AN5421

Application note

Getting started with STM32L5 Series microcontrollers and TrustZone® development

Introduction

This document aims to provide guidelines using the EWARM and MDKARM software tool-chains on STM32L5 Series microcontrollers.

This application note provides a basis for building and debugging secure and non-secure 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 Series microcontrollers when the TrustZone® is enabled through TZEN option bit.

1 General information

This document applies to the STM32L5 Series single-core Arm®-based microcontroller.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2 Arm® Cortex®-M33 core overview

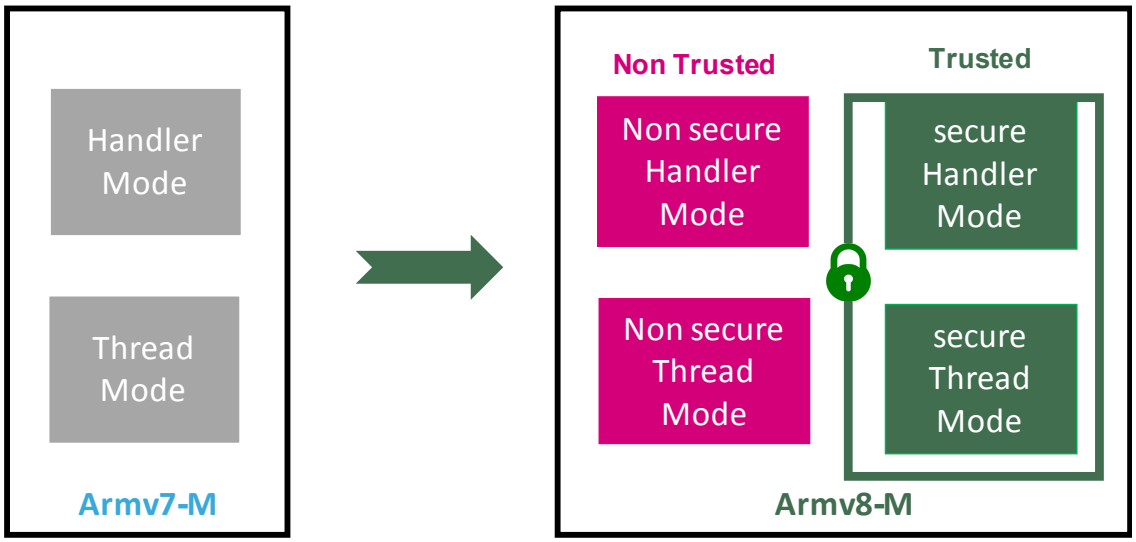
The Arm® Cortex®-M33 is the first full-feature implementation of Armv8-M with TrustZone® secure technology and digital signal processing functionality. The processor supports a large number of flexible configuration options to facilitate the deployment of a wide range of applications, and offers a dedicated 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
- non-secure 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 non-secure (NS) domain restricted to predefined entry points
- exception hardware automatically saves and clears secure register states when switching to non-secure
- extensive banking of interrupt or exception control, SysTick
- memory protection unit for each of the secure and non-secure 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 non-secure states.

- IDAU: provides a first memory partition as secure or, non-secure 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 non-secure 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 non-secure callable has a lower secure priority and non-secure 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 non-secure after reset. Even if all the Flash memory is non-secure 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.
- Non-secure 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.
  - Non-secure callable (NSC) : contains entry functions (for example entry points for APIs), this is to prevent non-secure 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 non-secure 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 non-secure information.

6.2 Non-secure debug access

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

7 Flash memory protection

7.1 Readout protection level when TrustZone® is disabled

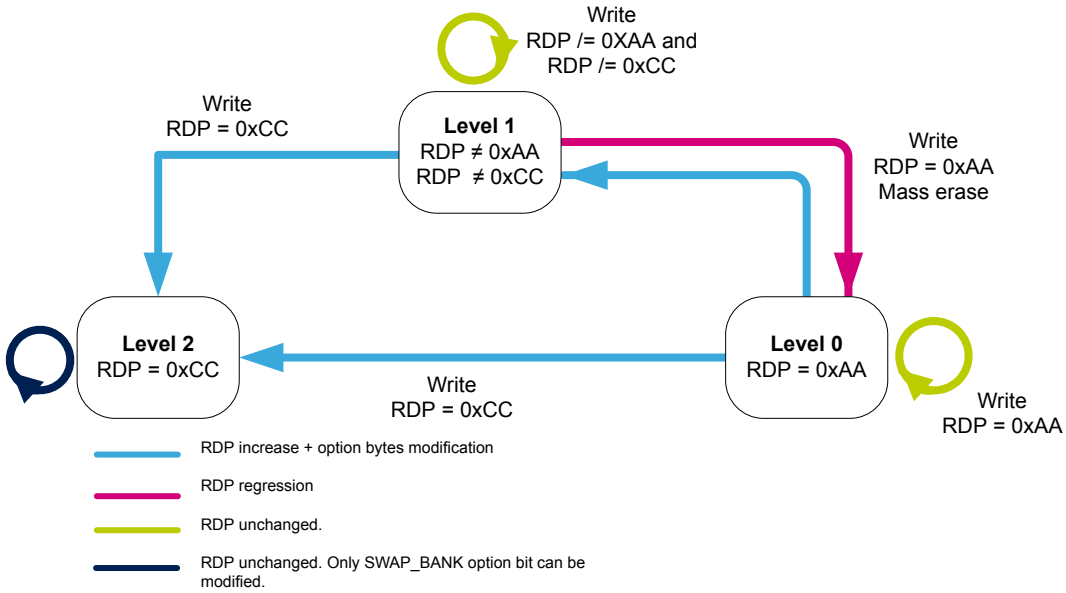
There are three readout protection levels as listed below:

- 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

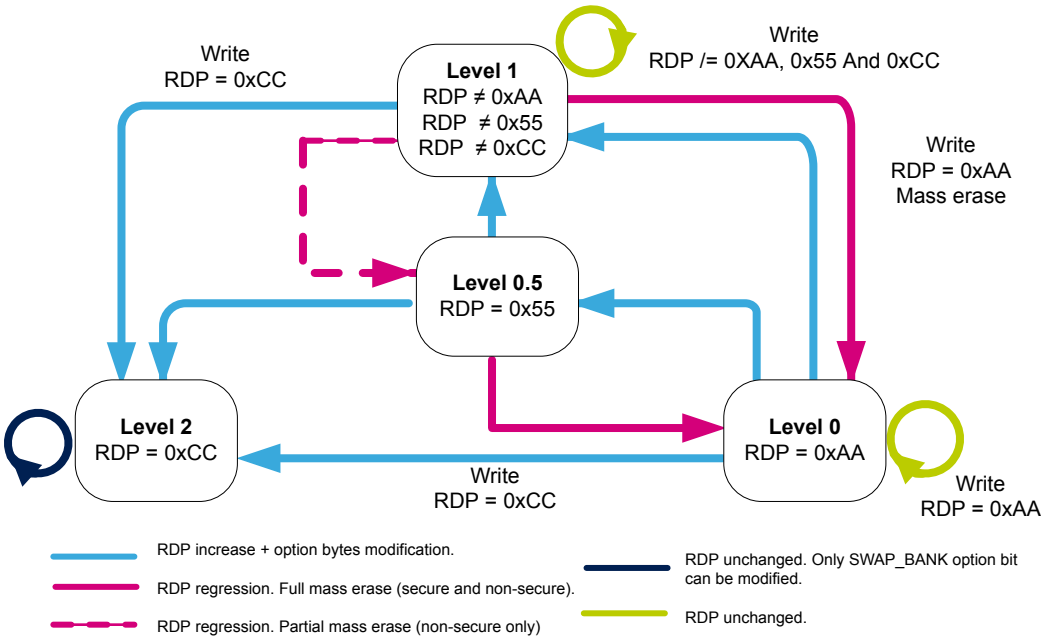
In addition to the RDP levels mentioned previously is set, there is a new RDP level named 0.5 which allows the following features:

- All read and write operations to / from the non-secure Flash memory are possible. The debug access to secure area is prohibited. Debug access to non-secure area remains possible.
- Non-secure debug mode: non-secure debug is possible when the CPU is in non-secure 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)



Note: RDP regression is only available through the debug interface or the system bootloader.

8 Starting with secure/non-secure project

EWARM and MDK-ARM provide very similar approaches to support STM32L5 Series microcontrollers. It is done using two separate projects: secure and non-secure.

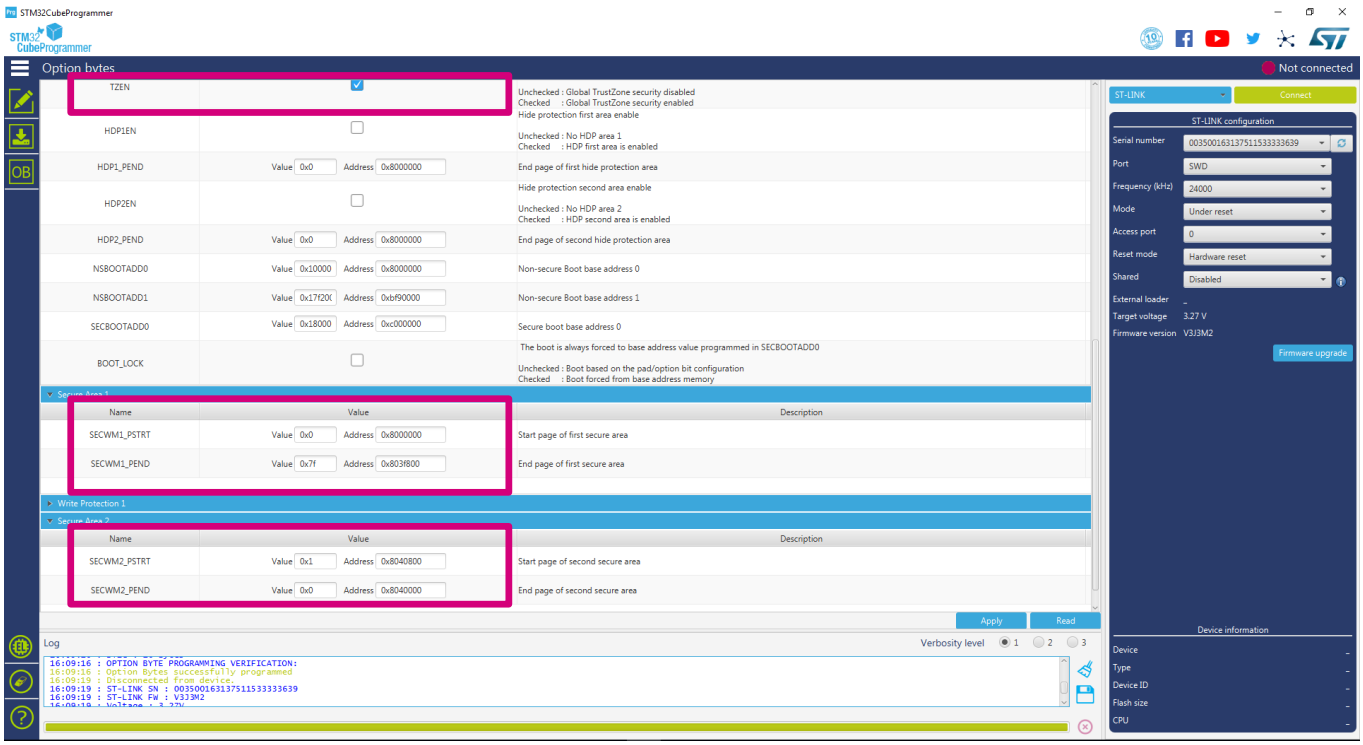
- Section 9 provides the MDK-ARM project instructions.
- Section 10 provides the instructions for EWARM.
- Section 11 provides the instructions for the CubeIDE.

Each section provides step by step instructions explaining the project setup of the secure and non-secure parts using STM32L5 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 non-secure 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



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.

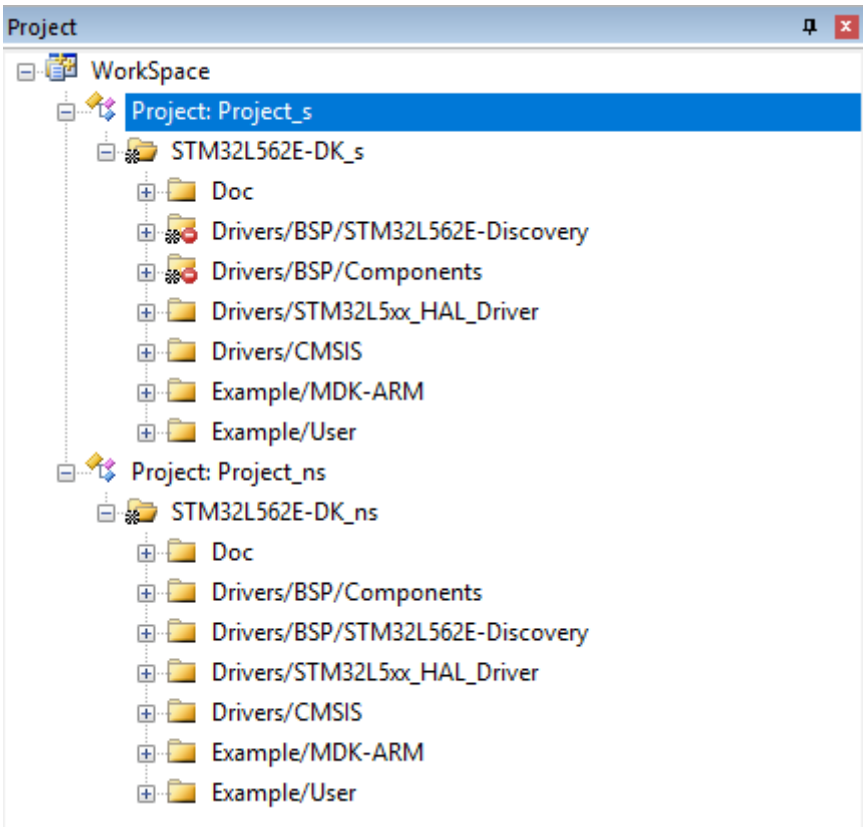
The MDK-ARM v5.27.0.0 and STM32L562-DK disco board are used for this section.

9.1 Secure project settings

This section outlines the secure project settings.

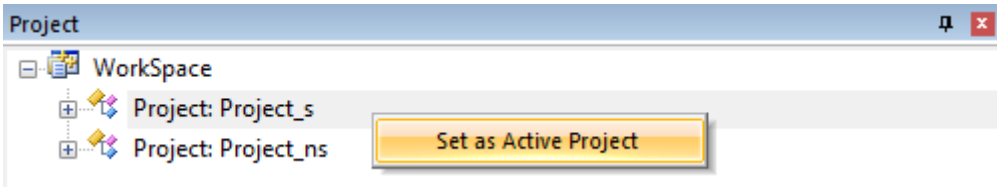
1. Open the multi-projects workspace file: “Project.uvmpw” that allows the user to work on both projects at the same time. The open project appears in the project explorer as illustrated in Figure 5.

Figure 5. MDK-ARM project structure



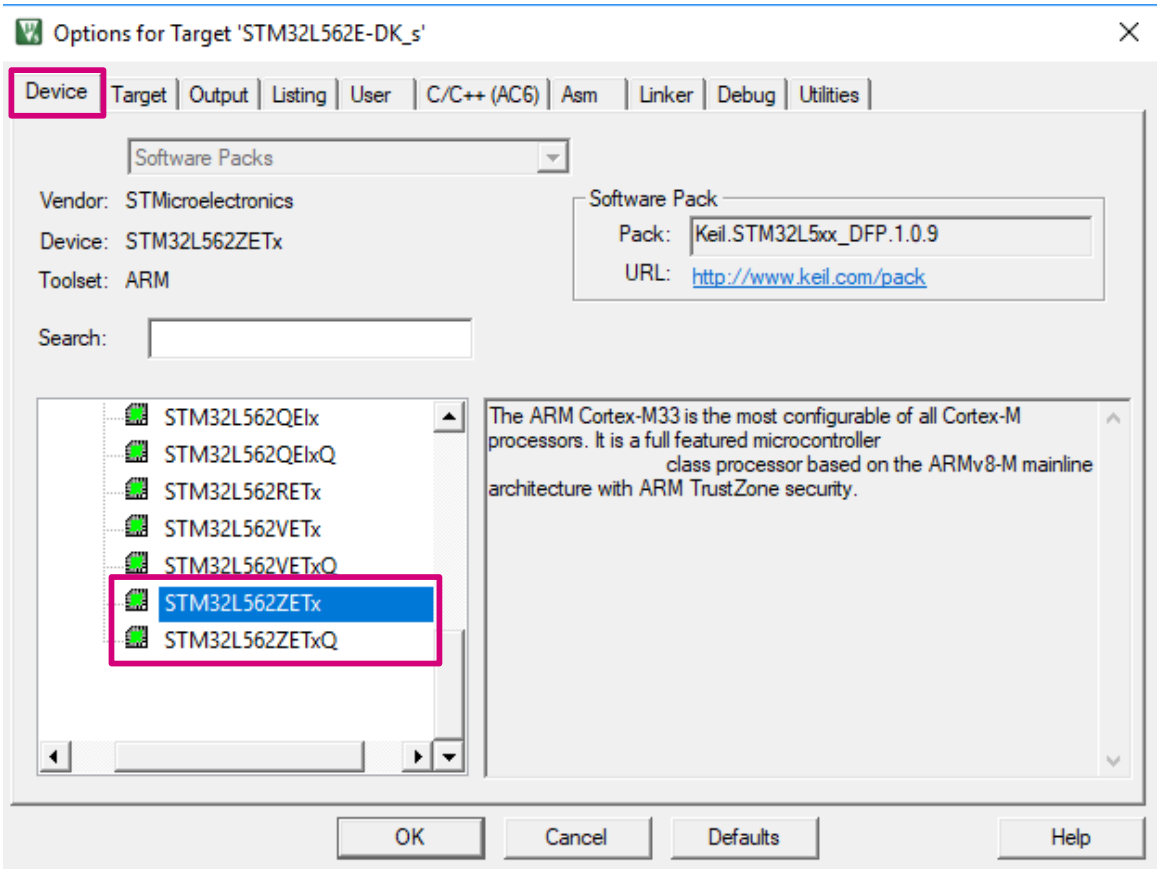
2. Set project_s as active project, see Figure 6.

Figure 6. Secure project selection



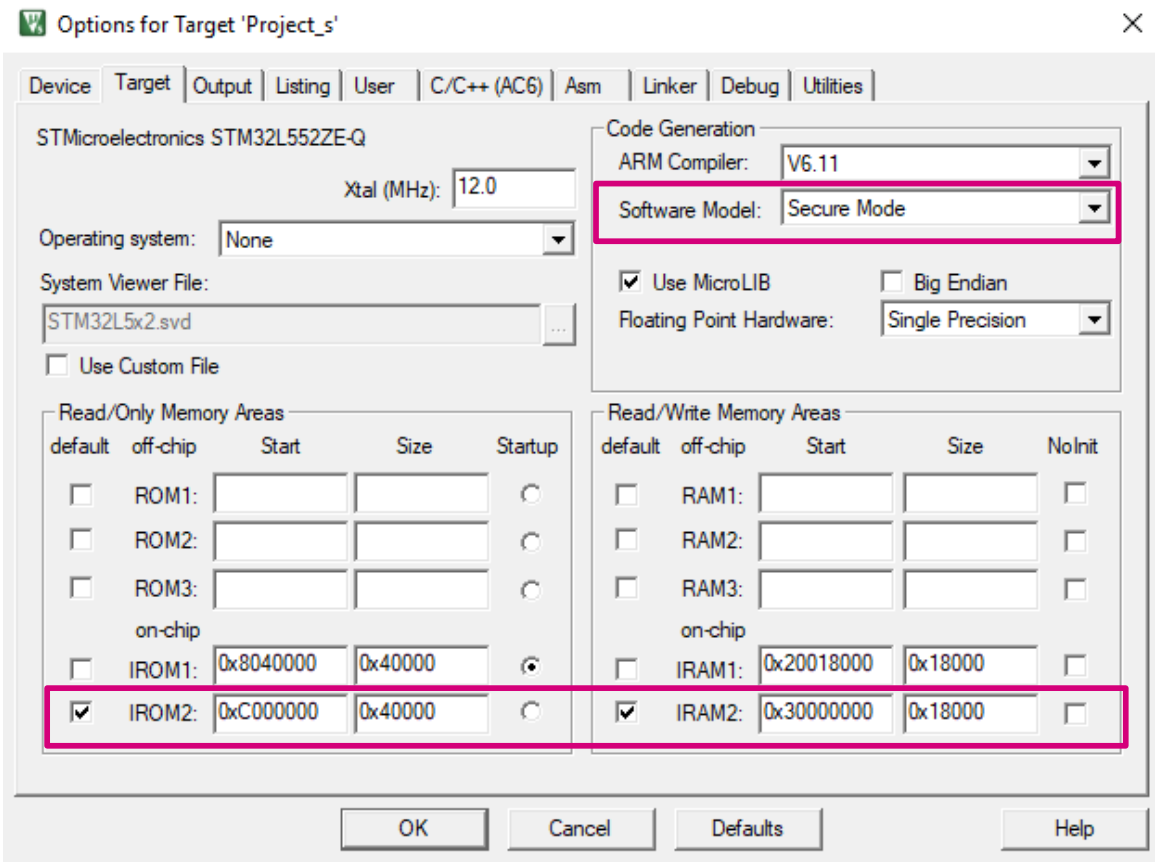
3. Select the correct device by opening the configuration window and selecting: **Project / Options for Target / Device** then select the device from the list (see Figure 7).

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



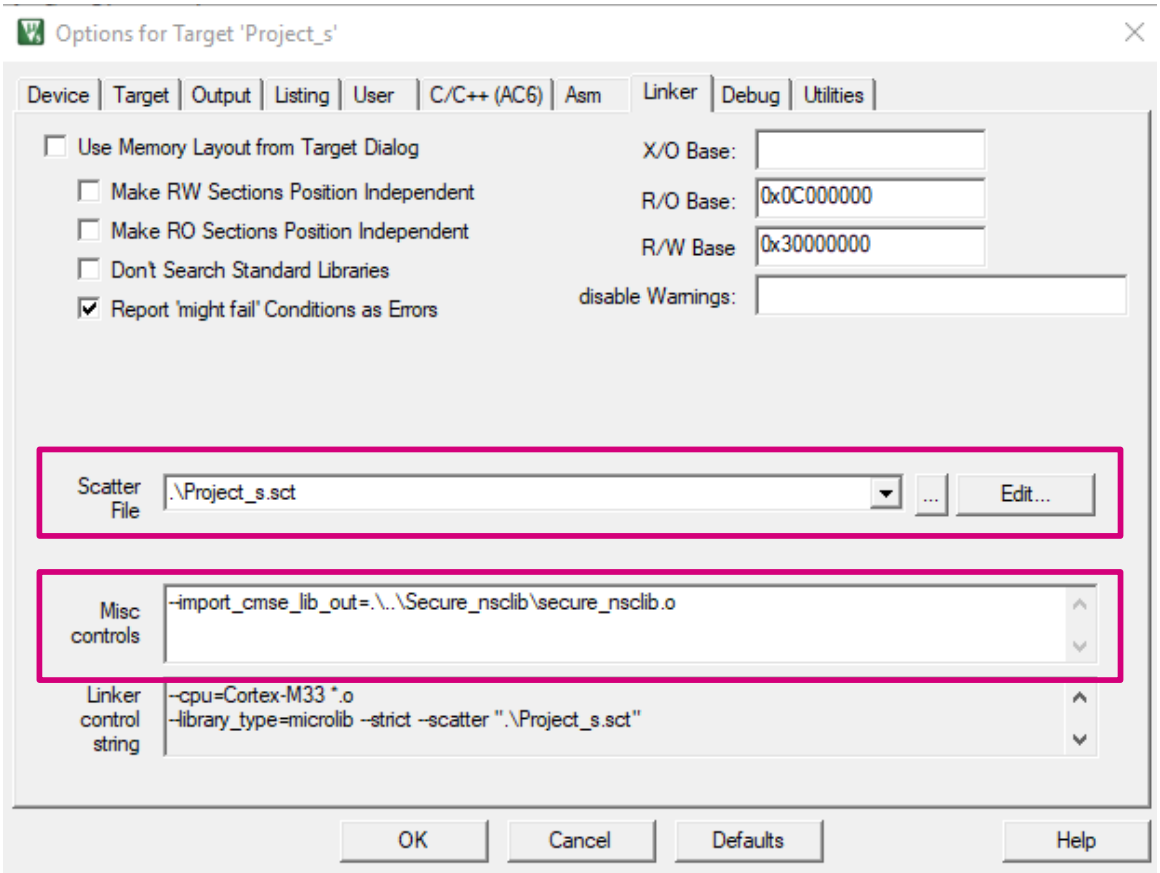
5. Ensure that the secure non-secure callable functions (NSC) object file “secure_nsclib.o” is defined in **Project / Options for Target / Linker** under **Misc Controls** section.

   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 non-secure 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.

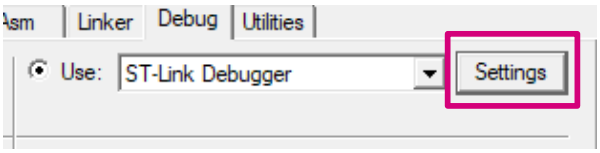
Figure 10. **Scatter file sample**

Project_s.sct

```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************

LR_IROM2 0x0C000000 0x00040000  {    ; load region size_region
  ER_IROM2 0x0C000000 0x0003E000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
   .ANY (+XO)
  }
  RW_IRAM2 0x30000000 0x00018000  {  ; RW data
   .ANY (+RW +ZI)
  }
}

LR_IROM3 0x0C03E000 0x00002000  {    ; load region size_region
  ER_IROM3 0x0C03E000 0x00002000  {  ; load address = execution address
   *(Veneer$$CMSE)                   ; check with partition.h
  }
}

```

6. Select "ST-LINK Deb ugger" 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]:

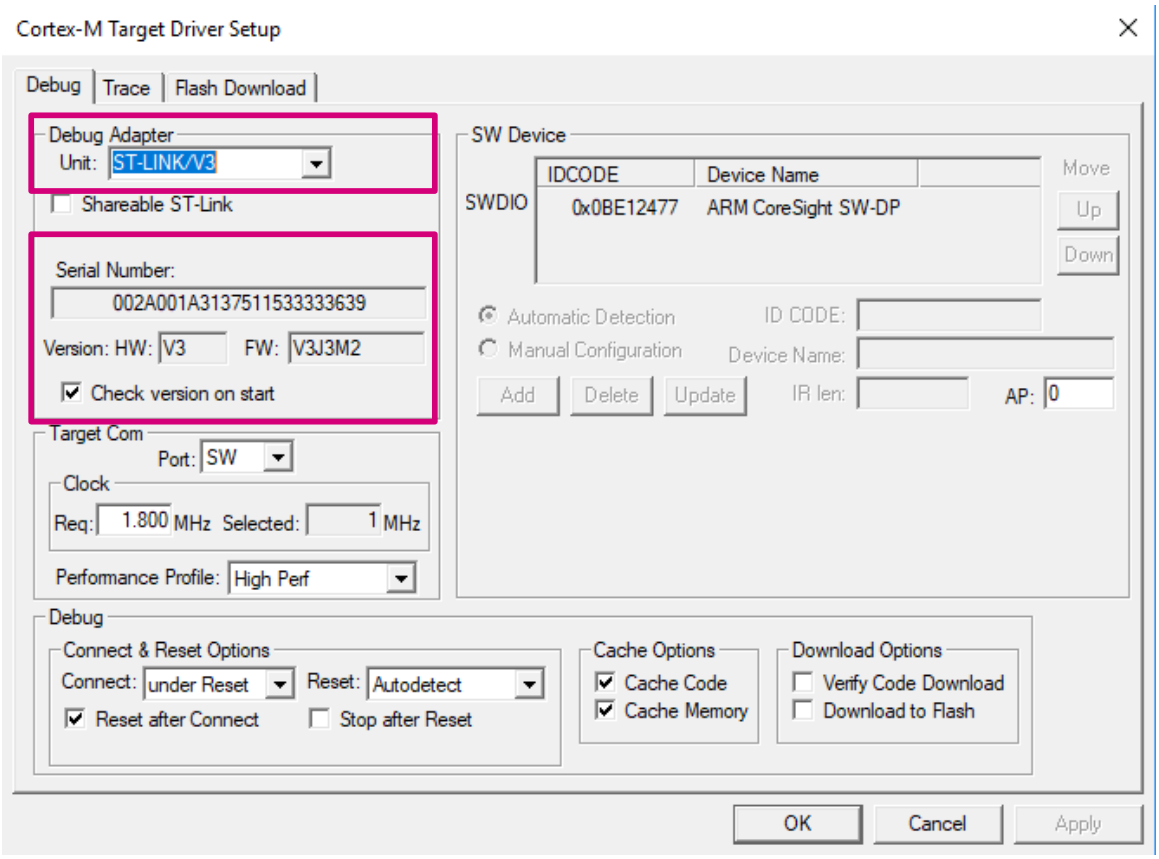
      All Armv8M based devices requires the processor SARMV8M.DLL. The TOOLS.INI file contains CPUDLL3 = SARMV8M.DLL (TDRV2, TDRV13, TDRV14, TDRV15, TDRV16).

      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.

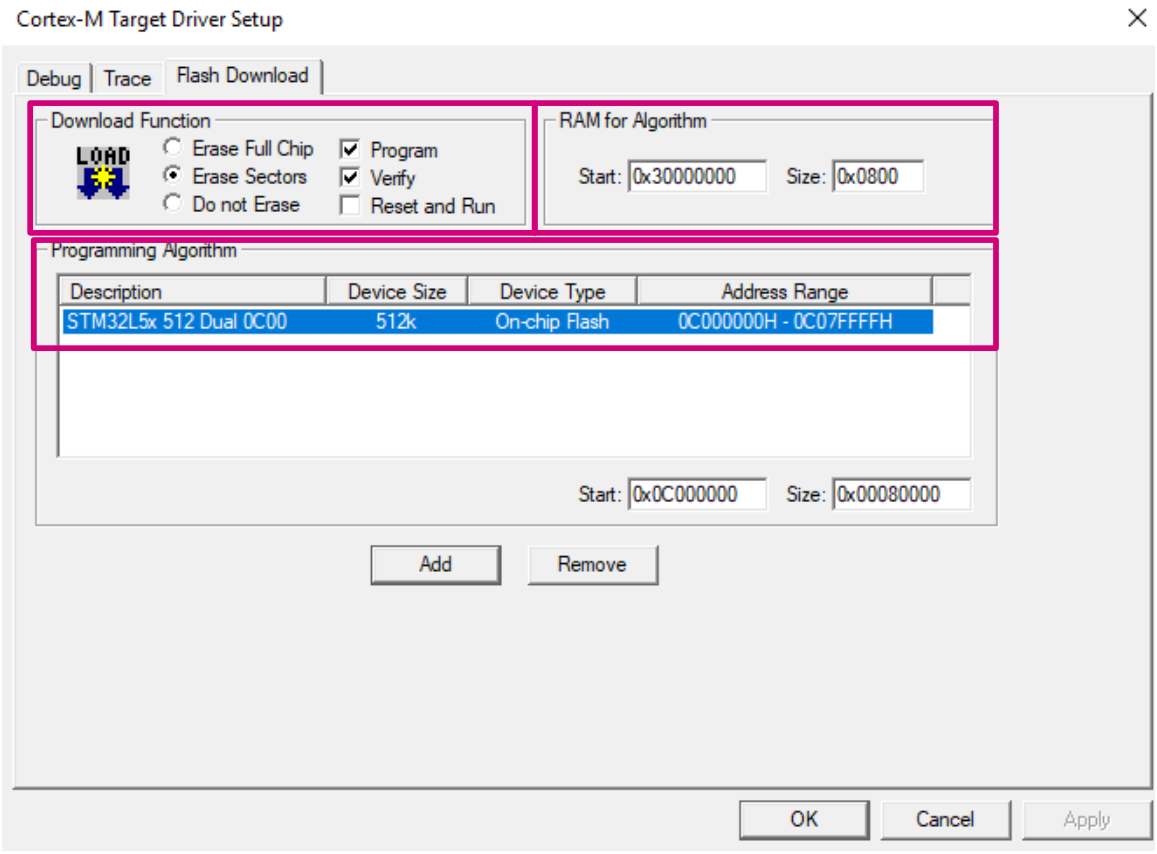
Figure 12. Debug configuration



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.

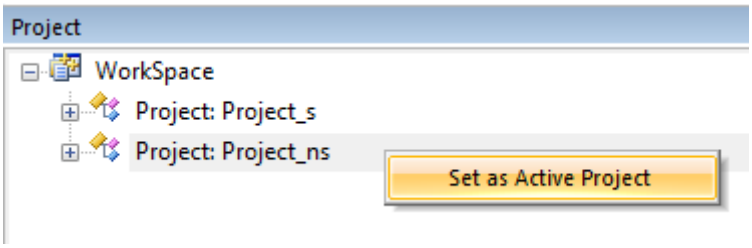
Figure 13. **Flash-loader settings**



9.2 Non-secure project settings

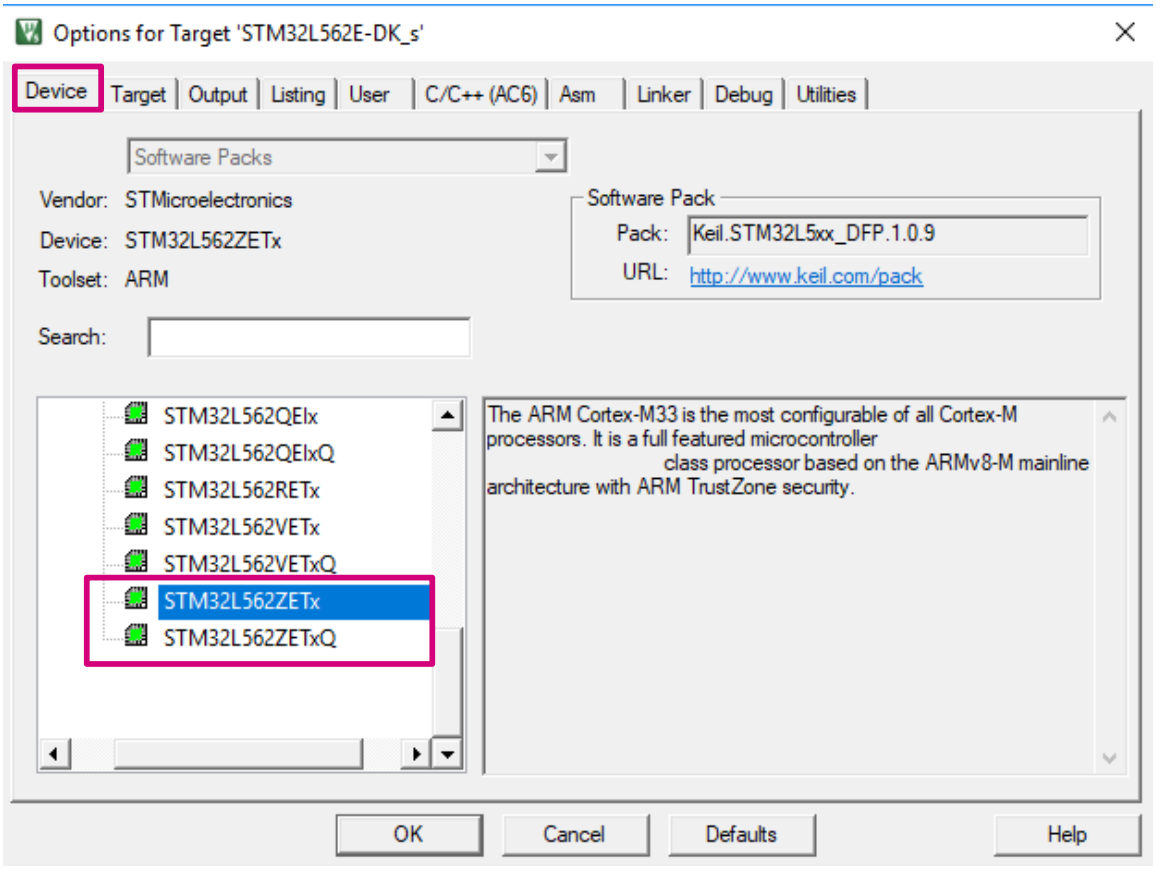
1. Set project_ns as active project (see Figure 14).

Figure 14. **Project_ns non-secure project selection**



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

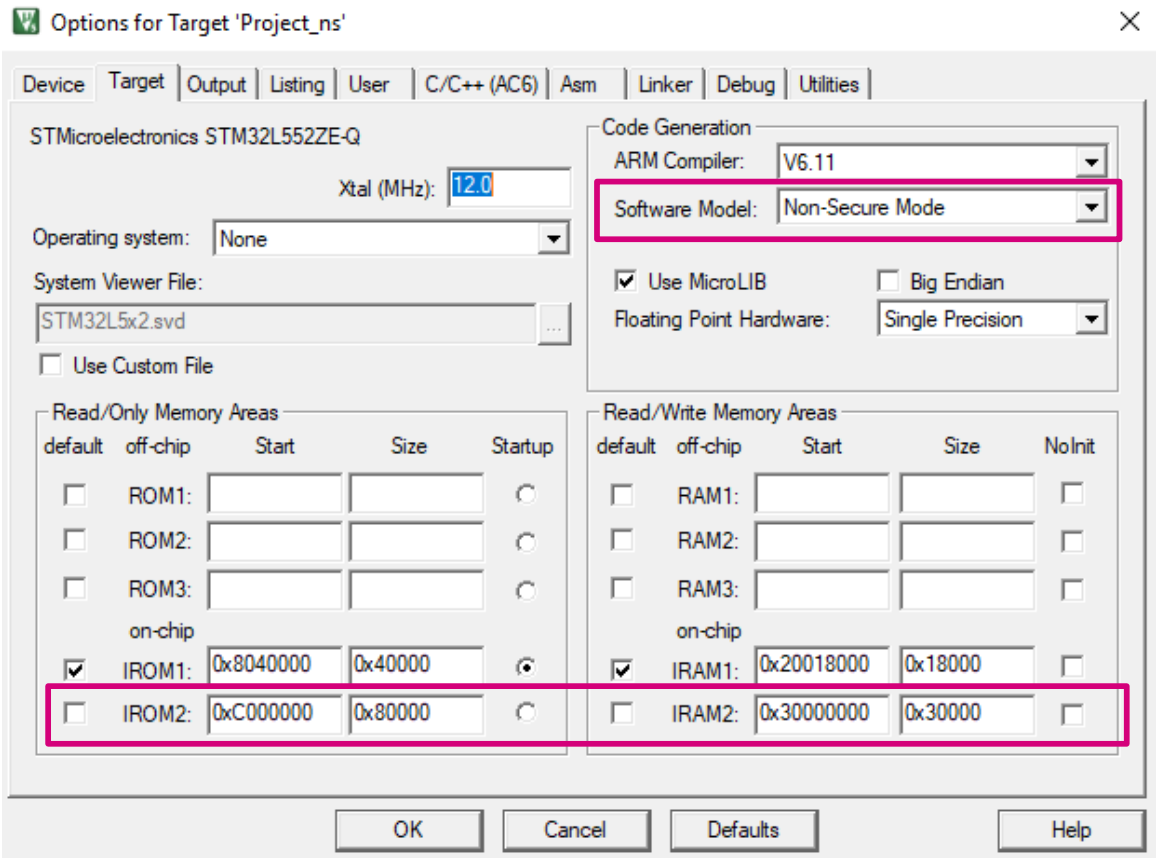
Figure 15. Device selection



3. Ensure the right memory area is selected from **Project / Options for Target / Target**:
   – Boot address 0: Flash at 0x08040000: non-secure Flash
   – Boot address 1: SRAM at 0x20018000: non-secure SRAM

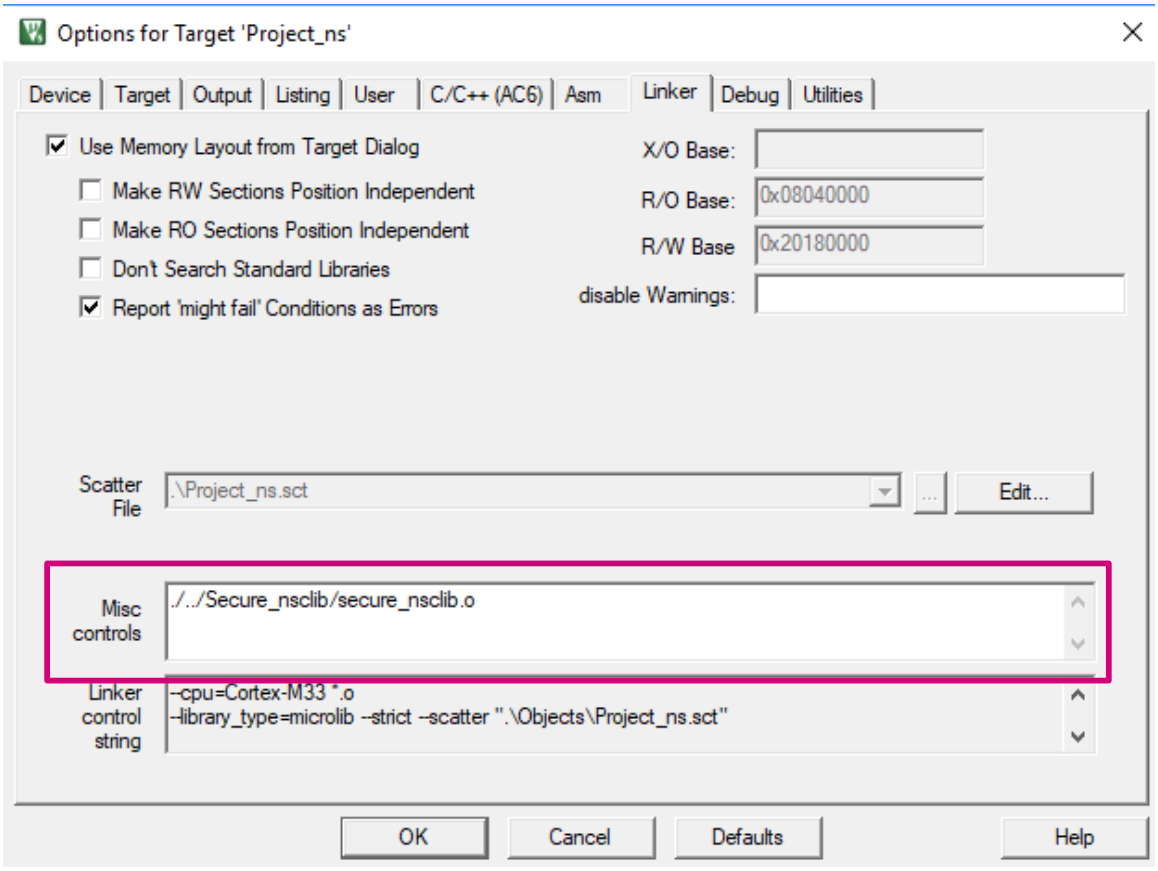
   The software model must be set in Non-secure mode (see Figure 16).

Figure 16. Memory configuration



4. Add the import library from the secure project: this file is automatically included at link time in the non-secure project. It allows the non-secure 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.

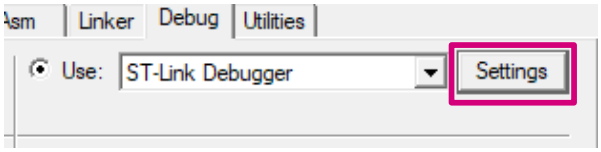
Figure 18. Scatter file sample

Project_ns.sct

```
1  ; *************************************************************
2  ; *** Scatter-Loading Description File generated by uVision ***
3  ; *************************************************************
4
5  LR_IROM1 0x08040000 0x00040000  {    ; load region size_region
6    ER_IROM1 0x08040000 0x00040000  {  ; load address = execution address
7     *.o (RESET, +First)
8     *(InRoot$$Sections)
9     .ANY (+RO)
10    .ANY (+XO)
11   }
12   RW_IRAM1 0x20018000 0x00018000  {  ; RW data
13    .ANY (+RW +ZI)
14   }
15 }
```

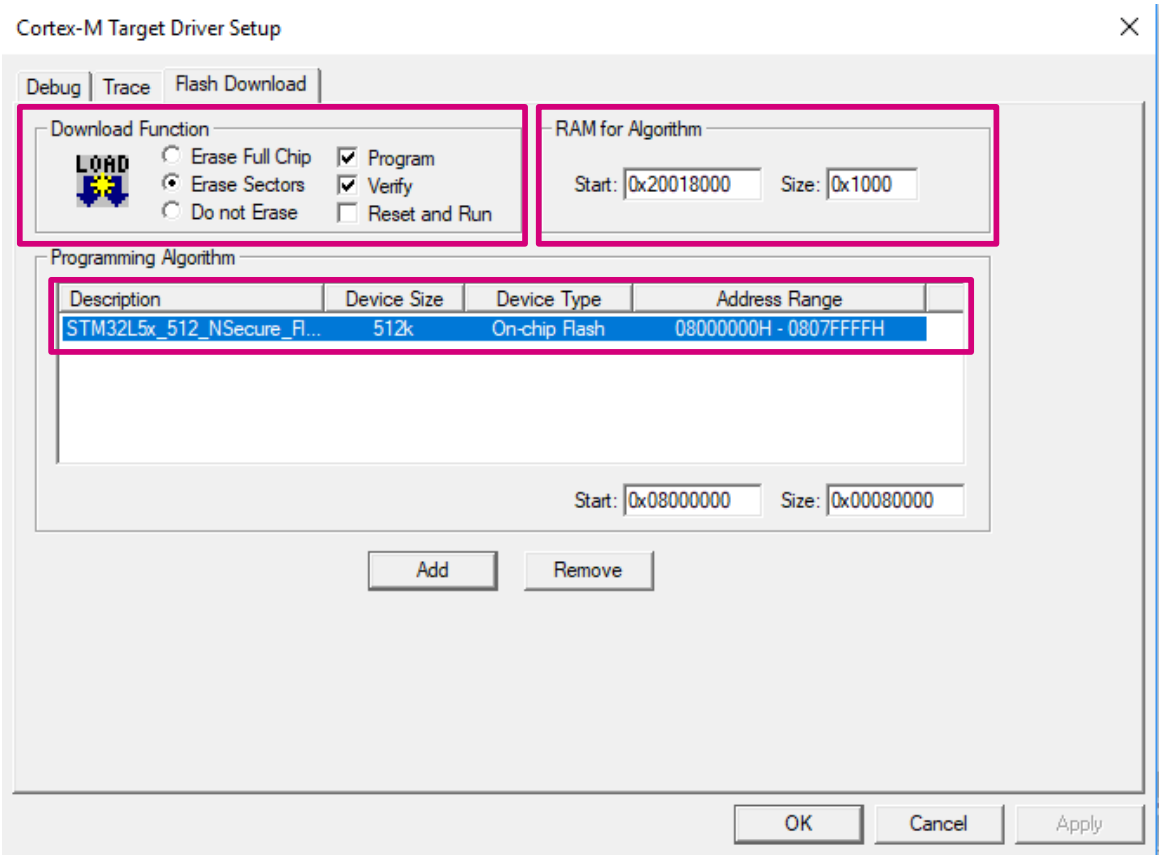
5. Select "ST-LINK debugger" from **Project / Options for Target / Debug** (see Figure 19).

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 the both projects.

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

Figure 21. Project batch setup

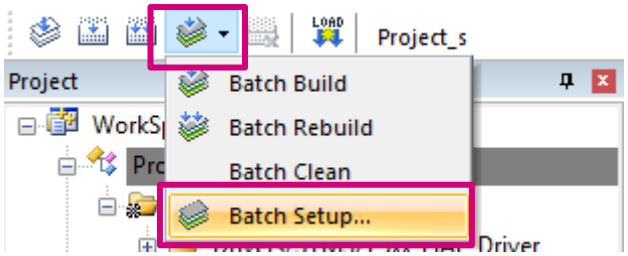
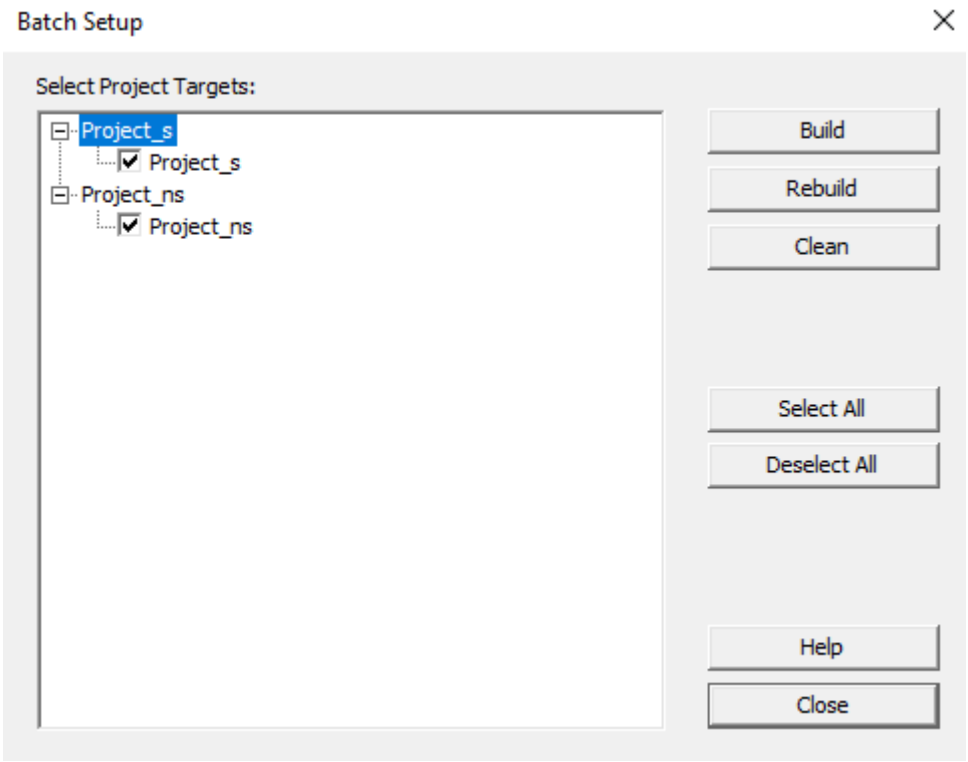
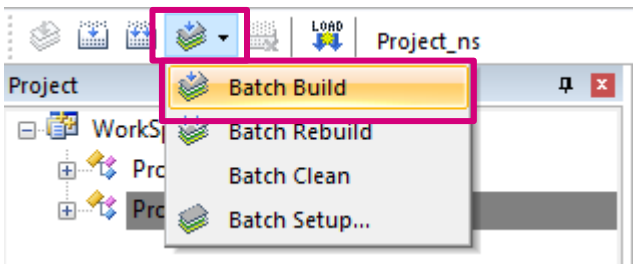


Figure 22. Project build ordering



Then, from the same menu, click on “Batch Build” to build both projects (see Figure 23).

Figure 23. Build both projects in one step

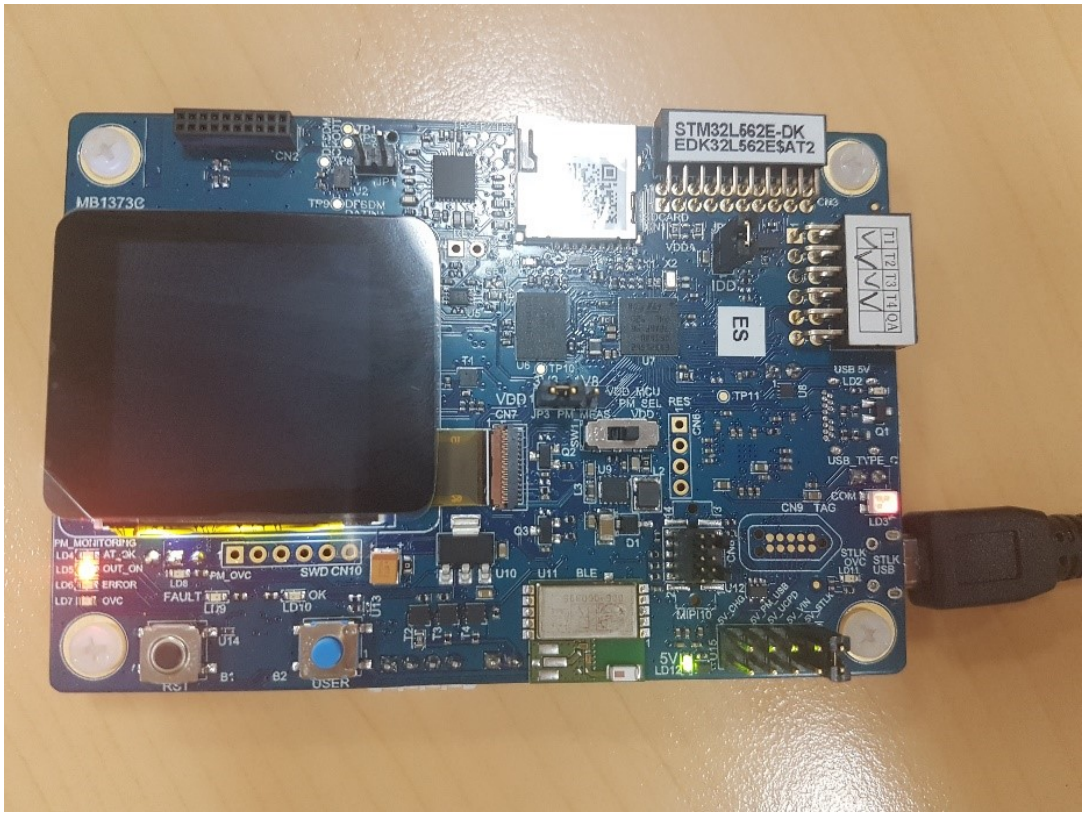


9.3 Execute from secure code to non-secure code

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

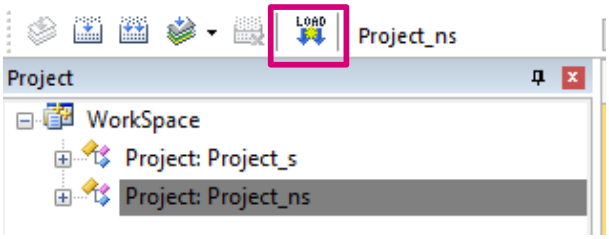
1. Connect the ST-LINKV3 programming and debugging tool on the Discovery board by plugging the USB cable to the board CN17 (ST-LINK USB connector). LD3 illuminates in red when the ST-LINKV3 is connected as illustrated in Figure 24.

Figure 24. STM32L562E-DK Discovery board in connected status



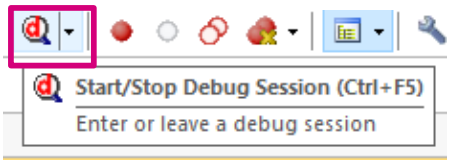
2. Select the **Project_ns** project as the active project then load the non-secure binary code. Select the **Project_s** project as the active project then load the secure binary code. This is illustrated in Figure 25.

Figure 25. Load the non-secure binary



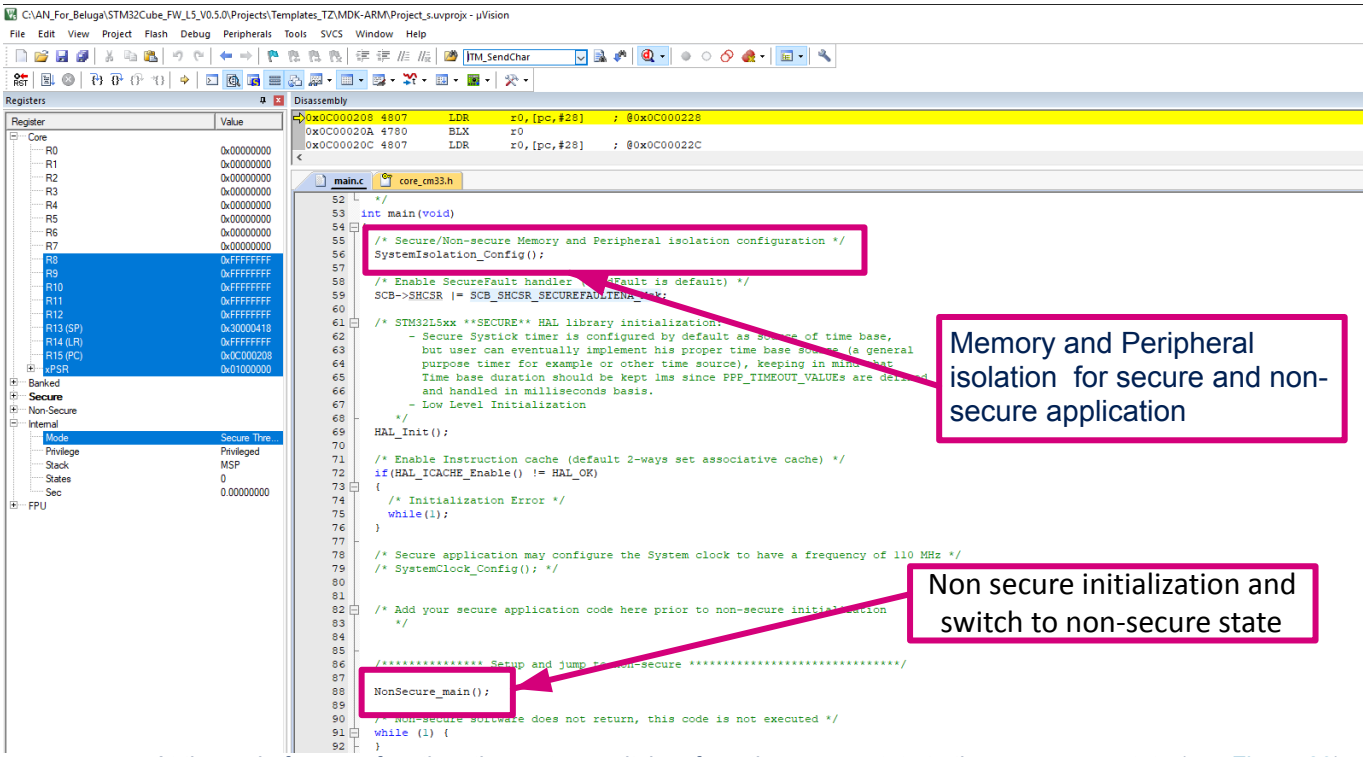
3. Start a debug session by clicking the “Download and Debug” button in the toolbar illustrated in Figure 26.

Figure 26. Download and debug button



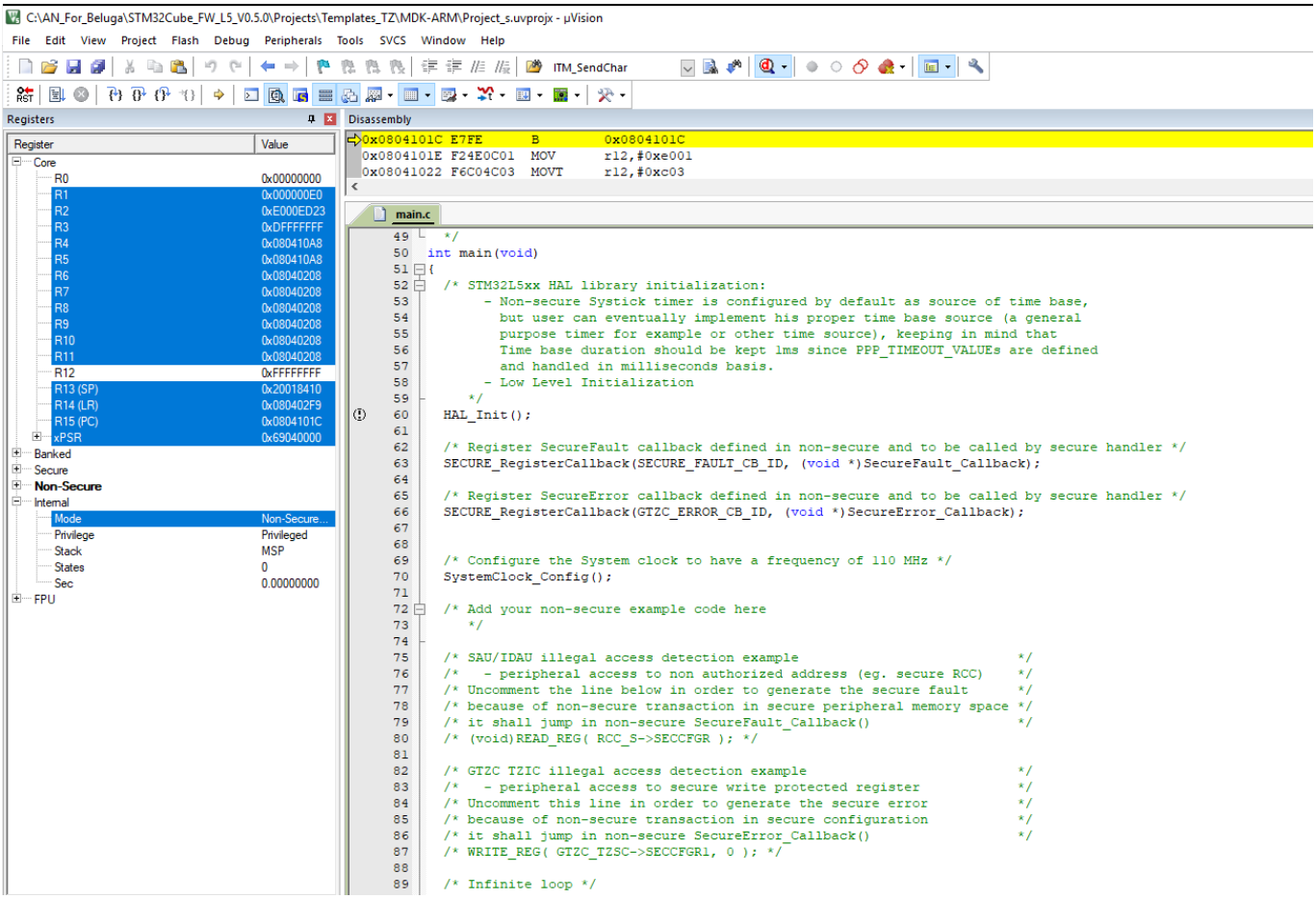
Note: The system always boots in secure code (main.c) at first and the secure application then launches the non-secure application as illustrated in below.

Figure 27. Main.c sample code



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

Figure 28. Code switch to non-secure code status



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

Figure 29. CPU status

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.

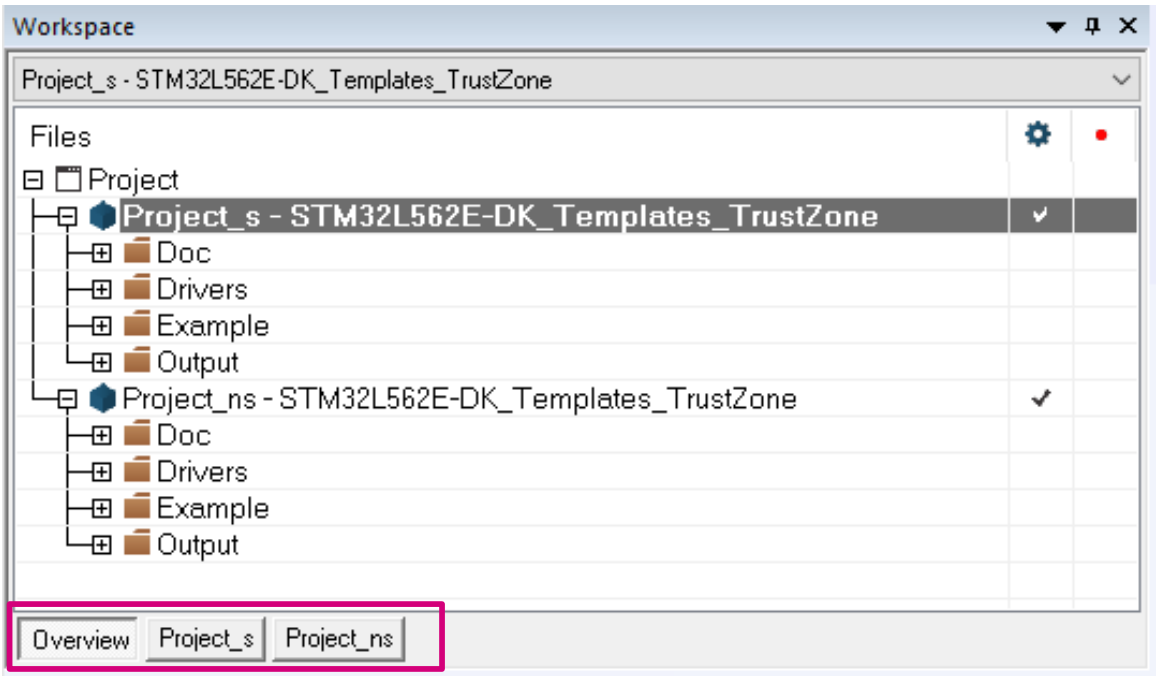
This part uses EWARM v8.40.1 and STM32L562-DK disco board.

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.

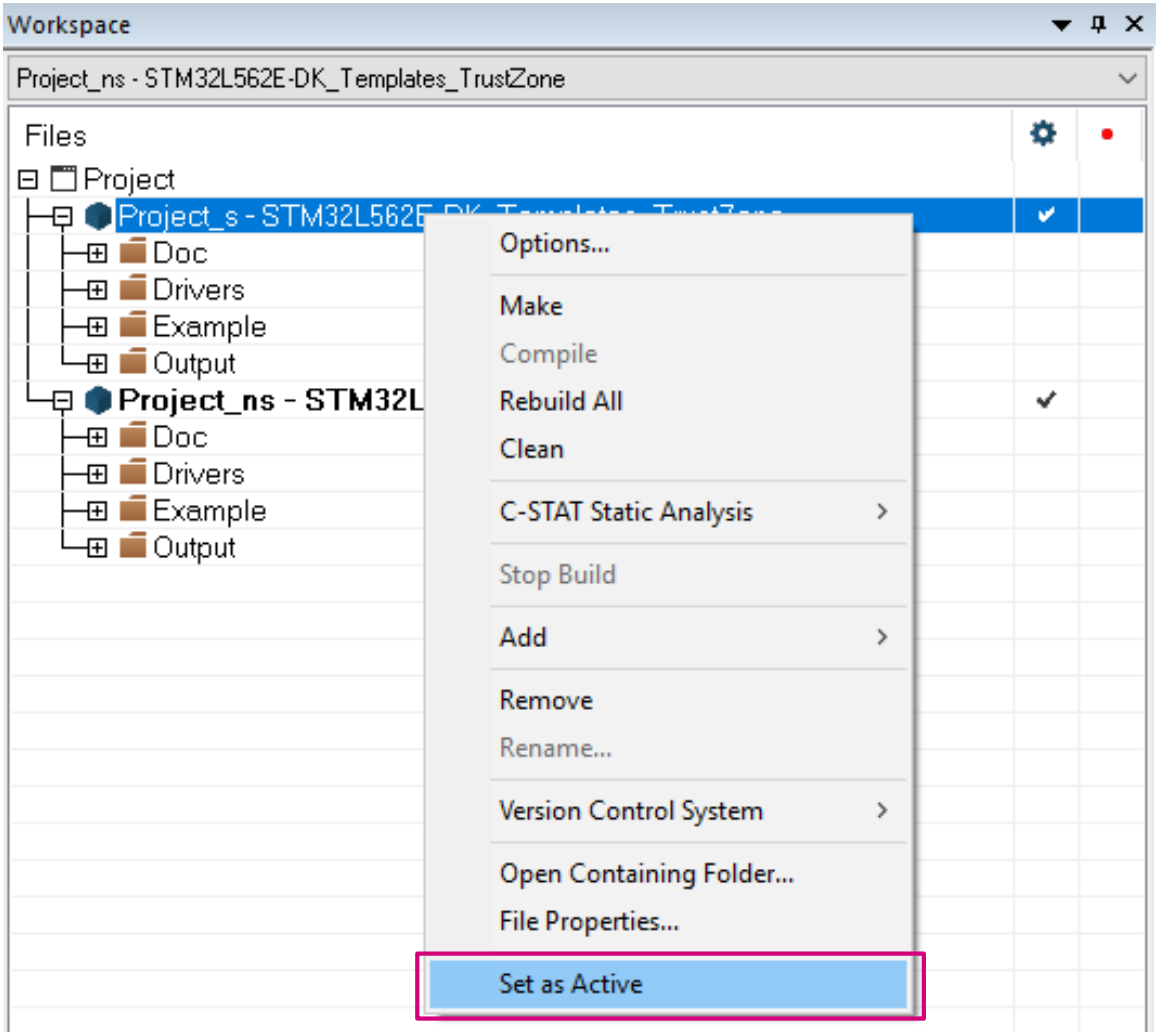
1. The open project appears in the project explorer view illustrated in Figure 30.

Figure 30. EWARM v8.40.1 project explorer view



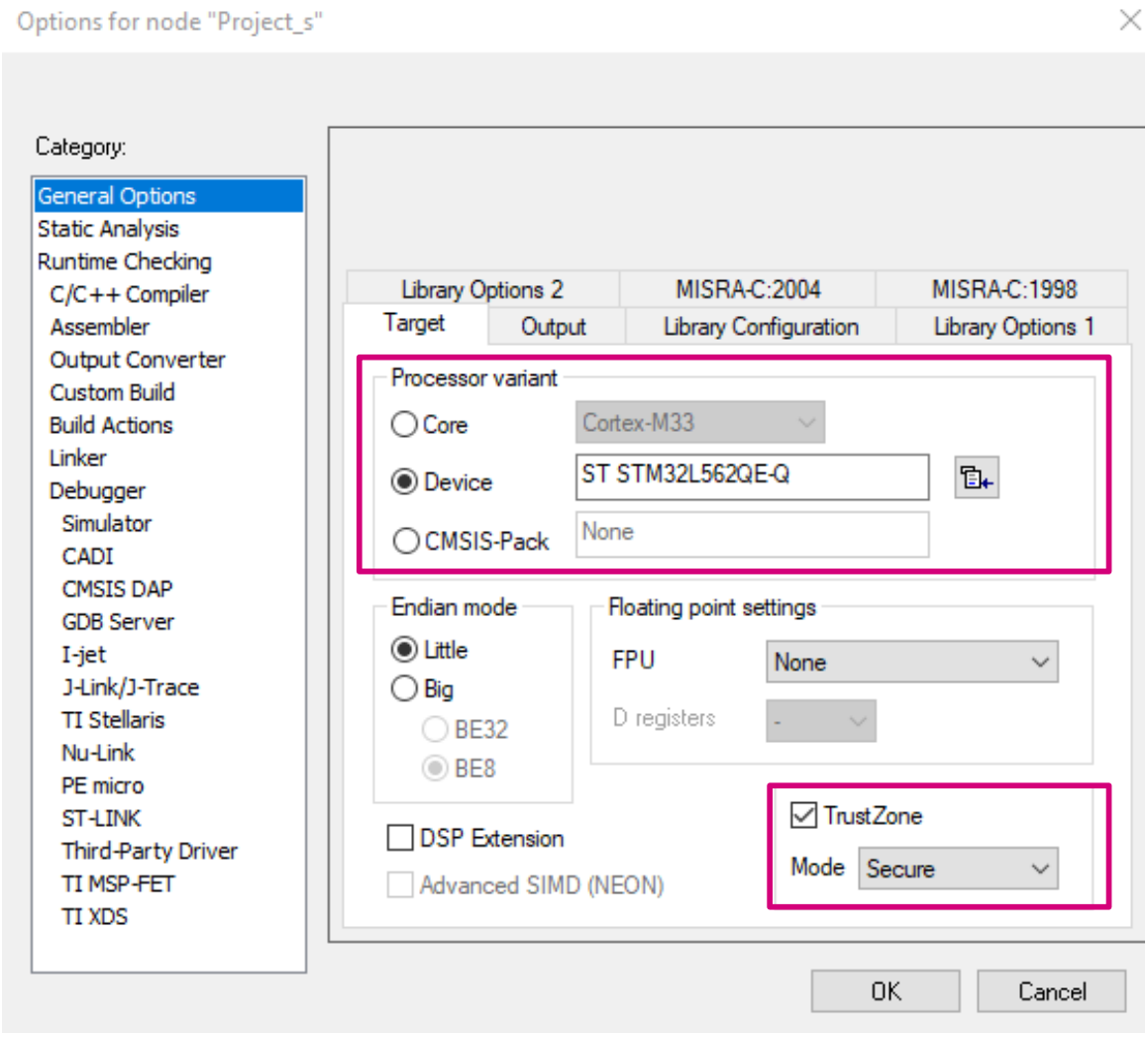
2. Set project_s-STM32L562E-DK_Templates_TrustZone as active project as illustrated in Figure 31.

Figure 31. Setting the project to active status



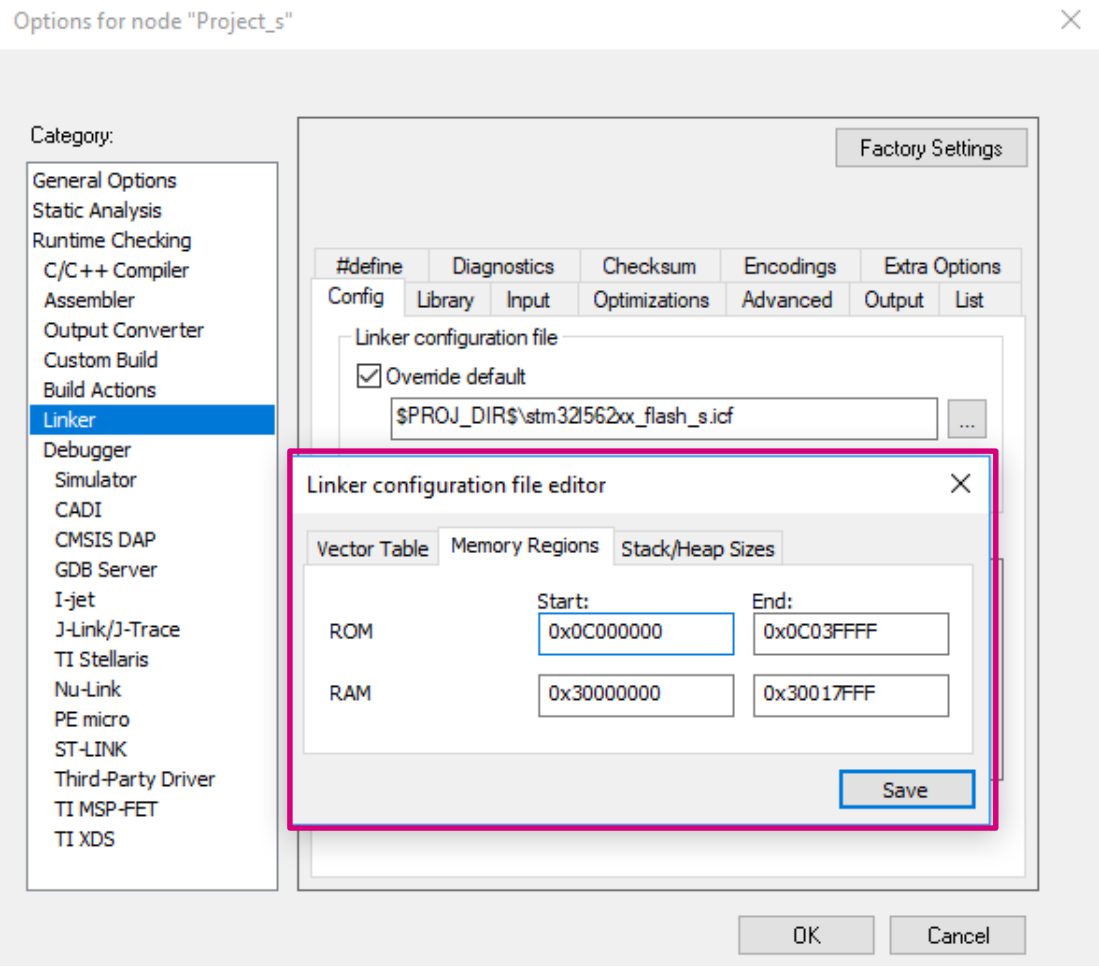
3. Open the configuration window by selecting **Project-s / Options / General Options** and select the correct device from "Processor variant" section.
   From "TrustZone" section, ensure that the mode selected is "Secure" and "TrustZone" checkbox is checked as shown in Figure 32.

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

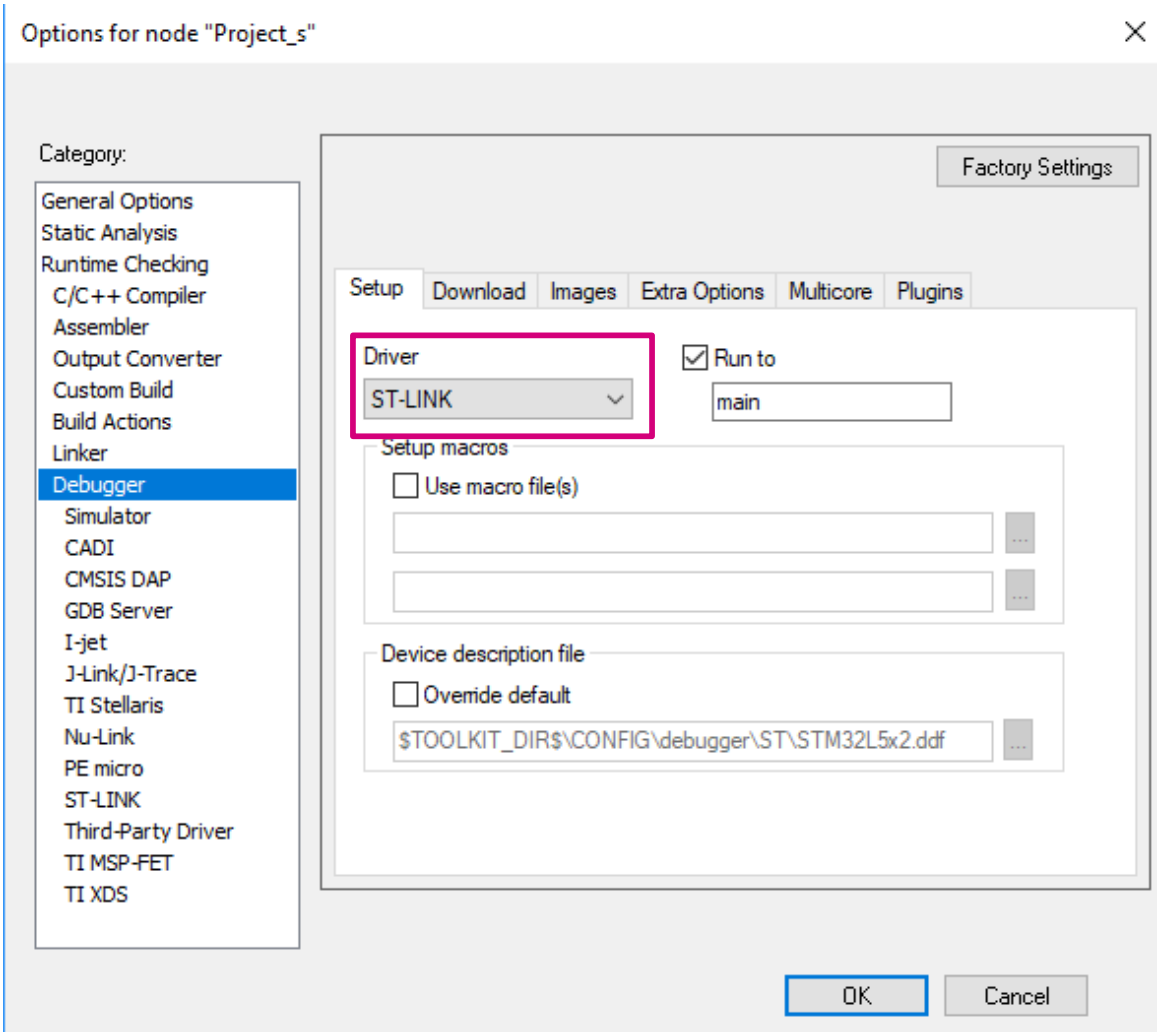
Figure 33. Linker configuration



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

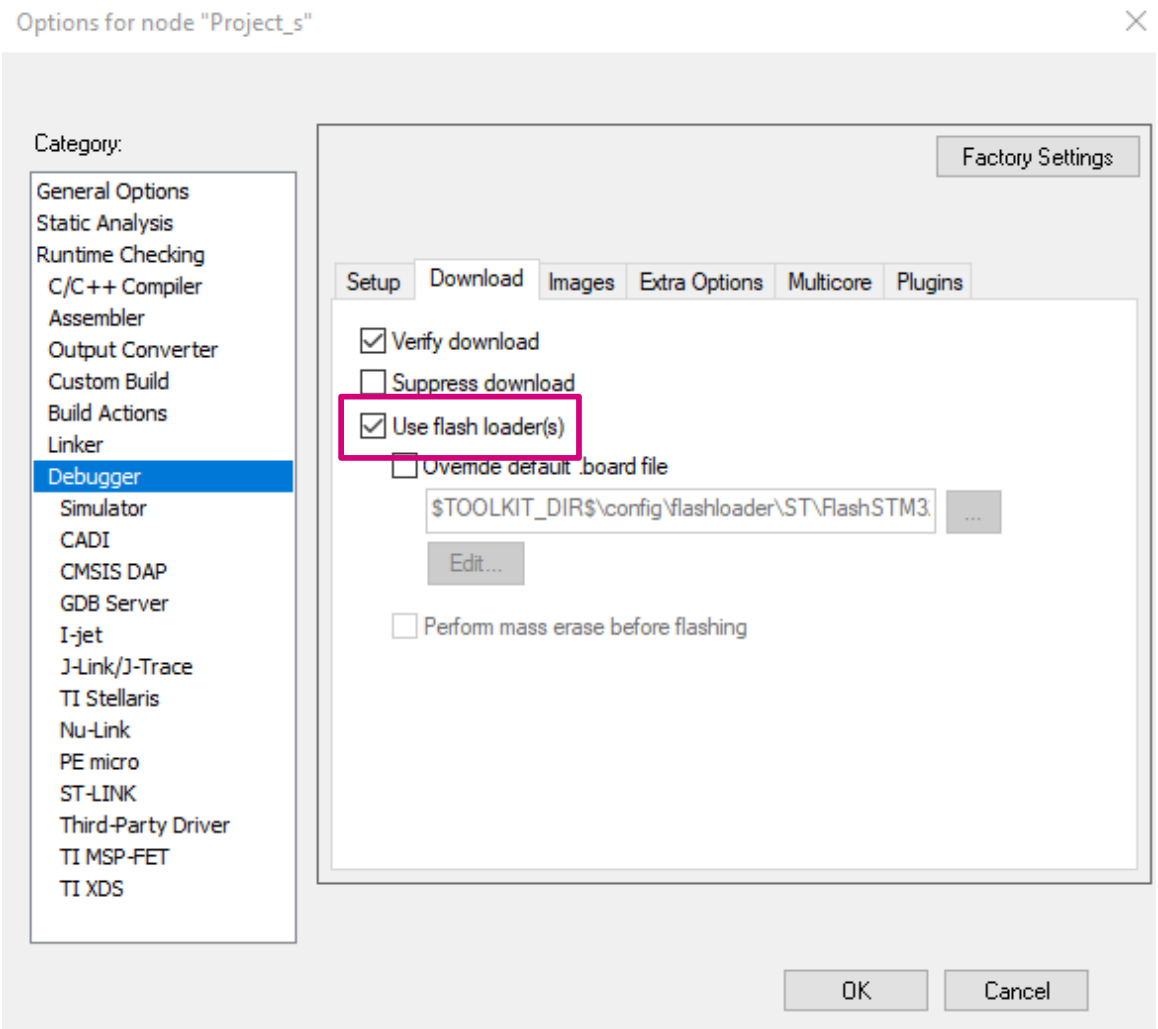
5. Open the debugger tab from: **Project / Options / Debugger**. From setup section, select ST-LINK as a debugger in the driver field (see Figure 34).

Figure 34. Project debugger setup



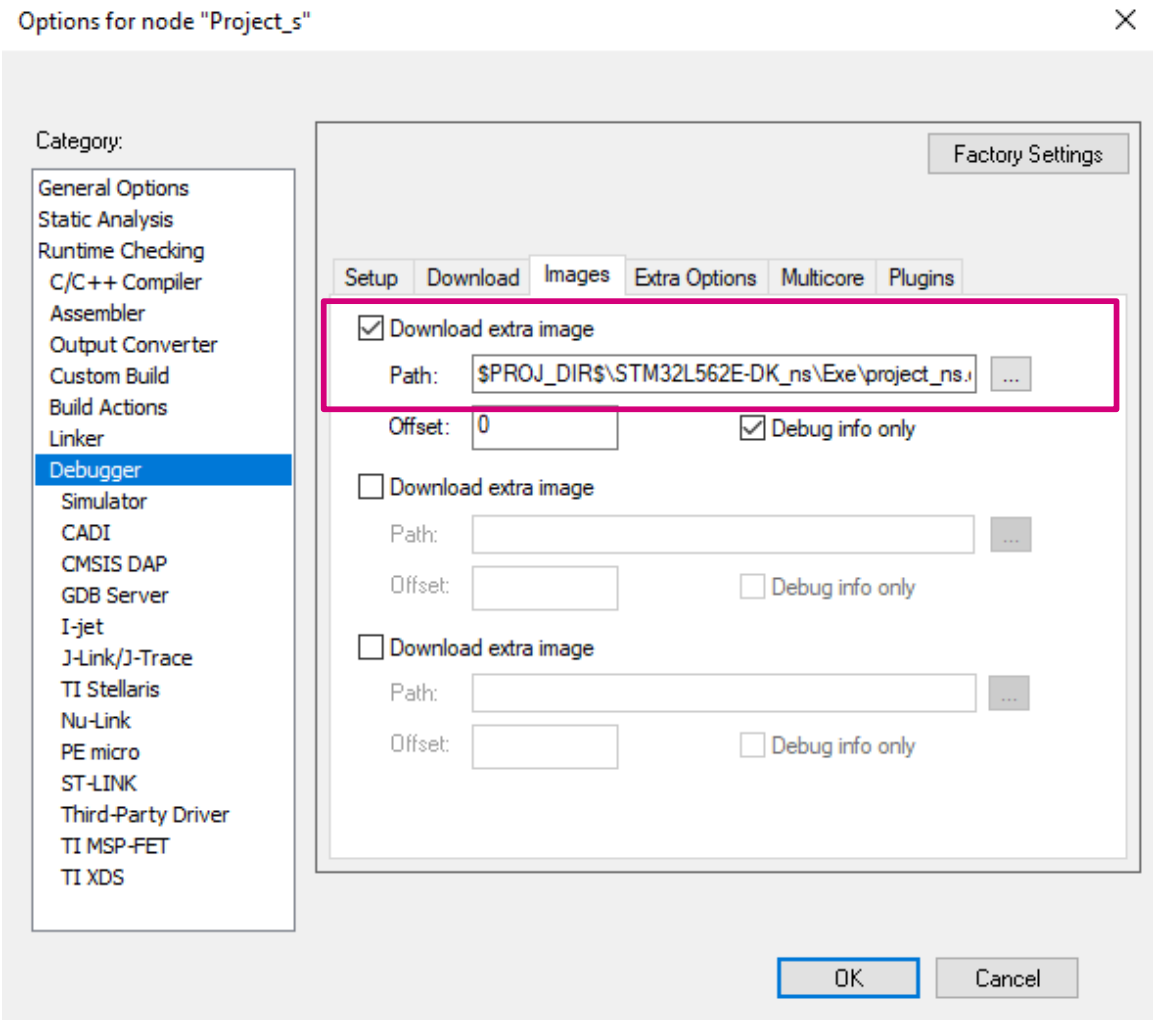
6. From the "Download" tab, ensure that "Use flash loader" is checked (see Figure 35).

Figure 35. FlashLoader selection



7. The secure project must specify the non-secure 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).

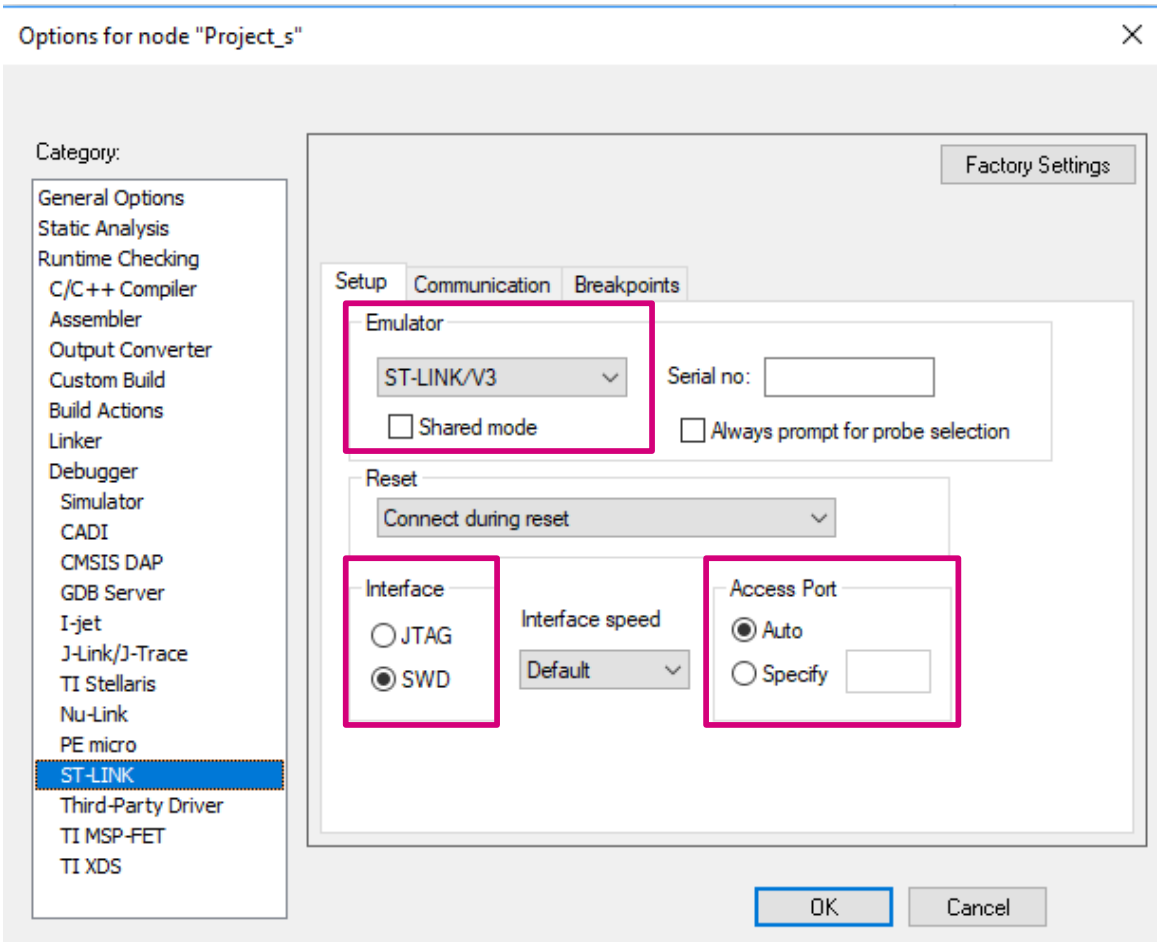
Figure 36. Selecting the non-secure output file as an extra image



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



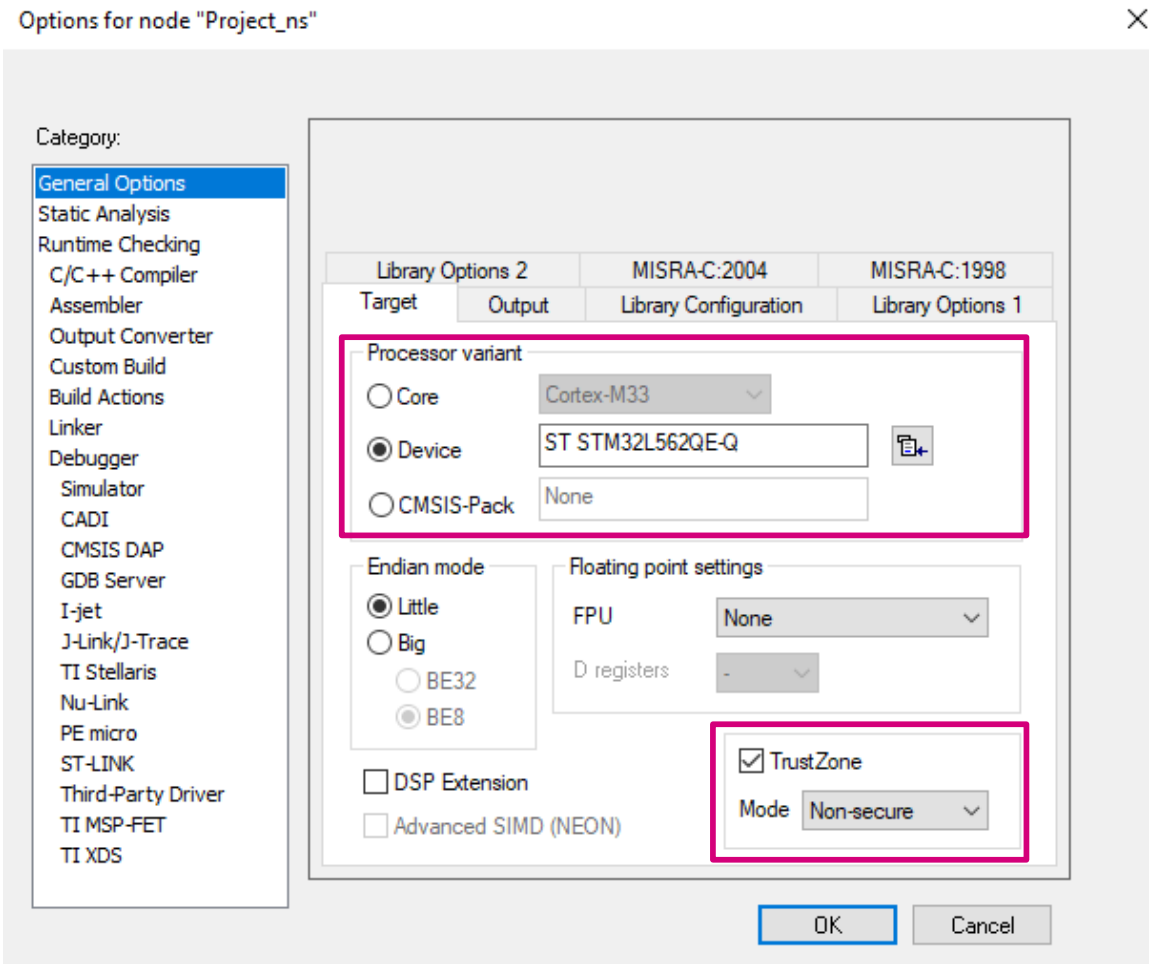
10.2 Non-secure project settings

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).

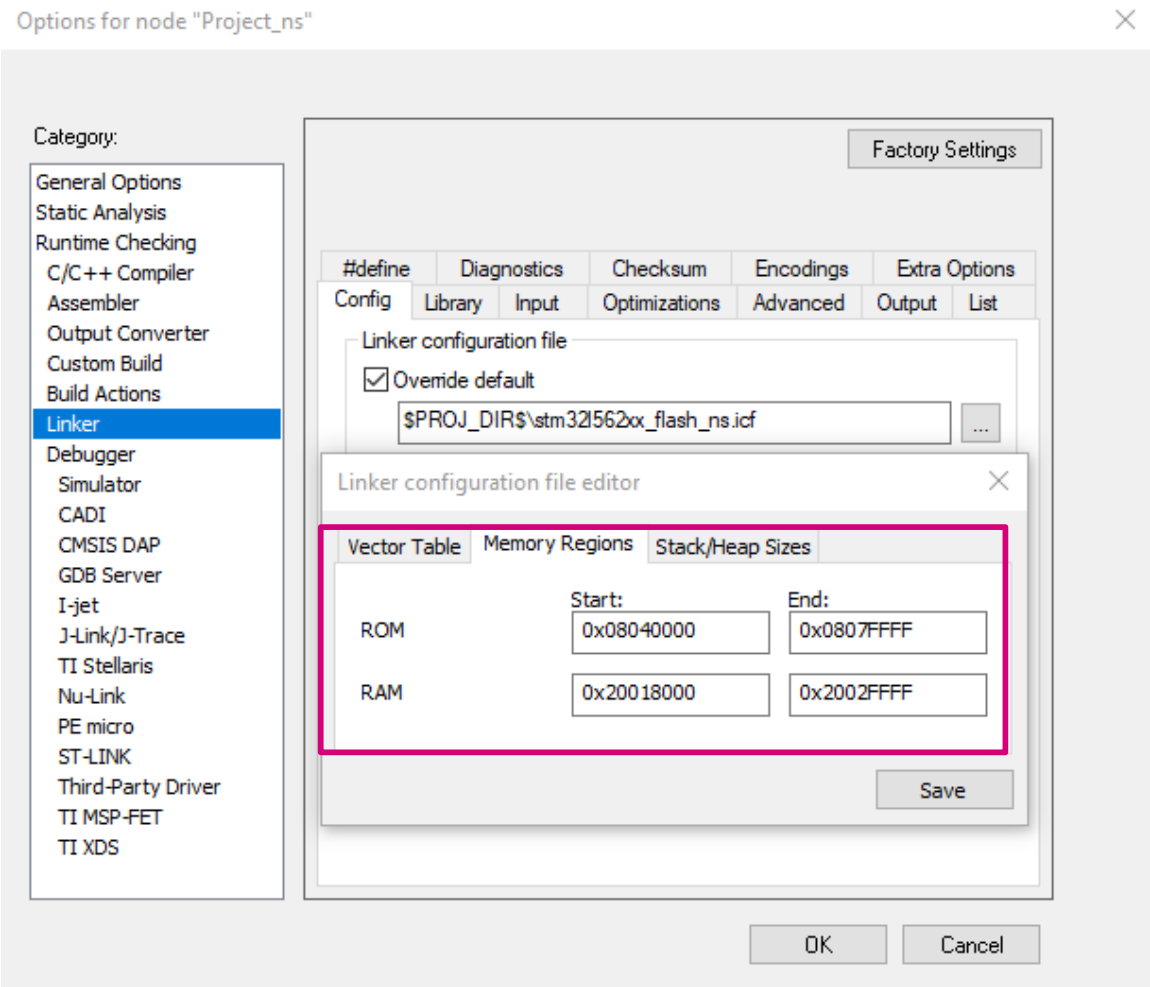
   From the TrustZone® section, ensure that the "Non-secure" mode is selected and the TrustZone® box is checked.

Figure 38. Project set up: general options



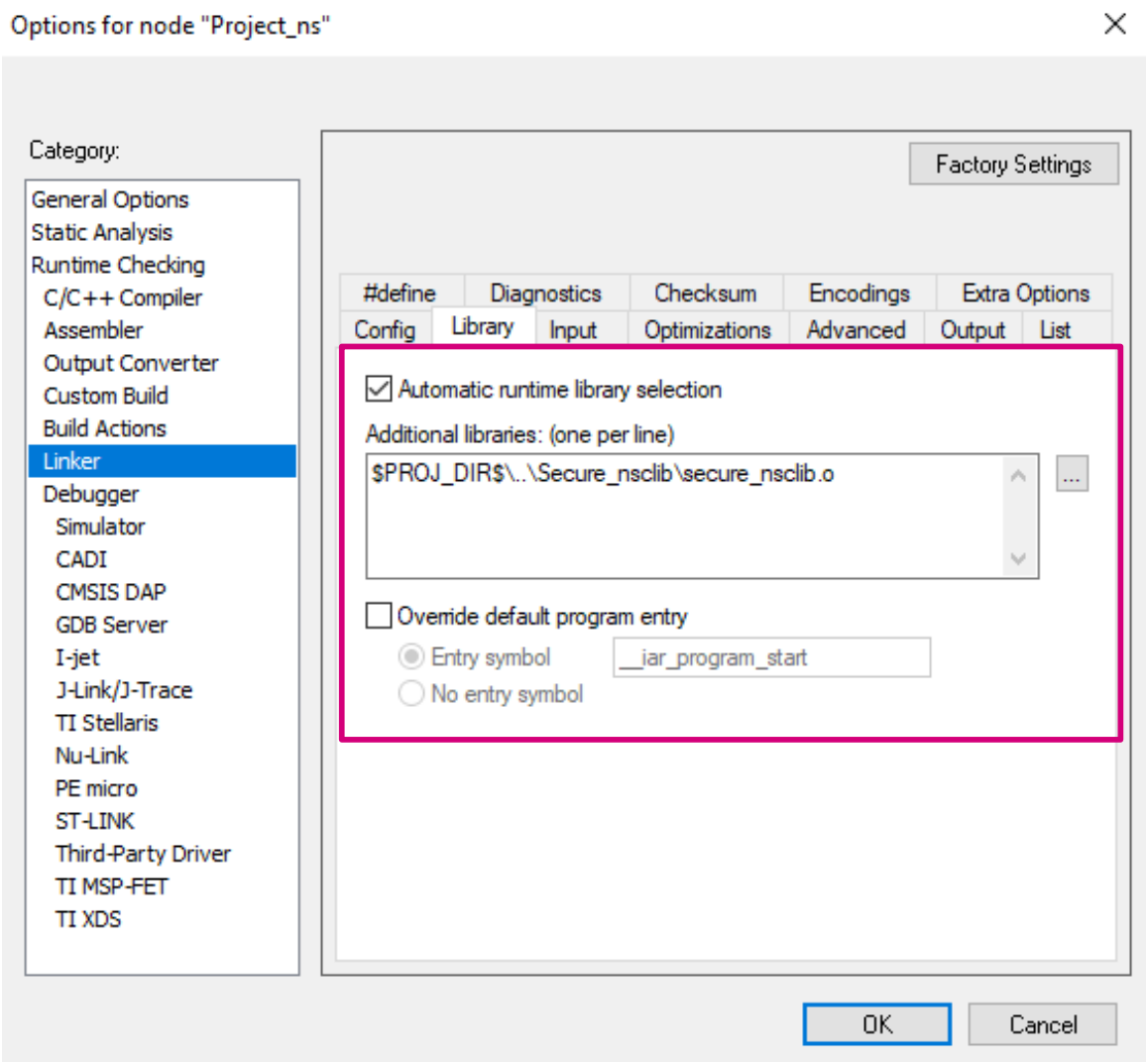
2. From **Project-s / Options / Linker / Linker configuration** file section (see Figure 39):
   – Click edit to display the linker configuration file editor.
   – Check the linker configuration file to make sure that the application has been linked to the right address:
     ◦ Boot address 0: Flash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

Figure 39. Project linker configuration



3. 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 non-secure project. It allows the non-secure part to call functions of the secure part.

Figure 40. Linker library setup



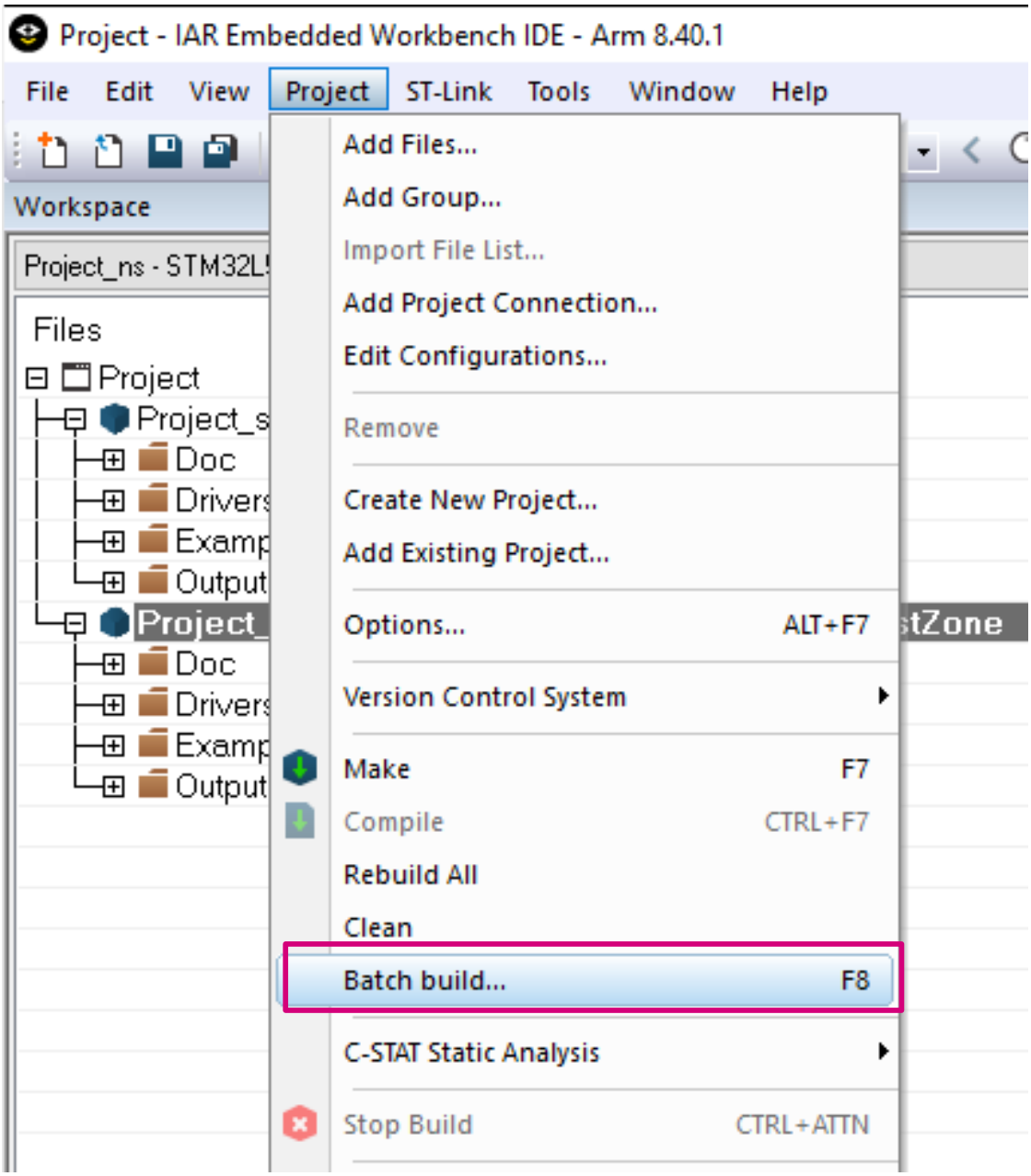
4. The other configurations are similar to the secure project.

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).

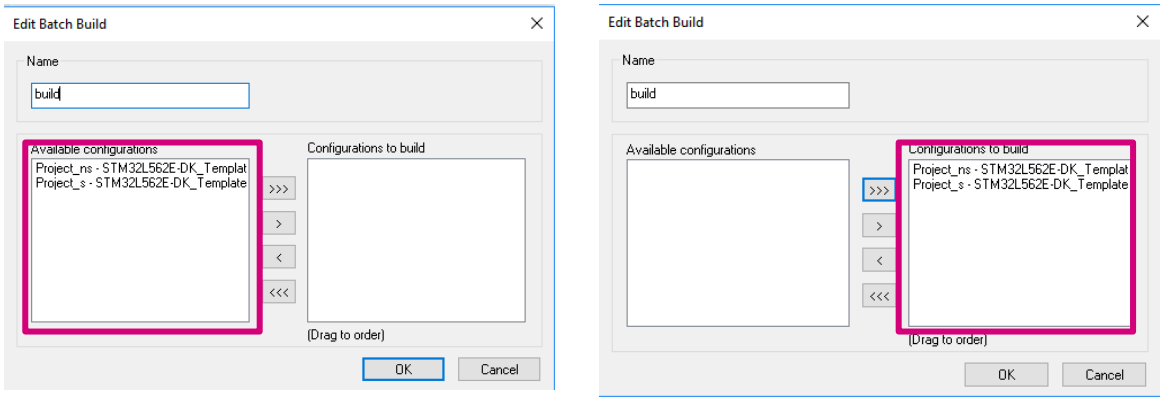
Figure 41. Project batch build



2. Add the two configuration 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 non-secure project. In order to build the secure project before the non-secure one, it must be first in the build order as illustrated below.

Figure 42. Project batch build order

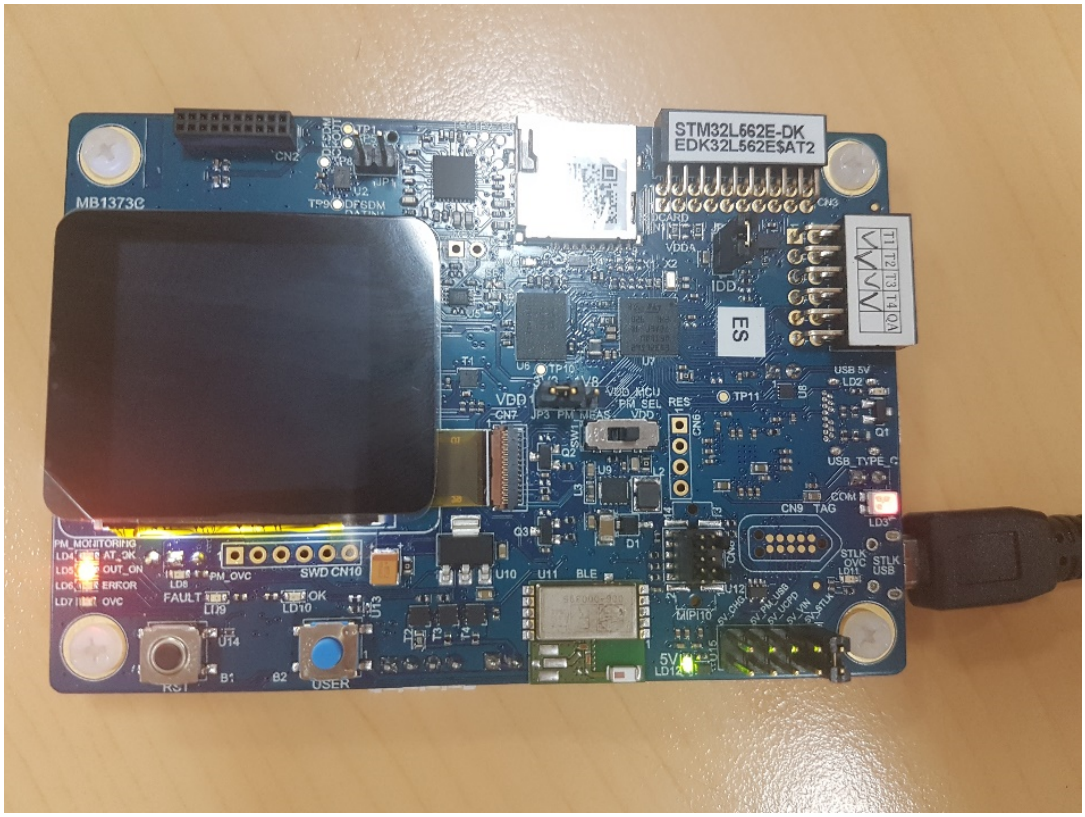


10.4 Execute from secure code to non-secure code

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

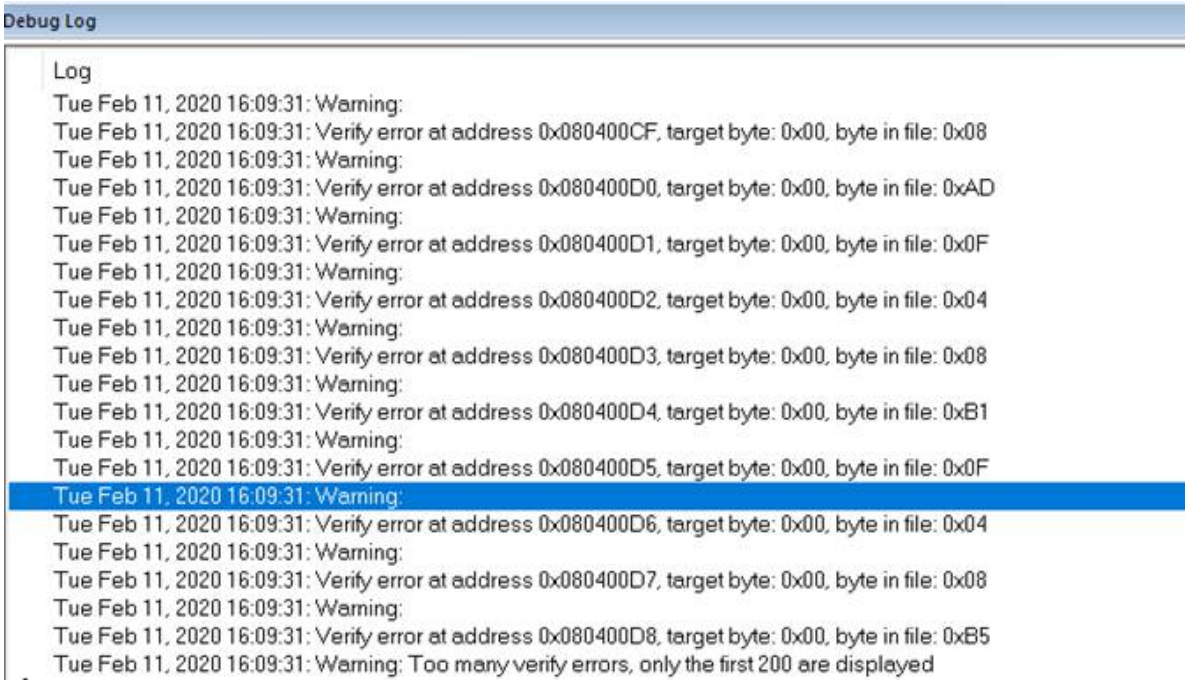


2. Select the Project_ns project as active project then load the non-secure binary code.
   Start a debug session by clicking the download and debug button in the toolbar to program the flash memory and start debugging (see Figure 44).

Figure 44. Download and debug launch button

Note: when trying to load the non-secure application, the following warning messages are displayed.

Figure 45. Non-secure application loading warning error message samples



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.

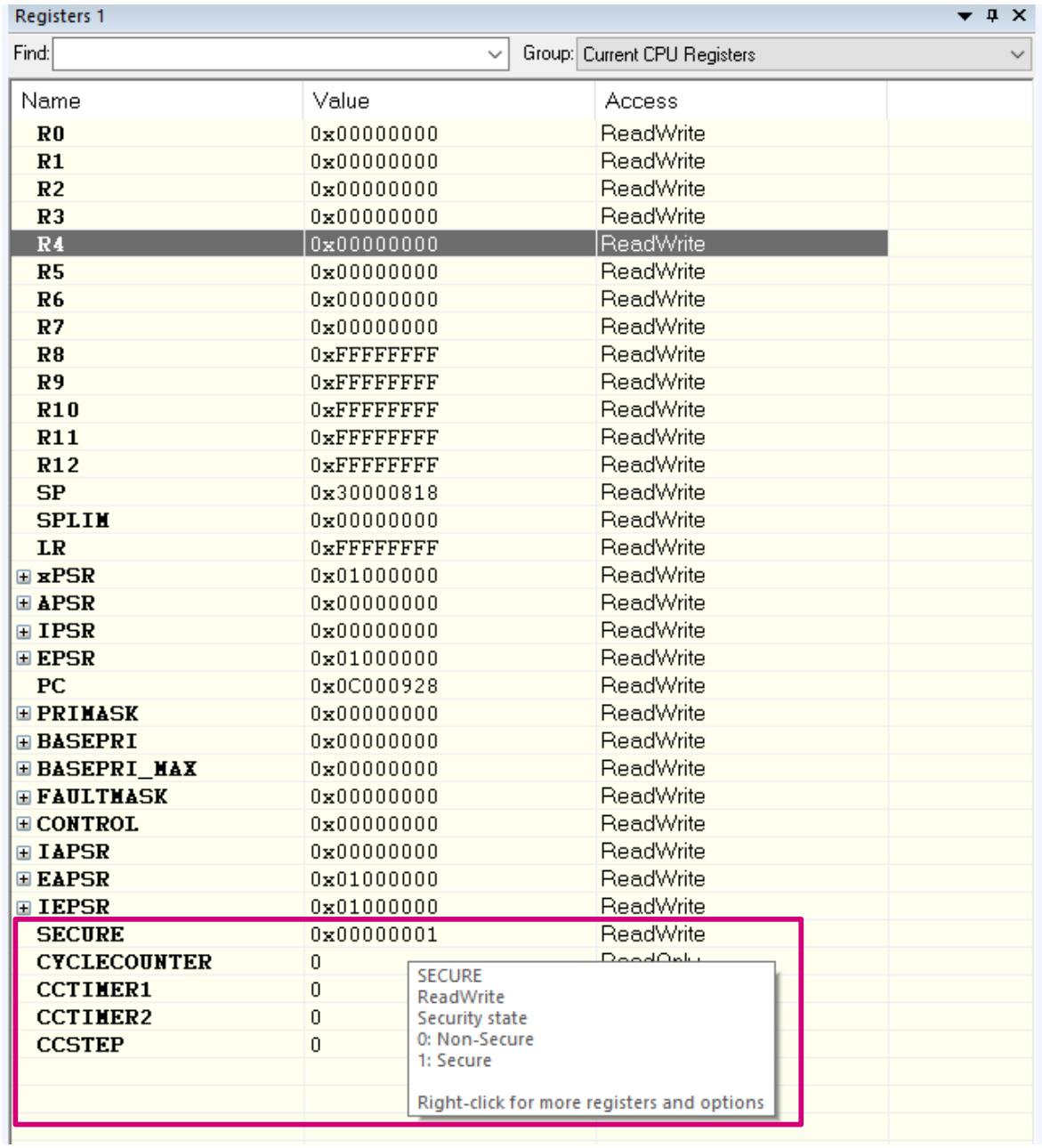
The debugger generates a secure transaction in a non-secure regions (@ 0x08040000 non secure flash) before SAU configuration. This access is not allowed and the content reads zero.

3. Select the Project_s project as active project then load the non-secure 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 non-secure application

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

Figure 46. Secure register location



0 = non-secure

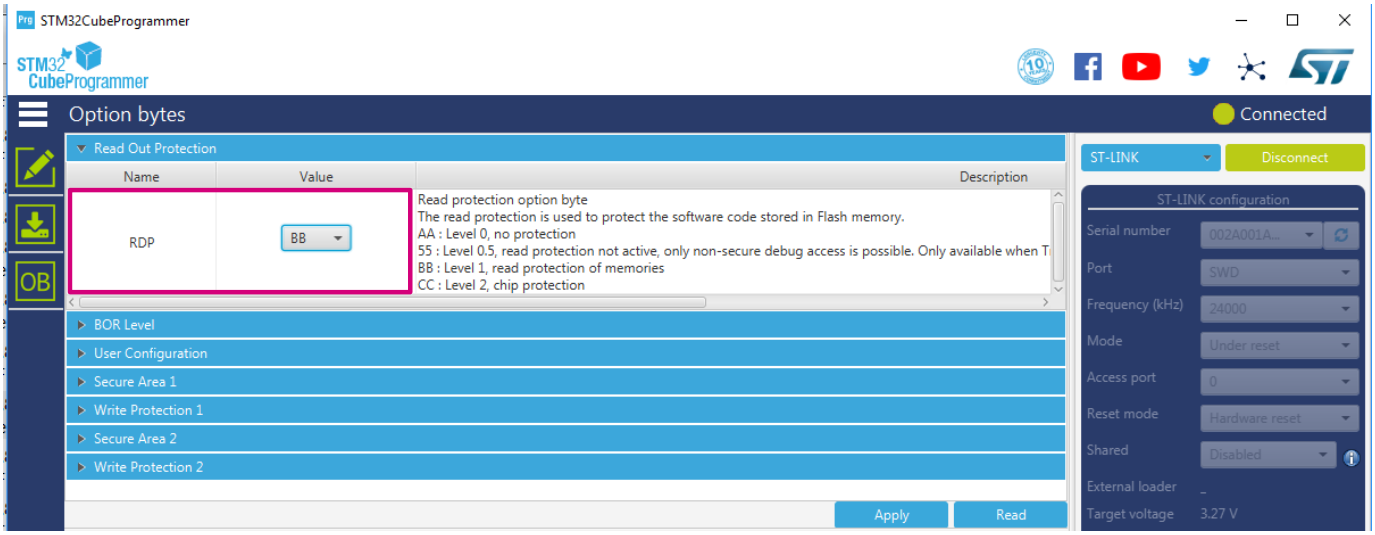
1 = secure

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 non-secure application. To connect to the STM32L552ZE-Q, proceed as follows:

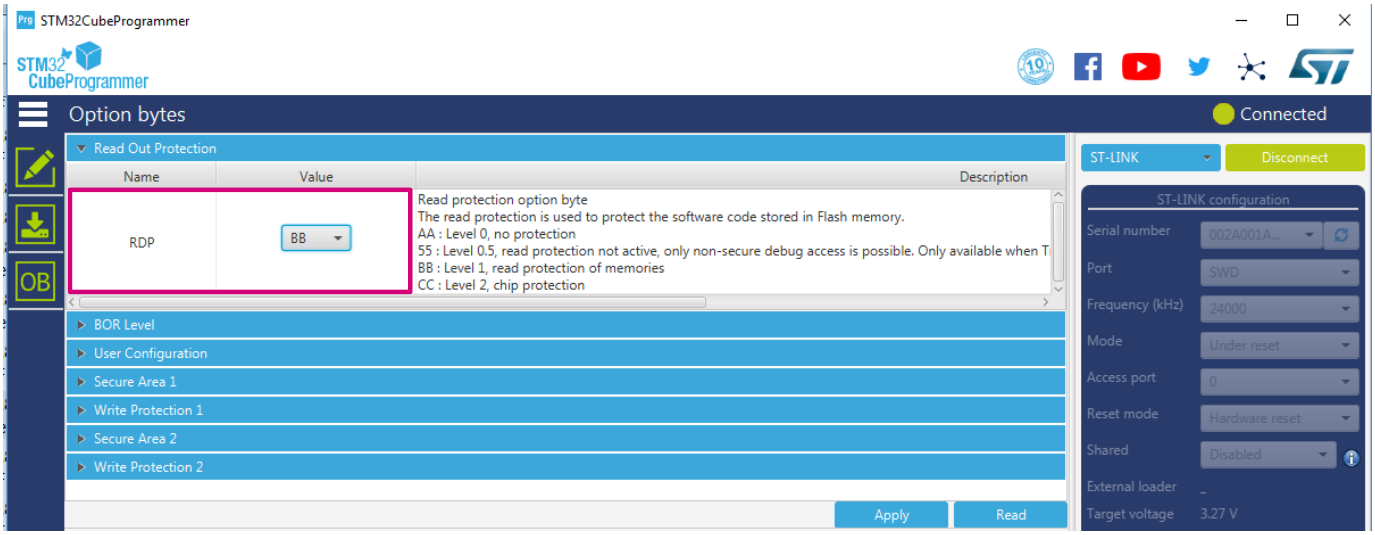
1. Setting the option bytes, illustrated in Figure 47:
   - TZEN = 1
   - DBANK = 1
   - SECWM2_STRT = 0x1
   - SECWM1_PEND = 0x0.

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



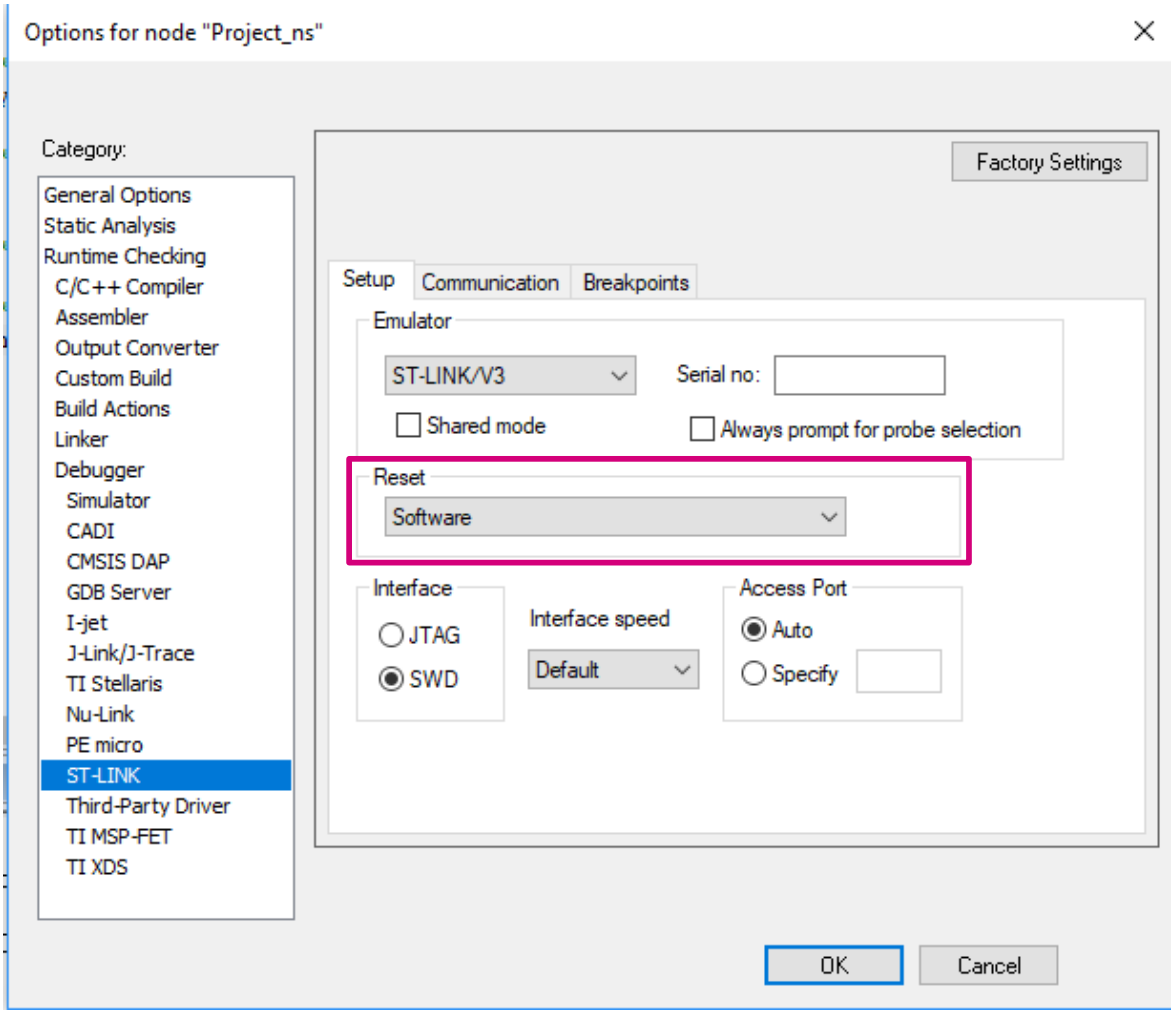
2. Load the non-secure binary (at 0x08040000) then load the secure binary (at 0x0C000000) as specified in the section above.
3. Using STM32CubeProgrammer to set RDP=0x55 to reduce debug to non-secure (see Figure 48).

Figure 48. RDP=0.5



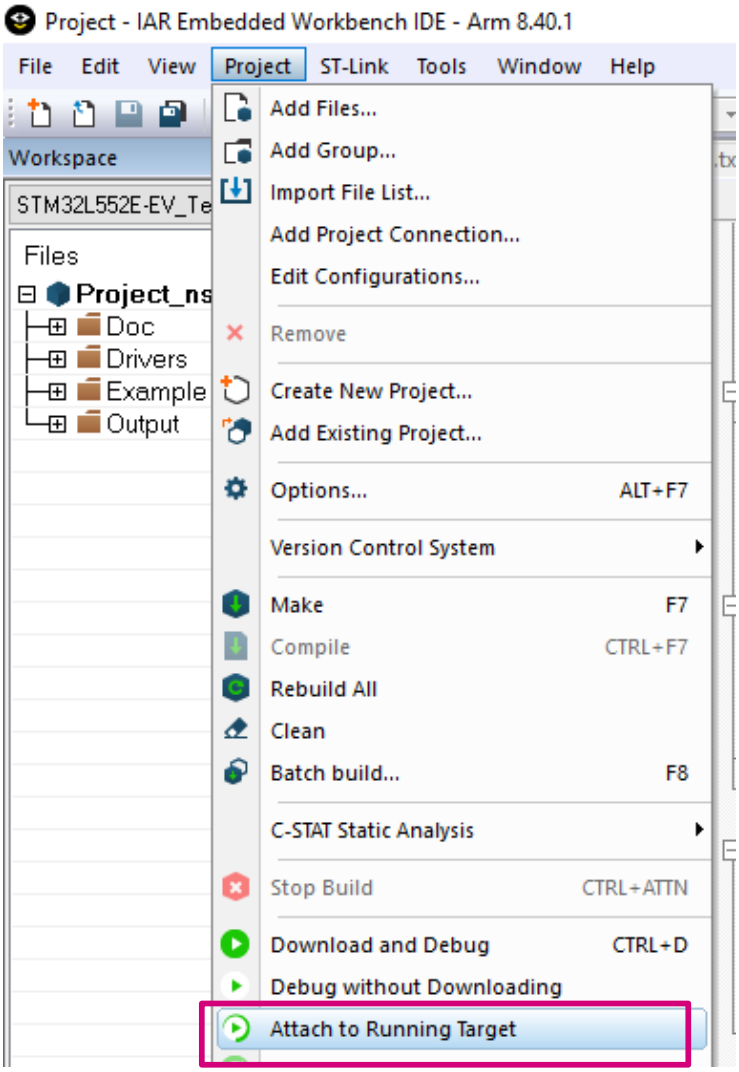
4. Change the Reset mode to software reset: **Project options / ST-LINK** in the "Setup" tab select "Software" from the "Reset" field as illustrated in Figure 49.

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 non-secure 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.

Contents

List of figures