



Getting started with STM32 MCU's and Arm®TrustZone® development

Introduction

This document aims to provide guidelines using the EWARM and MDKARM software tool-chains on STM32L5, STM32U3, and STM32U5 series microcontrollers.

This application note provides a basis for building and debugging secure and nonsecure applications for devices based on Arm® Cortex®-M33 (Armv8-M architecture).

This document first gives an overview of the Arm® Cortex®-M33 and the TrustZone® concept.

This application note then describes the way of use EWARM and MDKARM with STM32L5, STM32U3, and STM32U5 series microcontrollers when the TrustZone® is enabled through TZEN option bit.

1 General information

This document applies to the STM32L5, STM32U3, and STM32U5 series single-core Arm®Cortex®-M33 based microcontrollers with Arm®TrustZone®.

Note: Arm and TrustZone are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.



Reference documents

[1]	RM0438	Reference manual STM32L552xx and STM32L562xx advanced Arm®-based 32-bit MCUs
[2]	RM0456	Reference manual STM32U5 Series Arm®-based 32-bit MCUs
[3]	RM0487	Reference manual STM32U3 Series Arm®-based 32-bit MCUs
[4]	Reference Manual	Armv8-M Architecture Reference Manual available from the Arm® web site.

2 Arm® Cortex®-M33 core overview

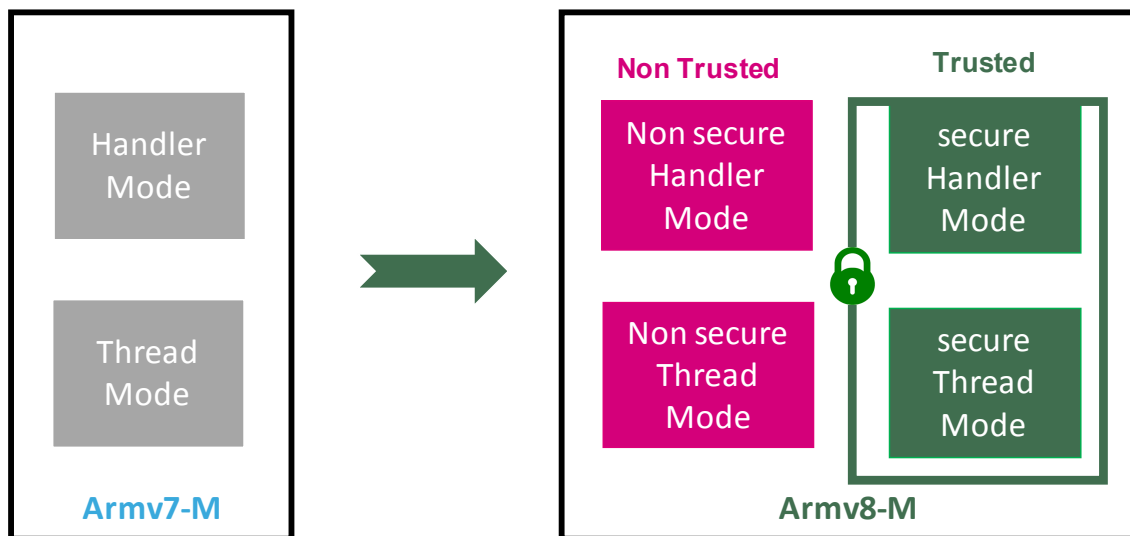
The Arm® Cortex®-M33 is the first full-feature implementation of Armv8-M with TrustZone® secure technology and digital signal processing functionality. The processor supports a large number of flexible configuration options to facilitate the deployment of a wide range of applications, and offers a dedicated co-processor interface for accelerating frequently used compute intensive operations. The Cortex®-M33 delivers an optimized balance between performance, power, security, and productivity.

3 TrustZone® concept of the Armv8-M

The Cortex®-M33 processor with TrustZone® has two security states (see Figure 1) and a number of associated features:

- secure state
- nonsecure state
- four stacks and four stack pointer registers
- hardware stack-limit checking
- support for programmable MPU-like security attribution unit (SAU)
- interface for system security notification
- visibility of secure code from a nonsecure (NS) domain restricted to predefined entry points
- exception hardware automatically saves and clears secure register states when switching to nonsecure
- extensive banking of interrupt or exception control, SysTick
- memory protection unit for each of the secure and nonsecure parts.

Figure 1. Security state in Armv8-M



Note: When the TrustZone® is enabled, by default the system starts up in the secure state.

4 SAU / IDAU - TrustZone® concept

TrustZone® security is activated by the TZEN option bit in the FLASH_OPTR register. When the TrustZone® is enabled, the security attribution unit (SAU) and implementation defined attribution unit (IDAU) define the access permissions based on secure and nonsecure states.

- IDAU: provides a first memory partition as secure or, nonsecure callable attributes. The IDAU memory map partition is not configurable and is fixed by hardware implementation.
- SAU: eight regions, used to overwrite IDAU in order to set secure areas and confirm nonsecure ones.
- The security state is selected based firstly on the IDAU security attribute, then combined with SAU security attribution. The resulting security attribution is the highest security setting of either IDAU and SAU.
- The "secure" security attribution priority has the highest secure priority, then nonsecure callable has a lower secure priority and nonsecure has the lowest secure priority. Any undefined region is secure by default

When the TrustZone® security is activated, the default security state is, for:

- The CPU: the Cortex®-M33 is in secure state after reset. The boot address must be in a secure address.
- The memory map: SAU is fully secure after reset. The whole memory map is fully secure. Up to eight SAU configurable regions are available for security attributions.
- Flash memory:
 - Flash security area is defined by watermark user options. All flash is fully secure.
 - Flash block based features are nonsecure after reset. Even if all the flash memory is nonsecure through IDAU/SAU and through the flash secure watermark option bytes, it is possible to configure volatile secure areas using the flash memory block based feature: any page is programmable on the fly as secure mode, using the flash interface block based configuration registers.
 - SRAM: all the SRAM is secure after reset. memory protection block based controller (MPCBB) is secure.
- Nonsecure memory view is identical to other Cortex®-M cores.
- The secure memory space is divided into two memory types:
 - Secure: containing secure program code and data, such as stack and heap.
 - Nonsecure callable (NSC): contains entry functions (for example entry points for APIs), this is to prevent nonsecure application from branching into invalid entry points.

5 Debugging modes

5.1 Invasive debug

Invasive debug is defined as a debug process where the user controls and observes the processor activity. Most debug features are considered as invasive debug as they enable the user to halt the processor and modify its state.

DBGEN and SPIDEN controls have invasive debug permissions.

5.2 Non-invasive debug

Non-invasive debug is defined as a debug process where the user observes the processor but does not control it. The Embedded Trace Macrocell™ (ETM) interface and the performance monitor registers are features of non-invasive debug.

NIDEN and SPNIDEN controls both have non-invasive debug permissions. Non-invasive debug is always permitted when invasive debug is permitted.

6 Debug access

6.1 Secure debug access

Secure debug access offers full visibility on all instruction execution, across all memory regions, and device peripherals. It allows the tracing and debugging of the secure and the nonsecure software running on the target. Debugging of secure firmware is only available in this mode.

Code running in secure state has access to both secure and nonsecure information.

6.2 Nonsecure debug access

The nonsecure debug view protects the secure memory and peripherals. These are invisible to the debugger in nonsecure mode. Debug and trace capabilities are limited to nonsecure system resources.

7 Flash memory protection

7.1 Readout protection level when TrustZone® is disabled

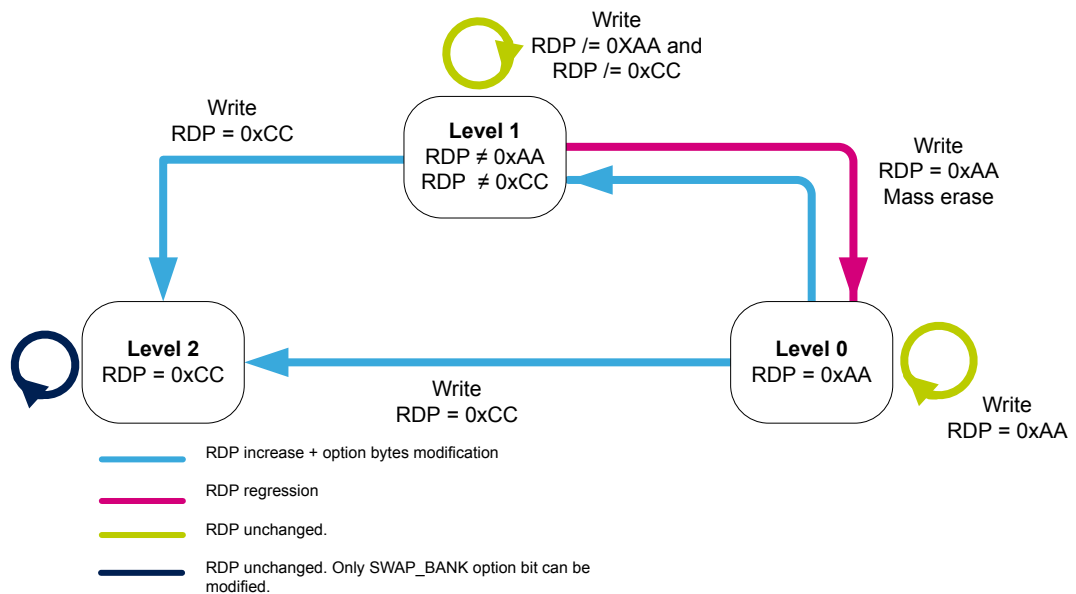
There are three readout protection levels as listed below:

- Level 0: all read/program/erase operations to and from the user Flash memory are allowed.
- Level 1: the Flash memory content is protected against debugger and potential malicious code stored in RAM.
- Level 2: all debug features are disabled, the boot from SRAM and from system memory are no longer available.

7.2 RDP level transition scheme when TrustZone® is disabled

The RDP level transition scheme when TZEN is cleared is illustrated in Figure 2.

Figure 2. RDP level transition scheme when TrustZone® is disabled (TZEN = 0)



7.3 Readout protection level when TrustZone® is enabled

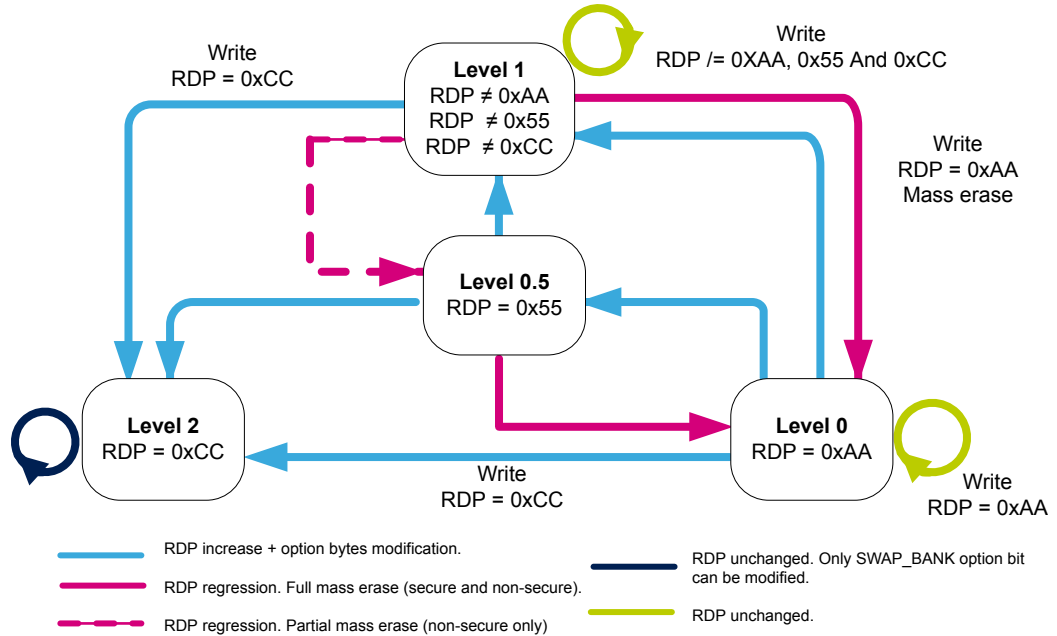
In addition to the RDP levels mentioned previously is set, there is a new RDP level named 0.5 that allows the following features:

- All read and write operations to / from the nonsecure flash memory are possible. The debug access to secure area is prohibited. Debug access to nonsecure area remains possible.
- Nonsecure debug mode: nonsecure debug is possible when the CPU is in nonsecure state.

7.4 RDP level transition scheme when TrustZone® is enabled

The RDP level transition scheme when TZEN is set is illustrated in Figure 3.

Figure 3. RDP level transition scheme when TrustZone® is disabled (TZEN = 1)



8 Starting with secure/nonsecure project

EWARM and MDK-ARM provide very similar approaches to support STM32L5, STM32U3, and STM32U5 series microcontrollers. It is done using two separate projects: secure and nonsecure.

- [Section 9](#) provides the MDK-ARM project instructions.
- [Section 10](#) provides the instructions for EWARM.
- [Section 11](#) provides the instructions for the CubeIDE.

Each section provides step by step instructions explaining the project setup of the secure and nonsecure parts using STM32L5/U5 series microcontrollers.

To begin with, use a template from STM32CubeL5 package (STM32Cube_FW_L5) that is composed from two sub-projects: one for the secure application part and the other for the nonsecure application part.

Before starting, the option bytes must be set using the STM32CubeProgrammer as detailed in the project readme.txt . This tool is available for download from www.st.com and illustrated in [Figure 4](#).

Figure 4. Configuration of option bytes using STM32CubeProgrammer

The screenshot displays the STM32CubeProgrammer interface. The main window shows the 'Option bytes' configuration page. The 'TZEN' option is checked. Below it, the 'SECWM1' and 'SECWM2' sections are visible, with their respective start and end addresses and values. The right sidebar shows the ST-LINK configuration and device information.

Name	Value	Address	Description
TZEN	<input checked="" type="checkbox"/>		Unchecked : Global TrustZone security disabled Checked : Global TrustZone security enabled Hide protection first area enable
HDP1EN	<input type="checkbox"/>		Unchecked : No HDP area 1 Checked : HDP first area is enabled
HDP1_PEND	Value: 0x0	Address: 0x8000000	End page of first hide protection area
HDP2EN	<input type="checkbox"/>		Unchecked : No HDP area 2 Checked : HDP second area is enabled
HDP2_PEND	Value: 0x0	Address: 0x8000000	End page of second hide protection area
NSBOOTADD0	Value: 0x10000	Address: 0x8000000	Non-secure Boot base address 0
NSBOOTADD1	Value: 0x17200	Address: 0x8090000	Non-secure Boot base address 1
SECBOOTADD0	Value: 0x18000	Address: 0xc000000	Secure boot base address 0
BOOT_LOCK	<input type="checkbox"/>		Unchecked : Boot based on the pad/option bit configuration Checked : Boot forced from base address memory
Secure Area 1			
Name	Value	Address	Description
SECWM1_PSTRT	Value: 0x0	Address: 0x8000000	Start page of first secure area
SECWM1_PEND	Value: 0x7f	Address: 0x8039800	End page of first secure area
Write Protection 1			
Secure Area 2			
Name	Value	Address	Description
SECWM2_PSTRT	Value: 0x1	Address: 0x8040800	Start page of second secure area
SECWM2_PEND	Value: 0x0	Address: 0x8040000	End page of second secure area

Log:

```

16:09:16 : OPTION BYTE PROGRAMMING VERIFICATION:
16:09:16 : Option Bytes successfully programmed
16:09:19 : Disconnected from device.
16:09:19 : ST-LINK SN : 00350016313751153333639
16:09:19 : ST-LINK FW : V33M2
16:09:19 : Disconnecting...
    
```

9 Using MDK-ARM for Cortex®-M33 with Trust Zone

The latest version of MDK-ARM (Keil®) is available for download from the official Arm® Keil® web site. MDK-ARM (Keil®) is installed by default in the "C:\Keil" directory on the PC local hard disk, the installer creates a start menu μVision® 5 shortcut.

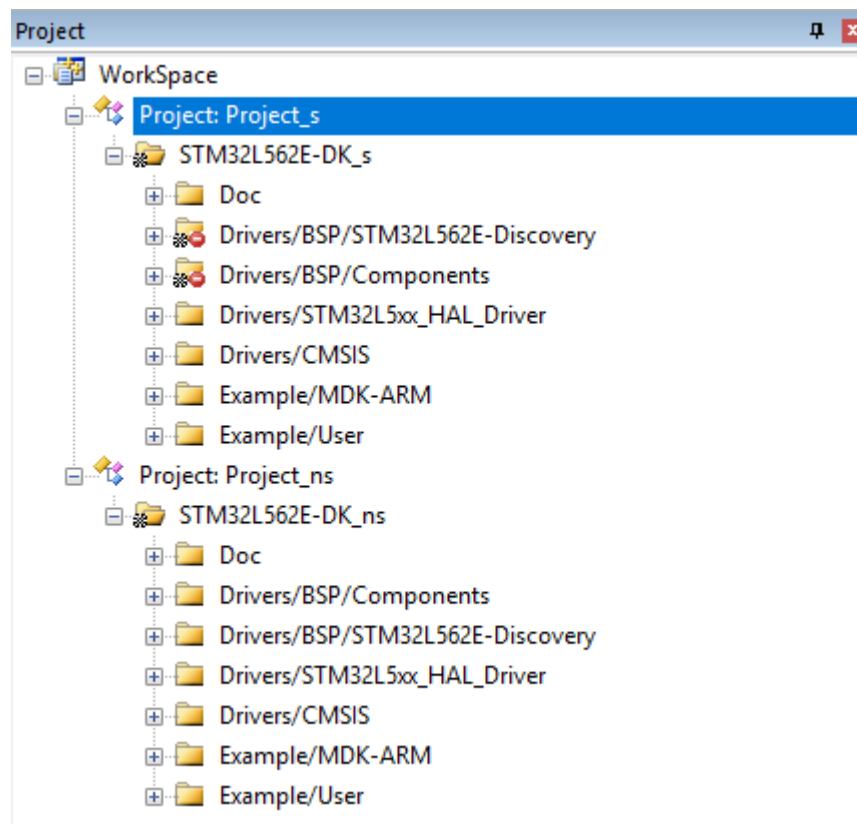
The MDK-ARM v5.27.0.0 and STM32L562-DK disco board are used for this section.

9.1 Secure project settings

This section outlines the secure project settings.

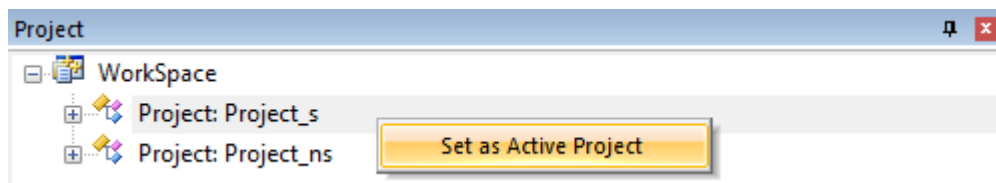
1. Open the multi-projects workspace file: "Project.uvmpw" that allows the user to work on both projects at the same time. The open project appears in the project explorer as illustrated in [Figure 5](#).

Figure 5. MDK-ARM project structure

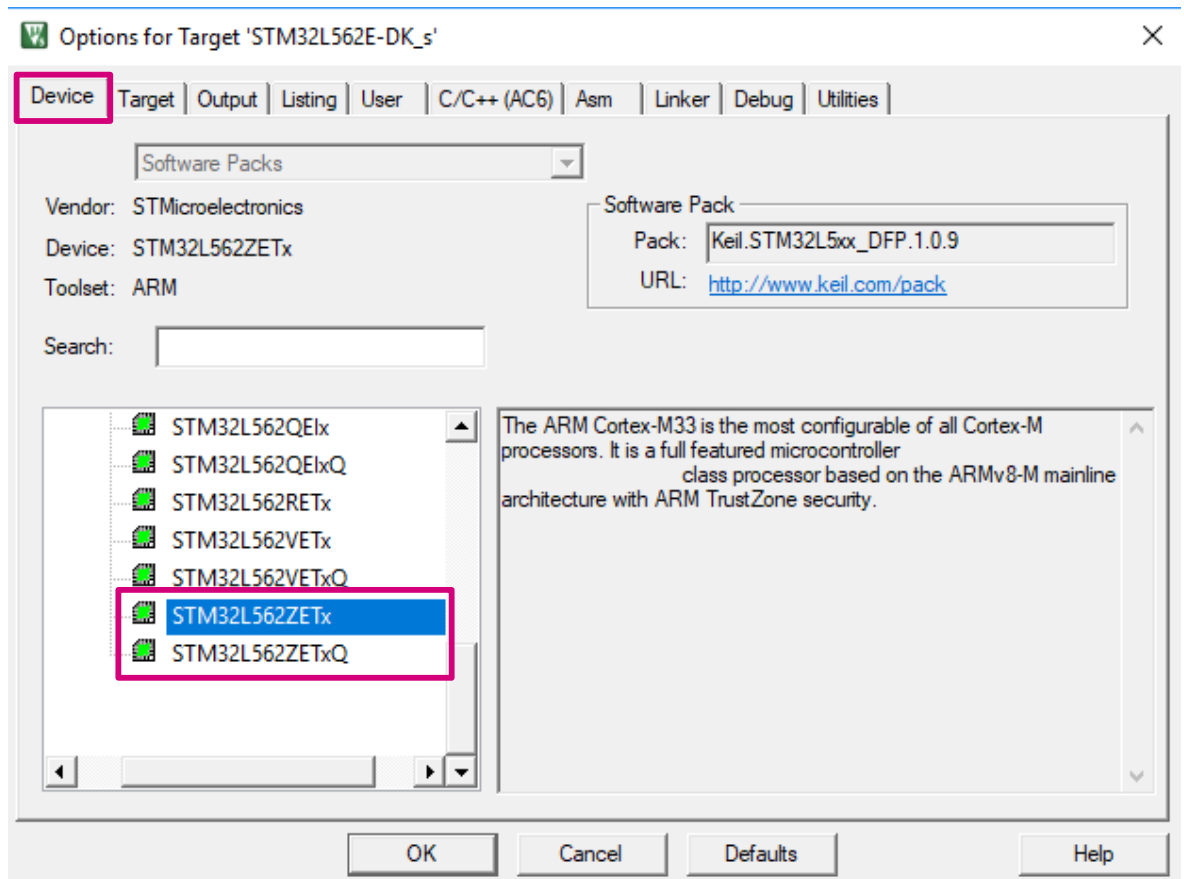


2. Set project_s as active project, see [Figure 6](#).

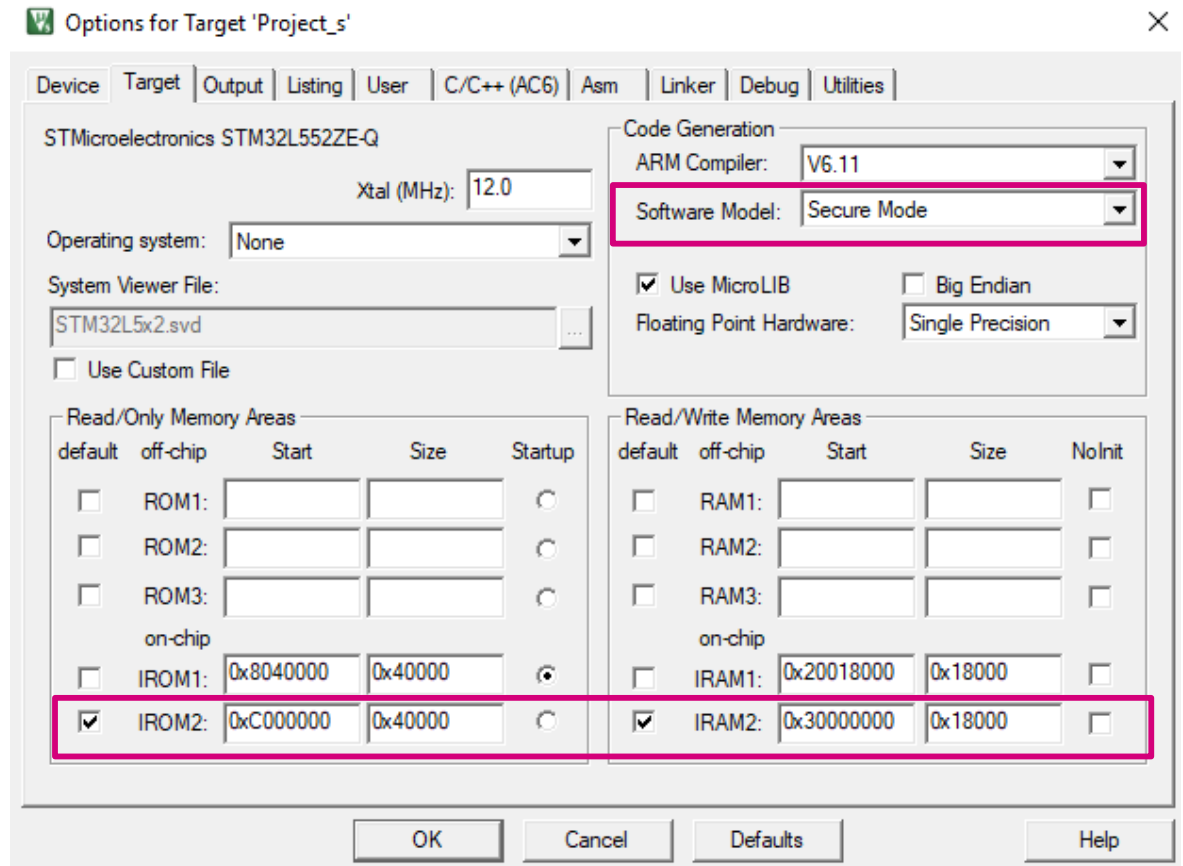
Figure 6. Secure project selection



3. Select the correct device by opening the configuration window and selecting: **Project / Options for Target / Device** then select the device from the list (see Figure 7).

Figure 7. Device selection


4. From **Project / Options for Target / Target / Code Generation** section, select the "Software Model" as "Secure". Ensure the right memory area is selected. See [Figure 8](#):
 - Secure Boot address : Flash at 0x0C000000 : secure Flash
 - Secure Boot address: SRAM1 at 0x30000000: secure SRAM

Figure 8. Project_s target options


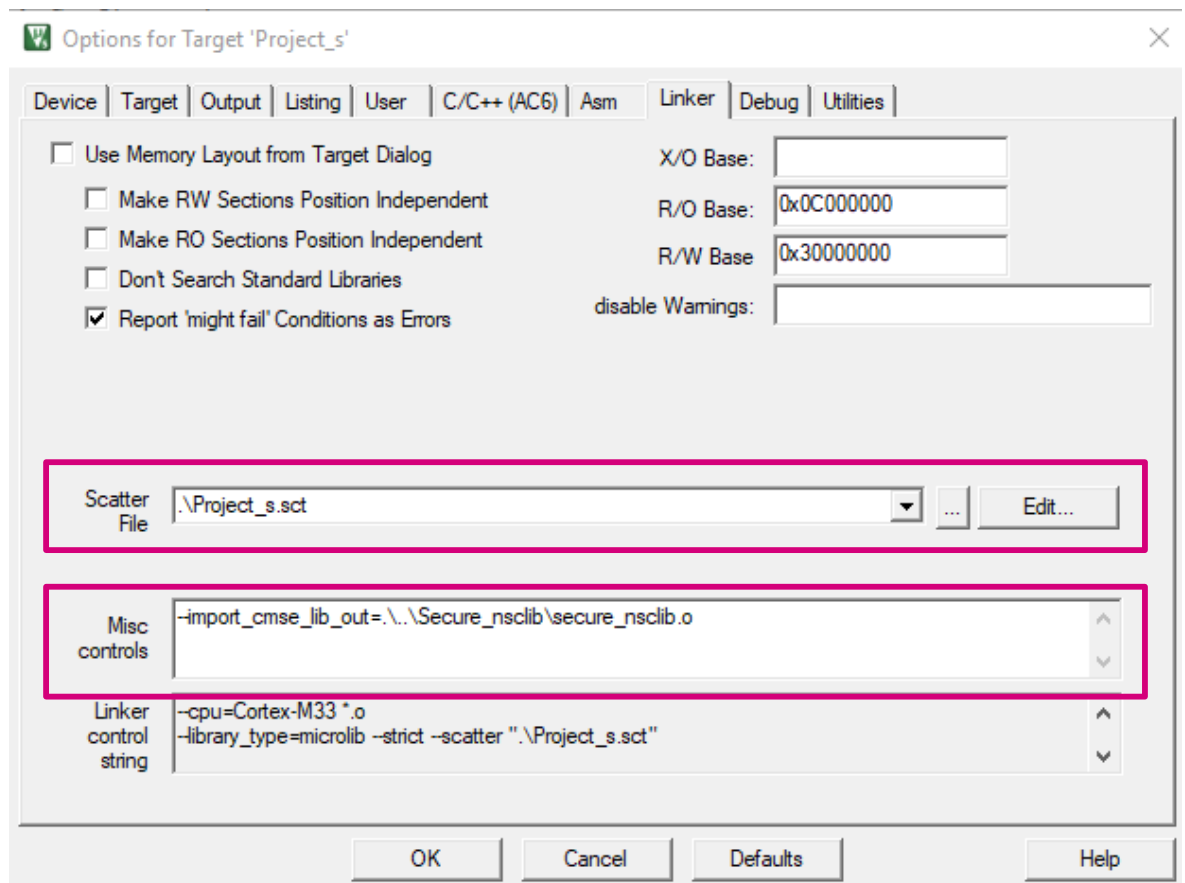
5. Ensure that the secure nonsecure callable functions (NSC) object file "secure_nsclib.o" is defined in **Project / Options for Target / Linker** under **Misc Controls** section.

Use the `[--import_cmse_lib_out ..\lib\nsclib_Secure.o]` command to create the output library: `nsclib_Secure.o`.

This file, automatically generated during the build of the secure project, contains all the nonsecure callable functions declared with the prefix: `__attribute__((cmse_nonsecure_entry))`.

See [Figure 9](#).

Figure 9. Project_s Linker configuration



Under scatter file section, check that this file contains the correct addresses as illustrated in [Figure 9](#).

This file is used by the Linker and determine how the memory layout is organized. A sample of a scatter file is given in [Figure 10](#).

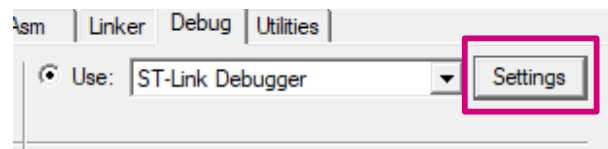
Figure 10. Scatter file sample

```

Project_s.ct
1 ; *****
2 ; *** Scatter-Loading Description File generated by uVision ***
3 ; *****
4
5 LR_IROM2 0x0C000000 0x00040000 { ; load region size_region
6   ER_IROM2 0x0C000000 0x0003E000 { ; load address = execution address
7     *.o (RESET, +First)
8     *(InRoot$$Sections)
9     .ANY (+RO)
10    .ANY (+XO)
11  }
12 RW_IRAM2 0x30000000 0x00018000 { ; RW data
13   .ANY (+RW +ZI)
14  }
15 }
16
17 LR_IROM3 0x0C03E000 0x00002000 { ; load region size_region
18   ER_IROM3 0x0C03E000 0x00002000 { ; load address = execution address
19     *(Veneer$$CMSE) ; check with partition.h
20  }
21 }
22
  
```

6. Select "ST-LINK Debugger" as the debugger from: **Project / Options for Target / Debug**. See Figure 11.

Figure 11. Target options debug



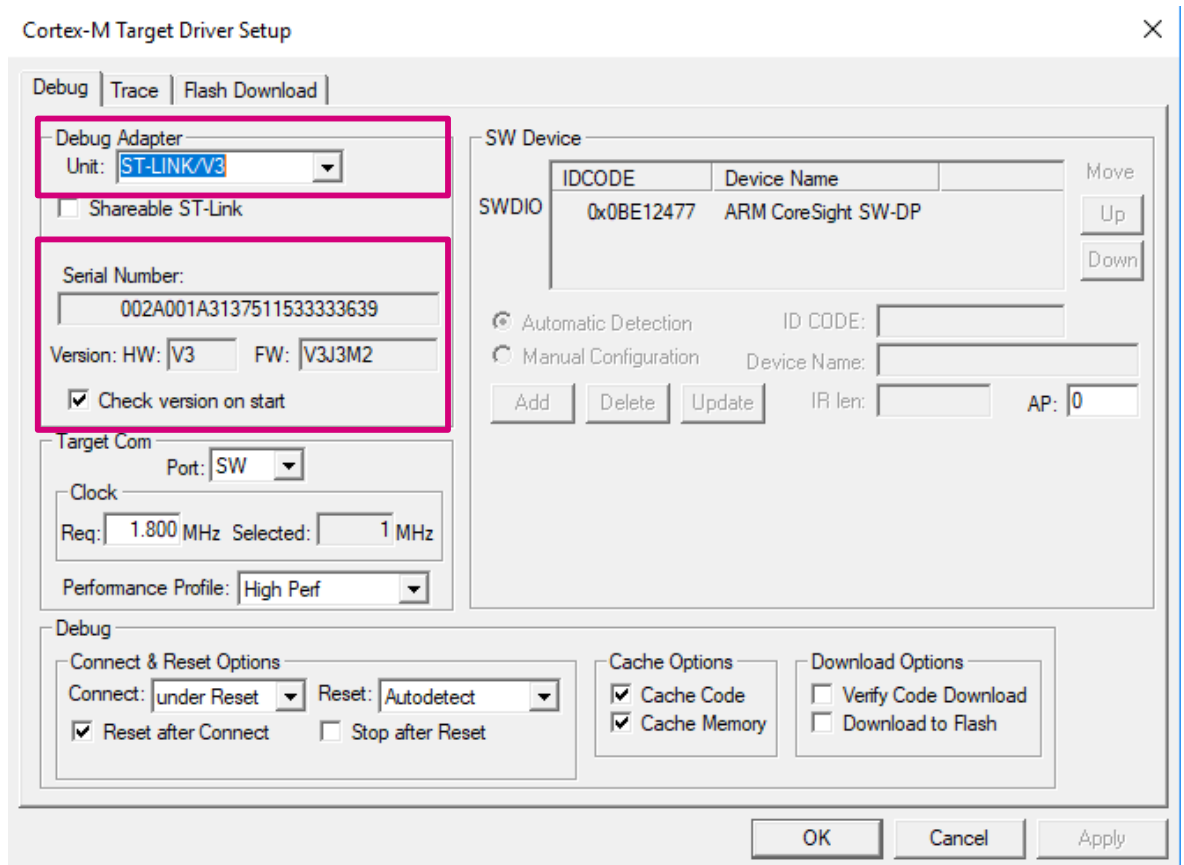
If "ST-LINK Debugger" does not appear in the list:

- a. Go to C:\Keil install directory
- b. Open TOOLS.INI file and apply the following changes:
 - i. Look for [ARMADS]:

All Armv8M based devices requires the processor SARMV8M.DLL. The TOOLS.INI file contains CPUDLL3 = SARMV8M.DLL (TDRV2, TDRV13, TDRV14, TDRV15, TDRV16).
The ST-Link driver is registered as TDRV6 in this example and could vary depending on the project: TDRV6=STLink\ST-LINKIII-KEIL_SWO.dll ("ST-Link Debugger").
 - ii. Add the TDRV6 to the list in CPUDLL3= SARMV8M.DLL:CPUDLL3 = SARMV8M.DLL (TDRV2, TDRV6, TDRV13, TDRV14, TDRV15, TDRV16).

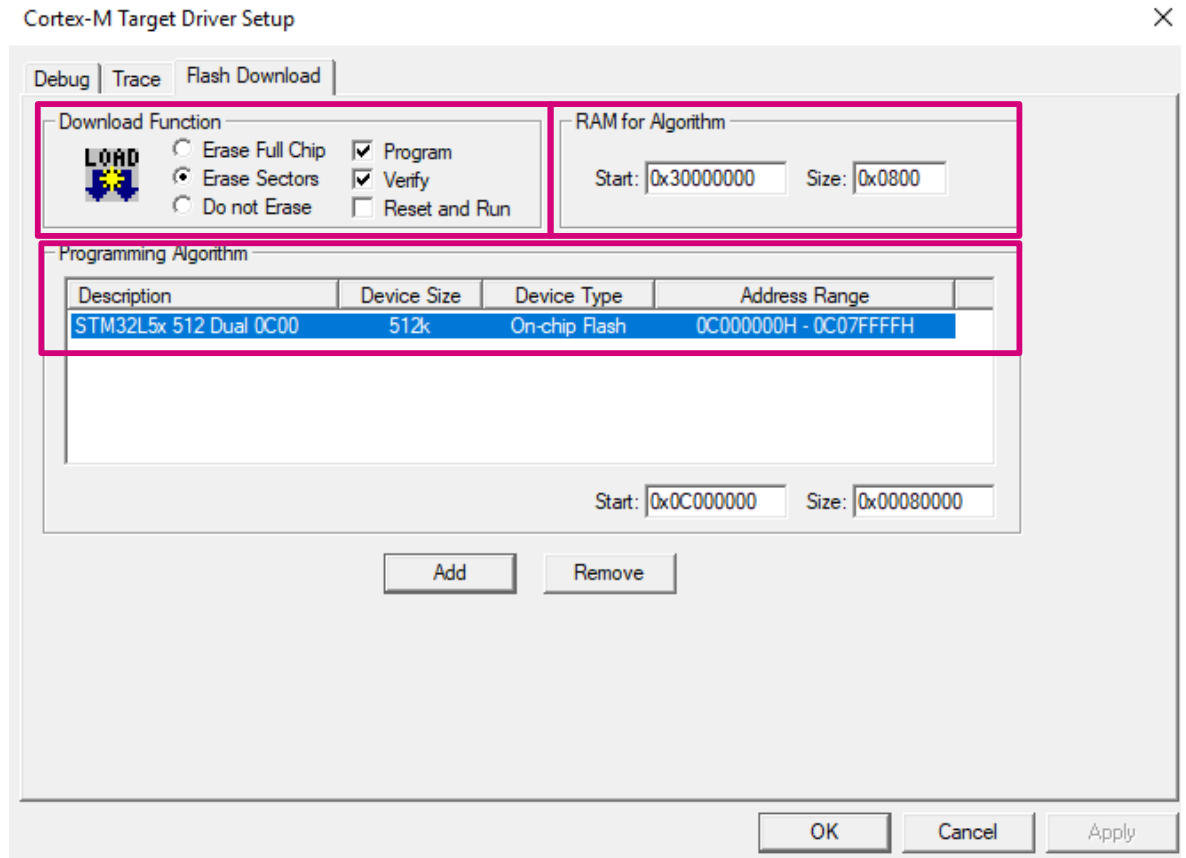
7. From "Debug" settings tab, ensure the debugger is connected as illustrated in Figure 12.

Figure 12. Debug configuration



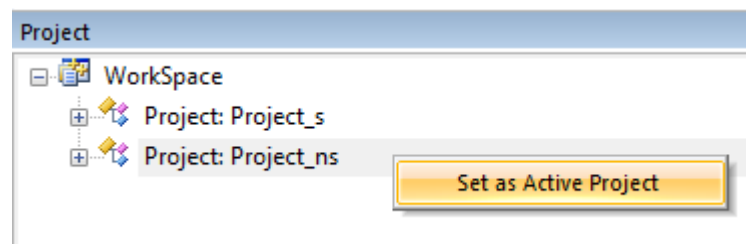
From the "Flash Download" tab, select the correct Flash-loader (see Figure 13):

- "Download Function": sets the Flash operations.
- RAM for algorithm: defines the address space where programming algorithms are loaded and executed. Usually, the address space is located in on-chip RAM.
- "Program Algorithm": contains the Flash programming definitions.

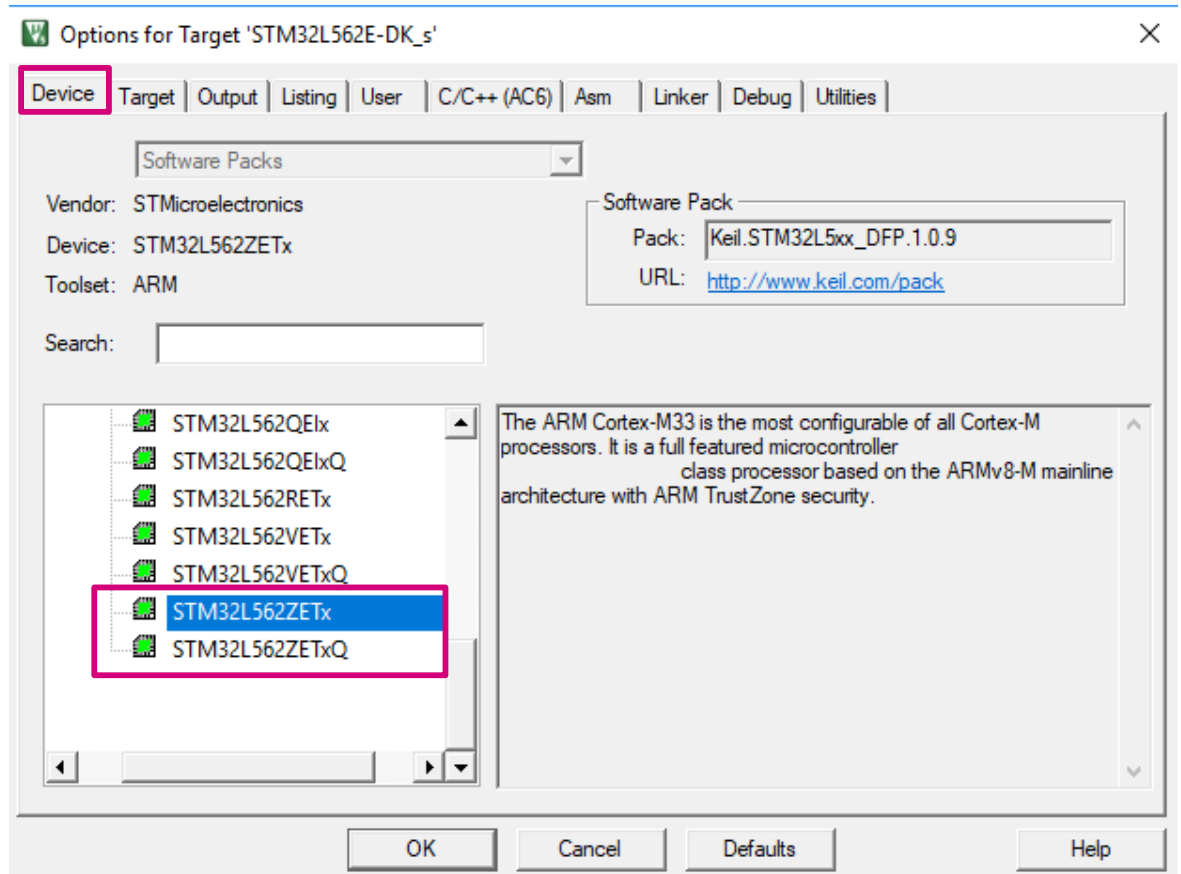
Figure 13. Flash-loader settings


9.2 Nonsecure project settings

1. Set project_ns as active project (see Figure 14).

Figure 14. Project_ns nonsecure project selection


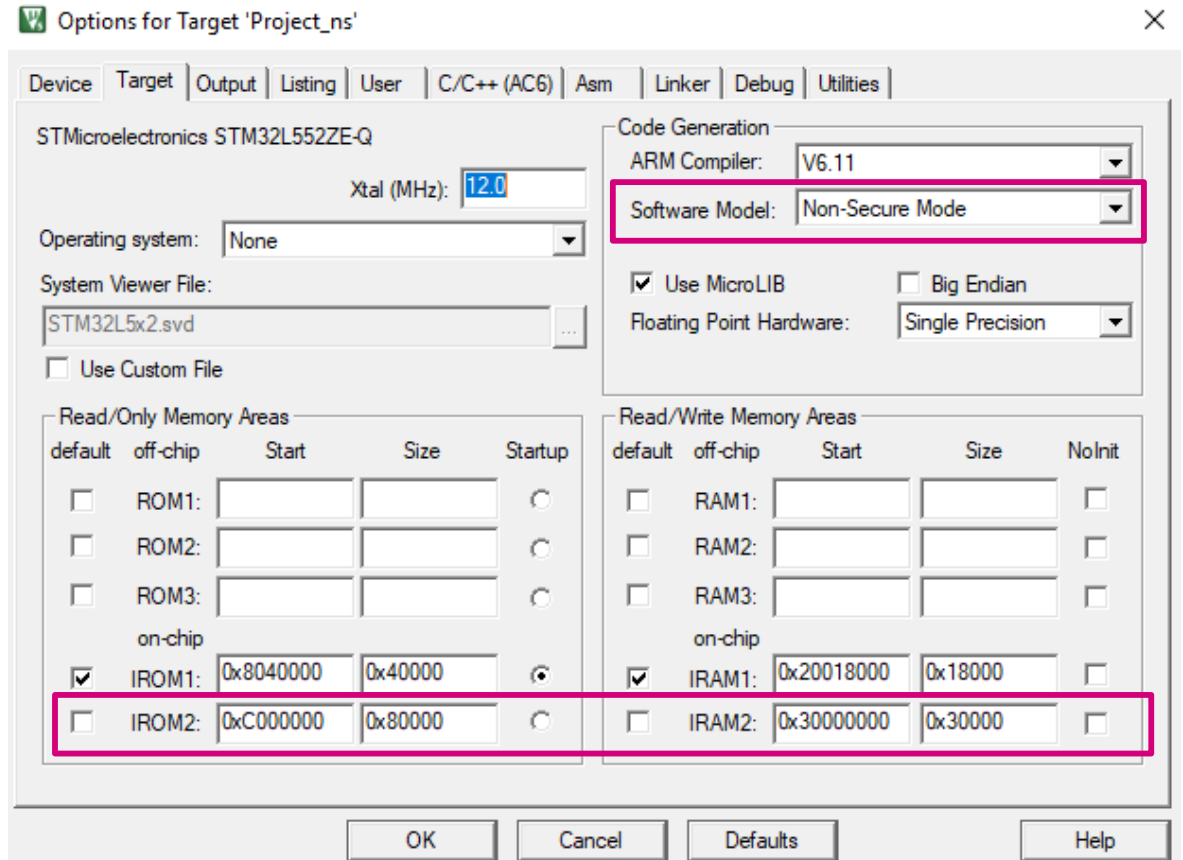
2. Select the correct device by opening the configuration window: Select **Project / Options for Target** (see Figure 15).

Figure 15. Device selection


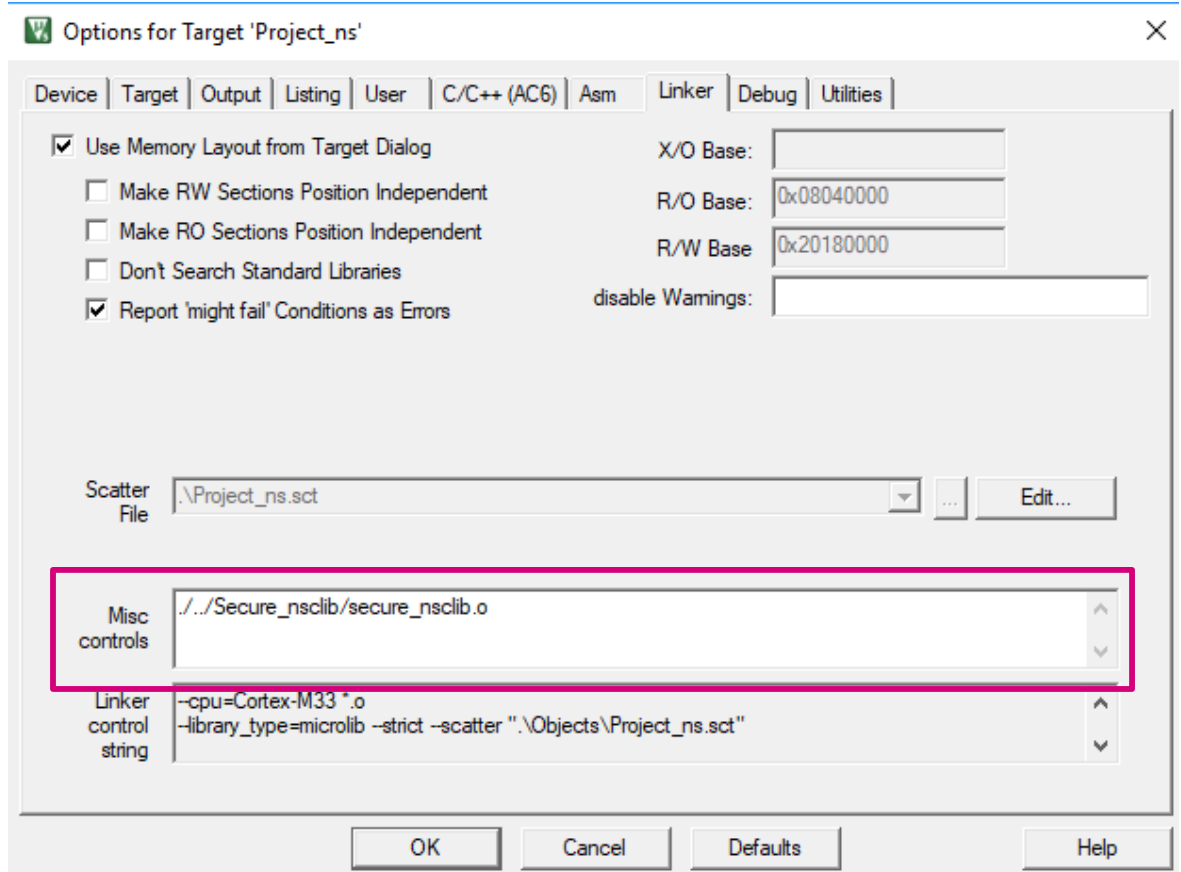
3. Ensure the right memory area is selected from **Project / Options for Target / Target:**

- Boot address 0: Flash at 0x08040000: nonsecure flash
- Boot address 1: SRAM at 0x20018000: nonsecure SRAM

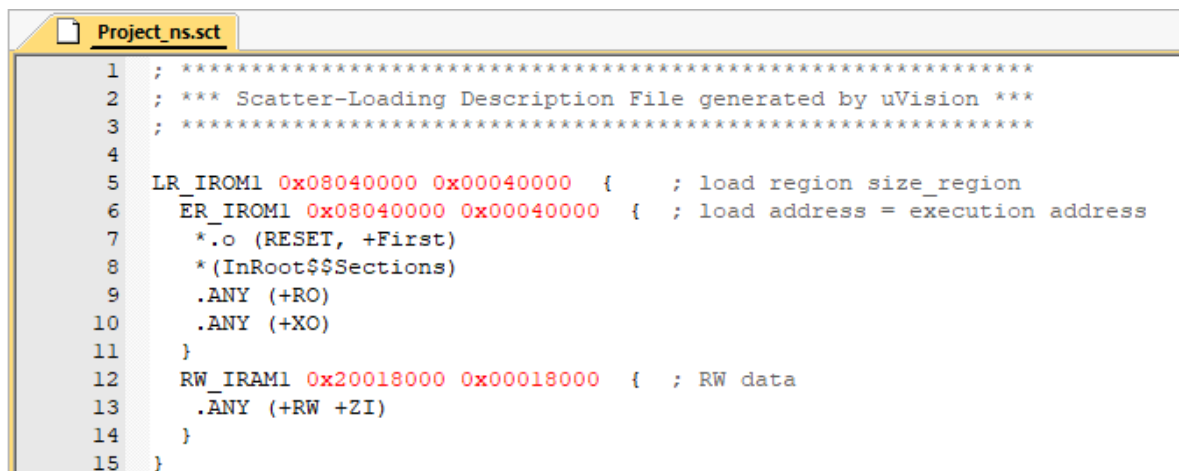
The software model must be set in nonsecure mode (see Figure 16).

Figure 16. Memory configuration


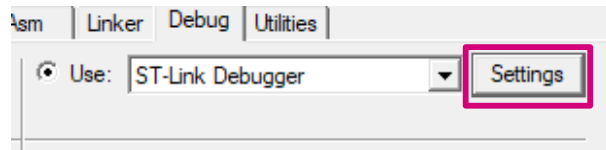
4. Add the import library from the secure project: this file is automatically included at link time in the nonsecure project. It allows the nonsecure part to call functions from the secure part (see Figure 17).

Figure 17. Linker options


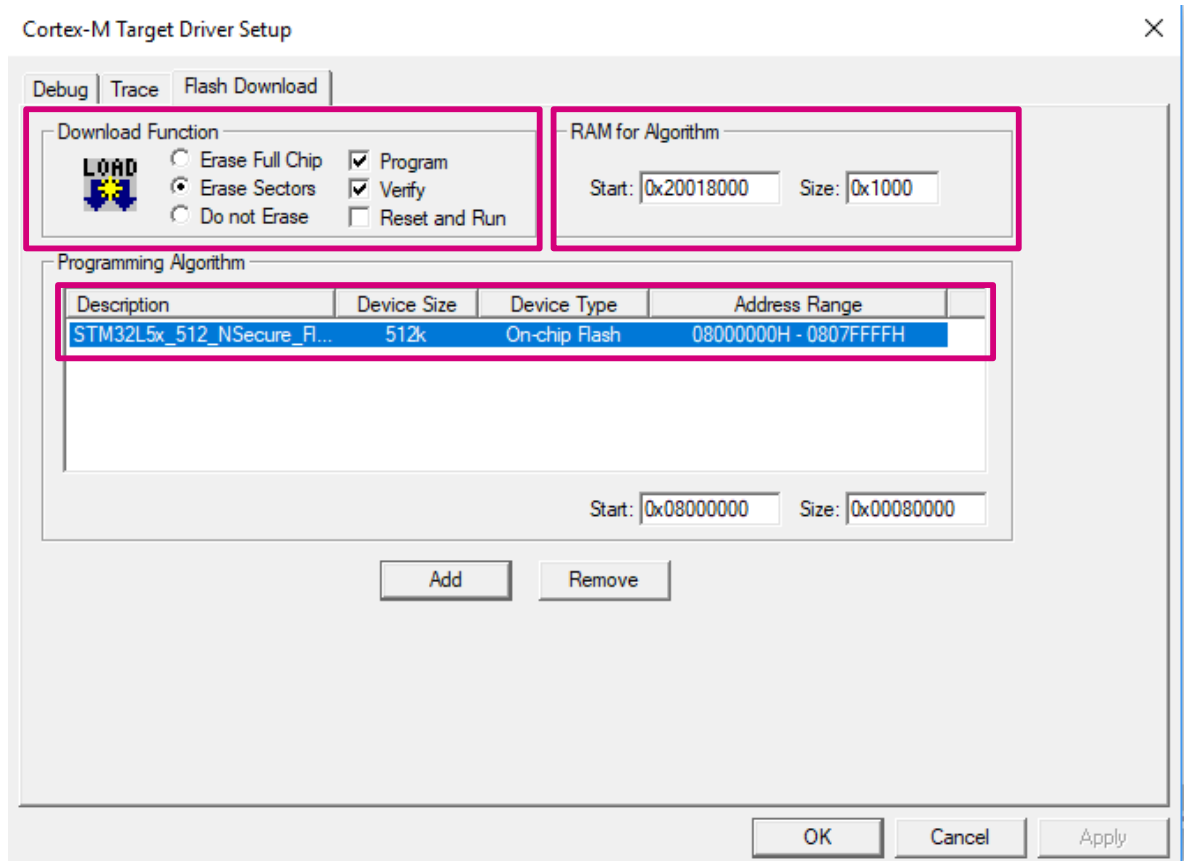
Under scatter file section, check that this file contains the correct addresses as shown in Figure 17. This file is used by the Linker and determines how the memory layout is organized. A sample of a scatter file is given in Figure 18.

Figure 18. Scatter file sample


5. Select "ST-LINK debugger" from **Project / Options for Target / Debug** (see Figure 19).

Figure 19. Debug settings


6. From **Debug settings / Flash Download** window (see Figure 20) select:
- Download function: sets the flash operations
 - RAM for algorithm: defines the address space where programming algorithms are loaded and executed. Usually, the address space is located in the embedded RAM.
 - Program algorithm: contains the definitions for programming flash.

Figure 20. FlashLoader configuration


9.2.1 Building a project

It is now possible to build both projects at the same time. From **Project / Batch Setup** (see [Figure 21](#) and [Figure 22](#)) or from the icon available from the menu bar go to the batch setup menu and select both projects.

Note: The secure project must be built first in order to create the import library for the nonsecure project. In order to build the secure project before the nonsecure one, it must be first in order.

Figure 21. Project batch setup

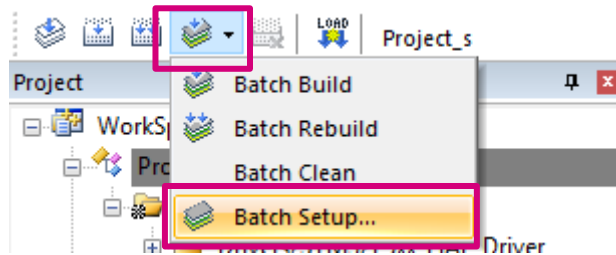
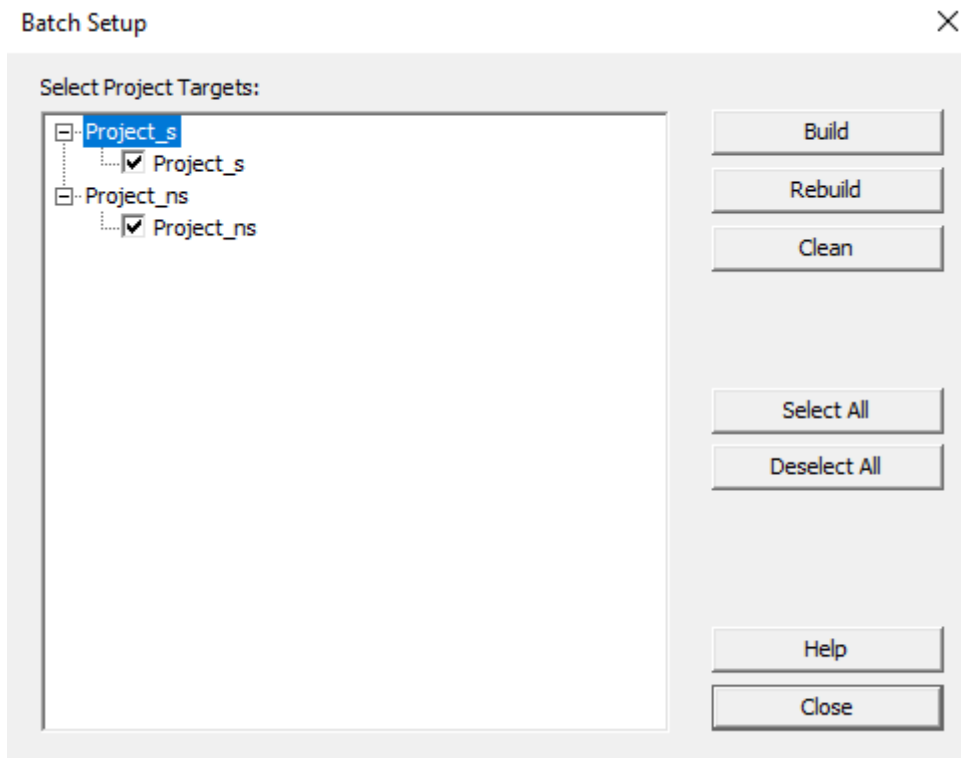
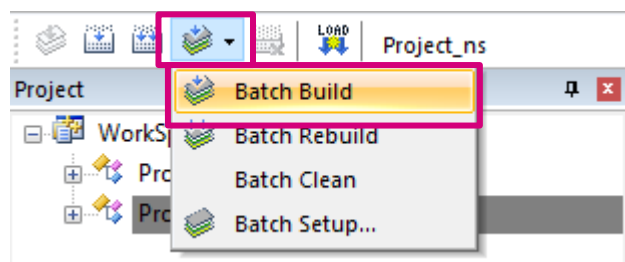


Figure 22. Project build ordering



Then, from the same menu, click on "Batch Build" to build both projects (see [Figure 23](#)).

Figure 23. Build both projects in one step



9.3 Execute from secure code to nonsecure code

Before downloading the projects, a connection to the STM32L562E-DK Discovery board must be made as follows:

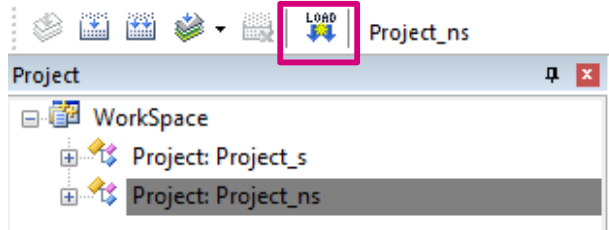
1. Connect the ST-LINKV3 programming and debugging tool on the Discovery board by plugging the USB cable to the board CN17 (ST-LINK USB connector). LD3 illuminates in red when the ST-LINKV3 is connected as illustrated in Figure 24.

Figure 24. STM32L562E-DK Discovery board in connected status



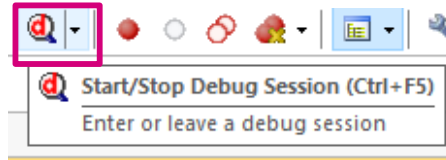
2. Select the **Project_ns** project as the active project then load the nonsecure binary code. Select the **Project_s** project as the active project then load the secure binary code. This is illustrated in Figure 25.

Figure 25. Load the nonsecure binary



3. Start a debug session by clicking the “Download and Debug” button in the toolbar illustrated in Figure 26.

Figure 26. Download and debug button



Note: The system always boots in secure code (main.c) at first and the secure application then launches the nonsecure application as illustrated in below.

Figure 27. Main.c sample code

```

52 | /*
53 | int main(void)
54 |
55 | /* Secure/Non-secure Memory and Peripheral isolation configuration */
56 | SystemIsolation_Config();
57 |
58 | /* Enable SecureFault handler (Default is default) */
59 | SCB->SHCSR |= SCB_SHCSR_SECUREFAULTEN_Msk;
60 |
61 | /* STM32L5xx **SECURE** HAL library initialization:
62 | - Secure SysTick timer is configured by default as 1000ms of time base,
63 |   but user can eventually implement his proper time base source (a general
64 |   purpose timer for example or other time source), keeping in mind that
65 |   Time base duration should be kept 1ms since PPP_TIMEOUT_VALUEs are defined
66 |   and handled in milliseconds basis.
67 | - Low Level Initialization
68 | */
69 | HAL_Init();
70 |
71 | /* Enable Instruction cache (default 2-ways set associative cache) */
72 | if(HAL_ICACHE_Enable() != HAL_OK)
73 | {
74 |     /* Initialization Error */
75 |     while(1);
76 | }
77 |
78 | /* Secure application may configure the System clock to have a frequency of 110 MHz */
79 | /* SystemClock_Config(); */
80 |
81 |
82 | /* Add your secure application code here prior to non-secure initialization
83 | */
84 |
85 | /****** Setup and jump to non-secure *****/
86 |
87 | NonSecure_main();
88 |
89 |
90 | /* Non-secure software does not return, this code is not executed */
91 | while (1) {
92 | }

```

At the end of secure function, the system switches from the secure state to the nonsecure state (see Figure 28).

Figure 28. Code switch to nonsecure code status

The screenshot shows the Keil MDK-ARM interface. On the left, the 'Registers' window is open, showing the 'Non-Secure' mode selected under the 'Internal' section. The 'Disassembly' window shows the assembly code for 'main.c'. A yellow highlight is placed on the instruction at address 0x0804101C: `0x0804101C E7FE B 0x0804101C`. The code includes comments for HAL initialization, secure error handling, and system clock configuration.

The secure status is provided from the status bar at the bottom of Keil® interface as illustrated in Figure 29.

Figure 29. CPU status

The screenshot shows the bottom status bar of the Keil MDK-ARM interface. The status bar contains the following information: 'ST-Link Debugger', 'Debug: Secure', 'CPU: Non-Secure', 't1: 0.00000000 sec', 'L50 C1', and 'CAP_NUM SCRL OVR R/W'. A red box highlights the 'Debug: Secure' and 'CPU: Non-Secure' text.

10 Using EWARM for Cortex M33 with TrustZone®

The latest version of IAR Embedded Workbench for Arm® (EWARM) is available to download from the official web site of IAR System.

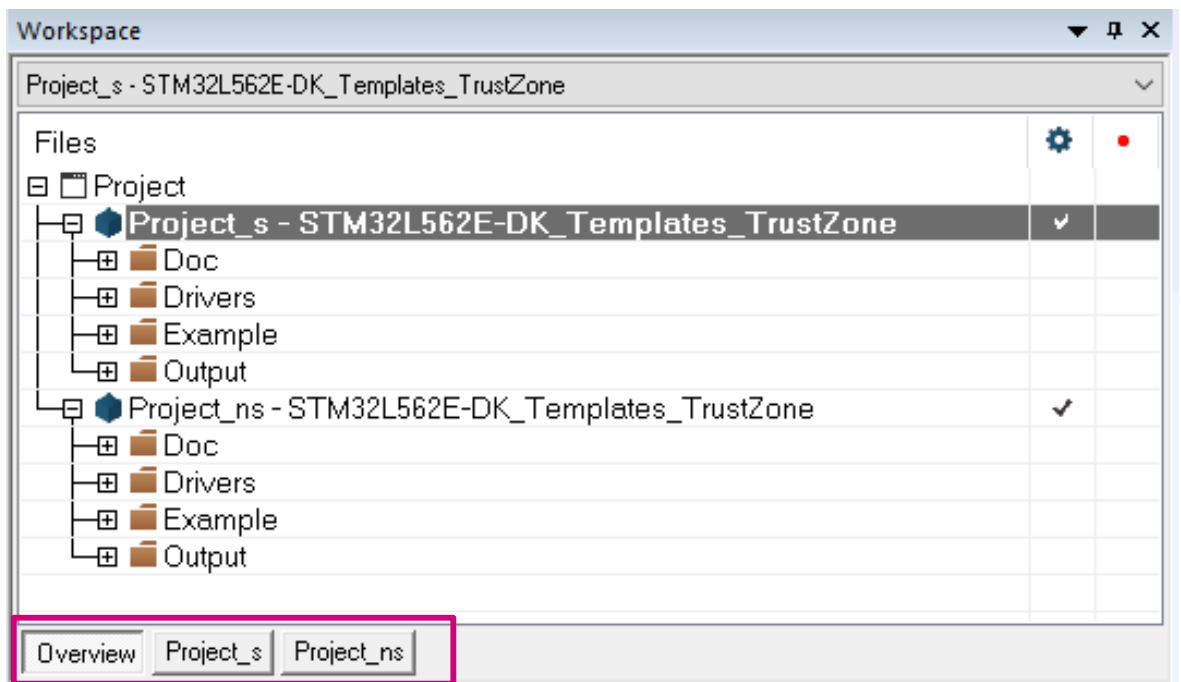
This part uses EWARM v8.40.1 and STM32L562-DK disco board.

10.1 Secure project settings

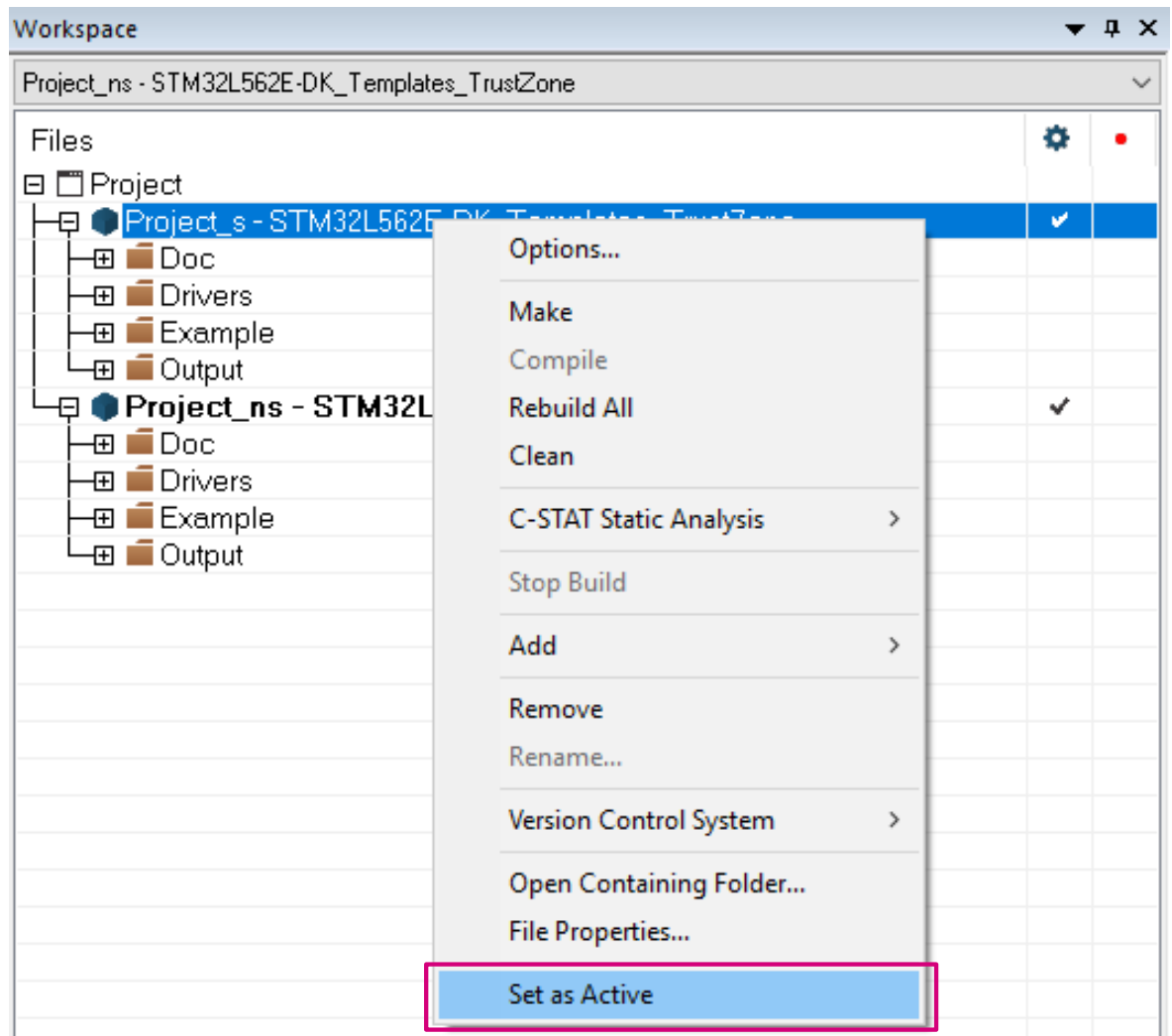
To configure a secure project, the first step is to open "Multi-projects" workspace file: Project.eww that allows the user to work on both projects at the same time.

1. The open project appears in the project explorer view illustrated in Figure 30.

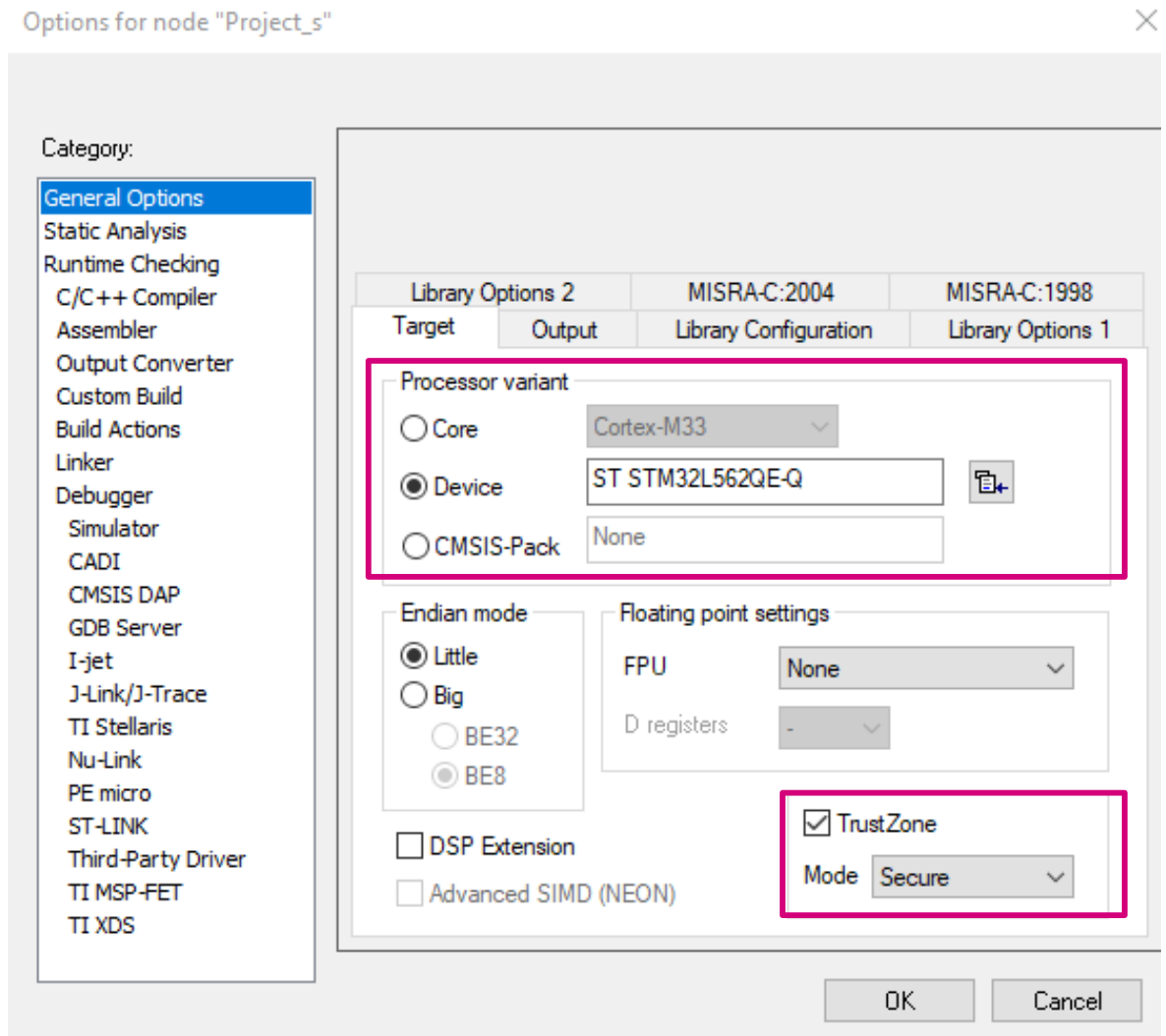
Figure 30. EWARM v8.40.1 project explorer view



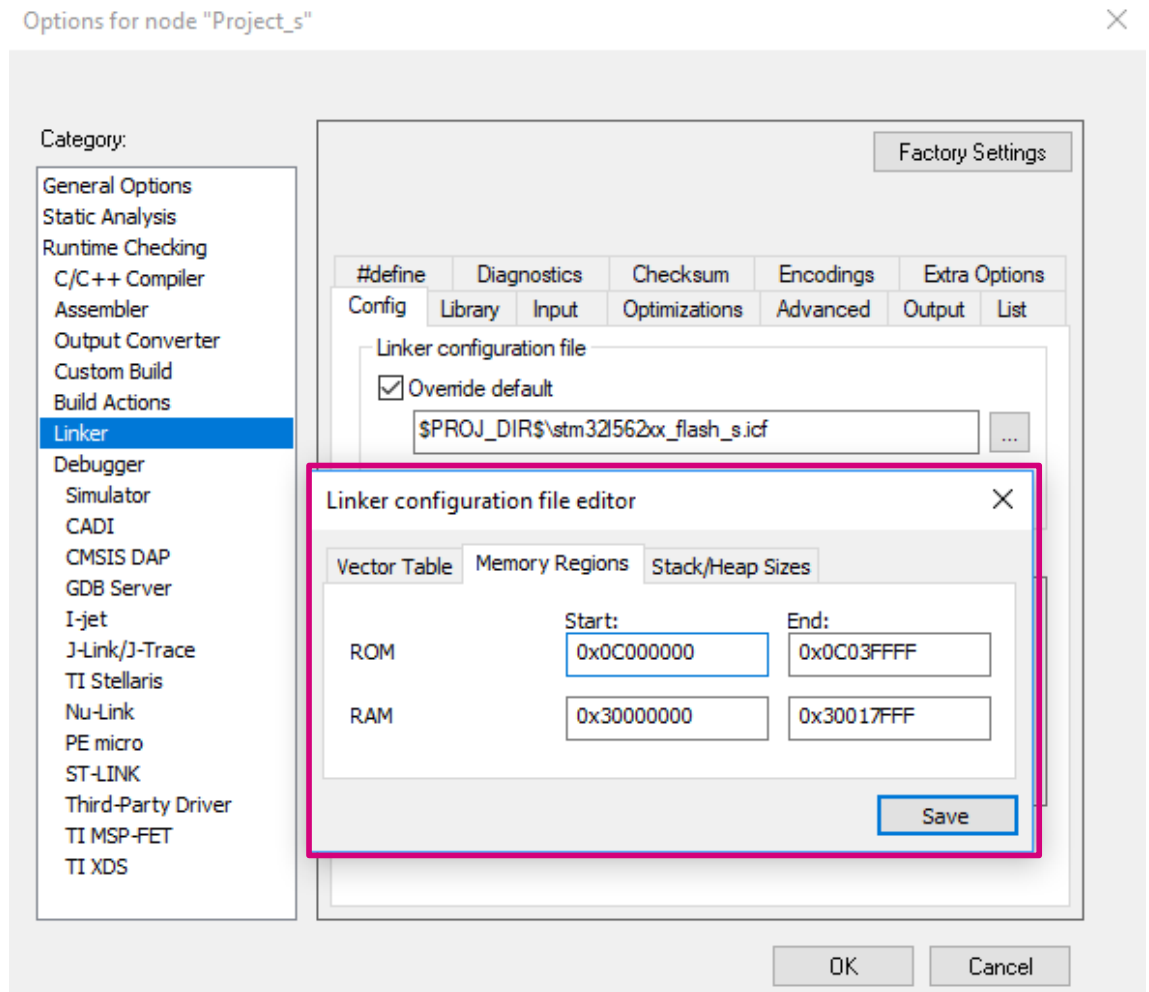
- Set project_s-STM32L562E-DK_Templates_TrustZone as active project as illustrated in Figure 31.

Figure 31. Setting the project to active status


- Open the configuration window by selecting **Project-s / Options / General Options** and select the correct device from "Processor variant" section.
From "TrustZone" section, ensure that the mode selected is "Secure" and "TrustZone" checkbox is checked as shown in Figure 32.

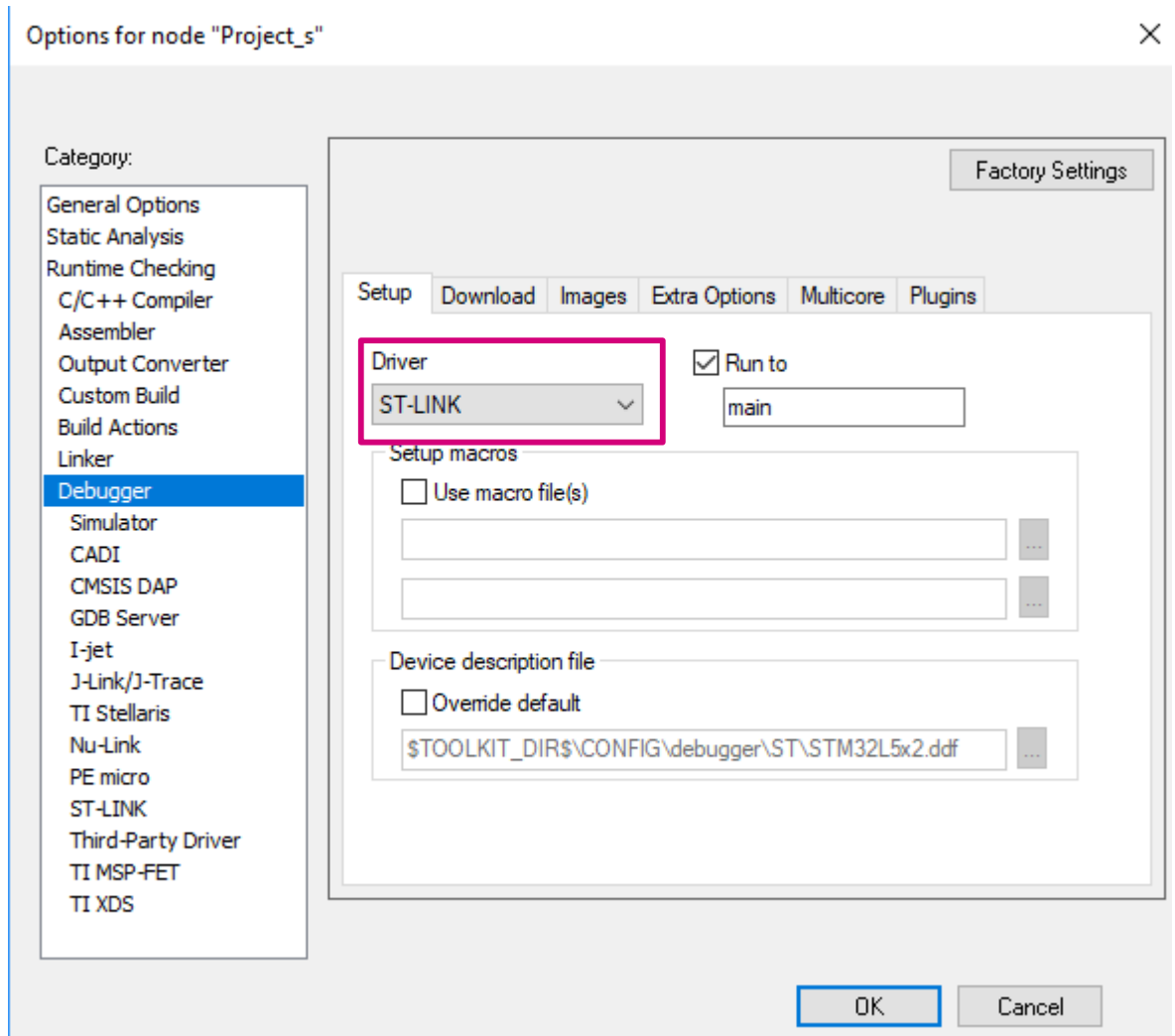
Figure 32. Device selection


4. From **Project-s / Options / Linker / Config** "Linker configuration file editor" section (see Figure 33):
 - a. Click Edit to display the linker configuration file editor.
 - b. Check the linker configuration file to make sure that the application has been linked to the right address:
 - Secure boot address : Flash at 0x0C000000 for the secure flash
 - Secure boot address: SRAM1 at 0x30000000 for the secure SRAM

Figure 33. Linker configuration


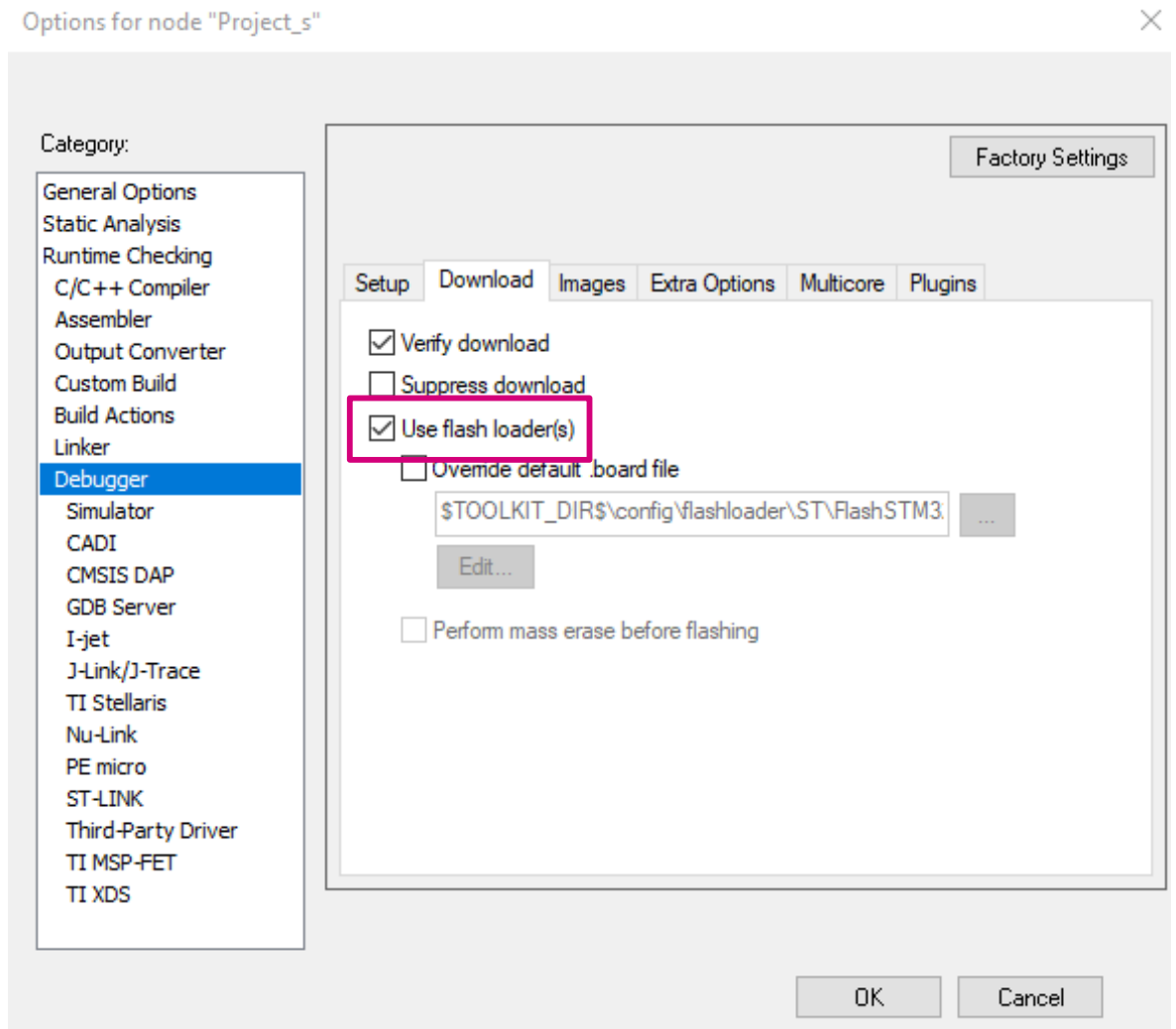
This .icf file contains all the information required by the linker.

- Open the debugger tab from: **Project / Options / Debugger**. From setup section, select ST-LINK as a debugger in the driver field (see Figure 34).

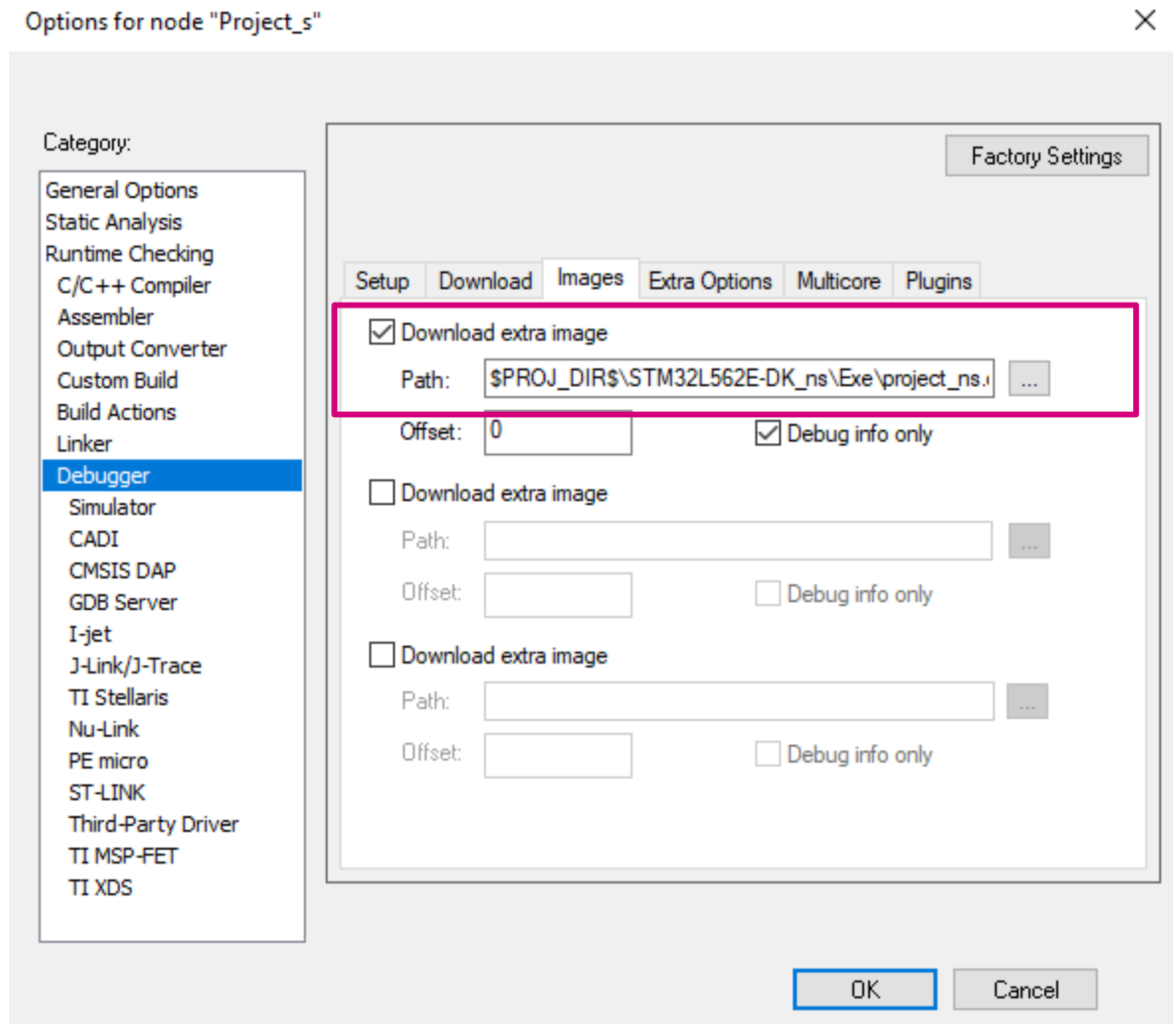
Figure 34. Project debugger setup


6. From the "Download" tab, ensure that "Use flash loader" is checked (see Figure 35).

Figure 35. FlashLoader selection

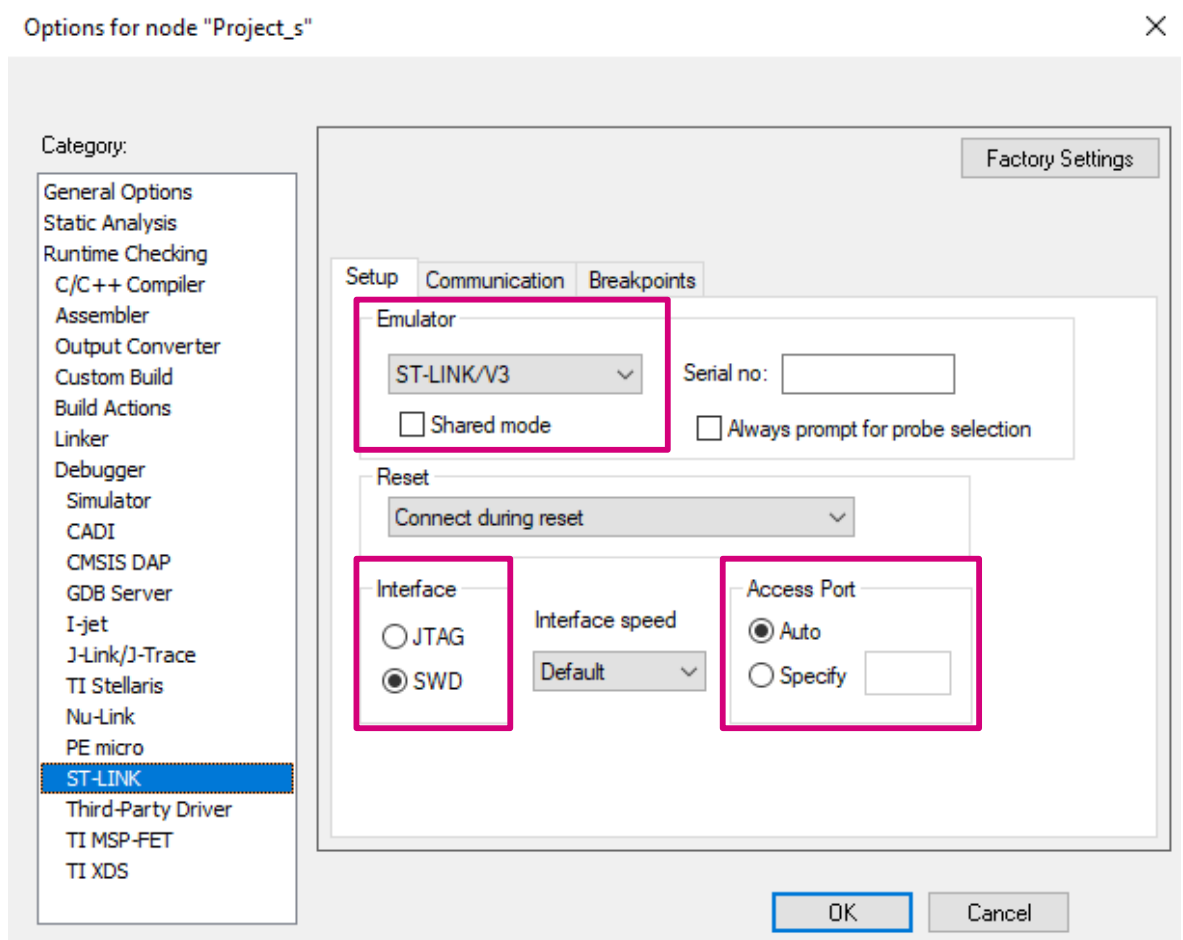


- The secure project must specify the nonsecure project output file as an extra image that must be loaded by the debugger. To do this, use: **Project / Options / Debugger / Images** and check the "Download extra image" check box (see Figure 36).

Figure 36. Selecting the nonsecure output file as an extra image


Debug info causes the debugger to only download debug information, and not the complete debug file.

8. From **Project / Options / ST-LINK "Setup"** tab, see [Figure 37](#):
 - Select the "ST-LINK debugger".
 - Select the reset type:
 - System reset: resets the core and peripherals.
 - Core reset: resets the core via the VECTRESET bit; the peripheral units are not affected.
 - Software reset: sets PC to the program entry address.
 - Hardware reset: the probe toggles the nSRST/nRESET line on the JTAG connector to reset the device. This reset usually resets the peripheral units also.
 - Connect during reset: ST-LINK connects to the target while keeping Reset active. Reset is pulled low and remains low while connecting to the target.
 - Select the communication interface:
 - JTAG: to use the JTAG interface.
 - SWD: to uses the SWO interface, which uses fewer pins than JTAG. Select SWD if the serial-wire output (SWO) communication channel is to be used.
 - Select the Access Port:
 - Auto: automatically uses the access port 0 for Cortex®-M33.
 - Manually: specify the access port to be used.

Figure 37. Project setup


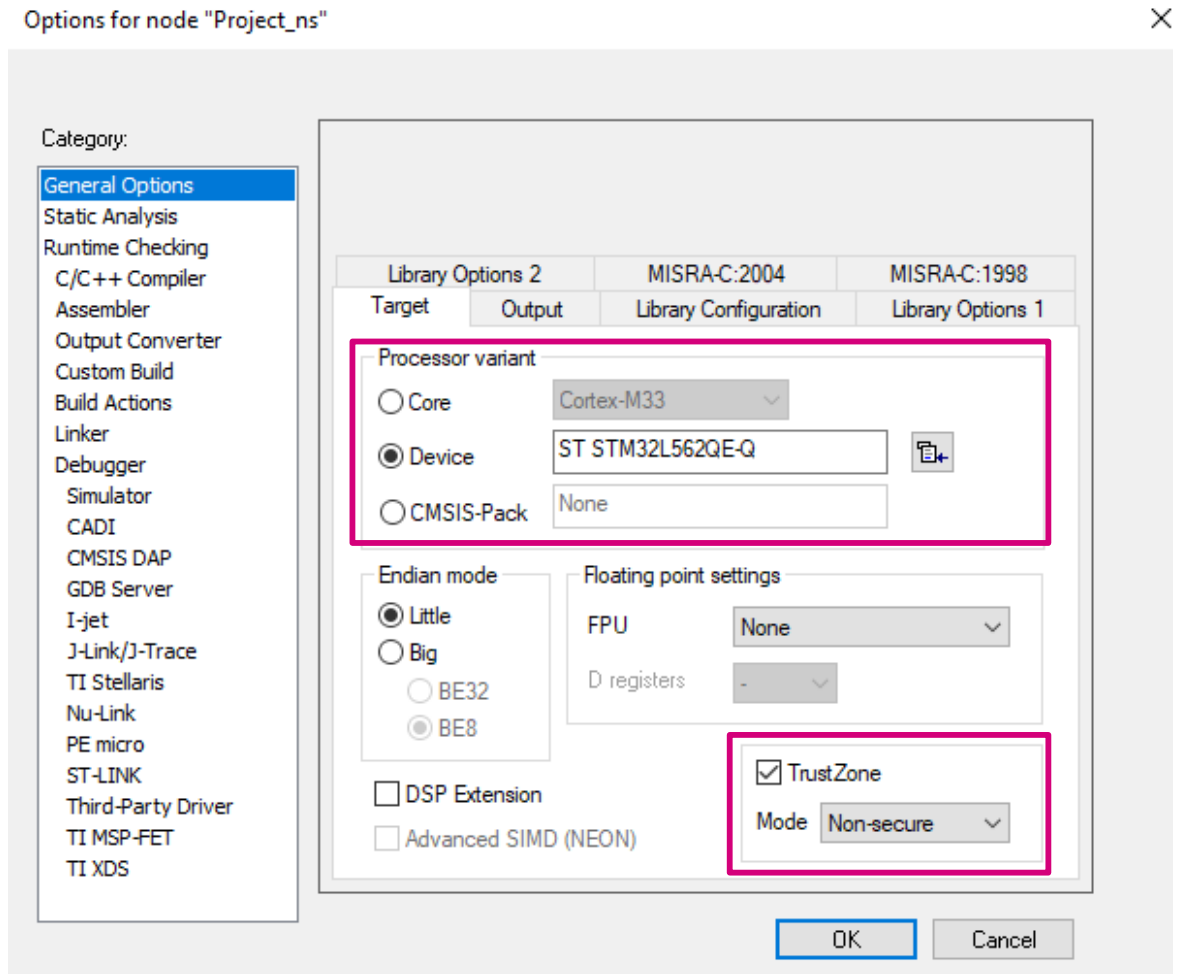
10.2 Nonsecure project settings

Set project_s-STM32L562E-DK_Templates_TrustZone as active project

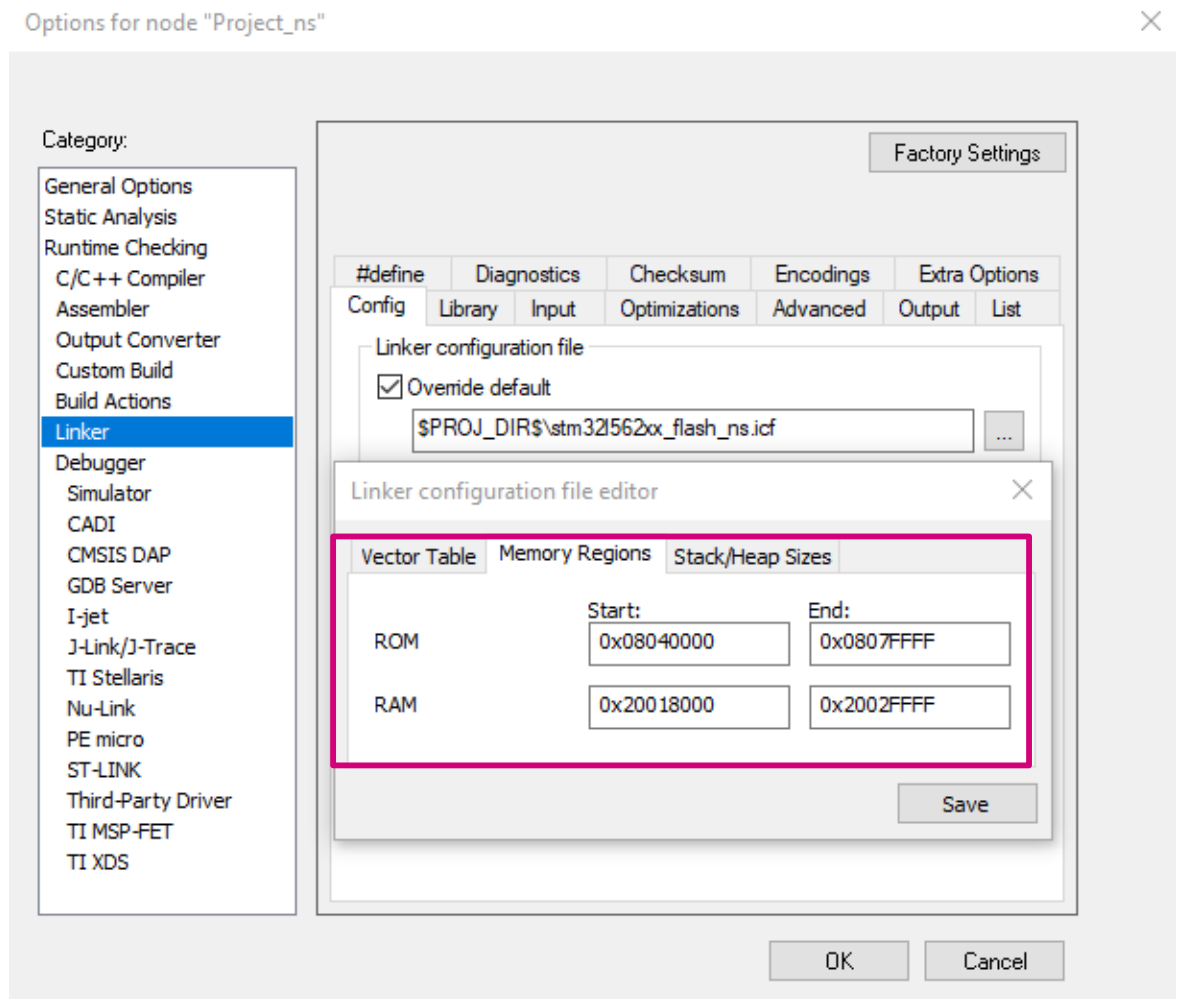
1. Open the configuration window by selecting **Project-s / Options/ General Options**. In the "Target" tab, select the correct device from processor variant section (see Figure 38).

From the TrustZone® section, ensure that the "Nonsecure" mode is selected and the TrustZone® box is checked.

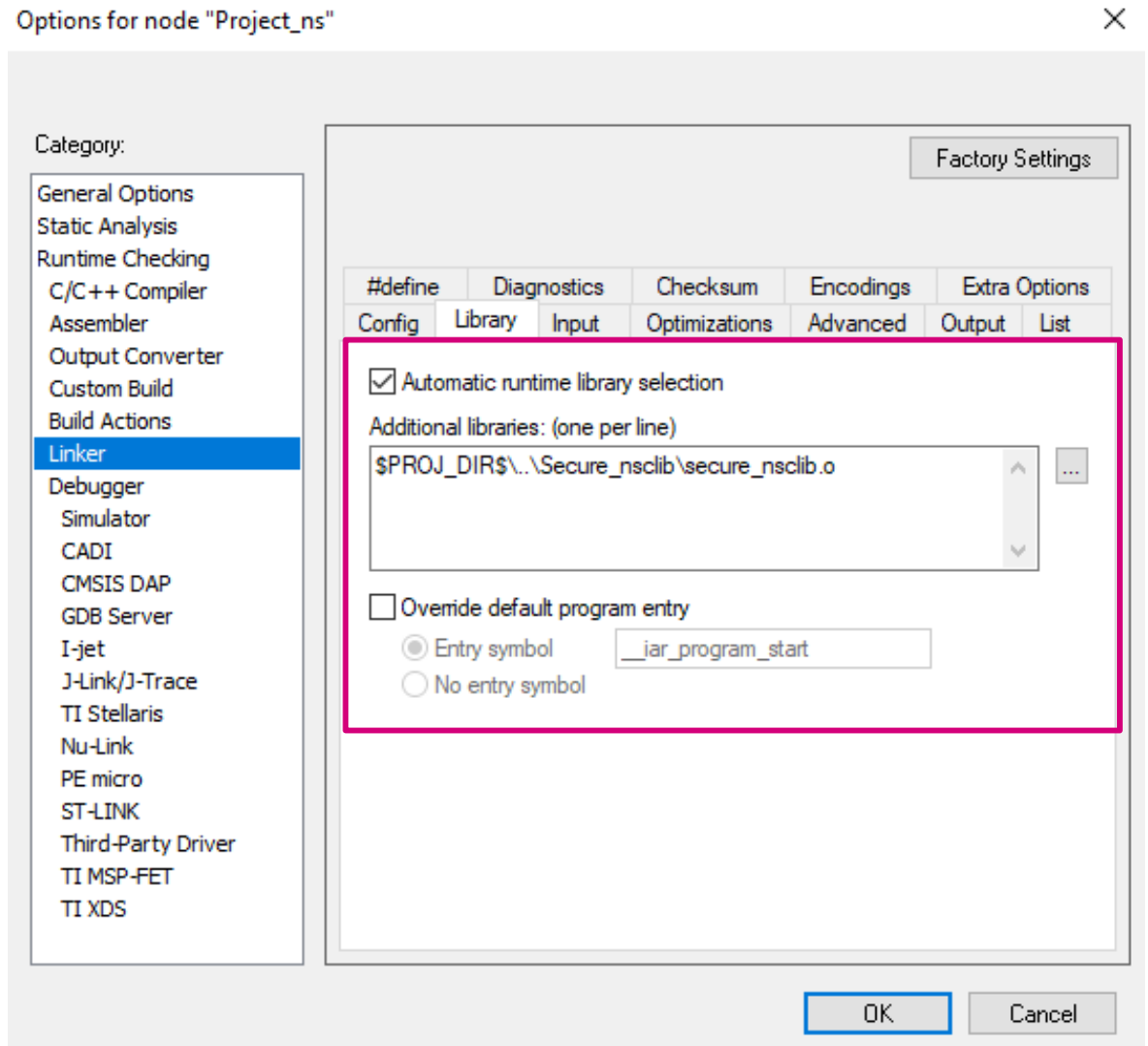
Figure 38. Project set up: general options



2. From **Project-s / Options / Linker / Linker configuration** file section (see Figure 39):
 - Click edit to display the linker configuration file editor.
 - Check the linker configuration file to make sure that the application has been linked to the right address:
 - Boot address 0: Flash at 0x08040000 (nonsecure flash)
 - Boot address 1: SRAM at 0x20018000 (nonsecure SRAM).

Figure 39. Project linker configuration


- From **Project-s / Options / Linker** in the "Library" (see Figure 40).
Add the imported library from the secure project. This file is automatically included at link time in the nonsecure project. It allows the nonsecure part to call functions of the secure part.

Figure 40. Linker library setup


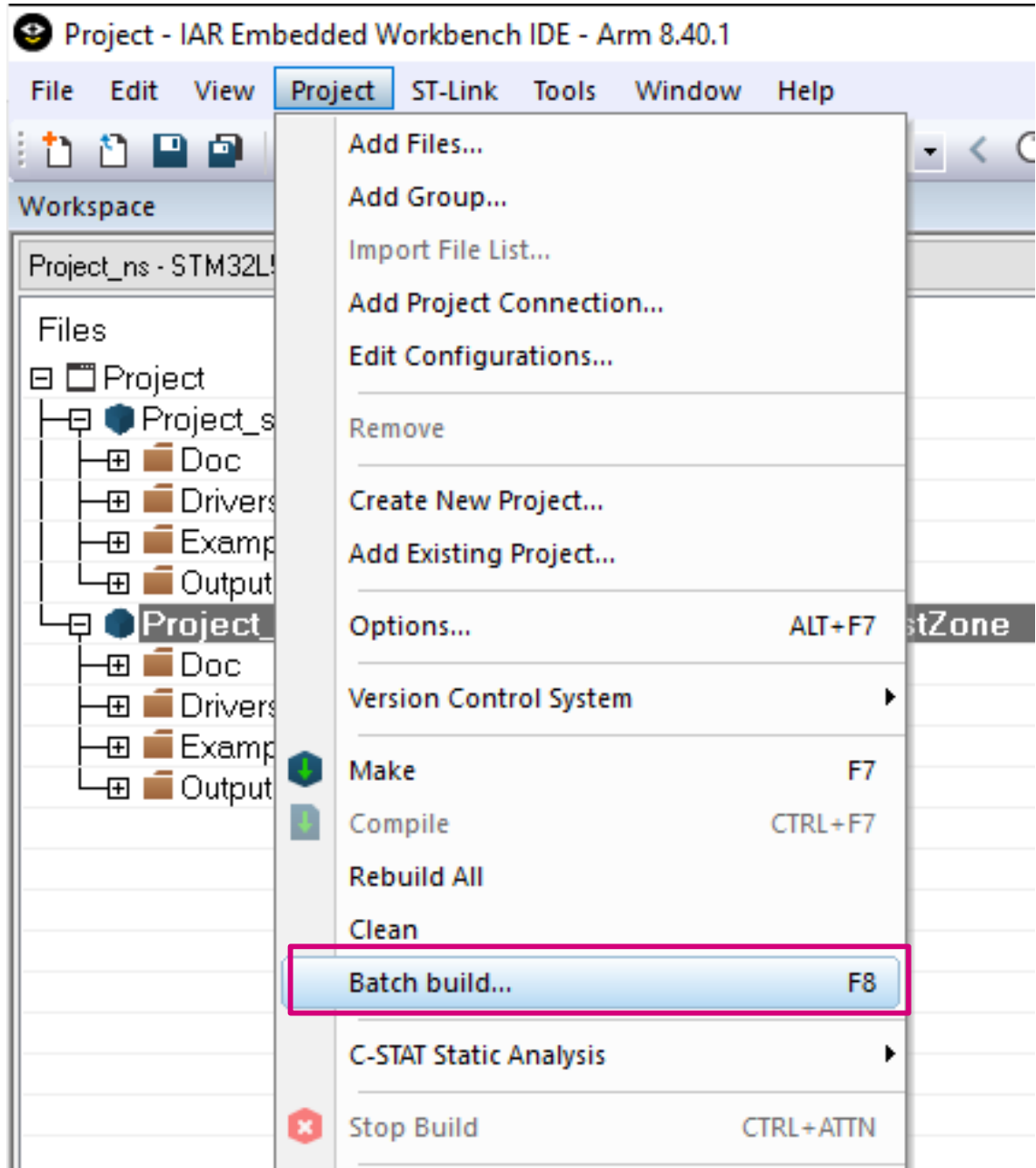
- The other configurations are similar to the secure project.

10.3 Build projects

Both projects are ready to be built.

1. Select **Project / Batch Build** or the icon available from the menu bar (see Figure 41).

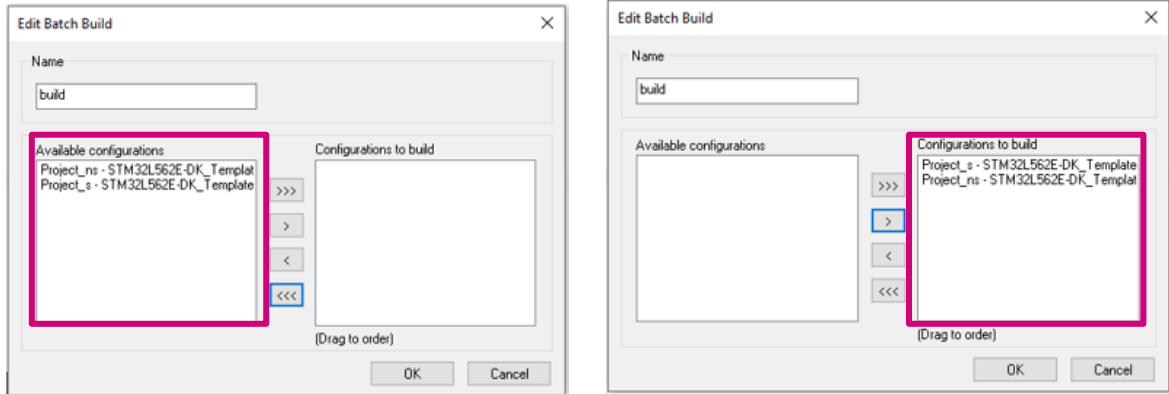
Figure 41. Project batch build



2. Add the two configurations to be built at the same time (see [Figure 42](#)).

Note: The secure project must be built first in order to create the import library for the nonsecure project. In order to build the secure project before the nonsecure one, it must be first in the build order as illustrated below.

Figure 42. Project batch build order



10.4 Execute from secure code to nonsecure code

In order to execute any code, it has to be downloaded to the board as follows:

1. Before downloading the project, connect to the STM32L562E-DK Discovery board as follows (see Figure 43):
 - Connect the ST-LINKV3 programming and debugging tool to the Discovery board by plugging the USB cable to the CN17 ST-LINK USB connector of the board.
 - LD3 illuminates in red when the ST-LINKV3 is connected.

Figure 43. STM32L562E-DK Discovery board in connected status



2. Select the Project_ns project as active project then load the nonsecure binary code. Start a debug session by clicking the download and debug button in the toolbar to program the flash memory and start debugging (see Figure 44).

Figure 44. Download and debug launch button



Note: when trying to load the nonsecure application, the following warning messages are displayed.

Figure 45. Nonsecure application loading warning error message samples

```

Debug Log
Log
Tue Feb 11, 2020 16:09:31: Warning:
Tue Feb 11, 2020 16:09:31: Verify error at address 0x080400CF, target byte: 0x00, byte in file: 0x08
Tue Feb 11, 2020 16:09:31: Warning:
Tue Feb 11, 2020 16:09:31: Verify error at address 0x080400D0, target byte: 0x00, byte in file: 0xAD
Tue Feb 11, 2020 16:09:31: Warning:
Tue Feb 11, 2020 16:09:31: Verify error at address 0x080400D1, target byte: 0x00, byte in file: 0x0F
Tue Feb 11, 2020 16:09:31: Warning:
Tue Feb 11, 2020 16:09:31: Verify error at address 0x080400D2, target byte: 0x00, byte in file: 0x04
Tue Feb 11, 2020 16:09:31: Warning:
Tue Feb 11, 2020 16:09:31: Verify error at address 0x080400D3, target byte: 0x00, byte in file: 0x08
Tue Feb 11, 2020 16:09:31: Warning:
Tue Feb 11, 2020 16:09:31: Verify error at address 0x080400D4, target byte: 0x00, byte in file: 0xB1
Tue Feb 11, 2020 16:09:31: Warning:
Tue Feb 11, 2020 16:09:31: Verify error at address 0x080400D5, target byte: 0x00, byte in file: 0x0F
Tue Feb 11, 2020 16:09:31: Warning:
Tue Feb 11, 2020 16:09:31: Verify error at address 0x080400D6, target byte: 0x00, byte in file: 0x04
Tue Feb 11, 2020 16:09:31: Warning:
Tue Feb 11, 2020 16:09:31: Verify error at address 0x080400D7, target byte: 0x00, byte in file: 0x08
Tue Feb 11, 2020 16:09:31: Warning:
Tue Feb 11, 2020 16:09:31: Verify error at address 0x080400D8, target byte: 0x00, byte in file: 0xB5
Tue Feb 11, 2020 16:09:31: Warning: Too many verify errors, only the first 200 are displayed
    
```

This is an expected behavior, as in the verification phase, the debugger attempts to read back the loaded content and compares it with the compiled binary. The debugger generates a secure transaction in a nonsecure regions (@ 0x08040000 nonsecure flash) before SAU configuration. This access is not allowed and the content reads zero.

3. Select the Project_s project as active project then load the nonsecure binary then start a debug session.

Note: The system always boots in secure code (main.c) at first and the secure application then launches the nonsecure application

4. The secure status is provided from secure register under CPU registers (see Figure 46).

Figure 46. Secure register location

Name	Value	Access
R0	0x00000000	ReadWrite
R1	0x00000000	ReadWrite
R2	0x00000000	ReadWrite
R3	0x00000000	ReadWrite
R4	0x00000000	ReadWrite
R5	0x00000000	ReadWrite
R6	0x00000000	ReadWrite
R7	0x00000000	ReadWrite
R8	0xFFFFFFFF	ReadWrite
R9	0xFFFFFFFF	ReadWrite
R10	0xFFFFFFFF	ReadWrite
R11	0xFFFFFFFF	ReadWrite
R12	0xFFFFFFFF	ReadWrite
SP	0x30000818	ReadWrite
SPLIM	0x00000000	ReadWrite
LR	0xFFFFFFFF	ReadWrite
+ xPSR	0x01000000	ReadWrite
+ APSR	0x00000000	ReadWrite
+ IPSR	0x00000000	ReadWrite
+ EPSR	0x01000000	ReadWrite
PC	0x0C000928	ReadWrite
+ PRIMASK	0x00000000	ReadWrite
+ BASEPRI	0x00000000	ReadWrite
+ BASEPRI_MAX	0x00000000	ReadWrite
+ FAULTMASK	0x00000000	ReadWrite
+ CONTROL	0x00000000	ReadWrite
+ IAPSR	0x00000000	ReadWrite
+ EAPSR	0x01000000	ReadWrite
+ IEPSR	0x01000000	ReadWrite
SECURE	0x00000001	ReadWrite
CYCLECOUNTER	0	Read Only
CCTIMER1	0	
CCTIMER2	0	
CCSTEP	0	

SECURE
 ReadWrite
 Security state
 0: Non-Secure
 1: Secure
 Right-click for more registers and options

0 = nonsecure

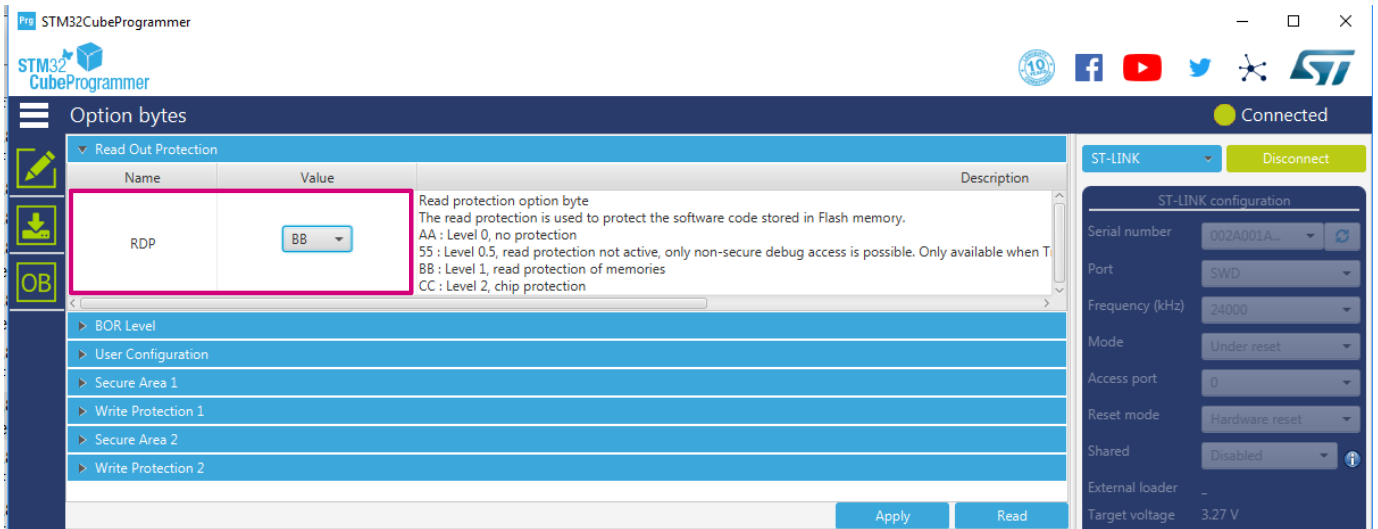
1 = secure

10.5 Connection issue to STM32L552ZE-Q when RDP is set to 0.5

The EWARM is able to connect to the device and debug the nonsecure application. To connect to the STM32L552ZE-Q, proceed as follows:

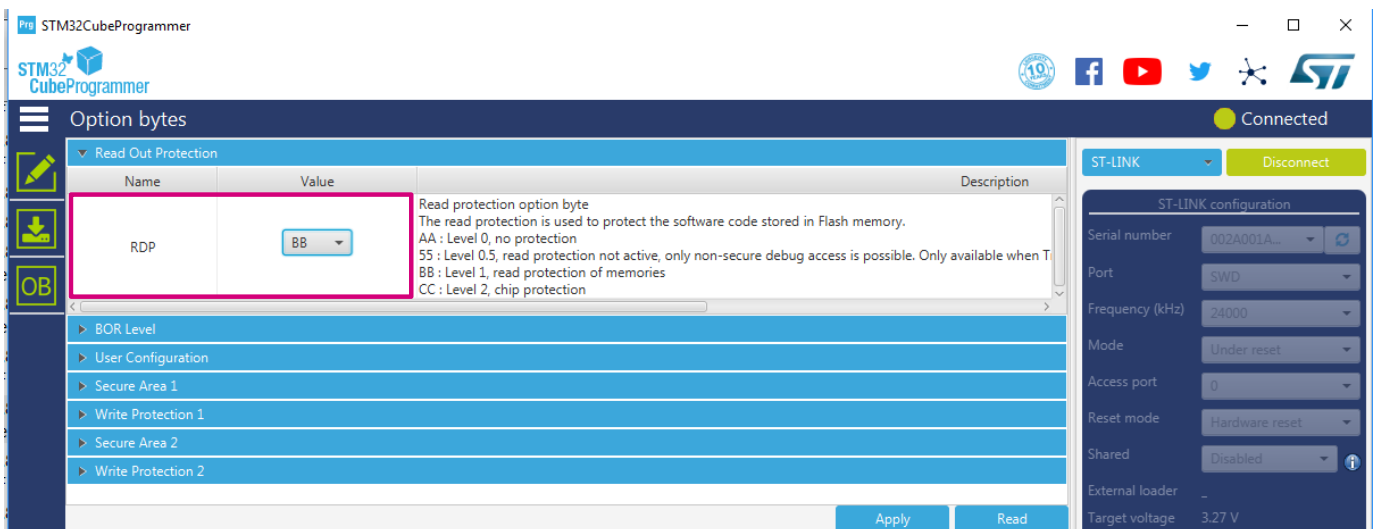
- Setting the option bytes, illustrated in Figure 47:
 - TZEN = 1
 - DBANK = 1
 - SECWM2_STRT = 0x1
 - SECWM1_PEND = 0x0.

Figure 47. Configuration of option bytes using STM32CubeProgrammer v2.2.0

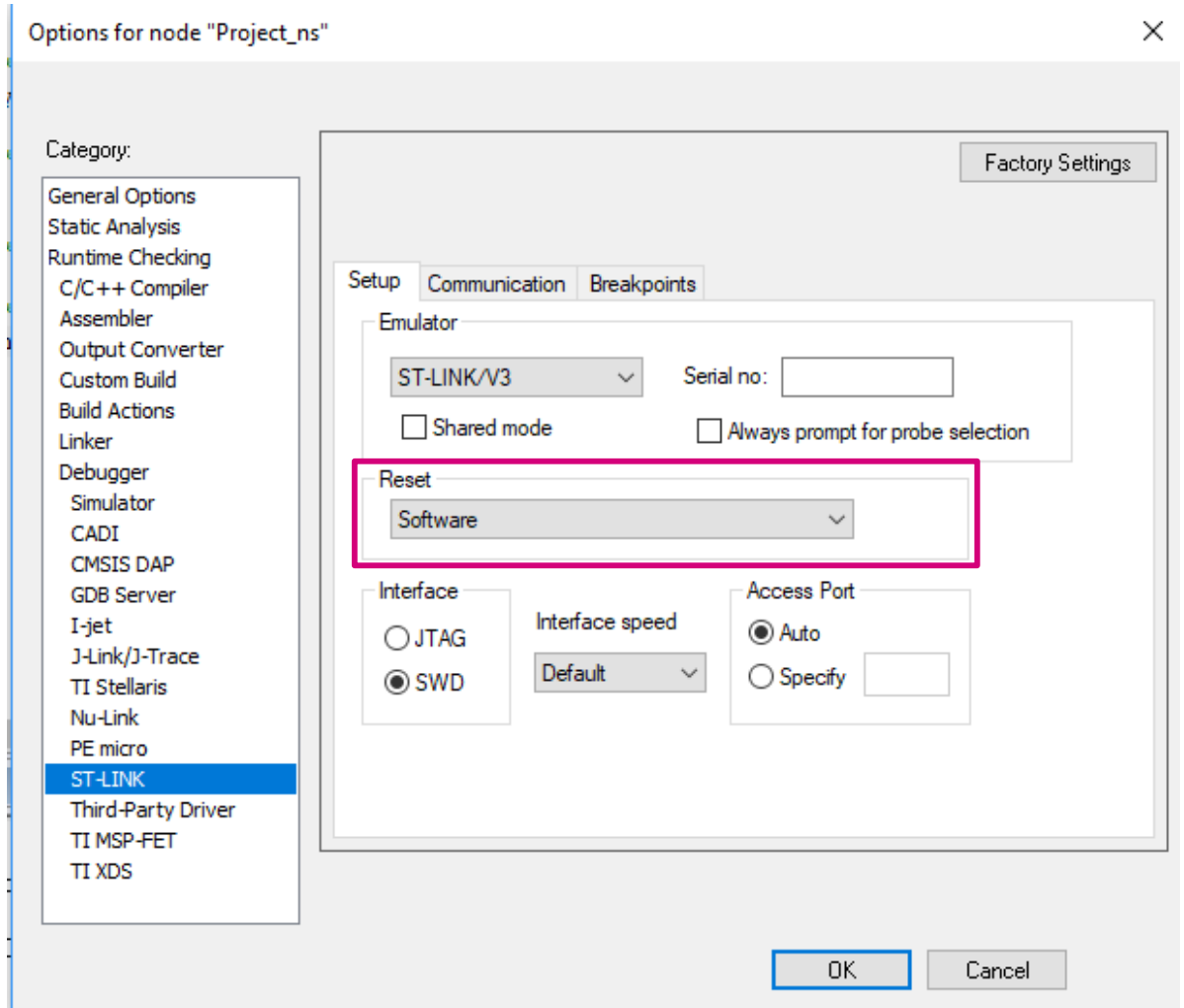


- Load the nonsecure binary (at 0x08040000) then load the secure binary (at 0x0C000000) as specified in the section above.
- Using STM32CubeProgrammer to set RDP=0x55 to reduce debug to nonsecure (see Figure 48).

Figure 48. RDP=0.5

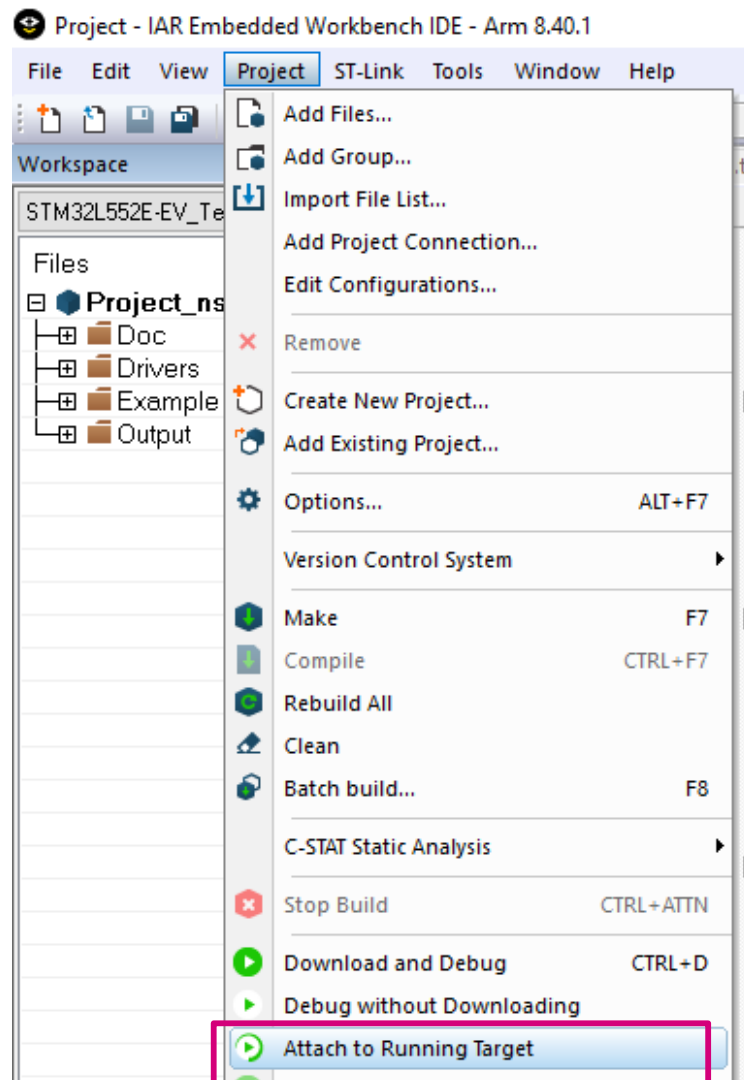


4. Change the Reset mode to software reset: **Project options / ST-LINK** in the "Setup" tab select "Software" from the "Reset" field as illustrated in Figure 49.

Figure 49. Reset mode selection


5. Connect to the device in Hot-plug mode from: **Project / Attach to the Running Target** (see Figure 50).

Figure 50. Attach to running target option



Note: IDEs do not support the nonsecure flash reprogramming in RDP level 0.5, only STM32CubeProgrammer allows it.

11 Using CubeIDE for Cortex®-M33 with TrustZone®

This part is explained in the *Getting started with STM32 development in CubeIDE* (AN5394), which is available on www.st.com.

Revision history

Table 1. Document revision history

Date	Version	Changes
21-Feb-2020	1	Initial release.
01-Aug-2022	2	Updated Figure 42. Project batch build order
15-May-2023	3	Updated: <ul style="list-style-type: none"> Product series to include STM32U5 series Section 1: General information
12-Sep-2023	4	Updated: <ul style="list-style-type: none"> Title Section 1: General information
23-Jul-2024	5	Updated: <ul style="list-style-type: none"> Title Section 1: General information
21-Jan-2025	6	Updated: <ul style="list-style-type: none"> Product series to include STM32U3 series

Contents

1	General information	2
2	Arm® Cortex®-M33 core overview	3
3	TrustZone® concept of the Armv8-M	4
4	SAU / IDAU - TrustZone® concept	5
5	Debugging modes	6
5.1	Invasive debug	6
5.2	Non-invasive debug	6
6	Debug access	7
6.1	Secure debug access	7
6.2	Nonsecure debug access	7
7	Flash memory protection	8
7.1	Readout protection level when TrustZone® is disabled	8
7.2	RDP level transition scheme when TrustZone® is disabled	8
7.3	Readout protection level when TrustZone® is enabled	8
7.4	RDP level transition scheme when TrustZone® is enabled	9
8	Starting with secure/nonsecure project	10
9	Using MDK-ARM for Cortex®-M33 with Trust Zone	11
9.1	Secure project settings	11
9.2	Nonsecure project settings	17
9.2.1	Building a project	22
9.3	Execute from secure code to nonsecure code	23
10	Using EWARM for Cortex M33 with TrustZone®	26
10.1	Secure project settings	26
10.2	Nonsecure project settings	34
10.3	Build projects	37
10.4	Execute from secure code to nonsecure code	39
10.5	Connection issue to STM32L552ZE-Q when RDP is set to 0.5	42
11	Using CubeIDE for Cortex®-M33 with TrustZone®	45
	Revision history	46
	List of figures	48

List of figures

Figure 1.	Security state in Armv8-M	4
Figure 2.	RDP level transition scheme when TrustZone® is disabled (TZEN = 0)	8
Figure 3.	RDP level transition scheme when TrustZone® is disabled (TZEN = 1)	9
Figure 4.	Configuration of option bytes using STM32CubeProgrammer	10
Figure 5.	MDK-ARM project structure	11
Figure 6.	Secure project selection.	11
Figure 7.	Device selection	12
Figure 8.	Project_s target options	13
Figure 9.	Project_s Linker configuration	14
Figure 10.	Scatter file sample	15
Figure 11.	Target options debug	15
Figure 12.	Debug configuration	16
Figure 13.	Flash-loader settings	17
Figure 14.	Project_ns nonsecure project selection	17
Figure 15.	Device selection	18
Figure 16.	Memory configuration	19
Figure 17.	Linker options.	20
Figure 18.	Scatter file sample	20
Figure 19.	Debug settings	21
Figure 20.	FlashLoader configuration	21
Figure 21.	Project batch setup	22
Figure 22.	Project build ordering	22
Figure 23.	Build both projects in one step	22
Figure 24.	STM32L562E-DK Discovery board in connected status	23
Figure 25.	Load the nonsecure binary	23
Figure 26.	Download and debug button.	24
Figure 27.	Main.c sample code	24
Figure 28.	Code switch to nonsecure code status.	25
Figure 29.	CPU status	25
Figure 30.	EWARM v8.40.1 project explorer view	26
Figure 31.	Setting the project to active status.	27
Figure 32.	Device selection	28
Figure 33.	Linker configuration.	29
Figure 34.	Project debugger setup	30
Figure 35.	FlashLoader selection	31
Figure 36.	Selecting the nonsecure output file as an extra image	32
Figure 37.	Project setup	33
Figure 38.	Project set up: general options	34
Figure 39.	Project linker configuration	35
Figure 40.	Linker library setup	36
Figure 41.	Project batch build	37
Figure 42.	Project batch build order	38
Figure 43.	STM32L562E-DK Discovery board in connected status	39
Figure 44.	Download and debug launch button.	40
Figure 45.	Nonsecure application loading warning error message samples	40
Figure 46.	Secure register location	41
Figure 47.	Configuration of option bytes using STM32CubeProgrammer v2.2.0	42
Figure 48.	RDP=0.5	42
Figure 49.	Reset mode selection	43
Figure 50.	Attach to running target option	44

IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2025 STMicroelectronics – All rights reserved