AN3992

PIC32CM LS00/LS60 Security Reference Guide

Introduction

This document is intended to help the developer to use the PIC32CM LS00/LS60 security features for building secure embedded applications.

The following application development aspects are covered in this document:

- Single and Dual-developer approach
- Secure solution development using the PIC32CM LS00/LS60 ecosystem
- Secure software protection using Arm® TrustZone® for ARMv8-M and Debug Access Levels
- System root of trust using Secure Boot with SHA256-based or HMAC-based authentication for PIC32CM LS00/LS60
- Security standard support with Device Identity Composition Engine (DICE) based on Unique Device Secret (UDS)
- Hardware/Software Cryptographic Accelerator (CRYA)
- System root of trust using Secure Boot with ATECC608B CryptoAuthentication™ Device for PIC32CM LS60

The use of key security features is illustrated using MPLAB Harmony v3 software examples on the following:

- Secure, Non-Secure, and Mix-Secure peripherals
- Data Flash and TrustRAM for storing and protecting application secrets using tamper detection, scrambling, and silent accesses

Table of Contents

1. Prerequisites

Chapter 2 and Chapter 3 of this document describes how to develop or launch an MPLAB Harmony v3-based TrustZone project for the PIC32CM LS00/LS60 Curiosity Pro board. The hardware and software requirements are listed as follows:

Hardware Requirements:

- 1x PIC32CM LS00/LS60 Curiosity Pro board

Software Requirements:

- MPLAB X IDE most up-to-date version
- MPLAB Code Configurator (MCC) for MPLAB Harmony v3 up-to-date version
  - `csp` package
  - `csp_apps_pic32cm_le_ls` package
- Trust Platform Design Suite (TPDS) v2 (PIC32CM LS60 only)

2. Introduction to PIC32CM LS00/LS60 Security Features

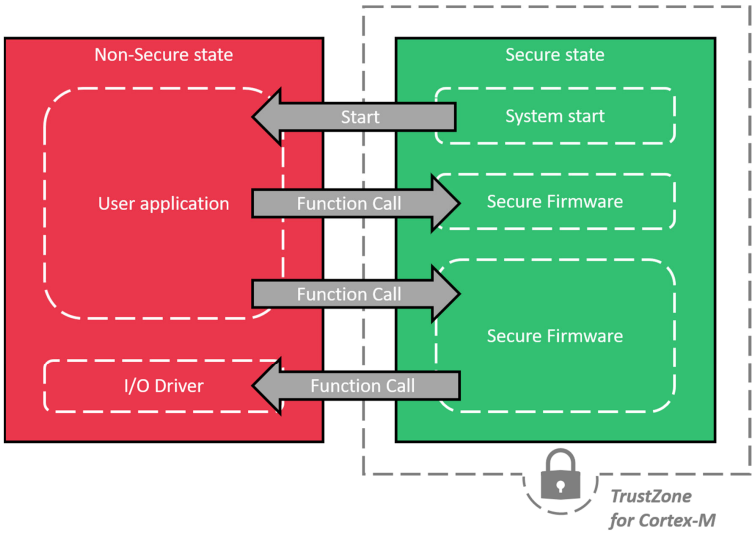
2.1 TrustZone for ARMv8-M

The central security element for the Microchip PIC32CM LS00/LS60 microcontrollers (MCUs) is the implementation of the TrustZone for an ARMv8-M device. The TrustZone technology is a System-on-Chip (SoC) and MCU system-wide approach to security that enables Secure and Non-Secure application code to run on a single MCU.

TrustZone for an ARMv8-M device is based on specific hardware that is implemented in the Cortex-M23 core, which is combined with a dedicated Secure instruction set. It enables creating multiple software security domains that restrict access to selected memory, peripherals, and I/O to trusted software without compromising system performance.

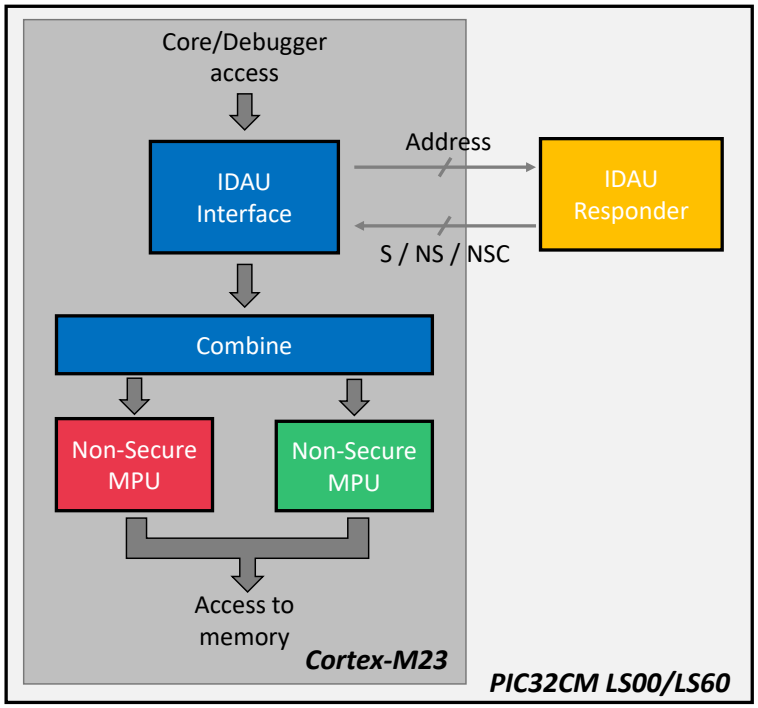
The main goal of the TrustZone for an ARMv8-M device is to simplify the security assessment of a deeply embedded device. The principle behind TrustZone for the ARMv8-M embedded software application is illustrated in the following figure.

Figure 2-1. Standard Interactions Between Secure and Non-Secure States



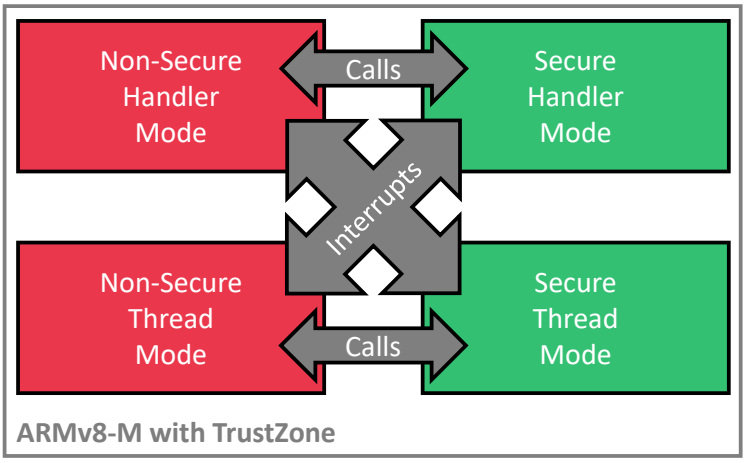
In the PIC32CM LS00/LS60 Cortex-M23 core implementation, the security management is done using the Implementation Defined Attribution Unit (IDAU). The IDAU interface controls the access to the execution of specific instructions which are based on the current core security state and the address of the instruction. The following figure illustrates the Core or Debugger access verification, performed by the system prior to allowing access to specific memory region.

Figure 2-2. IDAU Interface and Memory Accesses



Thanks to this implementation, a simple function call or interrupt processing, results in a path to a specific security state as illustrated in the following figure. This allows efficient calls by avoiding any code and execution overhead.

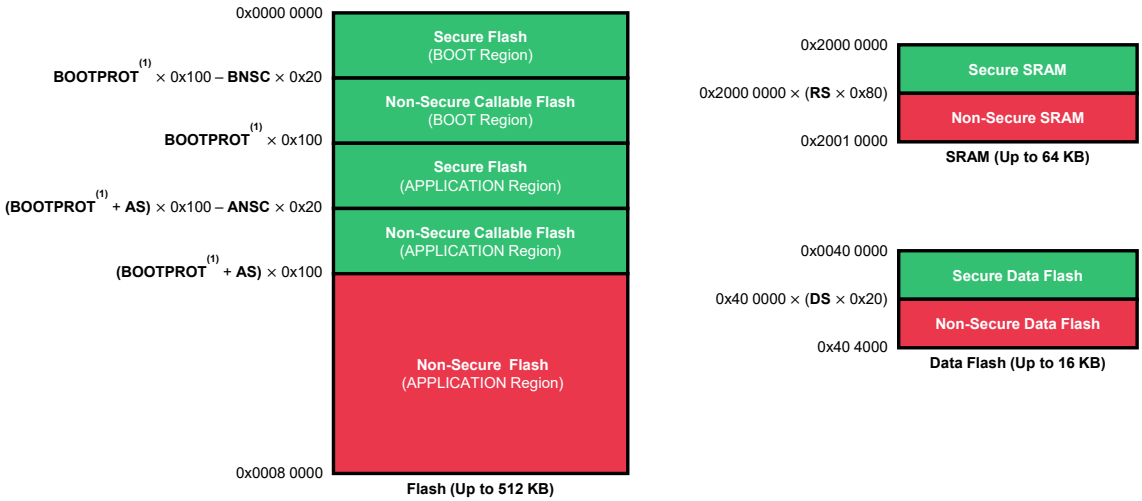
Figure 2-3. ARMv8-M with TrustZone States Transitions



2.1.1 Memory Security Attribution

To differentiate and isolate Secure code from Non-Secure code, the PIC32CM LS00/LS60 device family is partitioned with up to nine memory regions as illustrated in the following figure. Each region size is configurable using dedicated NVM Configuration bit fields, such as BNSC, BOOTPROT, AS, ANSC, DS, and RS.

Figure 2-4. PIC32CM LS00/LS60 Memory Mapping



Notes:

1. BOOTPROT = BS.
2. All the NVM Configuration bit field acronyms, shown in the figure above, are defined in the Glossary.

Each memory region is preconfigured in the hardware with one of the following attributes:

- **Secure (S):** Used for memory and peripherals, which are accessible only by secure software.
- **Non-Secure Callable (NSC):** A special type of secure memory location. It enables software transition from a Non-Secure to a Secure state.
- **Non-Secure (NS):** Used for memory and peripherals, which are accessible by all software running on the device.

The security attribute of each region will define the security state of the code stored in this region.

2.1.2 Secure and Non-Secure Function Call Mechanism

To prevent Secure code and data being accessed from a Non-Secure state, the Secure code must meet several requirements. The responsibility for meeting these requirements is shared between the MCU architecture, software architecture, and the toolchain configuration.

At the core level, a set of Secure instructions dedicated to ARMv8-M devices are used to preserve and protect the Secure register values during the CPU security state transition.

- **Secure Gateway (SG):** Used for switching from a Non-Secure to a Secure state at the first instruction of a Secure entry point
- **Branch with eXchange to Non-Secure state (BXNS):** Used by the Secure software to branch or return to the Non-Secure program
- **Branch with Link and eXchange to Non-Secure State (BLXNS):** Used by the Secure software to call the Non-Secure functions

At the toolchain level, a ‘C’ language extension (CMSE) provided by Arm must be used to ensure the use of ARMv8-M Secure instruction.

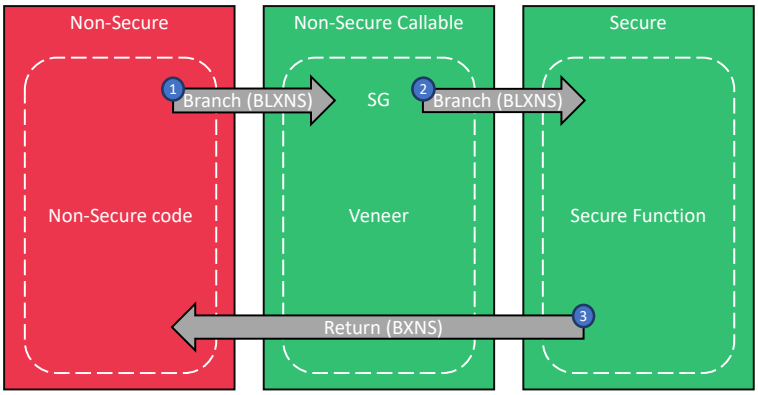
At the software architecture level, specific Secure and Non-Secure function call mechanisms must be used to ensure security, which are described in the following sections.

2.1.2.1 Non-Secure Callable APIs

When working with TrustZone for ARMv8-M, the application developer can define a set of Non-Secure Callable APIs which can be used to access the Secure code from the Non-Secure world. These APIs, known as Secure Gateways (SG) or veneers oversee the CPU security state switch, and allow the decoupling of Secure entry points from the rest of the Secure code. Therefore, they limit the amount of code that can be potentially accessed by the non-secure state.

SG are expected to be placed in the NSC memory regions, which are executable only when the CPU is in the non-secure state. The rest of the secure code is expected to be placed in the Secure memory regions, which are not accessible when the CPU is in the Non-Secure state as shown in the following figure:

Figure 2-5. Non-Secure Callable APIs Mechanism



Using Non-Secure Callable APIs requires the use of specific Cortex-M23 instructions that ensure security during the core security state switching. A direct API function call from the Non-Secure to the Secure software entry points is allowed only if the first instruction of the entry points is an SG and is in a Non-Secure callable memory location. The use of the special instructions (BXNS and BLXNS) are required to branch to Non-Secure code.

The following code illustrates a Secure function and its SG API declaration and definition using an XC32 toolchain with a 'C' language extension (CMSE):

`Nonsecure_entry.h`

```
/* Non-secure callable functions */
extern int nsc_func1(int x);
```

`Nonsecure_entry.c` (linked in the NSC memory region of the device):

```
/* Non-secure callable (entry) functions */
int __attribute__((cmse_nonsecure_entry)) nsc_func1 (int x)
{
    return secure_func1 (x);
}
```

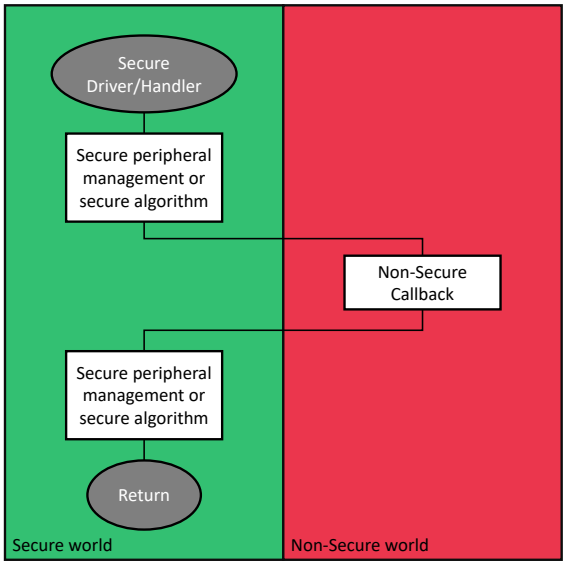
`Secure_function.c` (linked in the secure memory region of the device):

```
int secure_func1 (int x)
{
    return x + 3;
}
```

2.1.2.2 Non-Secure Software Callbacks

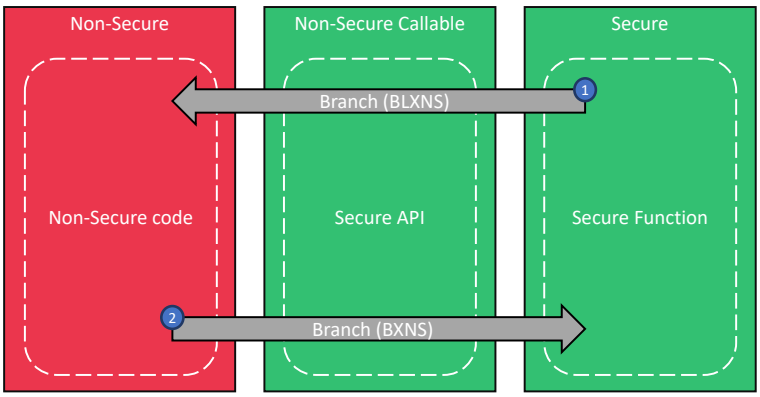
The Secure code can define and use software callbacks to execute functions from the Non-Secure world. This is a consequence of separating Secure and Non-Secure code into separate executable files. The following figure shows the software callback approach:

Figure 2-6. Non-Secure Software Callbacks Flow Chart



The management of callback functions can be performed using the BLXNS instruction. The following figure illustrates the Non-Secure callback mechanism:

Figure 2-7. Non-Secure Software Callback Mechanism

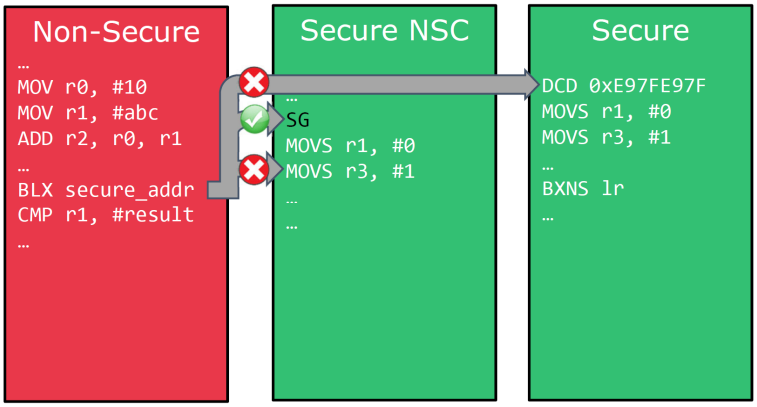


Note: The definition of Non-Secure software callback is done through a pointer to a Non-Secure code location. If not correctly checked in the Secure application, a wrong use of pointers can lead to a security weakness that enables the execution of any Secure functions by the Non-Secure code. To overcome these disadvantages, a set of CMSE functions based on the new Cortex-M23 Test Target (TT) instructions is provided.

2.1.2.3 Security State and Call Mismatch

Any attempts to access Secure regions from the Non-Secure code, or a mismatch between the code that is executed and the security state of the system results in a Hard Fault exception, as shown in the following figure:

Figure 2-8. Security State and Call Mismatch



2.1.3 Secure and Non-Secure Interrupts Handling

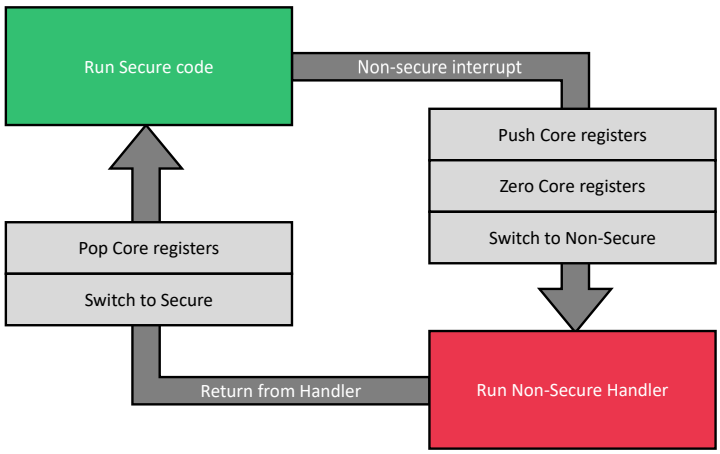
The Cortex-M23 (ARMv8-M architecture) uses the same exception stacking mechanism as the ARMv7-M architecture, where a subset of core registers is stored automatically into the stack (hardware context saving). This permits immediate execution of the interrupt handler without the need to perform a context save in the software. The ARMv8-M extends this mechanism to provide enhanced security based on two different stack pointers: a Secure stack pointer and a Non-Secure stack pointer.

According to the priority settings configured in the Nested Vector Interrupt Controller (NVIC), the Secure code can interrupt the Non-Secure code execution, and the Non-Secure code can interrupt the Secure code execution. The NVIC registers at the core level are duplicated. This allows two vector table definitions: one for Secure, and another for Non-Secure.

At product start-up, all interrupts are mapped by default to the Secure world (Secure vector table). Specific CMSIS functions accessible in the Secure world, allocate each interrupt vector to a Non-Secure handler (declared in the Non-Secure vector table).

As illustrated in the following figure, if the Secure code is running when a higher priority Non-Secure interrupt arrives, the core pushes all its register content into a dedicated Secure stack. Registers are then zeroed automatically to prevent any information from being read, and the core executes the Non-Secure exception handler. When the Non-Secure handler execution is finished, the hardware recovers all the registers from the Secure stack automatically. This mechanism is managed in hardware and does not require any software intervention. This allows a Secure handover from running Secure code to a Non-Secure interrupt handler, and returning to running Secure code.

Figure 2-9. Cortex-M23 Interrupt Mechanism



2.2 Peripherals Security Attribution

The PIC32CM LS00/LS60 family of devices extends the concept of TrustZone to its integrated peripherals and offers the possibility to allocate a specific peripheral to the Secure and Non-Secure world. The PIC32CM LS00/LS60 also embeds peripherals that can share their resources between Secure and Non-Secure applications called Mix-Secure peripherals. The management of each peripheral security attribution is done through the Peripheral Access Controller (PAC).

Note: The IDAU peripheral is always Secure and the DSU (Device Service Unit) peripheral is always Non-Secure. Refer to the “PIC32CM LE00/LS00/LS60 Family Data Sheet” for additional information.

2.2.1 Secure and Non-Secure Peripherals

In the following figure, the PAC controller embeds a set of registers that define the security attribution of each integrated peripheral of the system. These registers are configured at device startup by the ROM code which sets the PAC.NONSECx registers according to the user configuration stored in the User Row (UROW) fuses.

Figure 2-10. PAC NONSECx Registers Description

NONSECA	31:24								
	23:16								
	15:8	Reserved	Reserved	AC	PORT	FREQM	EIC	RTC	WDT
	7:0	GCLK	SUPC	OSC32KCTRL	OSCCTRL	RSTC	MCLK	PM	PAC
NONSECB	31:24								
	23:16								
	15:8								
	7:0			USB	HMATRIXHS	DMAC	NVMCTRL	DSU	IDAU
NONSECC	31:24								
	23:16			TRAM	OPAMP	I2S	CCL	TRNG	PTC
	15:8	DAC	ADC	TCC3	TCC2	TCC1	TCC0	TC2	TC1
	7:0	TC0	SERCOM5	SERCOM4	SERCOM3	SERCOM2	SERCOM1	SERCOM0	EVSYS

Important: The peripherals security attribution cannot be changed by accessing the PAC.NONSECx registers during application run-time unless the SECCFGLOCK bit is cleared before exiting the Boot ROM. Refer to Security Configuration Lock Bit (SECCFGLOCK) for more information. Any changes must be done using the User Row fuses and require a reset of the PIC32CM LS00/LS60 device. The application can read the PAC.NONSECx register to get the current attribution of integrated peripherals.

Peripherals can be categorized into two groups depending on their PAC security attribution and their internal secure partitioning capabilities (standard/mix-secure):

- **Secure peripheral:** A standard peripheral is configured as Secure in the PAC. The security attribution of the whole peripheral is defined by the associated NONSECx fuse set to zero. Secure accesses to the peripheral are granted where Non-Secure accesses are discarded (Write is ignored, Read 0x0) and a PAC error is triggered.
- **Non-Secure peripheral:** A standard peripheral is configured as Non-Secure in the PAC. The security attribution of the whole peripheral is defined by the associated NONSECx fuse set to one. Secure and Non-Secure accesses to the peripheral are granted.

When a peripheral is allocated to the Secure world, only secure accesses to its registers are granted, and interrupt handling should be managed in the Secure world only.

2.2.2 Mix-Secure Integrated Peripherals

The PIC32CM LS00/LS60 family of devices embed five Mix-Secure peripherals, which allow part of their internal resources to be shared between the Secure and Non-Secure applications. A complete list of the PIC32CM LS00/LS60 Mix-Secure peripherals and their resources are as follows:

- **Peripheral Access Controller (PAC):** Manages the peripherals security attribution (Secure or Non-Secure).
- **Non-Volatile Memory Controller (NVMCTRL):** Handles the Secure and Non-Secure Flash regions programming.
- **I/O Controller (PORT):** Supports individual allocation of each I/O to the Secure or Non-Secure applications.
- **External Interrupt Controller (EIC):** Supports individual assignment of each external interrupt to the Secure or Non-Secure applications.
- **Event System (EVSYS):** Supports individual assignment of each event channel to the Secure or Non-Secure applications.

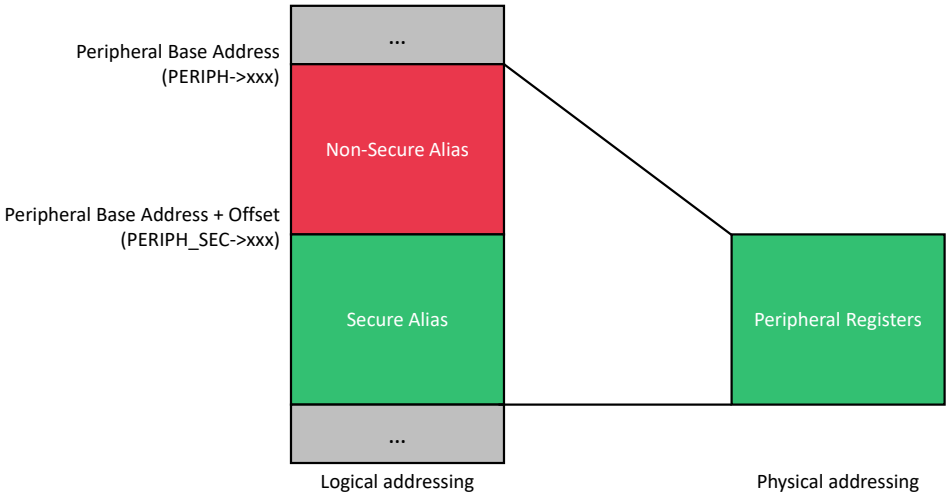
The capability for a Mix-Secure peripheral to share its internal resources depends on the security attribution of that peripheral in the PAC peripheral (PAC Secured or PAC Non-Secured):

- When a Mix-Secure peripheral is secured (NONSECx fuse set to zero), the Secure application can allocate internal peripheral resources to the Non-Secure application using dedicated registers
- When a Mix-Secure peripheral is Non-Secured (NONSECx fuse set to one), the peripheral behaves as a standard Non-Secure peripheral. Secure and Non-Secure accesses to the peripheral register are granted.

2.2.2.1 Mix-Secure Peripheral (PAC Secured)

When a Mix-Secure peripheral is PAC Secured (associated PAC NONSECx fuses set to zero), the peripheral is banked and accessible through two different memory aliases, as shown in the following figure:

Figure 2-11. PAC Secured Mix-Secure Peripheral Registers Addressing



The Secure world can then independently enable non-secure access to the internal peripheral resources using the NONSEC register, as shown in the following figure for the External Interrupt Controller:

Figure 2-12. External Interrupt Controller NONSEC Register

Name: NONSEC
Offset: 0x40
Reset: 0x00000000
Property: PAC Write-Protection, Write-Secure

This register allows to set the NMI or external interrupt control and status registers in non-secure mode, individually per interrupt pin.

Important: This register is only available for **PIC32CM LS00/LS60** and has no effect for **PIC32CM LE00**.

Bit	31	30	29	28	27	26	25	24
	NMI							
Access	RW/R/RW							
Reset	0							

Bit	23	22	21	20	19	18	17	16
Access								
Reset								

Bit	15	14	13	12	11	10	9	8
	EXTINT15	EXTINT14	EXTINT13	EXTINT12	EXTINT11	EXTINT10	EXTINT9	EXTINT8
Access	RW/R/RW	RW/R/RW	RW/R/RW	RW/R/RW	RW/R/RW	RW/R/RW	RW/R/RW	RW/R/RW
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	EXTINT7	EXTINT6	EXTINT5	EXTINT4	EXTINT3	EXTINT2	EXTINT1	EXTINT0
Access	RW/R/RW	RW/R/RW	RW/R/RW	RW/R/RW	RW/R/RW	RW/R/RW	RW/R/RW	RW/R/RW
Reset	0	0	0	0	0	0	0	0

The NONSEC register content can only be modified by the Secure world through the peripheral register secure alias (PERIPH_SEC.NONSEC).

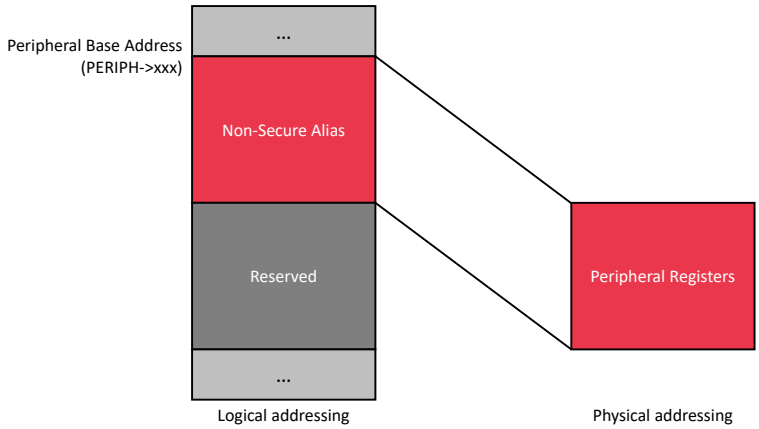
Setting a specific internal feature bitfield in the NONSEC register, enables access to the different bitfields associated to this feature in the peripheral non-secure alias.

2.2.2.2 Mix-Secure Peripheral (PAC Non-Secured)

When a Mix-Secure peripheral is PAC Non-Secured (associated NONSECx fuses set to one), the peripheral behaves as a standard Non-Secure peripheral.

Secure and Non-Secure accesses to the peripheral register are granted. The peripheral register mapping is shown in the following figure:

Figure 2-13. PAC Non-Secured Mix-Secure Peripheral Registers Addressing



Managing PAC Non-Secured (Mix-Secured) peripherals at the application level is like managing a standard Non-Secure peripheral.

2.3 Security Configuration Lock Bit (SECCFGLOCK)

The Security Configuration Lock bit is a bit from the BOCOR row that allows the modification of the security configurations during the application execution by programming the different IDAU, PAC, and NVMCTRL peripheral registers.

After exiting the Boot ROM:

- **If SECCFGLOCK = 1:**
  - The security configurations are locked so that no code (even Secure) can change them before next reset sequence.
  - The only way to update the security configurations is to reprogram the NVM Configuration rows then reset the device.
- **If SECCFGLOCK = 0:**
  - The security configurations can be modified during the application execution.
  - It is possible to update the security configurations by reprogramming the NVM Configuration rows, then resetting the device.

The SECCFGLOCK = 0 configuration brings added value to the Secure software code running from the Flash BOOT region compared to the one running from the Flash APPLICATION region, as it is possible to exit the Boot ROM without locking the security configuration bits.

Therefore, the Secure software code of the Flash BOOT region will have the responsibility to lock the security configuration before passing control to the Secure software code of the Flash APPLICATION region.

⚠ CAUTION If BOCOR.SECCFGLOCK = 0, to guarantee the security of the overall application, it is critical that the Secure software code of the BOOT region locks all the IDAU/PAC/NVMCTRL security configuration registers and restores the Debug Access Level configuration.

Refer to the Boot ROM section in the “PIC32CM LE00/LS00/LS60 Family Data Sheet” for additional information.

2.4 Debug Access Level (DAL) and Chip Erase

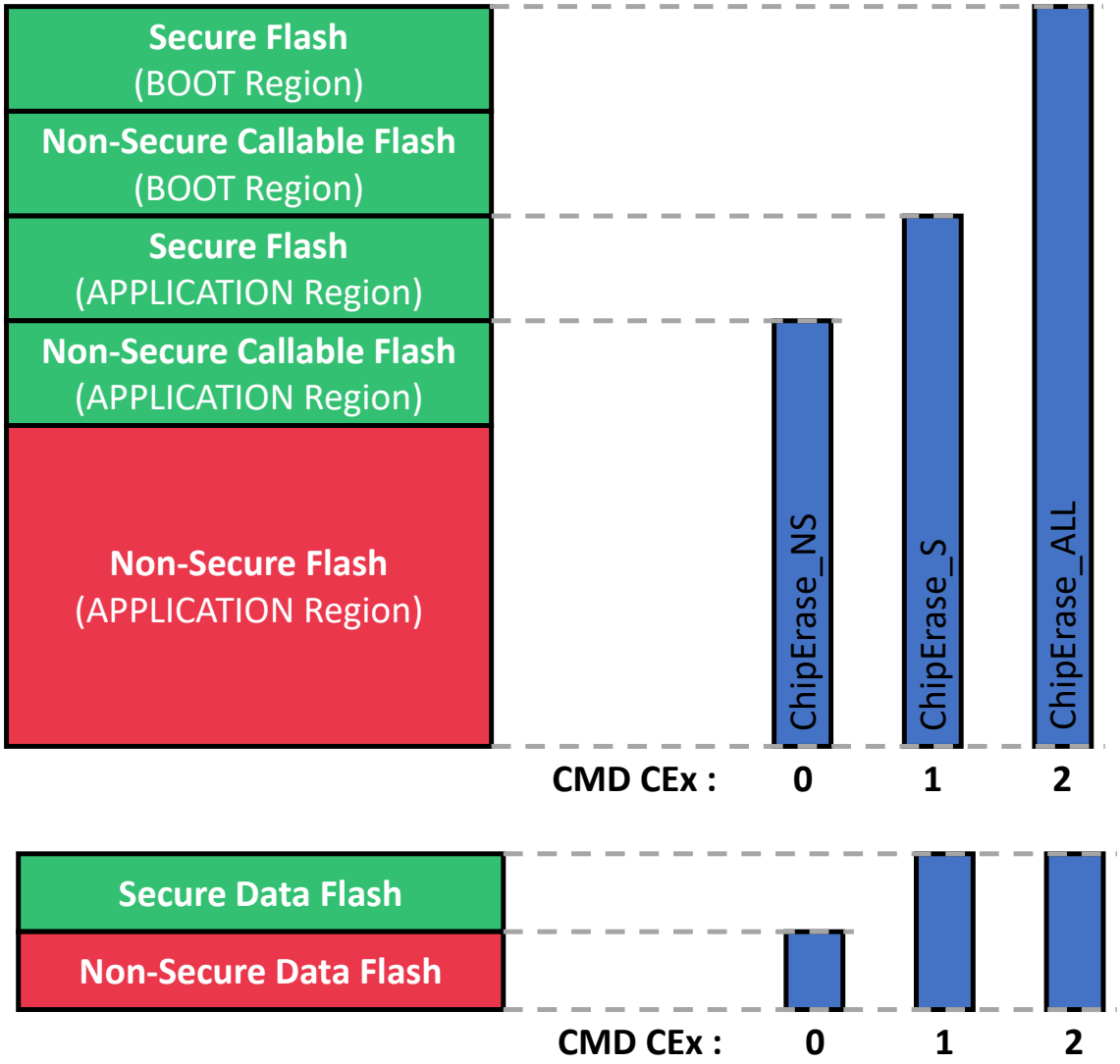
The PIC32CM LS00/LS60 family of devices have the following configurable Debug Access Levels (DAL), which restrict programming and debug access to the Secure and Non-Secure resources in the system.

- **DAL2:** Highest debug access level with no restrictions in term of memory and peripheral accesses.
- **DAL1:** Access is limited to the Non-Secure memory regions. Secure memory region accesses are forbidden.
- **DAL0:** No access is authorized except with a debugger using the Boot ROM Interactive Mode.

Note: For additional information on Boot ROM Interactive Mode, refer to the 'Boot ROM' chapter in the "PIC32CM LE00/LS00/LS60 Family Data Sheet".

The DAL is combined with three-key protected `ChipErase` commands, which provide three levels of Non-Volatile Memory erase granularity as shown in the following figure:

Figure 2-14. ChipErase Commands



The configuration of the `ChipErase` command protection key is done through the BOCOR bit field configuration, as shown in the following tables:

Table 2-1. PIC32CM LS00 BOCOR Mapping

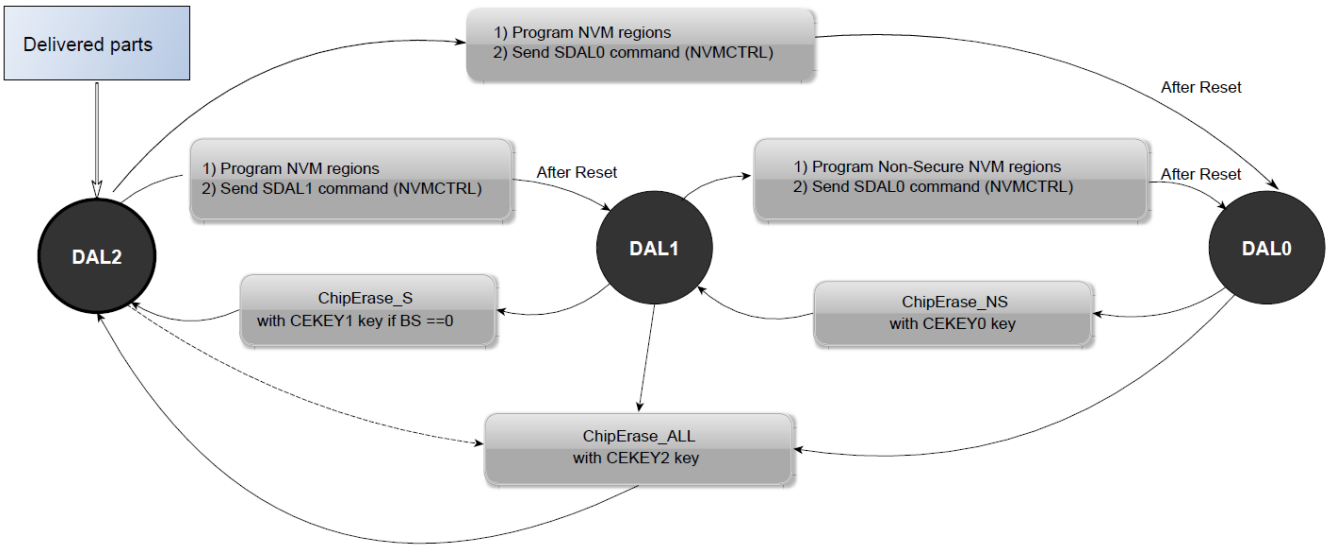
Offset	Bit Pos.	Name					
0x00-0x1	15:0	Reserved					
0x02	23:16	BNSC				Reserved	
0x03	31:24	Reserved			BNSC		
0x04	39:32	BOOTOPT					
0x05	47:40	BOOTPROT					
0x06	55:48	Reserved		DICEEN	SECCFGLOCK	BOOTPROT	
0x07	63:56	Reserved				BCREN	BCWEN
0x08-0x0B	95:64	BOCORCRC					
0x0C-0x0F	127:96	Reserved					
0x10-0x1F	255:128	CEKEY0					
0x20-0x2F	383:256	CEKEY1					
0x30-0x3F	511:384	CEKEY2					
0x40-0x4F	639:512	CRCKEY					
0x50-0x6F	895:640	BOOTKEY					
0x70-0x8F	1151:896	UDS					
0x90-0xDF	1791:1152	Reserved					
0xE0-0xFF	2047:1792	BOCORHASH					

Table 2-2. PIC32CM LS60 BOCOR Mapping

Offset	Bit Pos.	Name					
0x00-0x1	15:0	Reserved					
0x02	23:16	BNSC				Reserved	
0x03	31:24	Reserved			BNSC		
0x04	39:32	BOOTOPT					
0x05	47:40	BOOTPROT					
0x06	55:48	Reserved		DICEEN	SECCFGLOCK	BOOTPROT	
0x07	63:56	Reserved				BCREN	BCWEN
0x08-0x0B	95:64	BOCORCRC					
0x0C-0x0F	127:96	Reserved					
0x10-0x1F	255:128	CEKEY0					
0x20-0x2F	383:256	CEKEY1					
0x30-0x3F	511:384	CEKEY2					
0x40-0x4F	639:512	CRCKEY					
0x50-0x6F	895:640	BOOTKEY					
0x70-0x8F	1151:896	UDS					
0x90-0xAF	1407:1152	IOPROTKEY					
0xB0-0xDF	1791:1408	Reserved					
0xE0-0xFF	2047:1792	BOCORHASH					

The different `ChipErase` commands are used to increase the DAL level without compromising the code security. Therefore, erase the code before changing to a higher DAL level, as illustrated in the following figure:

Figure 2-15. PIC32CM LS00/LS60 DAL and ChipErase Mechanism

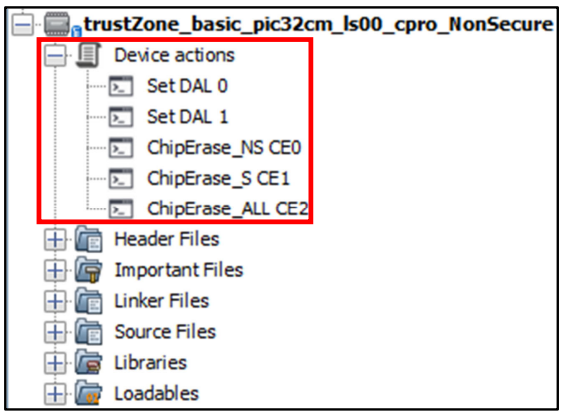


Note:

1. BS = BOOTPROT for PIC32CM LS00/LS60 family of devices.

The MPLAB® X IDE provides an easy method to set the DAL and `ChipErase` commands.

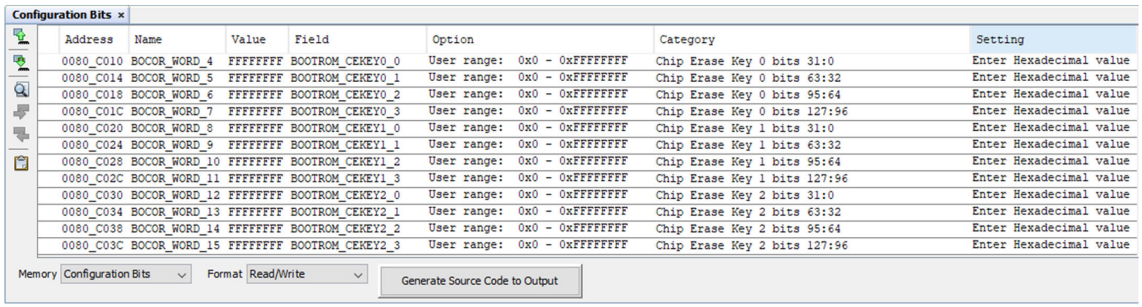
Figure 2-16. ChipErase Commands Under MPLAB X IDE Project Tree



Note: This feature is available when the PIC32CM LS00/LS60 project is opened in MPLAB X IDE.

The PIC32CM LS00/LS60 `ChipErase` key fuses are also available in the Configuration Bits window provided by the MPLAB X IDE by opening *Toolbar* > *Window* > *Target Memory Views* > *Configuration Bits:*

Figure 2-17. ChipErase Key Fuses Setting Under MPLAB X IDE Configuration Bits


Configuration Bits ×

Address	Name	Value	Field	Option	Category	Setting
0080_C010	BOCOR_WORD_4	FFFFFFFF	BOOTROM_CEKEY0_0	User range: 0x0 - 0xFFFFFFFF	Chip Erase Key 0 bits 31:0	Enter Hexadecimal value
0080_C014	BOCOR_WORD_5	FFFFFFFF	BOOTROM_CEKEY0_1	User range: 0x0 - 0xFFFFFFFF	Chip Erase Key 0 bits 63:32	Enter Hexadecimal value
0080_C018	BOCOR_WORD_6	FFFFFFFF	BOOTROM_CEKEY0_2	User range: 0x0 - 0xFFFFFFFF	Chip Erase Key 0 bits 95:64	Enter Hexadecimal value
0080_C01C	BOCOR_WORD_7	FFFFFFFF	BOOTROM_CEKEY0_3	User range: 0x0 - 0xFFFFFFFF	Chip Erase Key 0 bits 127:96	Enter Hexadecimal value
0080_C020	BOCOR_WORD_8	FFFFFFFF	BOOTROM_CEKEY1_0	User range: 0x0 - 0xFFFFFFFF	Chip Erase Key 1 bits 31:0	Enter Hexadecimal value
0080_C024	BOCOR_WORD_9	FFFFFFFF	BOOTROM_CEKEY1_1	User range: 0x0 - 0xFFFFFFFF	Chip Erase Key 1 bits 63:32	Enter Hexadecimal value
0080_C028	BOCOR_WORD_10	FFFFFFFF	BOOTROM_CEKEY1_2	User range: 0x0 - 0xFFFFFFFF	Chip Erase Key 1 bits 95:64	Enter Hexadecimal value
0080_C02C	BOCOR_WORD_11	FFFFFFFF	BOOTROM_CEKEY1_3	User range: 0x0 - 0xFFFFFFFF	Chip Erase Key 1 bits 127:96	Enter Hexadecimal value
0080_C030	BOCOR_WORD_12	FFFFFFFF	BOOTROM_CEKEY2_0	User range: 0x0 - 0xFFFFFFFF	Chip Erase Key 2 bits 31:0	Enter Hexadecimal value
0080_C034	BOCOR_WORD_13	FFFFFFFF	BOOTROM_CEKEY2_1	User range: 0x0 - 0xFFFFFFFF	Chip Erase Key 2 bits 63:32	Enter Hexadecimal value
0080_C038	BOCOR_WORD_14	FFFFFFFF	BOOTROM_CEKEY2_2	User range: 0x0 - 0xFFFFFFFF	Chip Erase Key 2 bits 95:64	Enter Hexadecimal value
0080_C03C	BOCOR_WORD_15	FFFFFFFF	BOOTROM_CEKEY2_3	User range: 0x0 - 0xFFFFFFFF	Chip Erase Key 2 bits 127:96	Enter Hexadecimal value

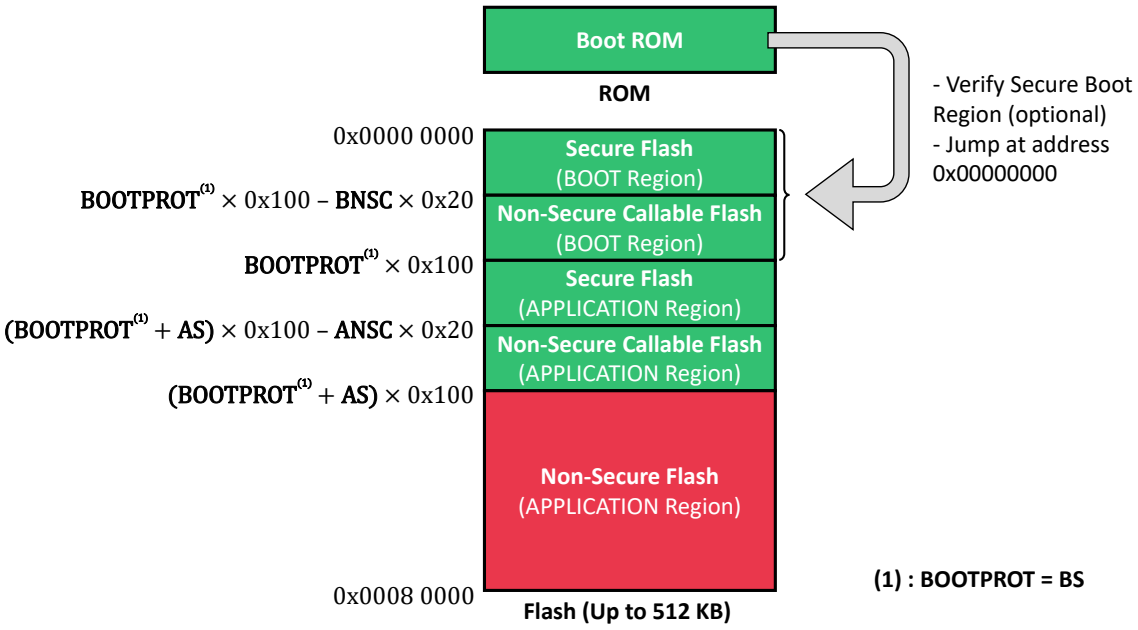
Memory: Configuration Bits Format: Read/Write Generate Source Code to Output

2.5 Secure Boot

The PIC32CM LS00/LS60 Boot ROM is always executed at product startup. This software is ROM coded into the device and cannot be bypassed by the user. Depending on the Boot Configuration Row (BOCOR) fuse setting, the Boot ROM knows if a Secure Boot region is defined in the system.

The Boot ROM can perform an integrity check (SHA-256) or authenticate (SHA-256 + BOOTKEY or HMAC + BOOTKEY) the firmware stored in the Secure Boot region prior to executing it. This verification mechanism is a key element to consider for ensuring the system root of trust during deployment and execution of the Secure firmware. The following figure illustrates the Secure Boot process with BS verification:

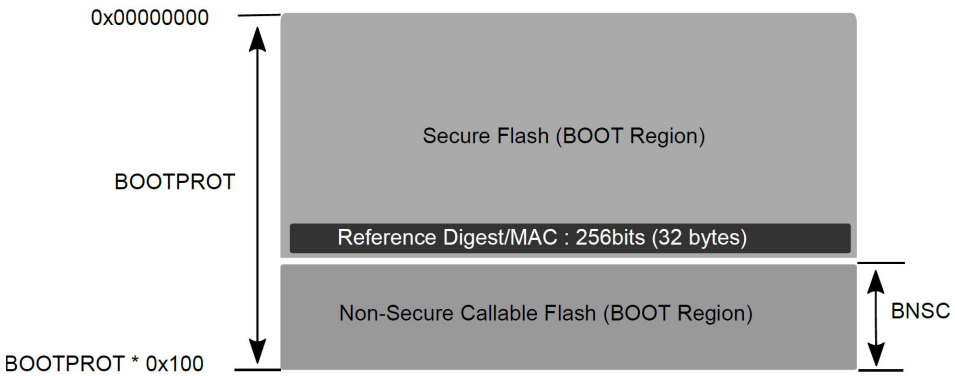
Figure 2-18. Secure Boot Process with BOOTPROT Verification



Note: The PIC32CM LS60 family of devices embeds an ATECC608B secure element allowing Secure Boot with ATECC608B. Refer to the chapter 'PIC32CM LS60 Secure Boot with ATECC608B CryptoAuthencitation™ Device' for more details

To validate the Secure Bootloader code stored in the device Flash BOOTPROT memory section, the ROM code computes the digest/MAC of the Flash BOOTPROT using the Crypto Accelerator (CRYA) and compares it to a reference digest/MAC (256 bits/32 bytes) stored in the device Secure Flash (BOOT region) memory section. This reference digest/MAC is stored in the last 256 bits of the Secure Flash (BOOT region) as shown in the following figure:

Figure 2-19. Reference Digest/MAC Location



If the verification result is equal to the reference digest/MAC, the Boot ROM starts the bootloader execution. Any mismatch in the value puts the device in an endless loop preventing Flash code execution. Only a `ChipErase_ALL` command allows the recovery from this device state. This command erases the full memory content and resets the fuses to their factory settings.

The following fuses are used in the Secure Boot process configuration:

- **BOOTPROT and BNSC:** Defines the configuration of the boot section in the product Flash. The size of the Secure and Non-Secure Callable boot sections can be customized according to the application needs. These fuses are used for security memory allocation in the product IDAU, and for integrity and authentication mechanisms when configured in the BOOTOPT fuse. Any change of the fuse setting requires a reset to be considered by the device, as only the Boot ROM can change the IDAU setting.
- **BOOTOPT:** Defines the type of verification to be performed.
- **BOOTKEY:** A 256-bit key used for the authentication mechanism.

Table 2-3. PIC32CM LS00/LS60 BootKey Fuse Address

BOCOR Offset	Bit Position	Name
0x50-0x6F	895:640	BOOTKEY

Table 2-4. PIC32CM LS00 Secure Boot Verification Methods

BOOTOPT	BOOTPROT Region Verification Method	BOCOR Row Verification Method
0	Secure Boot Disabled	
1	SHA-256	
2	SHA-256 with BOOTKEY (1)	
3	HMAC with BOOTKEY (1)	
Others	Reserved	

Table 2-5. PIC32CM LS60 Secure Boot Verification Methods

BOOTOPT	BOOTPROT region Verification Method	NVM Boot Configuration Row (BOCOR)
0	Secure Boot Disabled	
1	SHA-based Secure Boot	
2	SHA-based Secure Boot with BOOTKEY(1)	
3	HMAC-based Secure Boot with BOOTKEY(1)	
4	ATECC608B-based Secure Boot	SHA
5	ATECC608B-based Secure Boot	SHA with BOOTKEY(1)
6-255	ATECC608B-based Secure Boot	HMAC with BOOTKEY(1)

Notes:

1. BOOTKEY is defined in the BOCOR row.
2. The PIC32CM LS60 family of devices embeds an ATECC608B secure element, allowing more BOOTPROT verification methods. Refer to the *PIC32CM LS60 Secure Boot with ATECC608B CryptoAuthentication™ Device* for more details
3. Refer to the *PIC32CM LE00/LS00/LS60 Family Data Sheet* for additional information on the Secure Boot process and the verification methods.

2.6 Secure Boot Using ATECC608B CryptoAuthentication™ Device (PIC32CM LS60 only)

The PIC32CM LS60 device embeds a provisioned variant of the ATECC608B. The ATECC608B configuration of the PIC32CM LS60 family is nearly identical to that of the TrustFLEX secure element called ATECC608B-TFLXTLS.

The PIC32CM LS60 Boot ROM provides support for secure boot using the ATECC608B. The general approach is that the Boot ROM will use the ATECC608B to assist in authenticating and checking the integrity of an application code that is to be subsequently executed.

Note: Refer to the *PIC32CM LE00/LS00/LS60 Family Data Sheet* for more details on the embedded ATECC608B CryptoAuthentication Device.

To ease the ATECC608B configuration and provisioning for the customer, Microchip Technology has developed the Trust Platform Design Suite (TPDS) v2, which is a dedicated software platform for onboarding with embedded security. In addition to offering Secure Element and security solutions, Microchip also offers a full onboarding experience to start with embedded security (including but not limited to trainings and interactive application notes) to leverage Microchip's secure and optimized provisioning flow.

- Running a PIC32CM LS60 TrustZone Example with ATECC608B SHA256-based or ATECC608B HMAC-based Secure Boot for MPLAB Harmony v3

To benefit from the ATECC608B + TPDS v2 environment using a PIC32CM LS60 for Secure Boot, the following important steps must be realized:

1. Flash the `trustZone_kit_protocol` example available in *C:\<TPDSv2_install_folder>\tpds_core\TrustFLEX\01_accessory_authentication\firmware\LS60*.
2. Run the *Firmware Validation* use case for the ATECC608B-TFLXTLS under TPDS v2 by providing the Boot and Application hex files.
3. Flash the generated combined hex file and enjoy.

Note: The `trustZone_basic_with_SBoot_ls60` example can be used as support for steps two and three. The user must update the project configuration to switch to BOOTOPT >= 4.

2.7 Device Identity Composition Engine (DICE)

The Device Identifier Composition Engine (DICE) is a standard developed by the Trusted Computing Group (TCG) for implementing attestation in low-cost IoT devices.

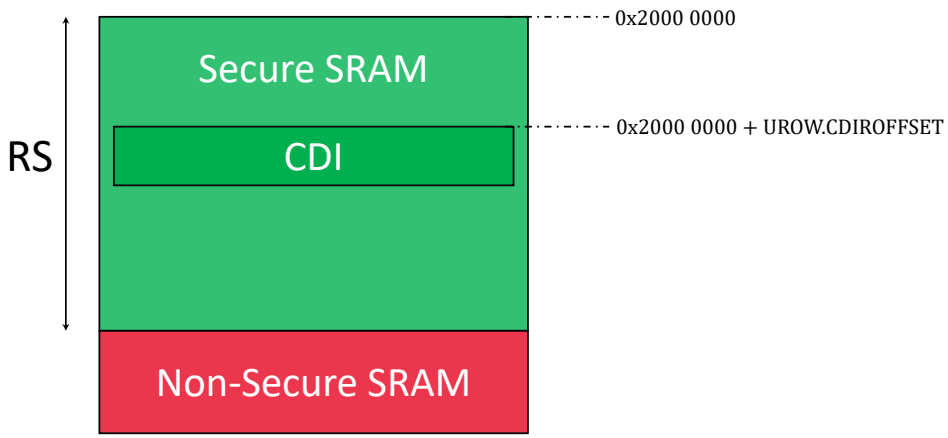
When enabled using BOCOR.DICEEN fuse, the DICE engine generates the Compound Device Identifier (CDI) at boot time that is based on a stored Unique Device Secret (UDS) key and the digest/MAC of the boot Flash image (BOOTPROT region).

The CDI is written in the SRAM at the offset specified by the UROW.CDIROFFSET fuse, making it available for the boot Flash code for the attestation purpose. The boot Flash code optionally can use the CDI to derive other keys for attestation and encryption.

The Boot ROM checks that CDIROFFSET is within the max SRAM range: if not, the CDI is not written.

Important: It is up to the user to ensure the CDI is written in the Secure SRAM region.

Figure 2-20. DICE CDI SRAM Location



2.7.1 Unique Device Secret (UDS)

The Unique Device Secret (UDS) is a 256-bit symmetric key used to generate the CDI. The UDS key is only accessible by the DICE engine and is only used for calculating the CDI at boot time.

Important: The UDS must be accessible only during Boot ROM execution: BOCOR.BCWEN and BOCOR.BCREN must be set both to '0' and BOCOR.CECFGLOCK must be set to '1' to follow DICE standard.

The Unique Device Secret must be provisioned on the BOCOR.UDS fuse. It must be unique for each device and have a strong entropy. If DICE is enabled but the UDS key is not provisioned (BOCOR.UDS is all 1's), all zeros are written to the CDI output.

Important: Devices can be factory programmed with securely key provisioned software.

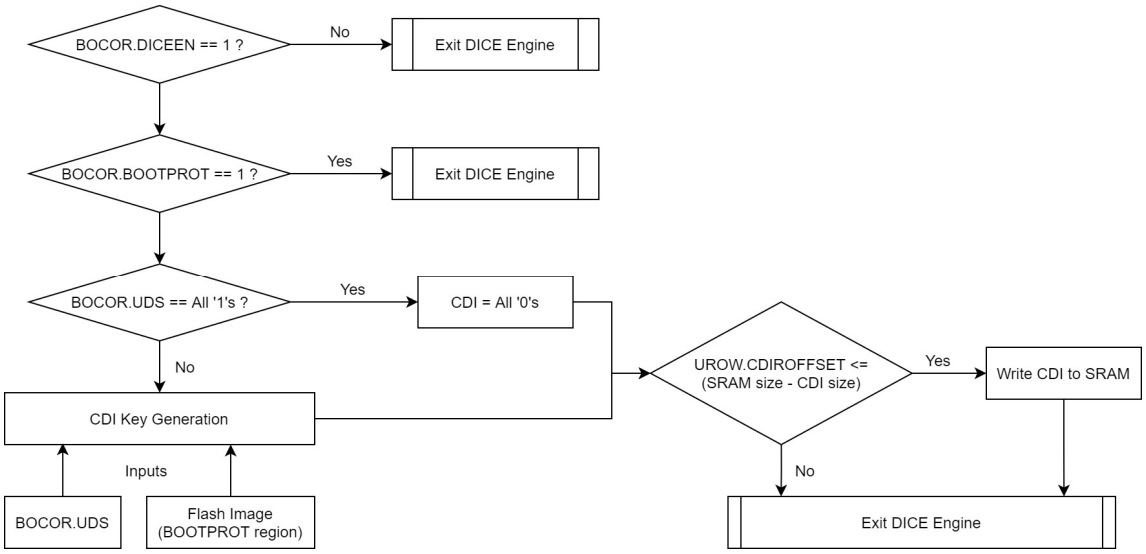
⚠ CAUTION A `ChipErase_ALL` (CE2) will reset the whole BOCOR including the provisioned UDS.

2.7.2 Compound Device Identifier

The Compound Device Identifier (CDI) is the output of the DICE module that is passed to the boot Flash code at a specified memory location in the SRAM. The CDI can be used by the boot Flash code directly for attestation or to derive other keys.

For additional information, refer to the “PIC32CM LE00/LS00/LS60 Family Data Sheet”.

Figure 2-21. DICE CDI Key Generation Flow



2.8 Cryptographic Accelerator (CRYA)

The PIC32CM LS00/LS60 products embed a hardware or software cryptographic accelerator (CRYA) which supports Advanced Encryption Standard (AES) encryption and decryption, Secure Hash Algorithm 2 (SHA-256) authentication, Galois Counter Mode (GCM) encryption, and authentication through a set of APIs.

Features:

The following are key features of CRYA:

- Advanced Encryption Standard (AES) compliant with FIPS Publication 197
  - Encryption with 128-bit, 192-bit, and 256-bit cryptographic key
  - Decryption with 128-bit, 192-bit, and 256-bit cryptographic key
- Secure Hash Algorithm 2 (SHA-256), compliant with FIPS Pub 180-4
  - Accelerates message schedule and inner compression loop
- Galois Counter Mode (GCM) encryption using AES engine and authentication
  - Accelerates the GF(2^{128}) multiplication for AES-GCM hash function

Refer to the *PIC32CM LE00/LS00/LS60 Family Data Sheet* for additional information on this feature.

3. PIC32CM LS00/LS60 Application Development (Developers A and B)

The combination of the system DAL and `ChipErase` commands with the TrustZone for Cortex-M architecture allows the developers to handle the following development and deployment approaches:

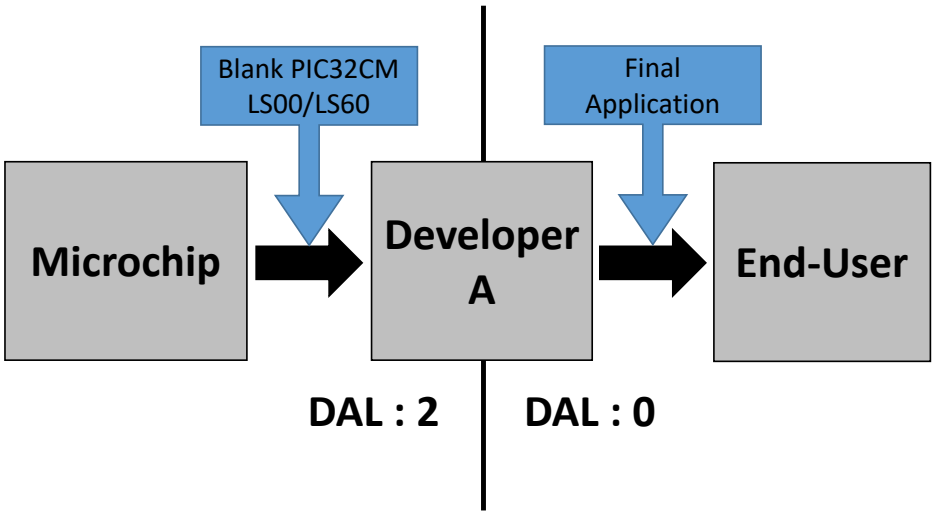
- Single-developer approach (Developer A)
- Dual-developer approach (Developer A + Developer B)

MPLAB Harmony v3 development tool coupled with MPLAB X IDE provides a full set of advanced features to accelerate the development of a PIC32CM LS00/LS60 application. The following sections illustrate the approaches to be followed by Developer A and Developer B to create and customize their application.

3.1 Single Developer Approach

In a single developer approach, the developer (Developer A) oversees the development and deployment of the Secure and Non-Secure code. The Developer A's application can be protected by using DAL0. The following figure illustrates a single developer approach on the PIC32CM LS00/LS60:

Figure 3-1. Single Developer Approach

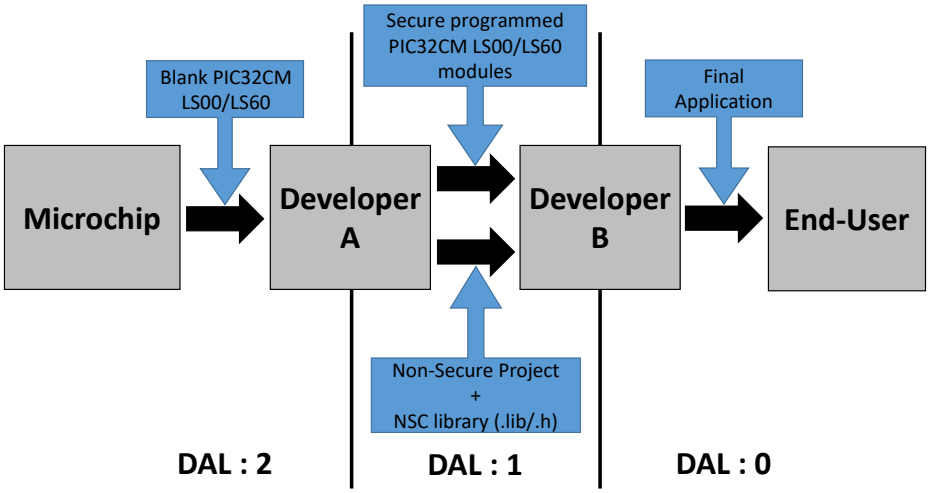


3.2 Dual-Developer Approach

In this approach, the first developer (Developer A) is in charge of developing the Secure application and its associated Non-Secure callable library (`.lib/.h`), and providing a prebuilt Non-Secure project to the second developer (Developer B). This Secure application is then loaded in the PIC32CM LS00/LS60 Flash and protected using the set DAL1 command to prevent further access to the Secure memory region of the device.

A second developer (Developer B) will then start their development on a programmed PIC32CM LS00/LS60 with limited access to Secure resources (call to Non-Secure API only). To achieve this, Developer B will use the Non-Secure project and the NSC library provided by Developer A. The following figure illustrates the dual-developer approach on the PIC32CM LS00/LS60:

Figure 3-2. Dual-Developer Approach



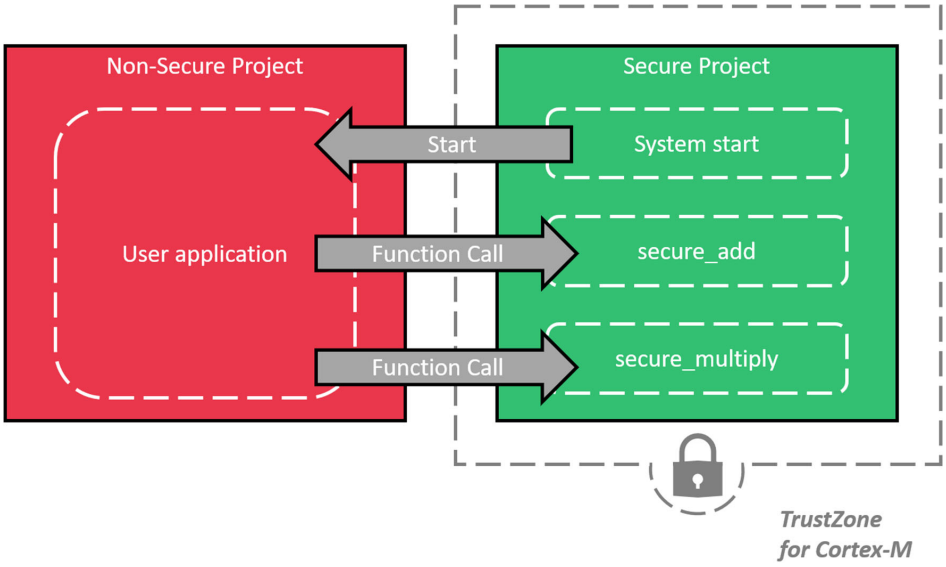
The following sections describe the application development and deployment process to be implemented for Developer A and Developer B.

3.3 Develop a TrustZone Example (Developer A)

To help Customer A to start with the PIC32CM LS00/LS60 (regardless of single or dual developer approaches), MPLAB Harmony v3 provides pre-configured `trustZone_basic_ls00` and `trustZone_basic_ls60` examples that illustrate the basic Secure and Non-Secure application execution as shown in the following figure. This template can be used to evaluate and understand the TrustZone for ARMv8-M implementation in the device, or as a start-up point for custom solution development.

Note: For more understanding, the `trustZone_basic` example name will be mentioned in the whole document starting now, which refers to the PIC32CM LS00 and PIC32CM LS60 MPLAB Harmony v3 examples.

Figure 3-3. TrustZone_basic Example Overview



3.3.1 Opening a PIC32CM LS00/LS60 TrustZone Example from MPLAB Harmony v3

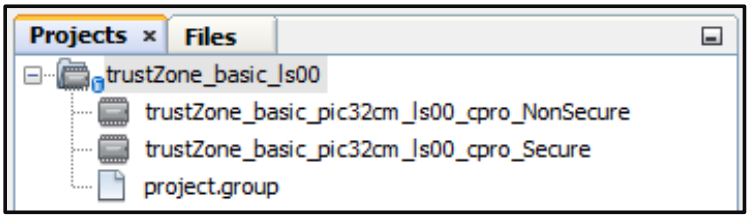
The `trustZone_basic` example is available in the MPLAB Harmony v3 framework.

Note: The user must ensure that the MPLAB Harmony v3 framework is installed. This folder can be downloaded using the MPLAB Code Configurator (MCC) Content Manager. If the MPLAB Harmony framework is not installed while opening an MCC project, the user can download the required package from the following path: *C:/Users/HarmonyFramework*).

Follow these steps to open the software project:

1. Open MPLAB X IDE.
2. Select *Toolbar > File > Open Project (Ctrl + Shift + O)*.
3. Navigate to *C:\<Harmony3_Framework_Path>\csp_apps_pic32cm_le_ls\apps\trustZone*.
4. Open the appropriate *trustZone_basic* example (depending on connected board).
   When opened, the *trustZone_basic* example must appear in the MPLAB X IDE project tree as shown in the following figure:

   Figure 3-4. TrustZone_basic_ls00 Example Under MPLAB X IDE



5. Set the Non-Secure project as main then run the application.

3.3.2 TrustZone_basic Example Description

The PIC32CM LS00/LS60 `trustZone_basic` example provided with MPLAB Harmony v3 is composed of pre-configured Non-Secure and Secure projects.

All the configuration aspects related to TrustZone for ARMv8-M implementation are already implemented to facilitate the development process. The following sections describe the content of the example, and the key elements to be modified to customize the solution according to the application needs.

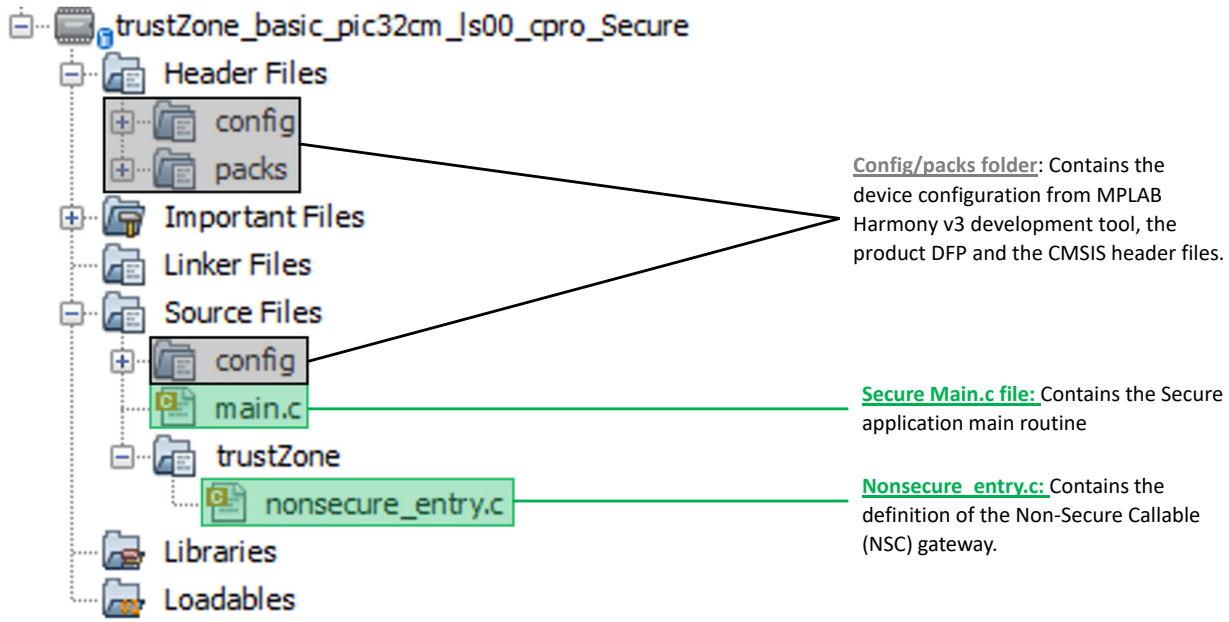
3.3.2.1 Secure Project Description

The goal of the Secure project included in the PIC32CM LS00/LS60 `trustZone_basic` example is to provide a configured development base for Secure code development. The Secure project is configured to illustrate the following aspects of a standard Secure application on the PIC32CM LS00/LS60:

- Device resources attribution for the Secure and Non-Secure applications (fuse settings).
- Initialization of system security.
- Declaration of secure gateways with the Non-Secure application (veneers).
- Secure call to the Non-Secure application.

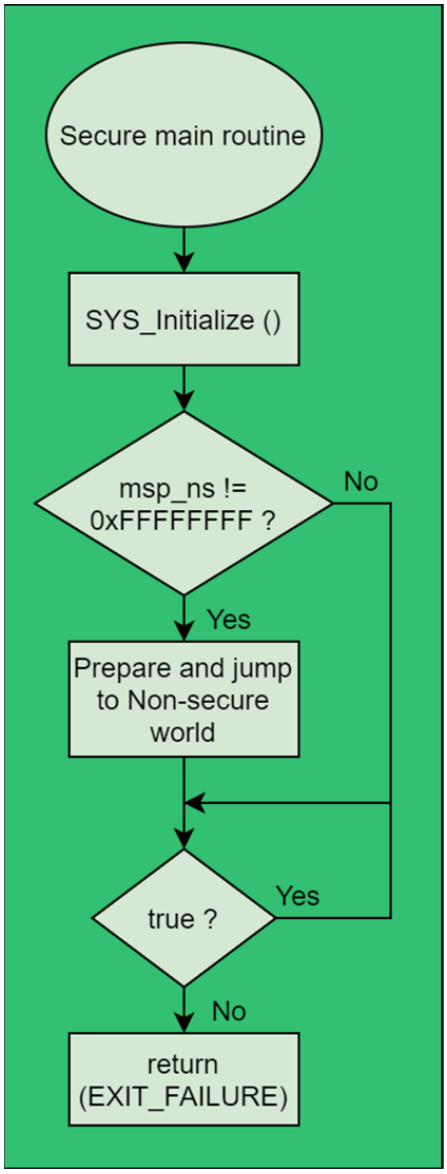
The following figure describes the file architecture of the configured Secure project:

Figure 3-5. TrustZone_basic Example: Secure Project Architecture



The following figure describes the main routine of the configured Secure project:

Figure 3-6. TrustZone_basic Example: Secure Project Main Flowchart

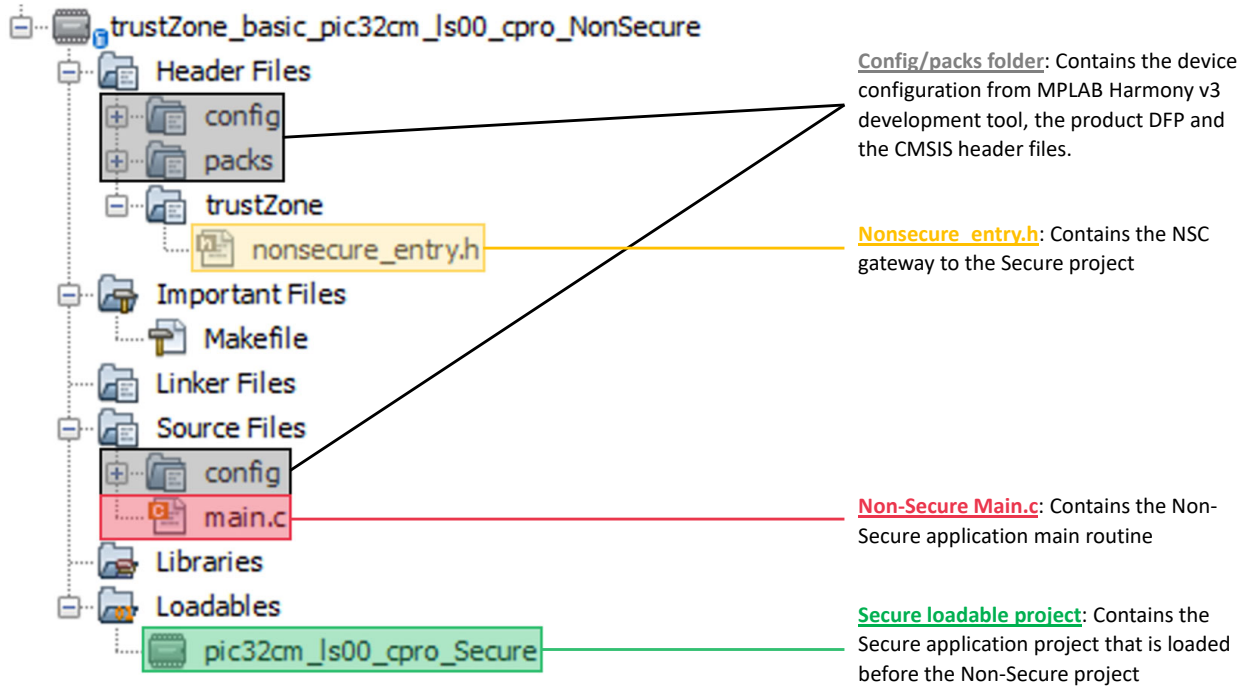


The Secure *main.c* file must be used as a starting point for any secure application development.

3.3.2.2 Non-Secure Project Description

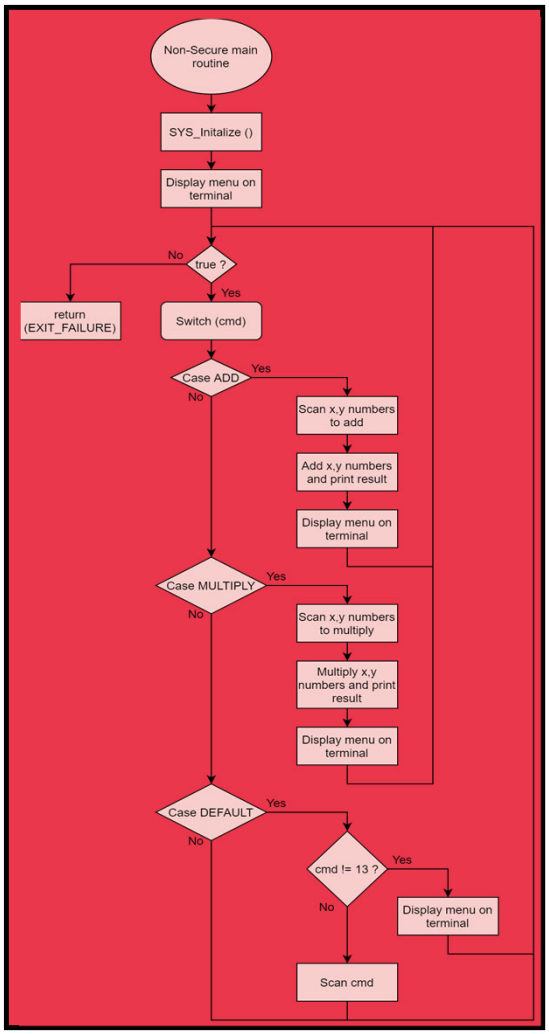
The Non-Secure project provided within the PIC32CM LS00/LS60 `trustZone_basic` example is a standard application that runs in a Non-Secure world. This application can use all system resources allocated to the Non-Secure world. It uses a programmed Non-Secure Callable (NSC) function using the `nonsecure_entry.h` provided by Developer A. The Non-Secure project architecture is shown in the following figure:

Figure 3-7. TrustZone_basic Example: Non-Secure Project Architecture



The following flowchart describes the main routine of the configured Non-Secure project:

Figure 3-8. TrustZone_basic Example: Non-Secure Project Main Flowchart



The Non-Secure main function illustrates the call of specific Secure code through gateways declared in the `nonsecure_entry.h` file provided by the Secure application.

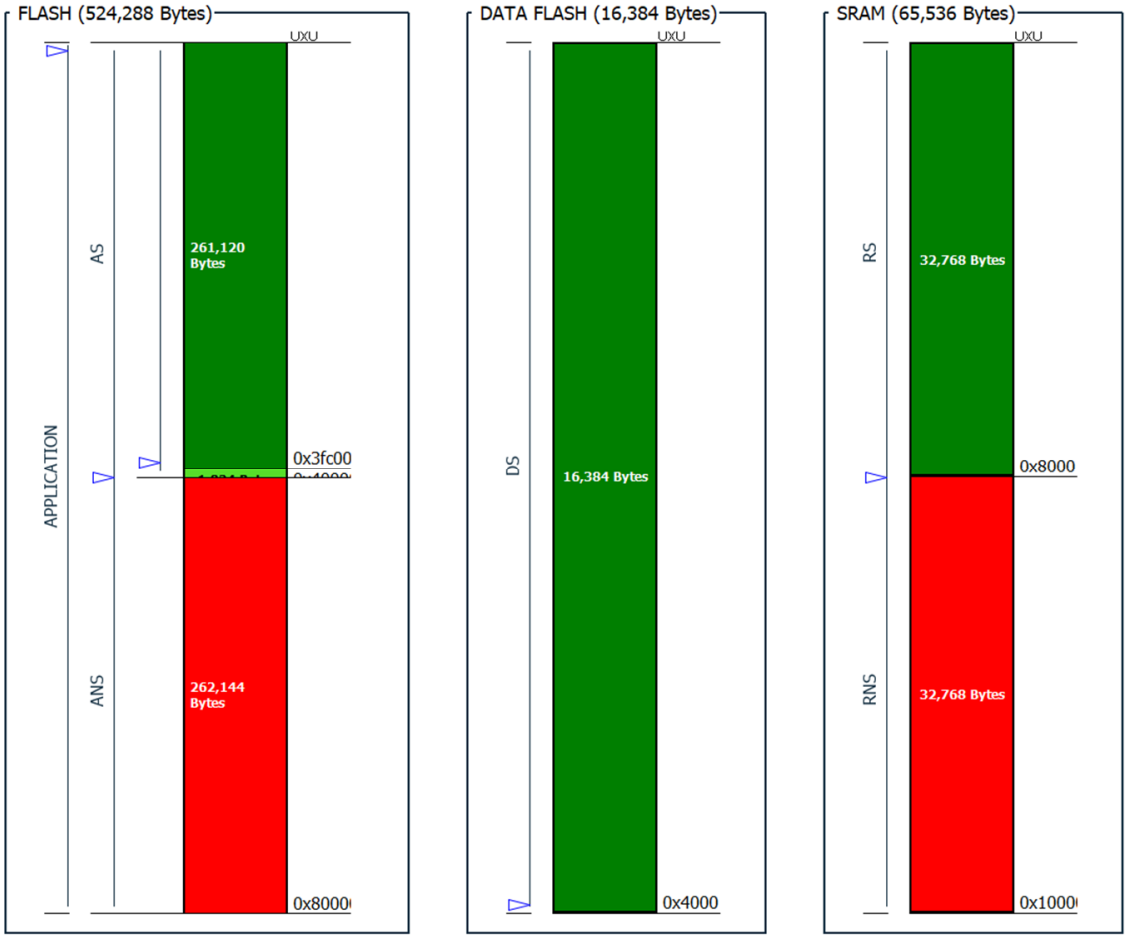
This Non-Secure `main.c` file can be used as a starting point for any Non-Secure applications development.

3.3.2.3 Memory Configuration with Arm® TrustZone® for Armv8-M Manager

The Arm® TrustZone® for Armv8-M Manager is a tool that allows the Memory Configuration and the Peripheral Configuration to be set with a GUI for the PIC32CM LS00/LS60 devices. To access this window, the MPLAB Harmony v3 development tool must be opened.

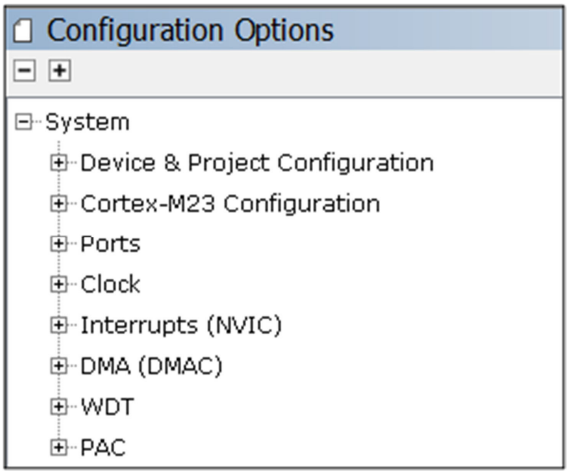
The following figure shows the PIC32CM LS00/LS60 memory configuration for the `trustZone_basic` example:

Figure 3-9. TrustZone_basic Example: Memory Configuration



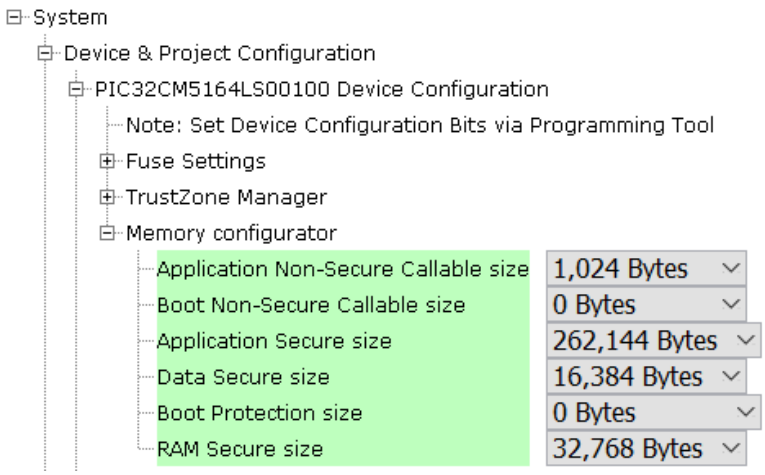
Users can configure the Memory Configuration of the PIC32CM LS00/LS60 by accessing the System block in the Project Graph view. Once selected, the following information will be displayed in the Configuration Options window.

Figure 3-10. Configuration Options Window



Under System, expand *Device & Project Configuration > PIC32CM5164LSXX100 Device Configuration > Memory Configurator*.

Figure 3-11. TrustZone_basic Example: Memory Configuration Through System Block



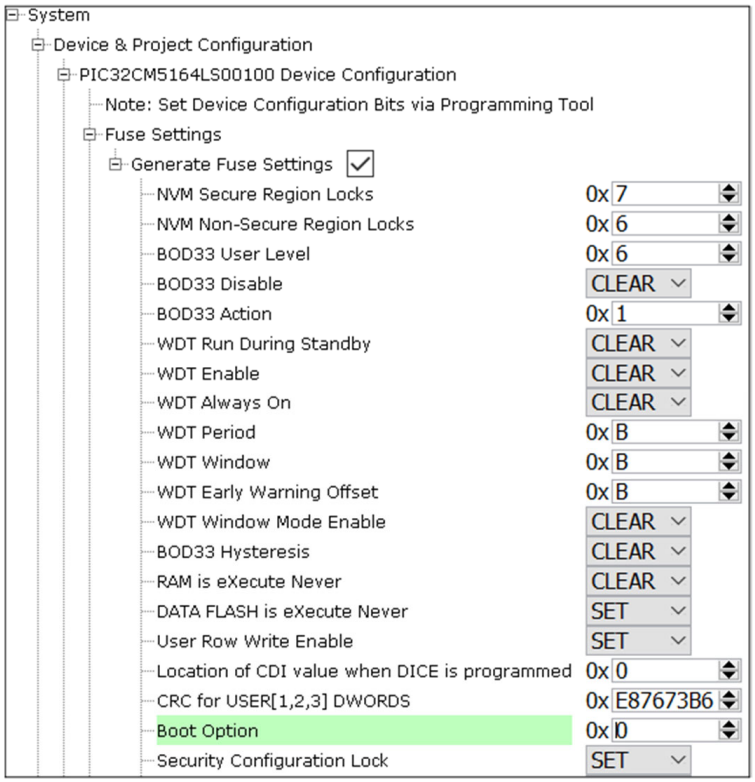
3.3.2.4 NVM Configuration Rows Settings

To ease the definition and modification of application fuses, MPLAB Harmony v3 development tool allows the manual configuration of all the NVM Configuration rows by accessing the Configuration Options window.

Important: The fuses used to partition the memory sections are directly computed and defined by the MPLAB Harmony v3 development tool regarding the memory configuration of the device while generating the project. These values are then set as preprocessed macros as shown in the next chapter.

Under System, expand *Device & Project Configuration* > *PIC32CM5164LSXX100 Device Configuration* > *Fuse Settings*.

Figure 3-12. NVM Configuration Bit fields Settings



These fuses define the configuration of Boot modes, `ChipErase`, system peripherals (BOD and watchdog), IDAU (Memory security attribution), and PAC (Peripheral security attribution), and must be modified according to the application needs.

Note: To get more information concerning the description of the different NVM Configuration rows and bit fields, refer to the *NVM Configuration Rows chapter* of the *PIC32CM LE00/LS00/LS60 Family Data Sheet*.

Any change to the fuse configuration requires a restart of the device, as fuses are handled by the Boot ROM executed at device start-up. The Boot ROM is responsible for copying the configuration of the fuses in the different peripheral registers, and then locking the configuration to any users (including Developer A) until the next boot.

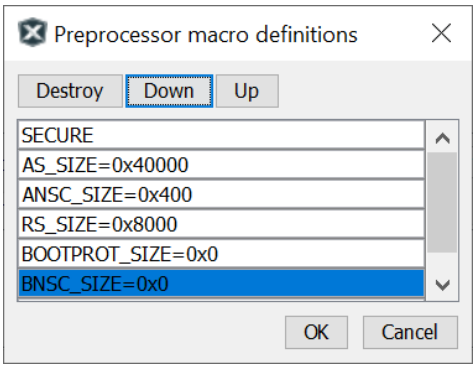
Note: Refer to the *Boot ROM* chapter of the “PIC32CM LE00/LS00/LS60 Family Data Sheet” for additional information on the Boot ROM.

3.3.2.5 Secure and Non-Secure Projects Linker Files

The Secure and Non-Secure projects share the same configured linker file which is available in the `PIC32CM-LS_DFP` folder. This linker file is built to fit with the project configuration available in project *Properties > XC32 (Global Options) > xc32-ld > Symbols & Macros > Preprocessor macro definitions*.

The following settings are generated by the MPLAB Harmony v3 development tool, which based on the memory configuration of the Arm® TrustZone® for Armv8-M Manager as shown in the following figure.

Figure 3-13. Preprocessor Macro Definitions



⚠ CAUTION The preprocessor macros reflect the size of the memory space and not the fuse value that explains the `_SIZE` suffix after the fuse name.

To ensure that the memory section definitions are in line with the fuse settings, the configured linker file is built as follows:

```
#if defined(SECURE_BOOTLOADER)
#  define _SECURE
#  define TZ_ROM_ORIGIN (ROM_ORIGIN)
#  define TZ_ROM_LENGTH (BOOTPROT_SIZE)
#  define TZ_ROM_NSC_ORIGIN ((ROM_ORIGIN + BOOTPROT_SIZE) - BNSC_SIZE)
#  define TZ_ROM_NSC_LENGTH (BNSC_SIZE)
#  define TZ_RAM_ORIGIN (RAM_ORIGIN)
#  define TZ_RAM_LENGTH (RS_SIZE)
#elif defined(SECURE)
#  define _SECURE
#  define TZ_ROM_ORIGIN (ROM_ORIGIN + BOOTPROT_SIZE)
#  define TZ_ROM_LENGTH (AS_SIZE)
#  define TZ_ROM_NSC_ORIGIN (ROM_ORIGIN + ((BOOTPROT_SIZE + AS_SIZE) - ANSC_SIZE))
#  define TZ_ROM_NSC_LENGTH (ANSC_SIZE)
#  define TZ_RAM_ORIGIN (RAM_ORIGIN)
#  define TZ_RAM_LENGTH (RS_SIZE)
#elif defined(NONSECURE)
#  define TZ_ROM_ORIGIN (ROM_ORIGIN + (BOOTPROT_SIZE + AS_SIZE))
#  define TZ_ROM_LENGTH (ROM_LENGTH - (BOOTPROT_SIZE + AS_SIZE))
#  define TZ_RAM_ORIGIN (RAM_ORIGIN + RS_SIZE)
#  define TZ_RAM_LENGTH (RAM_LENGTH - RS_SIZE)
#endif
```

Depending on the project tag (SECURE, NONSECURE or SECURE_BOOTLOADER), the appropriate memory space will be defined that allows for the modification of the project settings to fit with the application needs, without modifying the linker file.

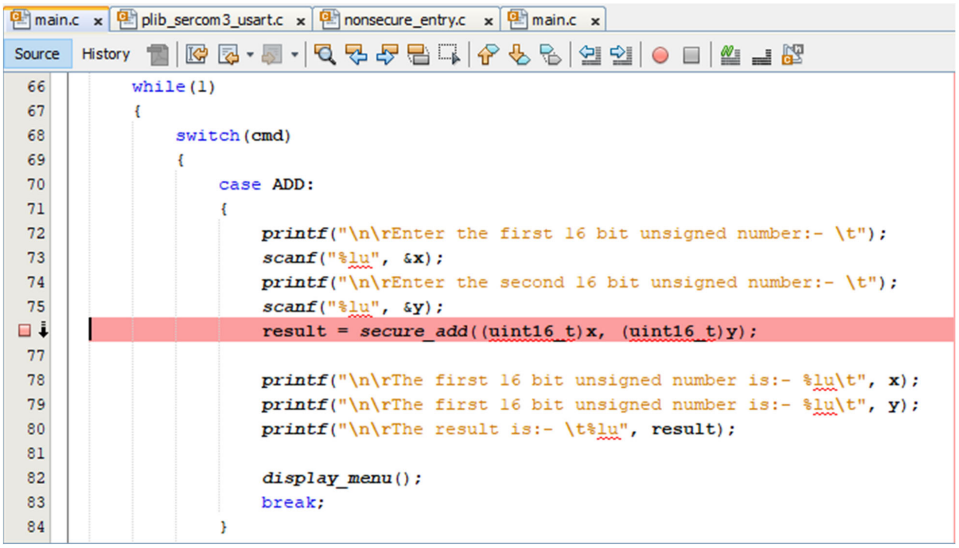
⚠ CAUTION The preprocessor macro values can be modified manually in MPLAB X IDE project properties. However, the MPLAB Harmony v3 and MPLAB X IDE projects configurations will not be aligned. Configure the memory configuration in Arm® TrustZone® for Armv8-M Manager from the MHC Harmony v3 development tool regarding the project needs, to ensure both MPLAB Harmony v3, and MPLAB X IDE project memory configurations are aligned.

3.3.3 Debugging the trustZone_basic Example

When the device is in DAL = 2, the debugging of the full example (Secure + Non-Secure project) is allowed. The following steps provide the capabilities of the MPLAB X IDE for debugging the TrustZone application:

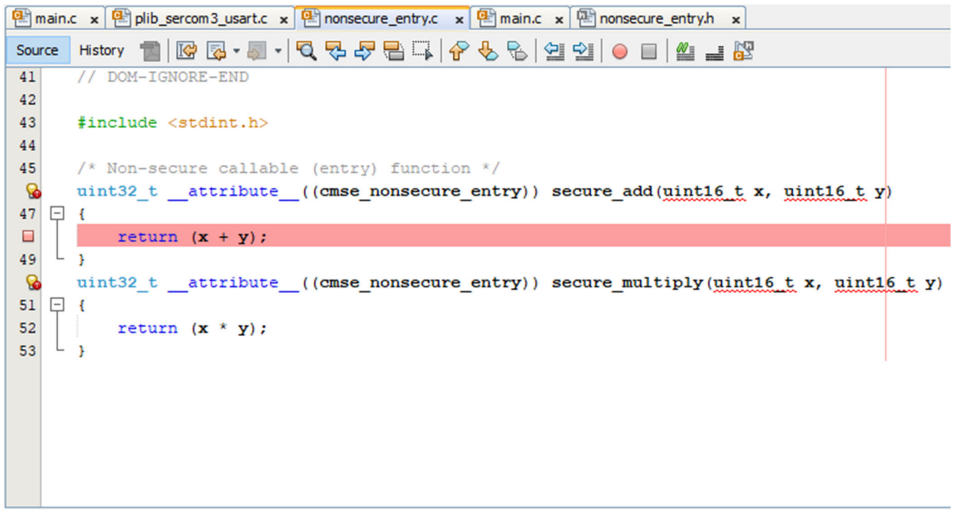
1. To build the example in MPLAB X IDE, click the  icon.
   Note: The Non-Secure project must be set as the main project to re-build and load the full example.
2. Ensure that the PIC32CM LS00/LS60 debugger I/O is connected to a computer and is recognized.
3. Add a breakpoint on the `result = secure_add(x,y)` line in the Non-Secure project `main.c` file.

Figure 3-14. Adding a Breakpoint Line



4. Add a breakpoint on the return line of the `secure_add` in the Secure project `nonsecure_entry.c` file.

Figure 3-15. Adding a Breakpoint Line to secure_add



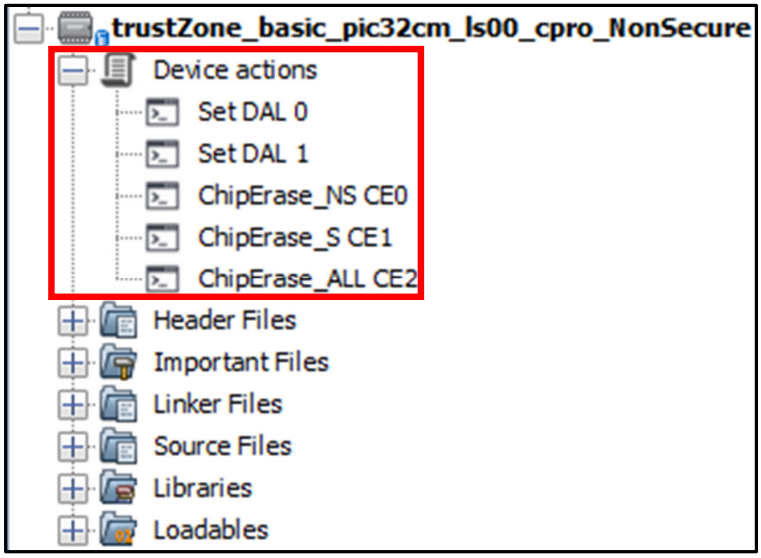
⚠ CAUTION When debugging the Secure application veneers, only hardware breakpoints must be used to stop code execution on an SG instruction. Using software breakpoints implies the addition of a BKP instruction before SG instruction, which triggers a Secure fault during the code execution. This behavior is normal, as the first instruction to be executed when accessing the NSC region must be an SG instruction.

5. Start the debugging session by clicking the  button, and continue the debugging by clicking the  button.
   As a result, the debugger must stop successively on:
   - The `result = secure_add(x,y)` function call (Non-Secure project)
   - The `secure_add` return (Secure project)

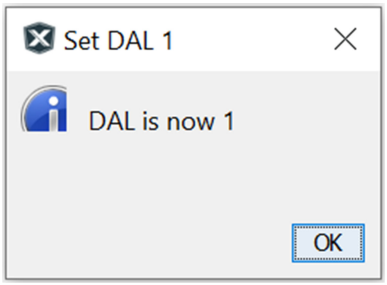
3.3.4 Protecting the Secure Project Using Debug Access Levels

In a dual-developer deployment approach, it is important to protect the Secure memory regions (Secure application) from further debugger accesses prior to delivering the programmed devices to Developer B. This can be done by changing the DAL to DAL = 1 using the *Device Actions* commands as shown in the projects tree.

1. Close the debug session (if running).
2. Expand the *Device Actions* commands in the project tree (Secure or Non-Secure).
   Figure 3-16. Device Actions Commands

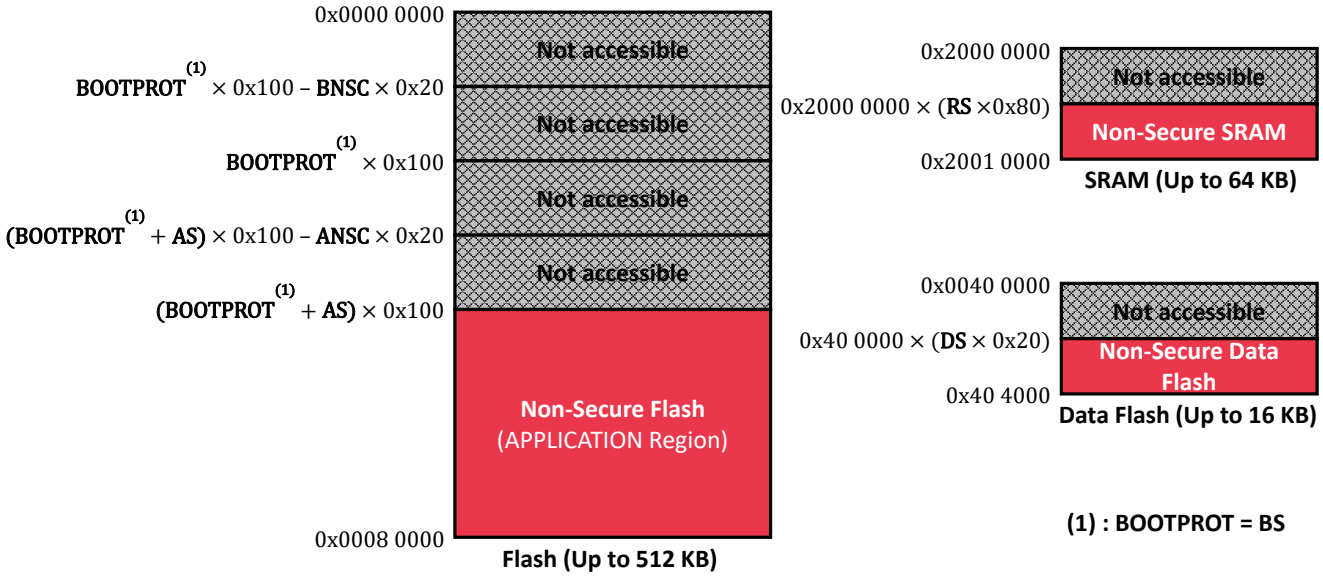


3. Double-click on the **Set DAL 1** command.
4. After the following confirmation message is displayed, click **OK**.
   Figure 3-17. DAL 1 Set Prompt Message



As a result, setting DAL = 1 prevents any future debug access to the Secure memory region of the device, as shown in the following figure:

Figure 3-18. DAL Protected Device Memory Region



Note:

1. BOOTPROT = BS.

Any future debug access to the Secure memory region will be refused by the device and reported as follows by MPLAB X IDE.

Figure 3-19. The MPLAB X IDE Error Message at Runtime on DAL Protected Area

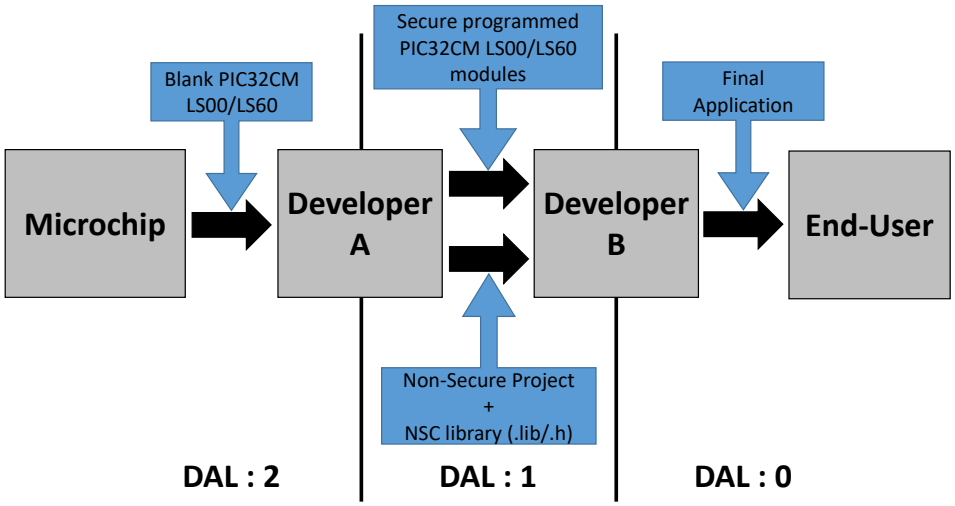
```
java.lang.RuntimeException: java.lang.RuntimeException: DAP transfer error 4
```

Important: Further development with the device may require the use of a standalone Non-Secure project. Refer to Develop a Non-Secure Project (Developer B). To re-enable debug access on the Secure memory regions, a `ChipErase_ALL` command (CE2) must be issued using the Device Actions commands. The whole device memory and fuse settings are erased, and the Secure application must be reprogrammed in the device.

3.4 Develop a Non-Secure Project (Developer B)

In the Developer B context, the development starts with a programmed PIC32CM LS00/LS60 device that contains a DAL1 protected Secure project with predefined veneers.

Figure 3-20. Develop a Non-Secure Project (Developer B)



In this context, it is mandatory for Developer A to provide Non-Secure resource attribution descriptions, and a Non-Secure callable function API library to Developer B.

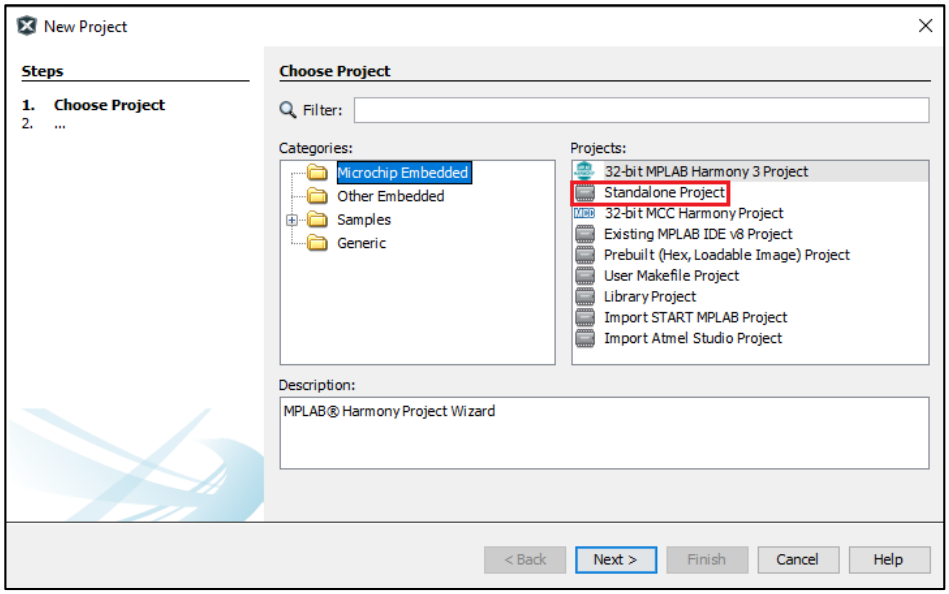
Ideally, Developer A provides a preconfigured MPLAB Harmony v3 Non-Secure project to Developer B. The following sections describe how to create and configure a Non-Secure project for a PIC32CM LS00/LS60 device embedding a programmed DAL1 protected Secure application.

3.4.1 Creating a Non-Secure Project from MPLAB X IDE

Follow these steps to create a Non-Secure project using MPLAB X IDE:

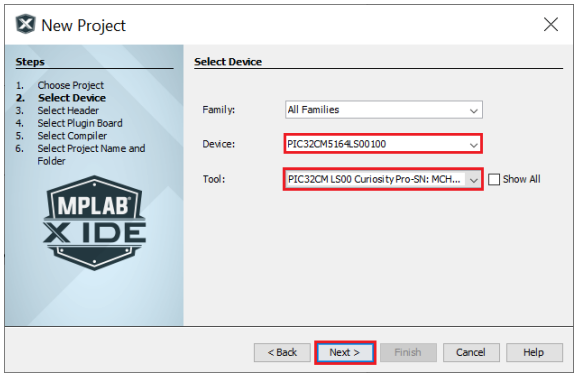
1. Open MPLAB X IDE.
2. Select *File > New Project (Ctrl + Shift + N)*, or from the Toolbar click the  button.
3. In the New Project Window, under Steps, select Choose Project.
4. In the right pane, under Categories, select *Microchip Embedded*, and then under Projects select *Standalone Project*.
5. Click **Next**.

Figure 3-21. MPLAB X IDE Standalone Project Creation: Choose Project Window



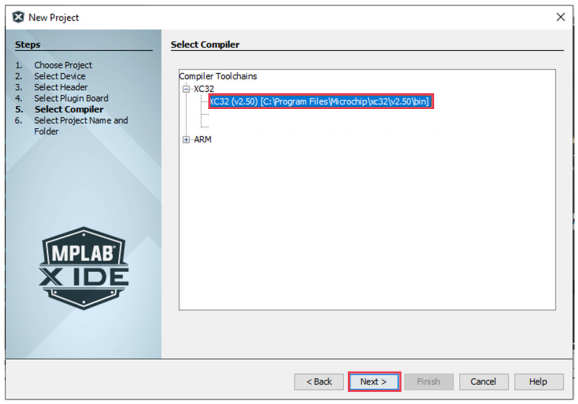
6. Select the option Select Device, and in the right pane select the PIC32CM LS00/LS60 device and the appropriate tool, if available.
7. Click **Next**.

Figure 3-22. MPLAB X IDE Standalone Project Creation: Select Device Window



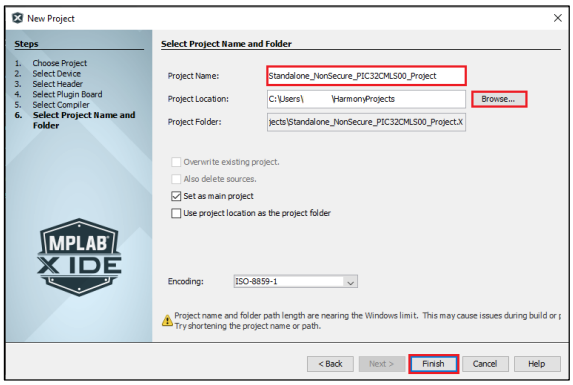
8. Select the option Select Compiler, and in the right pane choose the latest XC32 Compiler Toolchain installed, and then click **Next**.

Figure 3-23. MPLAB X IDE Standalone Project Creation: Select Compiler Window



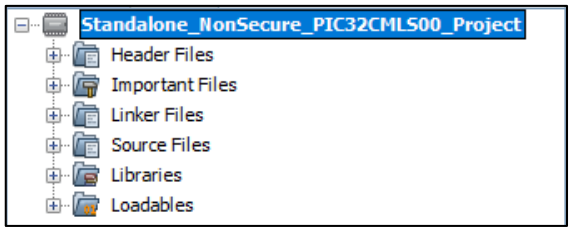
9. Select the option Select Project Name and Folder, and in the right pane enter details for Project Name and Project Location, and then click **Finish**.

Figure 3-24. MPLAB X IDE Standalone Project Creation: Select Project Name and Folder Window



The following Non-Secure project details will be displayed.

Figure 3-25. MPLAB X IDE Standalone Project Creation: Non-Secure PIC32CMLS00 Project Tree



3.4.2 Project Configuration

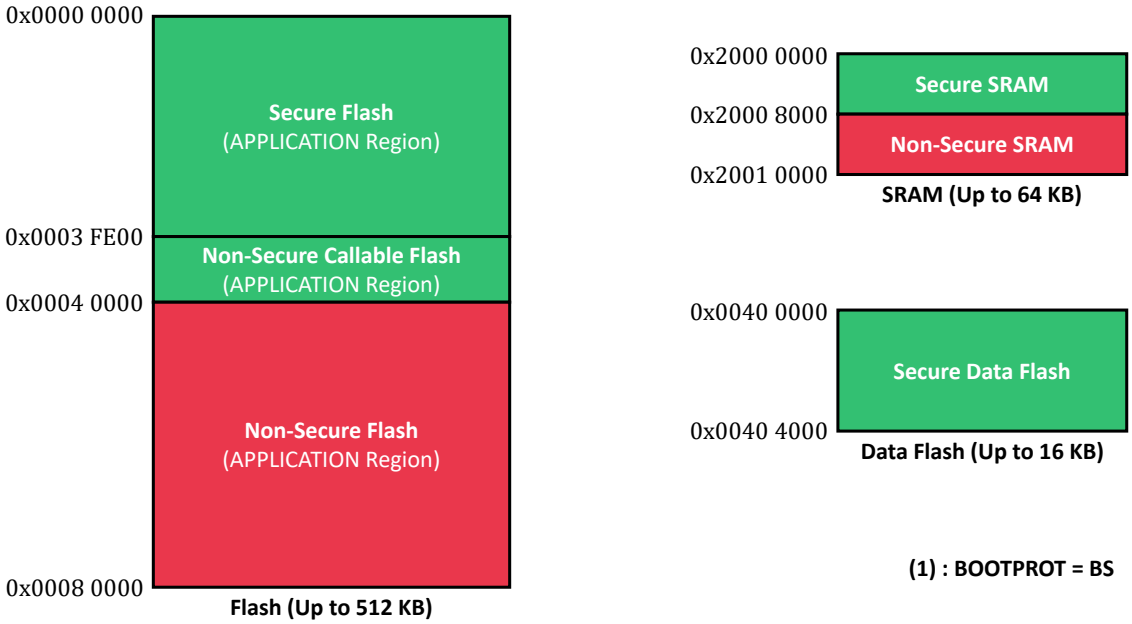
After creating a Non-Secure project, follow these steps to configure it according to the preprogrammed Secure project mapping and Secure gateway APIs:

1. Configure the project by aligning its linker file (using the pre-processor macros) to the Secure and Non-Secure memories attribution predefined by Developer A.
2. Link the secure gateway library to the project.
3. Add the `nonsecure_entry.h` file to the project.

3.4.2.1 Align Pre-processor Macros for Linker File to the PIC32CM LS00/LS60 Non-Secure Memories Attributions

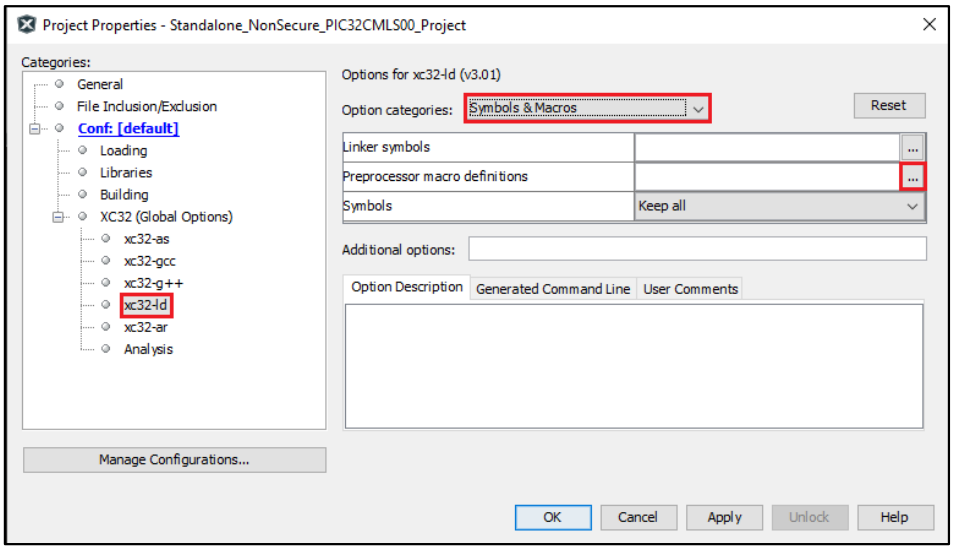
Follow these steps to align the project preprocessor macros to the linker file according to the Secure and Non-Secure memory space allocation as illustrated in the following figure:

Figure 3-26. TrustZone Example Memory Attribution



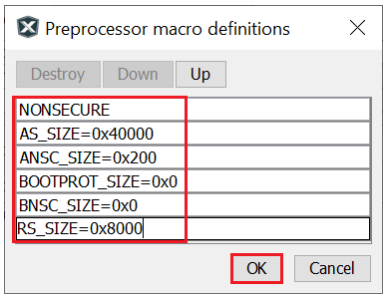
1. Open the Project Properties window and go to *XC32 (Global Options) > xc32-ld > Symbols & Macros*.
2. Click the icon to the right of the 'Preprocessor macro definitions' as shown below.

Figure 3-27. Symbols and Macros



3. Enter the macros to fit with the TrustZone example memory attribution, and then click **OK**.

Figure 3-28. Preprocessor Macro Definitions



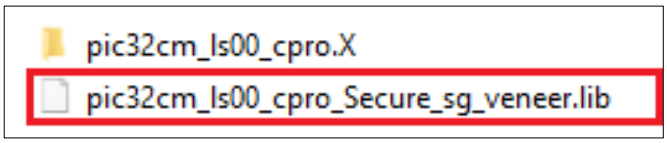
⚠ CAUTION The preprocessor macros reflect the size of the memory space and not the value of the fuse.

3.4.2.2 Adding and Linking Secure Gateway Library to Non-Secure Project

Follow these steps to add and link the secure gateway library that is generated during secure application development provided by Developer A:

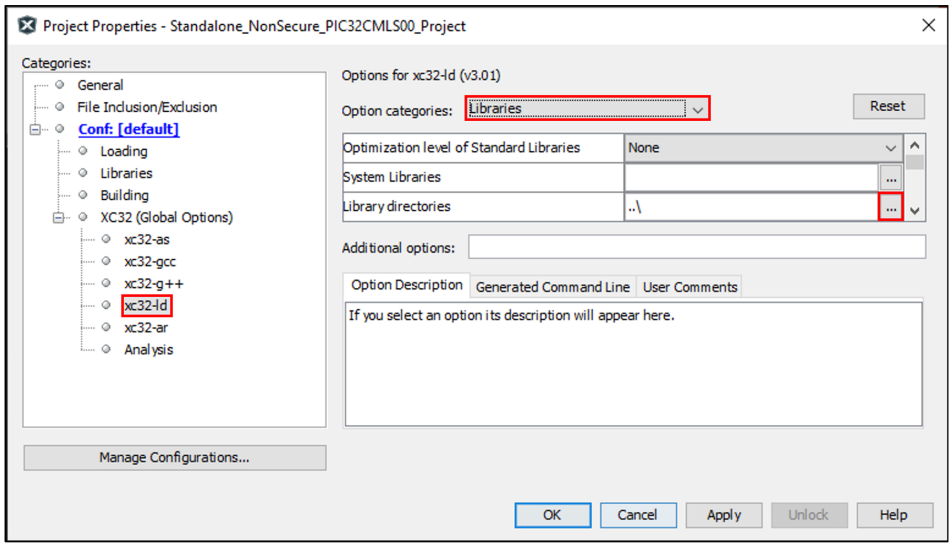
1. Copy the provided secure gateway library into the Non-Secure folder.

   Figure 3-29. Adding Secure Gateway Library to the Non-Secure Project



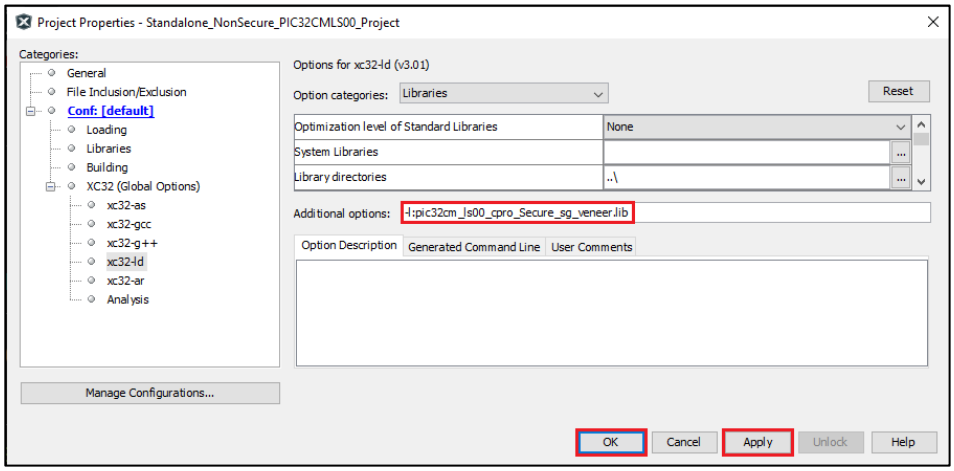
2. Link the secure gateway library to the Non-Secure project.
   a. Open the Project Properties.
   b. Under Categories, expand Conf (default) > XC32 (Global Options), and select xc32-ld.
   c. For Option Categories, choose Libraries.
   d. For Library directories select the library directory information.

   Figure 3-30. Adding Secure Gateway Library Directory to the Non-Secure Project



   e. For Additional options, enter the library name and apply the modifications, and then click **OK**.

Figure 3-31. Link Secure Gateway to the Non-Secure Project

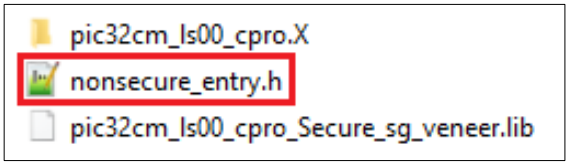


3.4.2.3 Adding and Including Secure Gateway Header File

To add and include the secure gateway header file, perform these actions:

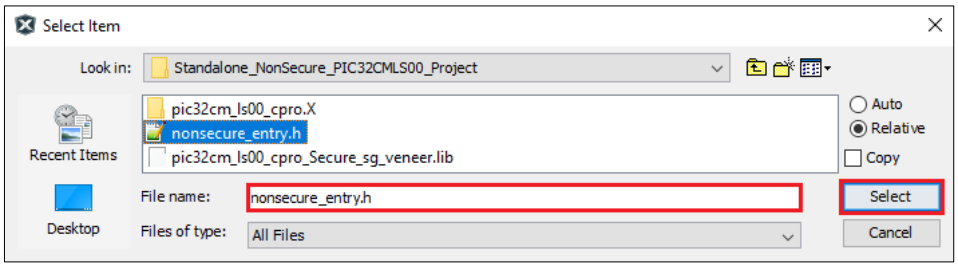
1. Copy the provided secure gateway header file into the Non-Secure folder.

   Figure 3-32. Adding Secure Gateway Header File into the Non-Secure Folder



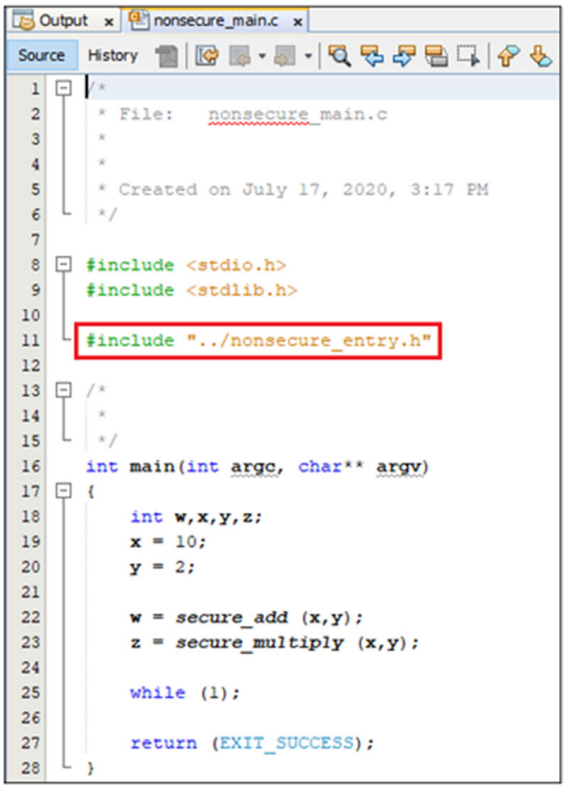
2. Right-click on the Non-Secure project in the project explorer, and then select **Add Existing Item**.
3. Select the secure gateway header file.

   Figure 3-33. Select Secure Gateway Header File



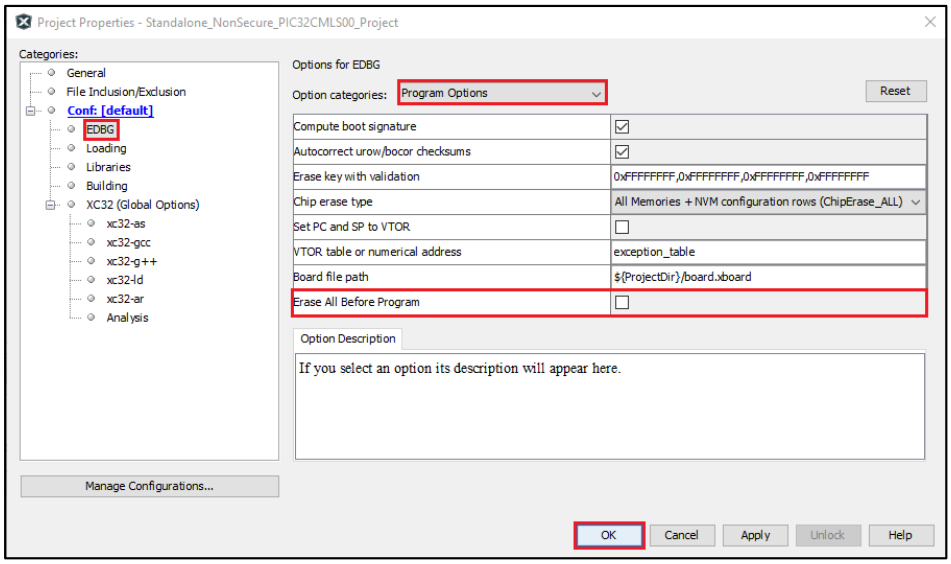
4. Add the secure gateway header at the beginning of the `nonsecure_main.c` file.

Figure 3-34. Include nonsecure_entry Header File



Important: Prior to loading the project on the target PIC32CM LS00/LS60 device, ensure that the Erase All Before Programming check box is cleared (under Program Options). This will prevent the process from executing a `ChipErase_ALL` command and erase the programmed PIC32CM LS00/LS60.

Figure 3-35. Uncheck Erase All Before Programming



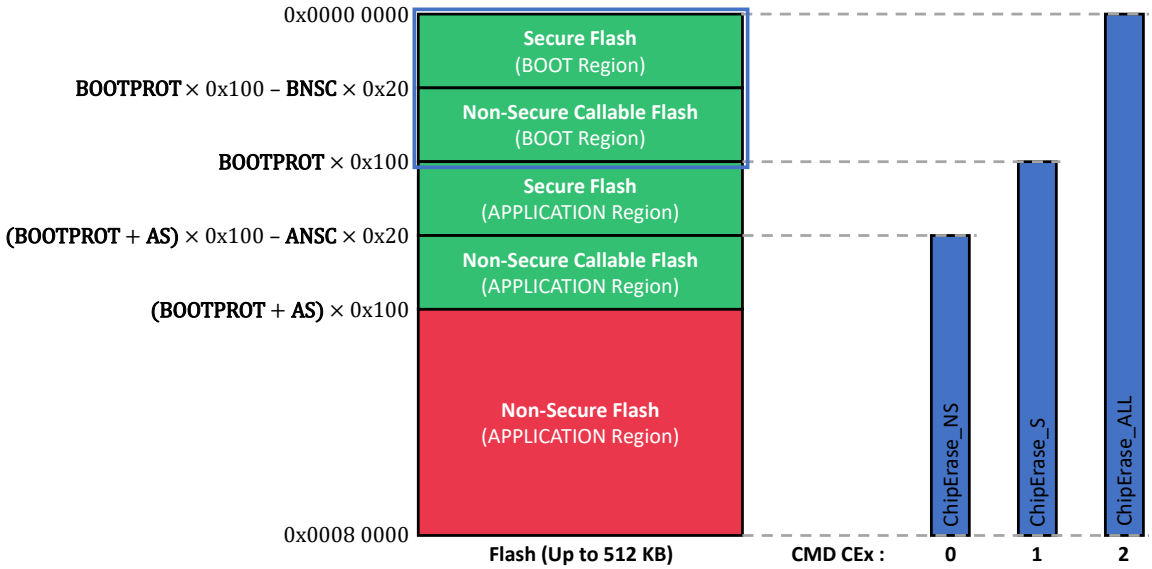
5. Build the project by clicking the  icon, and verify no error is reported by the build process.
6. Launch the debug session and verify whether the project is working.

Important: Debugging the Non-Secure project requires a compatible programmed Secure application that configures and starts the Non-Secure execution. If this Secure application is not available on the chip, the debug process will hang.

3.5 Developing TrustZone Example with SHA256-based or HMAC-based Secure Boot (Developer A)

The PIC32CM LS00/LS60 devices offer two configurable memory sections for storing the Secure boot program. These two sections are protected against `ChipErase_S` and `ChipErase_NS` offering possibilities to store the Secure bootloader code as shown in the following figure.

Figure 3-36. Application with Secure Boot Program



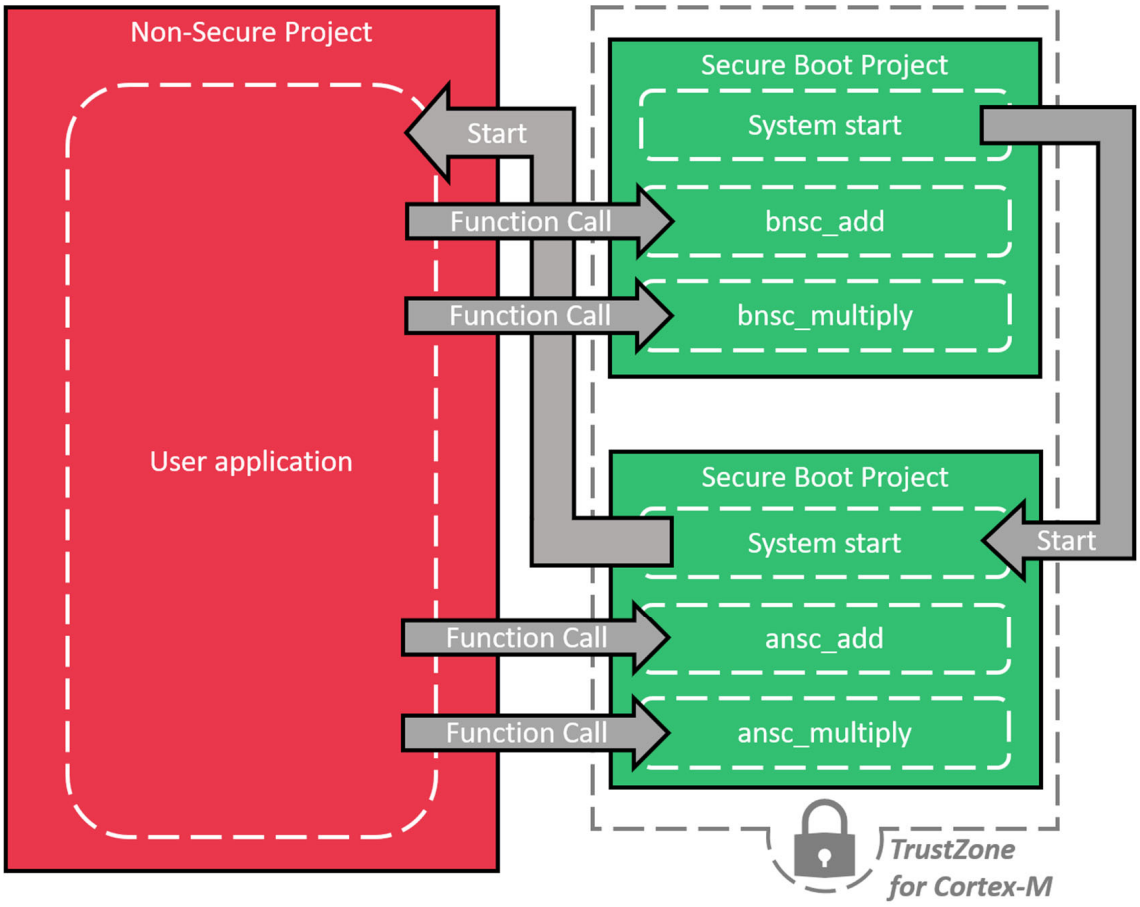
In addition to `ChipErase` protection, the product Boot ROM offers the possibility to perform an integrity check or authenticate the firmware stored in the Secure Boot section prior to executing it. This verification mechanism is a key element to consider for ensuring the system root of trust during deployment and execution of the Secure firmware.

3.5.1 Opening a PIC32CM LS00/LS60 TrustZone Example with SHA256-based or HMAC-based Secure Boot from MPLAB Harmony v3

To ease the development of an application with the Secure Boot program, MPLAB Harmony v3 provides predefined `trustZone_basic_with_SBoot_ls00` and `trustZone_basic_with_SBoot_ls60` examples. These examples can be used to evaluate and understand the solution architecture and to start the development of a custom application featuring a Secure Boot project.

Note: For a better understanding, the `trustZone_basic_with_SBoot` example name will be used to generalize it for PIC32CM LS00/LS60 devices.

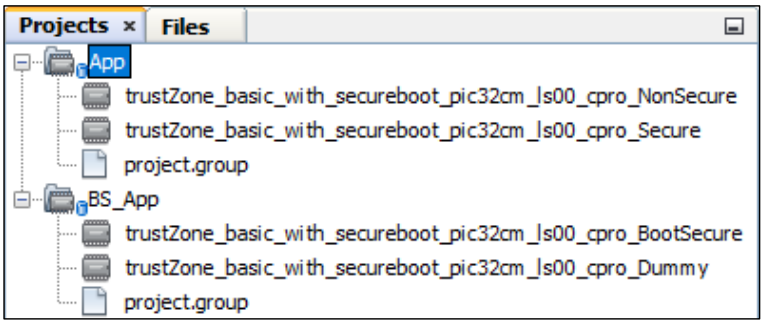
Figure 3-37. TrustZone_basic_with_SBoot Example Overview



To open the pre-configured `trustZone_basic_with_SBoot` example, follow these steps:

1. Open MPLAB X IDE.
2. Select *Toolbar > File > Open Project (Ctrl + Shift + O)*.
3. Navigate to *C:\<Harmony3_Framework_Path>\csp_apps_pic32cm_le_ls\apps\trustZone*.
4. Open the App and BS_App projects within the `trustZone_basic_with_SBoot` example (depending on connected board).
   When opened, the `trustZone_basic_with_SBoot` example must appear in the MPLAB X IDE project tree as shown in the following figure:

   Figure 3-38. TrustZone_basic_with_SBoot_ls00 Example: Project Tree

5. Run the application.
   a. Set the BootSecure project as main then run the example.
   b. Set the NonSecure project as main then run the example.

3.5.2 TrustZone_basic_with_SBoot Example Description

PIC32CM LS00/LS60 `trustZone_basic_with_SBoot` example provided with MPLAB Harmony v3 is similar to the `trustZone_basic` example described in the previous chapters; however, it embeds a Secure Boot program (stored in the BOOTPROT memory region of the device).

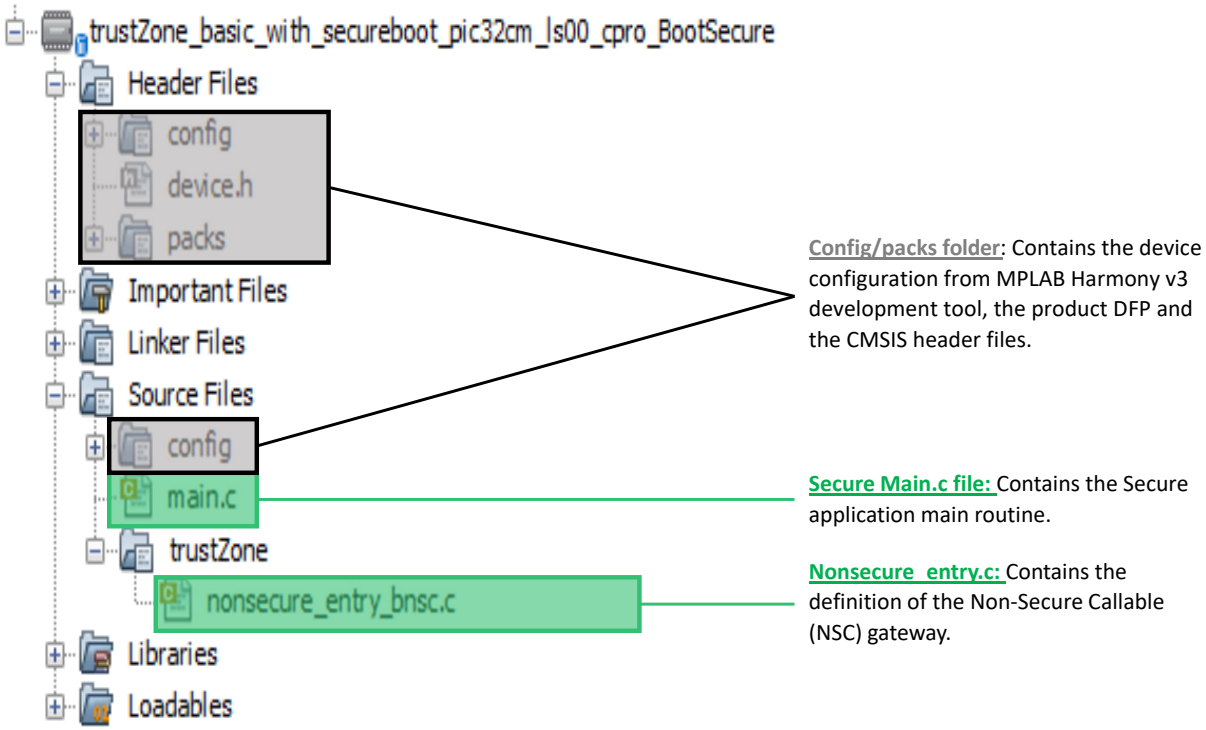
3.5.2.1 Secure Boot Project Description

The goal of the Secure Boot project included in the `trustZone_basic_with_SBoot` example is to provide a preconfigured development base for Secure boot code development on the PIC32CM LS00/LS60. The Secure project is preconfigured to illustrate the following aspects of a standard Secure application on the PIC32CM LS00/LS60:

- Definition and declaration of Secure Boot gateways with the Non-Secure world (veneers)
- Secure call to the Secure application

The following figure illustrates the file architecture of the pre-configured BootSecure project:

Figure 3-39. TrustZone_basic_with_SBoot Example: BootSecure Project Architecture



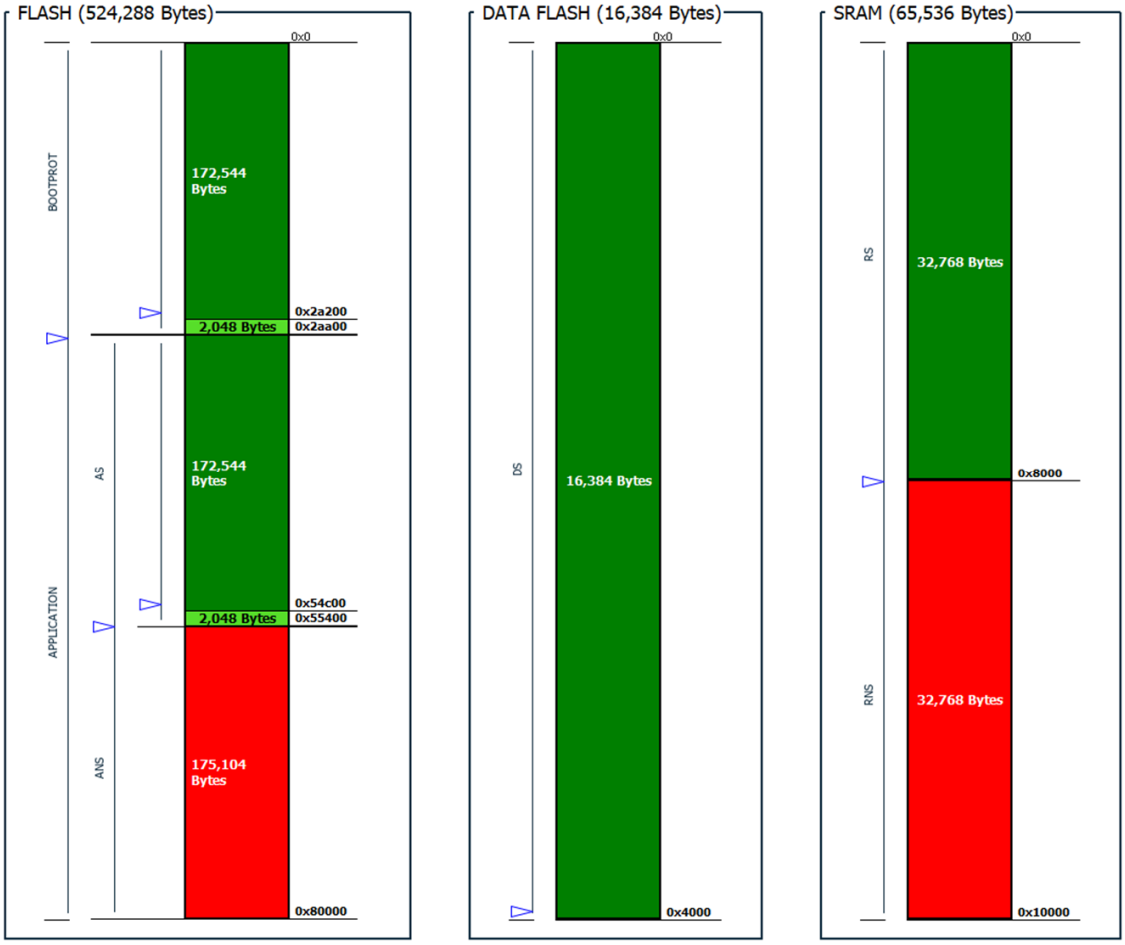
Note:

1. The Non-Secure project provided in the PIC32CM LS00/LS60 `trustZone_basic_with_SBoot` example has the same architecture than the one presented for the `trustZone_basic` example. But it also includes the BootSecure loadable project and the `nonsecure_entry_bnsc.h` file containing the boot secure functions.

3.5.2.2 Memory Configuration with Arm® TrustZone® for Armv8-M Manager

The following figure shows the PIC32CM LS00/LS60 memory configuration for the `trustZone_basic_with_SBoot` example:

Figure 3-40. TrustZone_basic_with_SBoot Example: Memory Configuration



3.5.2.3 NVM Configuration Bit Fields Configuration

The following fuse sizes are defined in the preprocessor macros for the project:

Figure 3-41. Preprocessor Macros Configuration

```
ANSC_SIZE=0x800
AS_SIZE=0x2aa00
BOOTPROT_SIZE=0x2aa00
RS_SIZE=0x8000
BNSC_SIZE=0x800
```

Note: The other fuses are set with their default value.

3.5.2.4 Enabling Secure Boot Process with BS Verification

Follow these steps to enable the Secure Boot process verification when working on an MPLAB Harmony v3 TrustZone project:

1. Perform a `ChipErase_ALL` command in Device Actions.
2. Run the `trustZone_basic_with_SBoot` application using MPLAB X IDE.
3. Change BOOTOPT fuse to 0x01, 0x02 or 0x03 using the Configuration Bits tool:

Figure 3-42. Secure Boot Process with BOOTPROT Verification

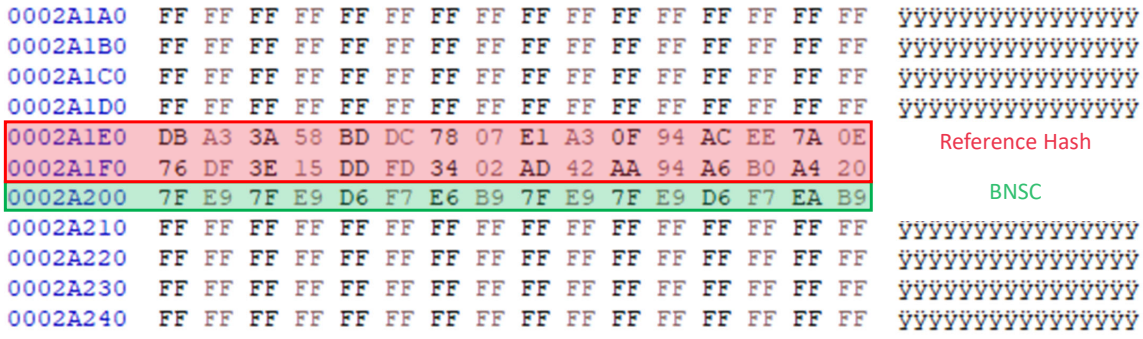
Configuration Bits ×

Address	Name	Value	Field	Option
			NONSECC_OPAMP	CLEAR
			NONSECC_TRAM	CLEAR
0080_401C	USER_WORD_7	00000000	BOOTROM_CDIROFFSET	User range: 0x0 - 0xFFFFFFFF
0080_4020	USER_WORD_8	D80A7A2B	BOOTROM_USERCRC	User range: 0x0 - 0xFFFFFFFF
0080_C000	BOCOR_WORD_0	F207FFFF	IDAU_BNSC	User range: 0x0 - 0x1FF
0080_C004	BOCOR_WORD_1	FFEAAA01	BOOTROM_BOOTOPT	User range: 0x0 - 0xFF
		000000AA	IDAU_BOOTPROT	User range: 0x0 - 0x7FF
			BOOTROM_SECCFGLOCK	SET
			BOOTROM_DICEEN	CLEAR
			NVMCTRL_BCWEN	SET
			NVMCTRL_BCREN	SET

Note: Setting the BOOTOPT fuse to 0x04, 0x05 or 0x06 on PIC32CM LS60 allows it to benefit from Secure Boot with the ATECC608B. Refer to the PIC32CM LS60 Secure Boot with ATECC608B CryptoAuthentication™ Device chapter for more details on this feature.

The reference hash will then be computed and written in memory automatically once BOOTOPT fuse is set as shown in the following figure:

Figure 3-43. Secure Boot Application Reference Hash



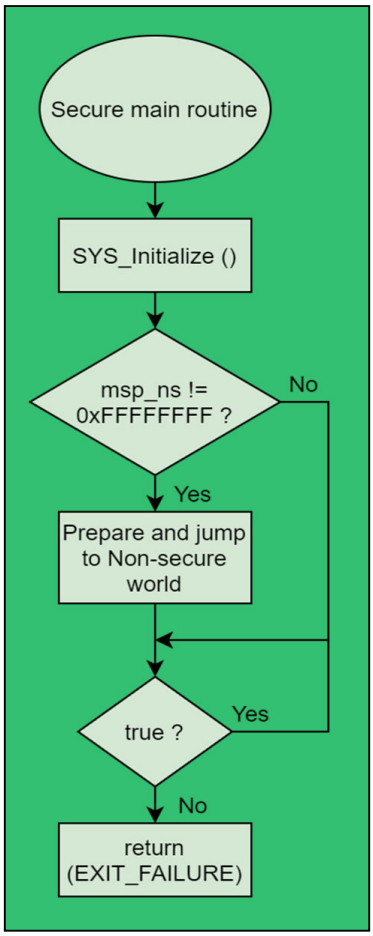
4. Software Use Case Examples

4.1 Non-Secure Peripheral (TC0)

This use case example describes how to configure a PIC32CM LS00/LS60 integrated peripheral (TC0) as a Non-Secure peripheral using MPLAB Harmony v3. In this example, the Secure project allocates the PORT and TC peripherals to the Non-Secure world, sets the system clocks and peripherals, and then jumps to the Non-Secure application.

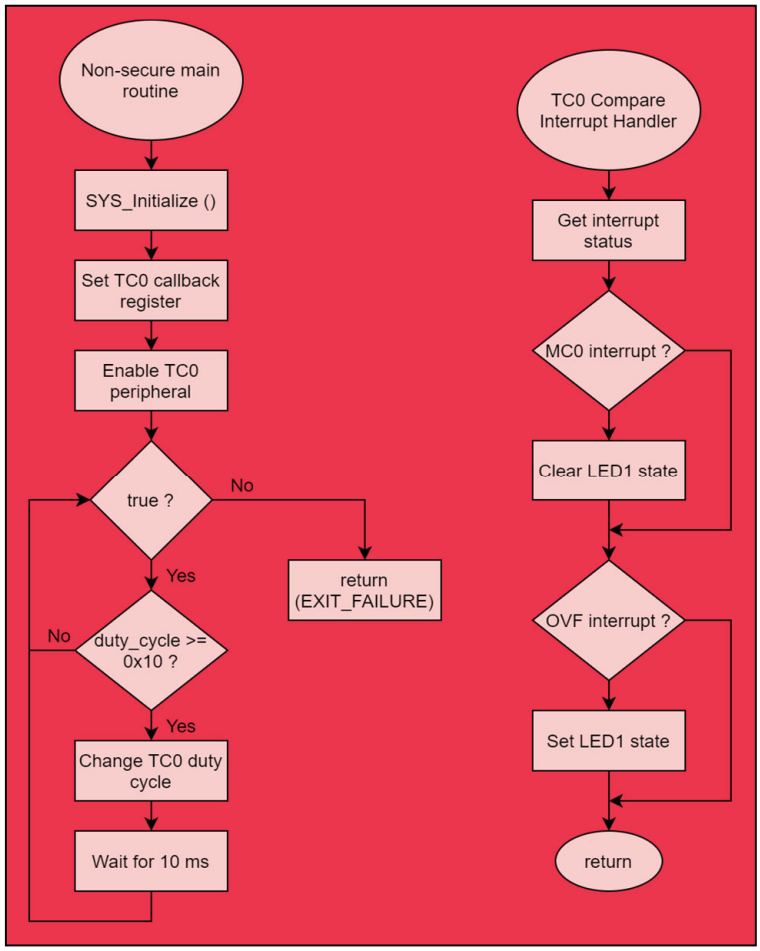
The Non-Secure application uses the TC0 to generate a PWM signal on PC19 (LED0). The following figure illustrates the execution flow of the Secure main routine:

Figure 4-1. Non-Secure TC0 MPLAB Harmony v3 Use Case: Secure Application Flowchart



The following figure illustrates the execution flow of the Non-Secure main routine:

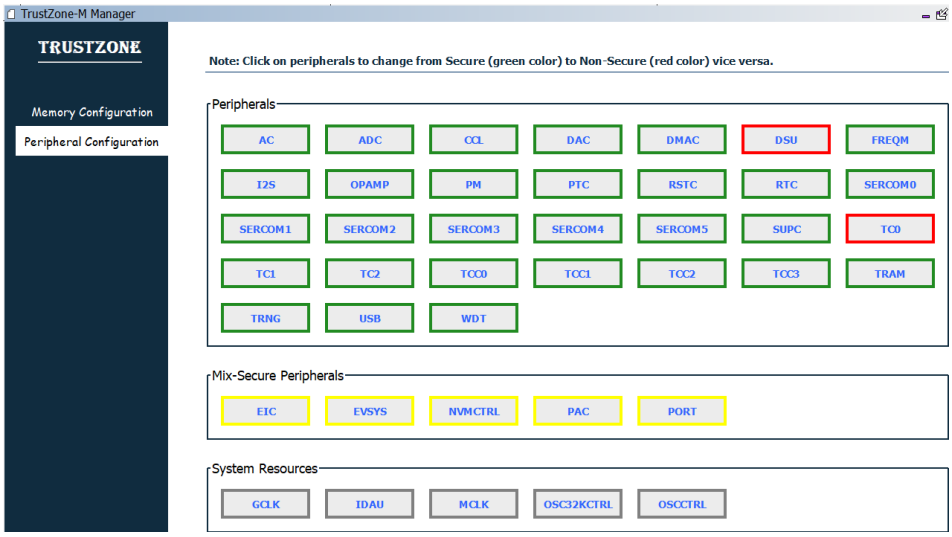
Figure 4-2. Non-Secure TC0 MPLAB Harmony v3 Use Case: Non-Secure Application Flowchart



To ease the secure state configuration of a peripheral, MPLAB Harmony v3 development tool provides the Peripheral Configuration tool in the Arm® TrustZone® for Armv8-M Manager.

For this example, TC0 is set as Non-Secure and the appropriate box is highlighted in red in the Peripheral Configuration window of the Arm® TrustZone® for Armv8-M Manager tool:

Figure 4-3. Non-Secure TC0 MPLAB Harmony v3 Use Case: Peripheral Configuration



The following pictures show the code generated by the MPLAB Harmony v3 development tool to allocate the TC0 peripheral to the Non-Secure world:

- TC0 allocation to the Non-Secure world in the fuses definition (Secure `initialization.c` file).

  Figure 4-4. Non-Secure TC0 MPLAB Harmony v3 Use Case: TC0 Defined as Non-Secure

```
#pragma config NONSECC_SERCOM5 = CLEAR
#pragma config NONSECC_TC0 = SET
#pragma config NONSECC_TC1 = CLEAR
```

- TC0 peripheral interrupt allocation to the Non-Secure world (`plib_nvic.c`).

  Figure 4-5. Non-Secure TC0 MPLAB Harmony v3 Use Case: TC0 Interrupt Defined as Non-Secure

```
NVIC_SetTargetState(TC0_IRQn);
```

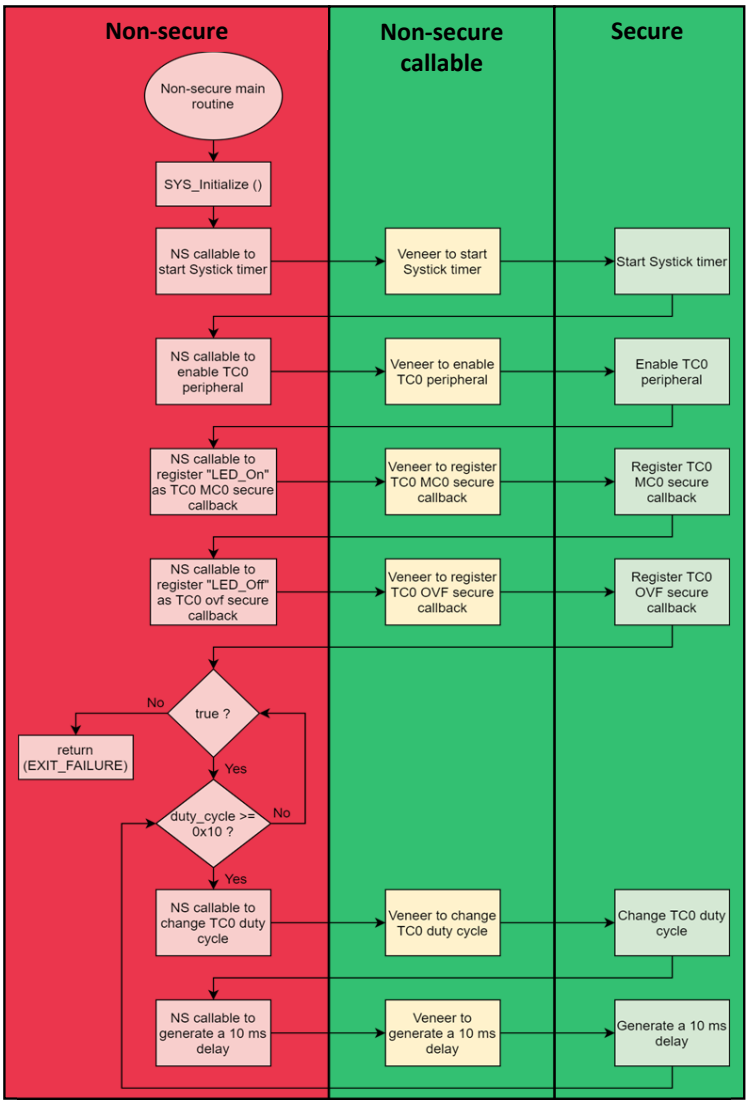
4.2 Secure Peripheral (TC0)

This use case example demonstrates how to configure a PIC32CM LS00/LS60 integrated peripheral (TC0) as a Secure peripheral. In this use case, the Secure project is in charge of configuring system resources and managing the TC peripheral. It also provides specific TC0 APIs and Non-Secure callbacks to the Non-Secure world.

Note: This use case secure main routine is the same as the Non-Secure TC0 use case, but the `SYS_Initialize ()` content is different. Refer to the Non-Secure TC0 MPLAB Harmony v3 Use Case for more details.

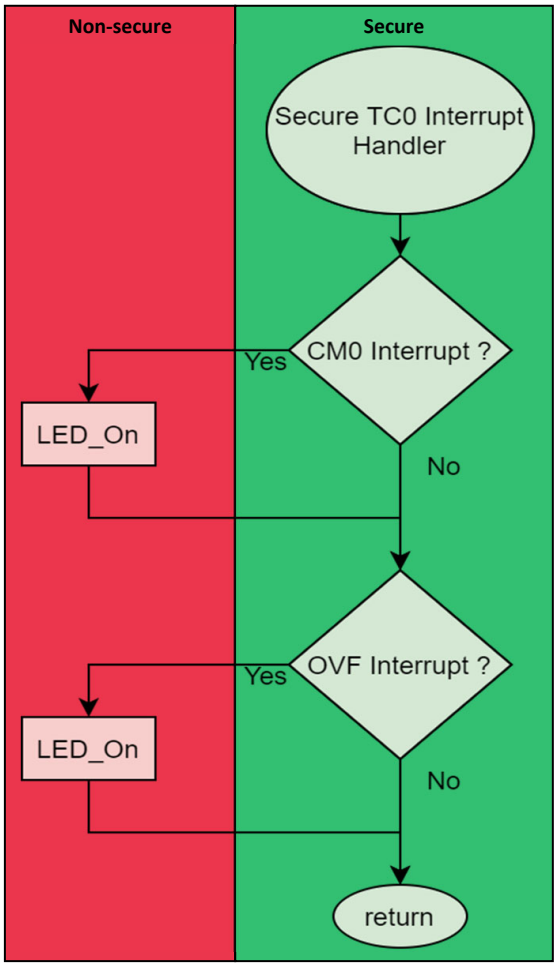
The following figure illustrates the Non-Secure main flowchart of this application:

Figure 4-6. Secure TC0 MPLAB Harmony v3 Use Case: Non-Secure Main Routine Flowchart



The following flowchart illustrates the TC0 interrupt handler routine for this use case:

Figure 4-7. Secure TC0 MPLAB Harmony v3 Use Case: TC Handler Flowchart

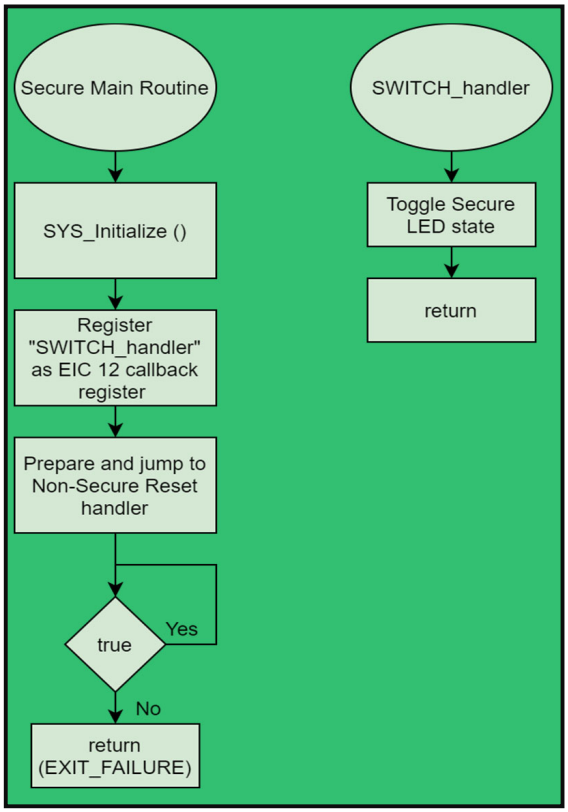


4.3 Mix-Secure Peripheral (EIC)

This use case example shows how to configure and use the PIC32CM LS00/LS60 Mix-Secure peripheral (EIC). Using this example, the user can configure two interrupt lines, EXTIN2 and EXTIN12, and then allocate them to the Non-Secure and Secure world.

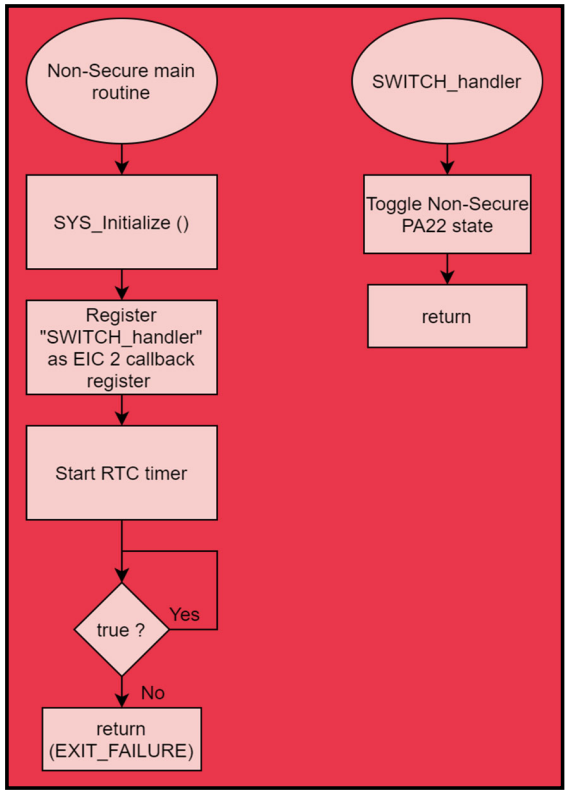
The following figure illustrates the execution flow of the Secure main routine:

Figure 4-8. Mix-Secure EIC MPLAB Harmony v3 Use Case: Secure Application Flowchart



The following figure illustrates the execution flow of the Non-Secure main routine:

Figure 4-9. Mix-Secure EIC MPLAB Harmony v3 Use Case: Non-Secure Application Flowchart



4.4 TrustRAM

The TrustRAM (TRAM) embedded in the PIC32CM LS00/LS60 offers the following advanced security features for secure information storage:

- Address and data scrambling
- Silent access
- Data remanence
- Active shielding and tamper detection
- Full erasure of scramble key and RAM data on tamper detection

In this example, the TrustRAM content is displayed and refreshed every second on a console (USART 3), allowing users to experiment with static and dynamic tamper detections coupled with a TrustRAM full erase.

Figure 4-10. TrustRAM MPLAB Harmony v3 Use Case: Output

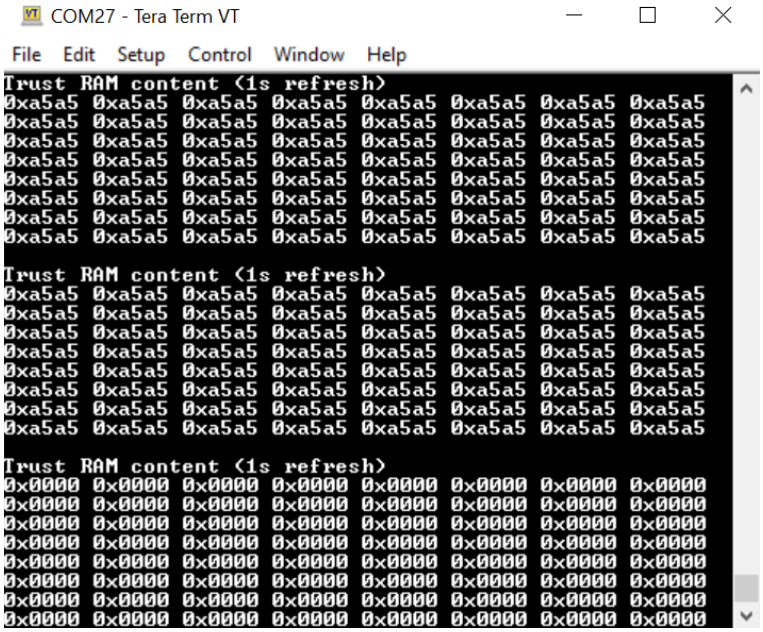


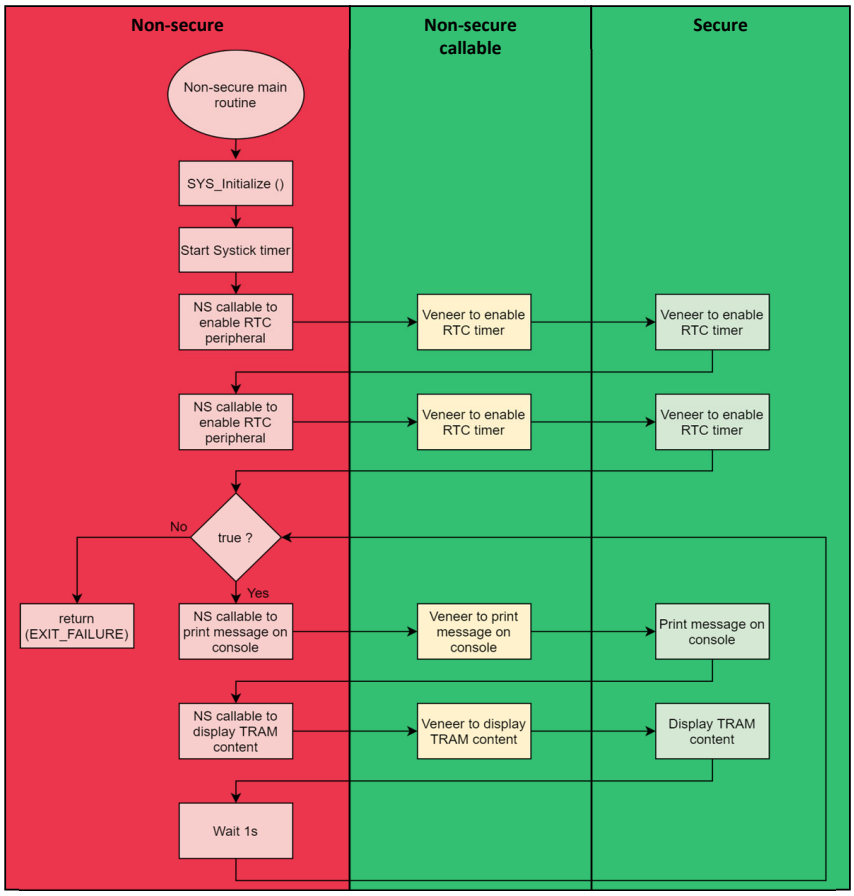
Figure 4-11. TrustRAM MPLAB Harmony v3 Use Case: TRAM Physical Content (Not Erased)

Address	Name	Hex	Decimal	Binary	Char
4200_5600	RAM0	0x48B76E91	121998...	01001000 10110111 01101110 10010001	'H·n□'
4200_5604	RAM1	0x48B76E91	121998...	01001000 10110111 01101110 10010001	'H·n□'
4200_5608	RAM2	0x48B76E91	121998...	01001000 10110111 01101110 10010001	'H·n□'
4200_560C	RAM3	0x48B76E91	121998...	01001000 10110111 01101110 10010001	'H·n□'
4200_5610	RAM4	0x48B76E91	121998...	01001000 10110111 01101110 10010001	'H·n□'
4200_5614	RAM5	0x48B76E91	121998...	01001000 10110111 01101110 10010001	'H·n□'
4200_5618	RAM6	0x48B76E91	121998...	01001000 10110111 01101110 10010001	'H·n□'
4200_561C	RAM7	0x48B76E91	121998...	01001000 10110111 01101110 10010001	'H·n□'

Note: This use case secure main routine is the same as the Non-Secure TC0 use case but the `SYS_Initialize()` content is different. Refer to the TrustRAM MPLAB Harmony v3 Use Case for more details.

The following figure illustrates the execution flow of the non-secure main routine:

Figure 4-12. TrustRAM MPLAB Harmony v3 Use Case: Non-Secure Application Flowchart



4.5 Data Flash

The Data Flash embedded in the PIC32CM LS00 offers the following advanced security features for secure information storage:

- Data Flash Scramble
- Silent access to selected row (TEROW)
- Tamper erase of selected row (TEROW) on tamper detection

There are two Data Flash use cases explained below, that illustrate the configuration of the NVMCTRL for Secure Data Flash management:

1. Data Flash Scrambling activated with key: 0x123456.
2. Silent access enabled on the first Data Flash ROW.

In these examples, the Secure Data Flash row 0 is erased and its content is displayed. Then the Secure Data Flash security features are enabled, the row 0 is filled with a 0xCAFEDECA pattern, and the row 0 content is displayed on the console.

4.5.1 Secure Data Flash with Data Flash Scramble

Figure 4-13. Secure Data Flash with Data Flash Scramble MPLAB Harmony v3 Use Case: Output

```
############################################
#        Secure DataFlash example          #
############################################

 - Erase TEROW

 - Print TEROW Page 0 & Page 1 content :
 Page 0 : 0x400000 : FFFFFFFF FFFFFFFF
 Page 0 : 0x400008 : FFFFFFFF FFFFFFFF
 Page 0 : 0x400010 : FFFFFFFF FFFFFFFF
 Page 0 : 0x400018 : FFFFFFFF FFFFFFFF
 Page 0 : 0x400020 : FFFFFFFF FFFFFFFF
 Page 0 : 0x400028 : FFFFFFFF FFFFFFFF
 Page 0 : 0x400030 : FFFFFFFF FFFFFFFF
 Page 0 : 0x400038 : FFFFFFFF FFFFFFFF
 Page 1 : 0x400040 : FFFFFFFF FFFFFFFF
 Page 1 : 0x400048 : FFFFFFFF FFFFFFFF
 Page 1 : 0x400050 : FFFFFFFF FFFFFFFF
 Page 1 : 0x400058 : FFFFFFFF FFFFFFFF
 Page 1 : 0x400060 : FFFFFFFF FFFFFFFF
 Page 1 : 0x400068 : FFFFFFFF FFFFFFFF
 Page 1 : 0x400070 : FFFFFFFF FFFFFFFF
 Page 1 : 0x400078 : FFFFFFFF FFFFFFFF

 - Enable Security Features (Data Flash Scramble - Tamper Erase)

 - Write 0xCAFEDECA pattern in TEROW

 - Print TEROW Page 0 & Page 1 content :
 Page 0 : 0x400000 : CAFEDECA CAFEDECA
 Page 0 : 0x400008 : CAFEDECA CAFEDECA
 Page 0 : 0x400010 : CAFEDECA CAFEDECA
 Page 0 : 0x400018 : CAFEDECA CAFEDECA
 Page 0 : 0x400020 : CAFEDECA CAFEDECA
 Page 0 : 0x400028 : CAFEDECA CAFEDECA
 Page 0 : 0x400030 : CAFEDECA CAFEDECA
 Page 0 : 0x400038 : CAFEDECA CAFEDECA
 Page 1 : 0x400040 : CAFEDECA CAFEDECA
 Page 1 : 0x400048 : CAFEDECA CAFEDECA
 Page 1 : 0x400050 : CAFEDECA CAFEDECA
 Page 1 : 0x400058 : CAFEDECA CAFEDECA
 Page 1 : 0x400060 : CAFEDECA CAFEDECA
 Page 1 : 0x400068 : CAFEDECA CAFEDECA
 Page 1 : 0x400070 : CAFEDECA CAFEDECA
 Page 1 : 0x400078 : CAFEDECA CAFEDECA
```

Figure 4-14. Secure Data Flash with Data Flash Scramble MPLAB Harmony v3 Use Case: Data Flash Physical Content (Not Erased)

									DATAFLASH Me...
0040_0000	EA9C	FEA8	EA9C	FEA8	EA9C	FEA8	EA9C	FEA8	....
0040_0010	EA9C	FEA8	EA9C	FEA8	EA9C	FEA8	EA9C	FEA8	....
0040_0020	EA9C	FEA8	EA9C	FEA8	EA9C	FEA8	EA9C	FEA8	....
0040_0030	EA9C	FEA8	EA9C	FEA8	EA9C	FEA8	EA9C	FEA8	....
0040_0040	EA9C	FEA8	EA9C	FEA8	EA9C	FEA8	EA9C	FEA8	....
0040_0050	EA9C	FEA8	EA9C	FEA8	EA9C	FEA8	EA9C	FEA8	....·
0040_0060	EA9C	FEA8	EA9C	FEA8	EA9C	FEA8	EA9C	FEA8	....
0040_0070	EA9C	FEA8	EA9C	FEA8	EA9C	FEA8	EA9C	FEA8	....

4.5.2 Secure Data Flash with Silent Access

Figure 4-15. Secure Data Flash with Silent Access MPLAB Harmony v3 Use Case: Output

```
##########################################
#         Secure DataFlash example       #
##########################################

- Erase TEROW

- Print TEROW Page 0 & Page 1 content :
Page 0 : 0x400000 : FFFFFFFF FFFFFFFF
Page 0 : 0x400008 : FFFFFFFF FFFFFFFF
Page 0 : 0x400010 : FFFFFFFF FFFFFFFF
Page 0 : 0x400018 : FFFFFFFF FFFFFFFF
Page 0 : 0x400020 : FFFFFFFF FFFFFFFF
Page 0 : 0x400028 : FFFFFFFF FFFFFFFF
Page 0 : 0x400030 : FFFFFFFF FFFFFFFF
Page 0 : 0x400038 : FFFFFFFF FFFFFFFF
Page 1 : 0x400040 : FFFFFFFF FFFFFFFF
Page 1 : 0x400048 : FFFFFFFF FFFFFFFF
Page 1 : 0x400050 : FFFFFFFF FFFFFFFF
Page 1 : 0x400058 : FFFFFFFF FFFFFFFF
Page 1 : 0x400060 : FFFFFFFF FFFFFFFF
Page 1 : 0x400068 : FFFFFFFF FFFFFFFF
Page 1 : 0x400070 : FFFFFFFF FFFFFFFF
Page 1 : 0x400078 : FFFFFFFF FFFFFFFF

- Enable Security Features (Silent Access - Tamper Erase)

- Write 0xCAFEDECA pattern in TEROW

- Print TEROW Page 0 & Page 1 content :
Page 0 : 0x400000 : CAFEDECA CAFEDECA
Page 0 : 0x400008 : CAFEDECA CAFEDECA
Page 0 : 0x400010 : CAFEDECA CAFEDECA
Page 0 : 0x400018 : CAFEDECA CAFEDECA
Page 0 : 0x400020 : CAFEDECA CAFEDECA
Page 0 : 0x400028 : CAFEDECA CAFEDECA
Page 0 : 0x400030 : CAFEDECA CAFEDECA
Page 0 : 0x400038 : CAFEDECA CAFEDECA
Page 1 : 0x400040 : CAFEDECA CAFEDECA
Page 1 : 0x400048 : CAFEDECA CAFEDECA
Page 1 : 0x400050 : CAFEDECA CAFEDECA
Page 1 : 0x400058 : CAFEDECA CAFEDECA
Page 1 : 0x400060 : CAFEDECA CAFEDECA
Page 1 : 0x400068 : CAFEDECA CAFEDECA
Page 1 : 0x400070 : CAFEDECA CAFEDECA
Page 1 : 0x400078 : CAFEDECA CAFEDECA
```

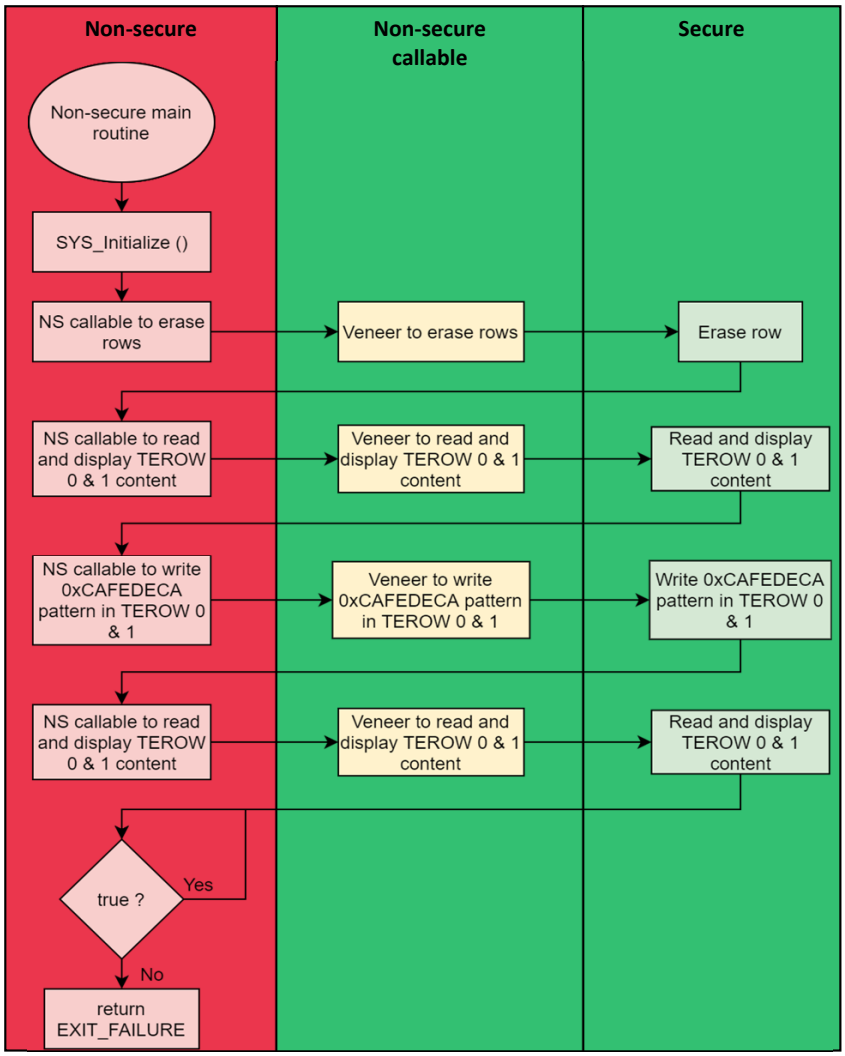
Figure 4-16. Secure Data Flash with Silent Access MPLAB Harmony v3 Use Case: Data Flash Physical Content (Not Erased)

									DATAFLASH Me...
0040_0000	35CA	21DE	01FE	35CA	35CA	21DE	01FE	35CA	.5.!...5 .5.!...5
0040_0010	35CA	21DE	01FE	35CA	35CA	21DE	01FE	35CA	.5.!...5 .5.!...5
0040_0020	35CA	21DE	01FE	35CA	35CA	21DE	01FE	35CA	.5.!...5 .5.!...5
0040_0030	35CA	21DE	01FE	35CA	35CA	21DE	01FE	35CA	.5.!...5 .5.!...5
0040_0040	35CA	21DE	01FE	35CA	35CA	21DE	01FE	35CA	.5.!...5 .5.!...5
0040_0050	35CA	21DE	01FE	35CA	35CA	21DE	01FE	35CA	.5.!...5 .5.!...5
0040_0060	35CA	21DE	01FE	35CA	35CA	21DE	01FE	35CA	.5.!...5 .5.!...5
0040_0070	35CA	21DE	01FE	35CA	35CA	21DE	01FE	35CA	.5.!...5 .5.!...5

Note:
The secure main routine for both examples is the same as the Non-Secure TC0 use case but the `SYS_Initialize()` content is different. Refer to Secure Data Flash with Silent Access MPLAB Harmony v3 Use Case: Non-Secure Application Flowchart use case for more details.

The following figure illustrates the execution flow of the non-secure main routine for Secure Data Flash with Silent Access use case:

Figure 4-17. Secure Data Flash with Silent Access MPLAB Harmony v3 Use Case: Non-Secure Application Flowchart



5. Glossary

The following table provides a list of NVM Configuration bit field acronyms used throughout this document and their definitions:

Acronyms	NVM Configuration Row	Definition
BOOTPROT (BOOT Protection)	BOCOR	Defines the size of the (S) + (NSC) sub-regions of the BOOT region
BOOTOPT (BOOT Option)	BOCOR	Defines the Secure Boot check applied to the BOOTPROT sub-region
BNSC (BOOT Non-Secure Callable)	BOCOR	Defines the size of the (NSC) sub-region of the BOOT region
AS (APPLICATION Secure)	UROW	Defines the size of the (S) + (NSC) sub-regions of the APPLICATION region
ANSC (APPLICATION Non-Secure Callable)	UROW	Defines the size of the (NSC) sub-region of the APPLICATION region
RS (SRAM Secure)	UROW	Defines the size of the (S) region of the SRAM region
DS (Data Flash Secure)	UROW	Defines the size of the (S) region of the Data Flash region

6. References

- PIC32CM LE00/LS00/LS60 Family Data Sheet (DS60001615)
- PIC32CM LE00/LS00/LS60 Family Silicon Errata and Data Sheet Clarifications (DS80000906)
- PIC32CM LE00/LS00/LS60 Curiosity Pro User Guide (DS70005443)

7. Revision History

Revision B - 01/2022

The following updates were incorporated for this revision:

- Updated the product naming throughout the document to add in the PIC32CM LS60
- Updated all references of TrustZone-M Manager to Arm® TrustZone® for Armv8-M Manager throughout the document
- Added content for the CRYA and ATECC608B to the Introduction
- Updated Prerequisites to include the PIC32CM LS60 Curiosity Pro Board and Trust Platform Design Suite
- Introduced new figures to include the PIC32CM LS60 in the following sections:
  - TrustZone for ARMv8-M
  - Single Developer Approach
  - Dual Developer Approach
  - Opening a PIC32CM LS00 TrustZone Example from MPLAB Harmony v3
  - Develop a Non-Secure Project (Developer B)
  - Creating a Non-Secure Project from MPLAB X IDE
  - Align Pre-processor Macros for Linker File to the PIC32CM LS00/LS60 Non-Secure Memories Attributions
  - Adding and Linking Secure Gateway Library to Non-Secure Project
  - Adding and Including Secure Gateway Header File
  - Opening a PIC32CM LS00 TrustZone Example with SHA256-based or HMAC-based Secure Boot from MPLAB Harmony v3
  - Non-Secure Peripheral (TC0)
- Added a PIC32CM LS60 BOCOR Mapping table to Debug Access Level (DAL) and Chip Erase
- Added new notes and a table to Secure Boot
- Added a new note to Develop a TrustZone Example (Developer A)
- Added a new note to Enabling Secure Boot Process with BS Verification
- Added the following NEW sections:
  - Crypto Accelerator (CRYA)
  - PIC32CM LS60 Secure Boot with ATECC608B CryptoAuthentication™ Device

Revision A - 04/2021

This is the initial release of this document.

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods being used in attempts to breach the code protection features of the Microchip devices. We believe that these methods require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Attempts to breach these code protection features, most likely, cannot be accomplished without violating Microchip's intellectual property rights.
- Microchip is willing to work with any customer who is concerned about the integrity of its code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "unbreakable." Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Trademarks

ISBN: 978-1-5224-9647-2

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4485-5910 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-72400 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820