value with the given data (specified in bytes)
5. Break the precalculated CRC32/SHA2-256 value into 32-bit format and store it into BCR configuration (APPDIGEST.DIGEST field).

In the event that an application digest check fails at boot, the application in the MAIN flash does not start. If the BSL is enabled, then the BSL is entered. If the BSL is not enabled, then the boot fails.

3.1.6.1 CRC32 Digest Check Example

Below is a calculation example for the CRC32 integrity check, if the user sets the start address as 0x0001.0000 and length as 8 bytes. The application data refers to Section 3.1.6.

	32-bit Hex	*8-bit Hex*
0x0001.000 :	0x01234567	67 45 23 01
0x0001.004 :	0x89ABCDEF	EF CD AB 89

Figure 3-3. Example Data for CRC32 Digest Check

Select the *JAMCRC* as the calculation model to get CRC32 output value. The *JAMCRC* CRC model has the reversed input/output bit, and uses the polynomial of 0x04C11DB7 with an initial value of 0xFFFFFFFF, and an output XOR value of 0x00000000.

Follow the steps below to calculate the example CRC32 value:

1. Select the application digest check policy as check with CRC (BOOTCFG6.APPDIGESTMODE = AABBh)
2. Set 32-bit starting address of the application digest check (APPDIGESTSTART.ADDRESS = 0001.0000h)
3. Set the length of the application (specified in bytes) for which the CRC digest check applies (APPDIGESTLENGTH.LENGTH = 8h)
4. Calculate the CRC32 output with the given input
   a. Reverse the input data endianness as CRC32 is calculated using an 8-bit format: 67, 45, 23, 01, EF, CD, AB, 89
   b. Combine the input 8-bit data as one string: 67452301EFCDAB89
   c. Calculate the CRC32 value (select *HEX* as the input encoding, *JAMCRC* as the model): 0x24648719 (CRC32 online tool)
5. Register CRC32 output value in BCR configuration (APPDIGEST.DIGEST = 2464.8719h). If there has multiple APPDIGEST.DIGEST registers, only the first 32-bit word of the DIGEST is used for CRC32

Note

Verify that the length of the application CRC32 digest check is an even number.

3.1.6.2 SHA2-256 Digest Check Example

Below is a calculation example for the SHA2-256 integrity check, if the user sets the start address as 0x0001.0000 and length as 8 bytes. Figure 3-4 shows the example application data.

	32-bit Hex	*8-bit Hex*
0x0001.000 :	0x01234567	67 45 23 01
0x0001.004 :	0x89ABCDEF	EF CD AB 89

Figure 3-4. Example Data for SHA2-256 Digest Check

Follow the steps below to calculate the example SHA2-256 value:

1. Select the application digest check policy as SHA2-256 (BOOTCFG6.APPDIGESTMODE = CCDDh)
2. Set 32-bit starting address of the application digest check (APPDIGESTSTART.ADDRESS = 0001.0000h)

3. Set the length of the application (specified in bytes) for which the SHA2-256 digest check applies (APPDIGESTLENGTH.LENGTH = 8h)
4. Calculate the SHA2-256 output with the given input
   a. Reverse the input 32-bit data endianness as SHA2-256 is calculated in byte: 67, 45, 23, 01, EF, CD, AB, 89
   b. Combine the input 8-bit data as one string: 67452301EFCDAB89
   c. Calculate the SHA2-256 value (select *HEX* as the input encoding): FDD232547CEEE35ADD35783CA7D518D2A49ABCCD0B60B028C8A9C629EBA6201F (SHA2-256 online tool)
   d. Break up the SHA2-256 value into 8 32-bit words: 0xFDD23254, 0x7CEEE35A, 0xDD35783C, 0xA7D518D2, 0xA49ABCCD, 0x0B60B028, 0xC8A9C629, 0xEBA6201F
5. Reverse the 32-bit output words endianness: 0x5432D2FD, 0x5AE3EE7C, 0x3C7835DD, 0xD218D5A7, 0xCDBC9AA4, 0x28B0600B, 0x29C6A9C8, 0x1F20A6EB
6. Store the 8 32-bit SHA2-256 checksums in the APPDIGEST.DIGEST field in order

3.1.7 BSL Policy

For the devices support ROM or Flash BSL, TI provides access ability to the device memory through the BSL interface. One default BSL invoke pin is used for hardware invoking BSL, see Section 3.2.2 for more details.

User can disable BSL through the BOOTCFG2.BSLMODE field, which fully disables BSL mode. The BSL software invoke (through the SYSRST) is also blocked.

Alternatively, disable BSL invoke pin check using the BOOTCFG1.BSL_PIN_INVOKE field blocks the BSL hardware entry. After disable the invoke pin check, the device does *not* check the invoke pin status during the boot, so that the BSL hardware entry is disabled. Users can enter the BSL using software invoke method in the application code.

3.1.8 BCR Checksum

The MSPM0 device provides a CRC check for BCR configuration data during the boot stage. If the device supports CRC32-ISO3309, the 32-bit CRC digest is applied. Otherwise, the CRC16-CCITT and 16-bit CRC digest is applied.

Below is the configuration for CRC calculation:

- Select a polynomial with the preselected CRC standard (CRC32-ISO3309 or CRC16-CCITT)
- Reflect the input and output value
- Initial value is set as 0xFFFFFFFF (or 0xFFFF)
- Final XOR value is set as 0x00000000 (or 0x0000)

TI recommends using the SysConfig tool to automatically calculate BCR checksum, see Section 4.2 for more details.

3.1.8.1 CRC Check Fail Handling

If the BCR configuration data fails the CRC check during boot, a catastrophic boot error results and the following limitations are imposed:

- The error cause is logged in the CFG-AP as a boot diagnostic
- The BSL is not invoked, even if the BSL was configured as enabled
- The user application is not started
- No application debug access is enabled
- A pending SWD factory reset command, if enabled or enabled with password, is honored
- A pending TI failure analysis flow entry, if enabled, is honored
- The boot process re-attempt up to 3 times
  - If the 2nd or 3rd attempt passes, the device boots normally
  - If the 3rd attempt does not pass, no further boot attempts are made until the next BOR or POR

3.2 BSL Configuration

The bootstrap loader (BSL) routine is the ROM based firmware used to load data to the device memory. Users set different properties for BSL entry and execution through the BSL configuration of the NONMAIN.

TI recommends using the SysConfig tool to customize the BSL configuration, see Section 4.3 for more details.

3.2.1 BSL Configuration ID

TI defines a default BSL configuration ID for different NONMAIN layouts. Users can overwrite BSL ID for application usage. Perform a factory reset to recover the default BSL configuration ID.

Default BSL Configuration ID shows the default values of each device:

Table 3-3. Default BSL Configuration ID

NONMAIN Layout Type	Device Family	Default Value
Type A	MSPM0L110x	0x00000001
	MSPM0L130x	
	MSPM0L134x	
	MSPM0Gx10x	
	MSPM0Gx50x	
Type B	MSPM0C1103	Not supported
	MSPM0C1104	
	MSPS003Fx	
Type C	MSPM0Lx22x	0x00000001
Type D	MSPM0C1105	0x05000000
	MSPM0C1106	
	MSPM0H321x	
Type E	MSPM0L111x	0x06000000
	MSPM0G511x	
	MSPM0G5187	
Type F	MSPM0Gx51x	0x04000000
	MSPM0G352x	

3.2.2 Invoke Pin Configuration

The bootloader supports hardware invoking after a BOOTRST by using a GPIO. TI devices are configured with a specific GPIO and polarity. See the device-specific data sheet for the default invoke pin assignment.

Uers modify the default invoke pin using the BSLCONFIG0 field. The invoke pin configuration contains the:

- Pad
- Port
- Pin
- Polarity

If the default invoke pin initial status in the schematic design matches the invoke pin configuration, the device enters the BSL after powers up and never run application code.

3.2.3 ROM-Based Communication Interface

MSPM0 supports configurable I2C or UART as the serial wire BSL interface. The ROM-based BSL allows the user to customize the UART or I2C pins, but not include the communication interface instance (for example, from UART0 to UART1). Users can customize the UART or I2C pins, see device-specific data sheet for the default pin assignment.

For the device with a USB interface, the ROM-based interface further supports the device firmware upgrade (DFU) option.

3.2.3.1 UART Interface

For the device support of the ROM BSL, the UART interface is available. The BSL configuration supports the modification of the UART pins configuration (pad and MUX) through the BSLPINCFG0 field. The attribute property of the device pin restricts the available pins as the UART instance is unchanged. See device-specific data sheet for the available pins.

The default baud rate of the UART is 9600. User can dynamically modify UART communication baud rate in the BSL command (the command is sent with the 9600 baud rate). Furthermore, the type-E layout of the NONMAIN support a modified default baud rate through the BSLCONFIG1.UART_DEFBAUDRATE field.

3.2.3.2 I2C Interface

For the device that supports the ROM BSL, the I2C interface is available. The BSL configuration supports the modification of the I2C pins configuration (pad and MUX) through the BSLPINCFG1 field. The attribute property of the device pin restrict the available pins for the specific I2C instance. See device-specific data sheet for the available pins.

The default target address of the I2C is 0x48. MSPM0 supports modify target address through the BSLCONFIG2.I2CTARGETADDR field. The maximum support is a 7-bit target address.

For more details on BSL UART and I2C interface, refer to the *MSPM0 Bootloader* user's guide.

3.2.3.3 USB Interface

For the device with a USB peripheral, the USB interface is available for ROM-based BSL. The BSL configuration does not support the USB pins configuration. Refer to device-specific technical reference manual, linked in Section 10.

3.2.4 Flash Plug-in Interface

The ROM bootloader provides an option to add a custom interface implementation to the ROM BSL core as a Flash plug-in. The Flash Plug-in function gives an advantage of customizing the interface, without reimplementing the complete BSL core.

With the flash plug-in interface, either:

- A new interface, which is not available in ROM BSL, can be added to the interface list for auto detection
  - Examples: SPI, CAN, and so forth

or

- ROM interface implementation (UART / I2C) can be overridden
  - Examples: instance selection, parameter modification, and so forth.

To use this option, load the flash plug-in image to main flash and register the image address in the BSL configuration in the non-main flash memory. Verify that the flash memory static write protection is enabled for the flash memory region where plug-in images are loaded, to prevent flash plug-in getting erased during the bootloading process. Learn more about flash memory static write protection in Section 3.1.3.

Follow the steps below to enable the flash plug-in interface:

1. Set the presence of the flash plug-in interface, the BSLPLUGINCFG.FLASHPLUGINEXISTS field.
2. Define the plug-in interface type in the BSLPLUGINCFG.PLUGINTYPE field. The type options are: UART, I2C, or any new interface.
3. Define the size of the SRAM that the flash plug-in interface (BSLPLUGINCFG.SRAMUSED field) consumes. The maximum support is 0xFF.
4. Define the flash plug-in interface function pointer for: Init, Receive, Send, and the Deinit API.

The flash plug-in interface does *not* overwrite the ROM-based BSL protocol. Verify that the user-defined API function defined follows the ROM BSL standard. See the *MSPM0 Bootloader* user's guide for more details.

SDK example provides a series of examples for the flash plug-in interface implementation.

3.2.5 Alternative BSL Interface

The ROM BSL provides an option to use an alternative (secondary) BSL. To achieve this, load the alternative BSL image in the main flash memory and register the image address in the BSLCONFIG1.ALTBSLCONFIG field.

When BSL is invoked, the ROM BCR checks for the alternative BSL presence (BSLCONFIG1.ALTBSLCONFIG field). If the alternative BSL is enabled, the ROM BCR branches to the alternative BSL. The ROM BSL is not expected to execute at this moment.

The BSL policy (Section 3.1.7) in the BCR configuration also applies to secondary BSL. When the BSL policy setting is disabled, a hardware pin check or software reset does not invoke a secondary BSL.

To avoid unintended erase during the bootloading process, do write-protect the flash memory region where the secondary BSL is loaded in Main Flash Static Write Protection (Section 3.1.3.1). Non-main write protection is optional in case of a secondary bootloader. After NONMAIN erase, the secondary BSL API pointer should be properly restored.

The secondary BSL allows for further flexibility for the BSL customization. Modify the full configuration of communication interfaces, protocol, and security features based on the user expectation. If supported by the device, follow the below steps to enable the secondary BSL:

1. Set the invocation of the alternate BSL in main flash region, using BSLCONFIG1.ALTBSLCONFIG field.
2. Set the secondary BSL entry address of main flash memory, using SBLADDRESS.ADDRESS field.

Refer to *MSPM0 Bootloader (BSL)* user's guide and the *MSPM0 Bootloader Implementation* application note for more details on secondary BSL implementation.

SDK example provides a series of examples for the secondary BSL interface implementation.

3.2.6 BSL Security Configuration

The BSL core provides a branch of security features to prevent potential attack:

- Access password
- Read-out
- Alert
- App integrity check

The security features always take effect in the ROM BSL and the flash plug-in interface (Section 3.2.4). Users can fully customize and overwrite the BSL behavior in the alternative BSL interface (Section 3.2.5), so that the security features are optional for alternative BSL.

3.2.6.1 Access Password

A user-specified password protects ROM BSL access. The password is located in PWDBSL.PASSWORD field. There is no option to disable the password for BSL access.

Before unlock BSL by sending the correct password to the BSL core, there is no access granted to most BSL functions. Refer to the *MSPM0 Bootloader (BSL)* user's guide for more details. If a wrong password is provided to the BSL, the BSL halts for 2 seconds, after which make a secondary attempt to send the correct password. After three failed password attempts, the security alert feature (Section 3.2.6.3) activates.

The BSL access password supports two types: 256-bit plain-text and 256-bit SHA encrypted. The 256-bit plain-text password is stored directly in the BSL configuration. The other password is a 256-bit SHA encrypted password (the length of original password is 256-bit), refer to Section 3.1.2.2 for the SHA2-256 calculation flow. The BSL configuration stores the default value for the plain-text password (256-bit, all bytes are 0xFF) or SHA encrypted password (256-bit, calculated from 256-bit data with all bytes as 0xFF).

3.2.6.2 Read-Out Feature

The device support ROM BSL has a configurable propertyi n the BSL configuration to read the device memory back through the BSL interface. All flash memories are fully accessible through the BSL host, while partial SRAM

memory is restricted to access. See section 3.3.1 (SRAM Memory Usage) of the *MSPM0 Bootloader (BSL) user's guide* for more details.

By default, the read-out feature is disabled for safety. User can enable the read-out features through the BSLCONFIG0.READOUTEN field in the BSL configuration.

3.2.6.3 Alert Feature

The alert feature is applied in ROM BSL for cases where there are occurrences of more than 3 incorrect BSL password checks. By default, the device does not perform additional actions when a security alert is raised. User can configure the alert behavior through the BSLCONFIG2.ALERTACTION field.

In the BSL configuration, two additional alert behaviors are supported:

- Trigger a BSL factory reset. If the sectors in the MAIN or NONMAIN flash are static-write protected, the sectors are not affected by the BSL factory reset.
- Reconfigure the NONMAIN region to disable the BSL. Reconfiguring is not supported if the NONMAIN flash is static-write protected.

Set the flash memory static write protection (Section 3.1.3) to retain specific MAIN flash memories or the entire NONMAIN flash memory.

3.2.6.4 Application Integrity Check

The BSL supports returning an application version number through the BSL interface. This support allows the BSL host to interrogate the firmware version without the ability to read the firmware. The version field address is 32 bits in length. Program the version value in the corresponding MAIN flash.

Use the BSLAPPVER.ADDRESS field to set the address of the application version word. The version data only returns if the specified address corresponds to a valid flash memory address. If the given flash address is not programmed, then the ROM BSL returns 0h.

3.2.7 BSL Checksum

The MSPM0 device provides a CRC check for the BSL configuration data during the boot stage. BSL CRC checksum shares the same calculation flow as BCR checksum. See Section 3.1.8 for a detailed introduction.

TI recommends using a SysConfig tool to automatically calculate the BSL checksum, see Section 4.3 for more details.

3.2.7.1 CRC Check Fail Handling

If the BSL configuration data fails the CRC check during BSL invocation, a catastrophic boot error results and the following limitations are imposed:

- The error cause is logged in the CFG-AP as a boot diagnostic
- The BSL is not invoked, even if the BSL is configured as enabled
- The user application is not started
- No application debug access is enabled
- The boot process re-attempts up to 3 times
  - If the 2nd or 3rd attempt pass, the device boots normally
  - If the 3rd attempt does not pass, no further boot attempts are made until the next BOR or POR

4 NONMAIN Configuration With SysConfig

This section introduces how to configure the various functional options of the NONMAIN memory using the TI graphical configuration tool, SysConfig.

4.1 SysConfig Introduction

The SysConfig tool is an intuitive and comprehensive collection of graphical utilities for configuring pins, peripherals, subsystems, and other components, which helps users manage, expose, and resolve conflicts visually so that more time is spent on creating applications. The output files of SysConfig include header and source files for use with software development kit (SDK) examples or to configure custom project.

The SysConfig tool is delivered as a standalone installer. Manually link the installer into Code Composer Studio (CCS), IAR, or Keil, or use the cloud tools portal. As part of the TI ecosystem, CCS has integrated SysConfig by default. See the SysConfig User Guide for more details.

Figure 4-1 shows the part of the tool which configures the NONMAIN region, and user can select the *question mark* for more details on the component.

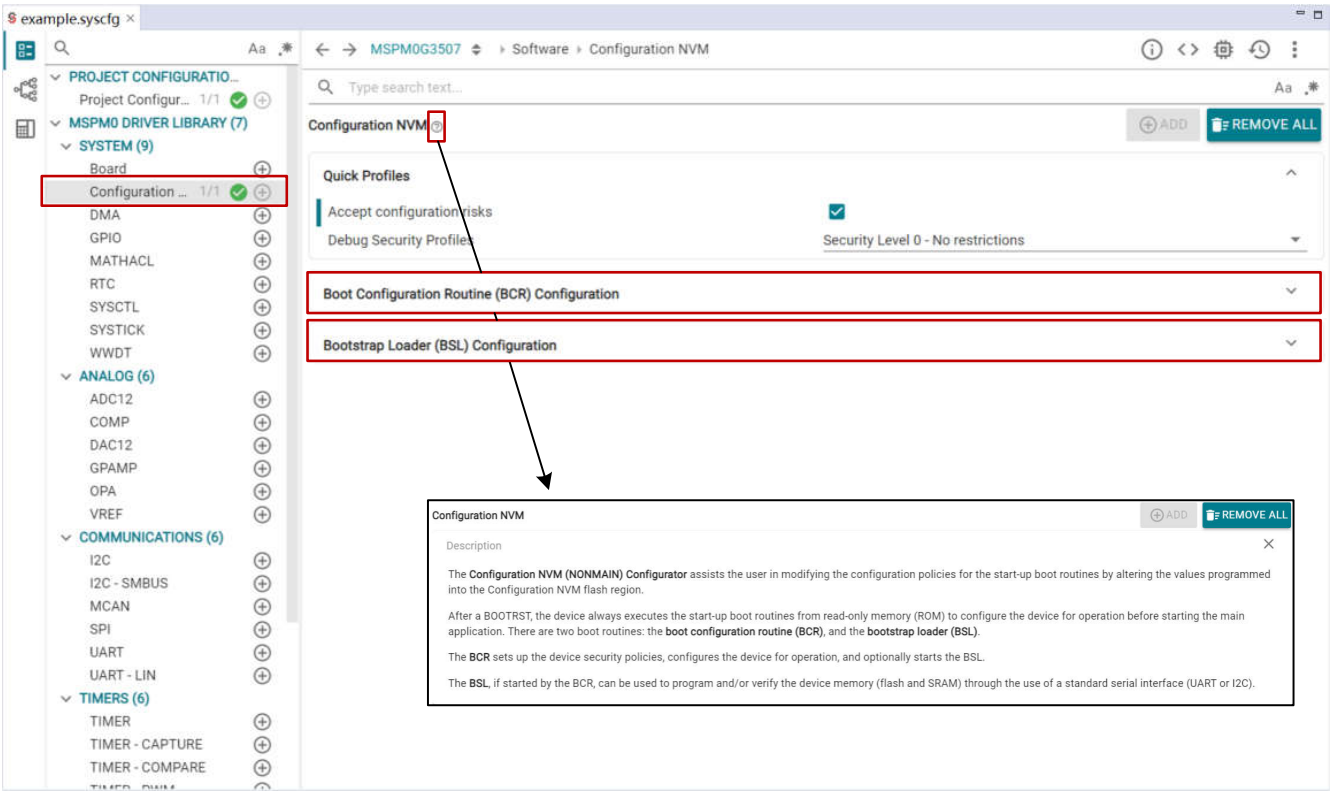


Figure 4-1. SysConfig Tool: Configure the NVM

4.2 BCR Configuration with SysConfig

Figure 4-2 shows the BCR configuration of the MSPM0G3507. According to the BCR configuration, the Debug Security Profiles can be configured as Level 0, Level 1, and Level 2, see Section 3.1.2 for a detailed introduction. The BCR Configuration ID is used with a default value and does not support modification in the tool. The CRC checksum is automatically calculated based on the customized BCR configuration.

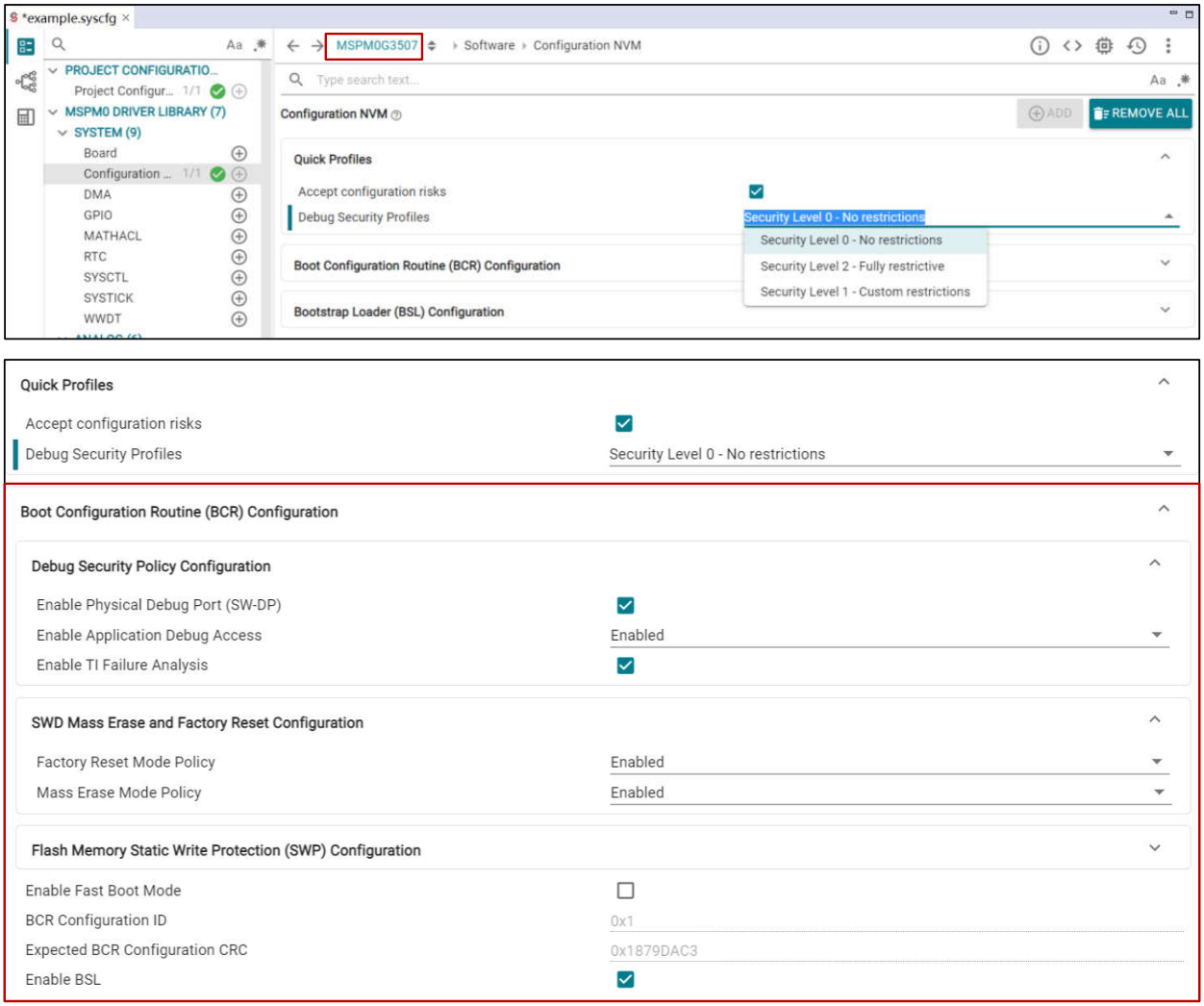


Figure 4-2. BCR Configurations

4.2.1 Password Configuration

Use a 128-bit plain text password to encrypt the SWD interface, Factory reset, and Mass Erase. For the devices supporting the SHA2-256 password, set the 256-bit encrypted password in NONMAIN BCR configuration. Figure 4-3 shows the SWD password configuration of MSPM0G3507 (TYPE-A) and shows the SWD password configuration of MSPM0G3519 (TYPE-F).

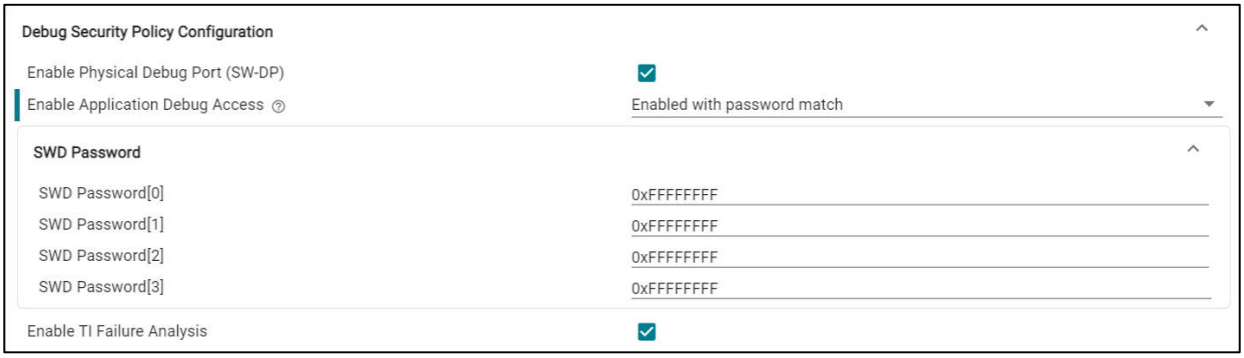


Figure 4-3. Plain Text SWD Password Configuration

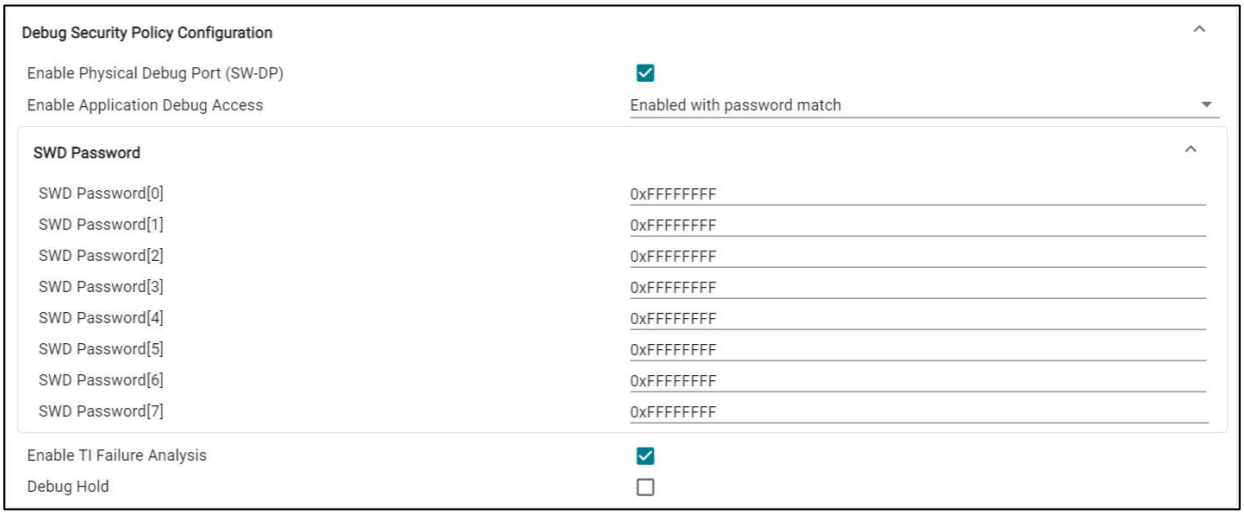


Figure 4-4. SHA2-256 SWD Password Configuration

The SysConfig tool integrates the SHA2-256 password generation flow (see Section 3.1.2.2), shown as Figure 4-5. If the device supports the SHA2-256 password, selecting the question mark next to the *SWD Password* to display the example process flow. A SHA256 online tool is also introduced in the example.

The example flow is designed for all 256-bit SHA2-256 password generation, including:

- SWD access
- Factory reset
- Mass erase
- BSL access

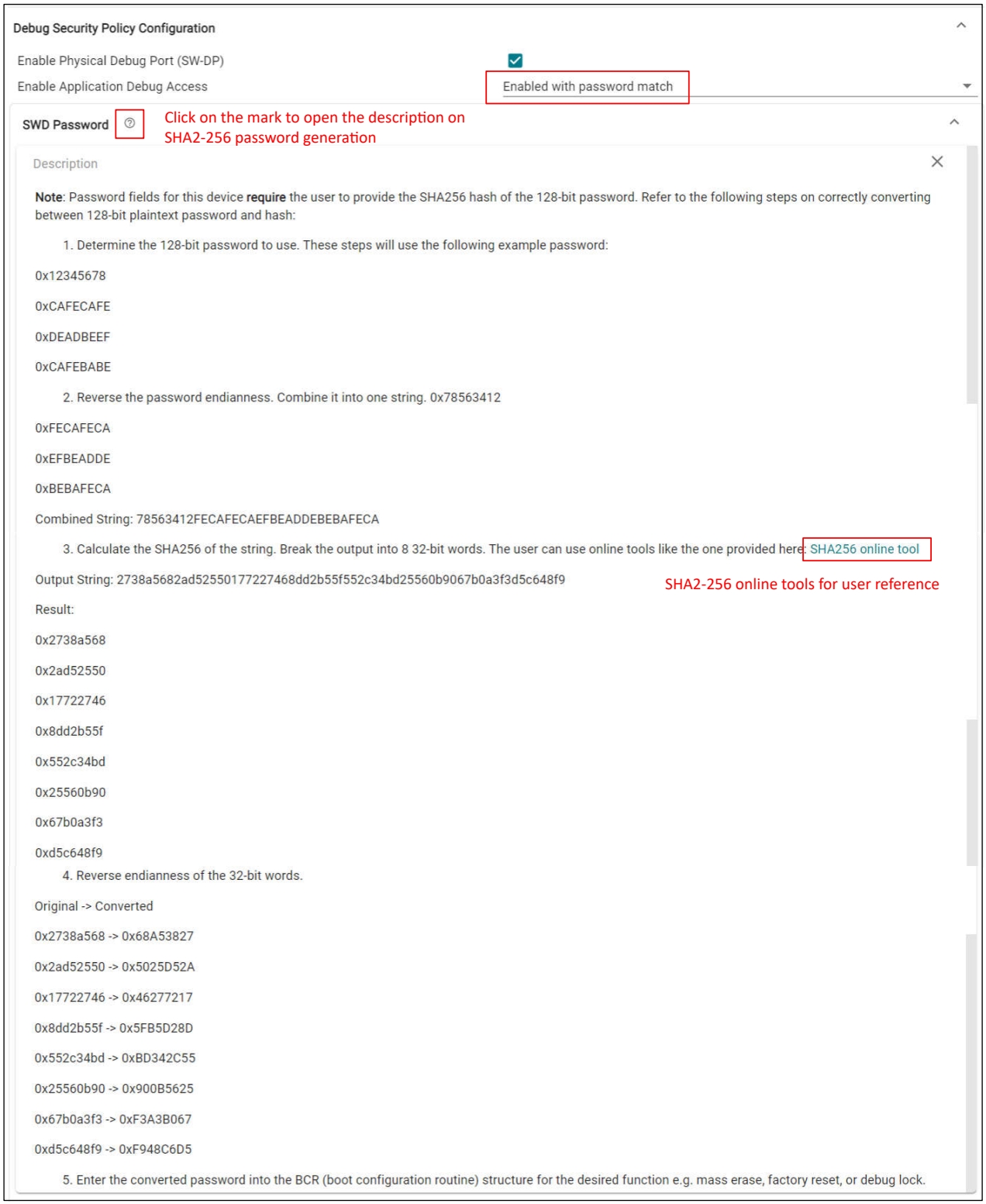


Figure 4-5. SHA2-256 Password Generation Flow

4.2.2 Flash Static Write Protection

To enable the Flash Static Write Protection, set the bit of the MAIN SWP as 0.Tthe minimum memory range is 1KB or 8KB and is dependent on the protected sector order. For more details, see Section 3.1.3.

Figure 4-6 shows how to configure the static write protection of MSPM0G3507. For FLASHSWP0 (Lower sectors), 1 bit represents 1 sector in flash memory (1KB), and a value of 0 indicates the write protection applies to the corresponding sector.

For FLASHSWP1 (the remaining sectors), 1 bit represents 8 sectors in flash memory. The lower 4 bits and higher 16-bit are ignored, which means FLASHSWP1 protects the memory address from 32KB to 128KB. A value of 0 indicates the write protection applies to the corresponding sectors.

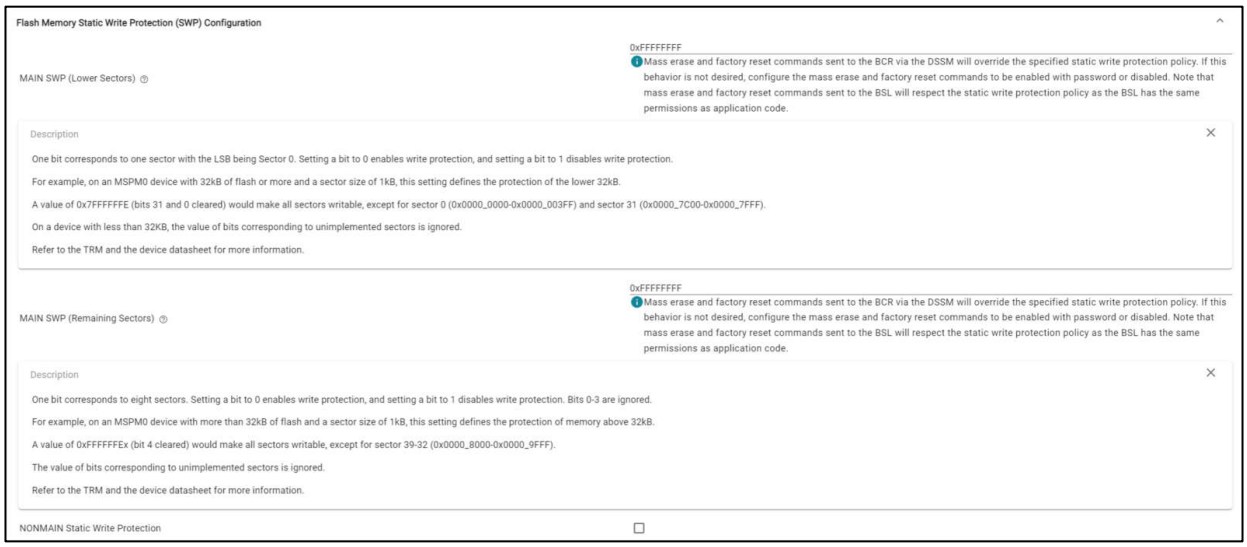


Figure 4-6. Flash Static Write Protection

The SysConfig tool also supports the NONMAIN static write protection, after the device locks the write/erase operation of FLASHCTL of the NONMAIN region are enabled. Only the factory reset generated using the DSSM command restore the NONMAIN.

4.2.3 Other BCR Configurations

Some BCR configurations in SysConfig are straightforward and can be enabled or disabled, including the following:

- Enable Fast Boot Mode
- Enable TI Failure Analysis
- Enable Debug Hold (if supported)
- Enable CSC Policy (if supported)
- Enable Flash Bank Swap Policy (if supported)
- Enable BSL Mode (if supported)
- Enable BSL Invoke Pin Check (if supported)
  - In the SysConfig tool, instead of the *BCR Configuration*, the *BSL Configuration* includes the selection of *Enable BSL Invoke Pin Check*, along with other BSL GPIO invoke configurations

Note

The SysConfig tool does not support Application Digest Check configuration. See Section 3.1.6 for more details.

4.3 BSL Configuration With SysConfig

Figure 4-7 shows the BSL configuration of the MSPM0L1107 (type-E layout).

Figure 4-7. BSL Configurations

4.3.1 BSL Access Password

For the device to support ROM or Flash BSL, the 256-bit BSL access password is always enabled. There are two types of password for MSPM0. The first type is a 256-bit plain text password, with the default of an 0xFF value. As Section 4.3 shows, the second type is SHA2-256 encrypted, the original password is all 0xFF (256-bit), so that the SysConfig tool stores the generated SHA2-256 value of the original password into the PWDBSL.PASSWORD field.

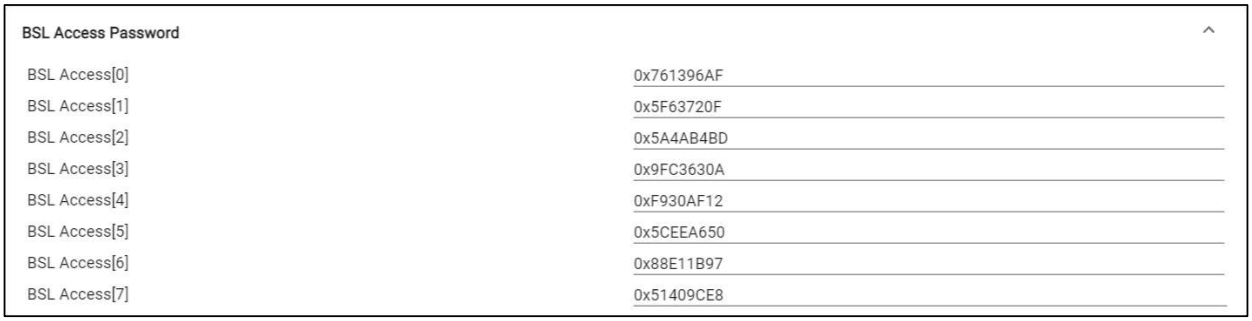


Figure 4-8. SHA2-256 BSL Default Access Password

4.3.2 BSL Invoke Pin Configuration

The device supports the ROM or Flash BSL using a customized invoke pin property. Section 4.3 shows the details of the configuration.

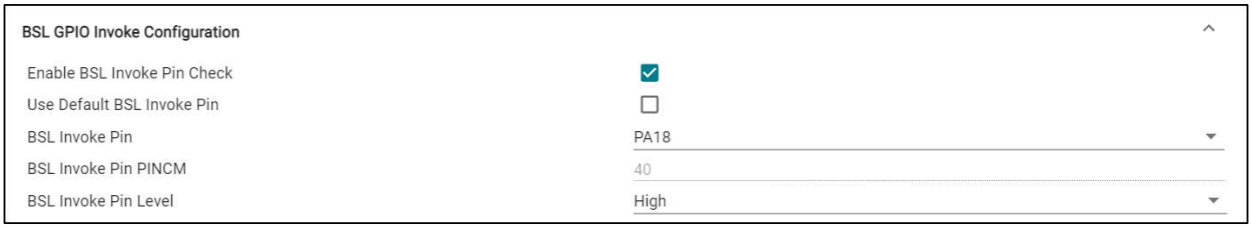


Figure 4-9. BSL Invoke Pin

If the user disables the BSL Invoke Pin Check in the BSL configuration, the BCR skips the hardware pin invoke check while software BSL entry is still available. After disabled, the BOOTCFG1.BSL_PIN_INVOKE field of the BCR configuration is set as 0xFFFF in the SysConfig tool, and BSL PIN configurations remained unchanged.

4.3.3 BSL Communication Interface

For the device supports the ROM BSL, provided are the I2C and UART interfaces as common BSL interfaces. For the device supports the USB peripheral, a USB interface supporting the DFU is applied as BSL interface.

The BSL configuration provides flexibility for user to customize the BSL interface, as shown in Section 4.3. By default, the BSL configuration supports pin selection and I2C target address change. For the device supports UART baud rate modification, the user selects nine different common baud rates for UART communication. Otherwise, UART initial baud rate is set as 9600. For higher data throughput, send the BSL command at the initial 9600 baud rate to modify the UART baud rate.

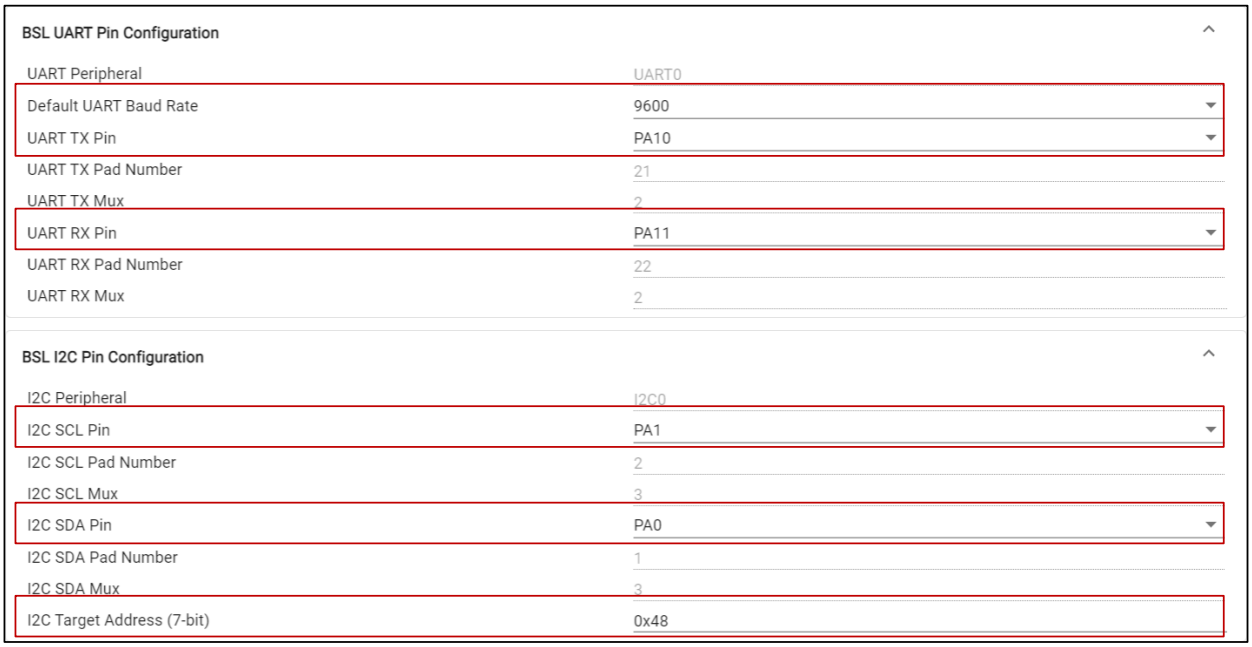


Figure 4-10. BSL Communication Interface

4.3.4 Flash Plug-in Interface

For the device supporting the ROM BSL, BSL configuration provides flexibility for the user to customize the BSL interface (such as SPI, UART, IIC, CAN, and so forth) which loads into the MAIN flash memory. Register the memory address of the flash plug-in interface in NONMAIN BSL configuration.

Section 4.3 shows that the flash plug-in interface has 4 APIs, including:

- Init
- Recieve
- Transmit
- De-init

The ROM BSL calls the 4 APIs through the hooks registered in the BSL configuration. For more details, see chapter 7 (*Interface Plug-in*) of the *MSPM0 Bootloader (BSL)* user's guide.

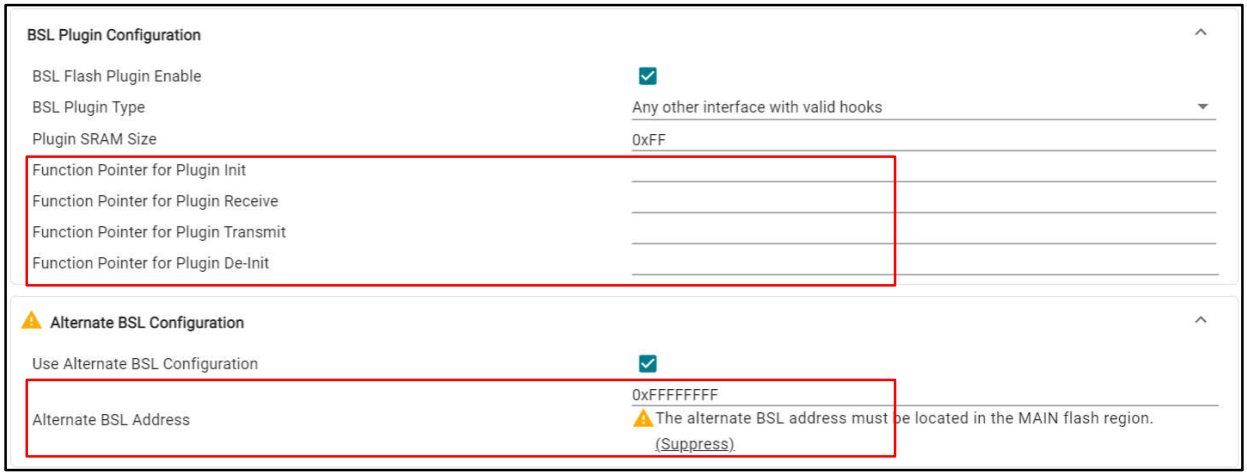


Figure 4-11. BSL Plug-in and Alternative BSL Configuration

4.3.5 Alternative BSL Interface

After selecting to use the Alternative BSL Configuration in SysConfig tool, the user can set the start address of the alternative BSL (located in MAIN flash) into BSL configuration register, as shown in Section 4.3. See section 6 (*Secondary Bootloader*) of the *MSPM0 Bootloader (BSL)* user's guide for more details.

After enabling the alternative BSL, the ROM BSL is skipped and alternative BSL is called when the BCR asserts the BSL invoke condition.

4.3.6 Other BCR Configurations

Other than BSL configuration ID, there are a few more features that support customization using SysConfig tool (not available for all NONMAIN layout type), as shown in Figure 4-7, including:

- BSL App Version
- BSL Read Out Enable
- BSL Security Alert Configuration
- Disable NRST

Whenever the BSL configuration changes, SysConfig tool automatically calculates the *CRC Checksum* value, so the user only focuses on the customized BSL features and can easily generate the NONMAIN firmware files.

5 NONMAIN Configuration in Application Code

The NONMAIN region also support dynamic modification with the FLASHCTL, which is widely used in application code and customized bootloader. Figure 5-1 and SDK example provide a generic flow for modifying the NONMAIN flash field in application codes for user reference.

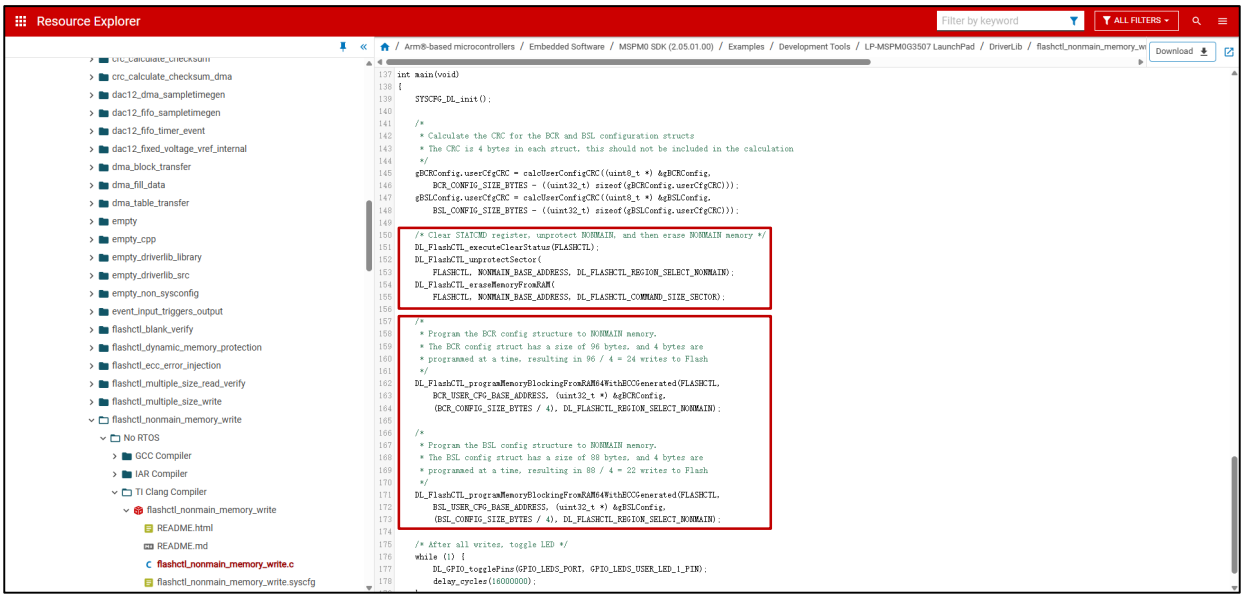


Figure 5-1. Example Project with NONMAIN Modification

ECC code is required to program with the NONMAIN memory, as the boot code reads the NONMAIN memory with ECC check. Every NONMIAN register modification requires the user to erase the entire NONMAIN sector and load in the new value of entire NONMAIN configuration register.

TI recommends erasing and programing the NONMAIN field in a safety state. Frequently updated NONMAIN configurations in the application code introduces the risk that the NONMAIN field becomes empty (unstable power supply, and so forth.).

6 NONMAIN Operation with IDE Tool

TI recommends using the SysConfig tool to manage the NONMAIN configuration. The SysConfig tool integrates into the Code Composer Studio (CCS). TI provides the standalone version of SysConfig to use with other IDE tools, such as Keil or IAR. For more details on using SysConfig with different IDE tools, see Section 10.

6.1 NONMAIN Configuration Files

Figure 6-1 shows that the SysConfig tool generates two files (boot_config.c and boot_config.h) to configure the NONMAIN flash memory.

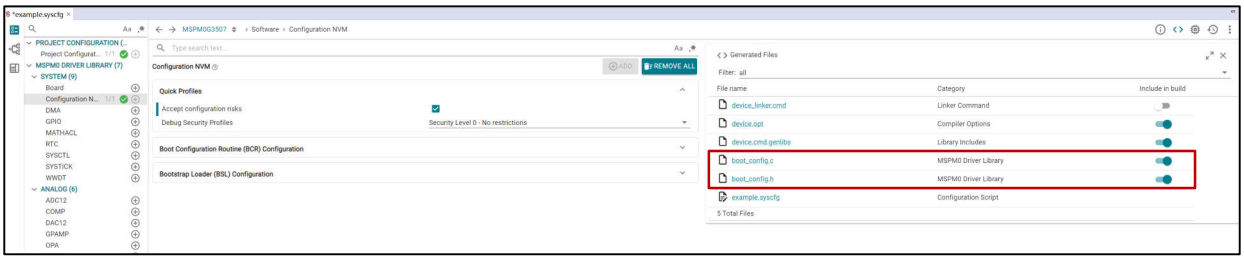


Figure 6-1. Configuration Files

The boot_config.c file includes all of the configurations of the NONMAIN structure, and the configurations automatically compile into the output file. The boot_config.c file includes the default value set in the SysConfig tool. User can *not* manually modify these two files, as the SysConfig tool manages and overwrites the files. After users modify the project-specific.syscfg files in the NVM component, these two files regenerate and update to the latest configuration.

To use the SysConfig to perform additional modifications, exclude the two files from build by unchecking the button *include in build*. Then, the SysConfig generated files (boot_config.c and boot_config.h) do not compile so the user can manually add the files into the project and perform additional modification.

6.2 Project Erase Property

The NONMAIN flash region is a nonvolatile memory which requires erase operation before introducing a new write value. To successfully load the NONMAIN region firmware into the device, set the erase property to erase the NONMAIN.

For Code Composer Studio, right-click on the project and open the project property. Figure 6-2 shows the details.

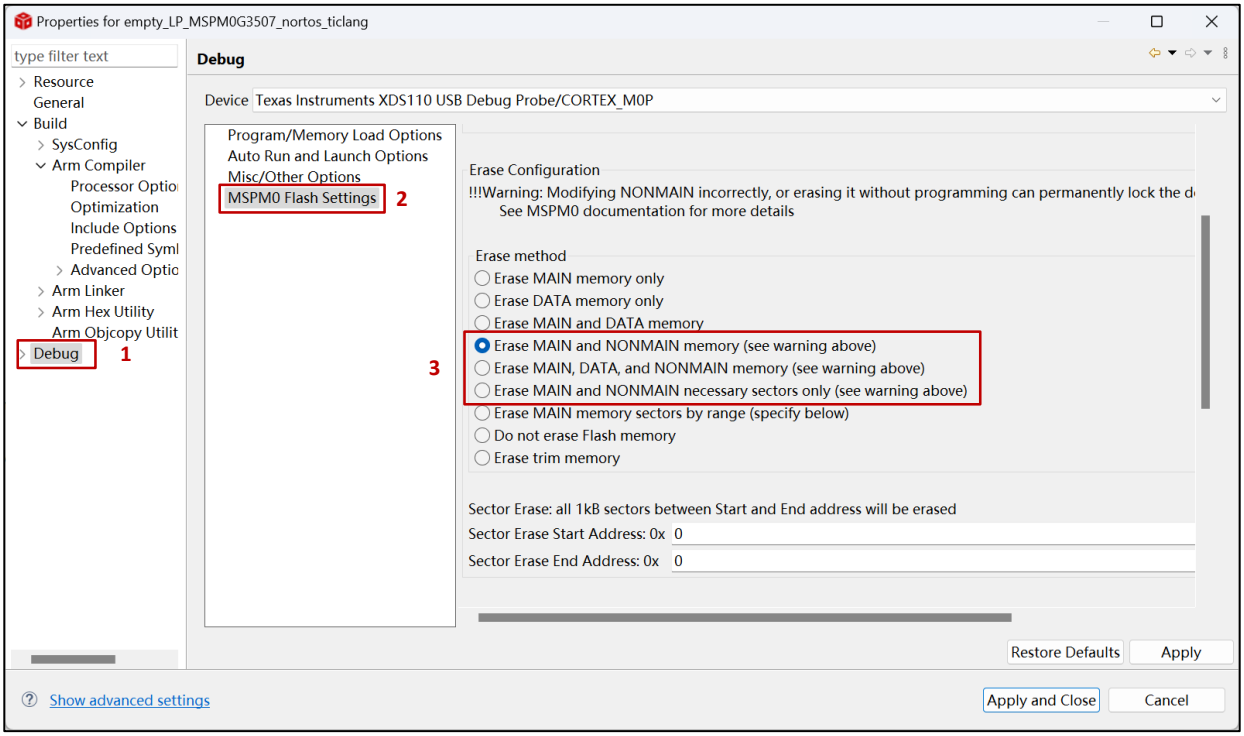


Figure 6-2. CCS Erase Property Settings

For Keil, select and open the options for target, add NONMAIN in debug settings.Figure 6-3 shows the Keli erase property settings.

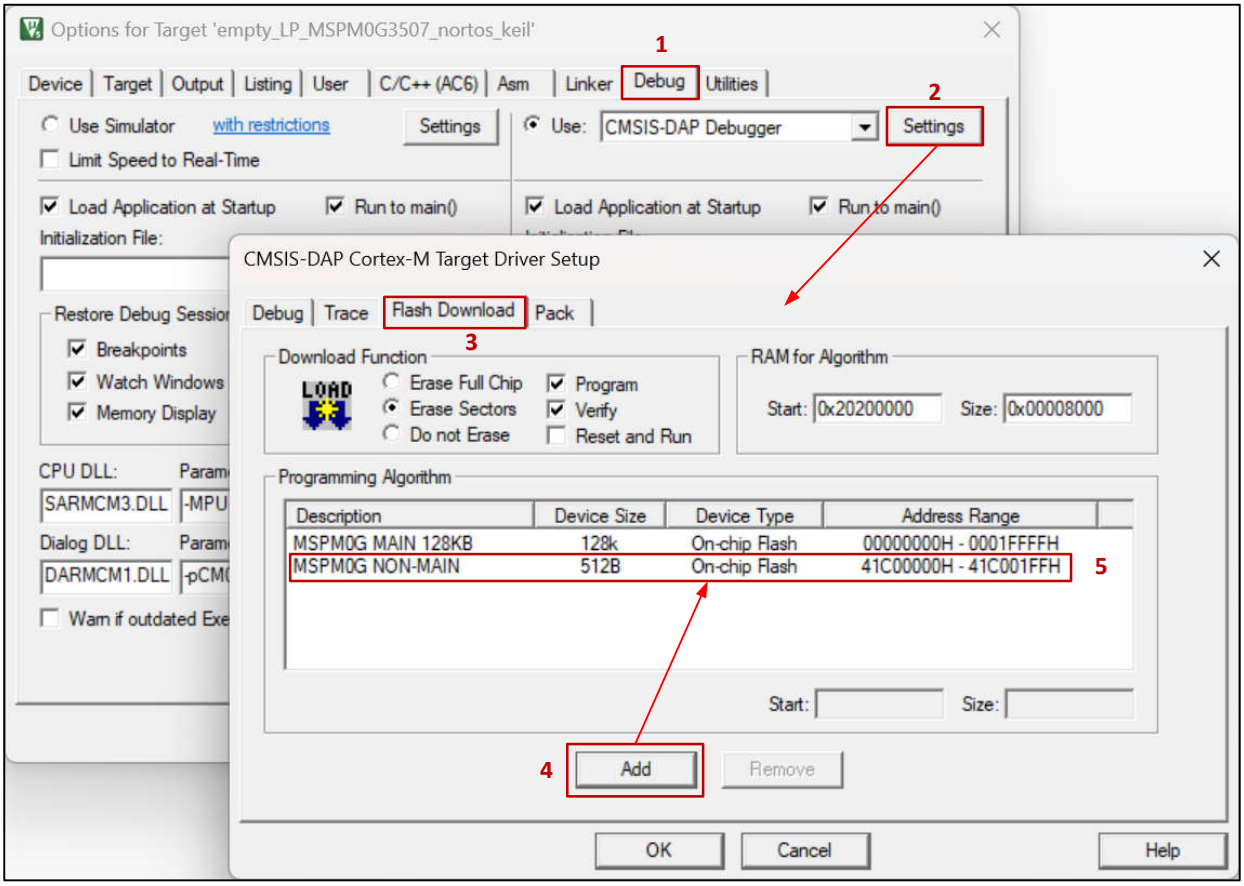


Figure 6-3. Keil Erase Property Settings

For IAR, right-click the project and open the options. Verify that the non-main region is added with an extra parameter. Figure 6-4 shows the IAR settings.

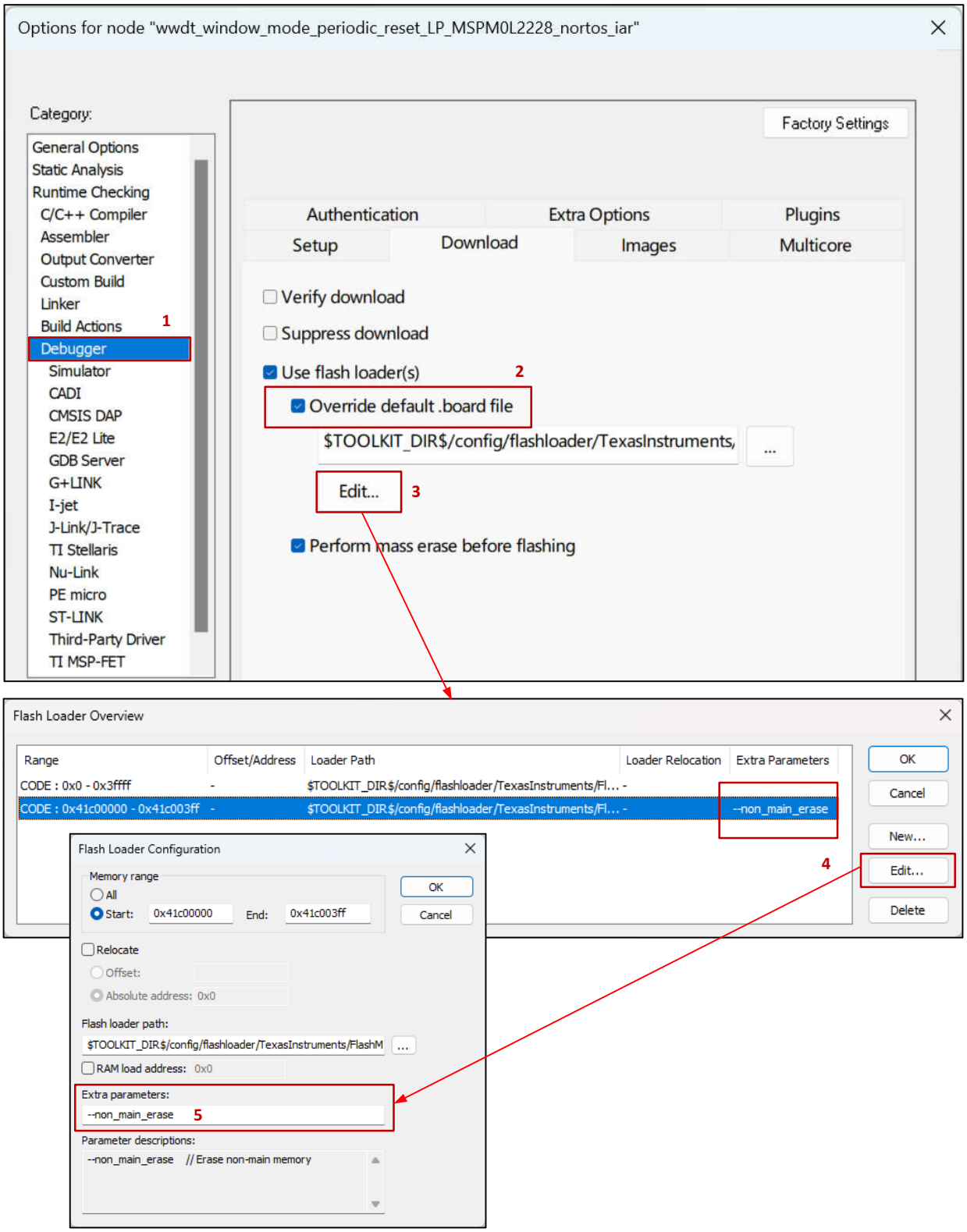


Figure 6-4. IAR Erase Property Settings

After NONMAIN configuration is finalized and programmed, TI recommends removing the NONMAIN field to avoid reprogramming NONMAIN with the same value each time.

6.3 Password-Protected Debug

When SWD debug policy is set as an encryption with a password, generate the DSSM command for SWD password authentication before debugging or downloading new firmware.

TI provides the .ccxml file for the MSPM0 debug configurations with the XDS110 tool. Figure 6-5 shows the steps to set the 128-bit password for following DSSM command:

1. In the *targetConfigs* folder, open the *.ccxml* file.
2. Select *Advanced* to open the XDS110 *Target Configuration*.
3. Select *MSPM0 device* to open the *Device Properties Page*.
4. Input 128-bit password in the SWD Password box.

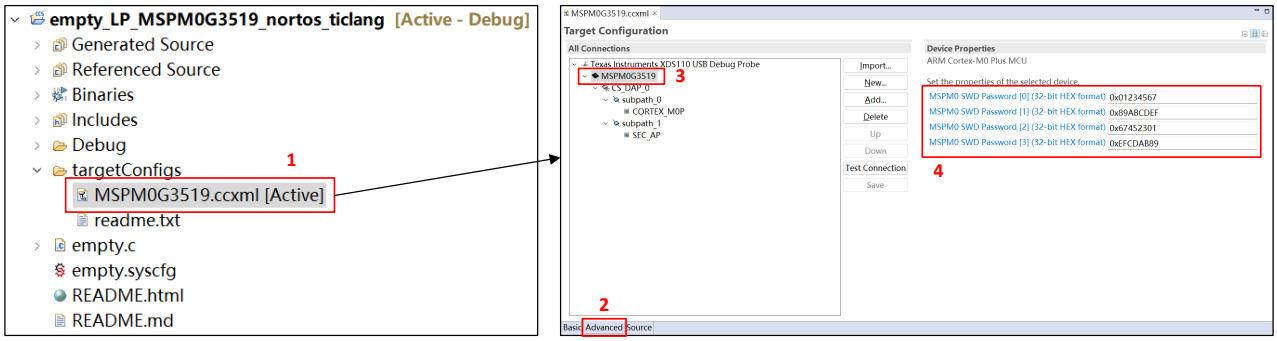


Figure 6-5. Set Password with .ccxml File

Note

The password set in the .ccxml file is applicable for the all DSSM operation passwords, including SWD access, Factory reset, and Mass erase.

After setting the correct password, run the DSSM script to unlock SWD interface. Figure 6-6 shows the steps to run DSSM script with CCS v12:

1. Under *View* in top menu, select *Target Configurations*.
2. Select the *.ccxml* file of the target project. The .ccxml file is set with a password.
3. Select *Launch the selected configuration*.
4. In the debug window, select *XDS110 USB Debug Probe*.
5. In the top menu, select *Scripts*.
6. Select the target *MSPM0 device* to open the command list.
7. Select *DebugAccessPasswordAuthentication_Auto* to generate the DSSM command. The CCS automatically calls the DSSM command and sends the password in the .ccxml file.
8. After the command execution completes and the device unlocks, perfom further debugs with the *Load Program or Load Symbol*.

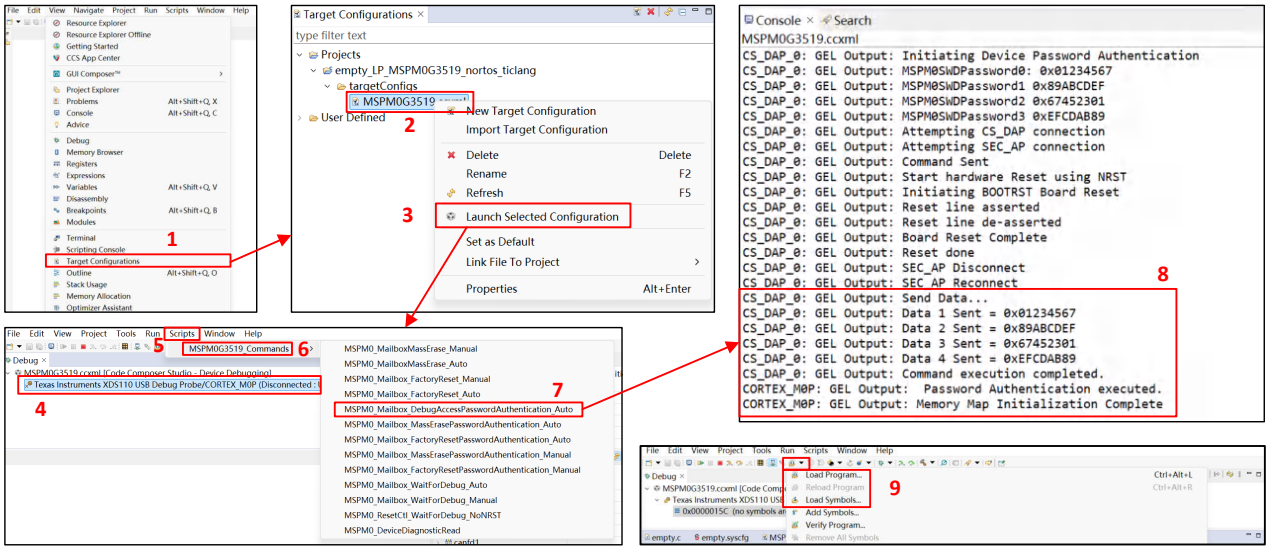


Figure 6-6. DSSM Command with SWD Password Authentication

If the user debugs in the CCS v20, the process for generating the DSSM command is slightly different. Figure 6-7 shows the process.

1. In the project, locate *targetConfigs* and select the *.ccxml* file.
2. Right-click the .ccxml file and select *Start Project-less Debug*.
3. At the top-menu, select *Scripts*, then select *MSPM0 Device Commands*.
4. Select and generate the DSSM command. Select *DebugAccessPasswordAuthentication_Auto* to unlock the SWD interface.
5. After successfully completing the command execution, locate the top menu. Select *Run* to *Debug Project* or *Flash Project*.

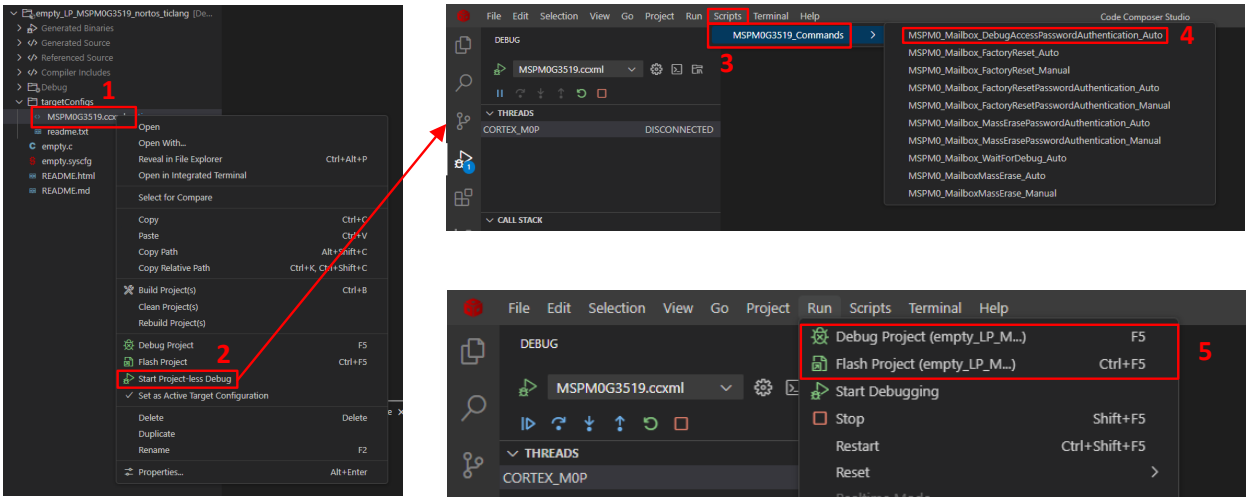


Figure 6-7. DSSM Command with CCS v20

Note

After successfully executing the password authentication sequence for debug access, the device remains in an unlocked state until the next BOOTRST.

7 NONMAIN Operation with Programmer Tool

The NONMAIN is a dedicated region of flash memory which stores the configuration data used by the BCR and BSL to boot the device. The region is not used for any other purpose. To change any parameter in the boot configuration, it is necessary to erase the entire NONMAIN sector and re-program both the BCR and BSL configuration structures with the desired settings.

Regarding as the risk that empty NONMAIN field permanently lock the device, TI recommend user to reduce the time when the NONMAIN is empty and wait for new data loading. Below two kinds of approach are preferred:

- Separately erase NONMAIN and load NONMAIN immediately.
- Erase MAIN, NONMAIN, and write NONMAIN, MIAN in sequence.
  - If the programmer can NOT verify the order of erase and program, then load the MIAN and NONMAIN firmware separately.

7.1 NONMAIN Operation with UniFlash

UniFlash is a software tool for programming on-chip flash on TI microcontrollers and wireless connectivity devices, and on-board flashes for TI processors. UniFlash provides both graphical and command-line interfaces.

To program NONMAIN flash memory, select the erase property which includes NONMAIN memory. Figure 7-1 shows this process.

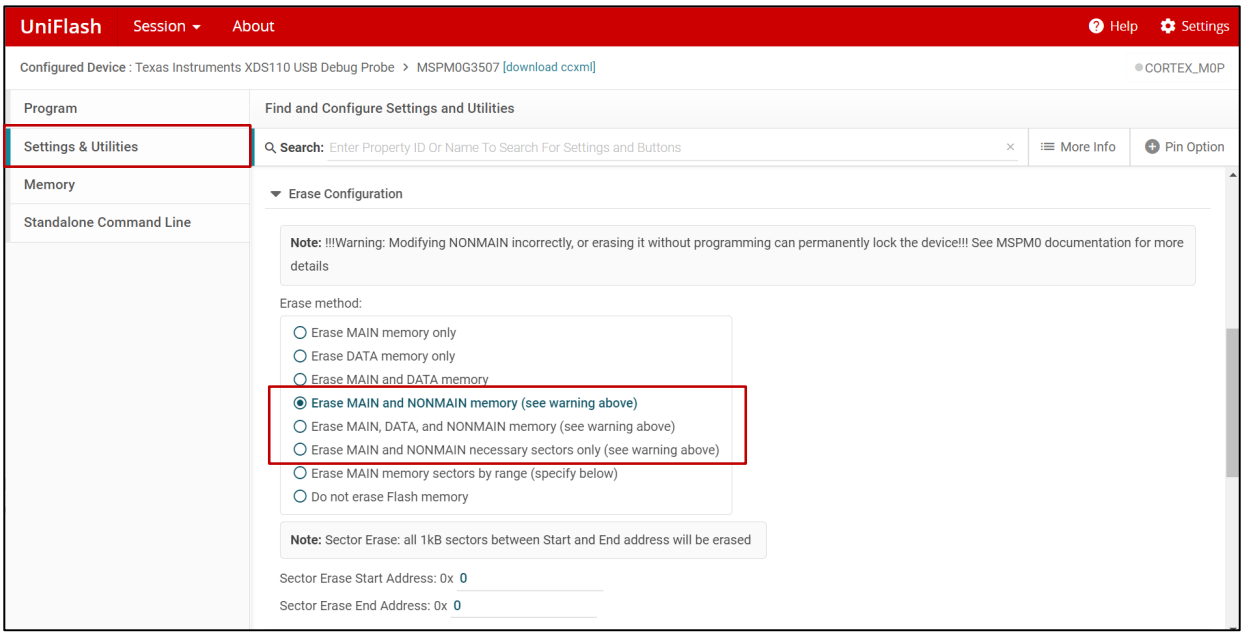


Figure 7-1. NONMAIN Operation with UniFlash

To program the NONMAIN only, follow the below steps:

1. Prepare the firmware to only include the NONMAIN region firmware. The TI-Hex or Intel-Hex format easily processes to get the separate NONMAIN region firmware.
2. Select the erase property as *Erase MAIN and NONMAIN necessary sectors (see warning above)*.
3. Program the device. Only the NONMAIN is erased and reprogrammed.

7.2 NONMAIN Operation with J-Flash

J-Flash is a flash programming software by SEGGER that allows users to program the internal and external flash memory of embedded microcontrollers (MCUs). Control J-Flash using a GUI (J-Flash) or a command line interface.

To program NONMAIN field, enable the NONMAIN bank and select the erase property to include NONMAIN field. Figure 7-2 shows the NONMAIN operation with J-Flash.

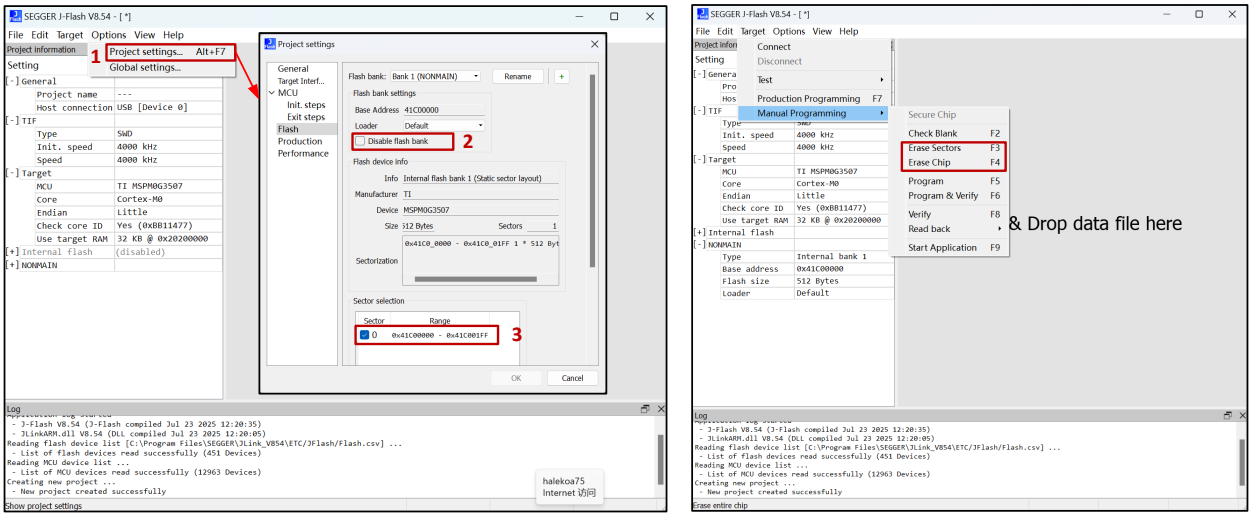


Figure 7-2. NONMAIN Operation with J-Flash

If selecting the manual programming method, make sure the device is not reset before loading the new NONMAIN firmware. If the device resets, the empty NONMAIN permanently locks the device.

Note

TI recommends using the latest J-Flash version for the new MSPM0 device.

To program the NONMAIN only, follow the below instructions:

1. Prepare the firmware, only including the NONMAIN region firmware.
2. Select the erase property to exclude the MAIN flash memory: either disable the MAIN flash or select none of the MAIN flash sectors.
3. Program the device with *erase sectors*. Only the NONMAIN field erases and reprograms.

7.3 NONMAIN Operation with C-GANG

The C-GANG is a low-cost gang programmer that programs up to six identical targets at once. C-GANG is compatible with a variety of TI microcontrollers, including the MSPM0 and MSP430 devices.

C-GANG supports programming the NONMAIN field, follow the below steps:

1. Select *Setup*. Select *Memory* to open the NONMAIN configurations window
2. Directly modify the NONMAIN configurations in C-GANG GUI (left side), or import configurations from the code file (right side)
3. In Memory Protection, select *Enable*.
4. Select *Secure / Protect* to separately erase and program the NONMAIN field.
5. Select *AUTO PROG* to sequentially erase MAIN and program the MAIN field. Erase and program the NONMAIN field.

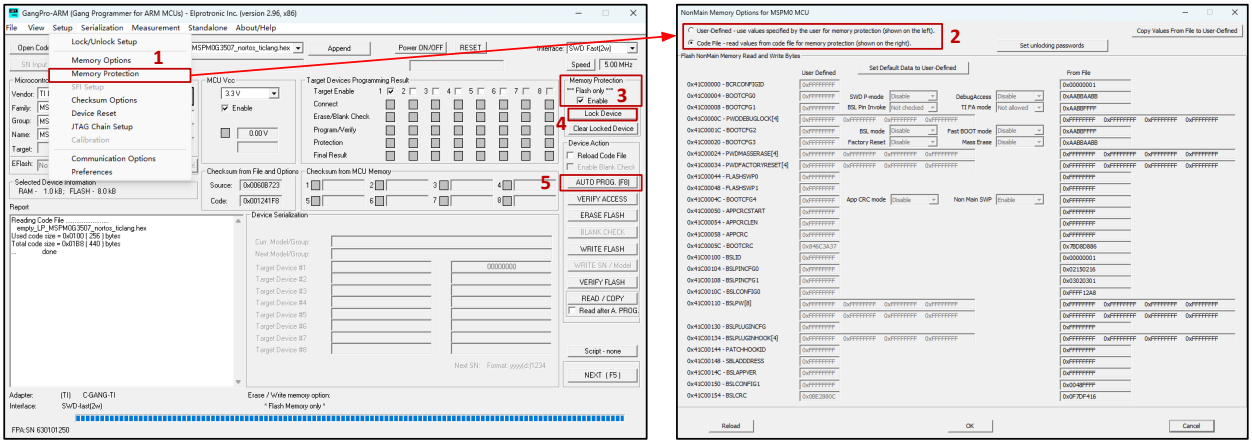


Figure 7-3. NONMAIN Operation with C-GANG

7.4 NONMAIN Operation with MSP-GANG

The MSP-Gang Programmer (MSP-GANG) is a MSPM0/MSP430™/MSP432™ integrated circuit device programmer that programs up to eight identical MSPM0/MSP430/MSP432 Flash or FRAM devices at the same time. The MSP-Gang programmer connects to a host PC using a standard RS-232 or USB connection and provides flexible programming options that allow the user to fully customize the process.

Note

MSP-GANG is available in limited quantities and does *not* support newer MSPM0 devices. Please refer to the C-GANG for new, fast, gang-programming options.

MSP-GANG support programming in the NONMAIN field, follow below steps:

1. Locate the Memory option and select *All Memory*.
2. Select *Secure / Protection Option* to open the NONMAIN configurations window.
3. Select *NONMAIN Memory Write Enable*.
4. Modify the NONMAIN configurations directly in the MSP-GANG GUI (left side) or import configurations from the code file (right side).
5. To erase and program the NONMAIN field, select *Secure / Protect*.

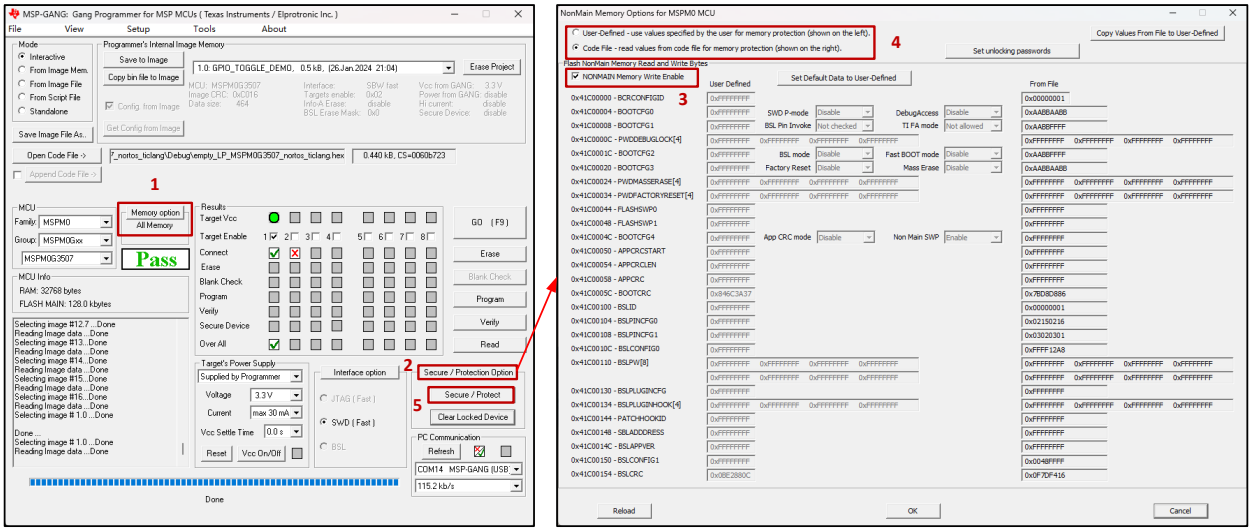


Figure 7-4. NONMAIN Operation with MSP-GANG

8 Frequently Asked Questions (FAQs)

8.1 MCU Locked State Analysis

If MSPM0 enters the locked state, the MSPM0 may *not* connect through the SWD interface. This section introduces the common reasons why the MCU is locked.

8.1.1 Hardware Issue Analysis

The following hardware factors can cause device lockup:

- Defects in the hardware circuit design
- Problem connecting to the debugger
- Periodic external reset signal (to NRST)

8.1.1.1 Hardware Circuit Design

Verify that the MSPM0 has the appropriate external capacitor in few power pins to run in a stable manner. Check the circuit connected to the VDD, Vcore (if supported), and NRST pin and the pin voltage.

Find reference circuits in the *Basic Application Schematic* section of device-specific data sheet. Expect that the Vcore voltage (if supported) is 1.35V (typical), representing that the internal PMU circuit for the digital core functions normally.

8.1.1.2 Debugger Connection

If debugging the MSPM0 device with a standalone debugger, check the hardware pin assignment. To successfully connect the device, verify that SWDIO and SWCLK are well-connected in correct order. Sometimes, the debugger does *not* power the device by default; check that the device is properly powered using the onboard or external power rail.

If debugging MSPM0 in a noisy environment, try a lower debugging rate, or add a small capacitor in parallel to the SWCLK and SWDIO pin.

8.1.1.3 External Reset Signal

If an external reset signal connects to the NRST line, disable the reset signal or keep the signal high when debugging the MSPM0 device. The reset signal (low active) generates to the NRST pin breaks the SWD connection. If a continuously reset signal is in place, the device can enter into a locked state.

8.1.2 Software Issue Analysis

The following software factors can cause device lockup:

- CPU enters a fault state
- BCR configuration
- Low power mode (STOP or STANDBY)
- SHUTDOWN IO state
- SWD IO (SWDLCK or SWDIO) function disabled
- WDT or IWDT reset (if enabled in application code)
- Software POR or BOOTRST

8.1.2.1 CPU Enters a Fault State

When CPU enters a fault state, the ARM device automatically performs a continual self-reset until the CPU leaves the fault state. The CPU enters the fault state during a nested exception (for example, double-hard fault or NMI), caused by illegal CPU activities, such as invalid address modification or peripheral misconfiguration.

When CPU enters a fault state, the continuous reset behavior breaks the SWD connection and device enters the locked state.

8.1.2.2 BCR Configuration

When BCR configuration is set to SW-DP disabled or SWD-access disabled, the device enters the locked state in debug mode. When BCR configuration is set to SWD-access encrypted, the device enters the locked state in debug mode if incorrect passwords are loaded.

Additionally, loading the incorrect NONMAIN configuration can result in device entering a permanently locked state.

8.1.2.3 Low Power Mode (STOP or STANDBY)

When the device enters STOP or STANDBY mode, the AHB-AP (ARM Debug) core is undiscoverable, making the device enter a locked state. To actively connect to the AHB-AP, verify that the debugger first connects the PWR-AP. For more details, refer to the *Hardware Programming and Debugger Guide for MSPM0* application note.

XDS110 support connecting device in STOP or STANDBY mode. For other 3rd party debuggers, such as J-Link, try the latest software version which fixes this low-power mode debug issue.

8.1.2.4 SHUTDOWN IO State

The digital IO pin states latch and retain upon entry to SHUTDOWN. The digital IO pin states include:

- Output low or high
- Pullup or pulldown
- Input or output enabled
- Drive configuration

After exiting SHUTDOWN mode, the IOs are held in the previous state until released by application software setting the RELEASE bit in the SHDNIOREL register along with the matching KEY value.

When exiting SHUTDOWN, the serial wire debug (SWD) pins also remain locked until application software sets the RELEASE bit. As a result, a debug connection cannot establish when waking up from SHUTDOWN mode until the IO are released by application software.

TI debug tool - XDS110, right now supports directly debug the device wake up from SHUTDOWN mode.

8.1.2.5 SWD IO Function

After a POR, when the boot process completes and the CPU starts the application, serial wire debug (SWD) IOs are configured in SWD mode (SWDIO is pulled high, and SWCLK is pulled low).

It is possible to re-configure the SWD pins as general purpose IO (GPIO) in software to enable use of these pins in an application when debug support is no longer required. To disable SWD functionality, set the DISABLE bit in the SWDCFG register in SYSCTL along with the KEY (62h).

Once the SWD pin functions are disabled, the device remains in the lock state, and the SWD pin functions can only be re-enabled by triggering a POR.

8.1.2.6 WDT or IWDT Reset

When the WDT or IWDT is not fed within the correct time window, a reset occurs and can cause the device to enter a locked state.

Normally, the debugger generate SYSRST through the SYSCTL to disable the WDT. If the reset does not generate,or is set as the CPU level reset, the WDT can cause the device to enter a locked state and assert a reset during the SWD connection stage.

For MSPM0 devices containing the IWDT peripheral, the debugger generates LFSS reset or IWDT reset to disable IWDT. Otherwise, the device can enter a locke state, as a SYSRST does not reset the LFSS component, including IWDT.

8.1.2.7 Software POR or BOOTRST

A software POR or BOOTRST breaks the SWD connection and can cause the device to enter a locked state if the break occurs when establishing the SWD connection.

Disable software POR and BOOTRST to stably debug the MSPM0 device.

8.2 Unlock the MSPM0 Device

This section describes unlocking the device when debug-access fails due to a software issue.

8.2.1 Force MCU to Enter BSL Mode

For the device supports BSL and BSL is set as enabled (default enabled), use the hardware invoke method to force the MCU to enter BSL mode. The MCU then executes the BSL code in the ROM and does not execute the application code. The device remains in SWD connection in BSL mode.

Below is the methods to force the MCU to enter BSL mode:

- Connect the BSL invoke pin (default is PA18) to the VCC (with or without pull up resistor). Re-power the device.
- Connect the BSL invoke pin (default is PA18) to the VCC (with or without pull up resistor). Force NRST low for more than 1s to trigger POR.

Note

When the MCU enter BSL mode, the MCU enters STANDBY0 mode if no BSL connection command is sent to UART or I2C interface within 10s.

This approach help users unlock the device if a software issue causes the MCU enter a locked state. The approach is not viable in the following situations:

- NONMAIN configuration has CRC check failure.
- NONMAIN configuration disables the SWD interface or locks the debug access.
- IWDT is enabled through the device supporting VBAT feature, requiring user to manually turn off the VBAT to disable the IWDT.
- The BSL hardware invoke method is disabled.

8.2.2 Send BSL Command

If the NONMAIN configuration disables the SWD interface or locks the debug access, send the BSL command to unlock the device after BSL mode entry.

BSL Communication Interface (see Section 3.2.3) is applied to send BSL command. The ROM BSL core supports the Factory Reset command to restore the default NONMAIN configurations. Refer to *MSPM0 Bootloader* user's guide for more details.

The approach is not viable in the following situations:

- BCR configuration sets Factory Reset as disabled.
- BCR configuration sets NONMAIN Static Write Protection as enabled.
- BSL is disabled in the BCR configuration.

8.2.3 Generate DSSM Command

To reset functionality, the MSPM0 supports the DSSM command, including Mass Erase and Factory Reset. Refer to Section 6.3, it takes the DebugAccessPasswordAuthentication_Auto command as an example to introduce how to demonstrate a DSSM command using the CCS tool. Other DSSM commands (Factory Reset or Mass Erase) have the same approach, including setting 4 32-bit passwords in the same location of the .ccxml file.

Below is the command list the CCS tool supports:

- DebugAccessPasswordAuthentication_Auto
- FactoryReset_Auto
- FactoryReset_Manual
- FactoryResetPasswordAuthentication_Auto
- FactoryResetPasswordAuthentication_Manual
- MassErase_Auto

- MassErase_Manual
- MassErasePasswordAuthentication_Auto
- MassErasePasswordAuthentication_Manual

TI recommends using FactoryReset_Auto to unlock the device. The FactoryReset_Auto command can generate the reset signal in the NRST pin and restore the default NONMAIN configurations.

In cases where software issues break the factory reset process (normally caused by software reset), select FactoryReset_Manual to unlock the device by following the below steps:

1. Follow the steps show in Section 6.3 to launch the configuration files (CCS v12), or start a project-less debug (CCS v20).
2. Keep NRST line in low state (connect to GND).
3. Generate the DSSM command with FactoryReset_Manual (NRST line keep low).
4. When *Press the reset button* appears in the CCS console window, release the NRST line (disconnect from GND).
5. The Factory Reset command is executed.

If the FactoryReset_Manual unlock device fails, further force MCU into BSL mode with BSL invoke pin connecting to VCC (if BSL mode exists and is enabled) and generate the FactoryReset_Auto command. During the command execution, keep BSL invoke pin (default is PA18) connected to VCC to avoid the device run application code.

This approach is not viable if:

- The NONMAIN configurations disable the SWD Factory Reset command.
- IWDT enables with a device-supporting VBAT feature, requiring the user to manually turn off the VBAT to disable the IWDT.

See section 7.1 (*Unlock MCU*) of the *MSPM0 MCUs Development Guide* user's guide to generate a Factory Reset using different tools.

8.3 Debug Error Overview

This section describes common debug errors found when using CCS tool and details possible recovery methods.

8.3.1 No Error Code: DAP Connection Error

Possible root causes:

- SWD connection issues
- Broken MSPM0 device
- CPU enters a fault state
- A software (POR or BOOTRST) or hardware reset occurs during connection

Possible recovery methods:

- Check SWD hardware connection
- Check that the VCC, NRST, and Vcore (if supported) are within the data sheet specifications
- Force NRST low, and reconnect the device. If DAP connection error does not appear, try methods detailed in Section 8.2

8.3.2 No Error Code: Connection to MSPM0 Core Failed

Possible root causes:

- NONMAIN configurations disables or encrypts the SWD interface
- CPU enters a fault state
- A software (POR or BOOTRST) or hardware reset occurs during connection

Possible recovery methods:

- Verify that no low pulse occurring on the NRST line
- Generate SWD access password authentication by DSSM if encrypts the SWD interface

- Follow steps listed in Section 8.2

8.3.3 Error - 6305: PRSC Module Failed to Write a Routine Register

Refer to Section 8.3.2.

8.3.4 Error - 260: An Attempt to Connect to the XDS110 Failed

Possible root causes:

- XDS110 is not connected or occupied.
- Invalid XDS110 firmware, invalid XDS110 serial number, and so forth.

Possible recovery methods:

- Repower the XDS110 and check the USB connection.
- Reset the XDS110 firmware.

8.3.5 Error - 261: Invalid Response From the XDS110

Possible root causes:

- XDS110 debugger enters a fault state.
- CPU enters a fault state.

Possible recovery methods:

- Repower the XDS110 debugger.
- Follow the instructions in Section 8.2.

8.3.6 Error - 615: Target Fails to Identify a Correctly Formatted SWD Header

Refer to Section 8.3.5.

8.3.7 Error - 1001: Requested Operation is not Supported on This Device

Possible root causes:

- A software or hardware reset occurs.
- CPU enters a fault state.

Possible recovery methods:

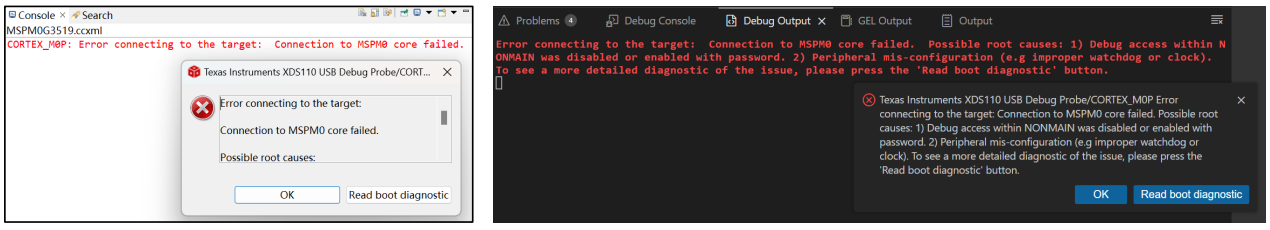
- Check that there is no low pulse occurring on the NRST line
- If the device enter a locked state, follow the instructions in Section 8.2.

8.3.8 Error - 2131: Unable to Access Device Register

Refer to Section 8.3.7.

8.4 MSPM0 Boot Diagnostic

When the device enters into the locked state and the DAP remains connected, the CCS tool provides the method to read the boot diagnostic of MSPM0 device, as shown in Figure 8-1.



CCS Version 12.x CCS Version 20.x

Figure 8-1. Read Boot Diagnostic Using the CCS Tool

Listed are common boot diagnostic read values.

Table 8-1. Common Boot Diagnostic Read Values

Device Diagnostic Read	Detailed Description
0x0000.0000	Boot failed. Possible reason is that the NRST line is held low or the clock supply is invalid.
0x0000.0007	Boot success. The application code is executed and leads the device to a lockup state.
0x0001.0136	Boot failed. Possible reason is that the NONMAIN CRC checksum verification fails.
0x0004.110A	Boot success. BSL is invoked and executed.
0x0007.0000	Boot failed. NMI error is triggered in boot code.

9 Summary

This user's guide provides an overview of the NONMAIN configuration among MSPM0 family and step-by-step instructions to configure the NONMAIN register. In addition to basic knowledge, the document also lists references and further reading materials for users.

10 References

- Texas Instruments, *MSPM0 C-Series 24MHz Microcontrollers* technical reference manual
- Texas Instruments, *MSPM0 G-Series 80MHz Microcontrollers* technical reference manual
- Texas Instruments, *MSPM0 H-Series 32MHz Microcontrollers* technical reference manual
- Texas Instruments, *MSPM0 L-Series 32MHz Microcontrollers* technical reference manual
- Texas Instruments, *MSPM0 MCUs Development Guide* user's guide
- Texas Instruments, *MSPM0 Bootloader* user's guide
- Texas Instruments, *MSPM0 Bootloader Implementation* application note
- Texas Instruments, *Flash Multi Bank Feature in MSPM0 Family* application note
- Texas Instruments, *Cybersecurity Enablers in MSPM0 MCUs* application note
- Texas Instruments, *Hardware Programming and Debugger Guide for MSPM0* application note
- Texas Instruments, *Secure Booting User's Guide*
- Texas Instruments, *MSPM0 SDK QuickStart Guide for Code Composer Studio v12 (Eclipse)*
- Texas Instruments, *QuickStart Guide for Code Composer Studio v20*
- Texas Instruments, *MSPM0 SDK QuickStart Guide for IAR*
- Texas Instruments, *MSPM0 SDK QuickStart Guide for Keil*
- *Texas Instruments, UniFlash flash programming tool*
- *Texas Instruments, C-GANG design and development tool*
- *Texas Instruments, MSP-GANG design and development tool*
- *SEGGER Microcontroller GmbH, J-Flash Program internal & external microcontroller flash tool*
- *Emn178.Github.io, CRC32 and SHA256 online tool*

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), 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 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, TI’s General Quality Guidelines, 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. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

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

Last updated 10/2025