M2351 TrustZone® Program Development

Application Note for 32-bit NuMicro® Family

Document Information

Abstract	Introduce TrustZone® programing including how to partition security attribution and how to develop program in the Keil® MDK environment, and show sample code.
Apply to	NuMicro® M2351 series

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

Table of Contents

1 Overview

In the IoT (Internet of Things) application, devices not only can communicate with each other through the Internet, but can be attacked through the Internet. The security is an important topic to protect device and information. The Arm® TrustZone® technology partitions hardware into Secure and Non-secure world (or trusted and un-trusted world). Users can implement their own authentication method additionally. With the TrustZone® technology and software method, microcontrollers (MCUs) can provide secure application and make it flexible to design. This document will introduce the TrustZone® technology including how to partition security attribution and how to develop program in the Keil® MDK environment, and show sample code.

2 TrustZone® Introduction

The Arm® TrustZone® technology partitions the system into two regions. One is Secure world and another is Non-secure world. The available microcontroller resources including Flash, SRAM, peripherals and peripheral interrupts security attribution can also be configured to Secure or Non-secure. After planning the security attribution of these resources, Non-secure world can only access Non-secure memories and resources, while Secure world can access all memories and resources, including Secure and Non-secure resources. Important data that needs protection can be placed and processed in the Secure world safely. Access of Secure world is limited. The protected data would not be stolen or broken by anyone or anyone untrusted.

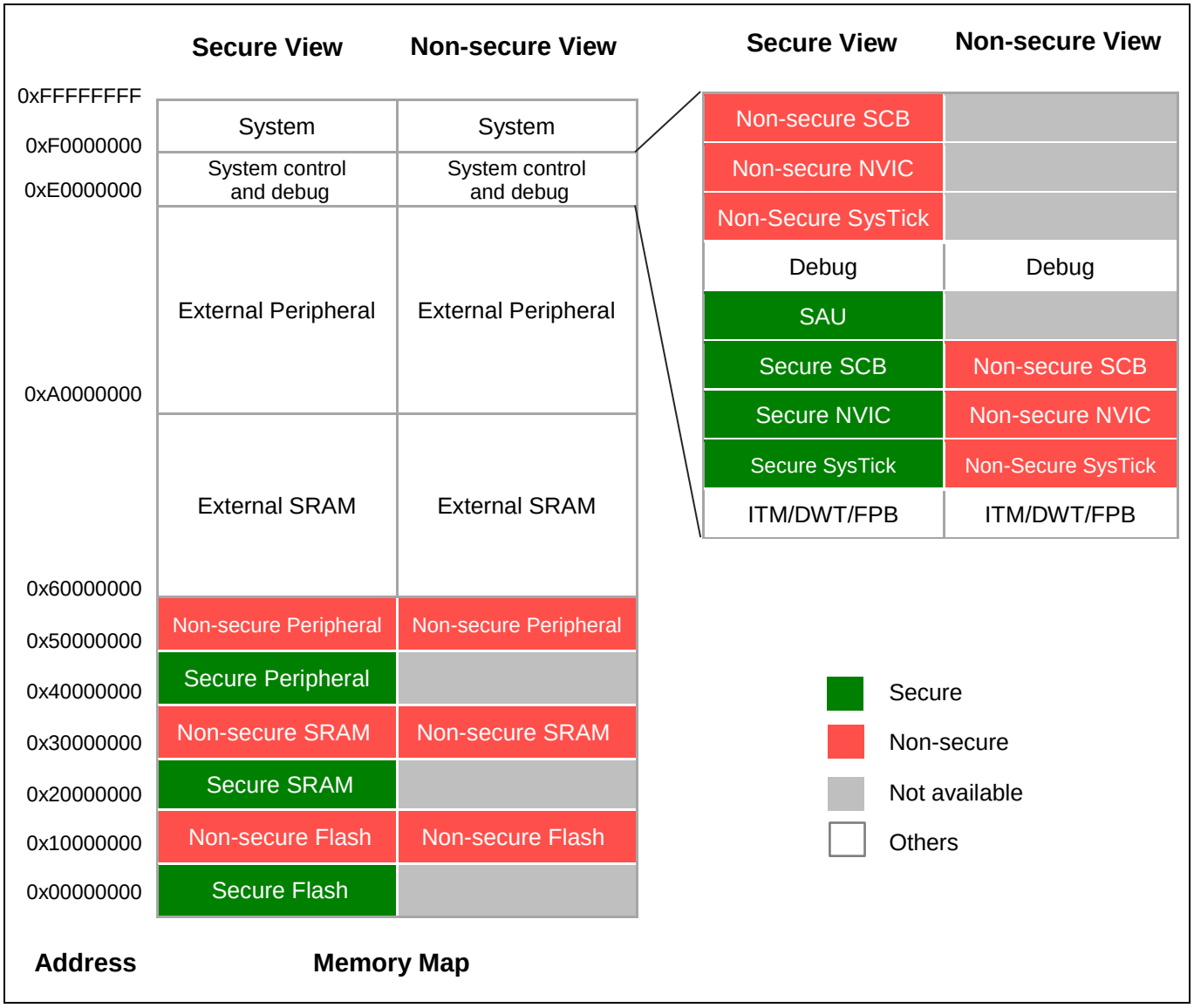


Figure 2-1 Secure and Non-secure World

Figure 2-1 shows the configuration of resource security attribution. The memory map, Flash, SRAM and peripherals are partitioned into Secure or Non-secure attribute. When the code is executed in Secure memory, the system is called Secure code. The Secure code can access all resources including Secure and Non-secure. When the code is executed in the Non-secure memory, the system is called Non-secure code. The Non-secure code can only access Non-secure resource. Beside the defined access authority, Secure code can provide Non-secure callable function to allow more authority for Non-secure code access.

2.1 Memory Map Security Attribution Configuration

In the Arm® TrustZone® technology, the memory map security attribution partition can be configured by IDAU (Implementation Defined Attribution Unit) and SAU (Security Attribution Unit). The security attribution can be configured as Secure(S), Non-secure(NS) and Non-secure callable(NSC). Non-secure callable entry function should be placed in Non-secure callable memory if Secure code provides Non-secure callable function. NuMicro® M2351 series define a fixed memory map security attribution by IDAU. User still can change security configuration partition by SAU. The result of security attribution is the higher security setting between IDAU and SAU. The priority of security attribution is Secure(S) has the highest secure priority, then Non-secure callable(NSC) has lower secure priority and Non-secure(NS) has the lowest secure priority. The undefined region is Secure(S) by default.

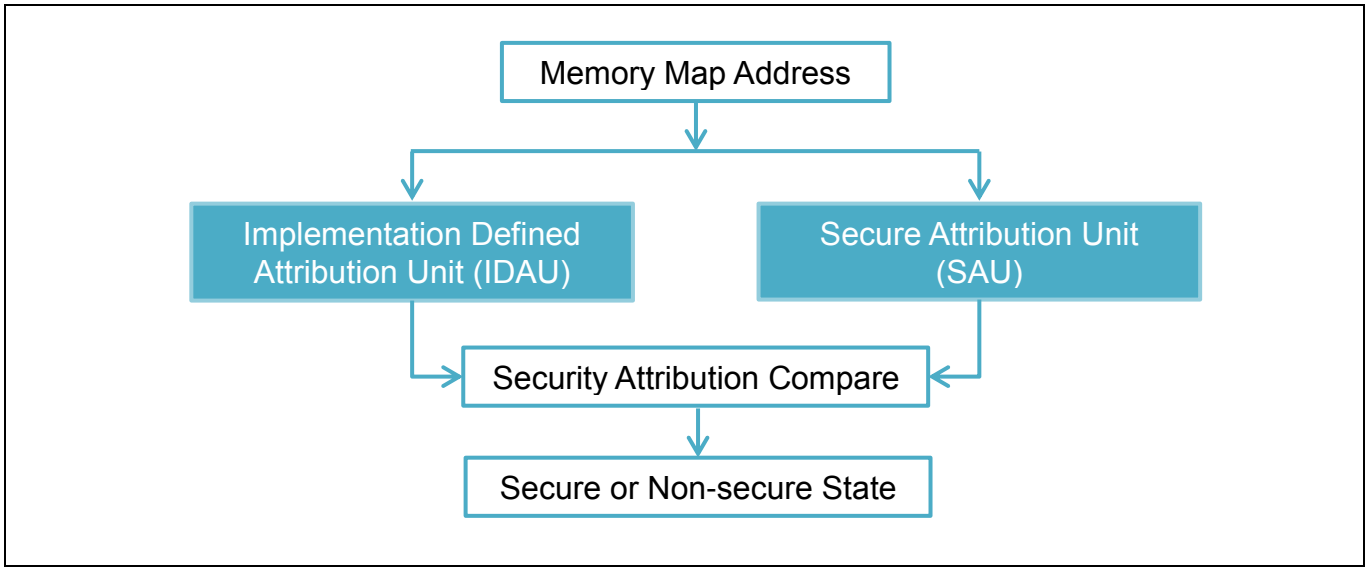


Figure 2-2 Memory Map Address Security Attribution Configuration

2.1.1 IDAU

Figure 2-3 shows the NuMicro® M2351 series memory map security attribution partition defined by IDAU. The memory map address bit 28 is used to define the security attribution. If the memory map address bit 28 is 0, it is Secure(S). In Flash and SRAM region, it is Non-

secure callable (NSC). If the memory map address bit 28 is 1, it is Non-secure(NS). The first 2 Kbytes of Flash are fixed to Secure(S). The system related register does not partition into certain security attribution. They are determined by current system state.

Figure 2-3 IDAU Defined Memory Map Address Security Attribution

2.1.2 SAU

User can change security configuration partition by SAU. Table 2-1 lists the SAU control registers. Figure 2-4 shows how to configure security attribution by SAU. All memory map can be configured to Secure(S) by setting SAU_CTRL.ALLNS=0 or be configured to Non-

secure(NS) by setting SAU_CTRL.ALLNS=1. To configure security attribution in detail, user can set the SAU_CTL, SAU_RNR, SAU_RBAR and SAU_RLAR register. Set SAU_CTL.ENABLE=1 to enable SAU, set SAU_RNR (which region to set), SAU_RBAR (with start address), SAU_RLAR (with end address) and SAU_RLAR.NSC. Set SAU_RLAR.NSC=0 to configure specify memory map region security attribution to Non-secure(NS) and set SAU_RLAR.NSC=1 to configure as Non-secure callable (NSC). The security attribution of the regions which do not specified by SAU is Secure(S).

SAU Register	Address	Description
SAU_CTRL	0xE000EDD0	SAU control register.
SAU_TYPE	0xE000EDD4	The number of SAU setting region, read only.
SAU_RNR	0xE000EDD8	SAU setting region.
SAU_RBAR	0xE000EDDC	SAU setting start address.
SAU_RLAR	0xE000EDE0	SAU setting end address and attribution.

Table 2-1 SAU Control Register

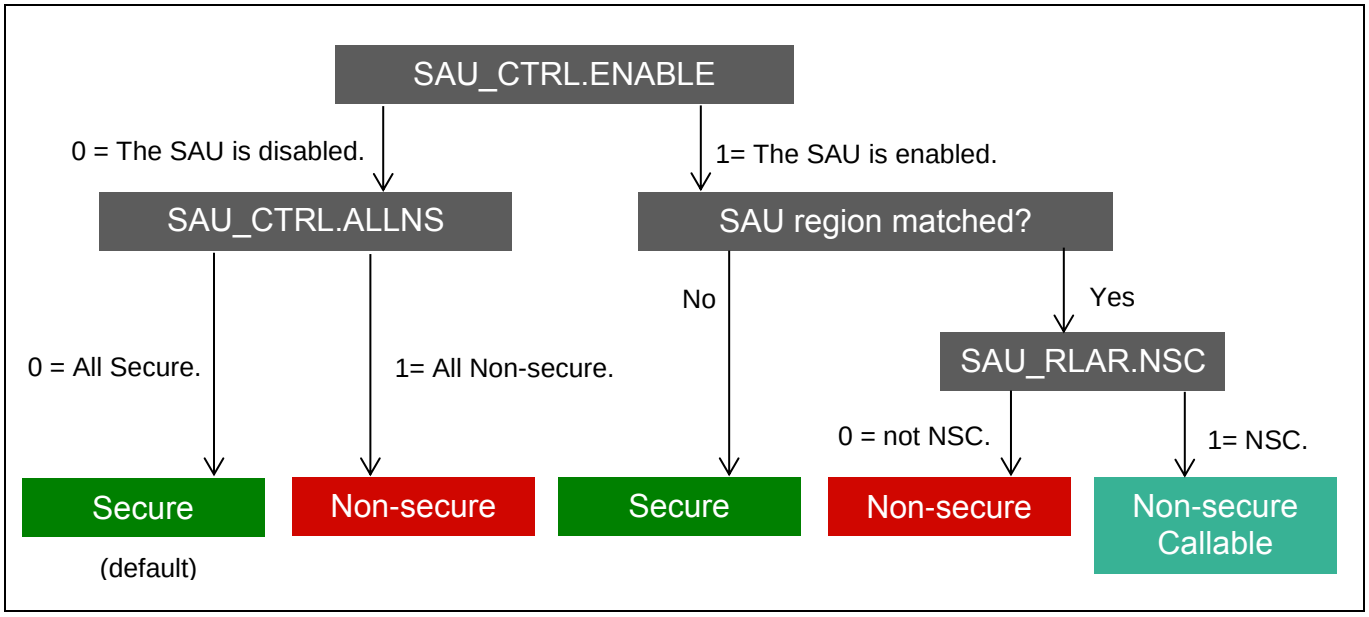


Figure 2-4 SAU Security Attribution Configuration

2.2 Secure and Non-secure State Switch

Functions in Secure code and Non-secure code can call each other. Secure code can call Non-secure function directly but Non-secure code cannot call Secure function directly. If Secure code allows Non-secure code to call a Secure function, it is a Non-secure callable function. Secure code should place related Non-secure callable entry function in Non-secure callable region. Non-secure code call Non-secure callable entry function then can call secure function.

2.2.1 Non-secure Code Calls Secure Function

When Non-secure code calls Secure function, it calls Non-secure callable entry function at first. The first instruction of Non-secure callable entry function is SG instruction. It is the entry point that allows the code state switches from Non-secure state to Secure state. Non-secure callable function is end with BXNS instruction and returns to Non-secure code.

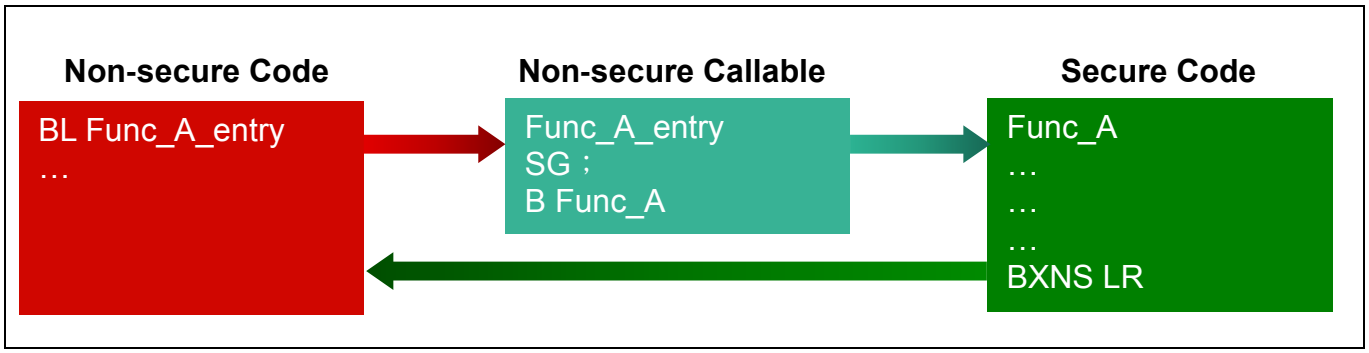


Figure 2-5 Non-secure Code Calls Secure Function

BL: Branch with link instruction

Func_A_entry: Non-secure callable entry function

SG: Secure gateway instruction

B: Branch instruction

Func_A: Secure function

BXNS: Branch with exchange to Non-secure state instruction

2.2.2 Secure Code Calls Non-secure Function

Secure code uses a BLXNS instruction to call Non-secure function. Before switching secure state to Non-secure state, return address and processor information are pushed into the Secure stack, while the return address on Link Register (LR) is set to a special value FNC_RETURN. Non-secure function completes with branch to FNC_RETURN address. The actual return address unstacks to Secure stack and returns to Secure state.

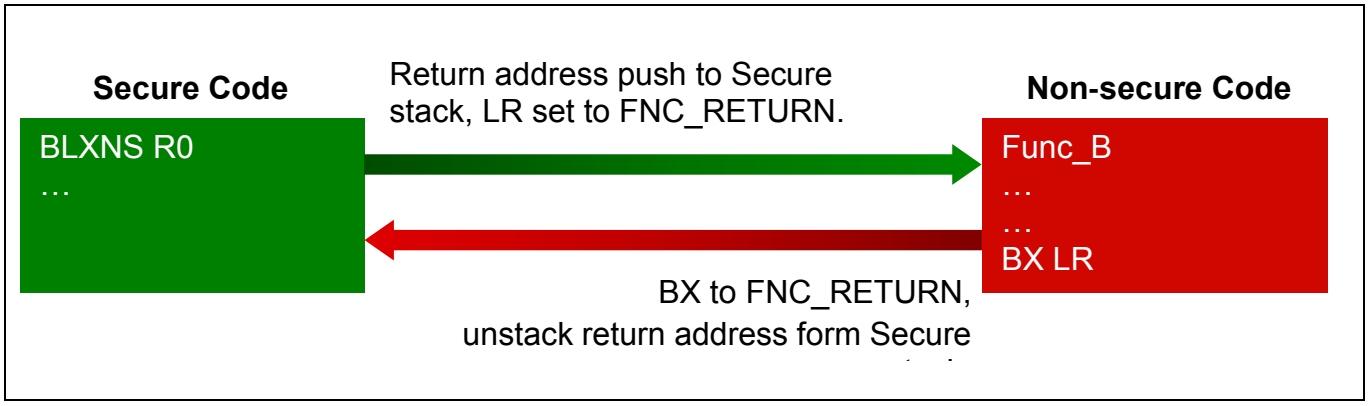


Figure 2-6 Secure Code Calls Non-secure Function

BLXNS: Branch with link and exchange to Non-secure state instruction

R0: Address of Non-secure function with LSB is 0

FNC_RETURN: 0xFEFFFFFF

Func_B: Non-secure function

BX: Branch with exchange instruction

2.2.3 Non-secure Security Attribution Check

Secure code provides Non-secure callable function for Non-secure code to call secure function. It also provides parameters and return values. Secure function can check the memory security by cmse_check_adress_range intrinsic before reading or modifying any data to prevent the important data from being stolen or broken. To use cmse_check_adress_range intrinsic, set the flag parameter as CMSE_NONSECURE, and the address range to check is from p to p+1. If the checked addresses are all in Secure memory, it returns the address p. Otherwise, it returns NULL. Another intrinsic cmse_check_pointed_object can also be used to check address security attribution.

```
void* cmse_check_adress_range(void *p, size_t size, int flags);
void* cmse_check_pointed_object(void *p, int flags);
```

3 Keil® MDK Development Environment

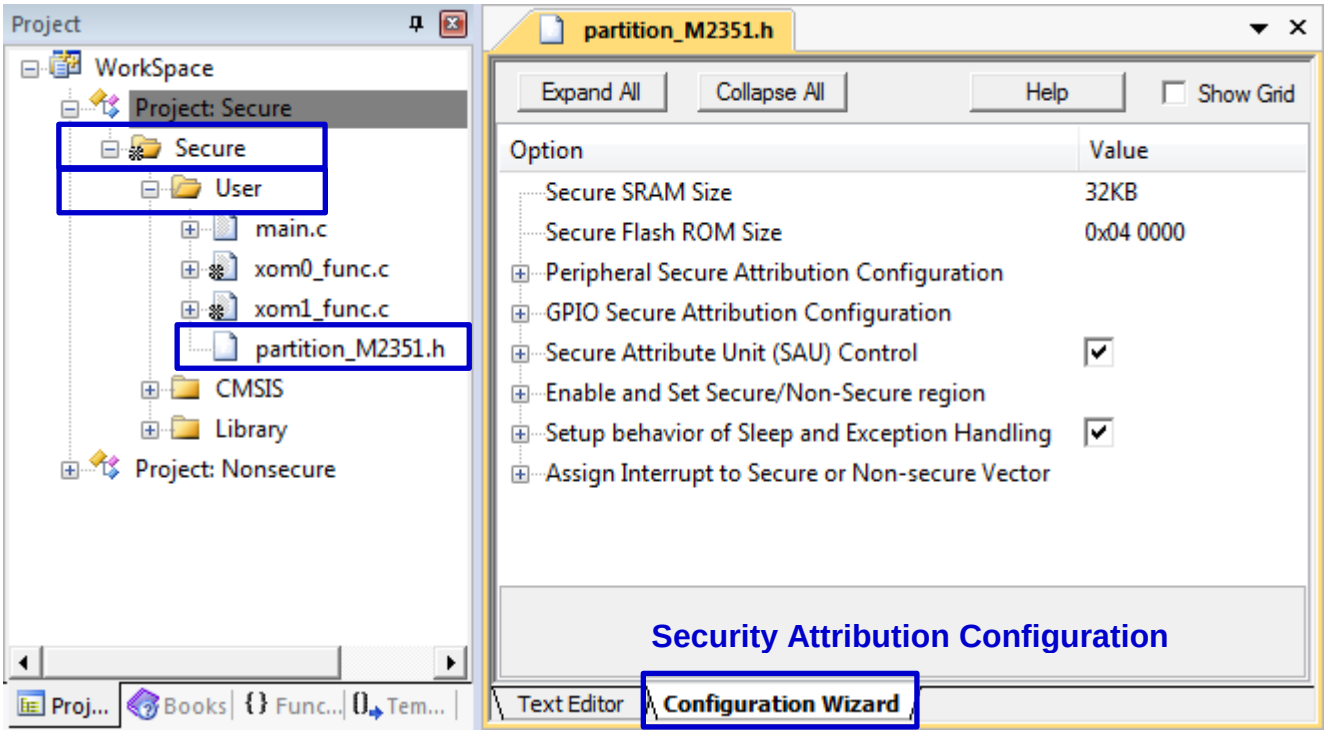
This section describes how to develop NuMicro® M2351 series TrustZone® program with Keil® MDK. The development environment has to support Cortex®-M23 core and Armv8-M architecture. The download and debug tool is Nuvoton Nu-Link Debugger.

The resource security attributions need to be planed first before developing TrustZone® program. User should determine which resources are planned for Secure code and which resources are planned for Non-secure code. The Secure and Non-secure code are two projects. Flash and SRAM address need to be specified to compile and download code correctly. If Secure code provides Non-secure callable function for Non-secure code, Secure project needs to create a Non-secure callable function library. If Non-secure code wants to use Non-secure callable function, Non-secure project has to add Non-secure callable function library. The state switched between Secure and Non-secure state can be observed when executing the code.

3.1 Security Attribution Configuration

Security attribution is configured in Secure code. The partition_M2351.h file provides a configuration wizard interface for user to configure the resources security attribution easily. The security attribution configuration includes memory map, Flash, SRAM, peripherals and peripheral interrupts. The following section will introduce how to configure security attribution through partition_M2351.h file configuration wizard interface.

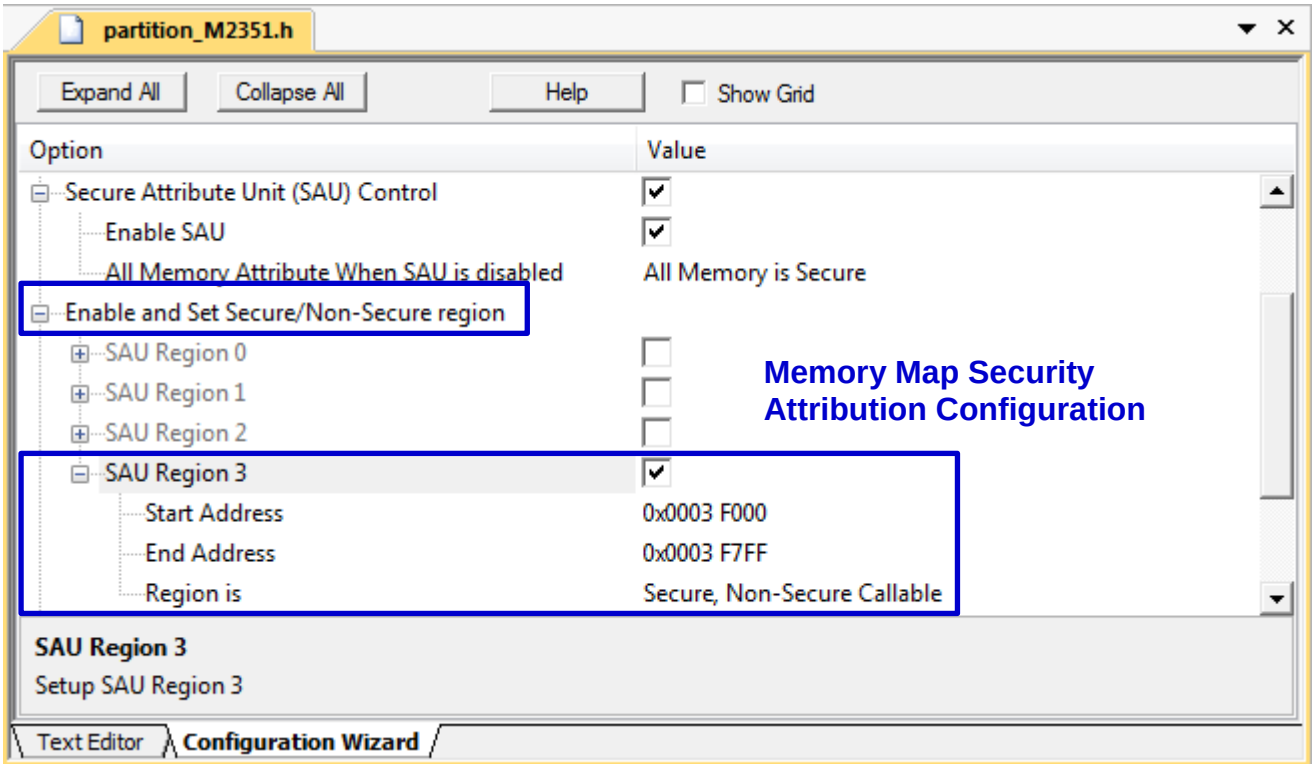
1. Click [Secure] to expand the folder. The [partition_M2351.h] file is in [User] folder.
2. Click [Configuration Wizard] to open Configuration Wizard page.



3.1.1 Memory Map

Memory map security attribution is set by SAU. For example, plan the address 0x3F000-0x3F7FF for Non-secure callable function and set the security attribution as Secure and Non-secure callable.

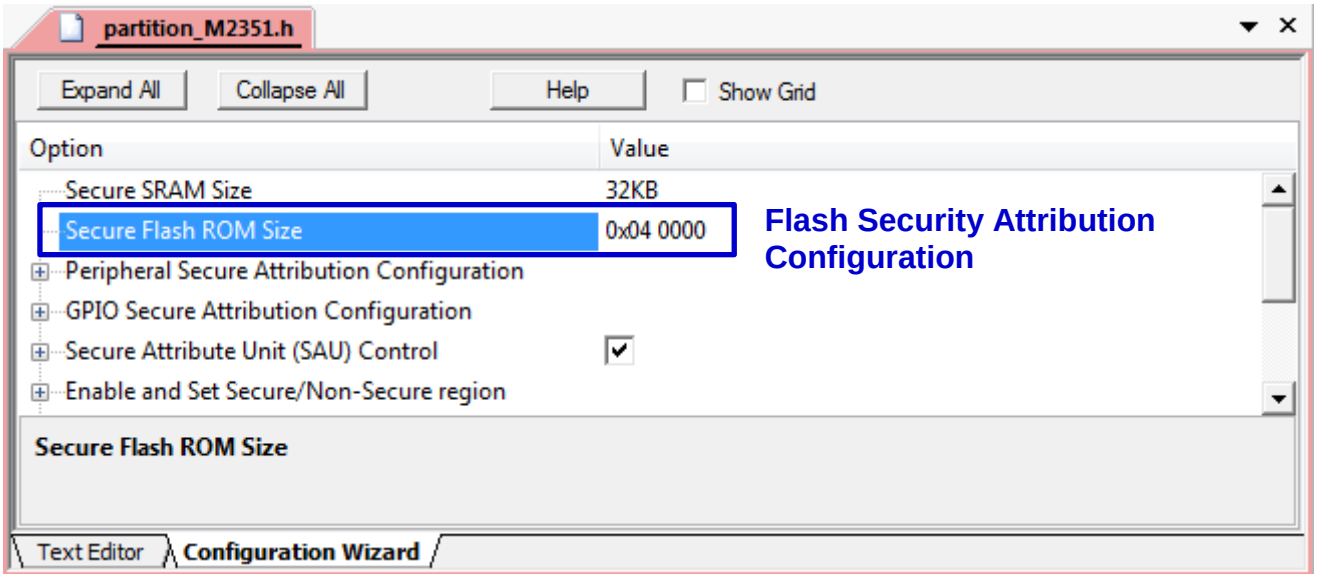
1. In the [Enable and Set Secure/Non-secure Region], select [SAU Region 3].
2. Specify the start address by setting [Start Address] to [0x00003F000].
3. Specify the end address by setting [End Address] to [0x00003F7FF].
4. In the [Region is], select [Secure, Non-secure Callable] from the pull-down menu.



3.1.2 Flash

Flash security attribution is set by the register NSCBA (Non-secure base address register, address 0x00200800). NSCBA sets the start address of Non-secure region in Flash and its read/write is through FMC. User can read register SCU_FNSADDR (Flash Non-secure address register, address 0x4002F028) to get current NSCBA setting. For example, set NSCBA value as 0x40000.

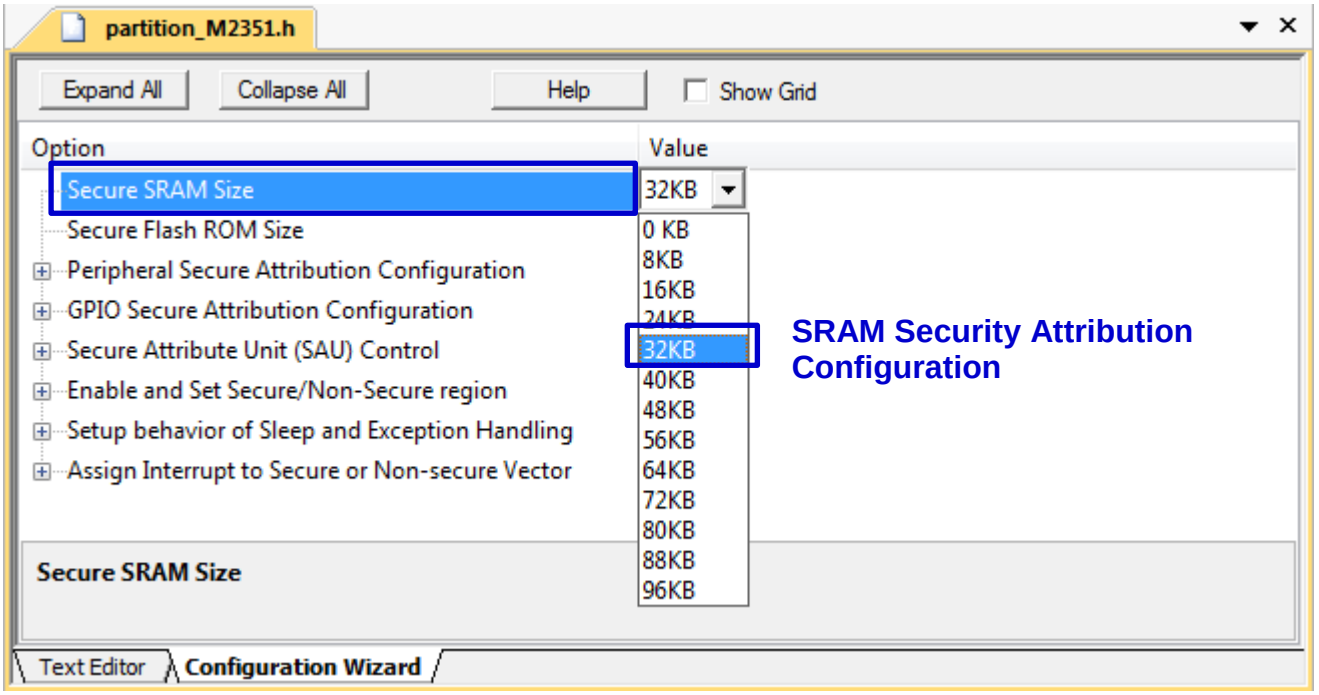
- In the [Secure Flash ROM Size], set Non-secure region start address as [0x040000].



3.1.3 SRAM

SRAM security attribution is set by the register SCU_SRAMNSSET (SRAM secure attribution set register, address 0x4002F024). For example, set the first 32 Kbytes SRAM as Secure (address 0x20000000-0x20007FFF) and set the following 64 Kbytes SRAM as Non-secure (address 0x30008000-0x30017FFF).

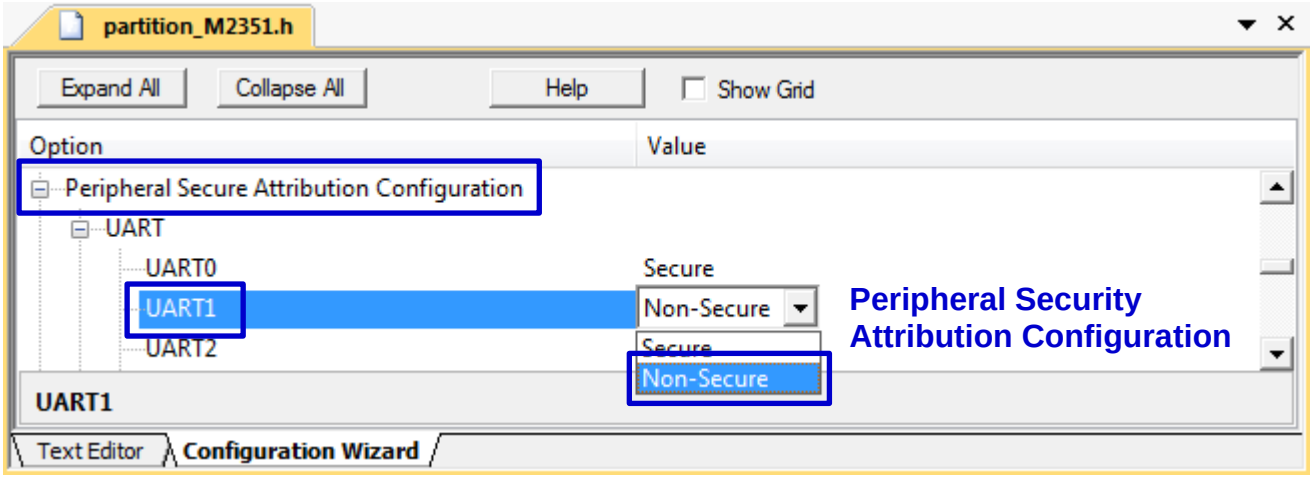
- In the [Secure SRAM Size], select [32KB] from the pull-down menu.



3.1.4 Peripheral

Peripheral security attributions are set by the register SCU_PNSSET0-SCU_PNSSET6 (Peripheral secure attribution set register 0-6, address 0x4002F000-0x4002F018) and SCU_IONSSET (I/O secure attribution set register, address 0x4002F01C). For example, set the UART1 security attribution as Non-secure.

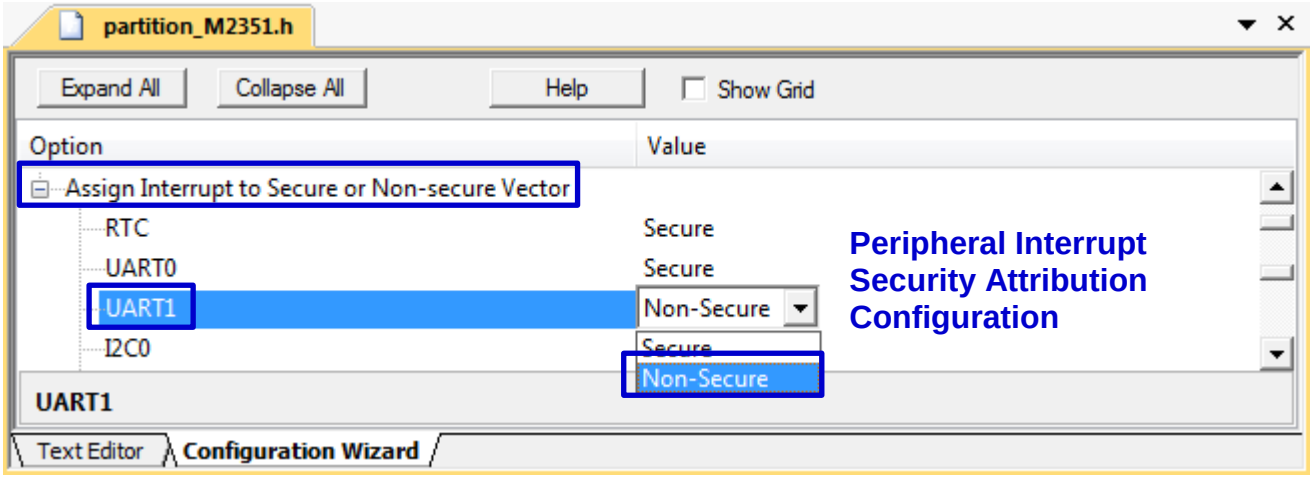
- In the [Peripheral Secure Attribution Configuration] → [UART1], select [Non-secure] from the pull-down menu.



3.1.5 Peripheral Interrupt

Peripheral interrupt security attributions are set by the register NVIC_ITNS0-NVIC_ITNS3 (Interrupt Target Non-secure Register 0-3, address 0xE000_E380-0xE000_E38C). For example, set UART1 interrupt security attribution as Non-secure.

In the [Assign Interrupt to Secure or Non-secure Vector] → [UART1] select [Non-Secure] from the pull-down menu.



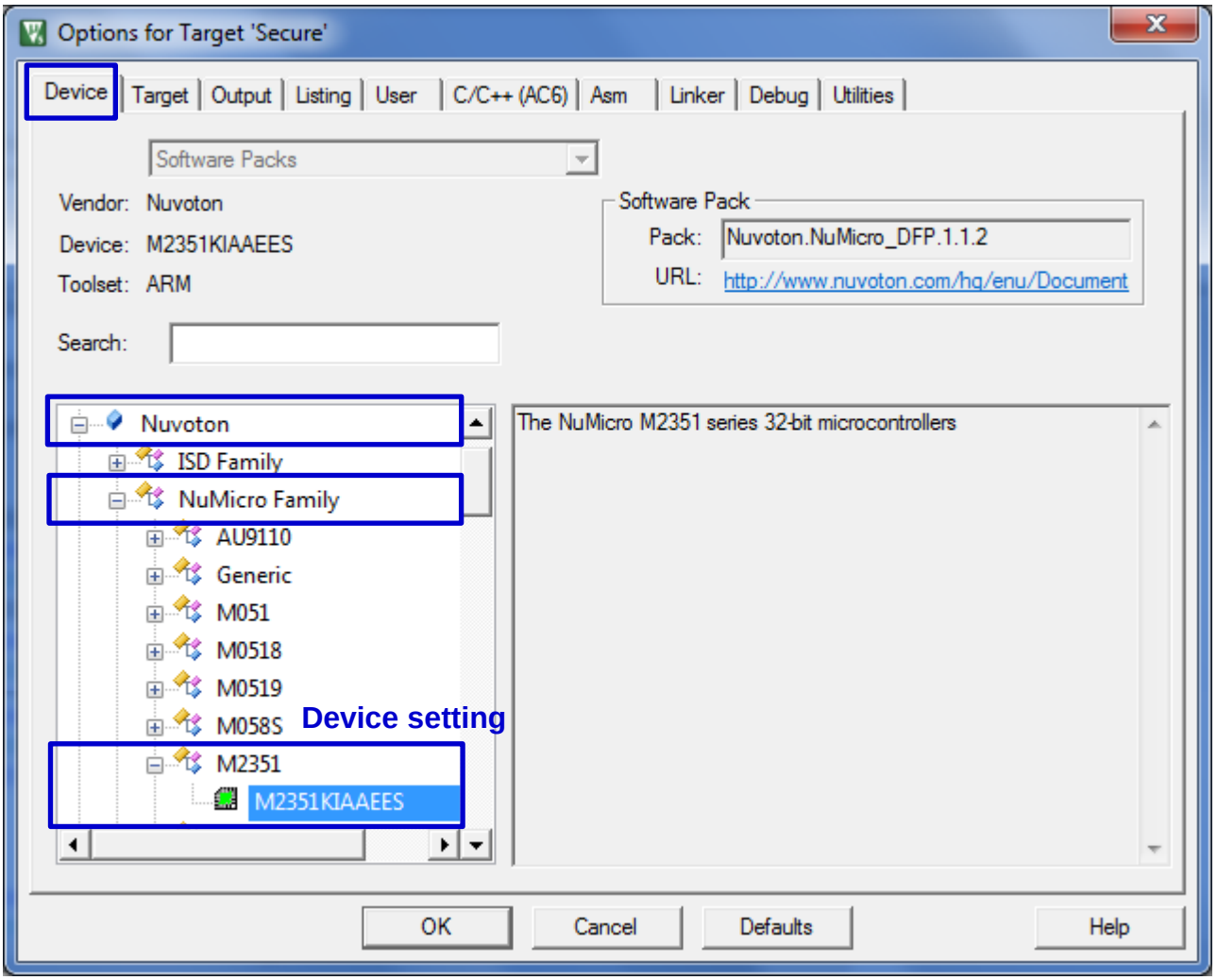
3.2 Secure and Non-secure Project Setting

This section describes Secure and Non-secure project setting. Flash and SRAM address has to be set according to the security attribution configuration. Secure project needs to create Non-secure callable function library if Secure code provides Non-secure callable function. Non-secure project needs to add Non-secure callable function library if Non-secure code wants to use Non-secure callable function.

3.2.1 Secure Code

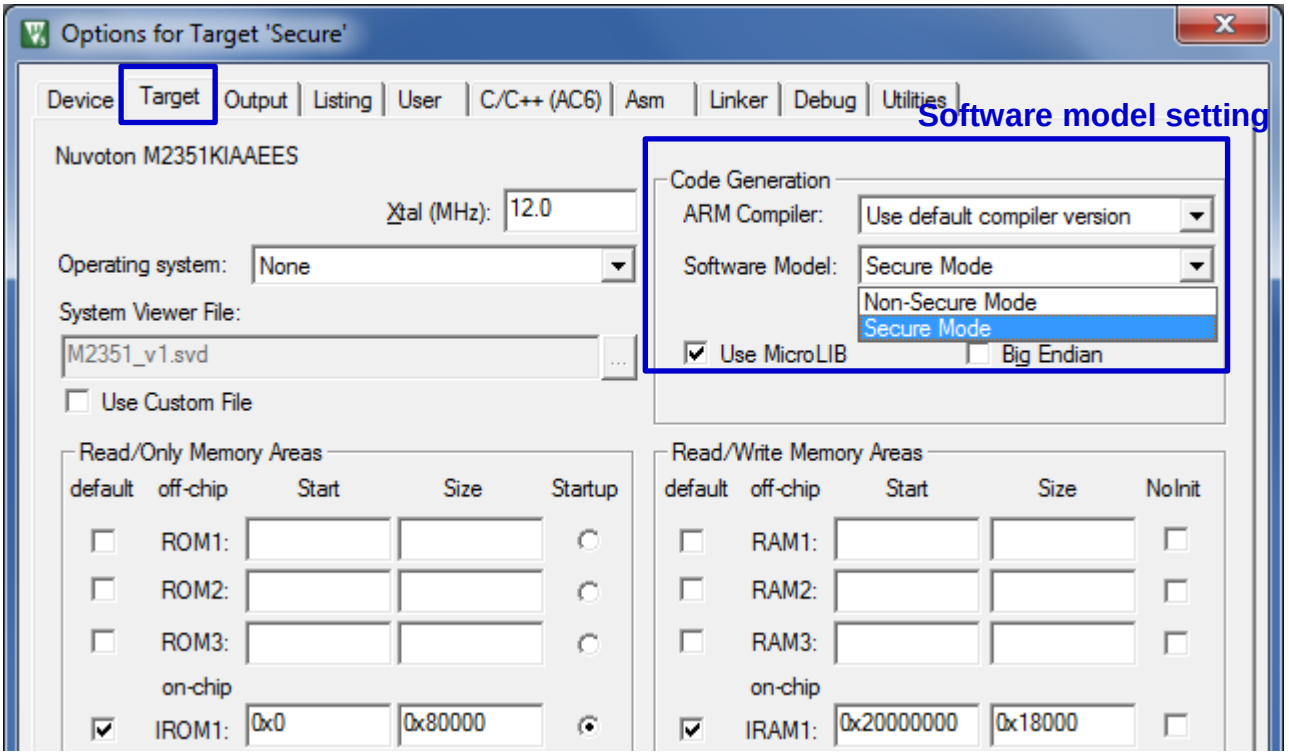
Device setting:

1. Open Secure project in the Keil® MDK environment.
2. Click [Project] on the menu bar and click [Options for Target] to open the configuration window.
3. Under the [Device] page, select Nuvoton → NuMicro Family → M2351 → M2351KIAAEES.



Software model setting:

1. In the [Options for Target] window, click [Target] page.
2. In the [Code Generation] section, select [Software Model] as [Secure Mode].



Flash, SRAM and Non-secure callable entry function location setting:

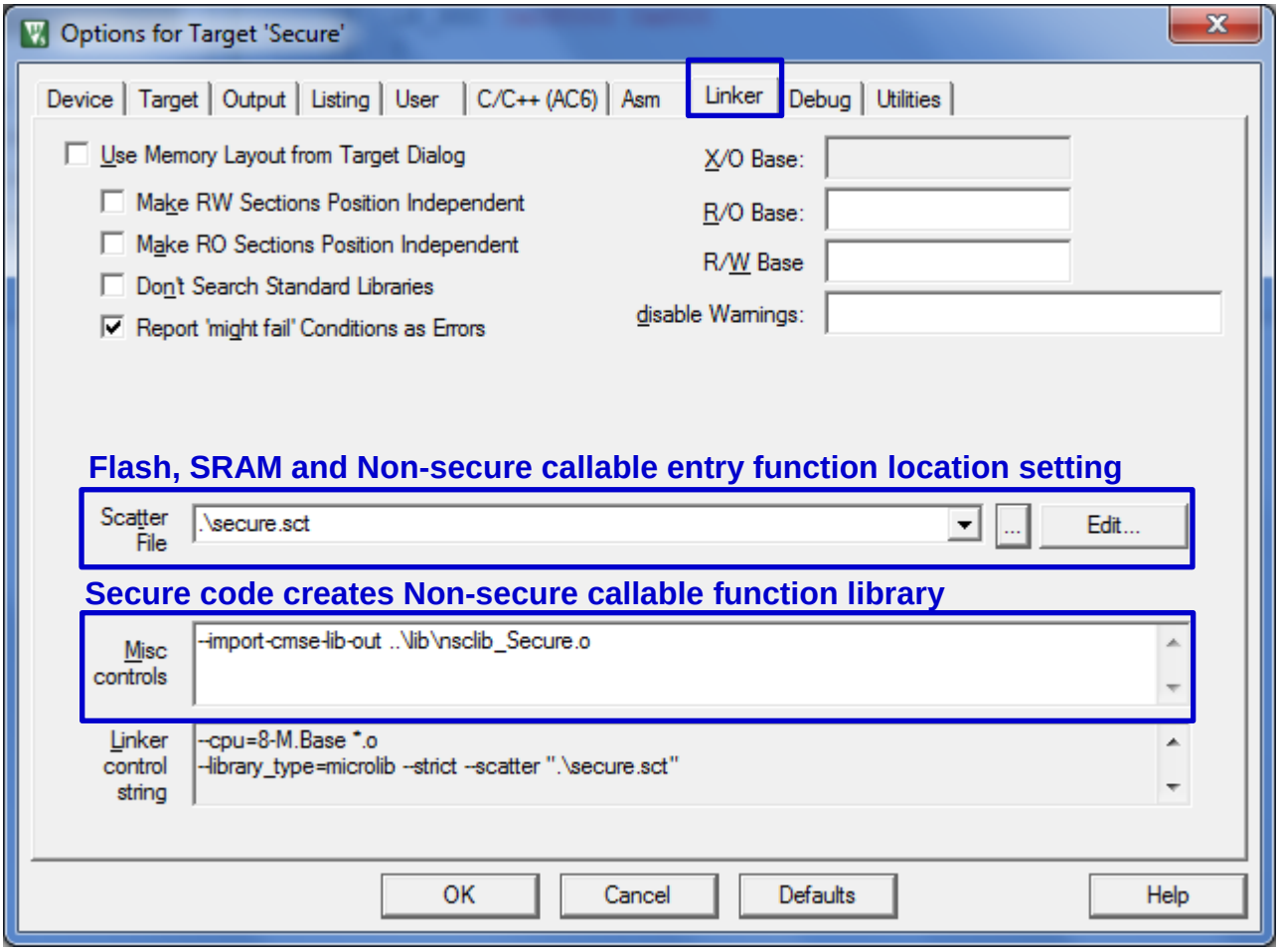
1. In the [Options for Target] window, click [Linker] page.
2. In the [Scatter File] section, click [Edit] button to edit [secure.sct]. Set Secure Flash start address as 0x00000000 and size as 0x10000. Set Secure SRAM start address as 0x20000000 and size as 0x800. Locate Non-secure callable entry function in Veneer$$CMSE region. Set the Non-secure callable region start address is 0x3F000 and size is 0x800.

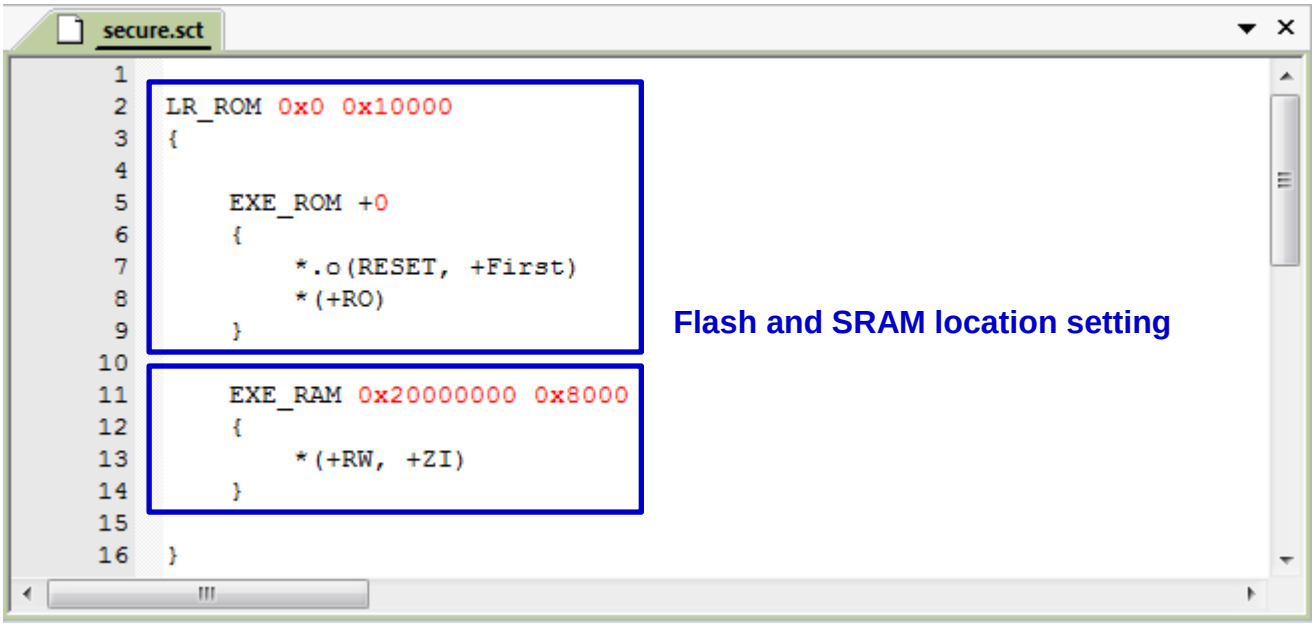
Secure code creates Non-secure callable function library:

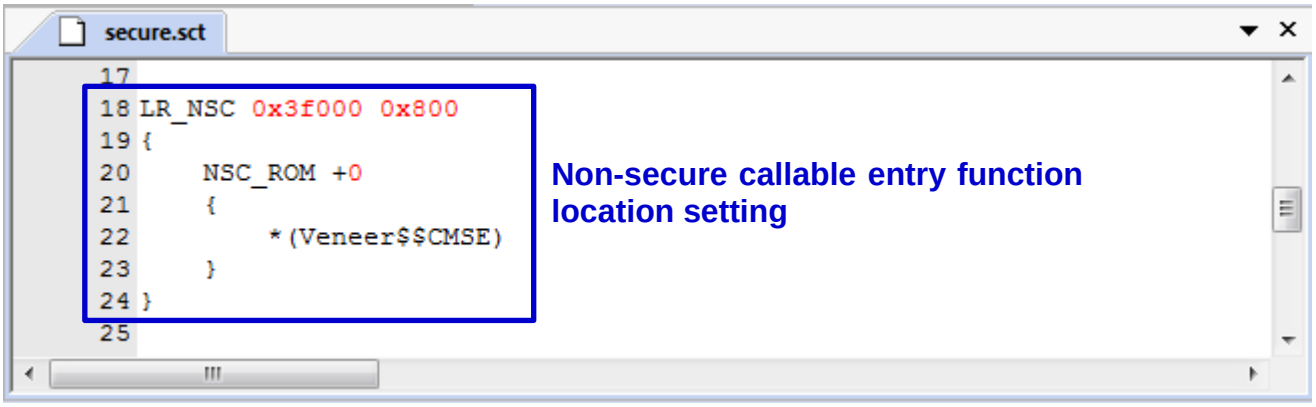
- In the same [Linker] page, under the [Misc controls] section, create the output library nsclib_Secure.o by adding command [--import-cmse-lib-out ..\lib\nsclib_Secure.o].

The Secure code adds “cmse_nonsecure_entry” attribute for Non-secure callable function.

```
attribute__((cmse_nonsecure_entry))
```

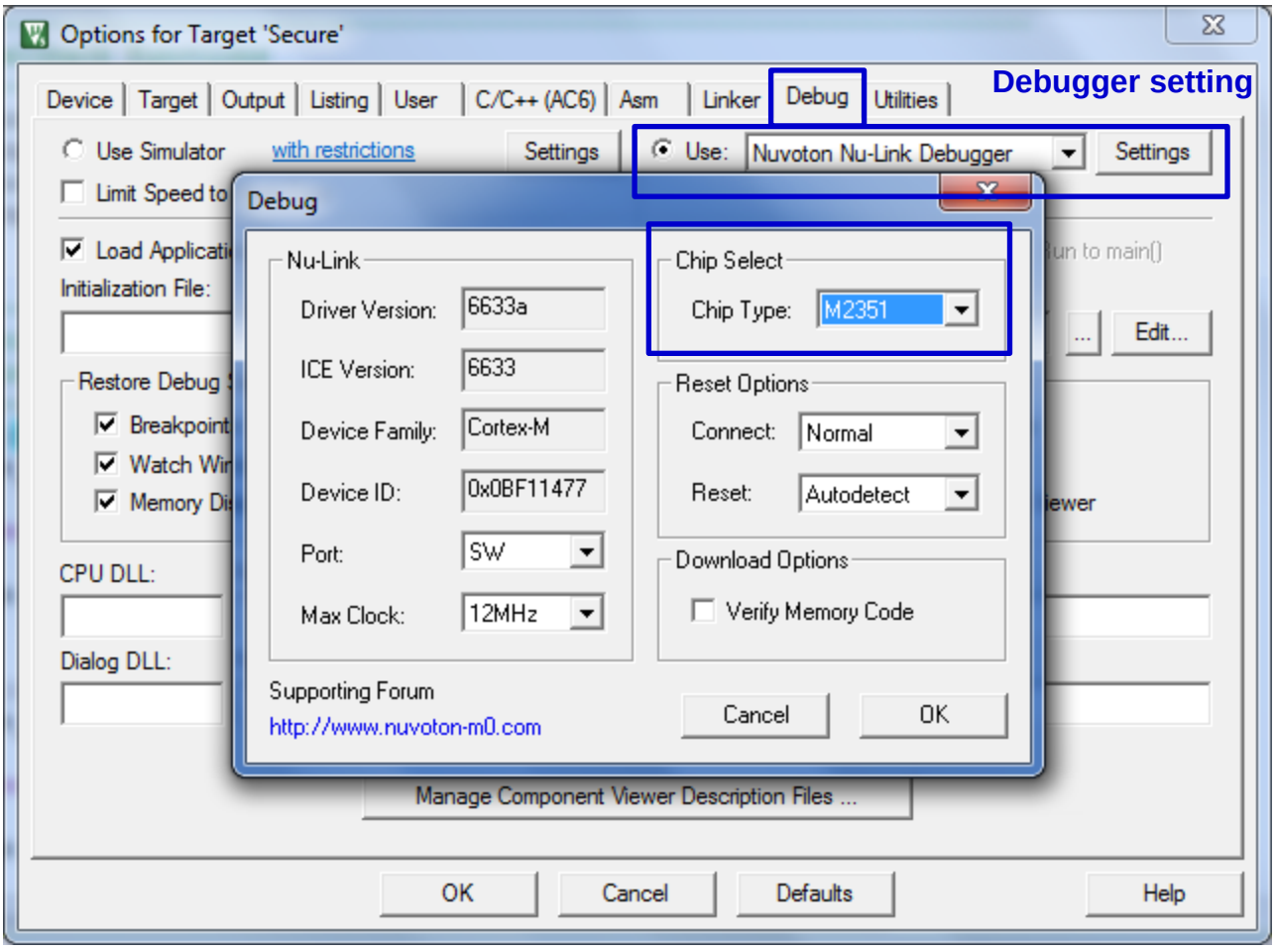

Options for Target 'Secure'
Device
Target
Output
Listing
User
C/C++ (AC6)
Asm
Linker
Debug
Utilities
Use Memory Layout from Target Dialog
Make RW Sections Position Independent
Make RO Sections Position Independent
Don't Search Standard Libraries
Report 'might fail' Conditions as Errors
X/O Base:
R/O Base:
R/W Base
disable Warnings:
Flash, SRAM and Non-secure callable entry function location setting
Scatter File
.\secure.sct
...
Edit...
Secure code creates Non-secure callable function library
Misc controls
--import-cmse-lib-out ..\lib\nsclib_Secure.o
Linker control string
--cpu=8-M.Base *.o
--library_type=microlib --strict --scatter ".\secure.sct"
OK
Cancel
Defaults
Help


secure.sct
1
2 LR_ROM 0x0 0x10000
3 {
4
5 EXE_ROM +0
6 {
7 *.o(RESET, +First)
8 *(+RO)
9 }
10
11 EXE_RAM 0x20000000 0x8000
12 {
13 *(+RW, +ZI)
14 }
15
16 }
Flash and SRAM location setting



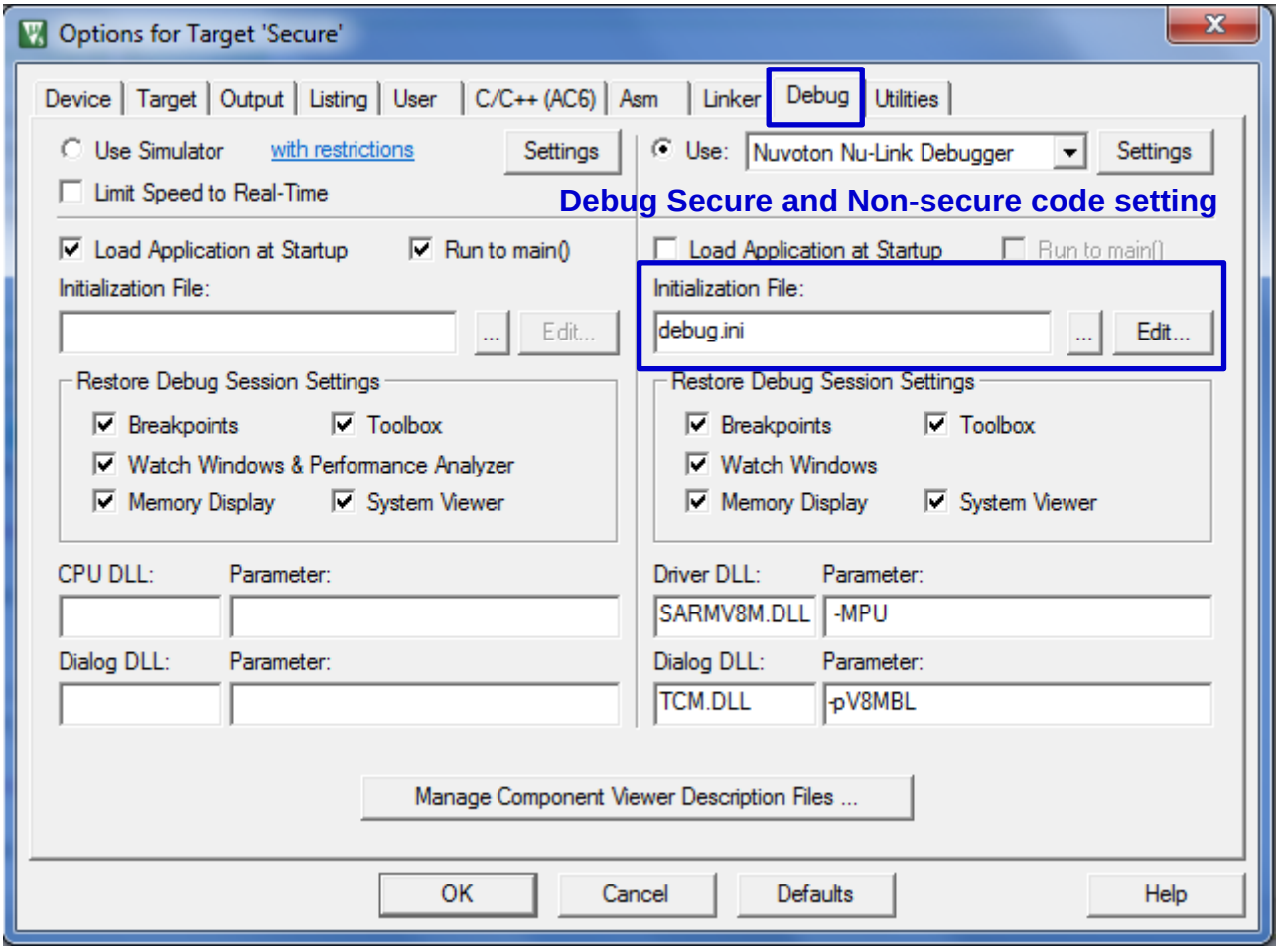
Debugger setting:

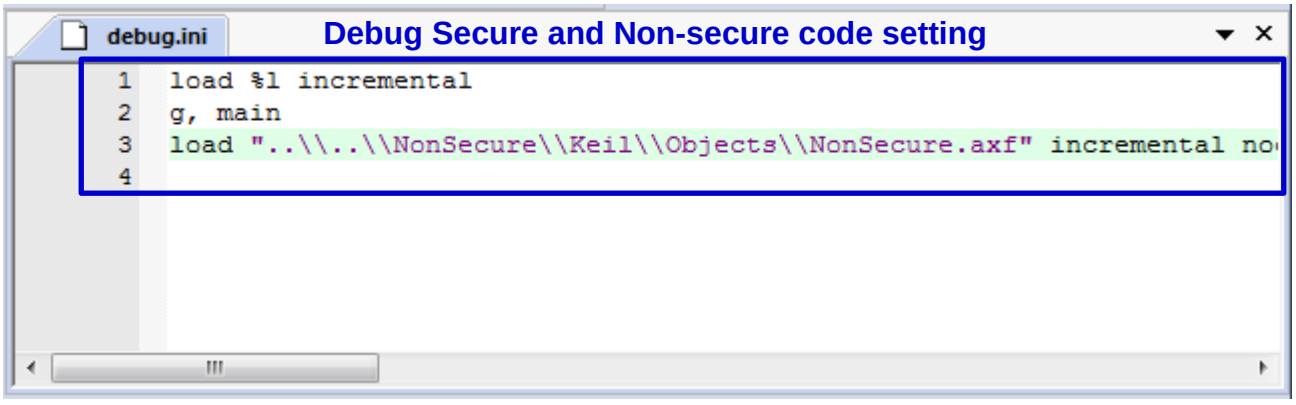
1. In the [Options for Target] window, click [Debug] to open [Debug] page.
2. Use [Nuvoton Nu-Link Debugger] as debugger.
3. Click [Settings] button to open [Debug] window.
4. Select [Chip Type] as [M2351] from the pull-down menu.



Debug Secure and Non-secure code setting:

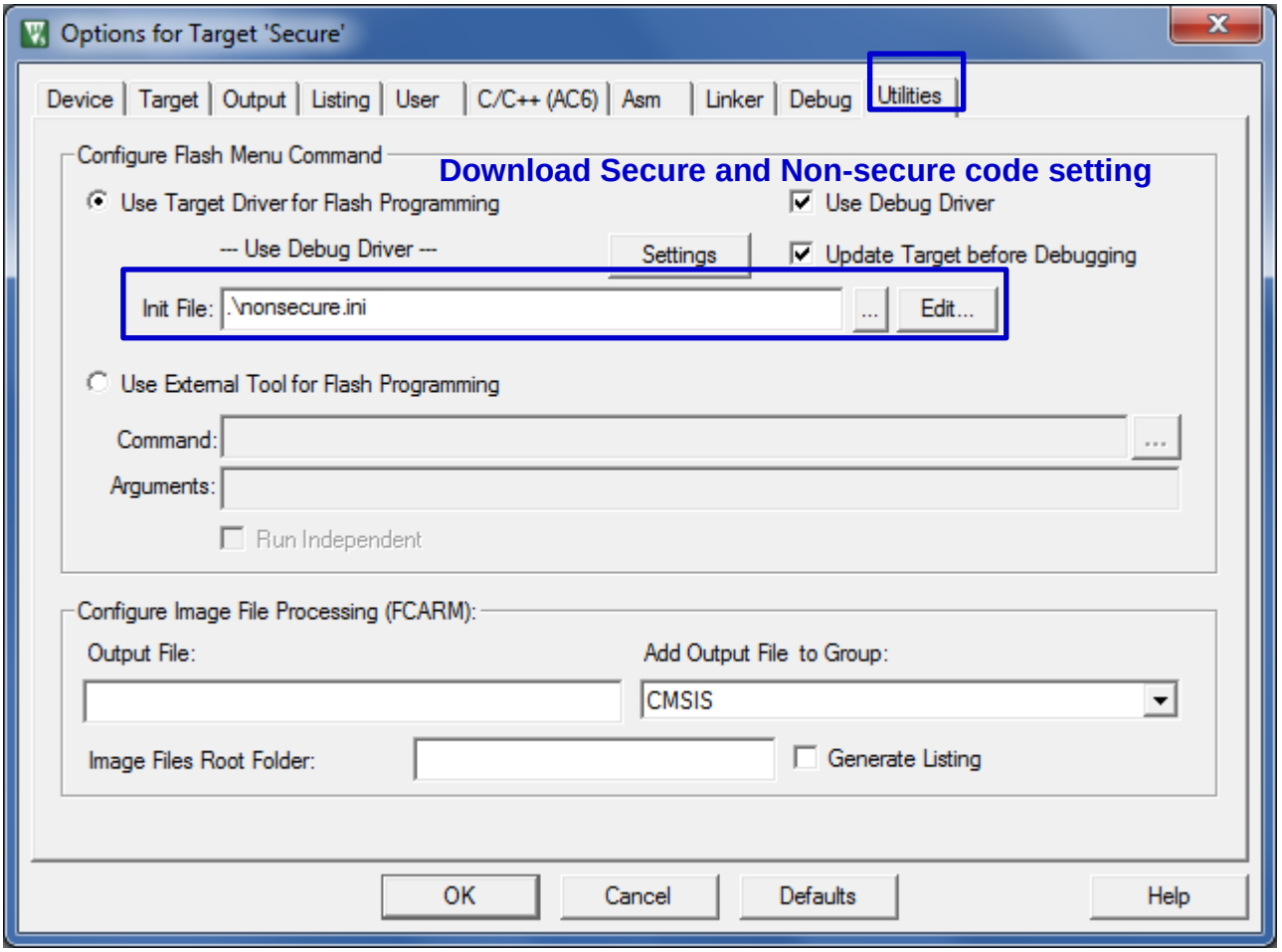
1. In the same [Debug] page, under the [Initialization File] section, click [Edit] button to edit [debug.ini]
2. Add command to load Non-secure image at the start of the debug session, to debug both Secure and Non-secure code.

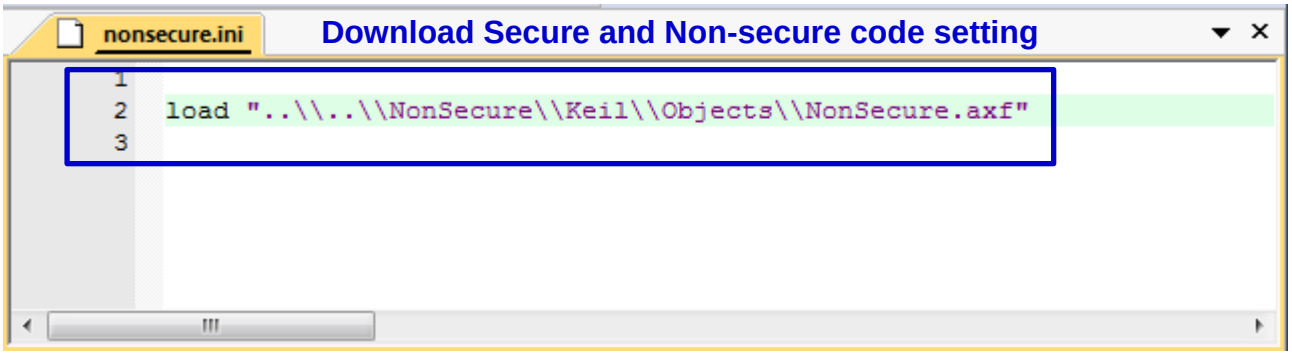




Download Secure and Non-secure code setting:

1. In the [Option for Target] window, click [Utilities] page.
2. In the [Configure Flash Menu Command] section [Init File] option.
3. Click [Edit] button to edit [nonsecure.ini] to download both Secure and Non-secure code.

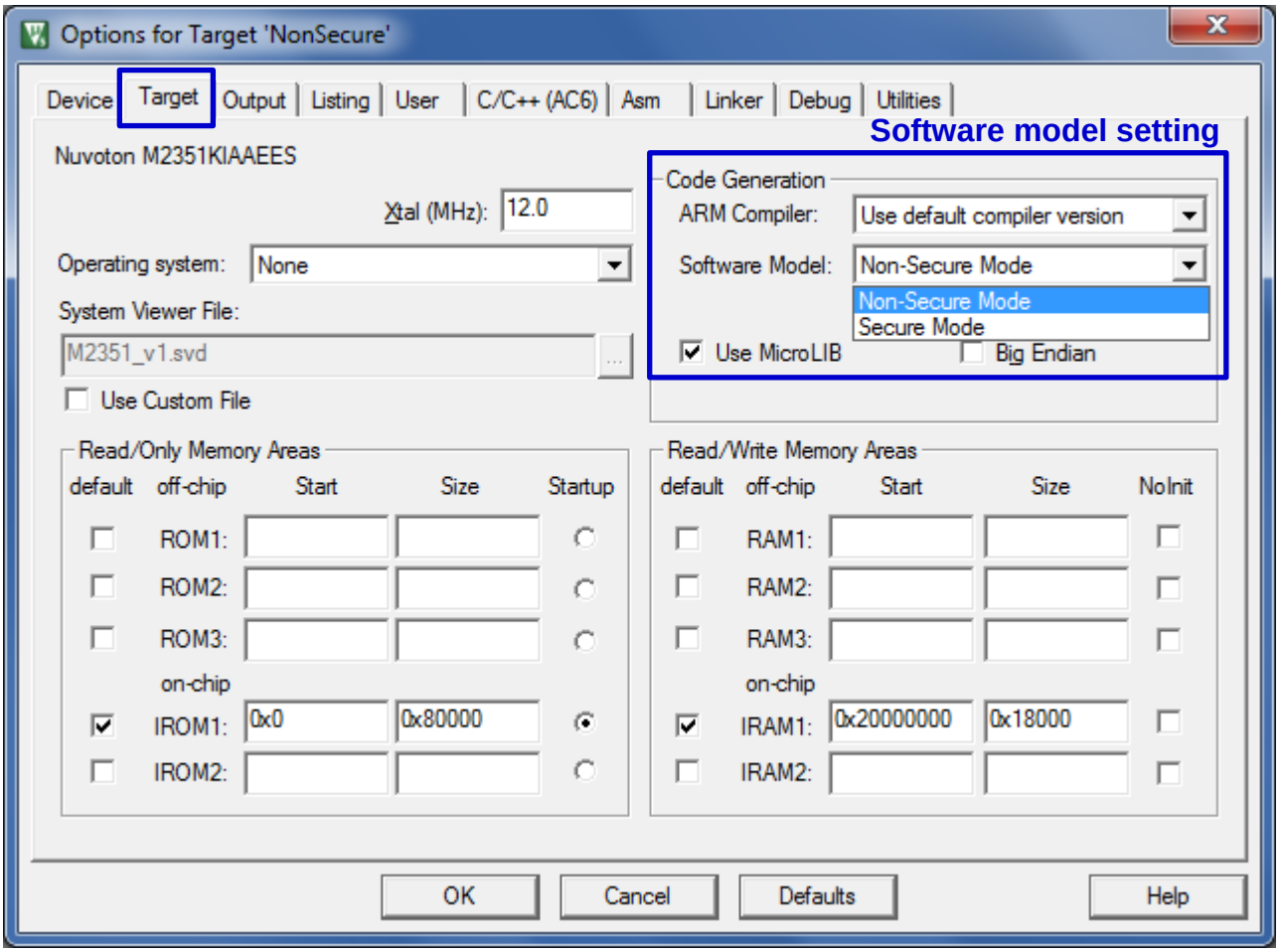




3.2.2 Non-secure Code

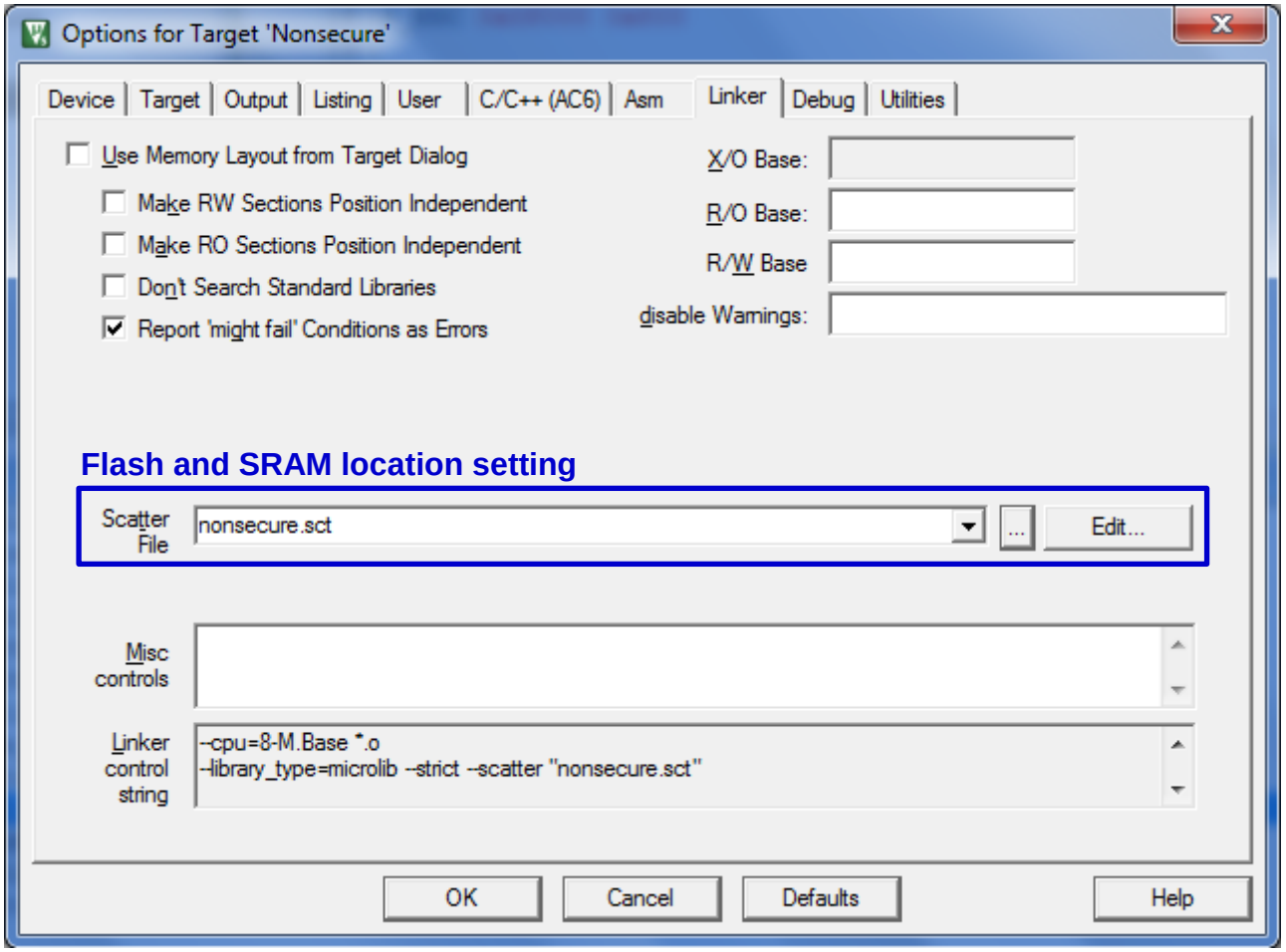
Software model setting:

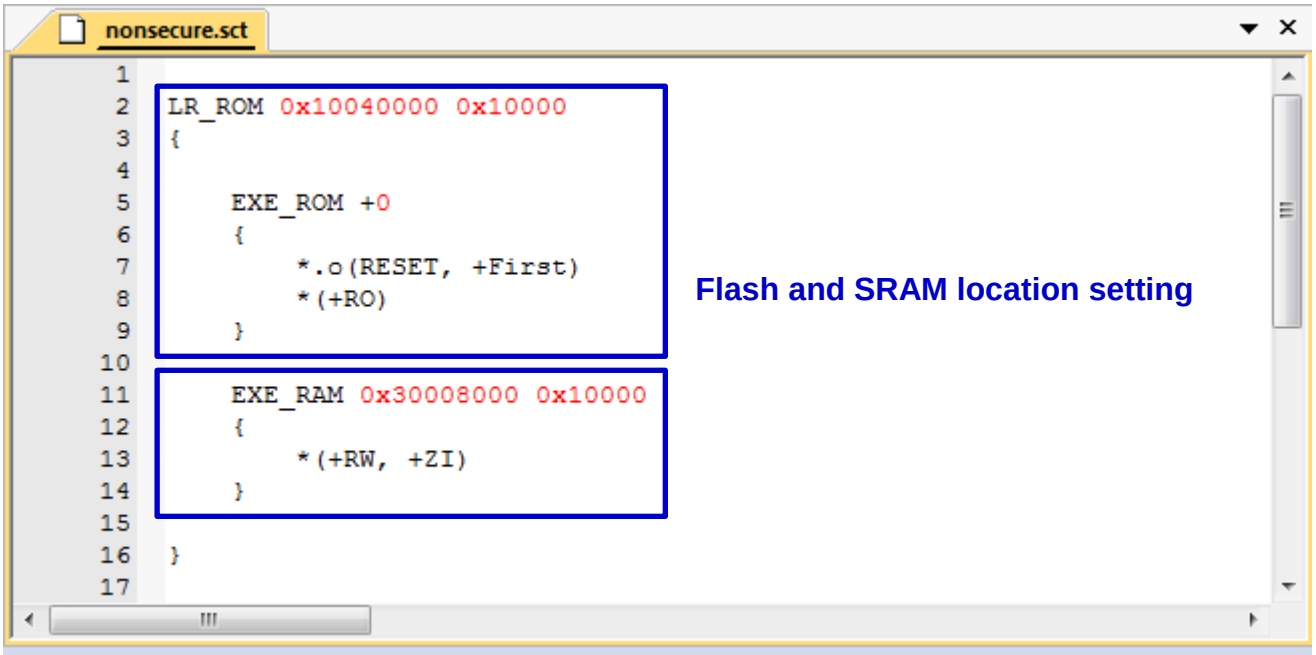
1. Open Non-secure project in the Keil® MDK environment.
2. Click [Project] on the menu bar and click [Options for Target] to open the configuration window.
3. In the [Options for Target] window, click [Target] page.
4. In the [Code Generation] section, select [Software Model] as [Non-secure Mode].



Flash and SRAM location setting:

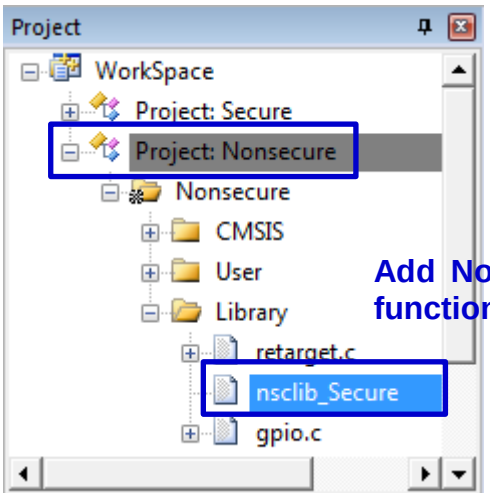
1. In the [Options for Target] window, click [Linker] page.
2. In the [Scatter File] section, click [Edit] button to edit [nonsecure.sct]. Set Non-secure code Flash start address as 0x10040000 and size as 0x10000. Set Non-secure SRAM start address as 0x30008000 and size as 0x10000.


Options for Target 'Nonsecure'
Device
Target
Output
Listing
User
C/C++ (AC6)
Asm
Linker
Debug
Utilities
Use Memory Layout from Target Dialog
Make RW Sections Position Independent
Make RO Sections Position Independent
Don't Search Standard Libraries
Report 'might fail' Conditions as Errors
X/O Base:
R/O Base:
R/W Base
disable Warnings:
Flash and SRAM location setting
Scatter File
nonsecure.sct
...
Edit...
Misc controls
Linker control string
--cpu=8-M.Base *.o
--library_type=microlib --strict --scatter "nonsecure.sct"
OK
Cancel
Defaults
Help


nonsecure.sct
1
2 LR_ROM 0x10040000 0x10000
3 {
4
5 EXE_ROM +0
6 {
7 *.o(RESET, +First)
8 *(+RO)
9 }
10
11 EXE_RAM 0x30008000 0x10000
12 {
13 *(+RW, +ZI)
14 }
15
16 }
17
Flash and SRAM location setting

Add Non-secure callable function library:

Add [nsclib_Secure.o] which is created by Secure project to use Non-secure callable function.



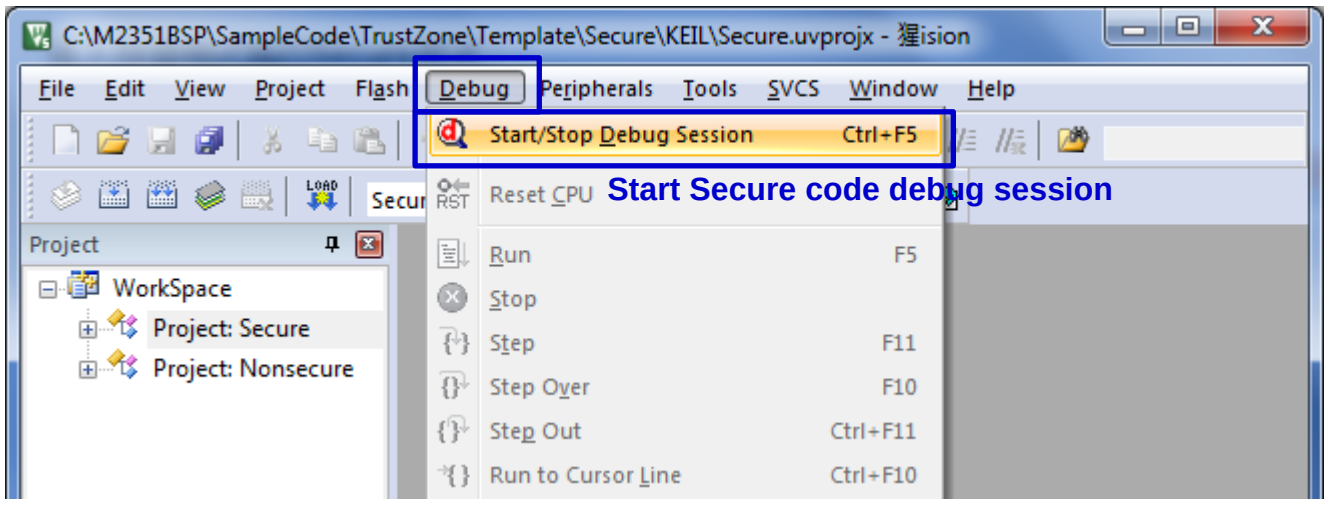
3.3 Secure and Non-secure State Switch

Secure and Non-secure code can run after compiling and downloading Secure and Non-secure project. The state switches between Secure and Non-secure state that can be observed by debug session. This section demonstrates how Secure code calls Non-secure function and Non-secure code calls Secure function by debug session.

3.3.1 Execute from Secure Code to Non-secure Code

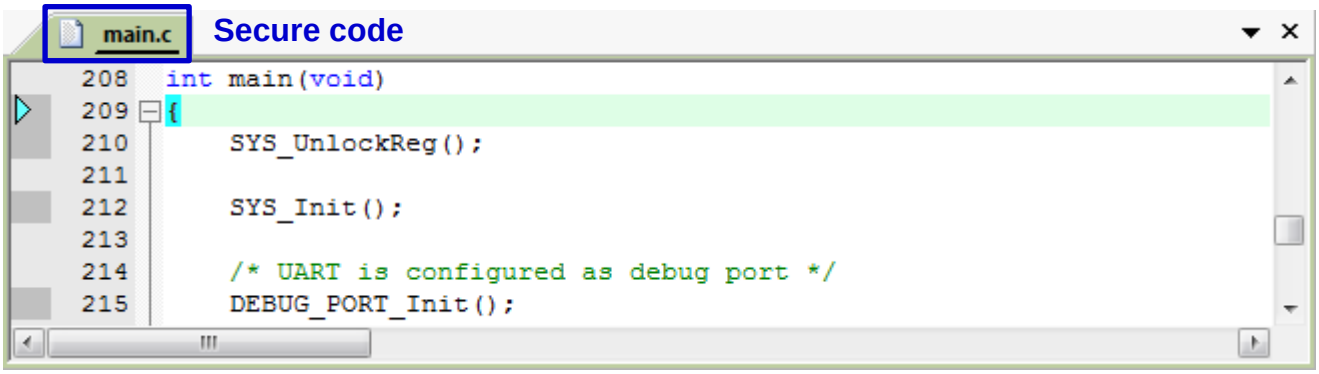
Start Secure code debug session:

Click [Debug] on the menu bar and click [Start Debug Session].



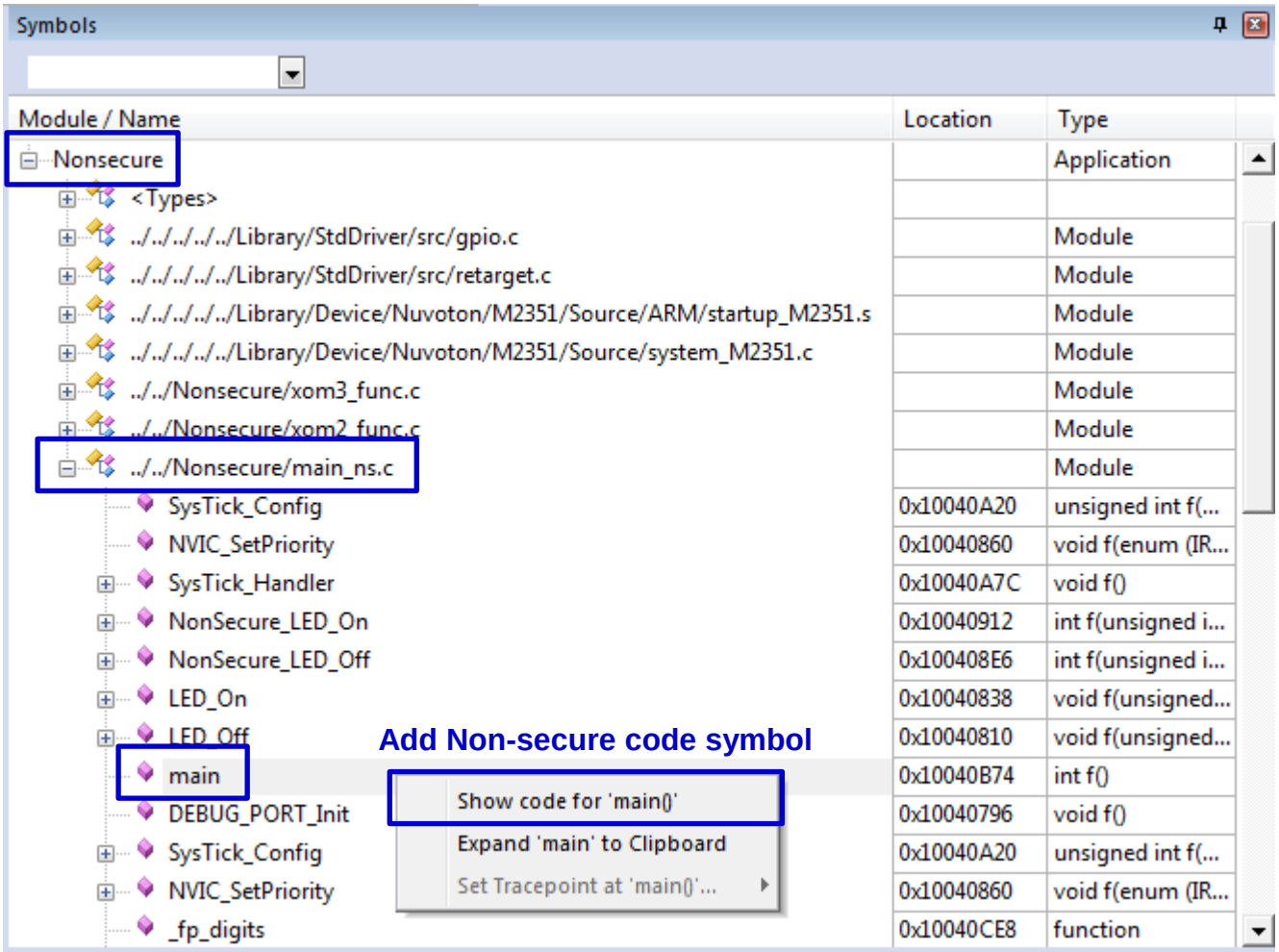
Secure code:

System executes Secure code (main.c) at first.



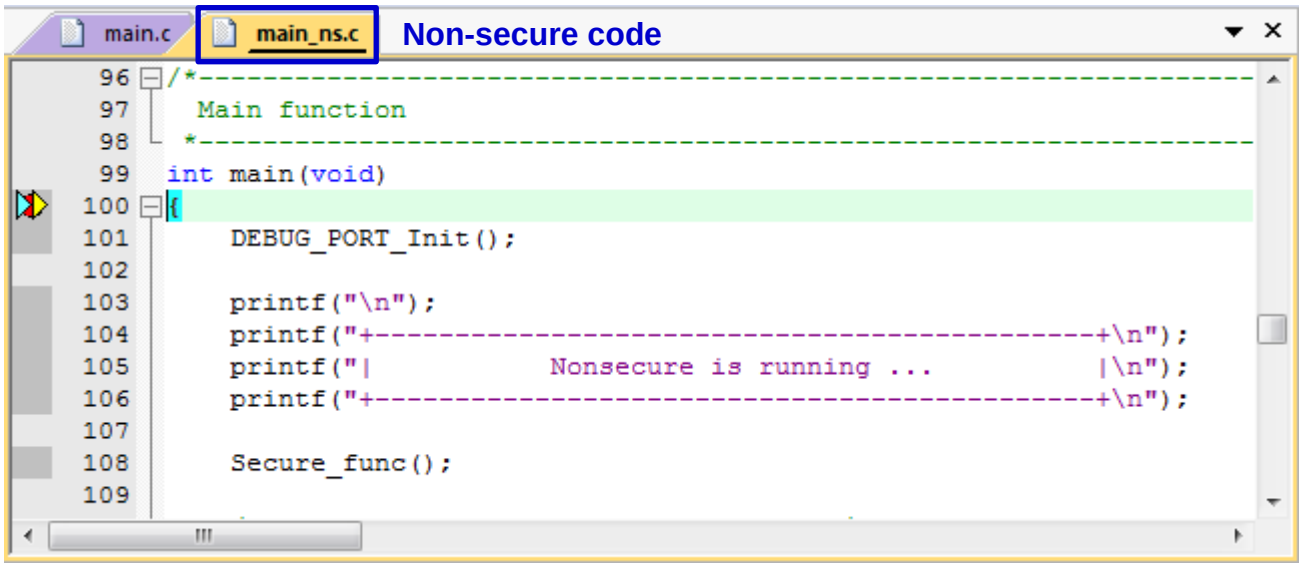
Add Non-secure code symbol:

Click [View] on the menu bar and click [Symbols Window] to open the configuration window. In [Nonsecure] project [main_ns.c] file, find the [main] function. Right click and select [Show code for 'main()'] to add Non-secure code symbol.



Non-secure code:

Then system can execute Non-secure code (main_ns.c).

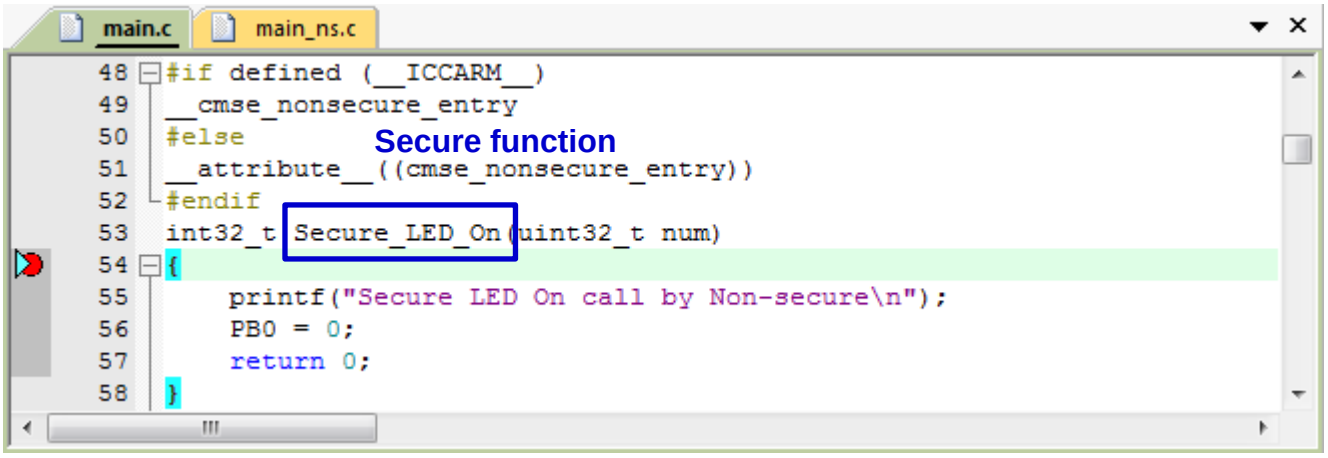


3.3.2 Non-secure Code Calls Secure Function

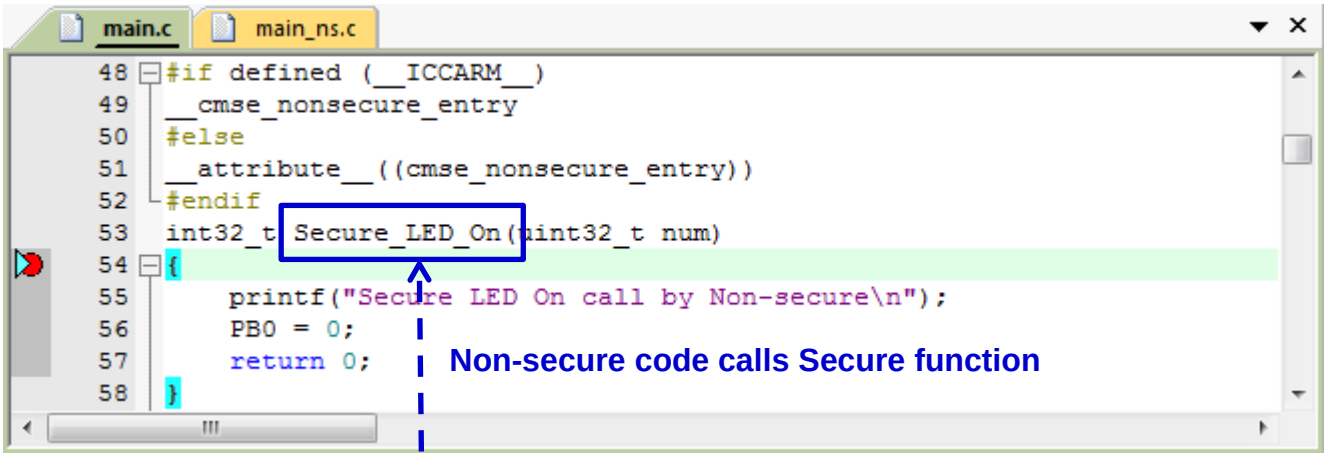
This section describes Non-secure code calls Secure function to show how Non-secure state switches to Secure state. When Non-secure code calls Non-secure callable function, it calls related Non-secure callable entry function in Non-secure callable region. The first instruction of Non-secure callable entry function is SG instruction. SG instruction is the entry point for Non-secure state switching to Secure state. Then Non-secure callable function can be called by Non-secure code.

If Non-secure code (main_ns.c) wants to call Non-secure callable function (Secure_LED_On) it calls VenTTLSecure_LED_On at first.

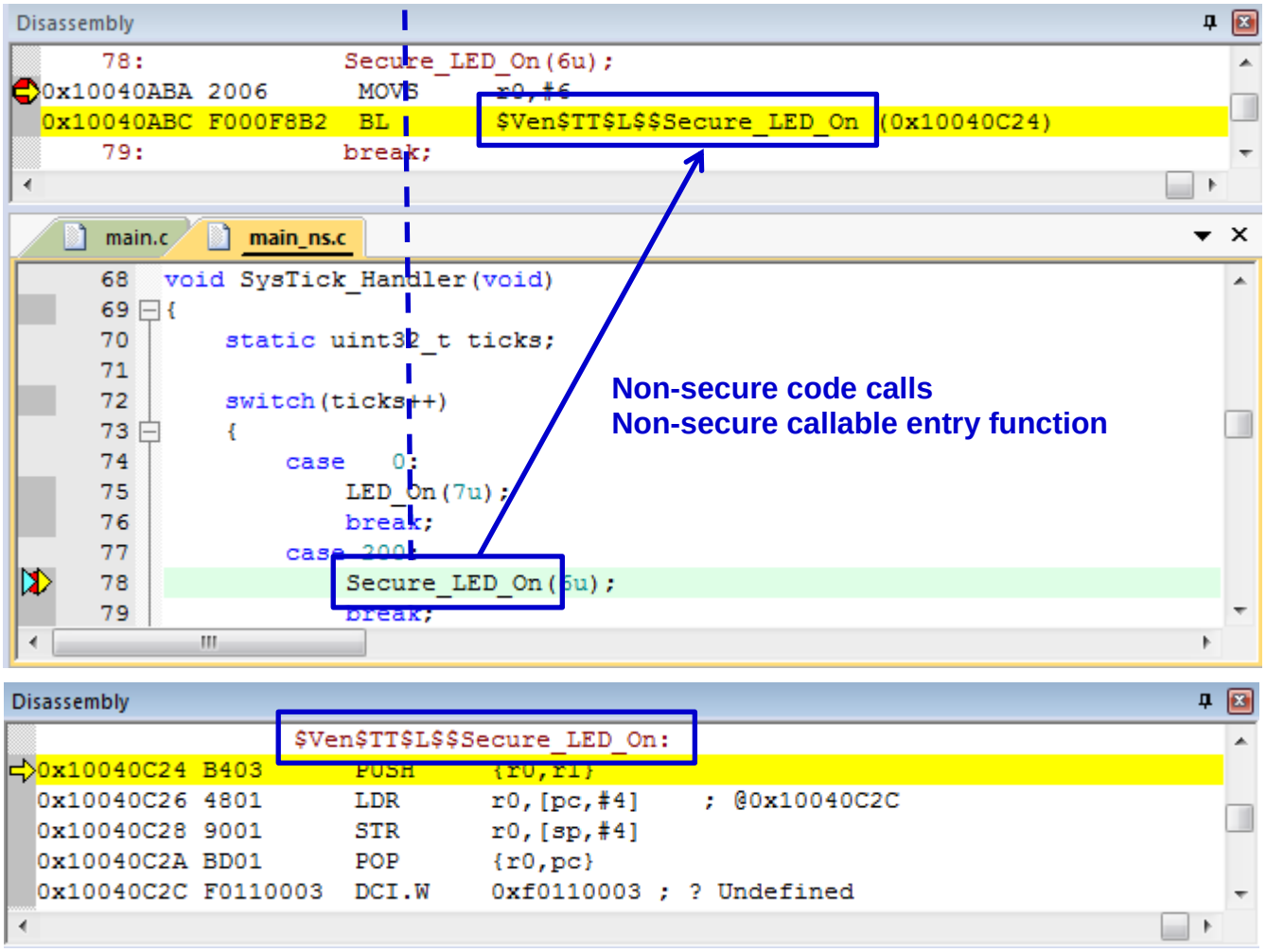
Secure code:



Secure code:

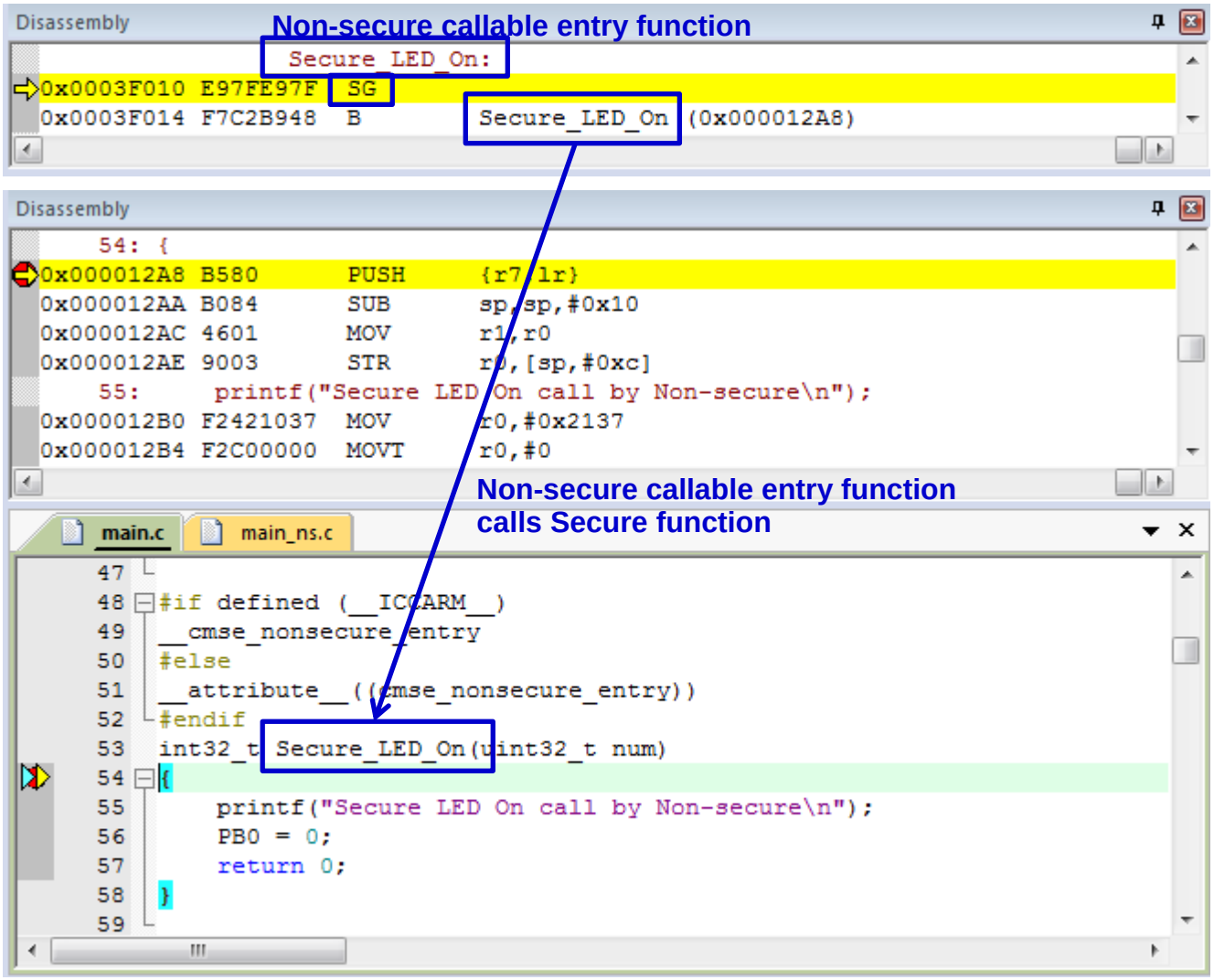


Non-secure code:



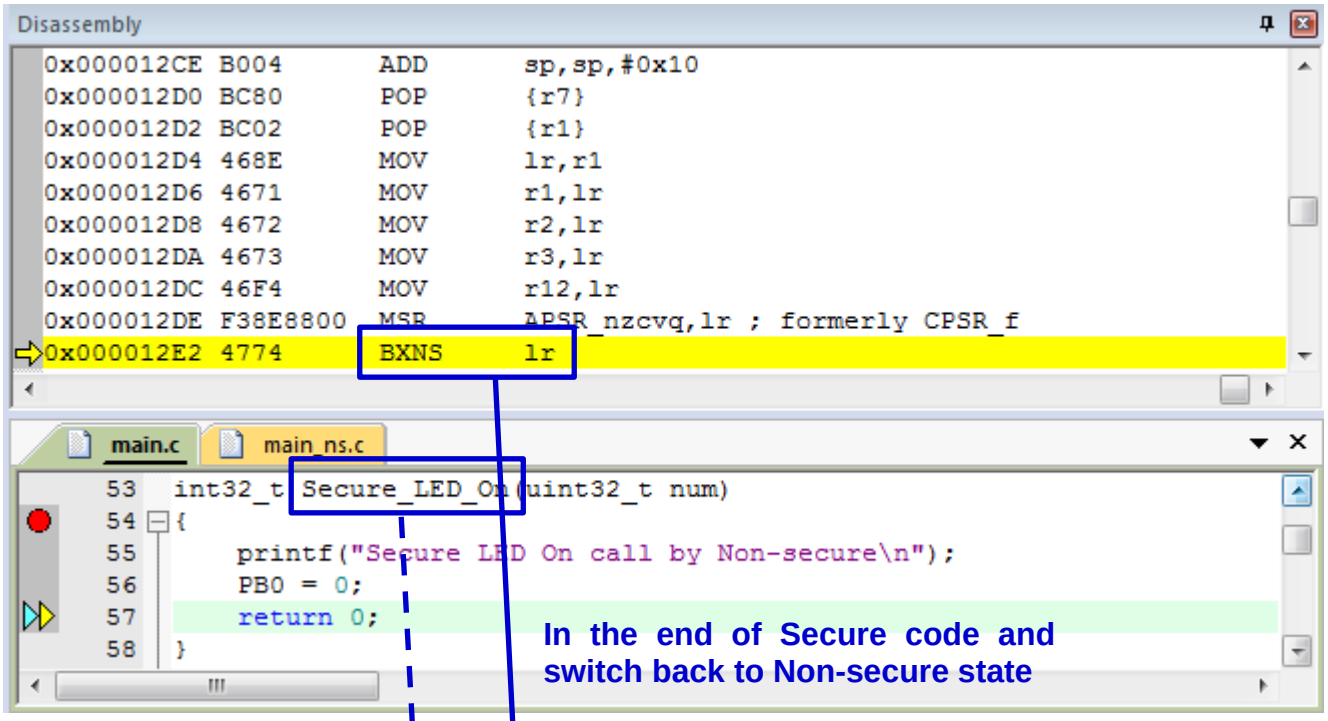
VenTTLSecure_LED_On calls Non-secure callable entry function (Secure_LED_On) in Non-secure callable region. System can switch from Secure state to Non-secure state through the first SG instruction and execute the Non-secure callable function (Secure_LED_On).

Secure code:

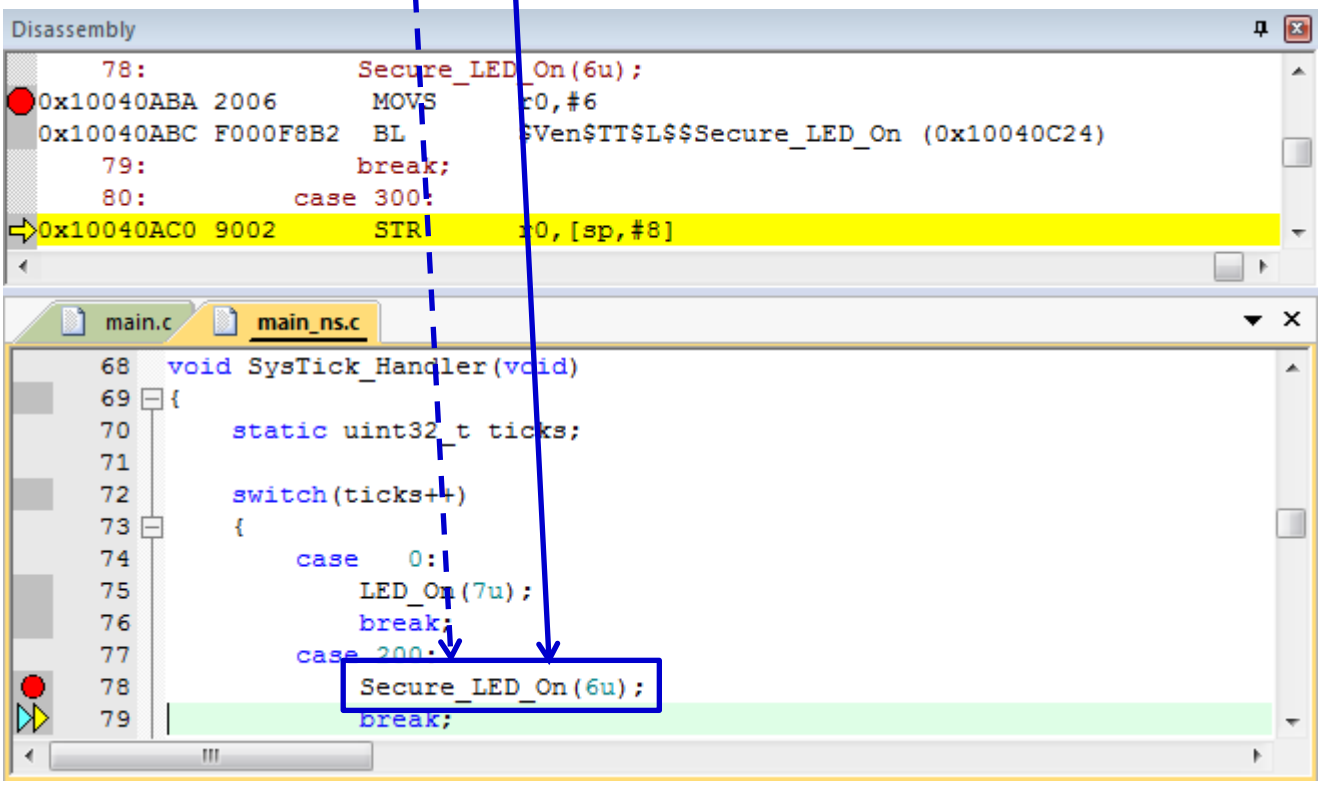


In the end of Secure function (Secure_LED_On), system switches from Secure state to Non-secure state through a BXNS instruction.

Secure code:



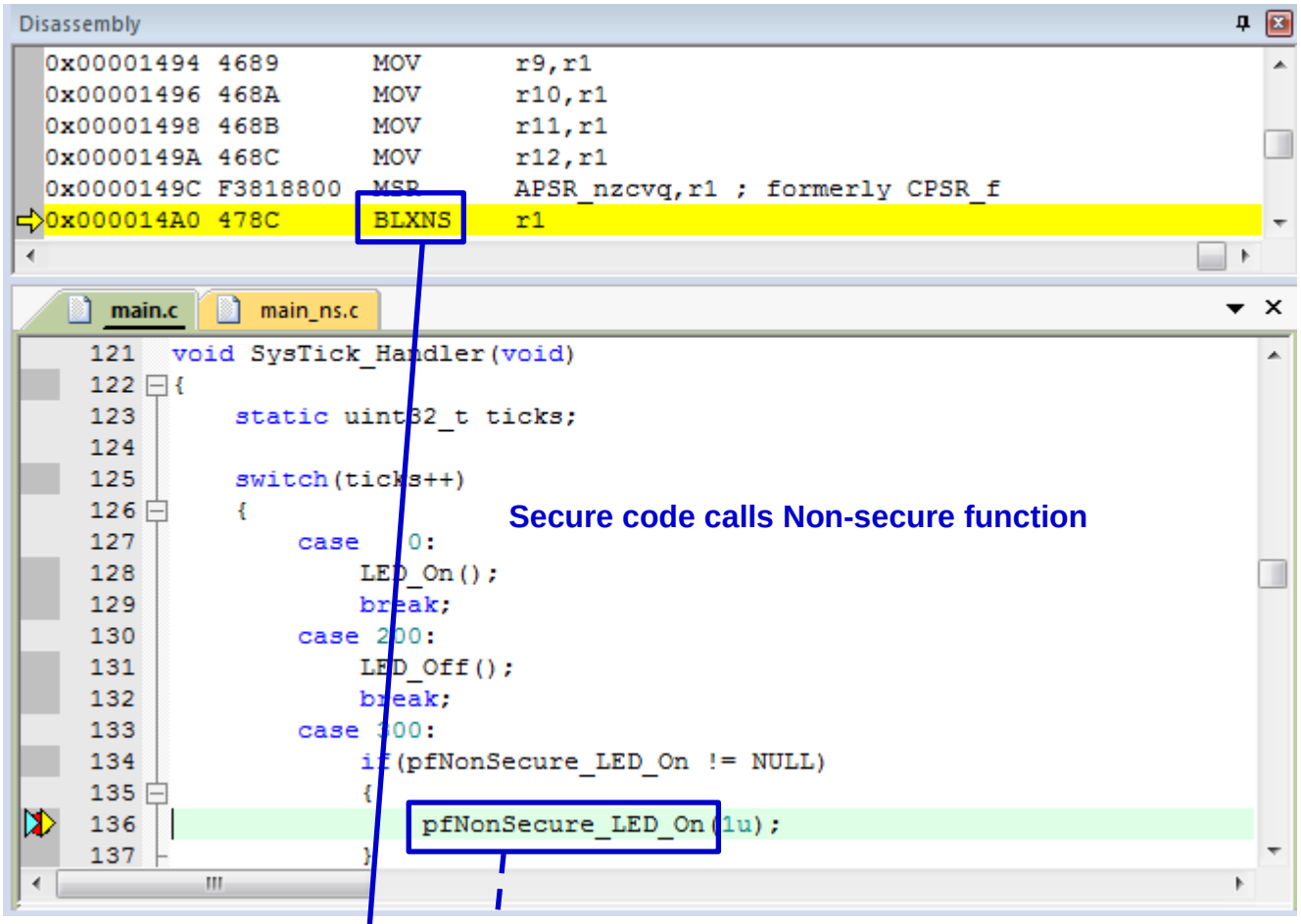
Non-secure code:



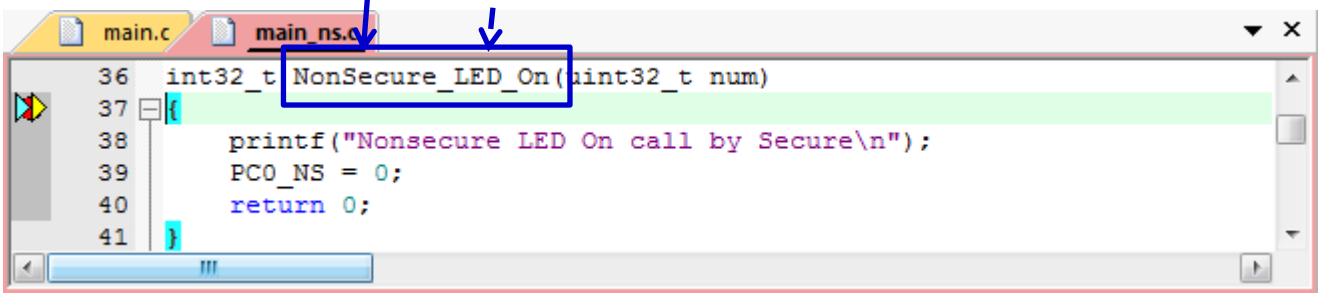
3.3.3 Secure Code Calls Non-secure Function

Secure code (main.c) calls Non-secure function (NonSecure_LED_On) by Non-secure function pointer (pfNonSecure_LED_On). The BLXNS instruction is the entry point system switches state from Secure state to Non-secure state.

Secure code:

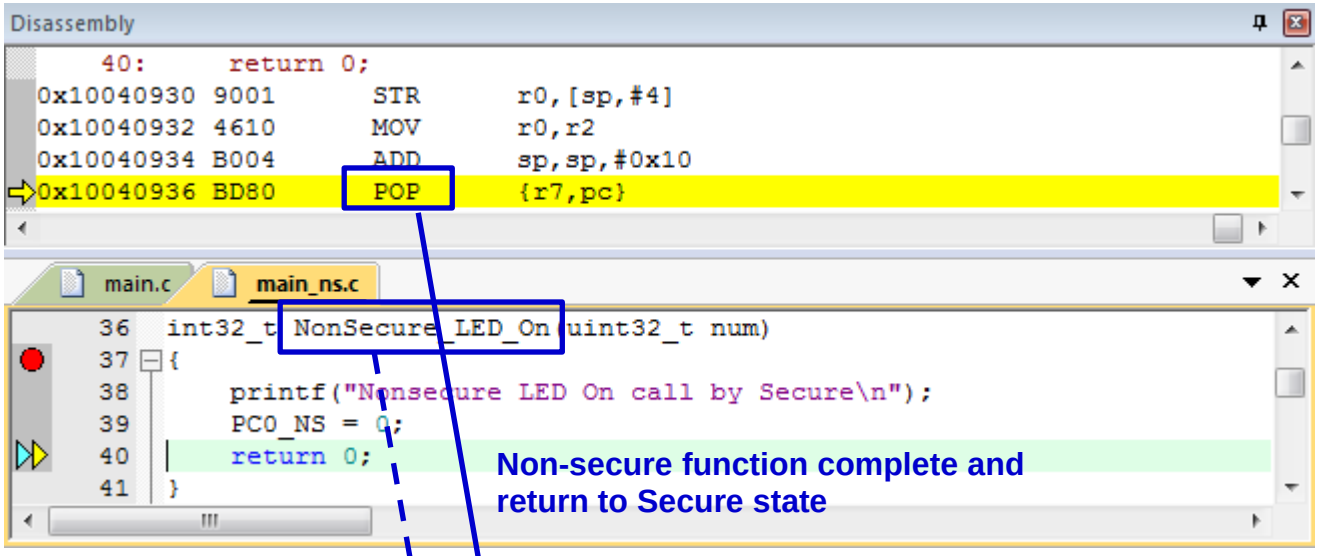


Non-secure code:

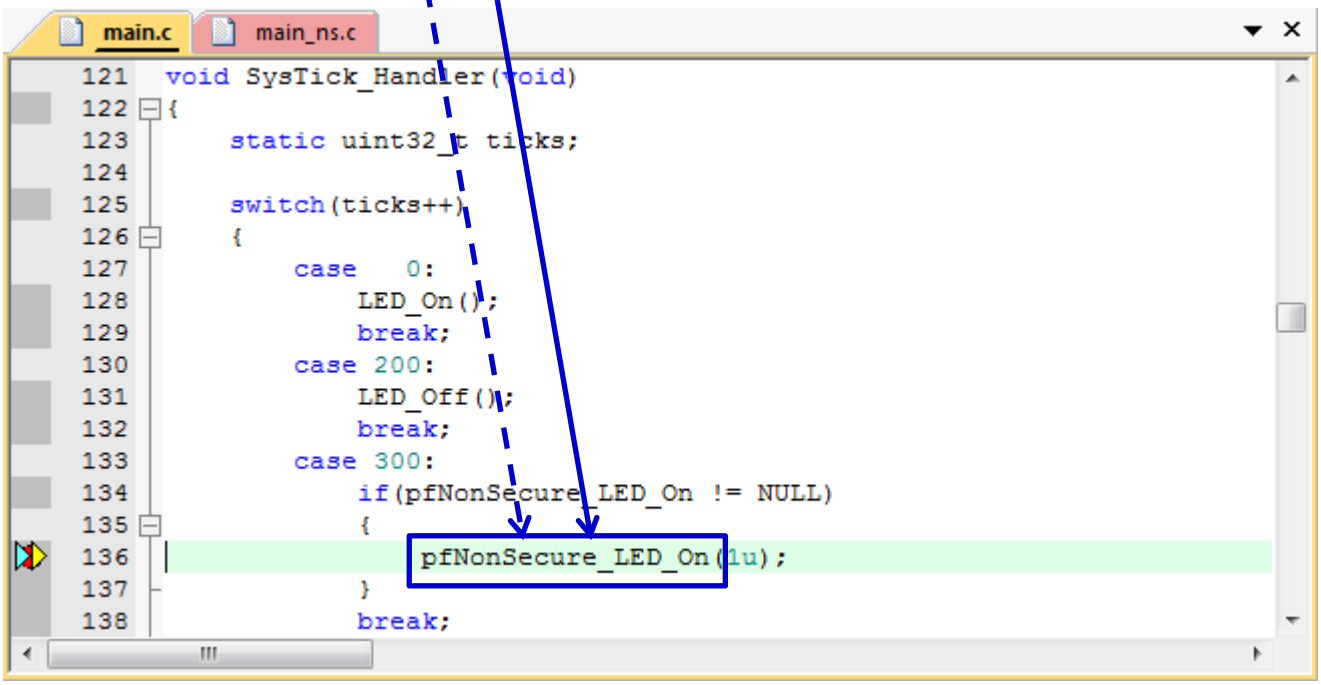


Non-secure function (NonSecure_LED_On) is complete and returns to Secure state.

Non-secure code:



Secure code:



4 Sample Code

The TrustZone® sample code shows security attribution configuration and the state switch between Secure and Non-secure state. Figure 4-1 shows the SAU memory map which is defined by SAU. SAU defines the Non-secure region where data does not need protection and anyone can read or modify it, including Non-secure Flash, SRAM, peripheral and Non-secure callable Flash. Other regions not defined by SAU is Secure and data in the Secure region cannot be read and modified by everyone. SAU defined security memory map and IDAU memory map are compared with each other to determine the security attribution result. In the sample code, SAU defined attribution has the higher secure priority. The security attribution result is based on SAU configuration.

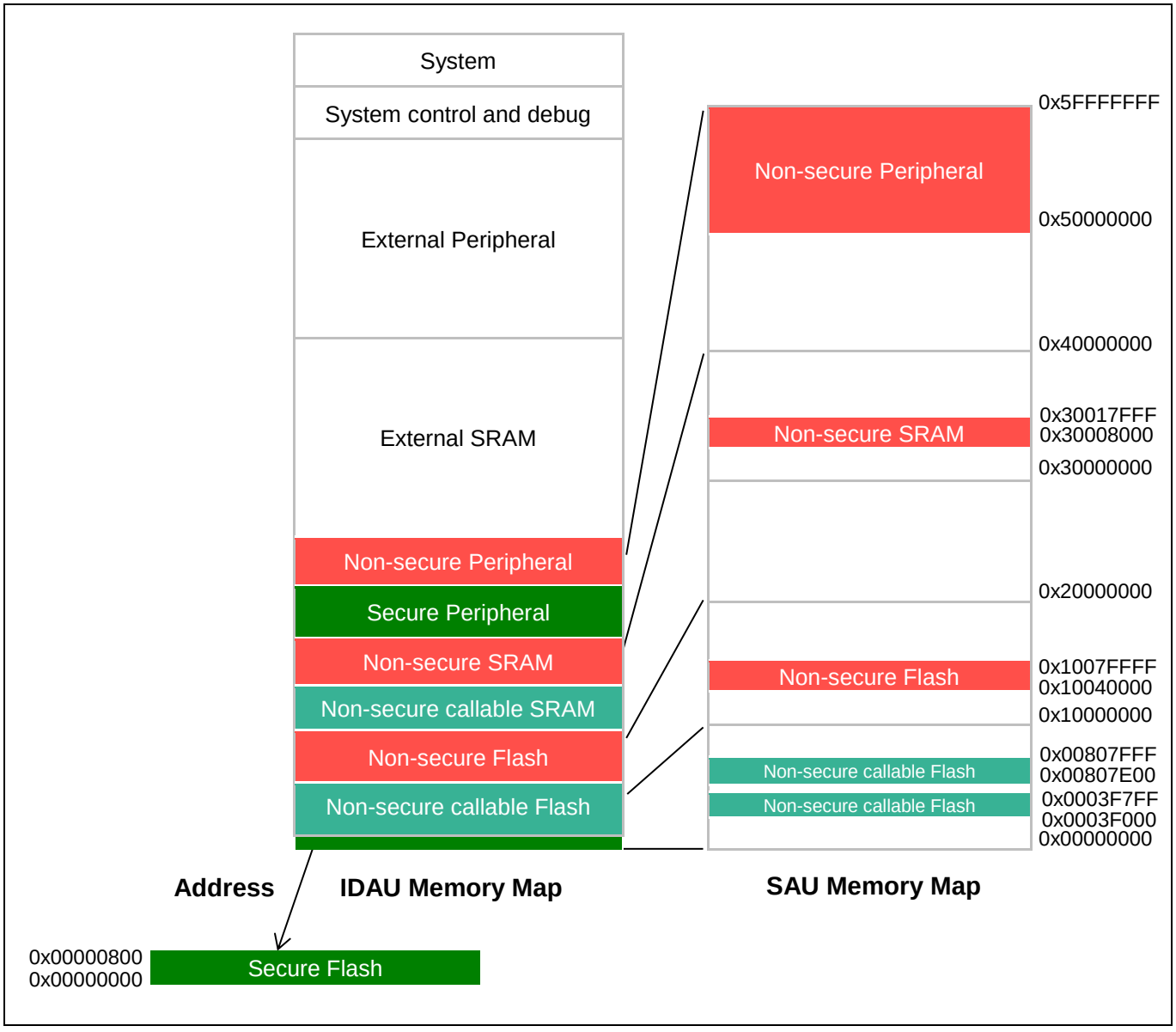


Figure 4-1 TrustZone® Sample Code SAU Memory Map

The TrustZone® sample code also shows Secure and Non-secure state switch. Secure and Non-secure codes configure their Systick independently. System enters Systick interrupt periodically. In Secure code, system toggles PB.1. In the Non-secure callable function, system toggles PB.0. In the Non-secure code, system toggles PC.1. When Secure code calls Non-secure function, system toggles PC.0. State switch between Secure and Non-secure state can be observed from GPIO status (high or low) change. Table 4-1 lists the GPIO and its related code state.

GPIO	Code State
PB.0	Secure code calls Non-secure function.
PB.1	Secure code.
PC.0	Non-secure code calls Secure function.
PC.1	Non-secure code.

Table 4-1 TrustZone® Sample Code GPIO and Related Code State

4.1 Security Attribution Configuration

This section demonstrates how to configure resources security attribution by programing including memory map, Flash, SRAM, peripherals and peripheral interrupts.

4.1.1 Memory Map

Memory map security attribution is set by SAU. For example, plan the address 0x3F000-0x3F7FF for Non-secure callable function and set security attribution as Secure and Non-secure callable.

```
/* Enable SAU */
SAU->CTRL = SAU_CTRL_ENABLE_Msk;

/* Set SAU region 3 */
SAU->RNR = 3;

/* Set SAU region 0 start address */
SAU->RBAR = (0x0003F000 & SAU_RLAR_BADDR_Msk);

/* Set SAU region 0 end address and attribute */
SAU->RLAR = (0x0003F7FF & SAU_RLAR_LADDR_Msk) | SAU_RLAR_NSC_Msk | SAU_RLAR_ENABLE_Msk;
```

4.1.2 Flash

Flash security attribution is set by the register NSCBA (Non-secure base address register, address 0x00200800). NSCBA sets the start address of Non-secure region in Flash and its read/write is through FMC. User can read the register SCU_FNSADDR (Flash Non-secure address register, address 0x4002F028) to get current NSCBA setting. For example, set the NSCBA value as 0x40000.

```
/* Non-secure code start address setup */
void FMC_NSBA_Setup(void)
{
    /* Check if NSBA value with current active NSBA */
    if(SCU->FNSADDR != 0x40000)
    {
        /* Unlock Protected Register */
        SYS_UnlockReg();

        /* Enable ISP and configuration update */
        FMC_DISABLE_ISP();
        FMC_ENABLE_CFG_UPDATE();

        /* Setting NSBA when it is empty */
        if( FMC_Read(0x200800) == 0xffffffful )
        {
            FMC_Write(0x200800, 0x40000);

            /* Force Chip Reset to valid new setting */
            SYS->IPRST0 = SYS_IPRST0_CHIPRST_Msk;
        }

        while(1);
    }

}
```

4.1.3 SRAM

SRAM security attribution is set by the register SCU_SRAMNSSET (SRAM secure attribution set register, address 0x4002F024). For example, set the address 0x0-0xBFFF as Secure and set the address 0xC000-0x17FFF as Non-secure.

```
/* SRAM Secure Attribution Configuration */
SCU->SRAMNSSET = 0x00000FC0;
```

4.1.4 Peripheral

Peripheral security attributions are set by the register SCU_PNSSET0-SCU_PNSSET6 (Peripheral secure attribution set register 0-6, address 0x4002F000-0x4002F018) and SCU_IONSSET (IO secure attribution set register, address 0x4002F01C). For example, set the UART1 security attribution as Non-secure.

```
/* Set UART0 Peripheral Secure Attribution */
SCU->PNSSET[3] |= SCU_PNSSET3_UART1_Msk;
```

Or set UART1 security attribution as Non-secure by SCU_SET_PNSSET().

```
/* Set UART0 Peripheral Secure Attribution */
SCU_SET_PNSSET(UART1_Attr);
```

4.1.5 Peripheral Interrupt

Peripheral interrupt security attributions are set by the register NVIC_ITNS0-NVIC_ITNS3 (Interrupt Target Non-secure Register 0-3, address 0xE000_E380-0xE000_E38C). For example, set the UART1 interrupt security attribution as Non-secure.

```
/* Set UART1 Interrupt Vector Secure Attribution */
NVIC->ITNS[1] |= BIT5;
```

Or set the UART1 interrupt security attribution as Non-secure by NVIC_SetTargetState().

```
/* Set UART1 Interrupt Vector Secure Attribution Configuration */
NVIC_SetTargetState(UART1_IRQn);
```

4.2 Secure and Non-secure State Switch

The system starts up in Secure code by default. This section demonstrates how Secure code is executed to Non-secure code. The Non-secure code can call Secure function and the Secure code can call Non-secure function, too.

4.2.1 Execute from Secure Code to Non-secure Code

Secure code:

Before executing Non-secure code, user needs to set Non-secure vector table address and Non-secure Main Stack Pointer, use a Non-secure function pointer and assign the value to Non-secure code Reset_Handler function. Use cmse_nsfptr_create intrinsic to clear LSB of Non-secure function address, then use Non-secure function pointer to call Non-secure function directly.

```
/* typedef for NonSecure callback functions */
typedef __attribute__((cmse_nonsecure_call)) int32_t (*NonSecure_funcptr)(uint32_t);

void Nonsecure_Init(void)
{
    /* Non-secure function pointer */
    NonSecure_funcptr fp;

    /* SCB_NS.VTOR points to the Non-secure vector table base address. */
    SCB_NS->VTOR = 0x10040000;

    /* 1st Entry in the vector table is the Non-secure Main Stack Pointer. */
    __TZ_set_MSP_NS(*((uint32_t *)SCB_NS->VTOR));

    /* 2nd entry contains the address of the Reset_Handler function */
    fp = ((NonSecure_funcptr)(*(((uint32_t *)SCB_NS->VTOR) + 1)));

    /* Clear the LSB of the function address to indicate the function-call
       will cause a state switch from Secure to Non-secure */
    fp = cmse_nsfptr_create(fp);

    /* Non-secure function call */
    fp(0);
}
```

4.2.2 Non-secure Code Calls Secure Function

Secure code:

Secure code adds “cmse_nonsecure_entry” attribute for Non-secure callable function (Secure_LED_On).

```
/* Secure function and Non-secure code callable */
attribute__((cmse_nonsecure_entry))
int32_t Secure_LED_On(uint32_t num)
{
    printf("Secure LED ON call by secure\n");
    PB0 = 0;
    return num * 3;
}
```

Non-secure code:

Then Non-secure code can call Non-secure callable function (Secure_LED_On) which is provided by Secure code.

```
/* NonSecure Callable Functions from Secure Region */
extern int32_t Secure_LED_On(uint32_t num);

/* Non-secure code call Secure function */
void SysTick_Handler(void)
{
    Secure_LED_On(6u);
}
```

4.2.3 Secure Code Calls Non-secure Function

Non-secure code:

Secure code can call Non-secure function directly. The following is a Non-secure function.

```
/* NonSecure functions used for callback */
int32_t NonSecure_LED_On(uint32_t num)
{
    printf("Nonsecure LED On call by Secure\n");
    PC0_NS = 0;
    return 0;
}
```

Call Non-secure callable function and return Non-secure function address to Secure code when executing Non-secure code.

```
/* NonSecure Callable Functions from Secure Region */
extern int32_t Secure_LED_On_callback(void *callback);

void main(void)
{
    /* register NonSecure callback in Secure application */
    Secure_LED_On_callback(&NonSecure_LED_On);
}
```

Secure code:

Secure code stores the Non-secure function address in Non-secure function pointer and clear the LSB of Non-secure function address by cmse_nsfptr_create intrinsic, then Secure code can call Non-secure function directly by Non-secure function pointer.

```
/* typedef for NonSecure callback functions */
typedef __attribute__((cmse_nonsecure_call)) int32_t (*NonSecure_funcptr)(uint32_t);

/* NonSecure callback function */
NonSecure_funcptr pfNonSecure_LED_On  = (NonSecure_funcptr)NULL;

/* Secure function for NonSecure callbacks exported to NonSecure application */
__attribute__((cmse_nonsecure_entry))
int32_t Secure_LED_On_callback(NonSecure_funcptr *callback)
{
    pfNonSecure_LED_On = (NonSecure_funcptr)cmse_nsfptr_create(callback);
    return 0;
}

/* Secure code call Non-secure function */
void SysTick_Handler(void)
{
    pfNonSecure_LED_On(1u);
}
```

4.2.4 Non-secure Security Attribution Check

Secure code provides Non-secure callable function for Non-secure code to call Secure function. It also provides parameter and return value. Secure function can check the memory security by cmse_check_adress_range intrinsic before reading or modifying any data to avoid the important data is stolen or broken.

Non-secure code:

If “ticks” variable value is greater than 600 then call Non-secure callable function to clear “ticks” variable value.

```
/* NonSecure Callable Functions from Secure Region */
extern void ResetTick(uint32_t* buf)
;
/* Non-secure code call Secure function */
void SysTick_Handler(void)
{
```

```
    static uint32_t ticks;

    if(tick>600)
        ResetTick(&ticks);
    else
        ticks++;
}
```

Secure code:

This Non-secure callable function is used to clear the specified variable value to 0. Check if the variable value is located in the Non-secure region before it is modified. Clear the variable value to 0 if the variable address is in the Non-secure region and add 1 to the counter.

```
/* Secure function and Non-secure code callable */
uint32_t g_u32Counter = 0;

__attribute__((cmse_nonsecure_entry))
void ResetTick(uint32_t* buf)
{
    /* Check buffer space from Non-secure */
    buf = cmse_check_address_range(buf, 1, CMSE_NONSECURE);
    if(buf==NULL) return;

    /* Set buffer value to 0 */
    *buf = 0;

    /* Record Non-secure Systick reset counter value */
    g_u32Counter++;
}
```

5 Conclusion

In the IoT (Internet of Things) application, devices not only can communication with each other through the internet but can be attacked through the Internet. The security is an important topic to protect device and information. The Arm® TrustZone® technology partitions hardware into Secure and Non-secure world. The device itself is reliable and Secure code is executed in Secure world. The information from Internet is unreliable such that Non-secure code is executed in Non-secure word.

Through IDAU which defines fixed memory map security attribution with the user configurable SAU, all of microcontroller resources can be configured to Secure and Non-secure including memory map, Flash, SRAM, peripherals and peripheral interrupts. After planning the security attribution of these resources, the Non-secure world can only access Non-secure memories and resources, while the Secure world can access all memories and resources, including secure and Non-secure.

The security attribution can be set by programming or a configuration wizard interface in the Keil® MDK development environment. The code can switch between Secure and Non-secure state. An authentication method can be added to Secure code to certify if the Internet information is trusted so as to provide authority for Non-secure code to access Secure resources. With TrustZone® hardware architecture and software authentication, the IoT application can be implemented safely and flexibly.

Revision History

Date	Revision	Description
2018.08.31	1.00	1. Initially issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.