Application Note

Enable TrustZone® for Secure Application in MSPM33 MCUs

Yuhao Zhao, Siddharth Bhat

Abstract

MSPM33 microcontrollers support Arm® TrustZone® to enable secure and nonsecure attributes on device resource for secure application. This document describes how to configure TrustZone® and global security controllers to definite secure and non-secure regions, how to develop TrustZone® based application to handle secure resources and non-secure resources in the code, and how to evaluate secure applications and non-secure applications together with TI tools.

Table of Contents

1 Introduction 2
1.1 Security Concept 2
1.2 Key Concepts 3
2 TrustZone Architecture 4
2.1 MSPM33 Family Overview 4
2.2 TrustZone Feature 4
2.3 TrustZone® Software Development 11
3 Develop TrustZone Based Application 12
3.1 Device Configuration 12
3.2 Develop Secure Application 13
3.3 Develop Non-Secure Application 28
4 Evaluate TrustZone Based Application 32
4.1 Environment Setup 32
4.2 Load Secure and Non-secure Images Together 32
4.3 Load Secure and Non-secure Images separately 34
4.4 Debug and Evaluate the Example 37
5 FAQ 39
5.1 How to Modify the Range of S/NS SRAM 39
5.2 How to Modify the Range of S/NS FLASH 40
5.3 How to Modify the Security Attribute on Peripherals 41
6 Summary 43
7 References 43

Trademarks

Arm® and TrustZone® are registered trademarks of Arm Limited.
All trademarks are the property of their respective owners.

Copyright © 2026 Texas Instruments Incorporated

1 Introduction

In the MSPM33 family, some series support TrustZone security extension for ARMV8-M. This enables programmers to restrict sections of memory from unauthorized users, and provides a hardware basis for implementing Trusted Execution Environments (TEEs). Some series also support global security controller (GSC) to enable additional security layers to configure the secure and privilege attribute for resources on the device.

1.1 Security Concept

In Armv8-M architecture with TrustZone, the processor has secure and non-secure states:

- Non-secure processor can only access non-secure resources.
- Secure processor can access both Non-secure and secure resources.

This enables both thread mode and handler mode to split to secure states, as shown in Figure 1-1. User can allocate code to one of four modes according to the need. Application threads can transition between secure and non-secure states at a function call or an interrupt call.

The division of secure and non-secure regions is based on a memory map, which can be configured by the combination of the Security Attribution Unit (SAU) and the Implementation Defined Attribution Unit (IDAU). This division enables the secure and non-secure partitioning on both system and software. Firstly, as shown in Table 2-2 nearly all MCU systems can be split to secure and non-secure by address. Secondly, the security of the memory also indicates which security state the code must execute in, and code is then allocated to either the Secure or Non-secure states.

Each bus transaction carries its security attribute and propagates the security state to all security aware bus slaves (peripherals, memory, and so on), so that they can enforce access control at a system level, rather than just the CPU.

For more information, see TrustZone technology for Armv8-M Architecture documentation or MSPM33C Security User's Guide.

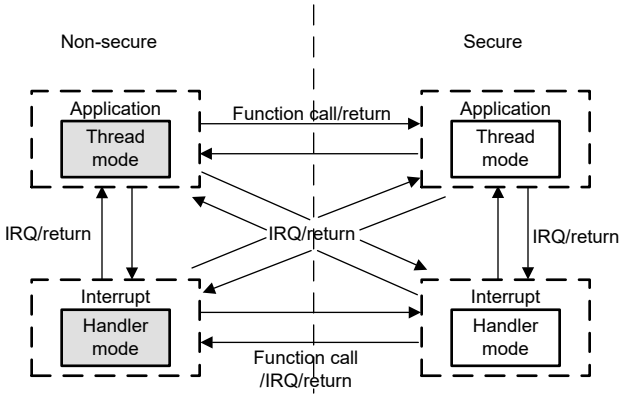


Figure 1-1. Processor Secure and Non-Secure State

Copyright © 2026 Texas Instruments Incorporated

1.2 Key Concepts

TrustZone®	TrustZone® is a security extension developed by ARM that creates two distinct execution environments in MCU: secure world and normal world.
Security Attribution Unit (SAU)	SAU is an internal, programmable security unit within an Armv8-M processor that allows software to define the security attributes of memory regions at runtime.
Implementation Defined Attribution Unit (IDAU)	IDAU is a hardware-fixed unit implemented by the chip vendor that determines the default security attributes of memory and peripherals.
Global Security Controller (GSC)	A system module from TI used to configure the security firewalls for secure and non-secure context of the application.
NONMAIN	A dedicated flash memory region which configures device boot related parameters. See *MSPM33 NONMAIN Flash Memory Configuration Guide* for the NONMAIN configuration.

Copyright © 2026 Texas Instruments Incorporated

2 TrustZone Architecture

2.1 MSPM33 Family Overview

The MSPM33 family has different series, and some series support different features for security usage.

Table 2-1 shows supported feature status for different devices. For more information, see the security user guide.

For MSPM33, in general security attribution is based on the configuration of both TrustZone and GSC. TrustZone is ARM-based security extension for ARM-Cortex-M33, while GSC is the security peripheral provided by TI.

Table 2-1. TrustZone® and GSC features on MSPM33 Family

Feature			MSPM33C32xx
TrustZone	Security Attribution unit (SAU)		√
	Implementation Defined Attribution Unit (IDAU)		√
Global Security Controller	SRAM Protection Controller (SPC)	Privilege Access Configure	√
		Secure Access Configure	√
	Flash Protection Controller (FPC)	Privilege Access Configure	√
		Secure Access Configure	√
		Write-Erase Protect	√
		Flash Hide Protection	√
	Peripheral Protection Controller (PPC)	Privilege Access Configure	√
		Secure Access Configure	√
	Secure Vector Table Offset Register(VTOR_S)		√
	Non-Secure Vector Table Offset Register(VTOR_NS)		√

2.2 TrustZone Feature

In TrustZone architecture, MCU resources are divided into a secure region and a non-secure region.

In general, there are two applications to handle different regions, one secure application and one non-secure application. Code is allocated to either the secure or non-secure states based on the memory security attributes. Hardware logic present in the TrustZone-enabled AMBA3 AXI bus fabric ensures that no secure world resources can be accessed by the normal world components.

2.2.1 Security States – Secure, Non-secure, Non-secure Callable

Typically, security attributes of a memory region are defined by ARMv8-M.

- Secure (S)
- Non-secure (NS)
- Non-secure Callable (NSC)

Secure addresses are used for memory and peripherals that are only accessible by Secure software or Secure masters. Non-secure addresses are used for memory and peripherals accessible by all software running on the device. NSC is a special type of Secure location. This type of memory is the only type which an Armv8-M processor permits to hold an SG instruction that enables software to transition from Non-secure to Secure state. The inclusion of NSC memory locations removes the need for Secure software creators to allow for the accidental inclusion of SG instructions, or data sharing encoding values, in normal Secure memory by restricting the functionality of the SG instruction to NSC memory only. For more information, please refer to TrustZone® technology for Armv8-M Architecture documentation.

In general, processor security states is determined by code address.

- If code being executed locates in non-secure region, then processor is in non-secure state.
- If code being executed locates in secure region or NSC region, then processor is in secure state.

Copyright © 2026 Texas Instruments Incorporated

When access data, if processor is in secure state, it can access data in Secure, NSC and Non-secure region. If processor is in non-secure state, it can only access data in non-secure.

When fetching an instruction, if processor is in secure state, it can fetch instruction from Secure region and NSC region. If processor is in non-secure state, it can fetch instruction from Non-secure and NSC region.

As shown in Figure 2-1, the processor can switch between security states at a function call boundary or by taking an exception that targets a different security state. Please find specific instructions in the Section 3.2.4, or find the software flow of secure and non-secure examples in Figure 3-13.

Secure gateway veneers decouple the addresses of Secure gateways (in NSC regions) from the rest of the Secure code. By maintaining a vector of Secure gateway veneers at a forever-fixed address, the rest of the Secure code can be updated independently of Non-secure code. This also limits the amount of code in NSC regions that potentially can be called by the Non-secure state.

Note

Processor states can be transitioned only by using the special instructions in the Section 3.2.4. An application call between S region and NS region without corresponded instruction will lead to related exception.

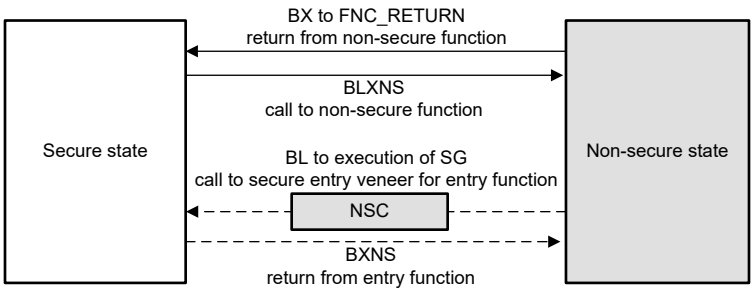


Figure 2-1. Transition between Secure State and Non-secure State

After power on, MSPM33 family will first execute TI ROM boot-code, which is immutable root of trust. CPU will be in secure and privilege in default. After successful boot, boot code issues BOOTDONE. At this point, SYSCTL issues a SYSRST to the device to trigger execution from MAIN flash memory. For HSFS state device, a MAIN flash program will always start from address 0x10000004 vector (Reset Handler, defined by VTOR_S) after boot code is finished, which is secure based on SAU setting.

Note

After BOOTRST(or higher level reset), both SAU and GSC will be reset. The whole FLASH and the whole SRAM will be secure, which means the application locates in 0x10000000 will be regarded as secure and initial processor state will be secure at well.

2.2.2 First Security Attribute Define – SAU and IDAU

The Cortex-M33 Security is provided by TrustZone® architecture as part of the Secure Extensions in ARMv8-m. Two key components of the security architecture are the Implementation Dependent Attribution Unit (IDAU) and the Secure Attribution Unit (SAU). Security state of a memory region is controlled by a combination of the internal Secure Attribution Unit (SAU) or an external Implementation Defined Attribution Unit (IDAU).

The CPU uses the highest setting of SAU and IDAU to provide a secure (S), non-secure callable (NSC) and non-secure (NS) attribute - The resulting attribute of a memory region, from where CPU may execute, is a combination of SAU and IDAU, which is equivalent to highest setting of the two to provide a secure (S), non-secure callable (NSC) and non-secure (NS) attribute. For more information on SAU and IDAU, please refer to MSPM33C Security User's Guide or Technical Reference Manual of specific MSPM33 devices.

Copyright © 2026 Texas Instruments Incorporated

Note

SAU can only be programmed in secure state.

For the bus transaction, AHB5 defines the secure transfer property. An interface that supports secure transfers has an additional signal HNONSEC to indicate whether a transfer is Secure or Non-Secure. Based on this information, bus fabric components can prevent Non-secure transactions from accessing Secure memories. Similarly, peripherals can also be assigned as Secure or Non-Secure. The bus system ensures that Secure peripherals are accessed by only Secure transfers.

Note

Cortex-M33 has two AHB interfaces, S-AHB and C-AHB, each interface would have their own HNONSEC signal.

For instruction fetches, HNONSEC is always determined by the Security state associated with the address.

For data accesses, the external attribute on AHB HNONSEC is based on the associated Security state specified by the SAU/IDAU.

The following table shows the secure configuration on SAU & IDAU. For MSPM33, IDAU is pre-defined by TI with bit 28 of address(0 for NS and 1 for NSC), and SAU is configurable. User can configure up to 8 SAU regions, and in default all addresses are as secure. Please follow the instructions in Section 3.2.2 to correctly configure the SAU.

Table 2-2. SAU & IDAU Configuration and Security Attribution

Region	Address Start	Address End	IDAU	SAU	SAU & IDAU Security Attribute
Non-secure FLASH	0x0000,0000	0x00FF,FFFF	NS	NS	NS
				NSC	NSC
				S(default)	S(default)
Secure FLASH	0x1000,0000	0x10FF,FFFF	NSC	NS	NSC
				NSC	NSC
				S(default)	S(default)
Non-secure SRAM	0x2000,0000	0x2FFF,FFFF	NS	NS	NS
				NSC	NSC
				S(default)	S(default)
Secure SRAM	0x3000,0000	0x3FFF,FFFF	NSC	NS	NSC
				NSC	NSC
				S(default)	S(default)
Non-secure Peripheral	0x4000,0000	0x4FFF,FFFF	NS	NS	NS
				NSC	NSC
				S(default)	S(default)
Secure Peripheral	0x5000,0000	0x5FFF,FFFF	NSC	NS	NSC
				NSC	NSC
				S(default)	S(default)
Non-secure External Memories	0x6000,0000 0x8000,0000 0xA000,0000 0xC000,0000	0x6FFF,FFFF 0x8FFF,FFFF 0xAFFF,FFFF 0xCFFF,FFFF	NS	NS	NS
				NSC	NSC
				S(default)	S(default)
Secure External Memories	0x7000,0000 0x9000,0000 0xB000,0000 0xD000,0000	0x7FFF,FFFF 0x9FFF,FFFF 0xBFFF,FFFF 0xDFFF,FFFF	NSC	NS	NSC
				NSC	NSC
				S(default)	S(default)

Copyright © 2026 Texas Instruments Incorporated

For MSPM33C32, final security attribute should consider GSC controller as well, which is described in Section 2.2.3.

Note

Processor states will only depend on memory security attribute base on SAU & IDAU Security Attribute, and GSC configuration on FLASH will not affect code security states or processor security states, because SAU & IDAU is inside CPUSS, while GSC is a security peripheral monitoring bus transactions.

For the accesses in MCU, in general there are multiple initiators, like CPU and DMA, but SAU & IDAU is applicable to CPU only. That's one reason to add global security controller(GSC) in the peripheral side for other initiators(like DMA)'s security attributes.

GSC also add second layer for security attributes on SRAM, FLASH, peripherals, enabling user to flexibly partition secure and non-secure regions.

2.2.3 Second Security Attribute Define – Global Security Controller

Global Security Controller (GSC) configures the secure and privilege attribute for resources on the device, which consists of multiple blocks for SRAM, FLASH and peripherals. For more information on GSC, please refer to MSPM33C Security User's Guide or Technical Reference Manual of MSPM33 devices.

The following table shows the secure and privilege configuration in GSC. For MSPM33, GSC is user-defined and can be configured during boot or runtime. Please follow the instructions in Section 3.2.3 to correctly configure the GSC.

Table 2-3. Secure and Privilege Configurations in GSC

	Register	Peripheral	Default Secure Access	Default Privilege Access
SRAM	SPC_SECATTRIB0(0-64KB) SPC_SECATTRIB1(64-128KB) SPC_SECATTRIB2(128-192KB) SPC_SECATTRIB3(192KB-256KB) SPC_PRIVATTRIB0(0-64KB) SPC_PRIVATTRIB1(64-128KB) SPC_PRIVATTRIB2(128-192KB) SPC_PRIVATTRIB3(192KB-256KB)	SRAM	Secure	Privilege in every chunk
FLASH	FPC_SECATTRIBA0(0-64KB,BANK0) FPC_SECATTRIBA1(0-64KB,BANK1) FPC_SECATTRIBB0(64KB-512KB, BANK0) FPC_SECATTRIBB1(64KB-512KB, BANK1) FPC_SECATTRIBA_DFLASH(0-32KB, DATA FLASH) FPC_PRIVATTRIBA0(0-64KB,BANK0) FPC_PRIVATTRIBA1(0-64KB,BANK1) FPC_PRIVATTRIBB0(64KB-512KB, BANK0) FPC_PRIVATTRIBB1(64KB-512KB, BANK1) FPC_PRIVATTRIBA_DFLASH(0-32KB, DATA FLASH)	FLASH	Secure	Privilege in every sectors or blocks
Analog	PPC_SECATTRIB_ADC	ADC0	Non-secure	/
		ADC1	Non-secure	/
	PPC_SECATTRIB_ACOMP	COMP0	Non-secure	/
		COMP1	Non-secure	/
	PPC_SECATTRIB_VREF	VREF	Non-secure	/

Copyright © 2026 Texas Instruments Incorporated

Table 2-3. Secure and Privilege Configurations in GSC (continued)

Communication	PPC_SECATTRIB_UNICOMM	UC1_0	Non-secure	/
		UC1_1	Non-secure	/
		UC2	Non-secure	/
		UC15_0	Non-secure	/
		UC15_1	Non-secure	/
		UC12	Non-secure	/
		UC13_0	Non-secure	/
		UC13_1	Non-secure	/
		UC13_2	Non-secure	/
		UC13_3	Non-secure	/
		UC14	Non-secure	/
	PPC_SECATTRIB_I2S	I2S0	Non-secure	/
		I2S1	Non-secure	/
	PPC_SECATTRIB_MCAN	MCAN0	Non-secure	/
		MCAN1	Non-secure	/
	PPC_SECATTRIB_QSPI	QSPI	Non-secure	/
System	PPC_SECATTRIB_GPIO	GPIOA	Non-secure	/
		GPIOB	Non-secure	/
		GPIOC	Non-secure	/
	PPC_SECATTRIB_DMA PPC_PRIVATTRIB_DMA	DMA0	Secure	Privilege
		DMA1	Non-secure	Non-privilege
	PPC_SECATTRIB_SYSCTL PPC_PRIVATTRIB_SYSCTL	SYSCTL	Secure	Privilege
	PPC_SECATTRIB_IOMUX	IOMUX	Non-secure	/
	PPC_SECATTRIB_WUC	WUC	Non-secure	/
	PPC_SECATTRIB_FRIREGS	FRI	Non-secure	/
	PPC_SECATTRIB_MEMCFG	MEMCFG	Non-secure	/
	PPC_SECATTRIB_LFSS	LFSS	Non-secure	/
	PPC_SECATTRIB_EAM	EAM	Non-secure	/
Timers	PPC_SECATTRIB_TIMER	TIMA0_0	Non-secure	/
		TIMA0_1	Non-secure	/
		TIMG4_0	Non-secure	/
		TIMG4_1	Non-secure	/
		TIMG4_2	Non-secure	/
		TIMG4_3	Non-secure	/
		TIMG12_0	Non-secure	/
		TIMG8_0	Non-secure	/
		TIMG8_1	Non-secure	/
	PPC_SECATTRIB_WDT	WWDT	Non-secure	/

Copyright © 2026 Texas Instruments Incorporated

Table 2-3. Secure and Privilege Configurations in GSC (continued)

Security	PPC_SECATTRIB_CRC	CRC	Secure	/
	PPC_SECATTRIB_AES	AES	Non-secure	/
	PPC_SECATTRIB_SHA	SHA	Non-secure	/
	PPC_SECATTRIB_PKA	PKA	Non-secure	/
	PPC_SECATTRIB_TRNG	TRNG	Secure	/
	PPC_SECATTRIB_DEBUGSS	Debug Subsystem	Secure	/
	PPC_SECATTRIB_KEYSTORECTL	Keystore	Non-secure	/
	PPC_SECATTRIB_GSC PPC_PRIVATTRIB_GSC	GSC	Secure	Privilege

Besides security attribute configuration, GSC can also set up other protections on MSPM33 memory, like write erase protection(WEP), flash hide protection(HDP). Please refer to Secure Storage section in MSPM33C Security User's Guide.

2.2.4 Security Attribute with TrustZone® and GSC

When CPU initiate the transaction, firstly it will be filtered by SAU & IDAU, then memory access will be filtered by GSC.

For first filter by SAU & IDAU, as mentioned in Section 2.2.1, if processor is in secure state, data access to S, NSC and NS(defined by SAU & IDAU) is allowed. If processor is in non-secure state, only data access to NS is allowed.

If transaction passes SAU & IDAU, then GSC filter this memory access on the bus. The security state of memory access is determined by processor security state and the security attribute of the location(defined by SAU & IDAU). Figure 2-2 shows the CPU access through SAU & IDAU and GSC.

- If processor state is non-secure, memory access should be non-secure.
- If processor state is secure, state of memory access is equal to the security attribute(defined by SAU & IDAU) of the location being accessed.

Note

If the processor state is S and the memory being accessed is NS—as defined by the SAU and IDAU—but the GSC marks that memory region as S, the access will pass the first layer of filtering (SAU and IDAU) but be rejected by the second layer (GSC). In this case, memory access is NS and cannot pass GSC for S attribute.

Please note there are ATTRIBVIOLS_CONFIG in GSC for the behavior when secure transaction accesses to non-secure resources defined by GSC. The default setting is access allowed(Figure 2-2 follows default setting). User can configure ATTRIBVIOLS_CONFIG to trigger GSC blocking behavior when secure transaction accesses to non-secure resources defined by GSC.

Copyright © 2026 Texas Instruments Incorporated

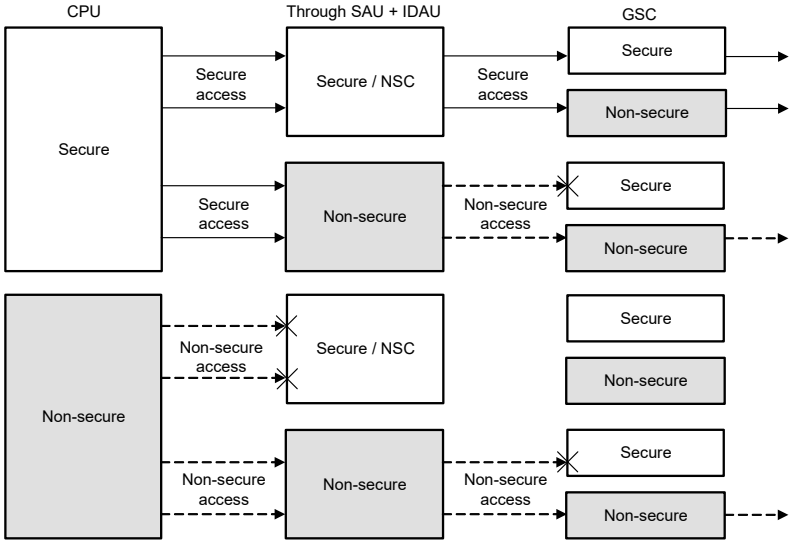


Figure 2-2. Transaction filtering with SAU+GSC and GSC(with default ATTRIBVIOLS_CONFIG)

Here is an example to show the final attribute with TrustZone® plus GSC. When secure application or non-secure application access data in 0x00000100 or 0x10000100, the following shows the final attribute with different configurations on SAU and GSC.

Table 2-4. Final Security Attribute with TrustZone® and GSC

CPU state	Access address	IDAU	SAU	SAU+IDAU	Bus Transaction[1]	GSC	Final attribute
Secure application/ NSC functions	0x00000100	NS	0x0-0x10000 S/NSC	S/NSC	S access	S	Access pass
						NS	Access pass or blocked by GSC(based on GSC ATTRIBVIOLS_CONFIG)
			0x0-0x10000 NS	NS	NS access	S	Blocked by GSC
						NS	Access pass
	0x10000100	NSC	0x0-0x10000 S	S	S access	S	Access pass
						NS	Access pass or blocked by GSC(based on GSC ATTRIBVIOLS_CONFIG)
			0x0-0x10000 NS/NSC	NSC	S access	S	Access pass
						NS	Access pass or blocked by GSC(based on GSC ATTRIBVIOLS_CONFIG)
Non-secure application	0x00000100	NS	0x0-0x10000 S/NSC	S/NSC	Blocked by SAU + IDAU	S	Blocked by SAU + IDAU
						NS	Blocked by SAU + IDAU
			0x0-0x10000 NS	NS	NS access	S	Blocked by GSC
						NS	Access pass
	0x10000100	NSC	0x0-0x10000 S	S	Blocked by SAU + IDAU	S	Blocked by SAU + IDAU
						NS	Blocked by SAU + IDAU
			0x0-0x10000 NS/NSC	NSC	S access	S	Access pass
						NS	Access pass or blocked by GSC(based on GSC ATTRIBVIOLS_CONFIG)

1. Bus components that are based on AMBA 5 can identify TrustZone transactions and differentiate between Secure and Non-secure transactions. This feature allows the Security state of each transaction to be propagated from the CPU to the interconnect. The transactions then pass on, with their Security state, to

Copyright © 2026 Texas Instruments Incorporated

other components in the system that require security awareness. A security flag (NS) on the bus identifies Secure TrustZone transactions.

2.2.5 Secure and Non-secure Debug Authentication

Debug authentication is also split to secure debug and non-secure debug. User can debug MCU in non-secure state even if the secure state is locked down.

Please refer to *MSPM33 NONMAIN Flash Memory Configuration Guide* for the information about configuration on debug authentication policy.

2.3 TrustZone® Software Development

To integrate using the SAU in the programmers code development we recommend using the CMSE library. This library is designed by Arm and allows the user to define functions as secure or non-secure. For more detailed information on the guidelines for this, see the ARMv8-M Secure software guidelines.

The ARMv8-M instruction uses BXNS (Branch and Exchange Non-secure), BLXNS(Branch with Link and Exchange Non-secure) and SG (secure gateway) to transition between non-secure and secure code. The BXNS and BLXNS are used to transition from secure code to non-secure code and the SG is used to transition from non-secure to secure. The SG command utilizes the NSC regions to make this transition. To use these instructions into the programming environment, the CMSE library integrates these functions into the function calls.

Note

BXNS and BLXNS are available for secure state only. When the processor is in the NS state, these instructions are undefined and trigger UsageFault if used.

For the instructions usage, see Section 3.2.4 for implementation in the code.

Copyright © 2026 Texas Instruments Incorporated

3 Develop TrustZone Based Application

This section introduces how to develop secure application and non-secure application based on TrustZone examples in SDK.

3.1 Device Configuration

3.1.1 NONMAIN FLASH

NONMAIN FLASH is a dedicated region to configure the parameter for boot. During ROM boot, MSPM33 supports to configure Global Security Controller(GSC) to define the FLASH regions as secure / non-secure and privilege / non-privilege, which is based on the configuration in NONMAIN FLASH.

Note

GSC can be configured during either boot or runtime.For the GSC configuration in the boot, only FLASH will be configured for Secure/non-secure attributes, because NONMAIN only supports to configure FLASH attribution. See the Section 3.1.1.1.For the GSC configuration in runtime, FLASH, SRAM, peripherals can be configured for Secure/non-secure attributes. See the Section 3.2.3 for GSC configuration in runtime.

3.1.1.1 Security and Privilege Configuration on FLASH

The user can configure the security/privilege protection on bank0, bank1 or DATA FLASH by setting registers below. During the boot, GSC is set according to the NONMAIN FLASH, and specific FLASH region is marked as secure or non-secure, privilege or non-privilege, which is the prerequisite for TrustZone.

For Security Protection on bank x(x=0,1), BANKx_SECURITY_PROTECTION_A field and BANKx_SECURITY_PROTECTION_B field are to define the secure regions in the whole 512KB memory. A value of 0 indicates the corresponding sectors are secure. For BANKx_SECURITY_PROTECTION_A, the 1 bit represents 1 sector in flash memory (2KB). For BANKx_SECURITY_PROTECTION_B, the 1 bit represents 8 sector in flash memory (16KB). For Privilege Protection on bank x(x=0,1), BANKx_PRIVILEGE_PROTECTION_A field and BANKx_PRIVILEGE_PROTECTION_B field are to define the secure regions in the whole 512KB memory. A value of 0 indicates the corresponding sectors are privilege. For BANKx_PRIVILEGE_PROTECTION_A, the 1 bit represents 1 sector in flash memory (2KB). For BANKx_PRIVILEGE_PROTECTION_B, the 1 bit represents 8 sector in flash memory (16KB). For Security Protection on DATA FLASH, DBANK_SECURITY_PROTECTION field is to define the secure regions in the whole data bank memory. A value of 0 indicates the corresponding sectors are secure. For DBANK_SECURITY_PROTECTION, the 1 bit represents 1 sector in flash memory (2KB). For Privilege Protection on DATA FLASH, DBANK_PRIVILEGE_PROTECTION field is to define the secure regions in the whole data bank memory. A value of 0 indicates the corresponding sectors are privilege. For DBANK_PRIVILEGE_PROTECTION, the 1 bit represents 1 sector in flash memory (2KB).

3.1.1.2 Other Configurations in NONMAIN FLASH

NONMAIN FLASH supports some more configurations, which may be useful in security application.

For example, user can also apply write/erase protection on bank0, bank1 or DATA FLASH, which can lock the specific range to avoid erase or modification on the sensitive code / data.

User can also set up secure debug policy or BSL configuration in the NONMAIN. For more information, please refer to *MSPM33 NONMAIN Flash Memory Configuration Guide*.

3.1.1.3 Configure NONMAIN FLASH in the Sysconfig

User can use Sysconfig to configure the NONMAIN FLASH. Below is the example to configure security protection on bank0 in NONMAIN. For the instructions to configure NONMAIN, please refer to MSPM33 NONMAIN Flash Memory Configuration Guide.

Copyright © 2026 Texas Instruments Incorporated

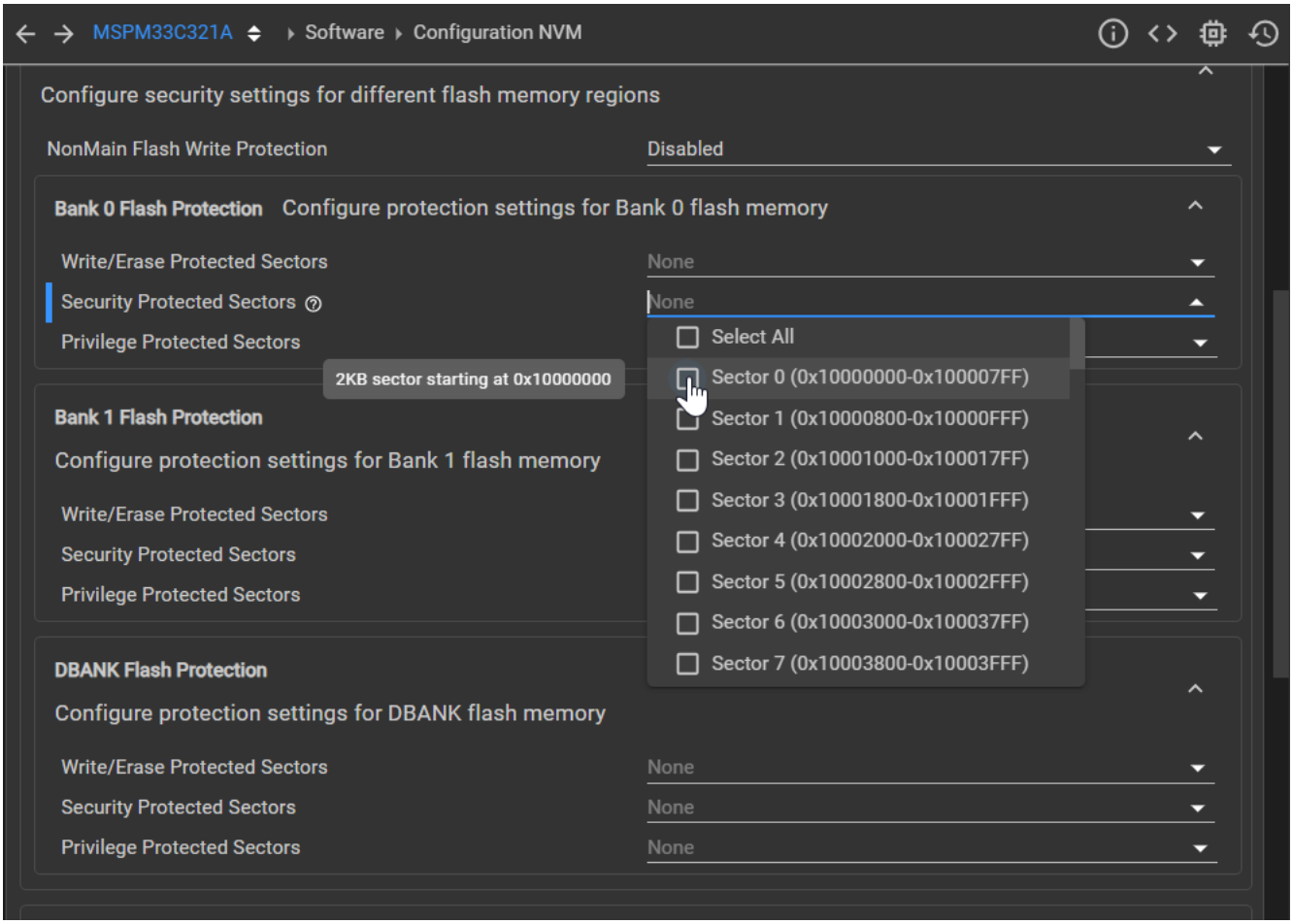


Figure 3-1. Configure NONMAIN via SysConfig

3.2 Develop Secure Application

During secure application development, typically the following content needs to be implemented.

- Set up project dependency between secure application/project and non-secure application/project, so transition between secure application and non-secure application can execute as expected. As shown in Section 3.2.1.
- Configure SAU and GSC to partition the secure and non-secure regions, as shown in Section 3.2.2 and Section 3.2.3.
- Define NSC functions to enable function call from NS application to S application, as shown in Section 3.2.4.
- Correctly setting on the memory assignment, as shown in Section 3.2.5.1.
- Develop general tasks/codes in secure application, as shown in Section 3.2.5.2.
- Correctly configure secure/non-secure IRQ, as shown in Section 3.2.5.3.
- Correctly set up instruction to branch to non-secure application., as shown in Section 3.2.5.4.

3.2.1 Project Dependency

To develop a secure application, developers must first pay attention to project dependency. Examples in mspm33_sdk\examples\nortos\LP_MSPM33C321A\security is a good template for code development on secure applications and non-secure applications.

In the example, non-secure example is dependent on secure example. The following are the steps to configure project dependency in secure application project.

1. Enable CMSE output library in Properties – Build – Tools – Arm Linker – Runtime Envrionment – Create a CMSE output library file. In the example, '${PROJECT_BUILD_DIR}/implib.o' as the output. The implib.o will be generated during compiling, and is referred to in Section 3.3.1 step 2

Copyright © 2026 Texas Instruments Incorporated

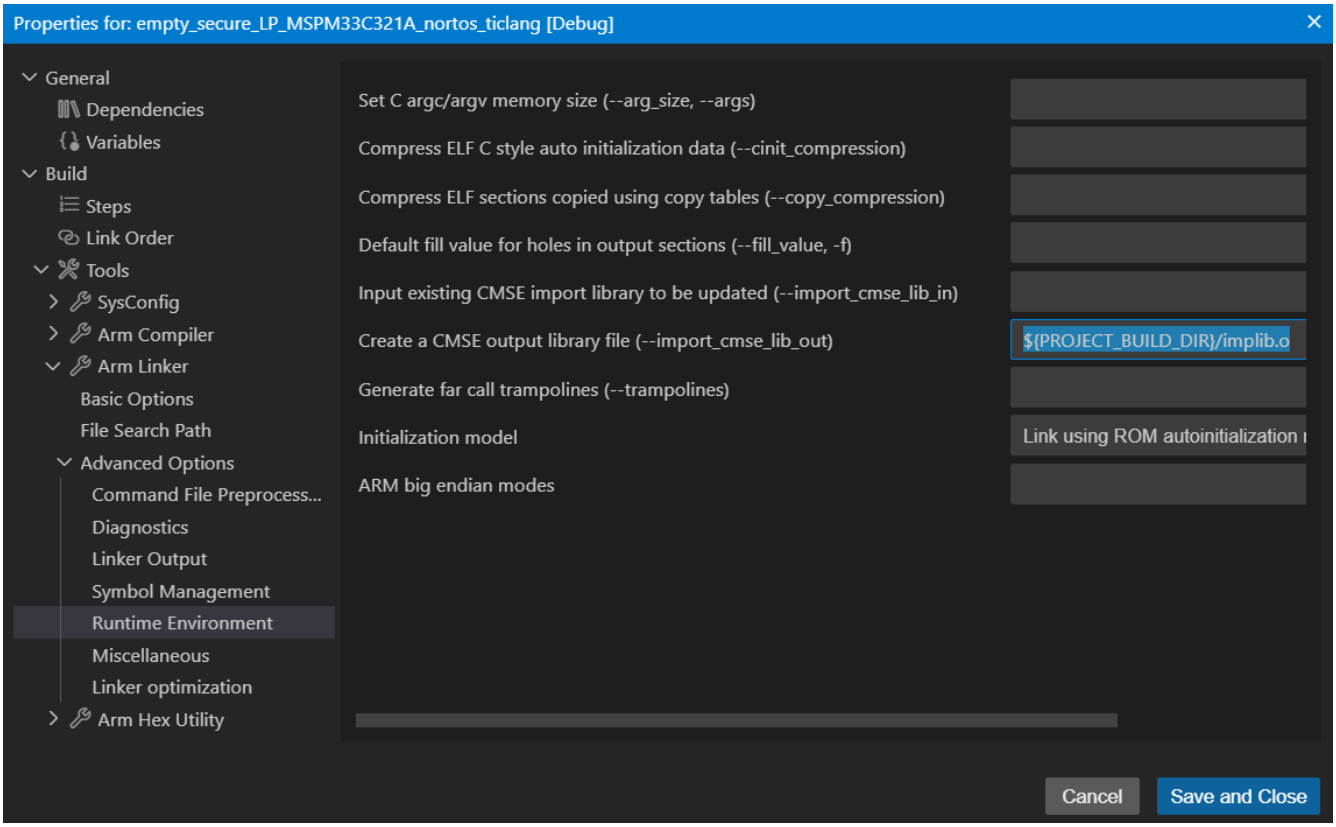


Figure 3-2. Enable CMSE Output Library

2. Enable Arm Hex Unity to output txt file. In Section 3.3.1 step 5(optional), a post-step command can execute to combine both S and NS txt to a final txt. Users can find final txt in the debug folder of the secure application project. The split txt files or combined txt file can also be used for mass production.

Copyright © 2026 Texas Instruments Incorporated

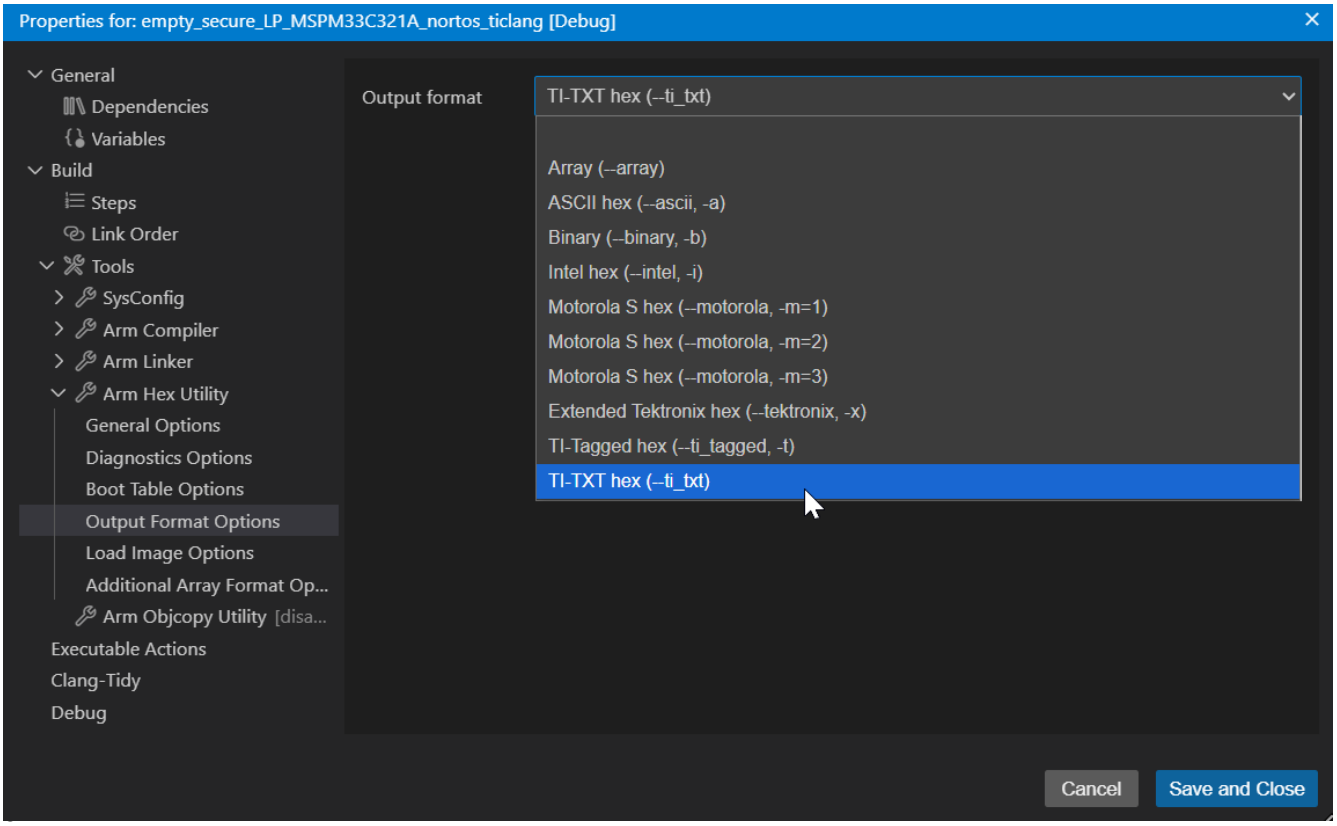


Figure 3-3. Enable TI-TXT Hex Output

4.

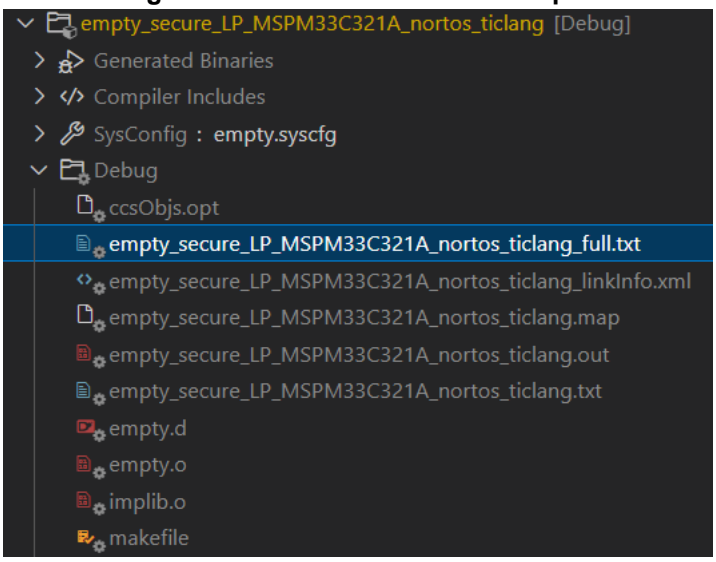


Figure 3-4. Generated Combined TI-TXT File

3.2.1.1 Structure of Secure Application Example

Here is the example of empty_secure_LP_MSPM33C321A_nortos_ticlang from SDK.

Table 3-1. Structure of Secure Application Example

File	Location	Description
empty.c	${PROJECT_ROOT}	C file for main()
empty.syscfg	${PROJECT_ROOT}	Configuration file for sysconfig
mspm33c321a_s.cmd	${PROJECT_ROOT}	Link command file to assign memory of secure application
startup_mspm33c321x_ticlang_s.c	${PROJECT_ROOT}	C file for interrupt vector table of secure application

Copyright © 2026 Texas Instruments Incorporated

Table 3-1. Structure of Secure Application Example (continued)

config.h	${PROJECT_ROOT}	Header file to define SAU region
partition.h	${PROJECT_ROOT}	Header file to define SAU region, based on config.h
nsc_example.c	${PROJECT_ROOT}	C file for the definition of non-secure callable(NSC) functions
sns_helper.h	${PROJECT_ROOT}	Header file for pre-defined attribute related to CMSE library

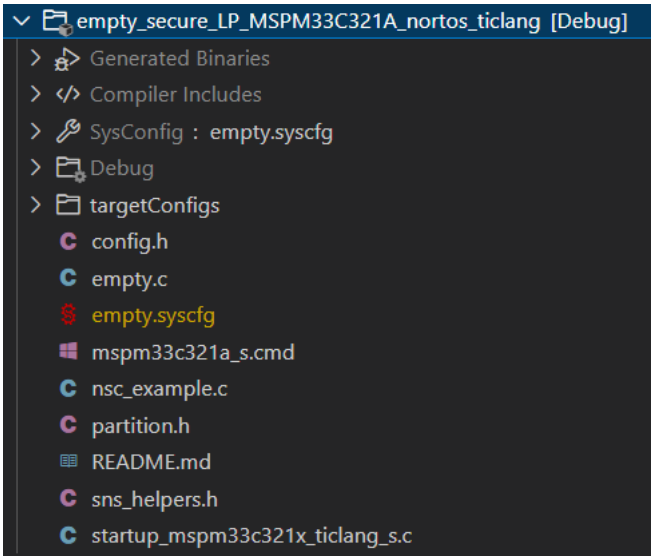


Figure 3-5. Structure of empty_secure_LP_MSPM33C321A_nortos_ticlang

3.2.2 SAU and IDAU Configuration

In MSPM33, IDAU is pre-defined by TI. See pre-definition in the Table 2-2. SAU can be configured by the user to define S/NS/NSC based on how the SAU_CTRL and SAU_RLAR are configured.

Table 3-2. SAU Control on the S, NS and NSC Properties

SAU_CTRL.ENABLE	SAU_CTRL.ALL_NS	SAU_RLAR.ENABLE	SAU_RLAR.NSC	SAU Security Attribute
0	0	-	-	All of the memory is marked *Secure* (Reset state)
0	1	-	-	All of the memory is marked *NS*
1	-	0	-	All of the region as per SAU_RBAR and SAU_RLAR is marked *Secure*
1	-	1	1	All of the region as per SAU_RBAR and SAU_RLAR is *NSC*
1	-	1	0	All of the region as per SAU_RBAR and SAU_RLAR is *NS*

Note

When users need to configure security attribution unit (SAU) for security attribution, configured address is based on logical address. In bank-swappable configuration, please carefully configure SAU based on logical address.

Users can configure SAU to configure the final security attribute of a different address. Any violation of the security property results in a SecureFault interrupt. When the device powers on and SAU is not configured, all memory is *Secure* by default. See Table 2-2 for the security attributes through SAU+IDAU.

Note

At reset before any SAU region is programmed, SAU_CTRL.ALLNS register bit selects the default internal security level. On reset, SAU_CTRL.ALLNS is always set to zero, setting all memory to Secure state. Setting SAU_CTRL.ALLNS prevents any external IDAU from overriding security level.

Copyright © 2026 Texas Instruments Incorporated

3.2.2.1 Configure SAU in the Project

In the security example, user can configure the specific address range for SAU in config.h and partition.h. The following is the configurable parameters in the example.

According to the default configuration in the example, SAU will be set as

- For FLASH, 0x10000-0x1FFFF(64KB) is NS(SAU region 0), 0xF800-0xFFFF(2KB) is NSC(SAU region 3). All other regions are S.
- For SRAM, 0x20010000-0x2001FFFF(64KB) is NS(SAU region 1). Other regions are S.
- For peripherals, 0x40000000-0x4FFFFFFF(all peripherals) is NS(SAU region 2).

Table 3-3. User-configurable Parameters in the Example

Location	Defined parameter	Description	Default value in the example
config.h	NON_SECURE_APP0_START	Non-secure app start address	0x10000U
	NON_SECURE_APP0_LEN	Non-secure app length	0x10000U (64KB)
	SECURE_NSC_APP0_START	NSC region start address	0xF800U
	SECURE_NSC_APP0_LEN	NSC region length	0x800U (2KB)
	NON_SECURE_APP0_SRAM_START	Non-secure SRAM start	0x20010000U
	NON_SECURE_APP0_SRAM_LEN	Non-secure SRAM length	0x10000U (64KB)

Copyright © 2026 Texas Instruments Incorporated

Table 3-3. User-configurable Parameters in the Example (continued)

partition.h	SAU_INIT_CTRL_ENABLE	SAU->CTRL register bit ENABLE	1
	SAU_INIT_CTRL_ALLNS	SAU->CTRL register bit, 1 – All Memory is NS, 0 – S/NS depends on the configurations	0
	SAU_INIT_REGION0	SAU Region 0: Non-Secure Application Flash	1
	SAU_INIT_NSC0	SAU Region 0 is NS or NSC	0 (NS)
	SAU_INIT_START0	Start address of SAU region 0	NON_SECURE_APP0_START
	SAU_INIT_END0	End address of SAU region 0	SAU_INIT_START0 + NON_SECURE_APP0_LEN -1
	SAU_INIT_REGION1	SAU Region 1: Non-Secure SRAM	1
	SAU_INIT_NSC1	SAU Region 1 is NS or NSC	0 (NS)
	SAU_INIT_START1	Start address of SAU region 1	NON_SECURE_APP0_SRAM_START
	SAU_INIT_END1	End address of SAU region 1	SAU_INIT_START1 + NON_SECURE_APP0_SRAM_LEN-1
	SAU_INIT_REGION2	SAU Region 2: Non-Secure Peripheral Access	1
	SAU_INIT_NSC2	SAU Region 2 is NS or NSC	0 (NS)
	SAU_INIT_START2	Start address of SAU region 2	0x40000000
	SAU_INIT_END2	End address of SAU region 2	SAU_INIT_START2 + 0x10000000 - 1
	SAU_INIT_REGION3	SAU Region 3: Non-Secure Callable Region	1
	SAU_INIT_NSC3	SAU Region 3 is NS or NSC	1 (NSC)
	SAU_INIT_START3	Start address of SAU region 3	SECURE_NSC_APP0_START + 0x10000000
	SAU_INIT_END3	End address of SAU region 3	SAU_INIT_START3 + SECURE_NSC_APP0_LEN - 1

Example will call configure_sau() to set up the SAU, which will execute following steps to define the S/NS/NSC region based on the above parameters.

1. Selecting which region to configure through the SAU->RNR register.
2. Selecting the regions base address using the SAU->RBAR register
3. Set the SAU->RLAR to configure the size of the region
4. Repeat steps 1 - 3 for all regions
5. Configure the SAU->CTL register to propagate all of the configured regions

In the example, steps1-3 will be repeat 4 times for region 0 – 3. For MSPM33, users can assign up to 8 regions on SAU.

3.2.3 Global Security Controller Configuration

The Global Security Controller (GSC) is a peripheral which is used to configure the security firewalls for secure and non-secure context of the application. It adds an extra layer of security on top of the TrustZone architecture. GSC contains SPC(SRAM protection controller), FPC(FLASH protection controller), PPC(peripheral protection controller). Any violation of the security property results in a non-maskable interrupt (NMI) which is logged into the EAM module.

Copyright © 2026 Texas Instruments Incorporated

Note

GSC can be reset by BOOTRST or higher level reset. SYSRST or CPURST cannot reset GSC.

For SRAM secure and privilege configuration, SRAM is configured by chunks of 16KB. Each byte of SPC_SECATTRIBx/ SPC_PRIVATTRIBx corresponds to 16 KB chunk. Each chunk implements security/privilege as 3x 4KB, 1x 2KB, 1x 1KB and 2x 512 byte to provide granular control to application.

Table 3-4. Configure by Chunks on SRAM

SRAM Chunk	Config Bit	4KB Chunk Size	8KB Chunk Size	16KB Chunk Size	Irregular Chunk-1	Irregular Chunk-2	Irregular Chunk-1
512B	0	0x0000000F	0x0000001F	0x0000007F	0x00000007 (2KB)	0x0000001F (8KB)	0x0000000F (4KB)
512B	1						
1KB	2						
2KB	3				0x00000018 (6KB)		
4KB	4	0x00000010					0x00000370 (13KB)
4KB	5	0x00000020	0x00000060		0x00000020 (4KB)	0x00000360 (9KB)	
4KB	6	0x00000040			0x00000340 (5KB)		
Reserved	7	-	--	-			
512B	8	0x00000F00	0x00001F00	0x00007F00			
512B	9						
1KB	10				0x00000C00 (3KB)	0x00001C00 (7KB)	0x00003C00 (11KB)
2KB	11						
4KB	12	0x00001000			0x00001000 (4KB)		
4KB	13	0x00002000	0x00006000		0x00002000 (4KB)	0x00002000 (4KB)	
4KB	14	0x00004000			0x00004000 (4KB)	0x00004000 (4KB)	0x00004000 (4KB)
Reserved	15	-	-	-	-	-	-

For FLASH secure and privilege configuration, FLASH is configured by sectors (0-64KB) or blocks (64KB-512KB). Note the address below is the physical address rather than the logic address, which must be paid attention when doing FLASH bank swapping.

Table 3-5. Configure by Sectors or Blocks on FLASH

FLASH BANK	Physical Address	Registers	Granular Control
BANK0	0x00000000-0x0000FFFF (0-64KB)	FPC_SECATTRIBA0 FPC_PRIVATTRIBA0	Sector(2KB)
	0x00010000-0x0007FFFF (64KB-512KB)	FPC_SECATTRIBB0 FPC_PRIVATTRIBB0	Block(8 sectors, 16KB)
BANK1	0x00080000-0x0008FFFF (0-64KB)	FPC_SECATTRIBA1 FPC_PRIVATTRIBA1	Sector(2KB)
	0x00090000-0x000FFFFF (64KB-512KB)	FPC_SECATTRIBB1 FPC_PRIVATTRIBB1	Block(8 sectors, 16KB)
DATA FLASH	0x80000000-0x80007FFF (0-64KB)	FPC_SECATTRIBA_DFLASH FPC_PRIVATTRIBA_DFLASH	Sector(2KB)

If the device is 512KB dual bank device, then every bank is 256KB rather than 512KB.

Copyright © 2026 Texas Instruments Incorporated

Note

When users need to configure global security controller(GSC) for security privilege attribution, FLASH hide protection, static write protection or others, related registers in GSC are based on physical banks. In bank-swappable configuration, carefully configure GSC based on physical address.

For peripheral secure and privilege configuration, most peripherals only support secure configuration, except DMA, SYSCTL and GSC.

The DMA is uniquely handled within the GSC framework as it is both an initiator and a target. When the DMA is configured for secure access then both the initiator and the target ports are set to secure state. The power on reset value for DMA instance 0 is always secure and privilege. Only the CPU in secure mode can configure the DMA0 channel registers. Once configured, all write and read access generated by DMA0 initiator ports are also secure. This ensure that after the device is powered on and the CPU begins code execution in secure mode, the DMA0 is in a secure state for moving data between peripherals and memory and does not require any additional configuration. The power on reset value for DMA instance 1 is always non-secure and unprivileged. This allows the a non-secure application to request data transfer service without any special handling.

Note

The GSC is also secure and privilege at power on reset. A non-secure or unprivileged application cannot directly modify the DMA property. This must be handled within a secure application.

3.2.3.1 Configure GSC in the Sysconfig

In the project, GSC is supported in Sysconfig. User can configure secure/non-secure and privilege/non-privilege on SRAM, FLASH and peripherals via Sysconfig.

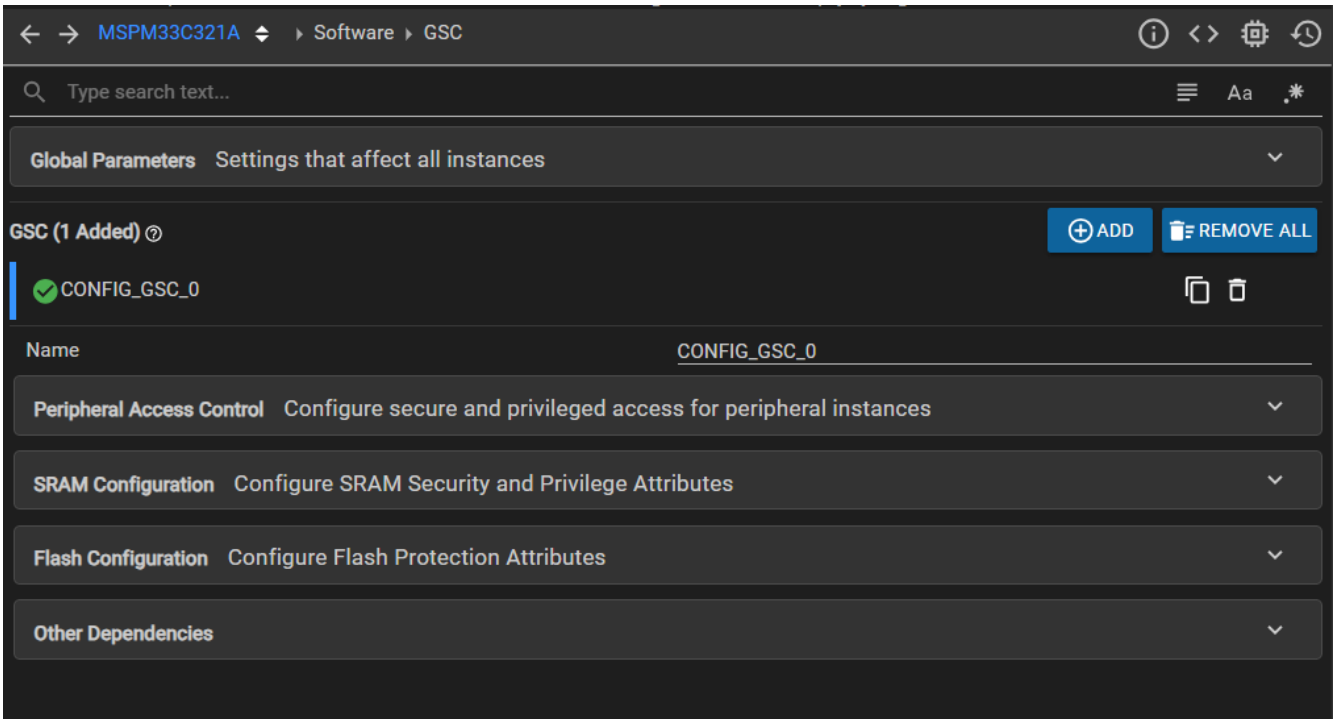


Figure 3-6. Configure GSC via SysConfig

Figure 3-7 shows how to configure the security attributes on SRAM of MSPM33C321A. User can configure SRAM by 16KB chunks, or configure custom chunk size in 'SRAM Detailed View'.

Figure 3-8 shows how to configure the security attributes on MAIN FLASH bank0 of MSPM33C321A. For first 32 sectors, security protection can be applied to each sector(2KB) individually. For the remaining sectors, security protection can be applied to each blocks(8 sectors) individually.

Copyright © 2026 Texas Instruments Incorporated

Figure 3-9 shows how to configure the security attributes on peripherals of MSPM33C321A. User can configure all peripherals in ANALOG, COMMUNICATION, SYSTEM, TIMERS, SECURITY.

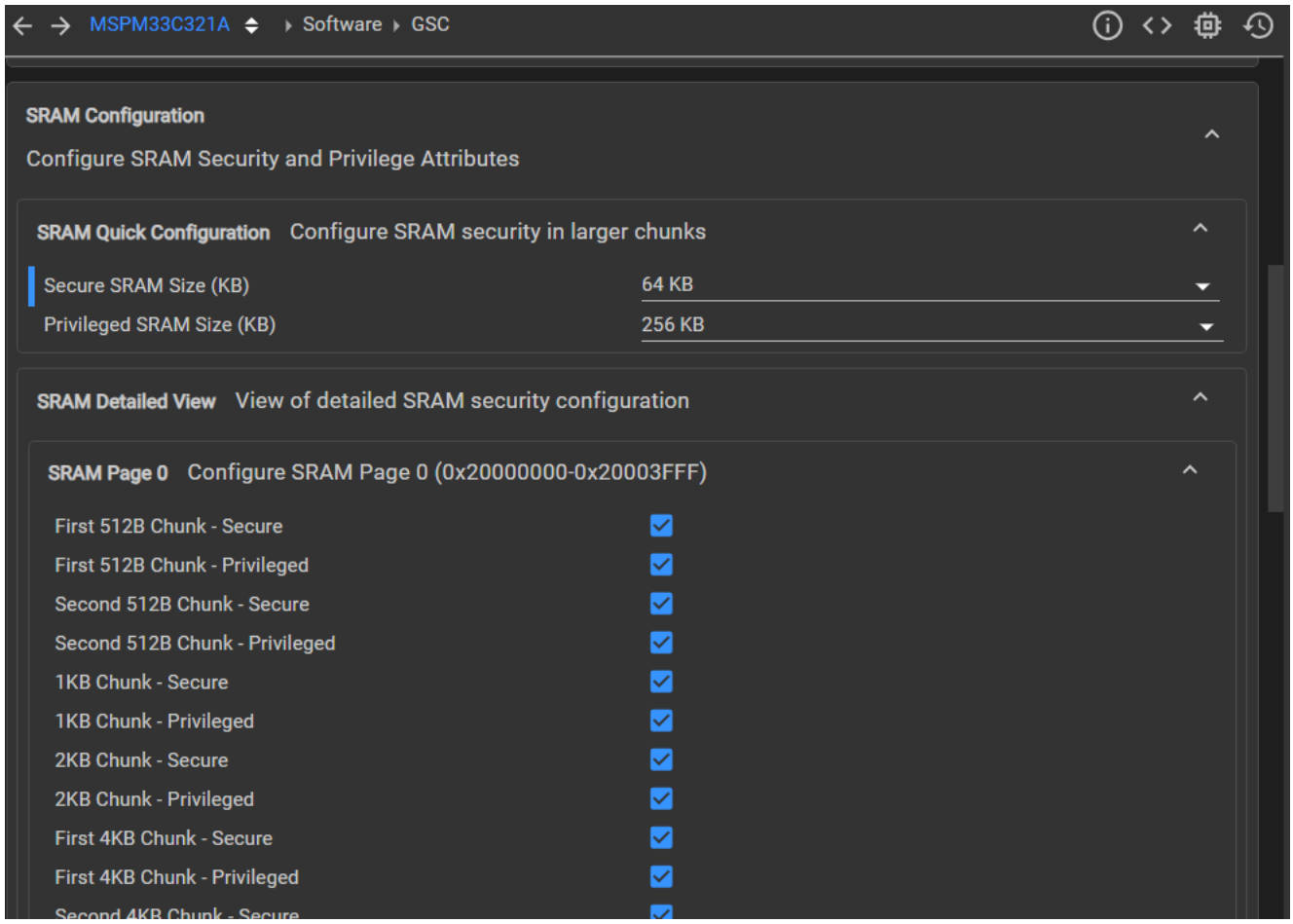


Figure 3-7. Configure SRAM Attribute via SysConfig

Copyright © 2026 Texas Instruments Incorporated

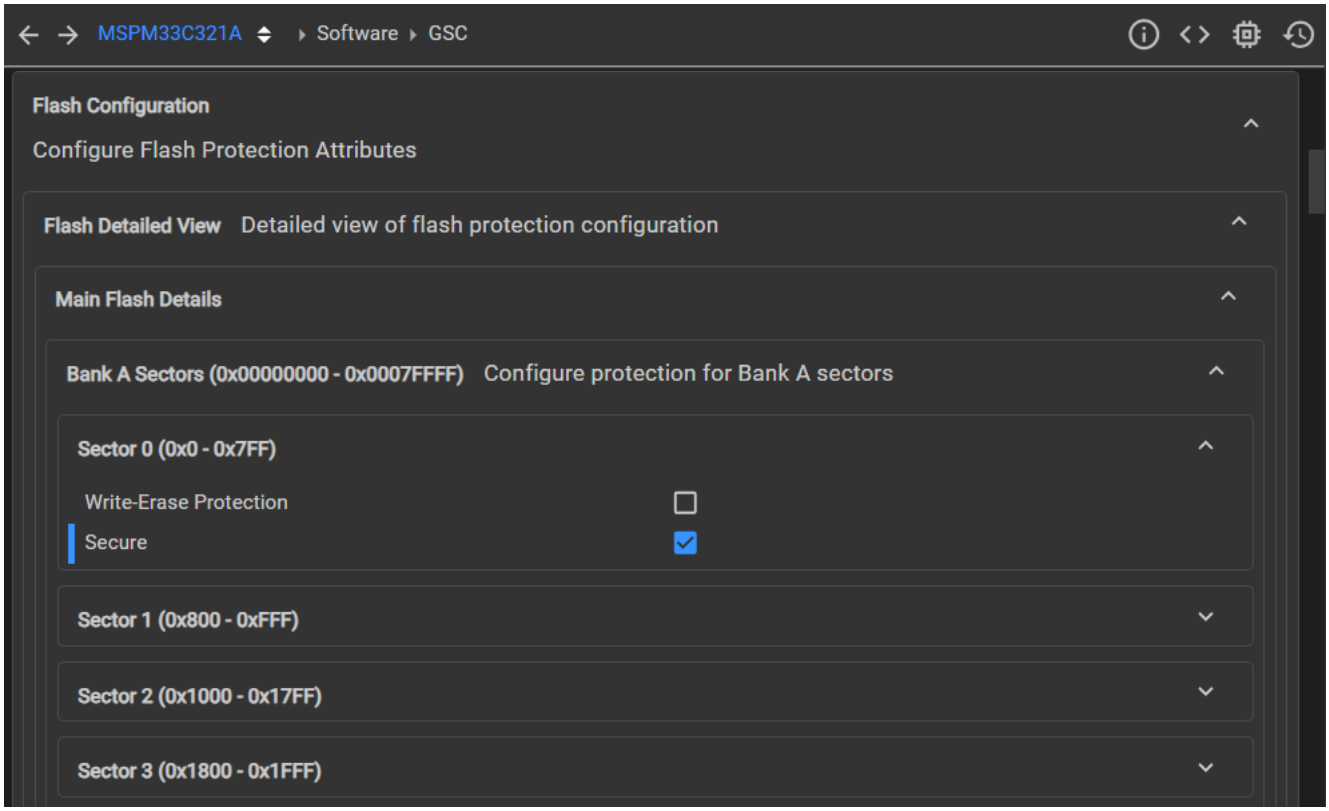


Figure 3-8. Configure FLASH Attribute via SysConfig

Copyright © 2026 Texas Instruments Incorporated

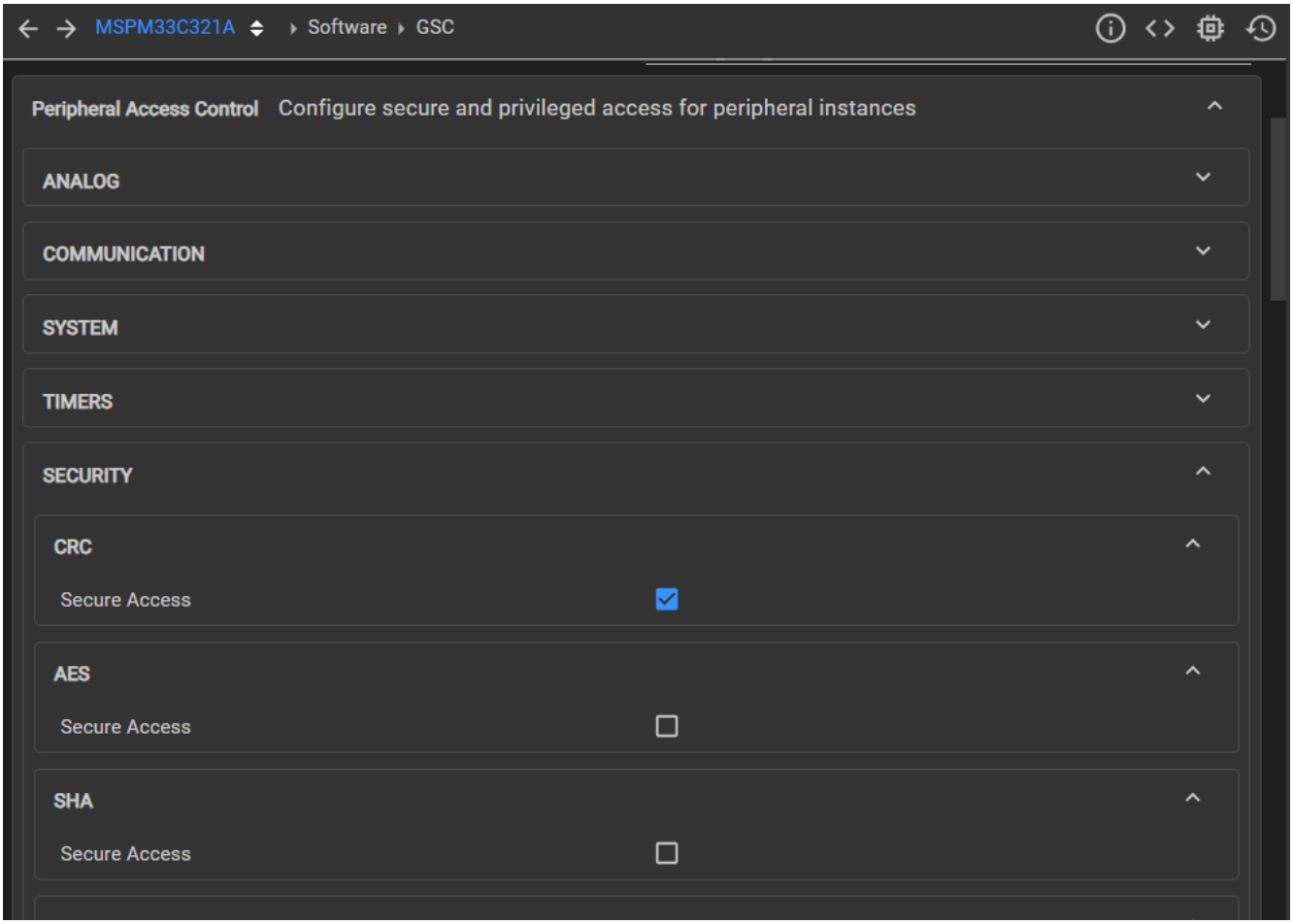


Figure 3-9. Configure Peripherals Attribute via SysConfig

3.2.3.2 Other Configurations in GSC

Global Security Controller(GSC) can not only define the secure/ non-secure or privilege/non-privilege attribute on the device resource, but also apply other protections like write erase protection and flash hide protection on the memory. In secure application, user may use these protection to implement secure storage. These protection implementation is not introduced in this document, please refer to Technical Reference Manual or Secure Storage section in security user guide.

Note

GSC can be configured during either boot or runtime.

For the GSC configuration in the boot, GSC is configured based on the NONMAIN configuration. Not all GSC features are supported to configure during boot. Please refer to *MSPM33 NONMAIN Flash Memory Configuration Guide* for NONMAIN configuration.

For the GSC configuration in runtime, GSC is configured in application code and user can configure GSC via SysConfig. Please refer to Secure Storage section in MSPM33C Security User's Guide.

3.2.4 Defining Functions as Secure and Non-Secure

The ARMv8-M instruction uses specific instructions to transition between non-secure and secure code, as shown in Figure 3-10. To use these instructions into the programming environment, the CMSE library integrates these functions into the function calls. For more information on the instructions, see Armv8-M Security Extension User Guide.

- SG – Secure gateway. Used for switching from Non-Secure to Secure state as the first instruction of Secure entry veneer

Copyright © 2026 Texas Instruments Incorporated

- BXNS – Branch with exchange to Non-Secure state. Used by Secure software to return from a Security entry point function.
- BLXNS – Branch with link and exchange to Non-Secure state. Used by Secure software to call Non-Secure functions.
- FNC_RETURN – When a BXNS instruction causes a transition to the Non-Secure state, it places a reserved address called FNC_RETURN in the link register. The Non-Secure function triggers the return to the Secure state by branching to this address.

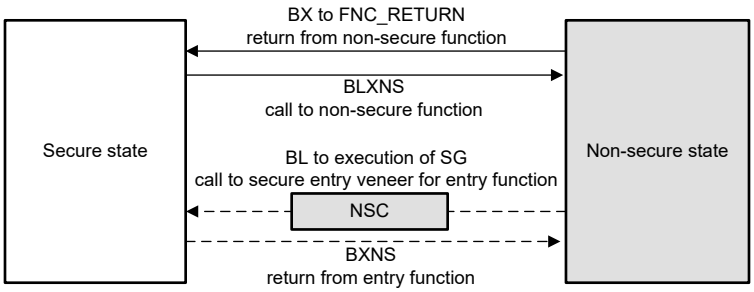


Figure 3-10. Transition between Secure State and Non-secure State

In the example, the core transitions from S to NS state via Non Secure Function calls(marked using NS_FUNC in the example), and the branch to the NS code is done by the BLXNS instruction. The core transitions from NS to S state via Secure Function Calls (marked using NS_FUNC_ENTRY, these work through NSC veneers), and the core must encounter a SG instruction (which is generated as a part of a Secure Gateway veneer by the compiler when marked with NS_FUNC_ENTRY). Users can find all definitions in the sns_helpers.h.

```
/**
 * @brief Non-Secure Function Call Attribute
 * @note Use this attribute for function pointers that call non-secure functions from secure code
 * @note Generates secure gateway veneers for transitions from secure to non-secure state
 */
#define NS_FUNC __attribute__((cmse_nonsecure_call))
/**
 * @brief Non-Secure Callable Entry Attribute
 * @note Use this attribute for secure functions that can be called from non-secure code
 * @note Functions with this attribute must be placed in NSC (Non-Secure Callable) memory regions
 */
#define NS_FUNC_ENTRY __attribute__((cmse_nonsecure_entry))
```

3.2.4.1 Define Non Secure Callable function in the Project

In the project, user can use NS_FUNC_ENTRY to definite the Non Secure Callable function.

The function itself does not reside in NSC region. The function is actually present in secure memory. NS_FUNC_ENTRY is defined as an **atribute** in the header file. This instructs the compiler to generate a "secure gateway veneer" which is nothing but :

1. SG instruction
2. branch instruction

later in the linker,

Veneer$$CMSE: {} > NSC_MEM

is present which will make this veneer go to the NSC region. The actual function implementation will go to secure region, as defined by the linker.

What the SG instruction is doing is changing the security state of the core from NS to S, banks the registers, sets bit[0] of LR register to 0, which indicates that the return will cause a transition back from S to NS

Copyright © 2026 Texas Instruments Incorporated

in short, SG is a privileged instruction which *must* be in NSC. The core understands that this means some NS program is requesting access to a S resource/code. And by design NS should not be able to simply change the core state to S.

In the secure example, NSC function is defined in nsc_example.c. There is also an nsc_example.h in non-secure example, so non-secure application can call the NSC function during runtime.

```
33  #include "sns_helpers.h"
34
35  NS_FUNC_ENTRY void hello_secure( )
36  {
37      volatile uint32_t ii = 0;
38      while ( ii == 0 )
39          ;
40  }
```

Figure 3-11. Definition of NSC Function in nsc_example.c of Secure Application

```
33  extern void hello_secure();
```

Figure 3-12. Extern Definition of NSC Function in nsc_example.h of Non-secure Application

3.2.5 Code Implementation on Secure Application

In the secure application, MCU must configure SAU and GSC for secure and non-secure and privilege and non-privilege configurations, which are introduced in previous sections. Besides, the MCU can also initialize secure peripheral and do secure tasks as needed. When secure application is finished, it must use the BLXNS instruction to branch to non-secure application. Figure 3-13 shows the flow of secure application and non-secure application.

In general, the secure application is able to access both secure region and non-secure regions, and non-secure application can only access to non-secure region. When non-secure application needs to access secure region or call the secure function, the non-secure application should call the non-secure callable function, which is defined in Section 3.2.4.1. Figure 3-13 shows the general flow for secure application and non-secure application, which is based on the security examples in the SDK.

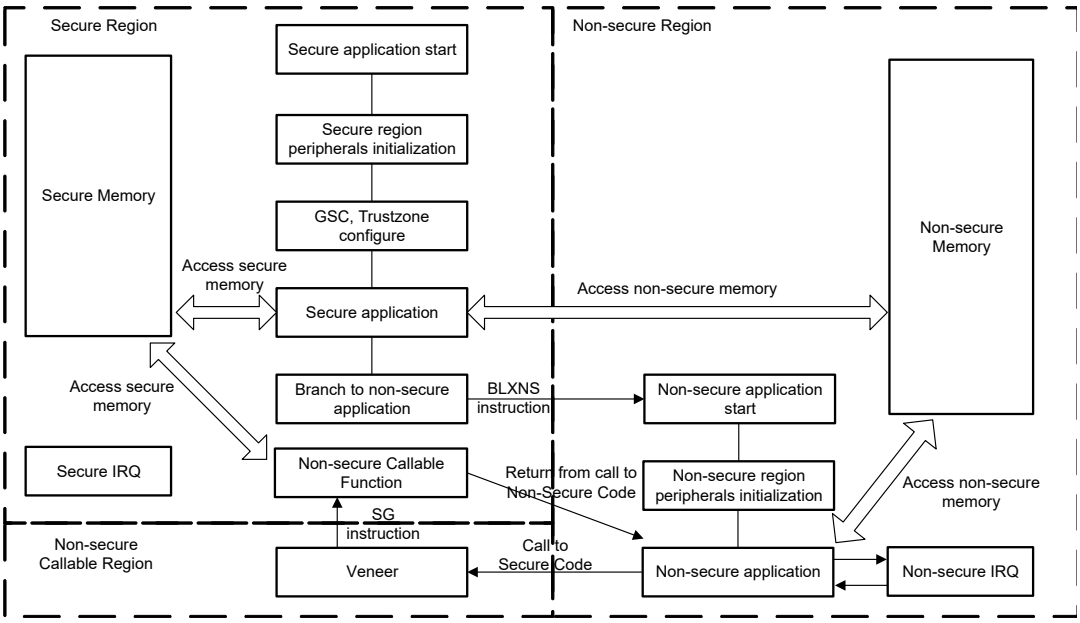


Figure 3-13. Flow for Secure Application and Non-secure Application

Copyright © 2026 Texas Instruments Incorporated

Before starting this section, users must configure correctly according to previous sections.

- Configure NONMAIN FLASH as required by following Section 3.1.1. Users can also refer to *MSPM33 NONMAIN Flash Memory Configuration Guide*.
- Configure the SAU in the code by following Section 3.2.2. Users can also refer to the example code in the SDK.
- Configure the GSC in the code or SysConfig by following Section 3.2.3. Users can also refer to the example code in the SDK.
- Create NSC function as needed by following Section 3.2.4. Users can refer to the example code in the SDK.

This section introduces how to modify liner command file(.cmd), how to develop secure application, how to set up S/NS IRQ and branch to NS application at the end.

3.2.5.1 Configure .cmd file to assign memory

To enable TrustZone® usage in the project, user should modify linker command file(.cmd) correctly to definite the memory for secure application and NSC functions.

For secure application, user should set up correct range of FLASH and SRAM. Please avoid address overlap between secure application and non-secure application. In the example, default FLASH range is 0x10000000-0x1000F7FF(62KB). Default SRAM range is 0x30000000-0x3000FFFF(64KB).

For NSC function, user should set up one memory for veneer. In the example default NSC_FLASH range is 0x1000F800-0x1000FFFF(2KB)

```
MEMORY
{
    FLASH           (RX)  : origin = 0x10000000, length = 0x0000F800
    NSC_FLASH       (RX)  : origin = end(FLASH), length = 0x00000800, fill = 0xFFFFFFFF,
    SRAM            (RWX) : origin = 0x30000000, length = 0x00010000
    DATA            (R)   : origin = 0x80000000, length = 0x00008000
    BCR_CONFIG      (R)   : origin = 0x80101800, length = 0x00000400
    BSL_CONFIG      (R)   : origin = 0x80101C00, length = 0x00000400
}
```

Figure 3-14. Setting for S region and NSC region

```
Veneer$$CMSE: align(32) {} > NSC_FLASH
```

Figure 3-15. CMSE veneer in NSC region

Note

Address overlap between secure application and non-secure application may lead to security violation behavior or secure data breach. Users should assign address carefully to partition secure and non-secure regions correctly.

3.2.5.2 Secure Application

In the secure application, user can initialize the secure peripherals as needed, and implement secure tasks. Secure application can handle secure resources or non-secure resources.

After properly configuration, secure peripherals can keep working no matter CPU is secure or non-secure state. But non-secure application is not able to access the registers of peripherals because of firewall from truszone.

3.2.5.3 Secure Interrupt and Non-secure Interrupt

Both secure application and non-secure application have its own interrupt vector table, so user can define IRQ handlers for both secure state and non-secure state, and appropriately the linker will place them. Please check '.intvecs' in cmd file of both applications.

Non-secure interrupts can still be serviced when executing a Secure function. Transitions between security states take place automatically during exception handling.

Copyright © 2026 Texas Instruments Incorporated

Armv8-M architecture definite the register ITNS(interrupt target non-secure state) in NVIC, which allows users to define whether the interrupt should target a secure state or non-secure state.

As shown in the Table 3-6, some IRQs are configurable, which means user can configure them to be handled by either secure handler or NS handler. And some of IRQs are banked, which means we can have two different IRQ handlers for S and NS, and set the ITNS register to redirect the interrupt to the appropriate handler.

For exception prioritisation. user may want to guarentee that critical secure exceptions are always given higher priority than NS exceptions. Secure exception priority can be enabled by setting AIRCR.PRIS.

Table 3-6. Configurability for interrupts and exceptions available in an Armv8-M processor

Exception number	Exception type	Type	Default target type	Note
1	Reset	Secure only	Secure	-
2	NMI	Configurable	Secure	Configurable by AIRCR.BFHFMNINS bit. However, AIRCR.BFHFNMINS bit should be set to 1 only when Secure state is unused and calling Secure functions or Secure exceptions should be strictly blocked.
3	HardFault	Configurable	Secure	Configurable by AIRCR.BFHFMNINS bit. However, AIRCR.BFHFNMINS bit should be set to 1 only when Secure state is unused and calling Secure functions or Secure exceptions should be strictly blocked
4	MemManage Fault	Banked	N/A	-
5	BusFault	Configurable	Secure	Configurable by AIRCR.BFHFMNINS bit. However, AIRCR.BFHFNMINS bit should be set to 1 only when Secure state is unused and calling Secure functions or Secure exceptions should be strictly blocked
6	UsageFault	Banked	N/A	-
7	SecureFault	Secure only	Secure	-
11	SVC	Banked	N/A	-
12	Debug monitor	Configurable	Depends on External Debug authentication signals	If Secure debug is enabled in the debug authentication interface, it will target the Secure state, else Debug Monitor exception targets Non-secure state
14	PendSV	Banked	N/A	
15	SysTick	Banked or Configurable	Banked if two SysTick timers are available. If there is only one SysTick timer, by default the SysTick exception targets Secure state.	When one SysTick timer is used, then ICSR.STTNS bit is used to configure the target state of SysTick. Two SysTick timers are always present on Mainline CPUs like Cortex-M33 and greater.
16-495	External interrupts	Configurable	Secure	Using NVIC_ITNS register the interrupts can be configured to either Secure or Non-secure state.

For more information, please refer to the *MSPM33 C3-Series 160MHz Microcontrollers* technical reference manual.

In the example, user can use NVIC_SetTargetState() to set configurable IRQ to non-secure. User can also use NVIC_GetTargetState() to check the IRQ S/NS status.

3.2.5.4 Branch to Non-secure Application

To branch to non-secure application, user should use BLXNS instruction to convert the core state from S to NS.

Copyright © 2026 Texas Instruments Incorporated

In the example, the core transitions from S to NS state via Non Secure Function calls(marked using NS_FUNC in the example), and the branch to the NS code is done by the BLXNS instruction. User can find the definition in sns_helper.h.

```
/* Get non-secure reset handler */
const uint32_t *p_ns_vector_table = (uint32_t *) NON_SECURE_APP0_START;
const uint32_t *p_ns_reset_address =
    (const uint32_t *) ((uint32_t) NON_SECURE_APP0_START +
            sizeof(uint32_t));
nonsecure_func_t p_ns_reset = (nonsecure_func_t)(*p_ns_reset_address);
```

3.3 Develop Non-Secure Application

After branching to non-secure application, the processor state transitions to non-secure as well.

During non-secure application development, typically the following content must be implemented.

- Set up project dependency between the secure application/project and the non-secure application/project, so th transition between secure application and non-secure application can execute as expected, as shown in Section 3.3.1.
- Declare NSC functions to enable function call from NS application to S application, as shown in Section 3.3.1.
- Correctly setting on the memory assignment, as shown in Section 3.3.2.1.
- Develop general tasks/codes in non-secure application, as shown in Section 3.3.2.2.
- Use NSC functions to branch to secure regions for specific secure tasks, as shown in Section 3.3.2.2.

3.3.1 Project Dependency

In the example, non-secure example is dependent on secure example. The following are the steps to configure project dependency in non-secure application project.

1. Set up project dependency in the project Properties-General-Dependencies-Project dependencies

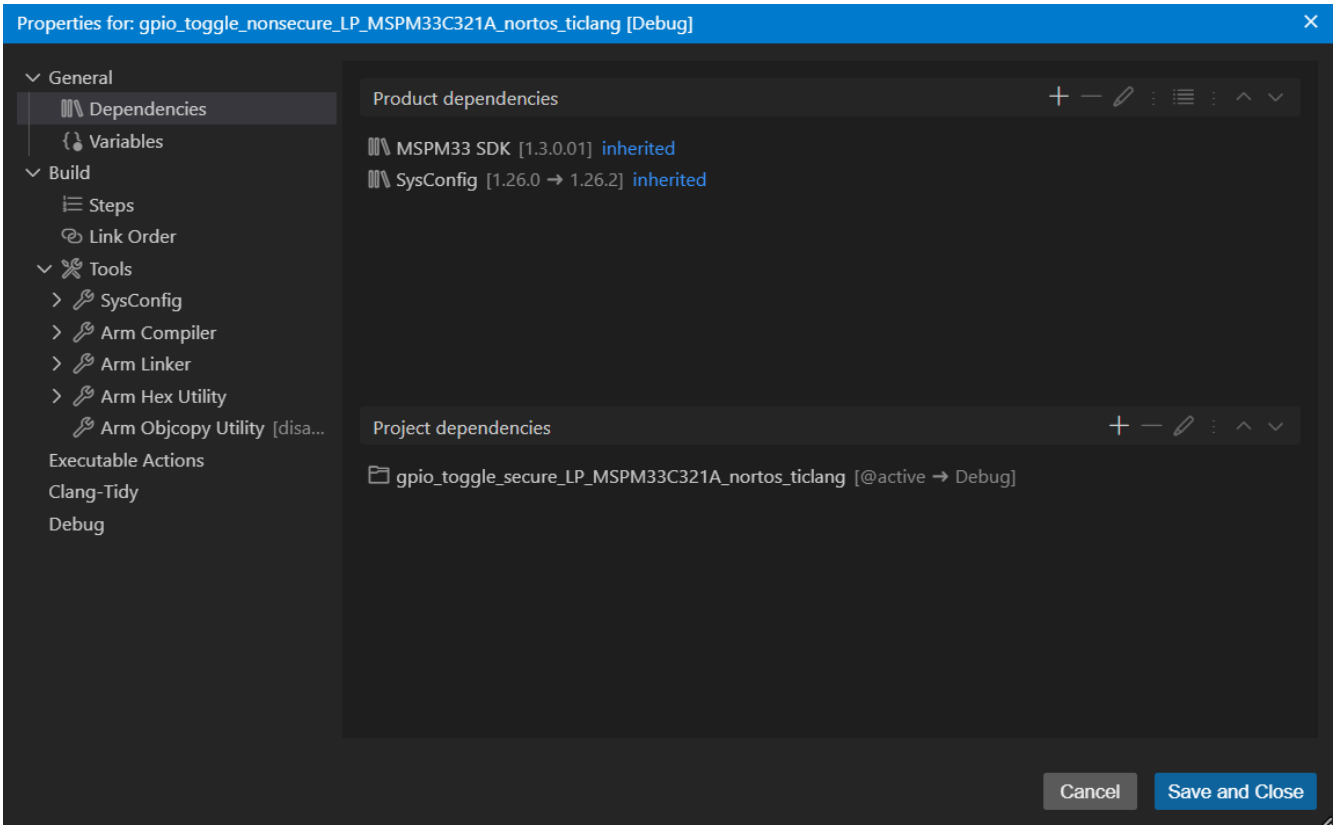


Figure 3-16. Project Dependency Setting

Copyright © 2026 Texas Instruments Incorporated

2. Set up path on CMSE output library from secure application. User can find it in Properties – Build – Tools – Arm Linker – File Search Path – Include library file or command file as input. The following is the example - ${CCS_WORKSPACE_DIR}/empty_secure_LP_MSPM33C321A_nortos_ticlang/Debug/implib.o

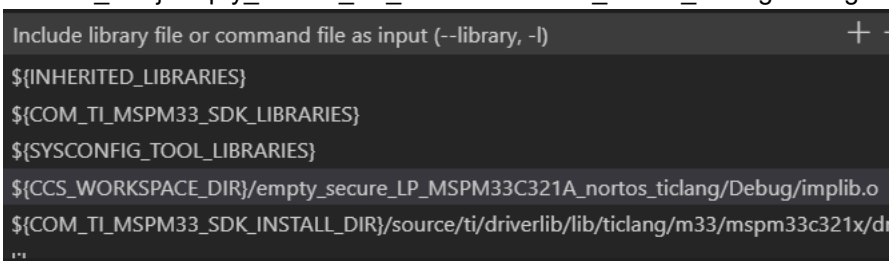


Figure 3-17. Set up Include Path on CMSE output library

3. Declare NSC function in nsc_example.h. It is not required to use NS_FUNC_ENTRY(cmse_nonsecure_entry) here. The following is the example.

```
33    extern void hello_secure();
```

Figure 3-18. Declare NSC function in nsc_example.h

4. Enable Arm Hex Unity to output txt file. In step 5(optional), a post-step command can execute to combine both S and NS txt to a final txt. User can find final txt in debug folder of secure application project. The split txt files or combined txt file can also be used for mass production.

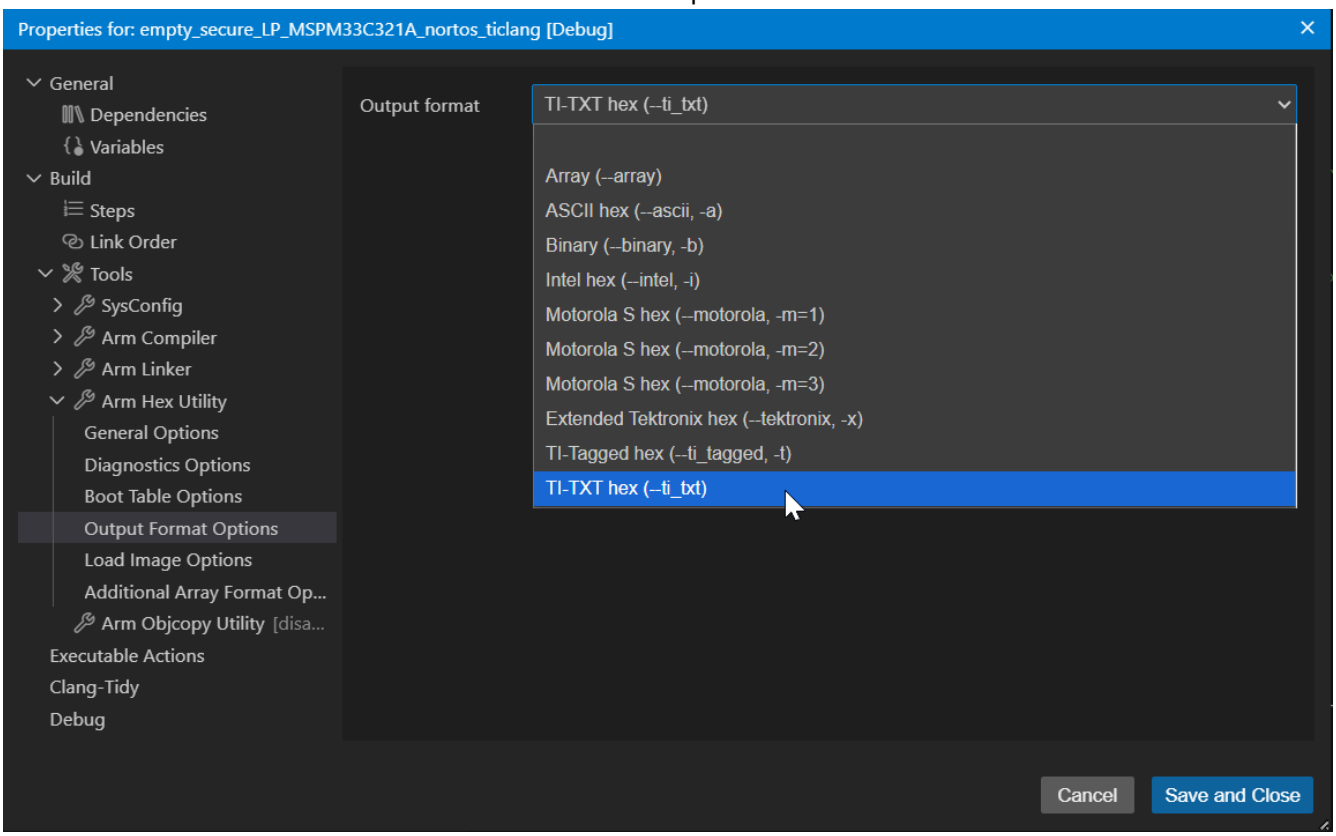


Figure 3-19. Enable TI-TXT Hex Output

5. Configure post-step command to combine secure application image and non-secure application image together. In the non-secure example, user can find in Properties – Build – Steps – Post-build step

```
${NODE_TOOL} ${COM_TI_MSPM33_SDK_INSTALL_DIR}/tools/security/merge_secure_nonsecure.js
${CCS_WORKSPACE_DIR}/empty_secure_LP_MSPM33C321A_nortos_ticlang/Debug/${REF_PROJECT_1_NAME}.txt
${CCS_WORKSPACE_DIR}/empty_nonsecure_LP_MSPM33C321A_nortos_ticlang/Debug/
empty_nonsecure_LP_MSPM33C321A_nortos_ticlang.txt
```


Copyright © 2026 Texas Instruments Incorporated

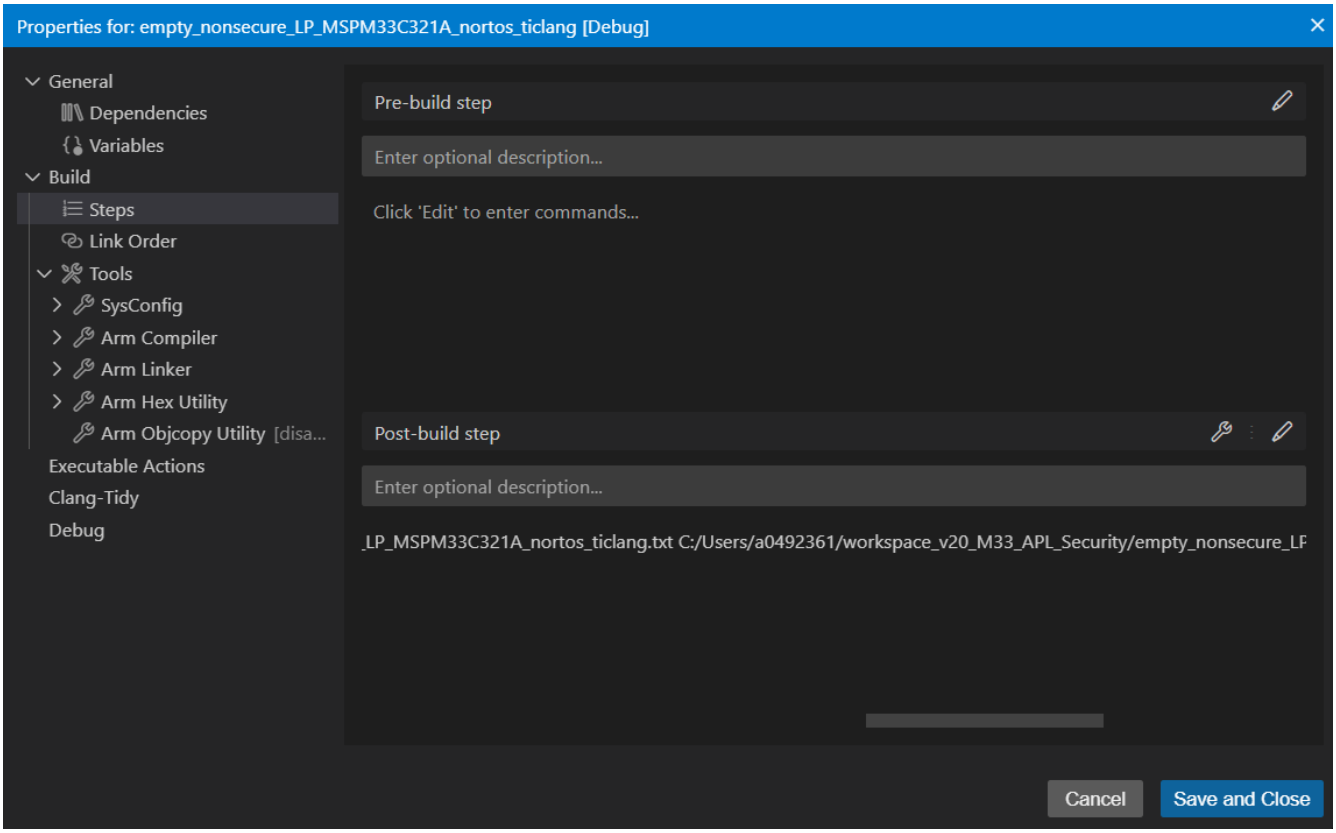


Figure 3-20. Configure post-step command

3.3.1.1 Structure of Non-secure Application Example

Here is the example of empty_nonsecure_LP_MSPM33C321A_nortos_ticlang from SDK.

Table 3-7. Structure of Non-secure Application Example

File	Location	Description
empty_ns.c	${PROJECT_ROOT}	C file for main()
empty.syscfg	${PROJECT_ROOT}	Configuration file for sysconfig
mspm33c321a_ns.cmd	${PROJECT_ROOT}	Link command file to assign memory of nonsecure application
startup_mspm33c321x_ticlang_ns.c	${PROJECT_ROOT}	C file for interrupt vector table of nonsecure application
nsc_example.h	${PROJECT_ROOT}	Header file to declare the definition of non-secure callable(NSC) functions. Related to nsc_example.c in secure example.

Copyright © 2026 Texas Instruments Incorporated

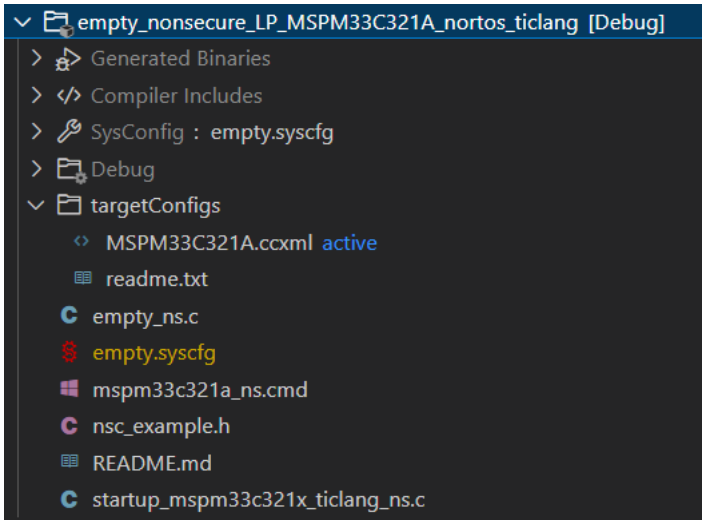


Figure 3-21. Structure of empty_nonsecure_LP_MSPM33C321A_nortos_ticlang

3.3.2 Code Implementation on Non-secure Application

In the non-secure application, the MCU can initialize non-secure peripherals and perform non-secure tasks as needed. When secure tasks are required, it should call to secure entry veneer to branch to secure regions, which are defined as NSC functions in secure applications. Figure 3-13 shows the general flow for secure applications and non-secure applications, which is based on the security examples in the SDK.

Before starting this section, users must configure correctly according to previous sections.

- Configure NONMAIN FLASH as needed by following Section 3.1.1. Users can refer to *MSPM33 NONMAIN Flash Memory Configuration Guide*.
- Create secure application by following Section 3.2, including TrustZone configuration, GSC configuration and NSC function definitions.

This section introduces how to modify liner command file(.cmd), how to develop non-secure applications and how to call NSC functions in NS application to implement transition from NS to S.

3.3.2.1 Configure .cmd file to assign memory

To enable TrustZone® usage in the project, user should modify .cmd file correctly to definite the memory for non-secure application..

For non-secure application, user should set up correct range of FLASH and SRAM. Please avoid address overlap between secure application and non-secure application. In the example, default FLASH range is 0x00010000-0x0001FFFF(64KB). Default SRAM range is 0x20010000-0x2001FFFF(64KB).

```
MEMORY
{
    FLASH           (RX)  : origin = 0x00010000, length = 0x00010000
    SRAM            (RWX) : origin = 0x20010000, length = 0x00010000
}
```

Figure 3-22. Setting for NS region

3.3.2.2 Non-secure Application and NSC Function Call

In the non-secure application, user can initialize the non-secure peripherals as needed, and implement non-secure tasks. Non-secure application can only access to non-secure resources.

To access secure resources, non-secure application needs to call NSC functions, which are defined in secure application and declared by nsc_fucntion.h in non-secure project.

Copyright © 2026 Texas Instruments Incorporated

4 Evaluate TrustZone Based Application

This section introduces how to evaluate secure applications and non-secure applications based on TrustZone examples in SDK.

4.1 Environment Setup

Software and tools:

- MSPM33 SDK: mspm33_sdk_1_03_00_01 (or later version)
- CCS 20.5 (or later version)

Hardware:

- MSPM33C321A samples

Find TrustZone related examples in *MSPM33 SDK\examples\nortos\LP_MSPM33C321A\security*

4.2 Load Secure and Non-secure Images Together

There are two methods to load TrustZone applications, and this section introduces the first method - load combined image.

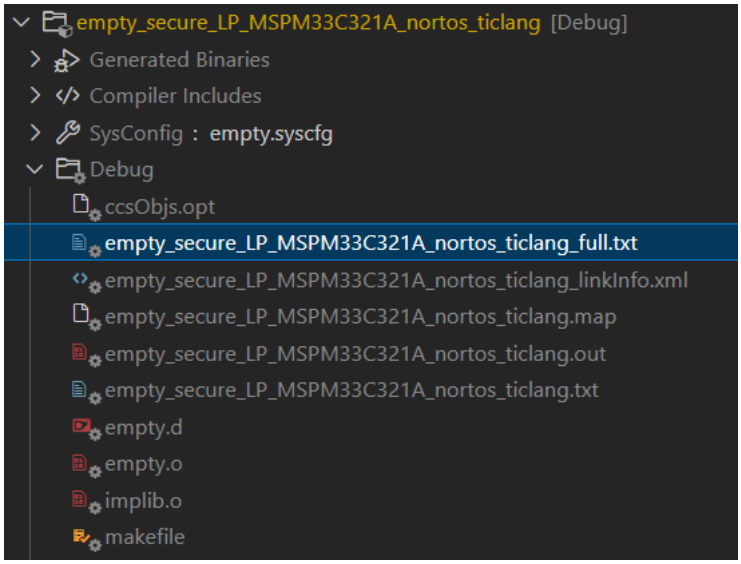
Note

This method currently works only for TIARMCLANG-based examples and is not supported for GCC examples.

4.2.1 Step 1: Build the Non-Secure Application

Before building, import both secure example and non-secure example. Non-secure example have project dependency on corresponding secure example.

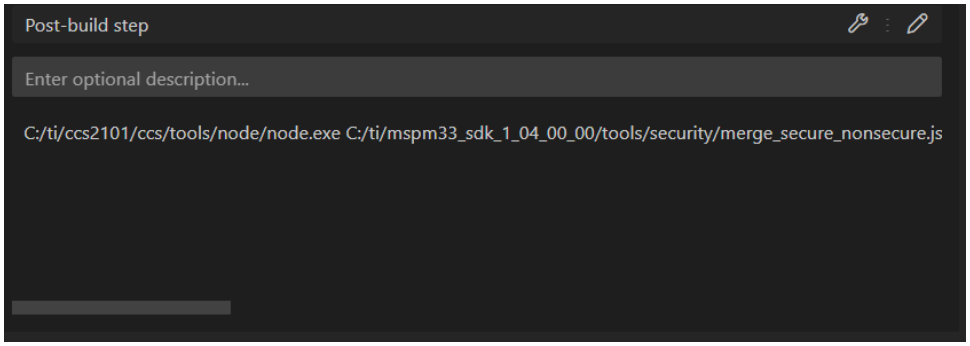
When users build the non-secure application, the secure application is automatically built and a combined <example_name>_full.txt file is generated in the debug folder of the secure application.



The method is based on the post-build step command of non-secure application. Users can locate it in Properties – Build – Steps – Post-build step. Merge_secure_nonsecure.js is executed after building non-secure application.

```
${NODE_TOOL} ${COM_TI_MSPM33_SDK_INSTALL_DIR}/tools/security/merge_secure_nonsecure.js $
{CCS_WORKSPACE_DIR}/${REF_PROJECT_1_NAME}/Debug/${REF_PROJECT_1_NAME}.txt ${CCS_WORKSPACE_DIR}/$
{ProjName}/Debug/${ProjName}.txt
```


Copyright © 2026 Texas Instruments Incorporated

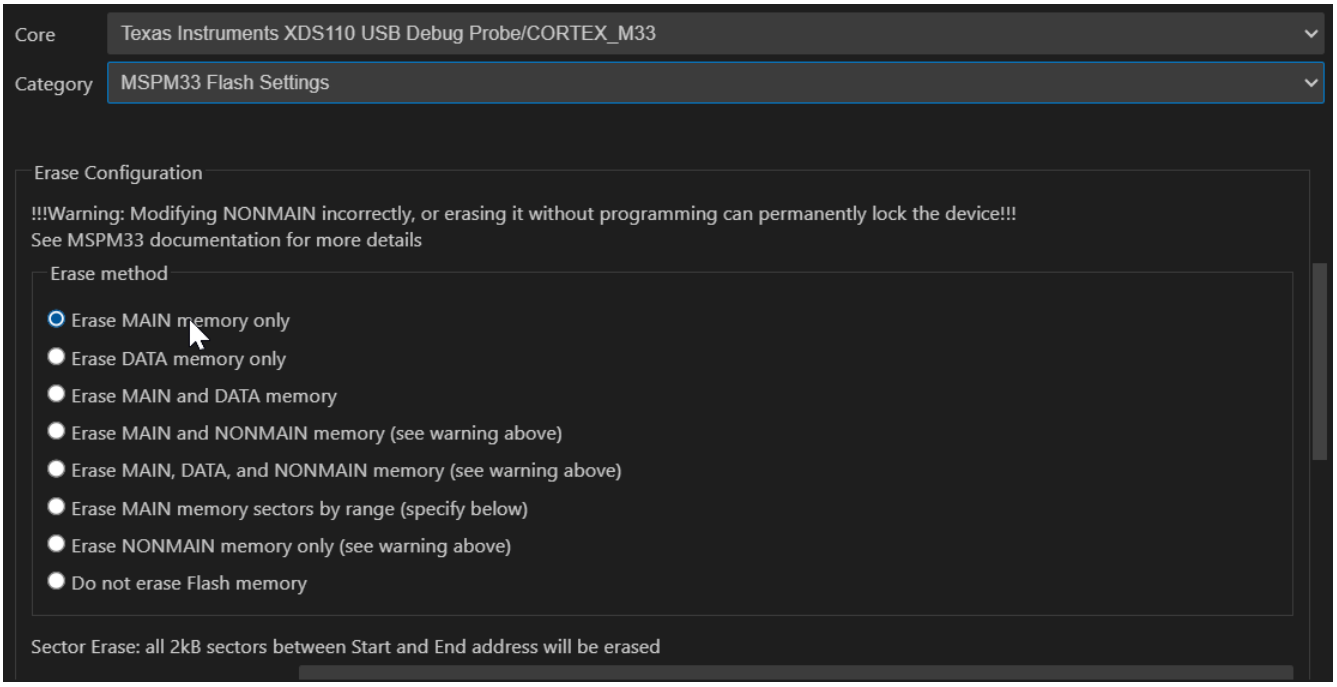


4.2.2 Step 2: Programming the full.txt File

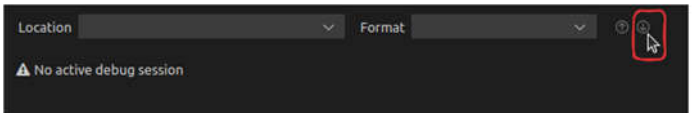
Load the generated <example_name>_full.txt file, which is present in the Debug folder of the Secure application, inside the workspace in CCS. Use the following steps:

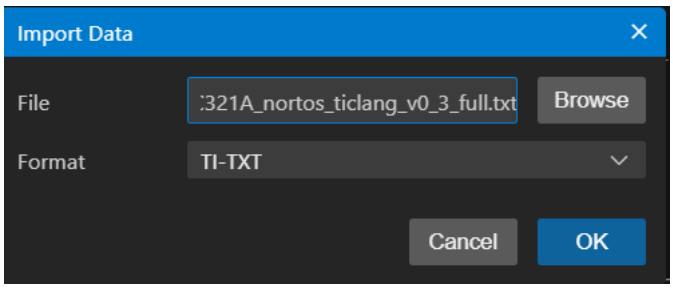
To load the image in CCS, do the following steps:

1. Before flashing the image ensure the erase configuration is set to *Erase MAIN memory only*.



2. To Flash the <example_name>_full.txt into the device, open the memory view (View - Memory) and then import the file. Refer to the following images for the flash operation.





3. After programming, users should find data in 0x10000000(secure image), 0x00010000(non-secure image) and 0x1000F800(NSC region) in memory view.

Copyright © 2026 Texas Instruments Incorporated

Location 0x10000000 Format 32-Bit Hex - TI Style

0x10000000	30020000	1000542D	1000634D	10006349	00000000	10006341	00000000	10006351	00000000	00000000	00000000	10006345
0x10000030	00000000	00000000	10006345	10006345	10006345	10006345	10006345	10006345	00000000	00000000	10006345	10006345
0x10000060	10006345	10006345	10006345	10006345	10006345	10006345	10006345	10006345	10006345	10006345	10006345	10006345
0x10000090	10006345	10006345	10006345	10006345	10006345	10006345	10006345	10006345	10006345	10006345	10006345	10006345
0x100000C0	10006345	10006345	10006345	10006345	10006345	10006345	10006345	10006345	10006345	10006345	10006345	10006345

Location 0x10000 Format 32-Bit Hex - TI Style

0x00010000	20040000	00012125	00012141	00012141	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00012141
0x00010030	00000000	00000000	00012141	00012141	00012141	00012141	00012141	00012141	00000000	00000000	00012141	00012141
0x00010060	00012141	00012141	00012141	00012141	00012141	00012141	00012141	00012141	00012141	00012141	00012141	00012141
0x00010090	00012141	00012141	00012141	00012141	00012141	00012141	00012141	00012141	00012141	00012141	00012141	00012141
0x000100C0	00012141	00012141	00012141	00012141	00012141	00012141	00012141	00012141	00012141	00012141	00012141	00012141

Location 0x1000F800 Format 32-Bit Hex - TI Style

0x1000F800	E97FE97F	B8CEF7F1	E97FE97F	BD26F7F2	E97FE97F	BC72F7F3	E97FE97F	B98CF7F4	E97FE97F	BEB6F7F2	E97FE97F	BA43F7F4
0x1000F830	E97FE97F	BFF6F7F3	E97FE97F	BDF5F7F4	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0x1000F860	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0x1000F890	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0x1000F8C0	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF

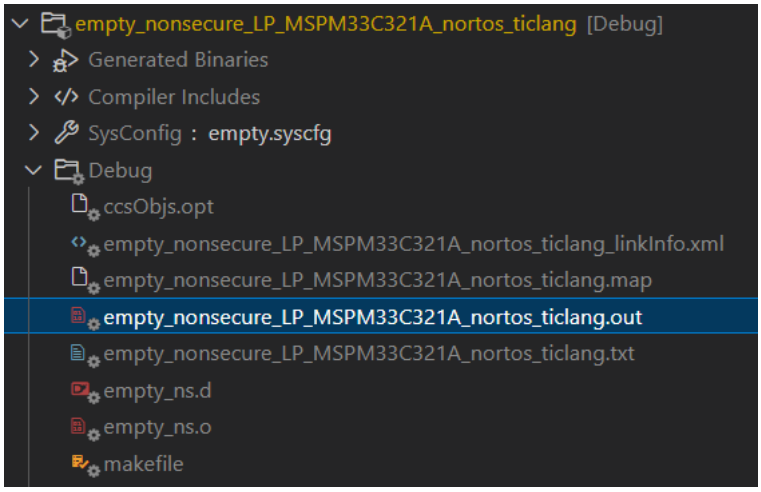
4.3 Load Secure and Non-secure Images separately

There are two methods to load TrustZone applications, and this section introduces the second method: loading images separately.

4.3.1 Step 1: Build the Non-Secure Application

Before building, import both secure examples and non-secure examples. Non-secure examples have project dependency on the corresponding secure example.

When building the non-secure application, locate the generated binaries in the debug folder. The <example_name>.out shall be flashed onto the device.

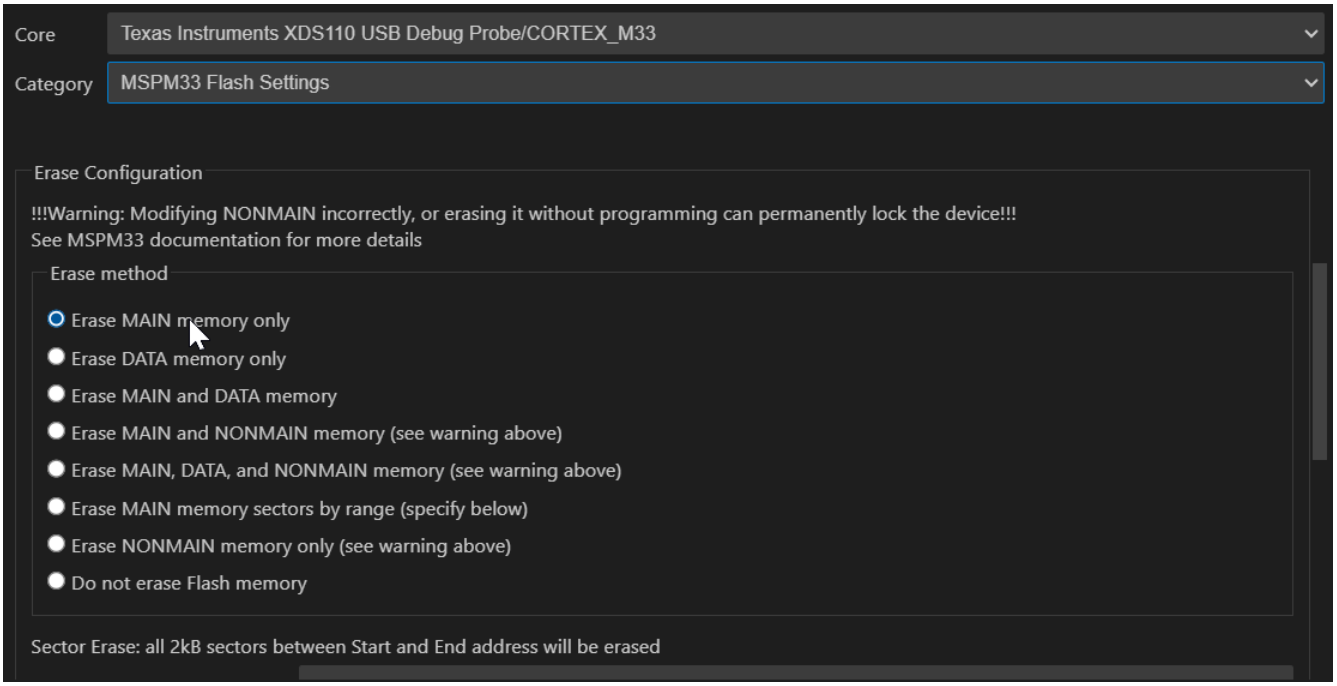


4.3.2 Step 2: Programming the Non-Secure Application

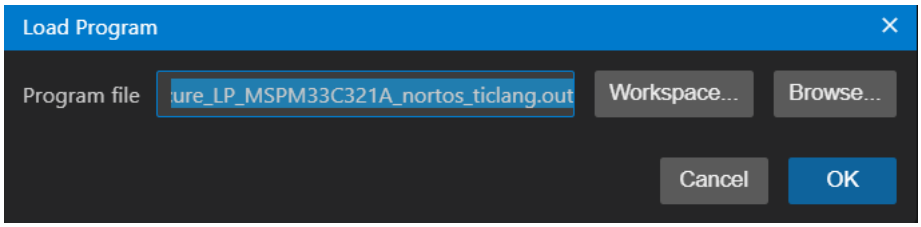
To load the image in CCS, do the following steps:

1. Before flashing the image, make sure the erase configuration is set to *Erase MAIN memory only*.

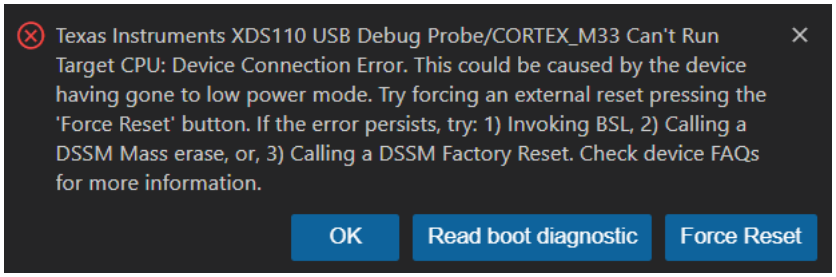
Copyright © 2026 Texas Instruments Incorporated



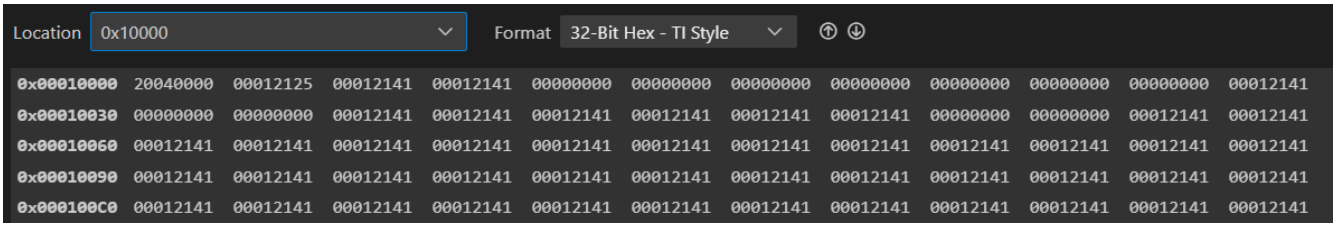
2. Load the binary. To do this select run-> load->load program. Then select the generated out file.



3. After loading the binary, there is an error that Device Connection Error. This is expected as there is no code in 0x0 address. Click *ok* first and try to reconnect the M33 core with *Force Reset*.



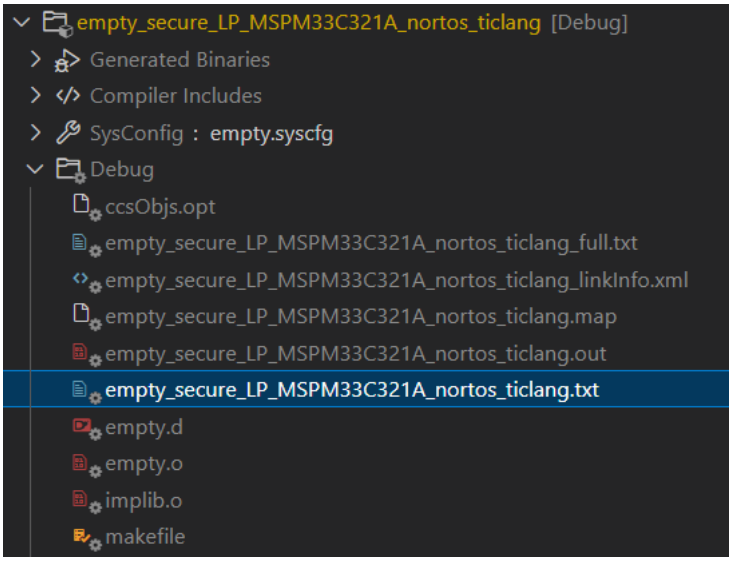
4. After reconnecting to the M33 core, users should find data in 0x00010000(non-secure image) in memory view.



4.3.3 Step 3: Build the Secure Application

When building the secure application, a user can then find generated binaries in the debug folder. The <example_name>.txt shall be flashed onto the device.

Copyright © 2026 Texas Instruments Incorporated



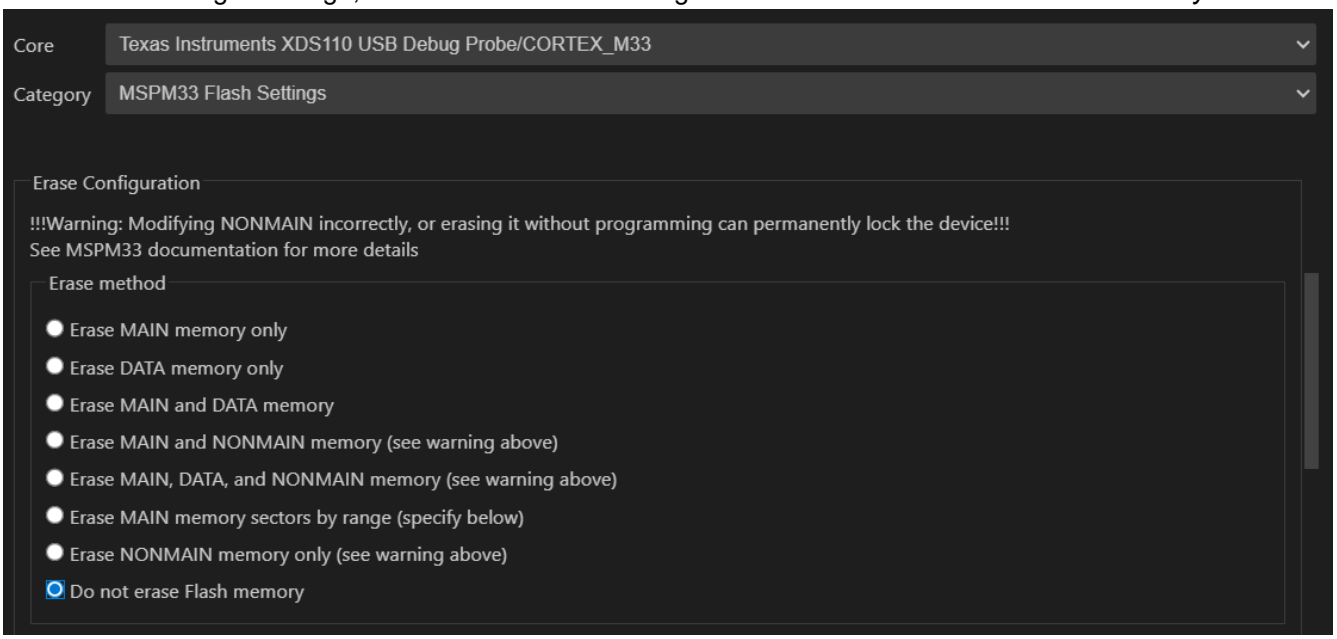
Note

<example_name>_full.txt is generated for method 1 in Section 4.2 . This enables generation in default, but is not used in method 2.

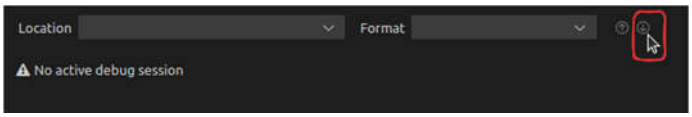
4.3.4 Step 4: Programming the Secure Application

To load the image in CCS, perform the following steps:

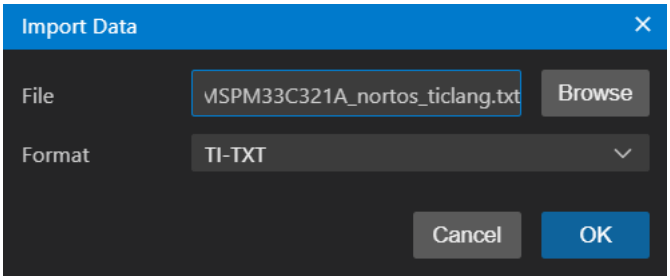
1. Before flashing the image, make sure the erase configuration is set to *Do not erase Flash memory*.



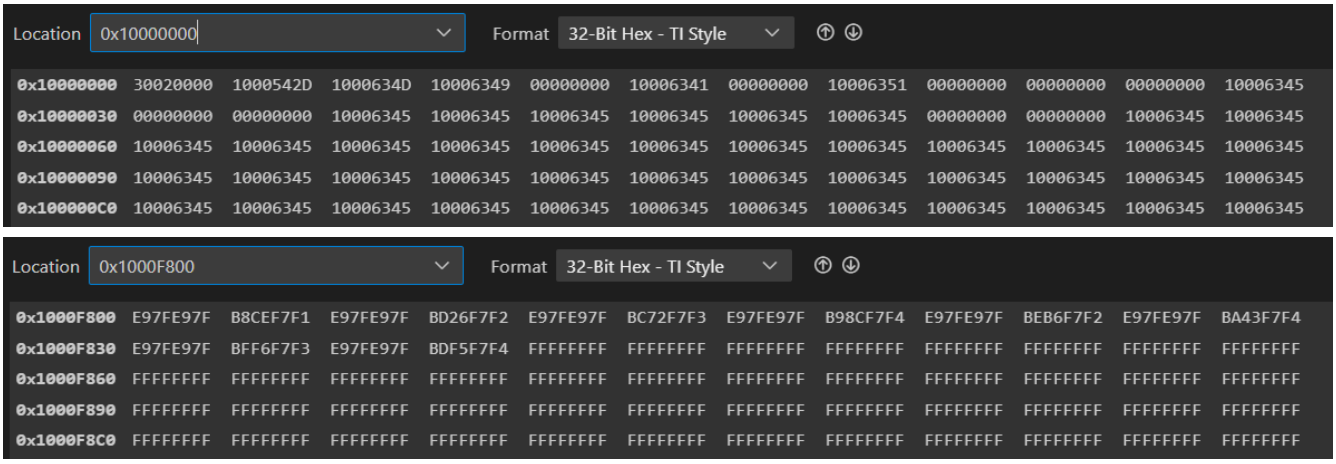
2. To Flash the Demo_secure_spdm_cnsa1_cnsa2_LP_MSPM33C321A_nortos_ticlang_v0_3 into the device, open the memory view (View - Memory) and then import the file. See the following images for the flash operation.



Copyright © 2026 Texas Instruments Incorporated



3. After programming, users should find data in 0x10000000(secure image) and 0x1000F800(NSC region) in memory view.

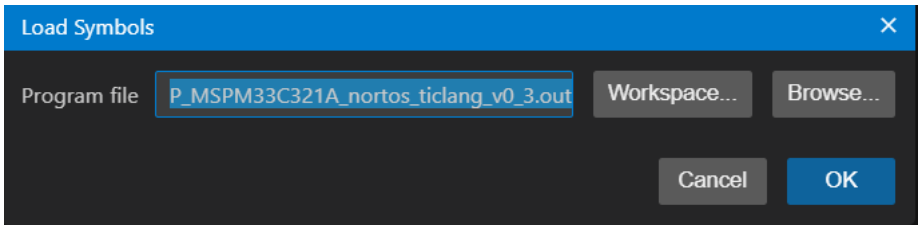


4.4 Debug and Evaluate the Example

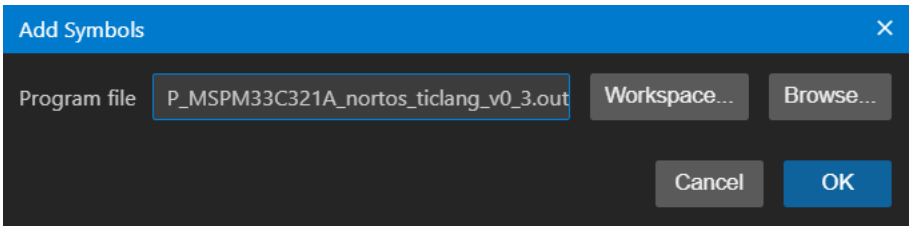
4.4.1 Add Symbols in the Debug

To debug two images together or add breakpoint as need, user should first add symbols.

To add first image's symbol, use Run – Load – Load Symbols



To add second image's symbol, use Run – Load – Add Symbols



4.4.2 Observe Results with Free Run

In the non-secure application, there are two while loops at the end of main(). One for pass condition and one for error condition.

Restart, free run for a while and pause. If the example works as expected, it should pause at the pass condition as shown in the following image.

Copyright © 2026 Texas Instruments Incorporated

```
108     /* Trap here on a pass condition */
109     while (1) {
110         __NOP();
111     }
112
113 err:
114     /* Trap here on an error condition */
115     while (1) {
116         __NOP();
117     }
```

4.4.3 Add Breakpoints to Observe the Process

Users can add breakpoints to secure application or non-secure application as needed. Make sure the related images' symbols have been added in advance by following Section 4.4.1.

Copyright © 2026 Texas Instruments Incorporated

5 FAQ

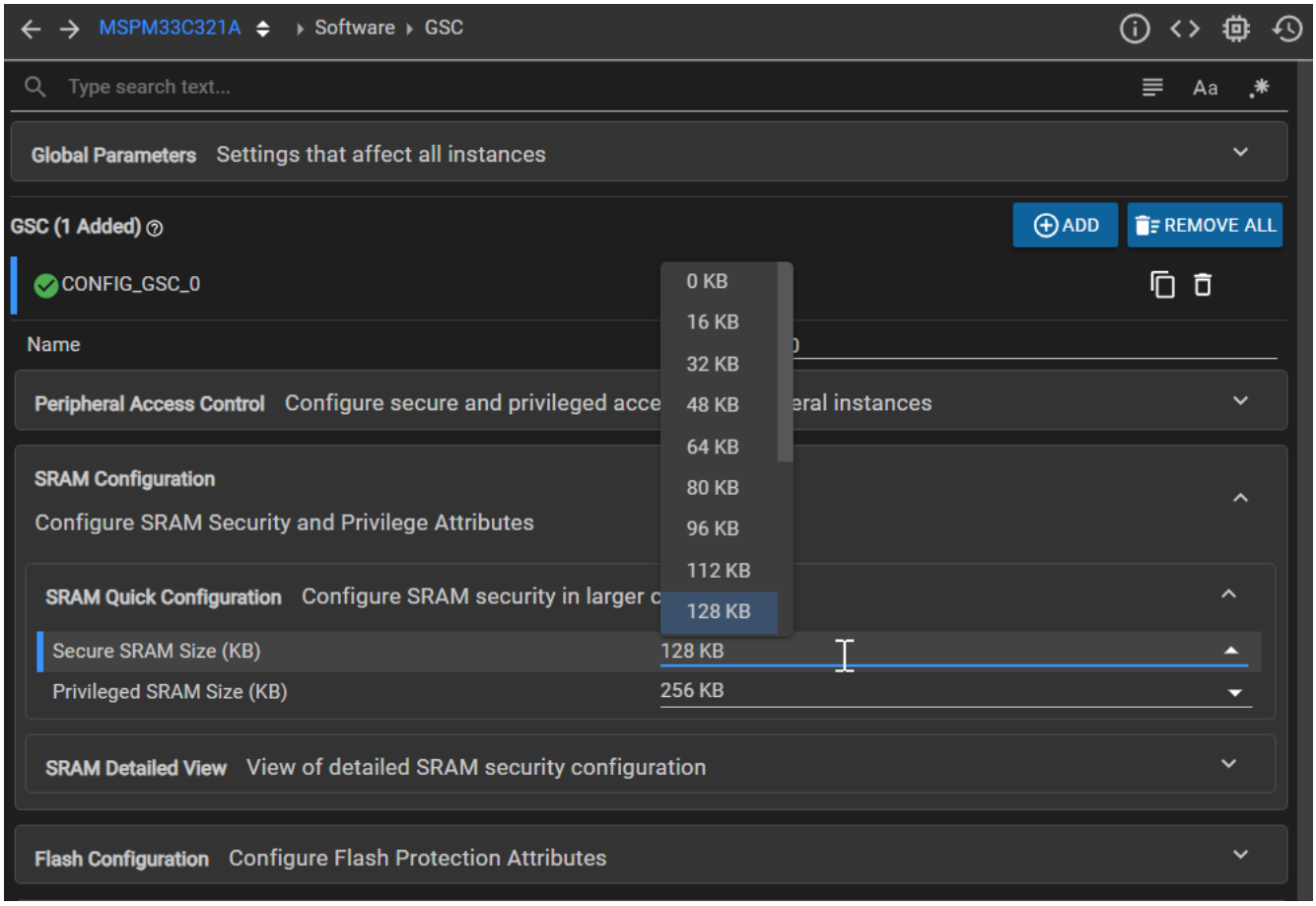
5.1 How to Modify the Range of S/NS SRAM

User may need to definite custom secure or non-secure region in SRAM. This section shows how to modify range of secure or non-secure SRAM in the example showed in Section 4.1.

1. For SAU configuration, user should modify NON_SECURE_APP0_SRAM_START and NON_SECURE_APP0_SRAM_LEN in config.h(in secure application) for non-secure SRAM. Below is the setting for non-secure SRAM from 0x20020000 to 0x2003FFFF. Please refer to Section 3.2.2 for the details on configuration.

```
/* Non-Secure SRAM Configuration */
#define NON_SECURE_APP0_SRAM_START 0x20020000U /* Non-secure SRAM start */
#define NON_SECURE_APP0_SRAM_LEN   0x20000U    /* Non-secure SRAM length (128KB) */
```

2. For GSC configuration, user should modify sysconfig(in secure application). Below is the setting for secure SRAM on first 128KB(0x0-0x1FFFF) and non-secure SRAM on remaining 128KB(0x20000-0x3FFFF)



Note

Typically security attribution setting is same in SAU configuration and GSC configuration. If two configurations are different, please refer to Section 2.2.4 for the final security attribution.

3. Modify cmd file in secure application for secure SRAM. Below is the setting for secure SRAM on first 128KB.

```
37  MEMORY
38  {
39      FLASH          (RX)  : origin = 0x10000000, length = 0x0000F800                  // 30KB Secure APP
40      NSC_FLASH      (RX)  : origin = end(FLASH), length = 0x00000800, fill = 0xFFFFFFFF, // 2KB for NSC Region
41      SRAM           (RWX) : origin = 0x30000000, length = 0x00020000                  // 128KB -> First SRAM Bank for Se
42      DATA           (R)   : origin = 0x80000000, length = 0x00008000
43      BCR_CONFIG     (R)   : origin = 0x80101800, length = 0x00000400
44      BSL_CONFIG     (R)   : origin = 0x80101C00, length = 0x00000400
45  }
```


Copyright © 2026 Texas Instruments Incorporated

4. Modify cmd file in non-secure application for non-secure SRAM. The following image is the setting for non-secure SRAM on the remaining 128KB.

```
37  MEMORY
38  {
39      FLASH           (RX)  : origin = 0x00010000, length = 0x00010000
40      SRAM            (RWX) : origin = 0x20020000, length = 0x00020000
41  }
```

5.2 How to Modify the Range of S/NS FLASH

Users may need to definite custom secure or non-secure region in FLASH. This section shows how to modify range of secure or non-secure FLASH in the example shown in Section 4.1.

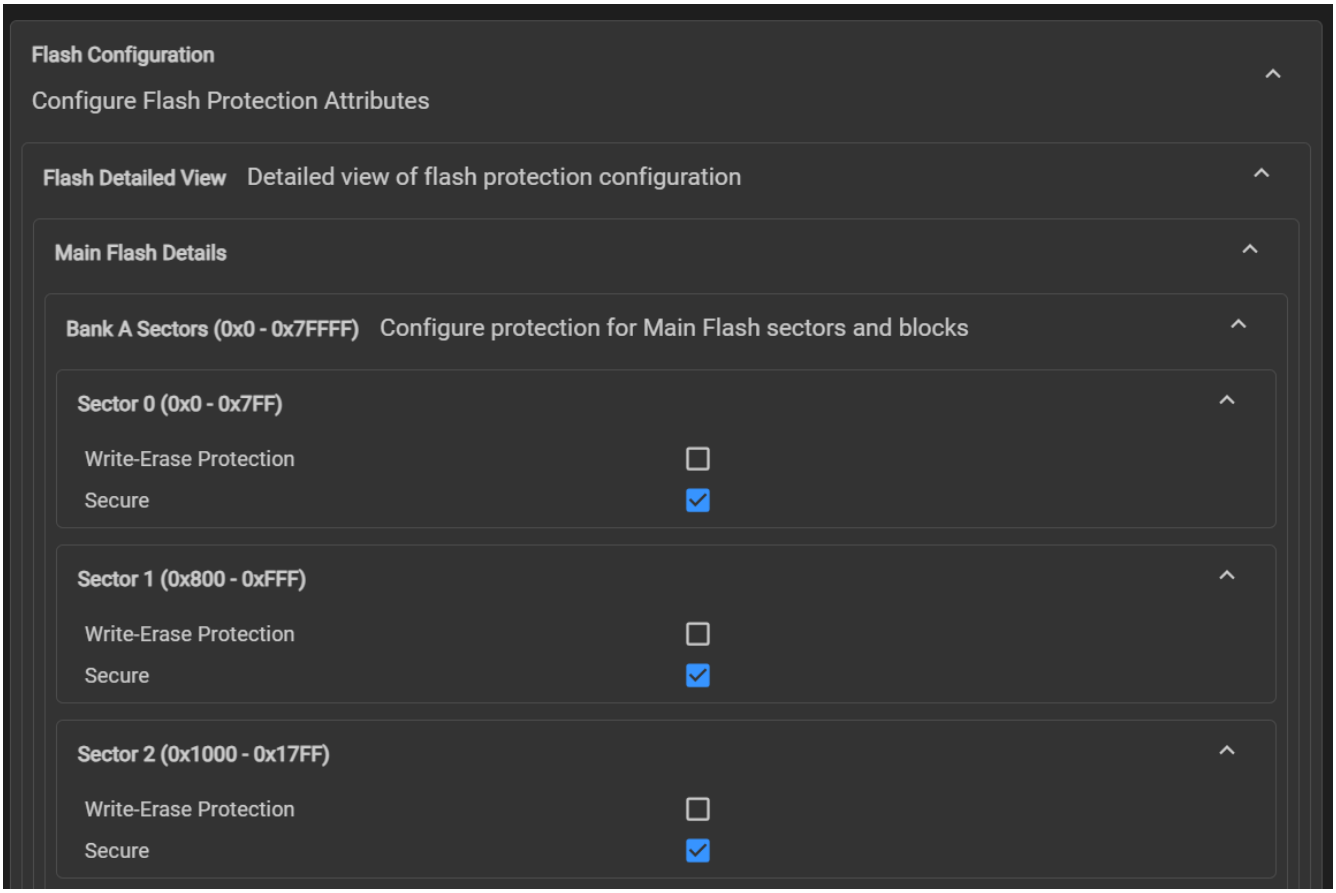
1. For SAU configuration, user should modify NON_SECURE_APP0_START and NON_SECURE_APP0_LEN in config.h(in secure application) for non-secure FLASH. Users should also modify SECURE_NSC_APP0_START and SECURE_NSC_APP0_LEN in config.h(in secure application) for non-secure callable region. The following images show the setting for non-secure FLASH from 0x00010000 to 0x0001FFFF and non-secure callable region from 0x0000F800 to 0x0000FFFF. See Section 3.2.2 for the details on configuration.

```
/* Non-Secure Application Memory Configuration */
#define NON_SECURE_APP0_START      0x10000U /* Non-secure app start address */
#define NON_SECURE_APP0_LEN        0x10000U /* Non-secure app length (64KB) */

/* Secure Non-Secure Callable (NSC) Region Configuration */
#define SECURE_NSC_APP0_START      0xF800U /* NSC region start address */
#define SECURE_NSC_APP0_LEN        0x800U  /* NSC region length (2KB) */
```

2. For GSC configuration, user should modify sysconfig(in secure application). Below is the setting for secure FLASH on first 64KB(0x0-0x0FFFF) and non-secure FLASH on remaining 960KB(0x10000-0x7FFFF).

Copyright © 2026 Texas Instruments Incorporated



Note

Typically security attribution setting is same in SAU configuration and GSC configuration. If two configurations are different, please refer to Section 2.2.4 for the final security attribution.

3. Modify cmd file in secure application for secure FLASH and non-secure callable FLASH. Below is the setting for secure FLASH on first 30KB and non-secure callable FLASH on next 2KB.

```
37  MEMORY
38  {
39      FLASH          (RX)  : origin = 0x10000000, length = 0x0000F800                  // 30KB Secure APP
40      NSC_FLASH      (RX)  : origin = end(FLASH), length = 0x00000800, fill = 0xFFFFFFFF, // 2KB for NSC Region
41      SRAM           (RWX) : origin = 0x30000000, length = 0x00020000                  // 128KB -> First SRAM Bank for Se
42      DATA           (R)   : origin = 0x80000000, length = 0x00008000
43      BCR_CONFIG     (R)   : origin = 0x80101800, length = 0x00000400
44      BSL_CONFIG     (R)   : origin = 0x80101C00, length = 0x00000400
45  }
```

4. Modify cmd file in non-secure application for non-secure FLASH. Below is the setting for non-secure FLASH(starting from 0x10000).

```
37  MEMORY
38  {
39      FLASH          (RX)  : origin = 0x00010000, length = 0x00010000
40      SRAM           (RWX) : origin = 0x20020000, length = 0x00020000
41  }
```

5.3 How to Modify the Security Attribute on Peripherals

Users may need to definite custom secure or non-secure attributes on peripherals. This section shows how to modify security attribute on peripherals in the example showed in Section 4.1.

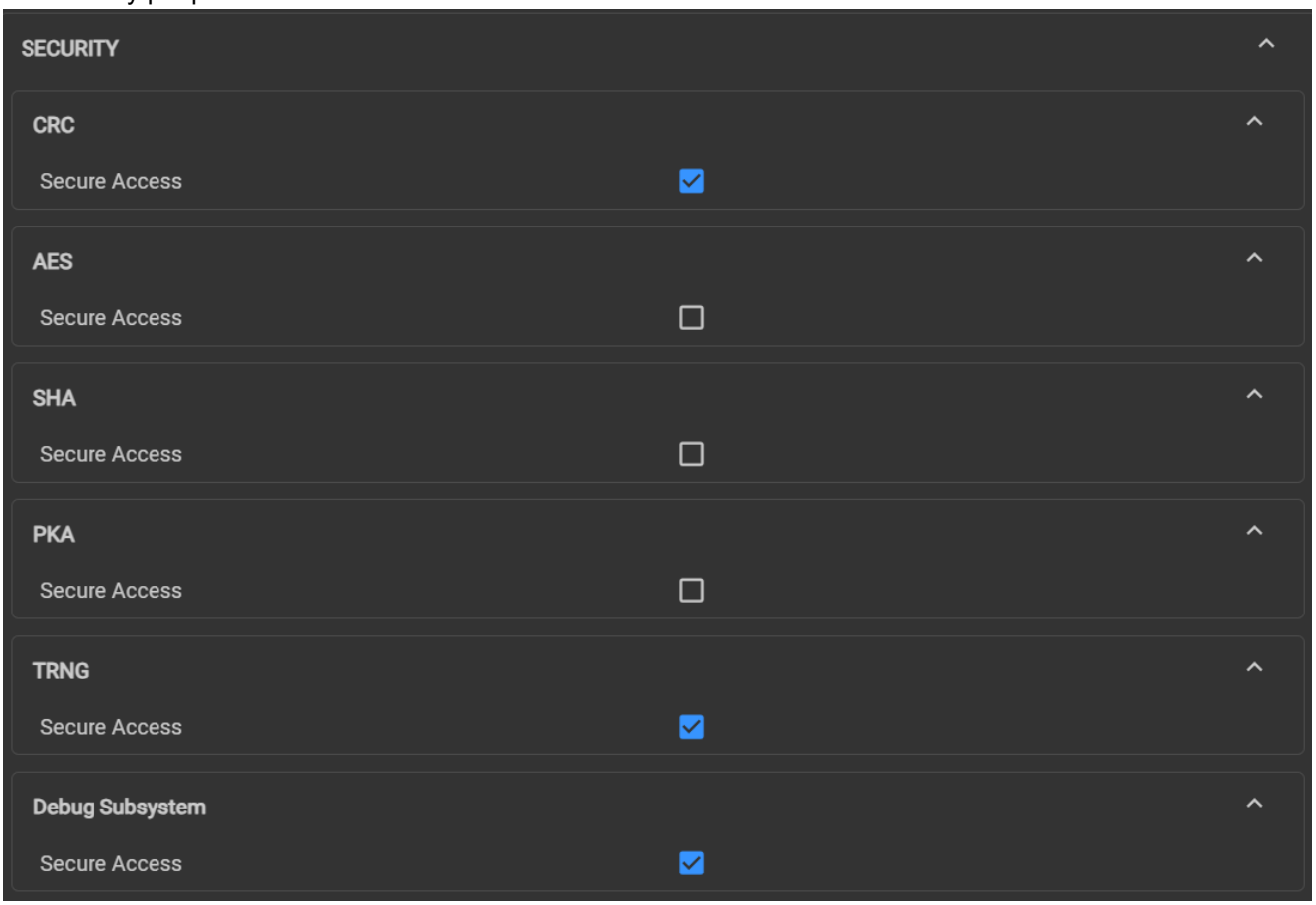
1. For SAU configuration, by default all peripherals are being marked as non-secure in partition.h(in secure application) . This allows the non-secure application to access all peripheral registers. User can also modify this range if needed. See Section 3.2.2 for the details on configuration.

Copyright © 2026 Texas Instruments Incorporated

2. **Note**

   When SAU is NS, security attribute for SAU+IDAU depends on IDAU(see Table 2-2). For peripheral address 0x40000000-0x4FFFFFFF(IDAU is NS), security attribute is NS. For peripheral address 0x40000000-0x4FFFFFFF(IDAU is NSC), security attribute is NSC. Only when SAU+IDAU = S/NSC, bus transaction will be S and can access S region defined by GSC(see Section 2.2.4).

3. For GSC configuration, user should modify sysconfig(in secure application). Below is the setting for some security peripherals.



Note

By default all peripheral address is set to non-secure in SAU configuration. When configuration in GSC is secure, please refer to Section 2.2.4 for the final security attribution.

Copyright © 2026 Texas Instruments Incorporated

6 Summary

Some MSPM33 MCUs supports TrustZone and are able to partition MCU resources to secure regions and non-secure regions. This document shows how to configure TrustZone and global security controller to build a trusted secure environment, and how to develop secure and non-secure application with the correct configuration. This document also shows how to evaluate TrustZone examples provided by TI.

7 References

1. Texas Instruments, *MSPM33 C3-Series 160MHz Microcontrollers*, technical reference manual.
2. Texas Instruments, *MSPM33C Security User's Guide*, user's guide.
3. Arm, *TrustZone® technology for Armv8-M Architecture documentation*, webpage.
4. Arm, *Armv8-M Security Extension User Guide*, user's guide.
5. Texas Instruments, *MSPM33 NONMAIN Flash Memory Configuration Guide*, user's guide.

Copyright © 2026 Texas Instruments Incorporated

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.

Copyright © 2026, Texas Instruments Incorporated

Last updated 10/2025