GigaDevice Semiconductor Inc.

GD32W51x TrustZone Development Guide

Application Note

AN103

Table of Contents

List of Figures

List of Tables

1. Introduction

This article describes developing the TrustZone program on the GD32W51x series. The TrustZone security attribute is the main ARMv8-M extension, which provides hardware security separation and protection. The sample code is also divided into two parts, security and non-security. The development process and key points for Keil and IAR are described.

In this article, the GD32W515P-EVAL development board was used, the chip model was GD32W515PIQ6, and the Cortex-M33 kernel with TrustZone was used, with max frequency up to 180MHz. The TrustZone security is activated by the TZEN option bit in the EFUSE_TZCTL register or the TZEN option bit in the option byte. The sample code is provided in 'GD32W515P_EVAL_Demo_Suites / 23_Trustzone' and is provided by the software package in **GD32MCU.COM**.

Applicable product: GD32W51x series

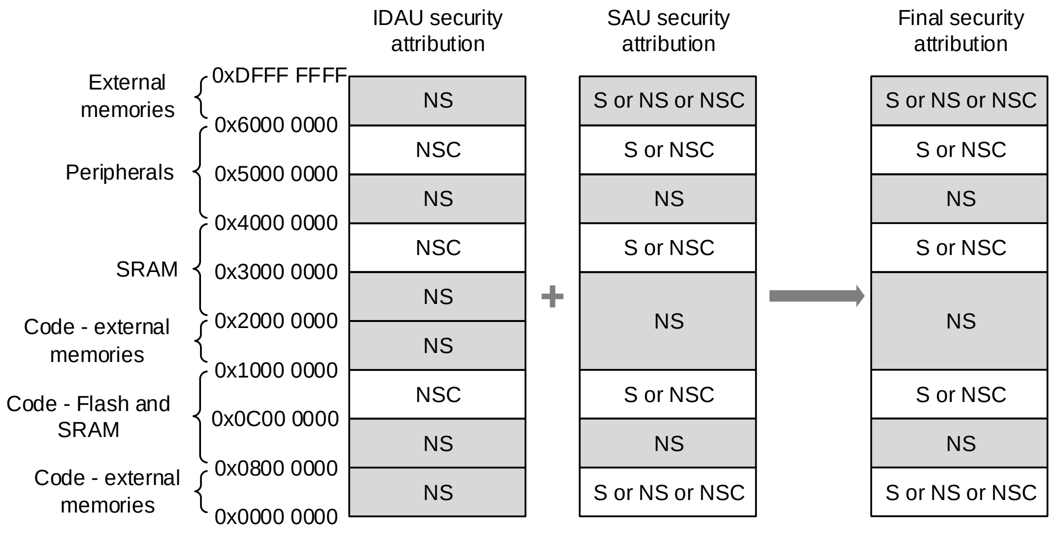
2. TrustZone introduction

The M33 kernel with TrustZone has both secure and non-secure states, which use a separate set of kernel registers and Systick. Memory is classified into secure (S), non-secure callable (NSC), and non-secure (NS) domain. A secure domain can access a non-secure domain, and a non-secure can only access a non-secure callable(in secure code), providing hardware protection for access. When TrustZone is enabled, SAU (security attribution unit) and IDAU (implementation defined attribution unit) are used together to set the security attribute of the memory address. The final security level is the attribute of higher security level defined in SAU and IDAU (S>NSC>NS). See ***Figure 2-1. Example of memory map security attribution vs SAU configuration regions***.

IDAU provides a hardware partition of non-secure (NS) and non-secure Callable (NSC) security attributes, which cannot be changed by software, and memory mapped partitions refer to Section 1.4 Memory Mapping in the GD32W51x_User manual.

SAU can be software-configured with up to 8 security attribute zones, which can be made secure (S), non-secure (NS), or non-secure callable (NSC).

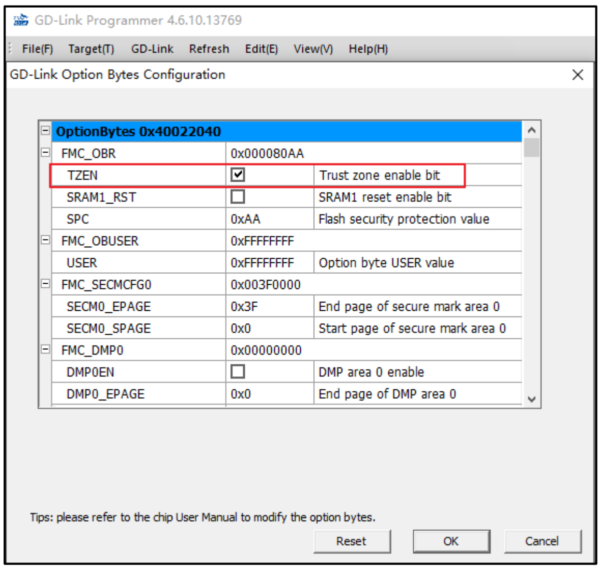
Figure 2-1. Example of memory map security attribution vs SAU configuration regions



GD32W51x uses TrustZone protection controller union (TZPCU) to manage the security attributes of peripherals, SRAM, and flash memory. TZPCU provides unauthorized access control, working with the kernel to achieve full features of TrustZone.

Ensure that the TZEN bit in the EFUSE_TZCTL register or the TZEN bit in the option byte is enabled before using TrustZone or TZPCU. Using GD-Link Programmer to enable TZEN bit in option bytes, as shown in ***Figure 2-2. Enable TrustZone using the GD-Link Programmer***. The GD-Link Programmer can be obtained from **GD32MCU.COM**.

Figure 2-2. Enable TrustZone using the GD-Link Programmer



GD-Link Programmer 4.6.10.13769
File(F) Target(T) GD-Link Refresh Edit(E) View(V) Help(H)
GD-Link Option Bytes Configuration

OptionBytes 0x40022040		
FMC_OBR	0x000080AA	
TZEN	☑	Trust zone enable bit
SRAM1_RST	☐	SRAM1 reset enable bit
SPC	0xAA	Flash security protection value
FMC_OBUSER	0xFFFFFFFF	
USER	0xFFFFFFFF	Option byte USER value
FMC_SECMCFG0	0x003F0000	
SECM0_EPAGE	0x3F	End page of secure mark area 0
SECM0_SPAGE	0x0	Start page of secure mark area 0
FMC_DMP0	0x00000000	
DMP0EN	☐	DMP area 0 enable
DMP0_EPAGE	0x0	End page of DMP area 0

Tips: please refer to the chip User Manual to modify the option bytes.

3. Software development

This article uses two development environments, MDK-ARM v5.28.0.0 and IAR v8.50.9. Please ensure that the corresponding software and device package are correctly installed. The software can be obtained from the official website of **KEIL** and **IAR**. The device package (GD32W51x_AddOn) can be obtained from **GD32MCU.COM**.

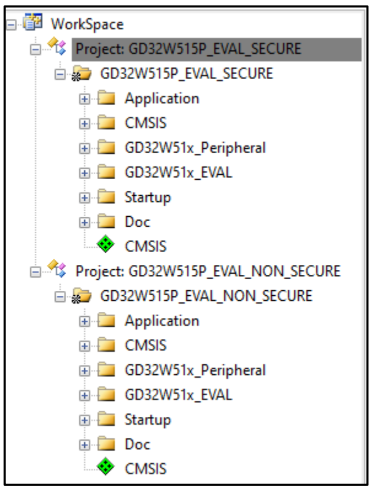
The sample project has secure and non-secure projects that use different Flash and SRAM. The secure project uses the first 256KB in the FMC (starting address 0x0C000000, size 0x40000), and the memory uses SRAM0 blocks (starting address 0x30000000, size 0x10000). The non-secure project uses the last 1792KB in the FMC (starting address 0x08040000, size 0x1C0000) and the memory uses SRAM1 block (starting address 0x20010000, size 0x10000).

3.1. Developing TrustZone in Keil

Multiple project can be used to develop two projects at the same time, as shown in ***Figure 3-1. Keil project file structure***, project path:

GD32W51x_Demo_Suites\GD32W515P_EVAL_Demo_Suites\Projects\23_Trustzone\MDK-ARM\GD32W515P_EVAL_TRUSTZONE.uvmpw

Figure 3-1. Keil project file structure

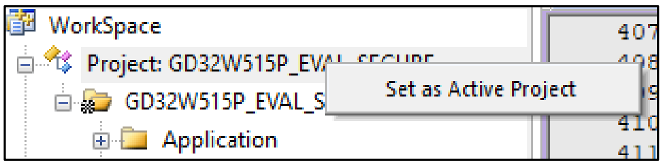


```
WorkSpace
  Project: GD32W515P_EVAL_SECURE
    GD32W515P_EVAL_SECURE
      Application
      CMSIS
      GD32W51x_Peripheral
      GD32W51x_EVAL
      Startup
      Doc
      CMSIS
  Project: GD32W515P_EVAL_NON_SECURE
    GD32W515P_EVAL_NON_SECURE
      Application
      CMSIS
      GD32W51x_Peripheral
      GD32W51x_EVAL
      Startup
      Doc
      CMSIS
```

3.1.1. Secure Project

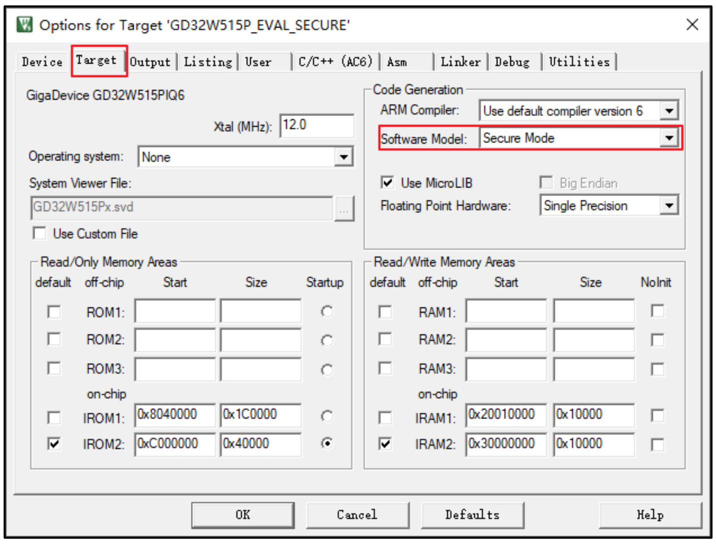
1. Select secure project as current project, as shown in ***Figure 3-2. Select secure project***.

Figure 3-2. Select secure project



2. In Project/Options for Target/Target, select Code Generation/Software Model as Secure Mode, as shown in *Figure 3-3. Select Secure mode*.

Figure 3-3. Select Secure mode


Options for Target 'GD32W515P_EVAL_SECURE'

Device | Target | Output | Listing | User | C/C++ (AC6) | Asm | Linker | Debug | Utilities

GigaDevice GD32W515PIQ6

Xtal (MHz): 12.0

Operating system: None

System Viewer File: GD32W515Px.svd

Use Custom File

Code Generation
ARM Compiler: Use default compiler version 6
Software Model: Secure Mode

Use MicroLIB [✓]   Big Endian

Floating Point Hardware: Single Precision

Read/Only Memory Areas

default	off-chip	Start	Size	Startup
	ROM1:			
	ROM2:			
	ROM3:			
	on-chip			
	IROM1:	0x8040000	0x1C0000	
✓	IROM2:	0xC000000	0x40000	●

Read/Write Memory Areas

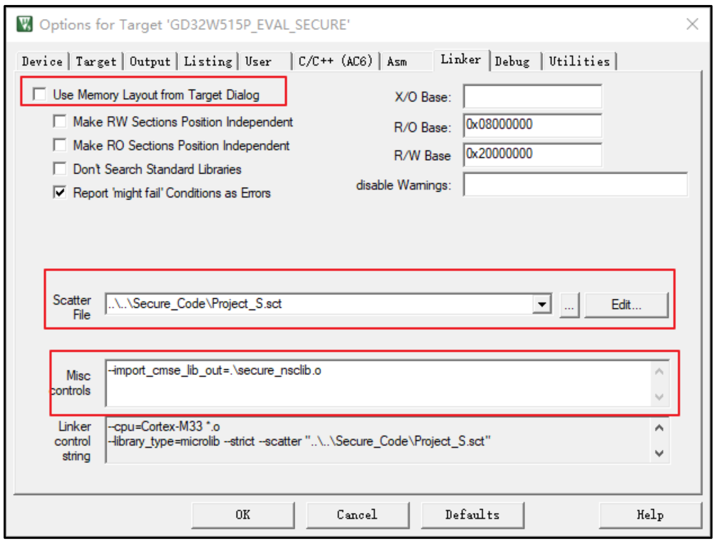
default	off-chip	Start	Size	NoInit
	RAM1:			
	RAM2:			
	RAM3:			
	on-chip			
	IRAM1:	0x20010000	0x10000	
✓	IRAM2:	0x30000000	0x10000	

OK | Cancel | Defaults | Help

3. Secure code and non-secure code use different Flash and SRAM, and use a scattered load file to assign the Flash and SRAM for secure code. In Project/Options for Target/Linker, select not to assign addresses using the Target interface, select scatter load file, and set the output library for NSC functions. See *Figure 3-4. Select scattered load file and NSC output library*.

The output command is --import_cmse_lib_out=.\secure_nsclib.o. The compiler will compile the code at compile time with __attribute((cmse_nonsecure_entry)) identifying code into.\secure_nsclib.o. Non-secure code can access function of secure_nsclib.o.

Figure 3-4. Select scattered load file and NSC output library


Options for Target 'GD32W515P_EVAL_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: 0x08000000

R/W Base 0x20000000

disable Warnings:

Scatter File: ..\..\Secure_Code\Project_S.sct

Misc controls: --import_cmse_lib_out=.\secure_nsclib.o

Linker control string:
```
--cpu=Cortex-M33 *.o
--library_type=microlib --strict --scatter "..\..\Secure_Code\Project_S.sct"
```

OK | Cancel | Defaults | Help

4. The secure project uses Flash address 0x0C000000 with size 0x40000 and SRAM address 0x30000000 with size 0x10000. NSC function is assigned to address 0x0C03E000 and size is 0x00002000. Project_S.sct code is shown in *Table 3-1. Project_S.sct code*.

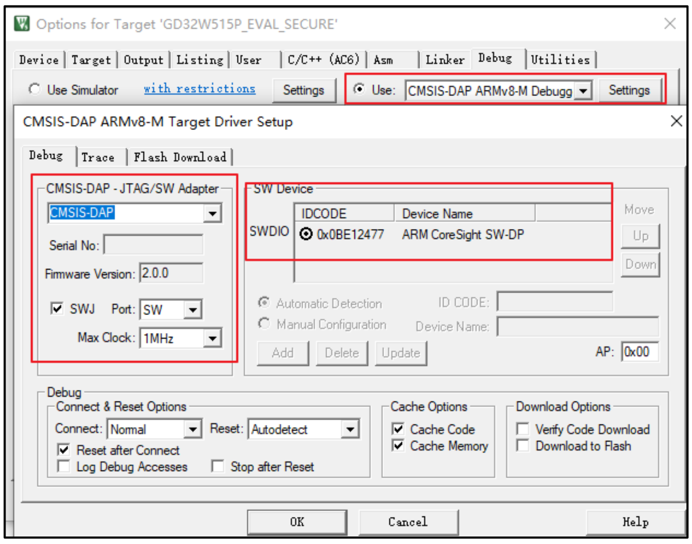
Table 3-1. Project_S.sct code

```
LR_IROM1  0x0C000000  0x00040000    {      ; load region size_region
  ER_IROM1  0x0C000000  0x0003E000    {   ; load address = execution address
    *.o (RESET,  +First)
    *(InRoot$$Sections)
    .ANY (+RO)
    .ANY (+XO)
  }
  RW_IRAM2  0x30000000  0x00010000   {   ; RW data
    .ANY (+RW +ZI)
  }
}

LR_IROM2  0x0C03E000  0x00002000   {
  ER_IROM2  0x0C03E000  0x00002000   {   ; load address = execution address
    *(Veneer$$CMSE)                        ; check with partition.h
  }
```

5. Use GD-Link for debugging and downloading. Connect GD-Link interface of Eval-board directly to PC-USB. Select CMSIS-DAP ARMv8-M Debugger under Project/Options for Target/Debug. Identify the debugger ID in Settings, use the SW interface and set the maximum clock to 1MHz. See ***Figure 3-5. Debugger setup in secure project***.

Figure 3-5. Debugger setup in secure project



CMSIS-DAP - JTAG/SW Adapter: CMSIS-DAP
Serial No:
Firmware Version: 2.0.0
☑ SWJ Port: SW
Max Clock: 1MHz

SW Device

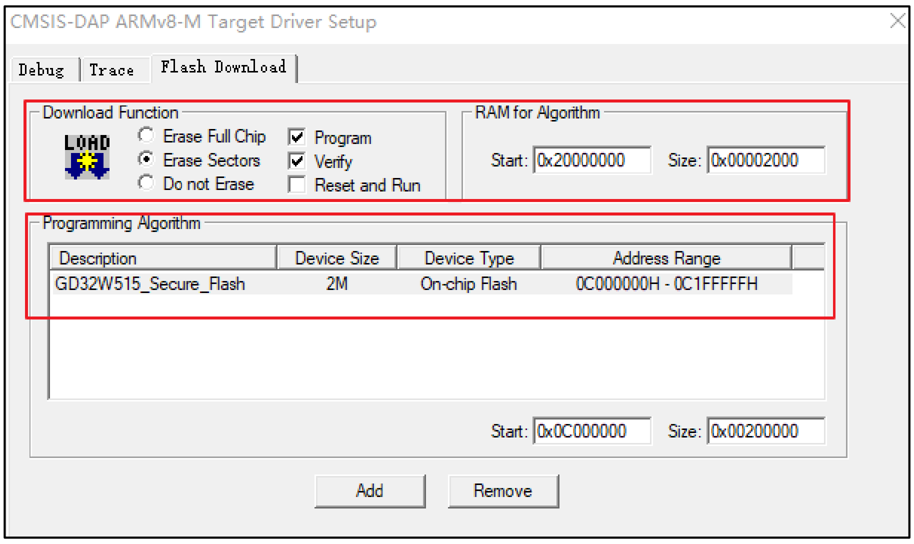
	IDCODE	Device Name
SWDIO	● 0x0BE12477	ARM CoreSight SW-DP

● Automatic Detection ID CODE:
○ Manual Configuration Device Name:
AP: 0x00

Debug
Connect & Reset Options: Connect: Normal Reset: Autodetect
☑ Reset after Connect
☐ Log Debug Accesses ☐ Stop after Reset
Cache Options: ☑ Cache Code ☑ Cache Memory
Download Options: ☐ Verify Code Download ☐ Download to Flash

6. Under Flash Download TAB, select Erase Sectors and uncheck Reset and Run. Add programming algorithm GD32W515_Secure_Flash. GD32W515_Secure_Flash is provided by the installed device package. See ***Figure 3-6. Download algorithm and function in secure project***.

Figure 3-6. Download algorithm and function in secure project



CMSIS-DAP ARMv8-M Target Driver Setup

Debug | Trace | Flash Download

Download Function
- Erase Full Chip
- [✓ Erase Sectors]
- Do not Erase
- [✓] Program
- [✓] Verify
- [ ] Reset and Run

RAM for Algorithm
Start: 0x20000000 Size: 0x00002000

Programming Algorithm

Description	Device Size	Device Type	Address Range
GD32W515_Secure_Flash	2M	On-chip Flash	0C000000H - 0C1FFFFFH

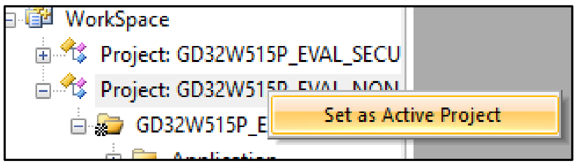
Start: 0x0C000000 Size: 0x00200000

Add | Remove

3.1.2. Non-secure Project

1. Select non-secure project as current project, as shown in ***Figure 3-7. Select non-secure project***.

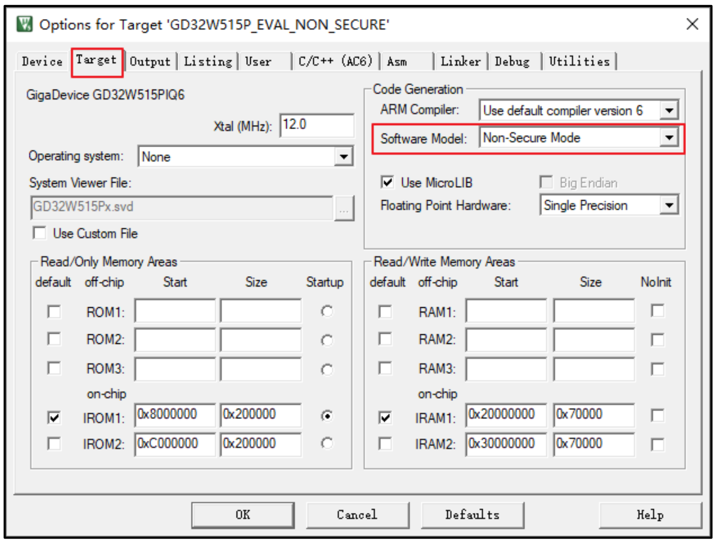
Figure 3-7. Select non-secure project



WorkSpace
Project: GD32W515P_EVAL_SECU
Project: GD32W515P_EVAL_NON
GD32W515P_E
Set as Active Project

2. In Project/Options for Target/Target TAB, select Code Generation/Software Model as Non-secure Mode, as shown in ***Figure 3-8. Select non-secure mode***.

Figure 3-8. Select non-secure mode



Options for Target 'GD32W515P_EVAL_NON_SECURE'

Device | Target | Output | Listing | User | C/C++ (AC6) | Asm | Linker | Debug | Utilities

GigaDevice GD32W515PIQ6

Xtal (MHz): 12.0

Operating system: None

System Viewer File: GD32W515Px.svd

[ ] Use Custom File

Code Generation
ARM Compiler: Use default compiler version 6
Software Model: Non-Secure Mode

[✓] Use MicroLIB [ ] Big Endian

Floating Point Hardware: Single Precision

Read/Only Memory Areas

default	off-chip	Start	Size	Startup
☐	ROM1:			○
☐	ROM2:			○
☐	ROM3:			○
	on-chip			
✓	IROM1:	0x8000000	0x200000	●
☐	IROM2:	0xC000000	0x200000	○

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
☐	RAM1:			☐
☐	RAM2:			☐
☐	RAM3:			☐
	on-chip			
✓	IRAM1:	0x20000000	0x70000	☐
☐	IRAM2:	0x30000000	0x70000	☐

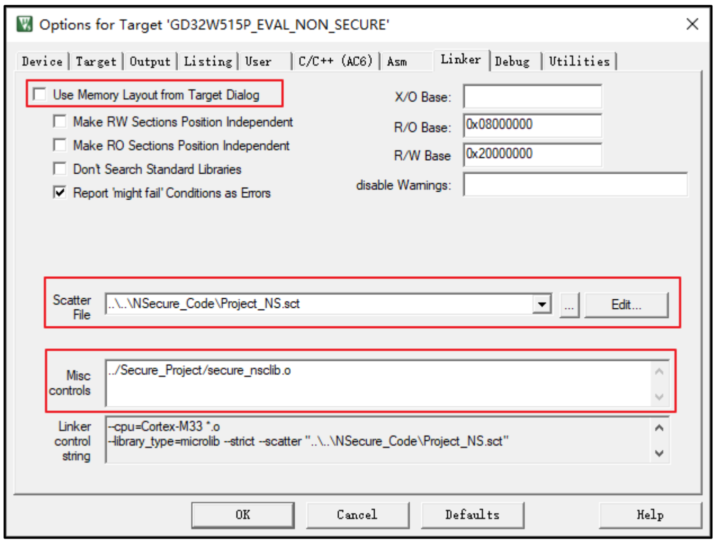
OK | Cancel | Defaults | Help

3. Secure code and non-secure code use different Flash and SRAM, and use a distributed load file to assign the Flash and SRAM for secure code. In Project/Options for Target/Linker,

select not to assign addresses using the Target interface, select the scattered load file, and use the NSC function library exported by the Security Project. See ***Figure 3-9. Select scattered load file and import NSC library***.

Library import command ../Secure_Project/secure_nsclib.o, non-secure project can directly call the __attribute((cmse_nonsecure_entry)) identification code in the security project.

Figure 3-9. Select scattered load file and import NSC library


Options for Target 'GD32W515P_EVAL_NON_SECURE'
Device | Target | Output | Listing | User | C/C++ (AC6) | Asm | Linker | Debug | Utilities
OK
Cancel
Defaults
Help

☐ 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: 0x08000000
R/W Base: 0x20000000
disable Warnings:

Scatter File: ..\..\NSecure_Code\Project_NS.sct

Misc controls: ../Secure_Project/secure_nsclib.o

Linker control string: --cpu=Cortex-M33 *.o
--library_type=microlib --strict --scatter "..\..\NSecure_Code\Project_NS.sct"

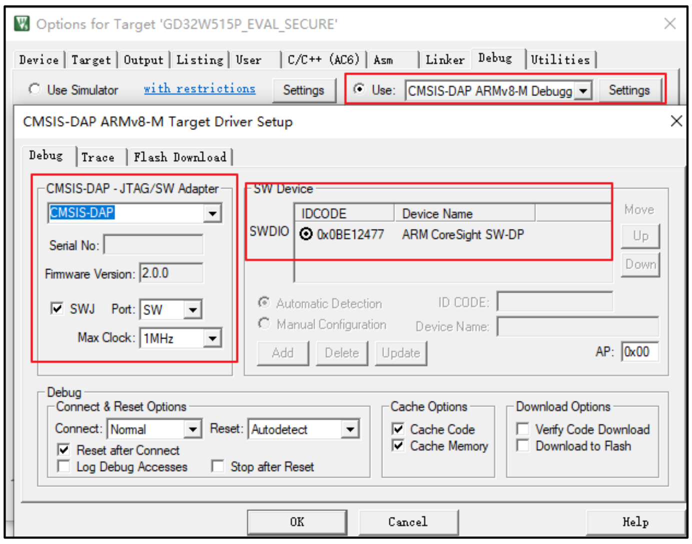
4. Non-secure code uses Flash address 0x08040000 with size 0x001C0000 and SRAM address 0x20010000 with size 0x10000. Project_NS.sct code is shown in ***Table 3-2. Project_NS.sct code***.

Table 3-2. Project_NS.sct code

```
LR_IROM1  0x08040000 0x001C0000   {      ; load region size_region
  ER_IROM1  0x08040000 0x001C0000   {   ; load address = execution address
   *.o (RESET,  +First)
   *(InRoot$$Sections)
   .ANY (+RO)
   .ANY (+XO)
  }
  RW_IRAM1  0x20010000 0x00010000   {   ; RW data
   .ANY (+RW +ZI)
  }
}
```

5. Use GD-Link for debugging and downloading. Connect GD-Link interface of Eval-board directly to PC-USB. Select CMSIS-DAP ARMv8-M Debugger under Project/Options for Target/Debug. Identify the debugger ID in Settings, use the SW interface and set the maximum clock to 1MHz. See ***Figure 3-10. Debugger setup in non-secure project***.

Figure 3-10. Debugger setup in non-secure project


Options for Target 'GD32W515P_EVAL_SECURE'

Device | Target | Output | Listing | User | C/C++ (AC6) | Asm | Linker | Debug | Utilities

Use Simulator with restrictions Settings [✓ Use:] CMSIS-DAP ARMv8-M Debugg Settings

CMSIS-DAP ARMv8-M Target Driver Setup

Debug | Trace | Flash Download

CMSIS-DAP - JTAG/SW Adapter: CMSIS-DAP
Serial No:
Firmware Version: 2.0.0
[✓] SWJ Port: SW
Max Clock: 1MHz

SW Device

	IDCODE	Device Name
SWDIO	0x0BE12477	ARM CoreSight SW-DP

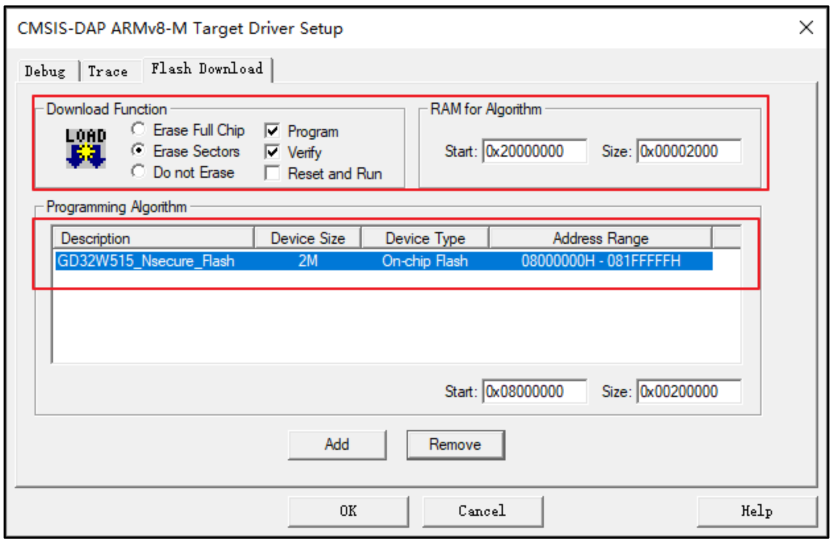
Move Up Down
Automatic Detection ID CODE:
Manual Configuration Device Name:
Add Delete Update AP: 0x00

Debug
Connect & Reset Options: Connect: Normal Reset: Autodetect
[✓] Reset after Connect
[ ] Log Debug Accesses [ ] Stop after Reset
Cache Options: [✓] Cache Code [✓] Cache Memory
Download Options: [ ] Verify Code Download [ ] Download to Flash

OK Cancel Help

6. Under Flash Download TAB, select Erase Sectors and uncheck Reset and Run. Add programming algorithm GD32W515_Nsecure_Flash. GD32W515_Nsecure_Flash is provided by the installed device package. See ***Figure 3-11. Download algorithm and function in non-secure project***.

Figure 3-11. Download algorithm and function in non-secure project


CMSIS-DAP ARMv8-M Target Driver Setup

Debug | Trace | Flash Download

Download Function
Erase Full Chip [✓] Program
[✓ Erase Sectors] [✓] Verify
Do not Erase [ ] Reset and Run

RAM for Algorithm
Start: 0x20000000 Size: 0x00002000

Programming Algorithm

Description	Device Size	Device Type	Address Range
GD32W515_Nsecure_Flash	2M	On-chip Flash	08000000H - 081FFFFFH

Start: 0x08000000 Size: 0x00200000

Add Remove

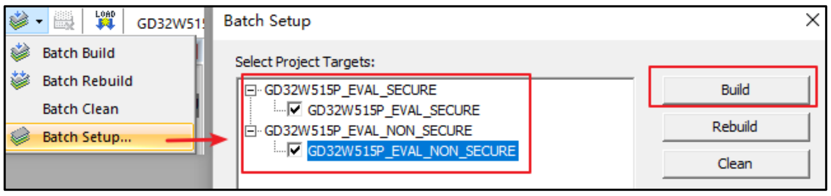
OK Cancel Help

3.1.3. Compile Project

Firstly compile secure project and then non-secure project. Non-secure project need secure_nsclib.o library generated in secure project. Compile the secure and insecure projects in order.

Batch setup can also compile two projects in order, as shown in ***Figure 3-12. Compile Project***. Select two projects and click Build. The order is determined by project directory sequence, which can be modified by project/Manage/Multi-Project Workspace.

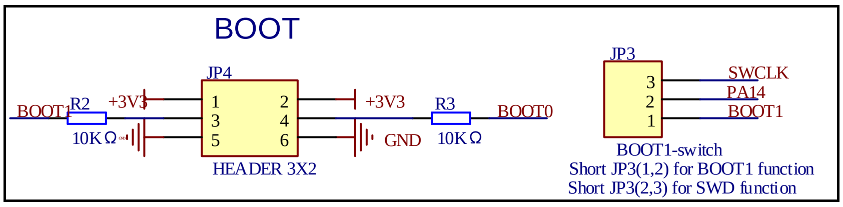
Figure 3-12. Compile Project



3.1.4. Download Project

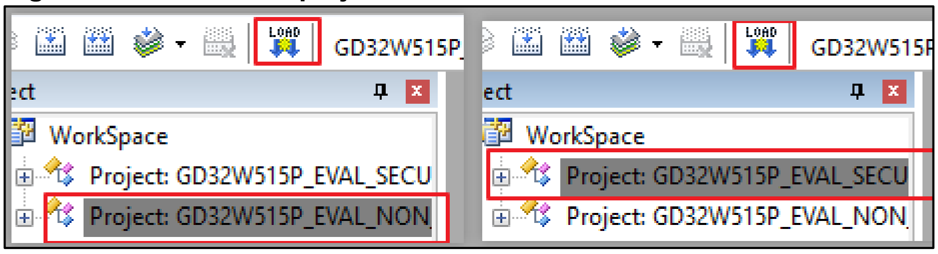
Set up Eval-board correctly before downloading code. JP4 (BOOT0/BOOT1) connects to L. MCU will boot from the secure code. JP3 connects to SWD. See *Figure 3-13. Boot option*. JP21 connects to USART for printf. Connect Eval-board GD-Link and USART to PC, and ensure that software drivers are correctly installed.

Figure 3-13. Boot option



Firstly download non-secure code and then secure code. MCU always boots from secure code and jumps to non-secure code. See *Figure 3-14. Download project*.

Figure 3-14. Download project


GD32W515P_
ect
WorkSpace
Project: GD32W515P_EVAL_SECU
Project: GD32W515P_EVAL_NON
GD32W515P
ect
WorkSpace
Project: GD32W515P_EVAL_SECU
Project: GD32W515P_EVAL_NON

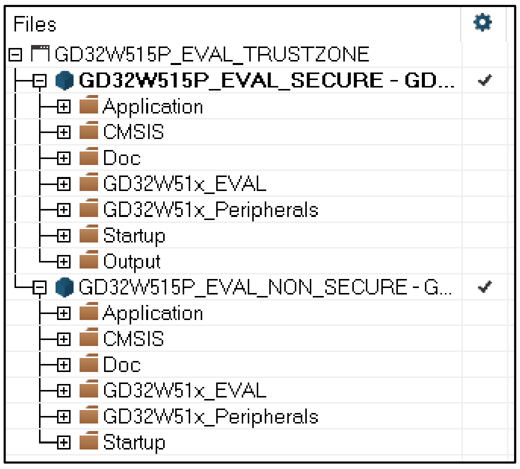
Reset and Run are not checked in the Flash Download option. Press manually Reset button to restart MCU. Two messages will be printed and two LED lights flash respectively.

3.2. Developing TrustZone in IAR

Two projects can be developed at the same time using multiple project methods. The file structure is shown in *Figure 3-15. IAR project file structure*, project path:

GD32W51x_Demo_Suites\GD32W515P_EVAL_Demo_Suites\Projects\23_Trustzone\EWARM\GD32W515P_EVAL_TRUSTZONE.eww

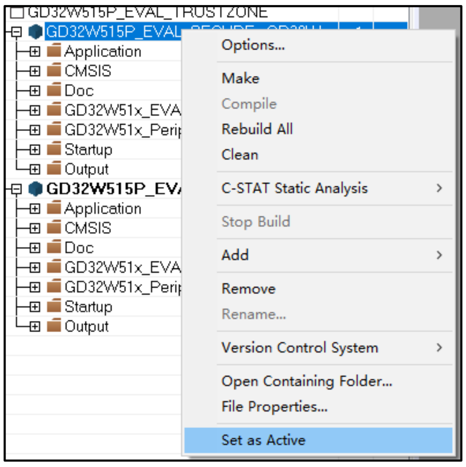
Figure 3-15. IAR project file structure



3.2.1. Secure Project

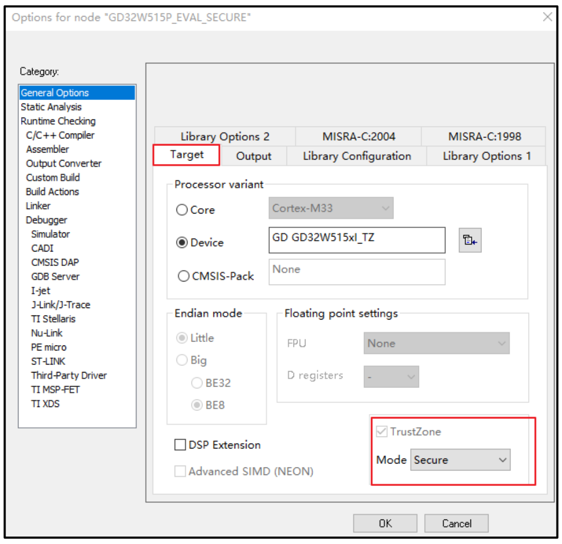
1. Select secure project as active project, as shown in *Figure 3-16. Select secure project*.

Figure 3-16. Select secure project



2. In Project / Options / General Options / Target, select TrustZone / Mode as Secure, as shown in *Figure 3-17. Select Secure mode*.

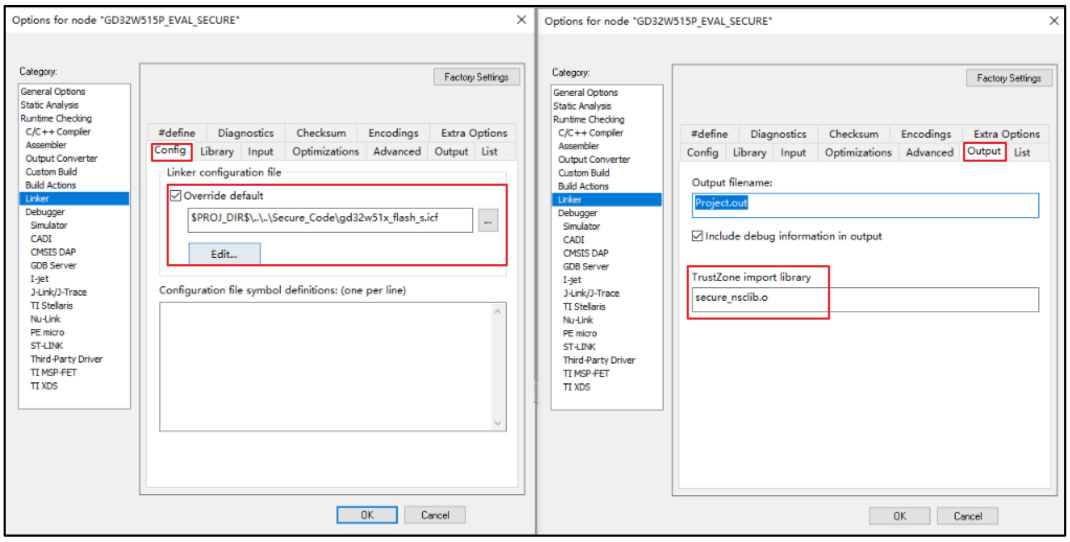
Figure 3-17. Select Secure mode



3. Secure code and non-secure code use different Flash and SRAM, and use a scattered load file to assign the Flash and SRAM for secure code. In Project / Options / Linker / Config, select scatter load file and set the output library for NSC functions. See *Figure 3-18. Select scattered load file and NSC output library*.

The output command is --import_cmse_lib_out=.\secure_nsclib.o. The compiler will compile the code at compile time with __attribute((cmse_nonsecure_entry)) identifying code into.\secure_nsclib.o. Non-secure code can access functions of secure_nsclib.o.

Figure 3-18. Select scattered load file and NSC output library


Options for node "GD32W515P_EVAL_SECURE"
Linker
Config
Linker configuration file
Override default
$PROJ_DIR$\..\..\Secure_Code\gd32w51x_flash_s.icf
Edit...
Output
Output filename:
Project.out
Include debug information in output
TrustZone import library
secure_nsclib.o
OK
Cancel

4. The secure project uses Flash address 0x0C000000 with size 0x40000 and SRAM address 0x30000000 with size 0x10000. NSC function is assigned to address 0x0C03E000 and size is 0x00002000. gd32w51x_flash_s.icf code is shown in *Table 3-3. gd32w51x_flash_s.icf code*.

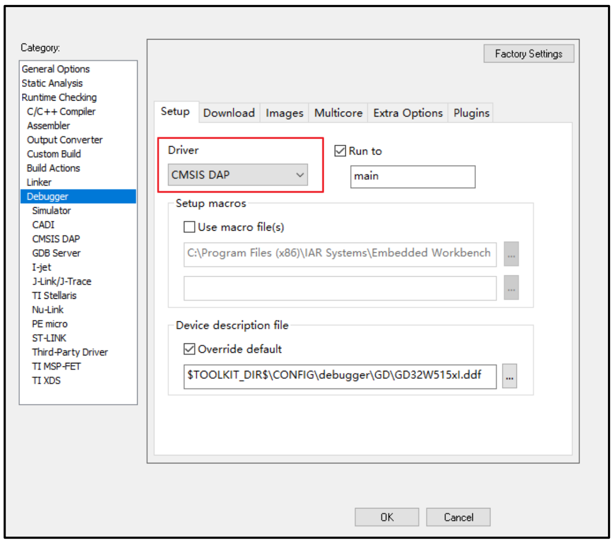
Table 3-3. gd32w51x_flash_s.icf code

```
/*-Specials-*/
```

```
define symbol __ICFEDIT_intvec_start__  = 0x0C000000;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__  = 0x0C000000;
define symbol __ICFEDIT_region_ROM_end__     = 0x0C03FFFF;
define symbol __ICFEDIT_region_RAM_start__  = 0x30000000;
define symbol __ICFEDIT_region_RAM_end__     = 0x3000FFFF;
......
define symbol __region_ROM_NSC_start__     = 0x0C03E000;
define symbol __region_ROM_NSC_end__       = 0x0C03FFFF;
......
define region ROM_region          = mem:[from __ICFEDIT_region_ROM_start__          to
__ICFEDIT_region_ROM_end__];
define region ROM_NSC_region    = mem:[from __region_ROM_NSC_start__                to
__region_ROM_NSC_end__];
define region RAM_region          = mem:[from __ICFEDIT_region_RAM_start__          to
__ICFEDIT_region_RAM_end__];
......
place in ROM_region        { readonly };
place in ROM_NSC_region  { section Veneer$$CMSE };
place in RAM_region        { readwrite, block HEAP, block CSTACK };
```

5. Use GD-Link for debugging and downloading. Select CMSIS-DAP under Project / Options / Debugger. See ***Figure 3-19. Debugger setup in secure project***.

Figure 3-19. Debugger setup in secure project



Driver: CMSIS DAP

☑ Run to: main

Setup macros

☐ Use macro file(s)

C:\Program Files (x86)\IAR Systems\Embedded Workbench

Device description file

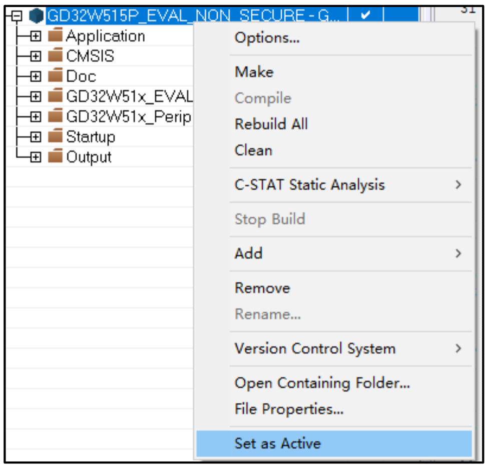
☑ Override default

$TOOLKIT_DIR$\CONFIG\debugger\GD\GD32W515xI.ddf

3.2.2. Non-secure Project

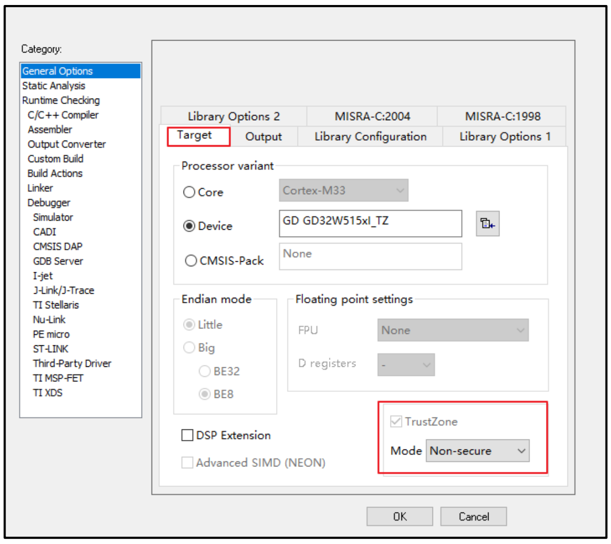
1. Select non-secure project as active project, as shown in ***Figure 3-20. Select non-secure project***.

Figure 3-20. Select non-secure project



2. In Project / Options / General Options / Target, select TrustZone / Mode as Non-secure, as shown in ***Figure 3-21. Select non-secure mode***.

Figure 3-21. Select non-secure mode



Processor variant: Core ○ Cortex-M33; [✓ Device] GD GD32W515xI_TZ; CMSIS-Pack ○ None

Endian mode: Little / Big; BE32 / BE8

Floating point settings: FPU None; D registers -

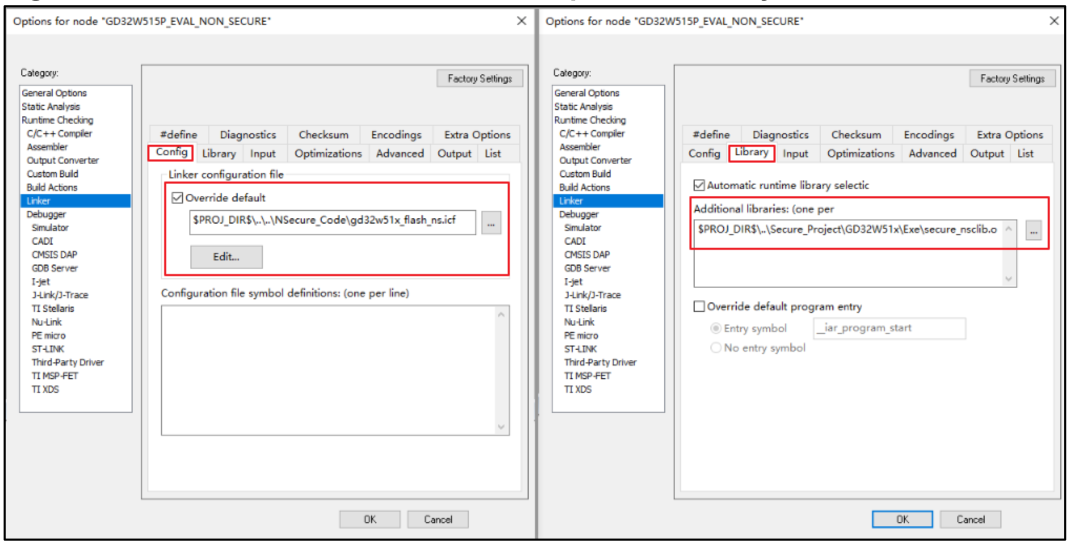
DSP Extension ☐
Advanced SIMD (NEON) ☐

TrustZone ✓
Mode Non-secure

3. Secure code and non-secure code use different Flash and SRAM, and use a scattered load file to assign the Flash and SRAM for secure code. In Project / Options / Linker / Config, select scatter load file and use the NSC function library exported by the Security Project. See ***Figure 3-22. Select scattered load file and import NSC library***.

Library import command $PROJ_DIR$\..\Secure_Project\GD32W51x\Exe\secure_nsclib.o, non-secure project can directly call the __attribute((cmse_nonsecure_entry)) identification code in the security project.

Figure 3-22. Select scattered load file and import NSC library



Linker configuration file
☑ Override default
$PROJ_DIR$\..\..\NSecure_Code\gd32w51x_flash_ns.icf
Configuration file symbol definitions: (one per line)

☑ Automatic runtime library selectic
Additional libraries: (one per
$PROJ_DIR$\..\Secure_Project\GD32W51x\Exe\secure_nsclib.o
☐ Override default program entry
◉ Entry symbol __iar_program_start
○ No entry symbol

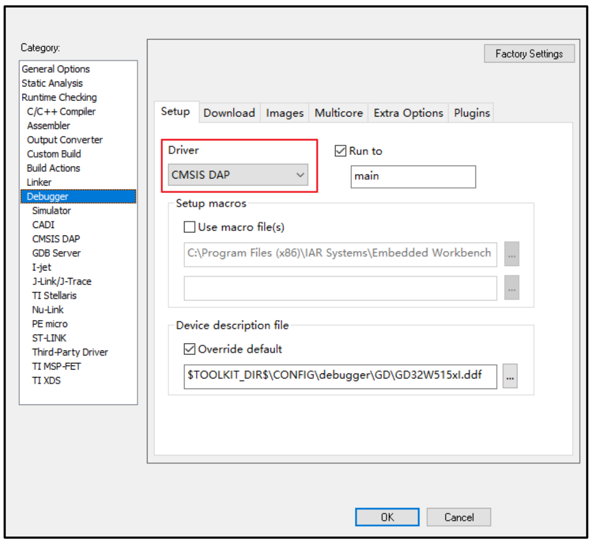
4. Non-secure code uses Flash address 0x08040000 with size 0x001C0000 and SRAM address 0x20010000 with size 0x10000. Project_NS.sct code is shown in ***Table 3-4. gd32w51x_flash_ns.icf code***.

Table 3-4. gd32w51x_flash_ns.icf code

```
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x08040000;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = 0x08040000;
define symbol __ICFEDIT_region_ROM_end__   = 0x081FFFFF;
define symbol __ICFEDIT_region_RAM_start__ = 0x20010000;
define symbol __ICFEDIT_region_RAM_end__   = 0x2001FFFF;
......
define region ROM_region    = mem:[from __ICFEDIT_region_ROM_start__ to
__ICFEDIT_region_ROM_end__];
define region RAM_region    = mem:[from __ICFEDIT_region_RAM_start__ to
__ICFEDIT_region_RAM_end__];
......
place in ROM_region   { readonly };
place in RAM_region   { readwrite, block HEAP, block CSTACK};
```

5. Use GD-Link for debugging and downloading. Select CMSIS-DAP under Project / Options / Debugger. See ***Figure 3-23. Debugger setup in non-secure project***.

Figure 3-23. Debugger setup in non-secure project



Driver: CMSIS DAP

☑ Run to: main

Setup macros

☐ Use macro file(s)

C:\Program Files (x86)\IAR Systems\Embedded Workbench

Device description file

☑ Override default

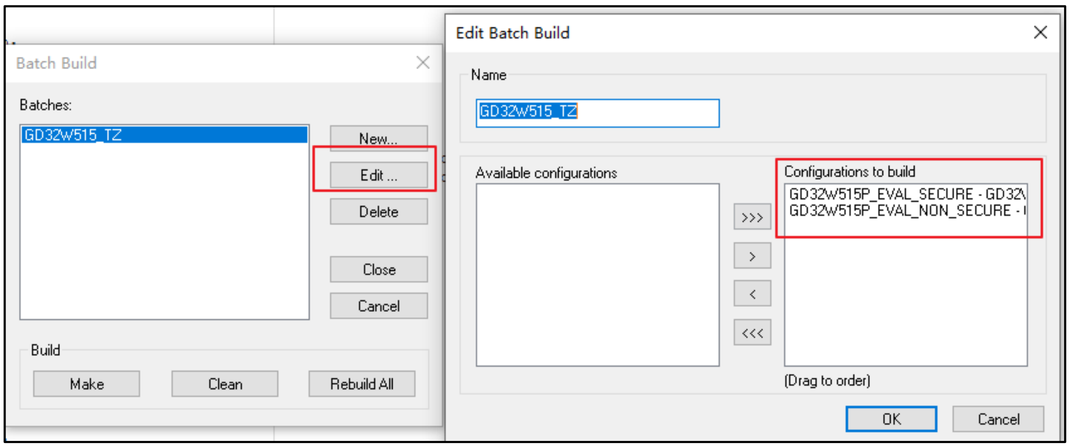
$TOOLKIT_DIR$\CONFIG\debugger\GD\GD32W515xl.ddf

3.2.3. Compile Project

Firstly compile secure project and then non-secure project. Non-secure project need secure_nsclib.o library generated in secure project. Compile the secure and insecure projects in order.

Batch setup can also compile two projects in order, as shown in ***Figure 3-24. Compile project***. The order is determined by the order of addition.

Figure 3-24. Compile project



GD32W515_TZ

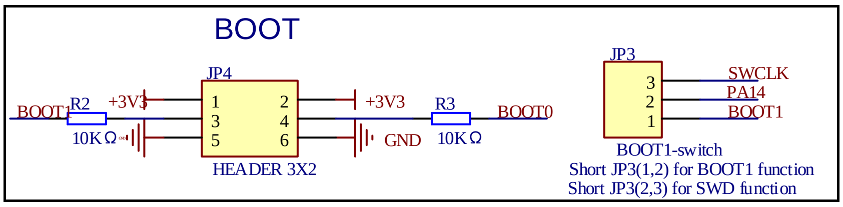
Name: GD32W515_TZ

Configurations to build:
GD32W515P_EVAL_SECURE - GD32\
GD32W515P_EVAL_NON_SECURE - I

3.2.4. Download Project

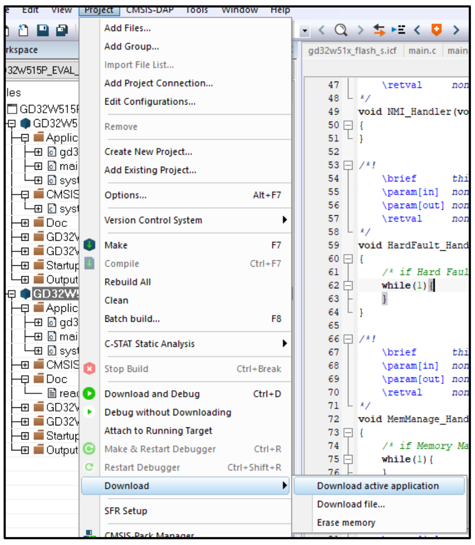
Set up Eval-board correctly before downloading code. JP4 (BOOT0/BOOT1) connects to L. MCU will boot from the secure code. JP3 connects to SWD. See ***Figure 3-25. Boot option***. JP21 connects to USART for printf. Connect Eval-board GD-Link and USART to PC, and ensure that software drivers are correctly installed.

Figure 3-25. Boot option



Firstly download non-secure code and then secure code. MCU always boots from secure code and jumps to non-secure code. See ***Figure 3-26. Download project***.

Figure 3-26. Download project



```
    \retval     non
*/
void NMI_Handler(vo
{
}

/*!
    \brief      thi
    \param[in]  non
    \param[out] non
    \retval     non
*/
void HardFault_Hand
{
    /* if Hard Faul
    while(1){
    }
}

/*!
    \brief      thi
    \param[in]  non
    \param[out] non
    \retval     non
*/
void MemManage_Hand
{
    /* if Memory Ma
    while(1){
```

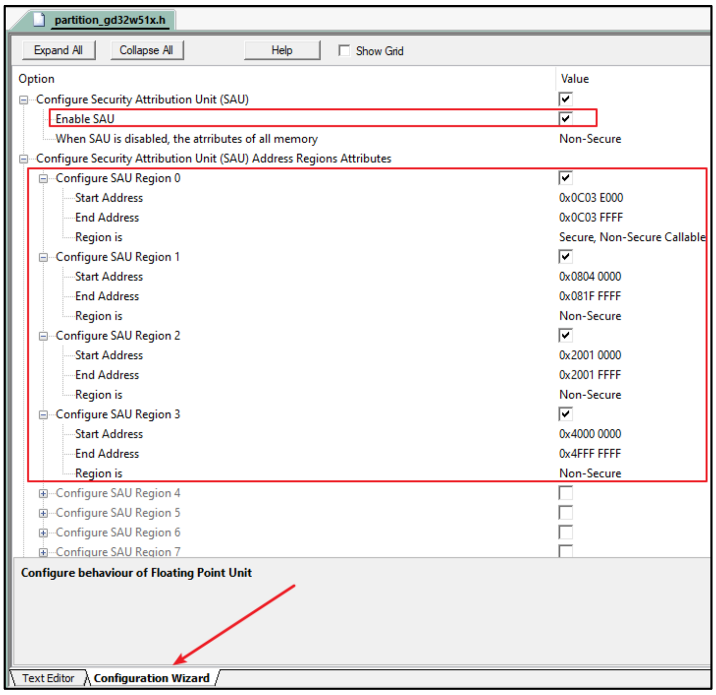
Reset and Run are not checked in the Flash Download option. Press manually Reset button to restart MCU. Two messages will be printed and two LED lights flash respectively.

4. Code interpretation

4.1. Secure Project

If Code Generation/Software Model is set to Secure Mode in the Project/Options for Target/Target TAB, __ARM_FEATURE_CMSE is defined as 3. After power-on and reset, MCU will execute system_gd32w51x.c/SystemInit()/sau_region_config() function to configure the SAU based on the partition_gd32w51x.h content. The partition_gd32w51x.h file can be configured using the Configuration Wizard interface and generate Text Editor code. As shown in ***Figure 4-1. SAU configuration***. SAU is enabled and four regions are configured. Region0 configures NSC addres, corresponding to Project_S.sct. Region1 configures non-secure code address. Region2 configures non-secure SRAM address, corresponding to Project_NS.sct. Region3 configures peripheral address to be non-secure. Peripheral can be accessed in both secure or non-secure code.

Figure 4-1. SAU configuration



Option	Value
Configure Security Attribution Unit (SAU)	✓
Enable SAU	✓
When SAU is disabled, the atrributes of all memory	Non-Secure
Configure Security Attribution Unit (SAU) Address Regions Attributes	
Configure SAU Region 0	✓
Start Address	0x0C03 E000
End Address	0x0C03 FFFF
Region is	Secure, Non-Secure Callable
Configure SAU Region 1	✓
Start Address	0x0804 0000
End Address	0x081F FFFF
Region is	Non-Secure
Configure SAU Region 2	✓
Start Address	0x2001 0000
End Address	0x2001 FFFF
Region is	Non-Secure
Configure SAU Region 3	✓
Start Address	0x4000 0000
End Address	0x4FFF FFFF
Region is	Non-Secure
Configure SAU Region 4	
Configure SAU Region 5	
Configure SAU Region 6	
Configure SAU Region 7	

Configure behaviour of Floating Point Unit

In main function of secure project, secure interrupt is enabled, Systick configuration, FMC configuration, TZBMPC configuration, LED configuration, USART configuration, and finally jump to non-secure code. See ***Table 4-1. main code in secure project***.

Table 4-1. main code in secure project

```
int main(void)
{
    /* enable SecureFault handler */
    SCB->SHCSR |= SCB_SHCSR_SECUREFAULTENA_Msk;
    /* configure systick */
    systick_config();
    /* configure mark secure pages */
    if(SECM_SPAGE != (FMC_SECMCFG0 & 0x3FF) || SECM_EPAGE != ((FMC_SECMCFG0 >>
16) & 0x3FF)){
        fmc_secmark_config();
    }
    if(0U == (FMC_OBR & FMC_OBR_TZEN)){
        /* enable trustzone */
        fmc_trustzone_enable();
    }
    /* configure TZBMPC */
    tzbmpc_config();
    led_config();
    com_config();
    /* setup and jump to non-secure */
    nonsecure_init();
    while(1){
    }
}
```

Function fmc_secmark_config() configures pages 0-63 of FMC. First 256KB is used for secure code.

Table 4-2. FMC configuration

```
void fmc_secmark_config(void)
{
    fmc_unlock();
    ob_unlock();
    /* configure mark secure pages */
    ob_secmark_config(SECM_SPAGE, SECM_EPAGE, SECM_INDEX0);
    ob_start();
    while(0U != (FMC_SECSTAT & (FMC_SECSTAT_SECBUSY))){
    }
    ob_reload();
    ob_lock();
    fmc_lock();
}
```

Function tzbmpc_config() configures SRAM1 as non-secure. This area is used for non-secure code.

Table 4-3. SRAM configuration

```
void tzbmpc_config(void)
{
        uint16_t block_number = 0U;
        /* enable TZPCU clock */
        rcu_periph_clock_enable(RCU_TZPCU);
        /* SRAM1 is used to nonsecure code, so all blocks of SRAM1 should set to nonsecure */
        for(block_number = 0U; block_number <= TZBMPC1_BLOCK_NUMBER; block_number++){
            tzpcu_tzbmpc_block_secure_access_mode_config(TZBMPC1,             block_number,
BLOCK_SECURE_ACCESS_MODE_NSEC);
        }
}
```

Function led_config() configures LED1 and LED2 ports as non-secure. LED1 and LED2 will ce controlled by non-secure code. Function com_config() configures USART2 (serial port) as secure. USART2 will be controlled by secure code.

entry_cb_func_register() and non_secure_print() are non-secure callable functions marked with __attribute((cmse_nonsecure_entry). Non-secure code can call these functions directly to pass non-secure function addresses and information to print.

If secure code needs to call non-secure function, use cmse_nsfptr_create(func_addr) to register the function address and __attribute((cmse_nonsecure_call)) to convert the function to non-secure. Finally use the function. In this demo, non-secure code first calls the entry_cb_func_register() to get address of toggle_led1, and then calls nonsecure_func() to toggle LED1. The code is shown in ***Table 4-4. Call non-secure function in secure code***.

Table 4-4. Call non-secure function in secure code

```
#define CMSE_NS_ENTRY  __attribute((cmse_nonsecure_entry))
#define CMSE_NS_CALL   __attribute((cmse_nonsecure_call))
typedef void CMSE_NS_CALL (*ns_fptr)(void);
ns_fptr nonsecure_func = (ns_fptr)NULL;
.......
CMSE_NS_ENTRY  void entry_cb_func_register(void *callback)
{
        if(callback != NULL){
        nonsecure_func = (ns_fptr)cmse_nsfptr_create(callback);
        }
}
```

4.2. Non-secure project

NSC functions can be used in non-secure code after imported secure_nsclib.o library. In main() function, entry_cb_func_register() passes address of toggle_led1 to security code. Then configure Systick of non-secure domain, initialize LED port. The while loop periodically flips LED2 and calls the NSC function non_secure_print to print the information.

Table 4-5. Call secure function in non-secure code

```
extern void entry_cb_func_register(void *callback);
extern void non_secure_print(const char * str);
int main(void)
{
    entry_cb_func_register((void *)toggle_led1);
    /* configure systick */
    systick_config();
    gd_eval_led_init(LED1);
    gd_eval_led_init(LED2);
    while(1){
        /* toggle LED2 */
        gd_eval_led_toggle(LED2);
        non_secure_print("non-secure code toggle LED2.\r\n");
        delay_1ms(1000);
    }
}
```

5. Revision history

Table 5-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	Mar.3 2023