UM1718
User manual

STM32CubeMX for STM32 configuration and initialization C code generation

Introduction

STM32CubeMX is a graphical tool for STM32 products. It is part of the STM32Cube initiative (see *Section 1: STM32Cube overview*), and is available as a standalone application as well as in the STM32CubeIDE toolchain.

STM32CubeMX has the following key features:

- **Easy microcontroller selection** covering the whole STM32 portfolio
- **Board selection** from a list of STMicroelectronics boards
- **Easy microcontroller configuration** (pins, clock tree, peripherals, middleware) and generation of the corresponding initialization C code
- **Easy switching to another microcontroller** by importing a previously-saved configuration to a new MCU project
- **Easy exporting of current configuration to a compatible MCU**
- **Generation of configuration reports**
- **Generation of embedded C projects** for a selection of integrated development environment tool chains (STM32CubeMX projects include the generated initialization C code, MISRA 2004 compliant STM32 HAL drivers, the middleware stacks required for the user configuration, and all the relevant files for opening and building the project in the selected IDE)
- **Power consumption calculation** for a user-defined application sequence
- **Self-updates** allowing the user to keep STM32CubeMX up-to-date
- Download and update of STM32Cube embedded software required for user application development (see *Appendix D* for details on the STM32Cube embedded software offer)
- Download of CAD resources (schematic symbols, PCB footprints, and 3D models)

Although STM32CubeMX offers a user interface and generates C code compliant with STM32 MCU design and firmware solutions, users need to refer to the product technical documentation for details on actual implementation of peripherals and firmware. The following documents are available on *www.st.com*:

- STM32 microcontroller reference manuals and datasheets
- STM32Cube HAL/LL driver user manuals for STM32C0 (UM2985), STM32F0 (UM1785), STM32F1 (UM1850), STM32F2 (UM1940), STM32F3 (UM1786), STM32F4 (UM1725), STM32F7 (UM1905), STM32G0 (UM2303), STM32G4 (UM2570), STM32H5 (UM3132), STM32H7 (UM2217), STM32H7RS (UM3309), STM32L0 (UM1749), STM32L1 (UM1816), STM32L4/L4+ (UM1884), STM32L5 (UM2659), STM32MP1/MP2 (https://wiki.st.com/stm32mpu/wiki/STM32CubeMX_release_note), STM32N6 (UM3425), STM32U0 (UM3307), STM32U5 (UM2883), STM32WL (UM2642), STM32WB (UM2442), STM32WBA (UM3131), and STM32WB0 (UM3363).

Contents

List of tables

List of figures

1 STM32Cube overview

STM32Cube is an STMicroelectronics original initiative to improve designer productivity significantly by reducing development effort, time, and cost. STM32Cube covers the whole portfolio of STM32 devices, based on 32-bit Arm®[(a)] Cortex® cores.

STM32Cube includes:

- A set of user-friendly software development tools to cover project development from conception to realization, among which are:
  - STM32CubeMX, a graphical software configuration tool that allows the automatic generation of C initialization code using graphical wizards
  - STM32CubeIDE, an all-in-one development tool with peripheral configuration, code generation, code compilation, and debug features
  - STM32CubeCLT, an all-in-one command-line development toolset with code compilation, board programming, and debug features
  - STM32CubeProgrammer (STM32CubeProg), a programming tool available in graphical and command-line versions
  - STM32CubeMonitor (STM32CubeMonitor, STM32CubeMonPwr, STM32CubeMonRF, STM32CubeMonUCPD), powerful monitoring tools to fine-tune the behavior and performance of STM32 applications in real time
- STM32Cube MCU and MPU Packages, comprehensive embedded-software platforms specific to each microcontroller and microprocessor series (such as STM32CubeH5 for the STM32H5 series), which include:
  - STM32Cube hardware abstraction layer (HAL), ensuring maximized portability across the STM32 portfolio
  - STM32Cube low-layer APIs, ensuring the best performance and footprints with a high degree of user control over hardware
  - A consistent set of middleware components, such as ThreadX, FileX / LevelX, NetX Duo, USBX, USB-PD, mbed-crypto, secure manager API, MCUboot, and OpenBL
  - All embedded software utilities with full sets of peripheral and applicative examples
- STM32Cube Expansion Packages, which contain embedded software components that complement the functionalities of the STM32Cube MCU and MPU Packages with:
  - Middleware extensions and applicative layers
  - Examples running on some specific STMicroelectronics development boards

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

2 Getting started with STM32CubeMX

2.1 Principles

Customers need to quickly identify the MCU that best meets their requirements (core architecture, features, memory size, performance…). While board designers main concerns are to optimize the microcontroller pin configuration for their board layout and to fulfill the application requirements (choice of peripherals operating modes), embedded system developers are more interested in developing new applications for a specific target device, and migrating existing designs to different microcontrollers.

The time taken to migrate to new platforms and update the C code to new firmware drivers adds unnecessary delays to the project. STM32CubeMX was developed within STM32Cube initiative which purpose is to meet customer key requirements to maximize software reuse and minimize the time to create the target system:

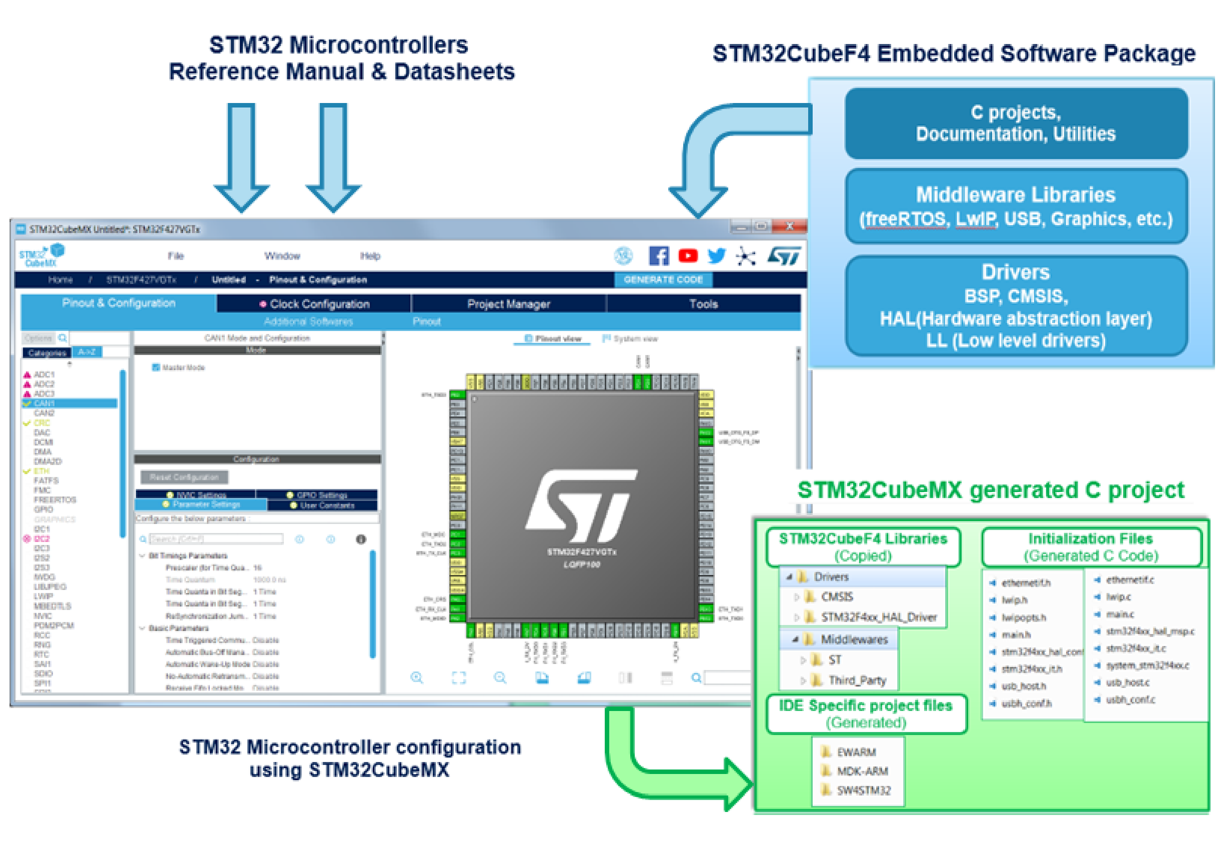
- Software reuse and application design portability are achieved through STM32Cube firmware solution proposing a common Hardware Abstraction Layer API across STM32 portfolio.
- Optimized migration time is achieved thanks to STM32CubeMX built-in knowledge of STM32 microcontrollers, peripherals and middleware (LwIP and USB communication protocol stacks, FatFs file system for small embedded systems, FreeRTOS).

STM32CubeMX graphical interface performs the following functions:

- Fast and easy configuration of the MCU pins, clock tree and operating modes for the selected peripherals and middleware
- Generation of pin configuration report for board designers
- Generation of a complete project with all the necessary libraries and initialization C code to set up the device in the user defined operating mode. The project can be directly open in the selected application development environment (for a selection of supported IDEs) to proceed with application development (see *Figure 1*).

During the configuration process, STM32CubeMX detects conflicts and invalid settings and highlights them through meaningful icons and useful tool tips.

Figure 1. Overview of STM32CubeMX C code generation flow



2.2 Key features

STM32CubeMX comes with the following features:

- **Project management**

  STM32CubeMX allows the user to create, save, and load previously saved projects:
  - When STM32CubeMX is launched, the user can choose to create a new project or to load a previously saved project.
  - Saving the project saves user settings and configuration performed within the project in an .ioc file to be used when the project will be loaded in STM32CubeMX again.

  STM32CubeMX also allows the user to import previously saved projects in new ones.

  STM32CubeMX projects come in two flavors:
  - MCU configuration only: .ioc file is saved in a dedicated project folder.
  - MCU configuration with C code generation: in this case .ioc files are saved in a dedicated project folder along with the generated source C code. There can be only one .ioc file per project.
- **Easy project creation starting from an MCU, a board, or an example**

  The new project window allows the user to create a project by selecting a microcontroller, a board, or an example project from STMicroelectronics STM32 portfolio. Different filtering options are available to ease the MCU and board selection. There is also the possibility to select an MCU through the Cross selector tab by comparing characteristics to those of competitors. Comparison criteria can be adjusted.
- **Easy pinout configuration**
  - From the **Pinout** view, the user can select the peripherals from a list and configure the peripheral modes required for the application. STM32CubeMX assigns and configures the pins accordingly.
  - For more advanced users, it is also possible to directly map a peripheral function to a physical pin using the **Pinout** view. The signals can be locked on pins to prevent STM32CubeMX conflict solver from moving the signal to another pin.
  - Pinout configuration can be exported as a .csv file.
- **Complete project generation**

  The project generation includes pinout, firmware and middleware initialization C code for a set of IDEs. It is based on STM32Cube embedded software libraries. The following actions can be performed:
  - Starting from the previously defined pinout, the user can proceed with the configuration of middleware, clock tree, services (such as RNG, CRC) and peripheral parameters. STM32CubeMX generates the corresponding initialization C code. The result is a project directory including generated main.c file and C header files for configuration and initialization, plus a copy of the necessary HAL and middleware libraries as well as specific files for the selected IDE.
  - The user can modify the generated source files by adding user-defined C code in user dedicated sections. STM32CubeMX ensures that the user C code is preserved upon next C code generation (the user C code is commented if no longer relevant for the current configuration).
  - STM32CubeMX can generate user files by using user-defined freemarker .ftl template files.

- From the **Project Settings** menu, the user can select the development toolchain (IDE) for which the C code has to be generated. STM32CubeMX ensures that the IDE relevant project files are added to the project folder so that the project can be directly imported as a new project within STM32Cube or third party IDEs (IAR™ EWARM, Keil™ MDK-Arm, KITWARE™ CMake, FSF™ Makefile).

- **Power consumption calculation**

  Starting with the selection of a microcontroller part number and a battery type, the user can define a sequence of steps representing the application life cycle and parameters (choice of frequencies, enabled peripherals, step duration). STM32CubeMX Power Consumption Calculator returns the corresponding power consumption and battery life estimates.

- **Clock tree configuration**

  STM32CubeMX offers a graphic representation of the clock tree as it can be found in the device reference manual. The user can change the default settings (clock sources, prescaler and frequency values). The clock tree is then updated accordingly. Invalid settings and limitations are highlighted and documented with tool tips. Clock tree configuration conflicts can be solved by using the solver feature. When no exact match is found for a given user configuration, STM32CubeMX proposes the closest solution.

- **Automatic updates of STM32CubeMX and STM32Cube MCU packages**

  STM32CubeMX comes with an updater mechanism that can be configured for automatic or on-demand check for updates. It supports self-updates as well as firmware library package updates. The updater mechanism also allows deleting previously installed packages.

- **Report generation**

  .pdf and .csv reports can be generated to document the user configuration work.

- **Support of embedded software packages in CMSIS-Pack format (Software Packs)**

  STM32CubeMX allows getting and downloading updates of embedded software packages delivered in CMSIS-Pack format. Selected software components belonging to these new releases can then be added to the current project.

- **Generating Software Packs with STM32PackCreator**

  STM32PackCreator is a graphical tool installed with STM32CubeMX in the Utilities folder. It allows the user to create Software Packs and STM32Cube Expansion packages enhanced for STM32CubeMX. It can be launched from the ST Tools tab found in the Tools view.

- **Contextual help**

  Contextual help windows can be displayed by hovering the mouse over Cores, Series, Peripherals and Middleware. They provide a short description and links to the relevant documentation corresponding to the selected item.

- **Access to ST tools**

  From STM32CubeMX project, the Tools tab allows the user to launch Tools directly or to access tools download pages on *www.st.com*.

- **Video tutorials**

  STM32CubeMX allows the user to browse and play video tutorials. The video tutorial browser is accessible from the Help menu. The tutorials can be accessed through multiple locations, including the Help menu and the following areas:

  a) STM32CubeMX Main Tabs: within the Pinout & Configuration, Clock Configuration, Project Manager, and Tools tabs.

  b) MCU Finder Tabs within MPU/MCU Selector, Board Selector, Example Selector, Cross Selector

  c) Embedded Software Package Manager window: specifically from any software expansion packages (X-CUBE packages) developed by STMicroelectronics, with an associated video tutorial

  d) Specific IPs: directly from any peripheral (IP) with an associated video tutorial.

2.3 Rules and limitations

- C code generation covers only peripheral and middleware initialization. It is based on STM32Cube HAL firmware libraries.
- STM32CubeMX C code generation covers only initialization code for peripherals and middleware components that use the drivers included in STM32Cube embedded software packages. The code generation of some peripherals and middleware components is not yet supported.
- Refer to *Appendix A* for a description of pin assignment rules.
- Refer to *Appendix B* for a description of STM32CubeMX C code generation design choices and limitations.

3 Installing and running STM32CubeMX

3.1 System requirements

3.1.1 Supported operating systems and architectures

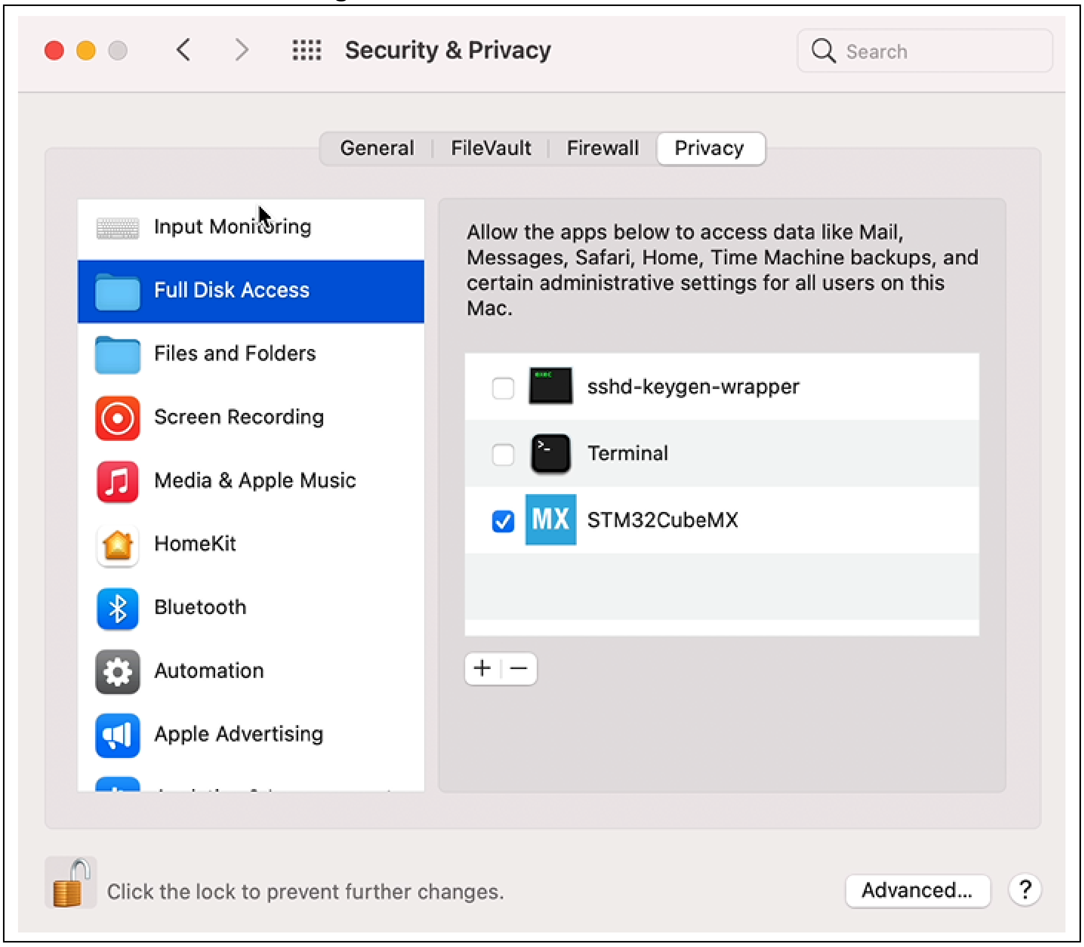
- Windows® 10 32 bits (x86) or 64 bits (x64), and Windows® 11 64 bits (x64)
- Linux®: Ubuntu® LTS 22.04 and LTS 24.04, and Fedora® 42
- macOS® 15 (Sequoia), macOS® 26 (Tahoe): x86_64 and ARM-aarch64 architectures

Note: Windows is a trademark of the Microsoft group of companies.
Linux® is a registered trademark of Linus Torvalds.
Ubuntu® is a registered trademark of Canonical Ltd.
Fedora® is a trademark of Red Hat, Inc.
macOS® is a trademark of Apple Inc., registered in the U.S. and other countries and regions.

For macOS the full disk access is required to load project files or install other packages from the file system. To enable full disk access for STM32CubeMX:

1. Go to “System preferences” and click to open “Security & Privacy” window (*Figure 2*)
2. Select “Privacy” tab
3. Select “Full Disk Access” from the left panel
4. Click the checkbox to enable full disk access to STM32CubeMX

Figure 2. Full disk access for macOS



3.1.2 Memory prerequisites

- Recommended minimum RAM: 2 Gbytes

3.1.3 Software requirements

If the initial installation was completed with administrator privileges, the user also needs these privileges to download and install the latest update package. Additionally, the user needs administrator rights to successfully apply the update at the next start of STM32CubeMX.

Java™ Runtime Environment

For STM32CubeMX 6.13 the bundled JRE is openJDK Runtime Environment Temurin™ 21.0.3+9 (build 21.0.3+9-LTS) and JavaFX-21.0.3.

Starting with version V6.2.0, STM32CubeMX embeds the Java Runtime Environment (JRE™[a]) required for its execution and no longer uses the one installed on the user machine.

- For STM32CubeMX 6.3 the bundled JRE is AdoptOpenJDK-11.0.10+9 and JavaFX-11.0.2
- For STM32CubeMX 6.2 the bundled JRE is Liberica 1.8.0_265 of BellSoft

Versions earlier than STM32CubeMX V6.2.0 require to install a JRE, whose constraints are:

- 64-bit version mandatory, 32-bit version not supported
- the STM32PackCreator companion tool requires JRE supporting JavaFX
- minimum JRE version is 1.8_45 (known limitation with 1.8_251)
- version 11 is supported, versions 7, 9, 10, 12 and upper are not supported

STMicroelectronics promotes the use of the following JREs:

- Oracle[a], subject to license fee
- Amazon Corretto™[a], no-cost solution based on OpenJDK, JDK installer recommended.

STM32CubeMX operation is not guaranteed with other JREs.

macOS software requirements

- Xcode must be installed on macOS computers

3.2 Installing/uninstalling STM32CubeMX standalone version

3.2.1 Installing STM32CubeMX standalone version

To install STM32CubeMX:

1. From an Internet browser, open the page www.st.com/stm32cubemx
2. Click “Get Software” to go to the software download section

On Windows

a) On STM32CubeMX-Win line, click “Get software” to download the package

b) Extract (unzip) the downloaded package

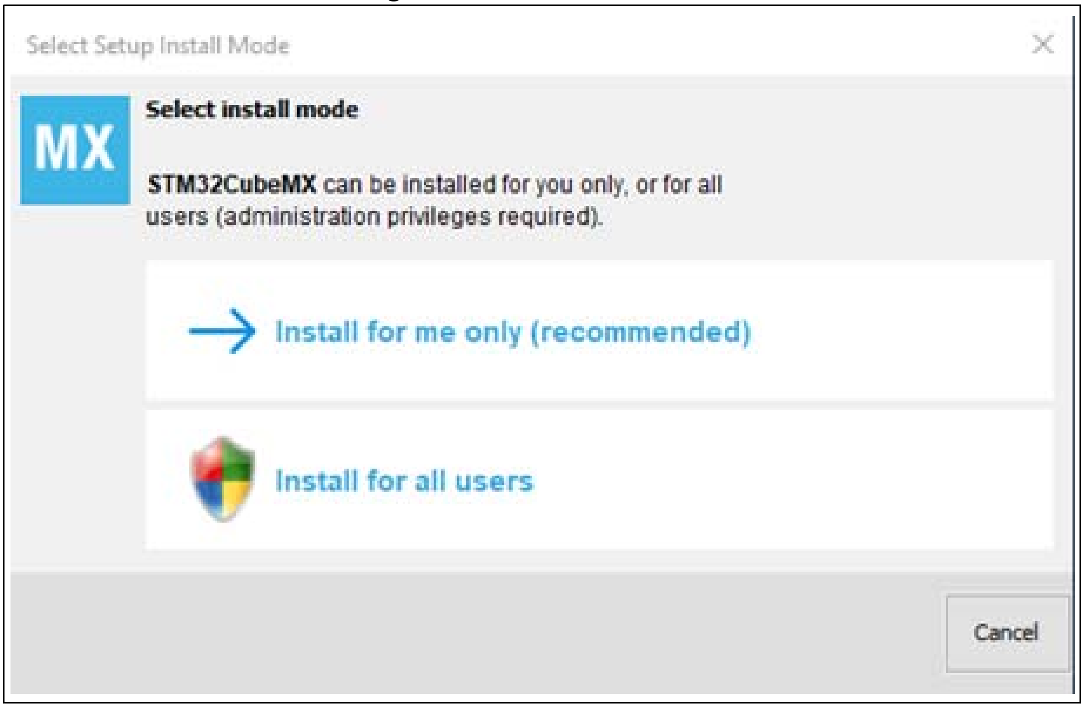
c) Double-click on SetupSTM32CubeMX-VERSION-Win.exe to launch the installation wizard

d) The installation wizard is displayed (see *Figure 3*), it gives the choice between two modes, namely “Install for all users”, and “Install for me only (recommended)”

a. Oracle and Java are registered trademarks of Oracle and/or its affiliates.

a. All other trademarks are the properties of their respective owners.

Figure 3. Select install mode



If you choose “Install for all users” mode:

- > Enter administrator credentials
- > Welcome panel (*Figure 4*)
- > License agreement (*Figure 5*)
- > Terms of use (*Figure 6*)
- > The default installation path is set to C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeMX (*Figure 7*)
- > The shortcuts for all users are created by default (*Figure 8*)
- > Package installation (*Figure 9*)
- > Installation script (*Figure 10*)

If you choose “Install for me only (recommended)” mode:

- > Welcome panel (*Figure 4*)
- > License agreement (*Figure 5*)
- > Terms of use (*Figure 6*)
- > The installation path is set on the home director by default (*Figure 11*): note that the default installation folder is, by default, a system hidden folder
- > The shortcut can be created only for the current user (*Figure 12*)
- > Package installation (*Figure 13*)
- > Installation script (*Figure 14*)

Figure 4. Welcome panel

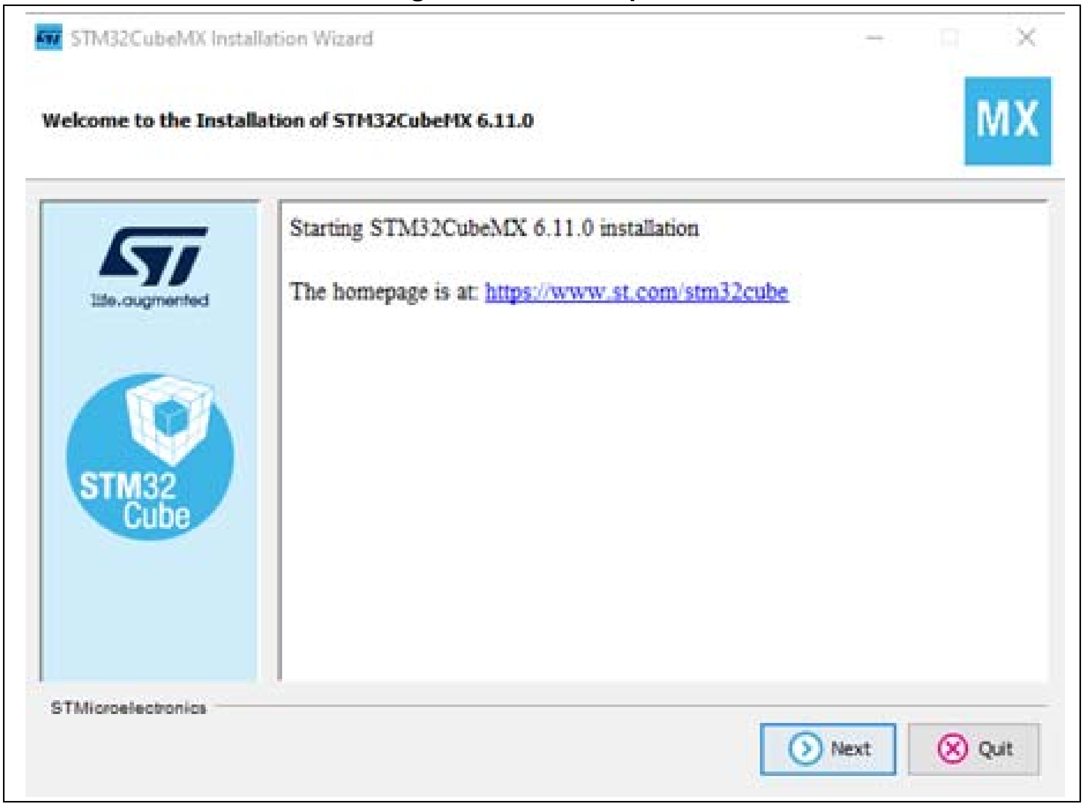


Figure 5. License agreement

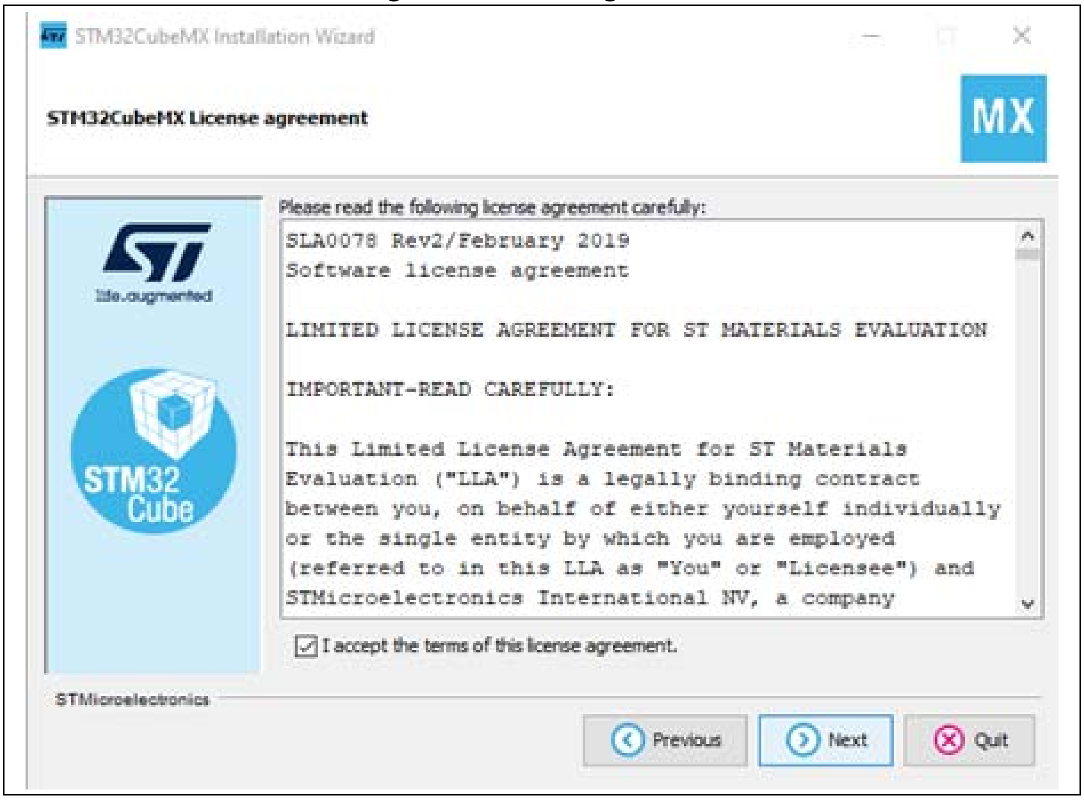


Figure 6. Terms of use

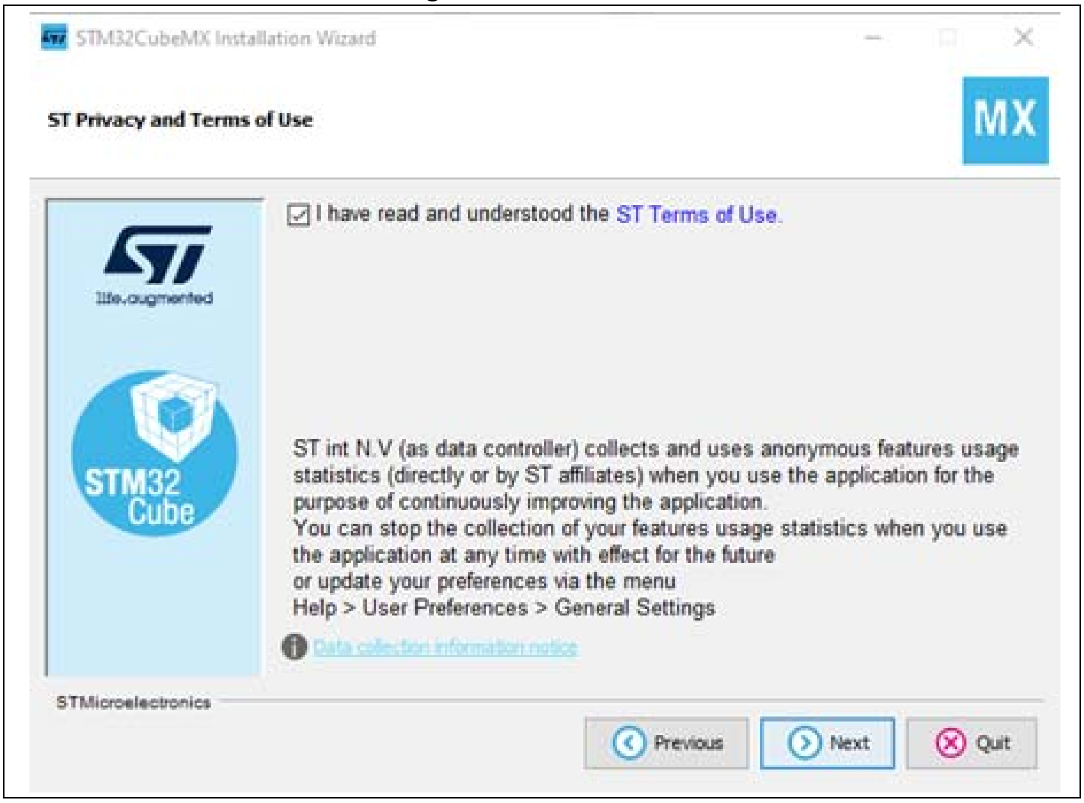


Figure 7. Default installation path

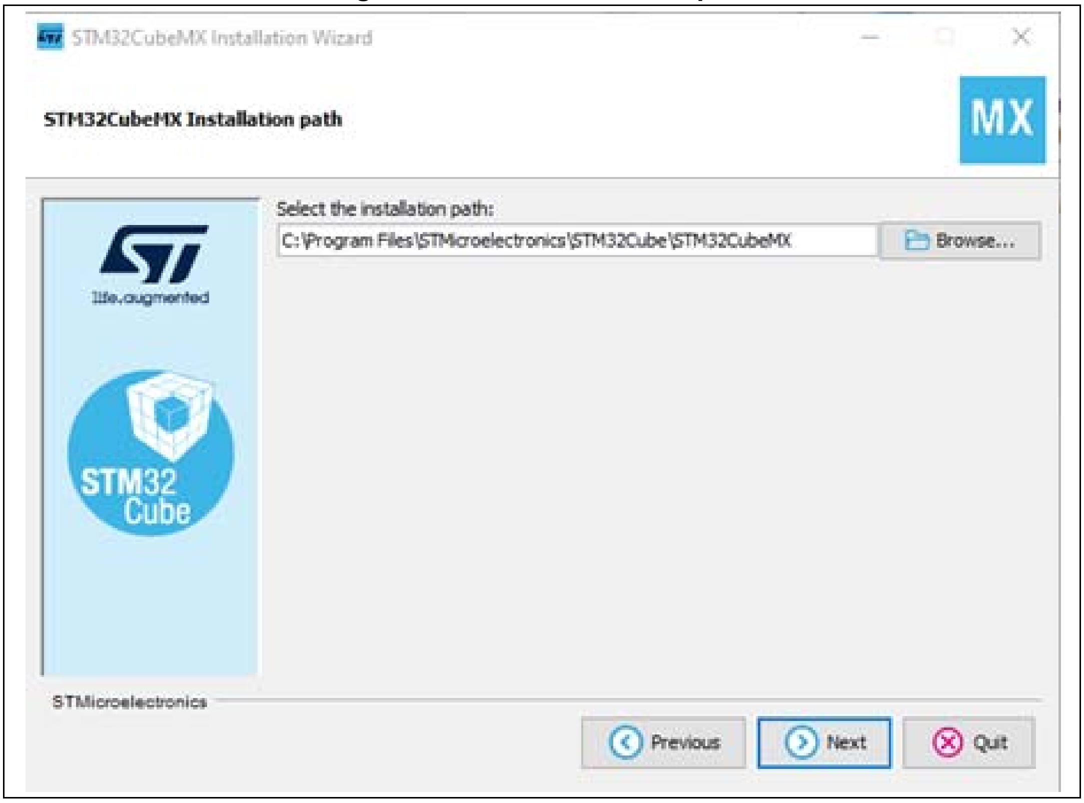


Figure 8. Setup of shortcuts

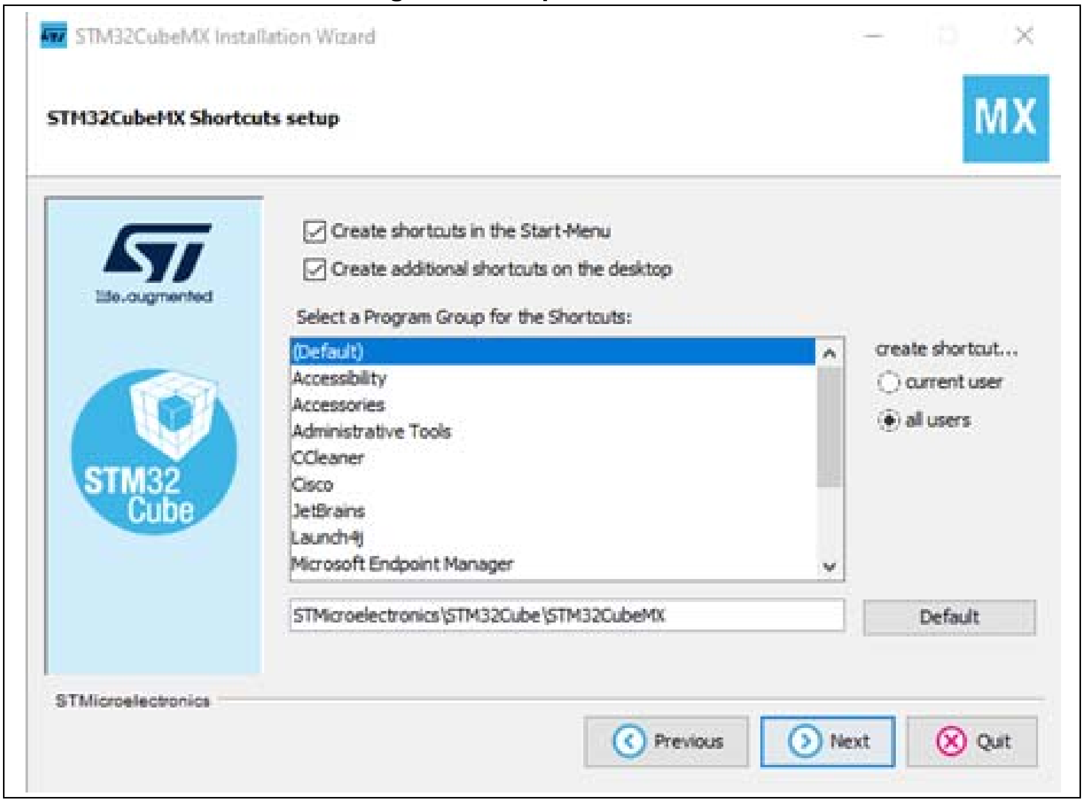


Figure 9. Package installation

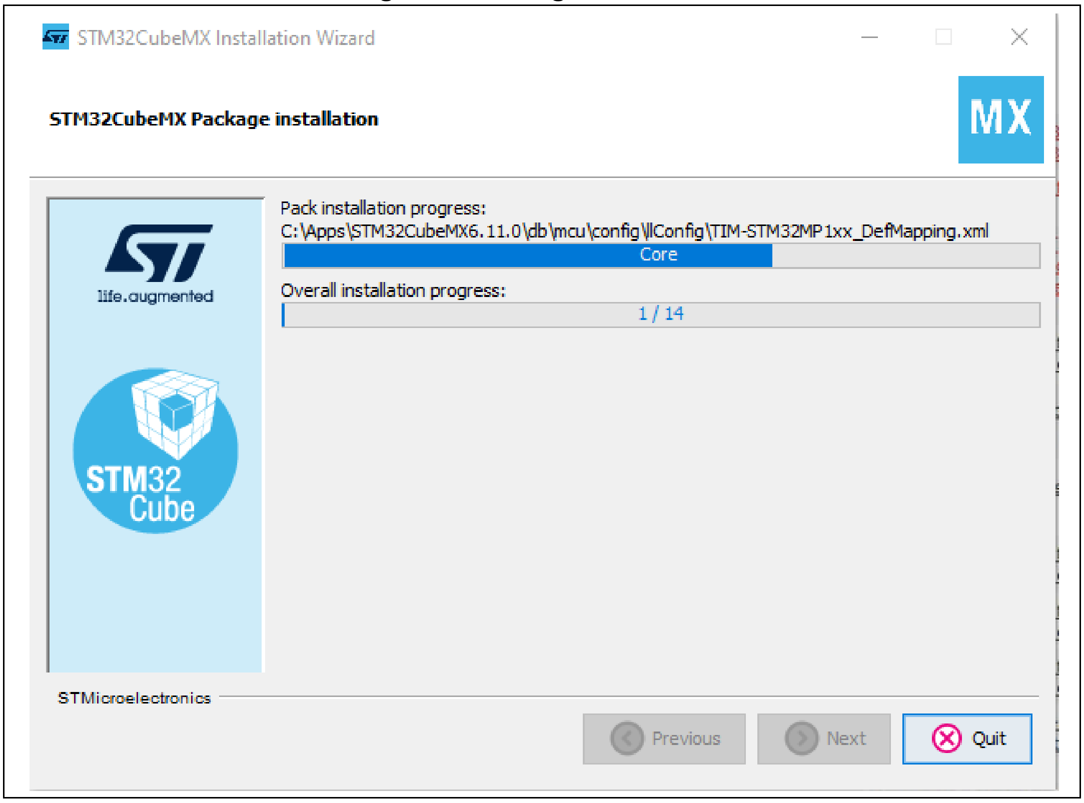


Figure 10. Installation script

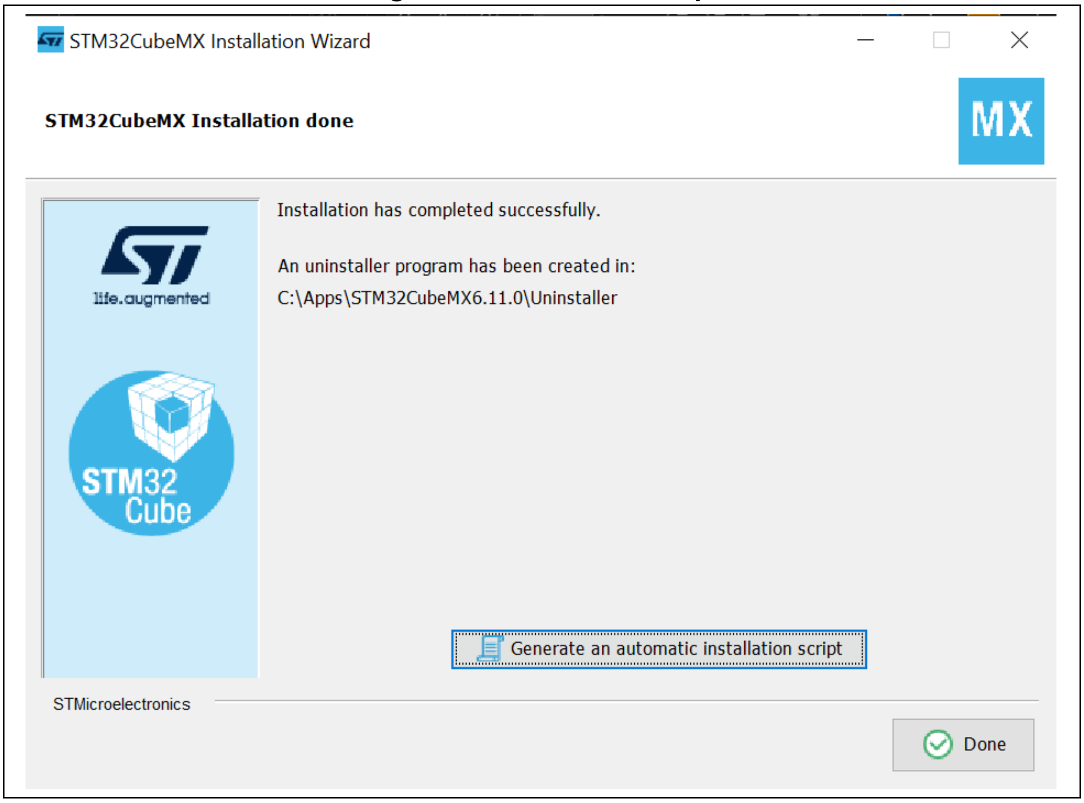


Figure 11. Installation path

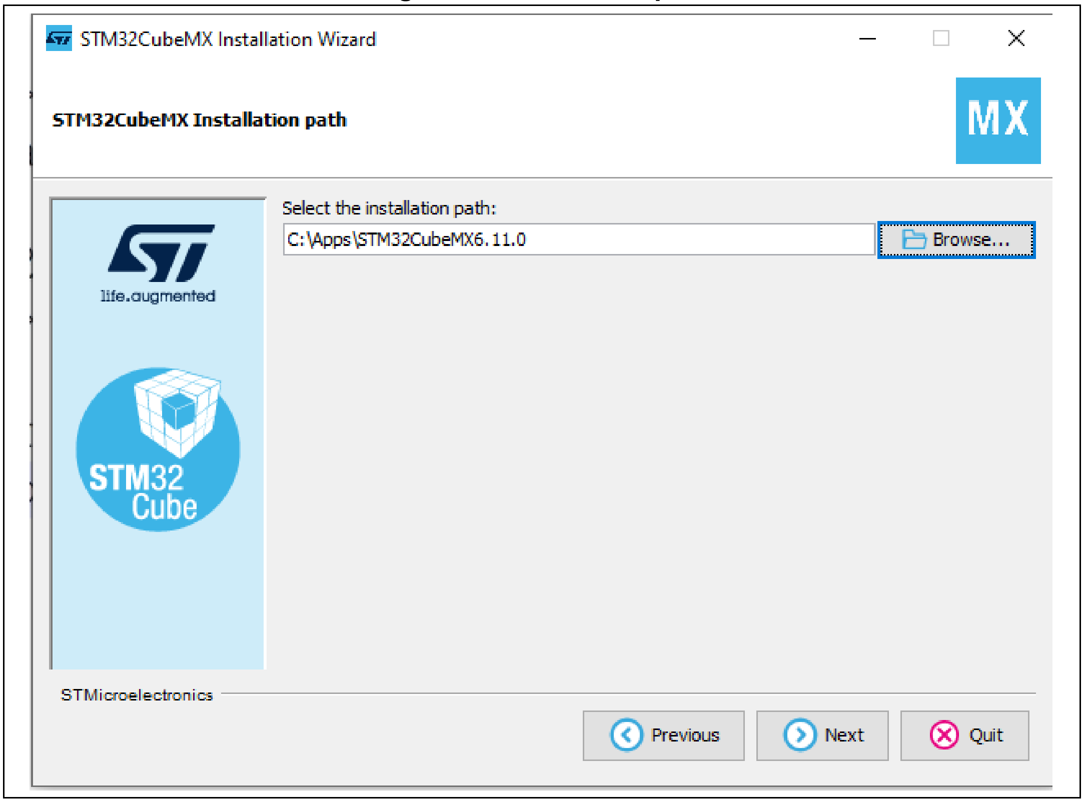


Figure 12. Current user shortcut creation

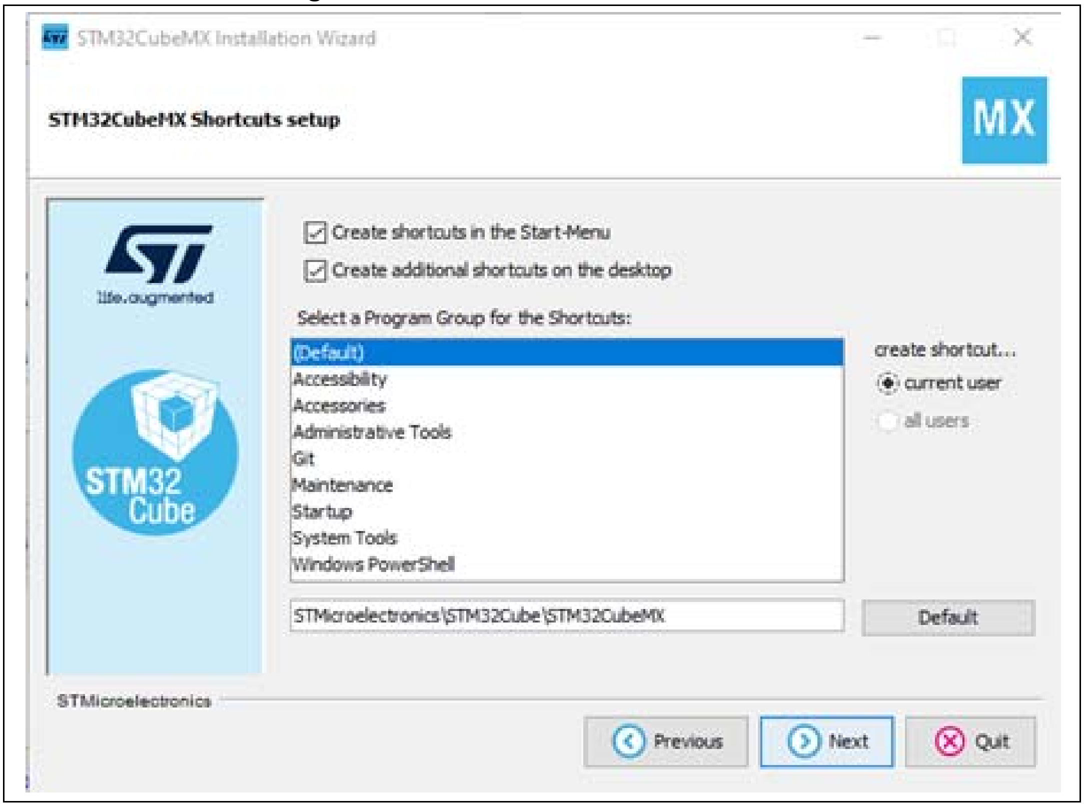


Figure 13. Package installation

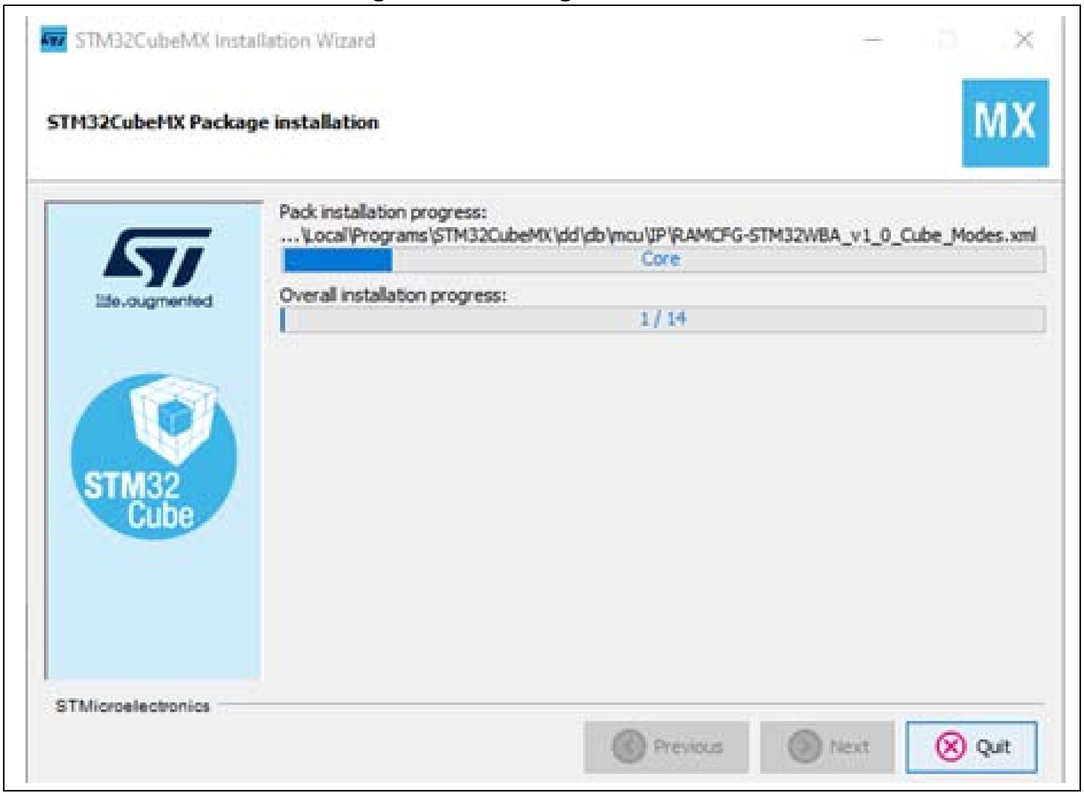
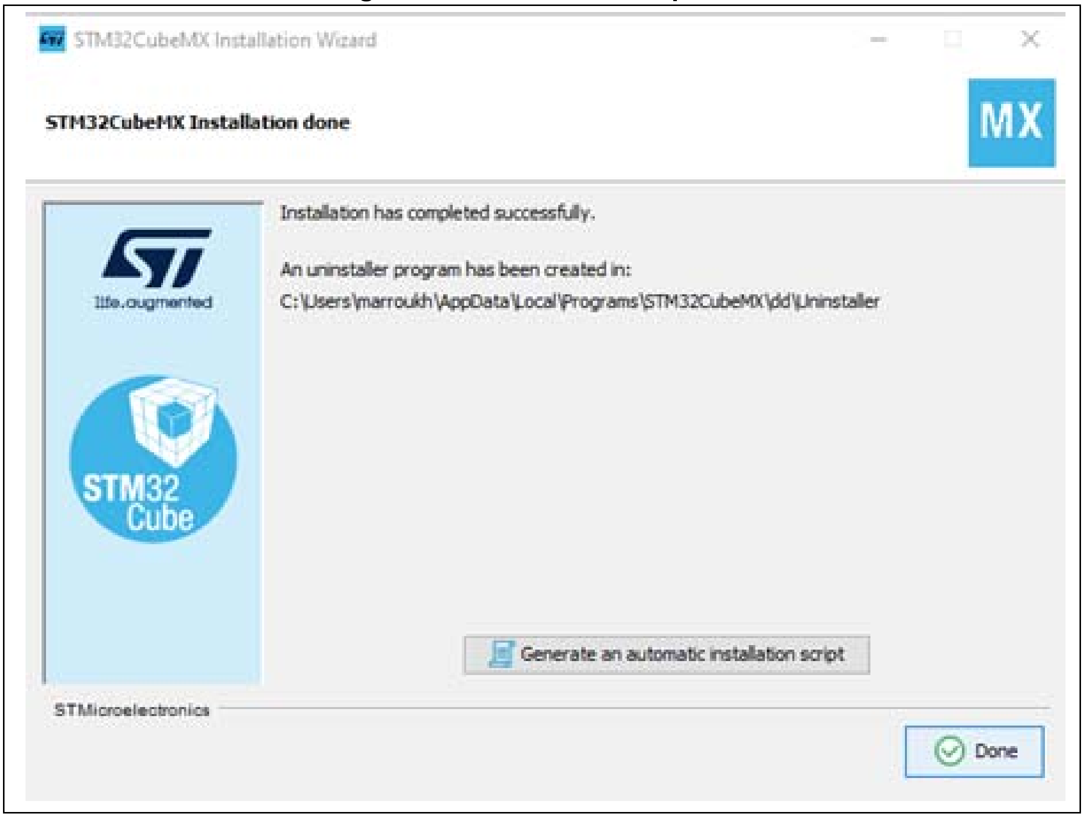


Figure 14. Installation completed



Note: Upon successful installation, the STM32CubeMX icon is displayed on the desktop and the application is available from the Program menu. STM32CubeMX .ioc files are displayed with a cube icon, double-clicking on it opens the project in STM32CubeMX. Only the latest installation of STM32CubeMX is enabled in the Program menu. Previous versions can be kept on your PC (not recommended) when different installation folders have been specified. Otherwise, the new installation overwrites the previous one(s).

On Linux:

a) On **STM32CubeMX-Lin** line, Click “Get software” to download the package
b) Extract (unzip) the downloaded package
c) Make sure you have administrator rights to access the target installation directory. You can run the installation as root (or sudo) to install STM32CubeMX in shared directories.
d) Do **chmod 777 SetupSTM32CubeMX-VERSION** to change the properties, so that the file is executable
e) Double-click on the **SetupSTM32CubeMX-VERSION** file, or launch it from the console window

On macOS:

a) On **STM32CubeMX-Mac** line, Click “Get software” to download the package
b) Extract (unzip) the downloaded package
c) Make sure you have administrator rights
d) Double-click **SetupSTM32CubeMX-VERSION.app** application file to launch the installation wizard

In case of error, try to fix it: - $sudo xattr -cr <Folder where the zip was extracted>

To ensure the best performance, choose the appropriate STM32CubeMX installer for your Mac. Currently two installers are available, one for Arm64 architecture, one for Intel-based Macs.

3.2.2 Installing STM32CubeMX from command line

There are two ways to launch an installation from a console window: either in console interactive mode or via a script.

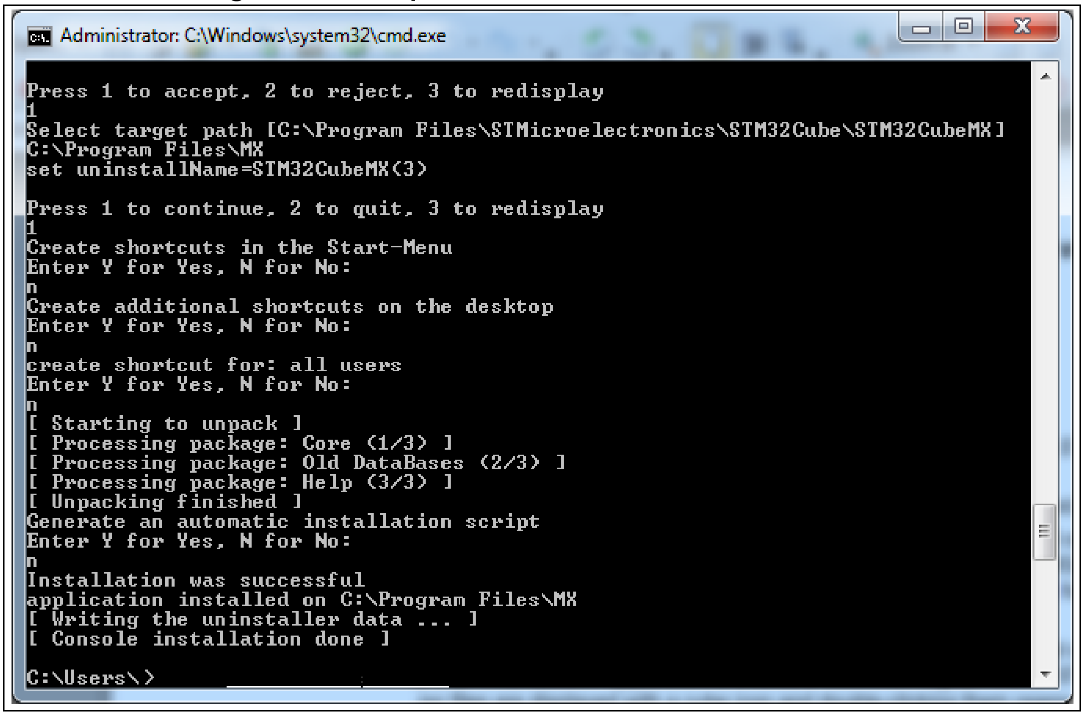
Interactive mode

To perform interactive installation, proceed as follows:

1. Extract (unzip) to folder the auto-extract installation file (SetupSTM32CubeMX-VERSION-Win.exe)
2. Open a standard console window to install for the current user, or the console window with administrator rights to install for all users
3. Go to the extracted folder (cd <folder path>)
4. Run the command **jre\bin\java -jar SetupSTM32CubeMX-<VERSION>.exe -console**

At each installation step, an answer is requested (see *Figure 15*).

Figure 15. Example of installation in interactive mode

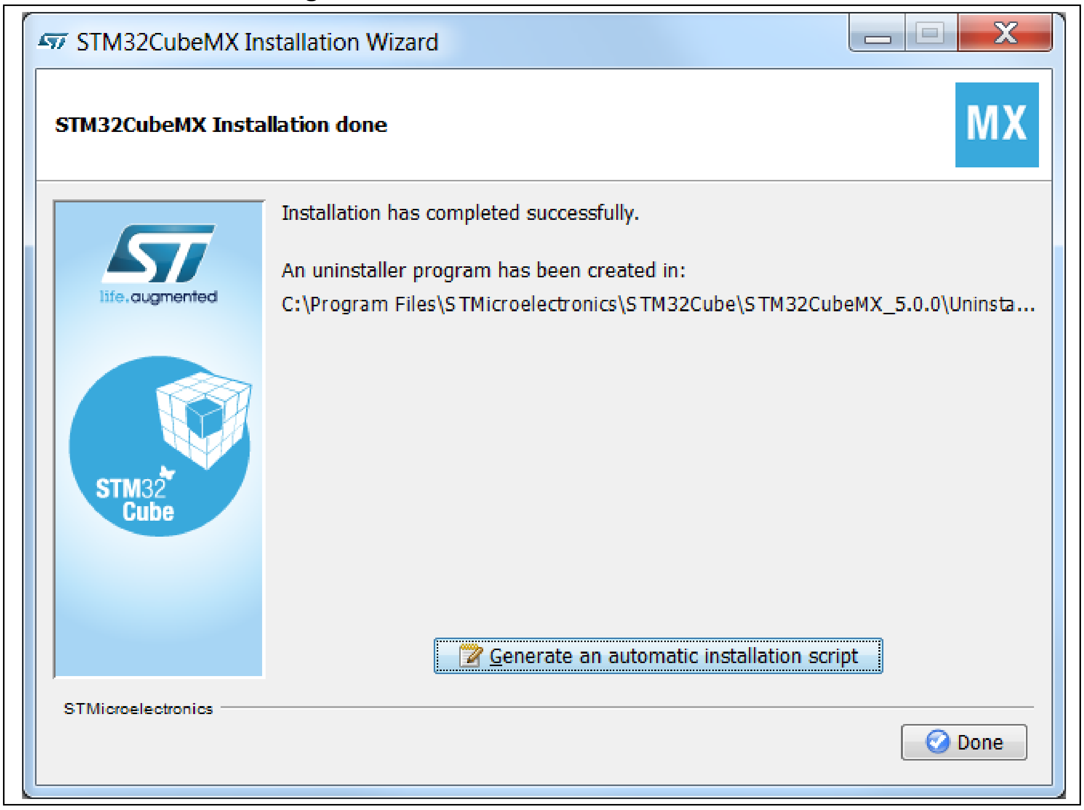


Note: During the installation, ignore the warnings.

Auto-install mode

At end of an installation, performed either using STM32CubeMX graphical wizard or console mode, it is possible to generate an auto-installation script containing user preferences (see *Figure 16*).

Figure 16. STM32Cube installation wizard



You can then launch the installation by typing, from a console window (with or without administrator rights, according to your needs), the command:

`SetupSTM32CubeMX-VERSION-Win.exe ABSOLUTE_PATH_TO_AUTO_INSTALL.xml`

3.2.3 Uninstalling STM32CubeMX standalone version

Uninstalling STM32CubeMX on macOS®

- Move STM32CubeMX.VERSION.app to the trash
- Use the following command line:
  - For STM32CubeMX 6.2.x and later versions:
    cd SetupSTM32CubeMX-VERSION.app/Contents/Resources/Uninstaller
    ./uninstall.sh
  - For STM32CubeMX 6.1.x and older versions:
    java -jar SetupSTM32CubeMX-VERSION.app/Contents/Resources/Uninstaller/uninstaller.jar.

Uninstalling STM32CubeMX on Linux

- From a shell prompt by launching the uninstall script
  - For STM32CubeMX 6.2.x and later versions:
```
cd <STM32CubeMX installation path>/Uninstaller
uninstall.sh
```
  - For STM32CubeMX 6.1.x and older versions:
    java -jar <STM32CubeMX installation path>/Uninstaller/uninstaller.jar.
- From a file explorer
  - Go to <STM32CubeMX installation path>/Uninstaller
  - For STM32CubeMX 6.2.x and later versions: double-click the uninstall.sh script
  - For STM32CubeMX 6.1.x and older versions: double-click the start uninstall desktop shortcut

Uninstalling STM32CubeMX on Windows

- Through the Windows Control Panel:
  a) Select **Programs and Features** from the **Windows Control** Panel to display the list of programs installed on your computer.
  b) Right-click **STM32CubeMX** and select **uninstall**.
- From a shell prompt, by using the following commands:
  - For STM32CubeMX 6.10.x and later versions:
    with administrator rights:
```
cd <STM32CubeMX installation path>/Uninstaller
admin_uninstall.bat
```
    without administrator rights:
```
cd <STM32CubeMX installation path>/Uninstaller
uninstall.batcd <STM32CubeMX installation path>/Uninstaller
uninstall.bat
```
  - From STM32CubeMX 6.2.x to STM32CubeMX 6.9.x versions:
```
cd <STM32CubeMX installation path>/Uninstaller
admin_uninstall.bat
```
  - For STM32CubeMX 6.1.x and older versions:
```
java -jar <STM32CubeMX installation path>/Uninstaller/uninstaller.jar
```
- Through a Windows File Explorer window:
  a) For STM32CubeMX 6.2.x and later versions:
     Go to the Uninstaller folder in STM32CubeMX installation directory, then:
     > with administrator rights, right-click on admin_uninstall.bat and “run as administrator”
     > without administrator rights, click on uninstall.bat
  b) For STM32CubeMX 6.1.x and older versions:
     Go to the Uninstaller folder in STM32CubeMX installation directory
     Double-click startuninstall.exe, or double-click the uninstall shortcut on the desktop

3.3 Launching STM32CubeMX

When running STM32CubeMX behind a proxy, see *Section 3.4.1*.

3.3.1 Running STM32CubeMX as a standalone application

To run STM32CubeMX as a standalone application on Windows, select STM32CubeMX from Program Files > ST Microelectronics > STM32CubeMX,or double-click STM32CubeMX icon on your desktop.

To run STM32CubeMX as a standalone application on Linux, launch the STM32CubeMX executable from STM32CubeMX installation directory.

To run STM32CubeMX as a standalone application on macOS, launch the STM32CubeMX application from the launchpad.

Note: There is no STM32CubeMX desktop icon on macOS.

3.3.2 Running STM32CubeMX in command-line mode

To facilitate its integration with other tools, STM32CubeMX provides command-line modes. Thanks to the commands listed in *Table 1* it is possible to:

- load an MCU
- load an existing configuration
- save a current configuration
- set project parameters and generate corresponding code
- generate user code from templates
- load a board identified through its part number
- refresh the list of embedded software packages (packs and STM32Cube MCU packages) and install/remove a package
- select additional software (packs) components to add to the project.

Three command-line modes are available:

- To run STM32CubeMX in interactive command-line mode, use the following command lines:
  - On Windows:

```
cd <STM32CubeMX installation path>
jre\bin\java -jar --add-opens java.desktop/java.awt=ALL-UNNAMED --
addexports java.desktop/sun.awt=ALL-UNNAMED -
Djavax.net.ssl.trustStoreType=WINDOWS-ROOT STM32CubeMX.exe -i
```

  - On Linux:

```
cd <STM32CubeMX installation path>
./STM32CubeMX -i
```

  - On macOS:

```
cd <STM32CubeMX installation path> cd Contents/MacOs
./STM32CubeMX -i
```

The “MX>” prompt is displayed, indicating that the application is ready to accept commands.

- To run STM32CubeMX in command-line mode, getting commands from a script, use the following command lines:
  – On Windows:
```
cd <STM32CubeMX installation path>
jre\bin\java -jar STM32CubeMX.exe -s <script filename>
```
  – On Linux and macOS:
```
./STM32CubeMX -s <script filename>
```
  All the commands to be executed must be listed in the script file. An example of script file content is shown below:
```
load STM32F417VETx
project name MyFirstMXGeneratedProject
project toolchain "MDK-ARM v4"
project path C:\STM32CubeProjects\STM32F417VETx
project generate
exit
```
- To run STM32CubeMX in command-line mode getting commands from a script and without UI, use the following command lines:
  – On Windows:
```
cd <STM32CubeMX installation path>
jre\bin\java -jar STM32CubeMX.exe -q <script filename>
```
  – On Linux and macOS:
```
./STM32CubeMX -q <script filename>
```
  Here again, the user can enter commands when the MX prompt is displayed.

Table 1. Command line summary

Command line	Purpose	Example
help	Displays the list of available commands.	help
swmgr refresh	Refreshes the list of embedded software package versions available for download.	swmgr refresh
swmgr install stm32cube_<series> _<version> ask	Installs the specified STM32Cube MCU package version.(1)	swmgr install stm32cube_f1_1.8.0 ask
swmgr remove stm32cube_<series> _<version>	Removes the specified STM32Cube MCU package version.	swmgr remove stm32cube_f1_1.8.0
swmgr install <packVendor>.<packName>. <packVersion> ask	Installs the specified pack version.	swmgr install STMicroelectronics. X-CUBE-NFC4.1.4.1 ask
swmgr remove <packVendor>.<packName>. <packVersion>	Removes the specified pack version.	swmgr remove STMicroelectronics. X-CUBE-BLE1.4.2.0

Table 1. Command line summary (continued)

Command line	Purpose	Example
pack enable <vendor> <pack>[/bundle] <version> <class> <group>[/subgroup] [variant]	Selects a software pack component to add in the project. The presence of “/” in the second and/or the fifth parameter(s) indicates, respectively, the explicit mention of a bundle and/or a subgroup (reference: Arm CMSIS pack pdsc format). To find out the pack / bundle / class / group / subgroup names of the component to enable, select the component and click “Hide/Show details” from the Additional Software window.	pack enable STMicroelectronics “X-CUBE-BLE1/BlueNRG-MS” 1.0.0 "Wireless" "Controller"
pack validate	Applies in the project all pack components enabled since the “pack validate” command was last called.	pack validate
load <mcu>	Loads the selected MCU.	load STM32F101RCTx load STM32F101Z(F-G)Tx
load <board part number> <allmodes|nomode>	Loads the selected board with all peripherals configured in their default mode (allmodes) or without any configuration (nomode).	loadboard NUCLEO-F030R8 allmodes loadboard NUCLEO-F030R8 nomode
config load <filename>	Loads a previously saved configuration.	config load “C:\Cube\ccmram\ccmram.ioc”
config save <filename>	Saves the current configuration.	config save “C:\Cube\ccmram\ccmram.ioc”
config saveext <filename>	Saves the current configuration with all parameters, including those for which values have been kept to default.	config saveext “C:\Cube\ccmram\ccmram.ioc”
config saveas <filename>	Saves the current project under a new name.	config saveas “C:\Cube\ccmram2\ccmram2.ioc”
csv pinout <filename>	Exports the current pin configuration as a csv file. This file can be (later) imported into a board layout tool.	Csv pinout mypinout.csv
script <filename>	Runs all commands in the script file. There must be one command per line.	script myscript.txt
project couplefilesbyip <0|1>	This option allows the user to choose between 0 (to generate the peripheral initializations in the main) and 1 (to generate each peripheral initialization in dedicated .c/.h files).	project couplefilesbyip 1
setDriver <Peripheral Name> <HAL | LL>	For the supported series, STM32CubeMX can generate peripheral initialization code based on LL or on HAL drivers. This command line allows the user to choose, for each peripheral, between HAL- and LL-based code generation. By default code generation is based on HAL drivers.	setDriver ADC LL setDriver I2C HAL

Table 1. Command line summary (continued)

Command line	Purpose	Example
generate code <path>	Generates only "STM32CubeMX generated" code and not a complete project (including STM32Cube firmware libraries and toolchains project files). To generate a project, use "project generate".	generate code C:\mypath
set tpl_path <path>	Sets the path to the source folder containing the .ftl user template files. All the template files stored in this folder are used for code generation.	set tpl_path C:\myTemplates\
set dest_path <path>	Sets the path to the destination folder that will hold the code generated according to user templates.	set dest_path C:\myMXProject\inc\
get tpl_path	Retrieves the path name of the user template source folder.	get tpl_path
get dest_path	Retrieves the path name of the user template destination folder.	get dest_path
SetStructure <Advanced/Basic>	Selects the project structure to generate.	SetStructure Basic
SetCopyLibrary <copy all / copy only / copy as reference>	Selects how the reference libraries are copied to the projects.	SetCopyLibrary "copy all"
project setCustomFWPath <CustomFwLocation>	Specifies a path to STM32Cube MCU software libraries different from STM32Cube repository path (specified under Help > Updater settings).	project SetCustomFwPath "F:/SharedRepository/STM32Cube_FW_F0_V1.11.0"
project toolchain <toolchain>	Specifies the toolchain to be used for the project. Use the "project generate" command to generate the project for that toolchain.	EWARM MDK-Arm STM32CubeIDE Makefile CMake
project name <name>	Specifies the project name.	project name ccmram
project path <path>	Specifies the path where to generate the project.	project path C:\Cube\ccmram
project compiler <compiler/linker>	Set Compiler/Linker. Possible values: – GCC – Starm-Clang	project compiler GCC project compiler Starm-Clang
project generate	Generates the full project.(1)	project generate
login < email_adress> <password> <remember_me>	Allows you to login to download software packages.	login john.smith@st.com mypassword y
exit	Ends STM32CubeMX process.	exit

1. Use the login command before using this command.

3.4 Getting updates using STM32CubeMX

STM32CubeMX implements a mechanism to access the Internet and to:

- download embedded software packages: STM32Cube MCU packages (full releases and patches) and third-party packages (.pack) based on the Arm® CMIS pack format
- manage a user-defined list of third-party packs
- check for STM32CubeMX and embedded software packages updates
- perform self-updates of STM32CubeMX
- refresh STM32 MCUs descriptions and documentation offer.

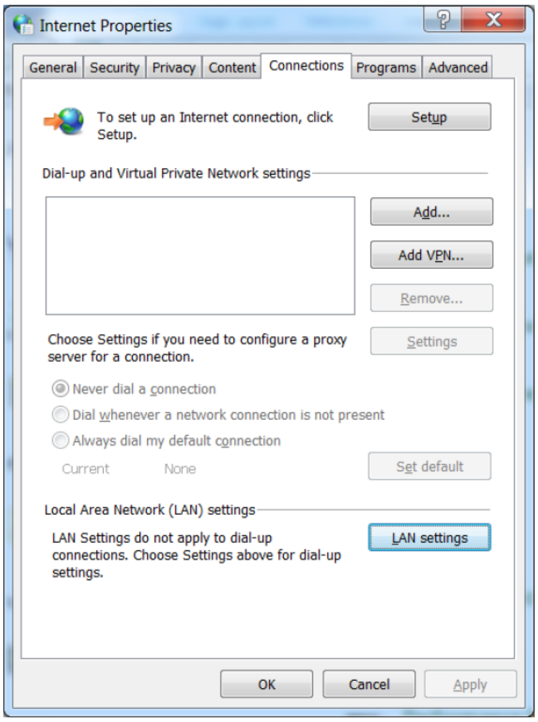
Installation and update related submenus are available under the **Help** menu and from the home page as well.

Off-line updates can also be performed on computers without Internet access (see *Section 3.4.3*). This is done by browsing the filesystem and selecting available STM32Cube MCU packages.

If the PC on which STM32CubeMX runs is connected to a computer network using a proxy server, STM32CubeMX needs to connect to that server to access the Internet, get self-updates and download firmware packages. Refer to *Section 3.4.2* for a description of this connection configuration.

To view Windows default proxy settings, select Internet options from the Control panel and select LAN settings from the **Connections** tab (see *Figure 17*).

Figure 17. Displaying Windows default proxy settings



Several proxy types exist, and different network configurations are possible:

- Without proxy: the application directly accesses the web (Windows default configuration).
- Proxy without login/password
- Proxy with login/password: when using an Internet browser, a dialog box opens and prompts the user to enter its login/password.
- Web proxies with login/password: when using an Internet browser, a web page opens and prompts the user to enter its login/password.

If needed, contact your IT administrator for proxy information (proxy type, http address, port).

STM32CubeMX does not support web proxies. In this case, the user cannot benefit from the update mechanism and must manually copy the STM32Cube MCU packages from http://www.st.com/stm32cube to the repository. To do it, follow the sequence below:

1. Go to http://www.st.com/stm32cube and download the relevant STM32Cube MCU package from the *Associated Software* section.
2. Unzip the zip package to your STM32Cube repository. Find out the default repository folder location in the **Updater Settings** tab as shown in *Figure 18* (you might need to update it to use a different location or name).

3.4.1 Running STM32CubeMX behind a proxy server

When proxies are implementing full SSL inspection, STM32CubeMX must be configured to use the proxy certificate.

- On Windows:

  Typically, it comes down to using Windows certificate list.

  a) there is no additional configuration necessary to run STM32CubeMX executable (it is already configured to use Windows certificate list)

  b) the command line must be adjusted to run STM32CubeMX from the command line:

  ```
  cd <STM32CubeMX install path>
  jre\bin\java -Djavax.net.ssl.trustStoreType=WINDOWS-ROOT -jar
  STM32CubeMX.exe
  ```

- On Mac/Linux and on Windows systems when the proxy certificate is not in Windows certificate store, the certificate must be manually imported. This is done using keytool from a command prompt, as follows:

  ```
  $ cd  <CUBEMX_INSTALL_DIR>/jre
  $ bin/keytool -importcert -alias <your certificate alias name> -
  keystore lib/security/cacerts -file  <path to you proxy certificate
  file>.crt
  ```

  When prompted, enter the password: *changeit*

  When prompted, accept to trust the certificate: *yes*

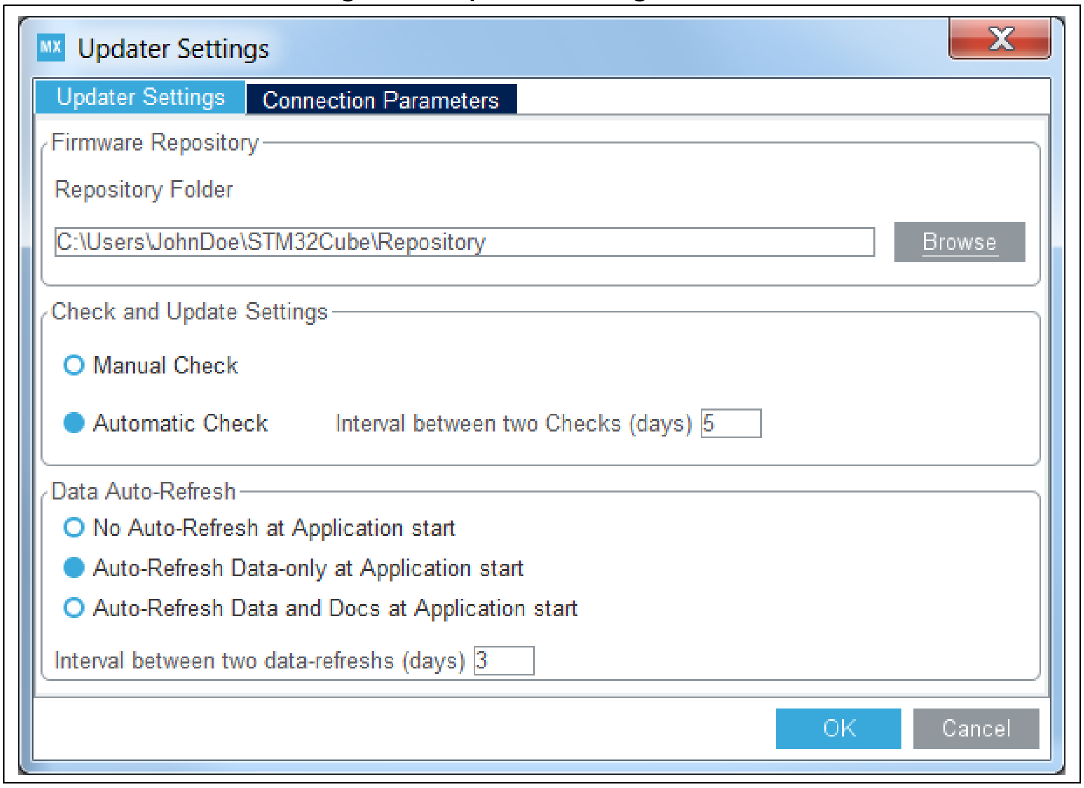
  Then (Windows only) edit file *<CUBEMX_INSTALL_DIR>/STM32CubeMX.l4j.ini* and remove the line: **-Djavax.net.ssl.trustStoreType=WINDOWS-ROOT**

3.4.2 Updater configuration

To perform STM32Cube new library package installation or updates, the tool must be configured as follows:

1. Select **Help > Updater Settings** to open the **Updater Settings** window.
2. From the **Updater Settings** tab (see *Figure 18*)
   a) Specify the repository destination folder where the downloaded packages will be stored.
   b) Enable/Disable the automatic check for updates.

Figure 18. Updater Settings window



3. In the **Connection Parameters** tab, specify the proxy server settings appropriate for your network configuration by selecting a proxy type among the following possibilities (see *Figure 19*):
   – No Proxy
   – Use System Proxy Parameters

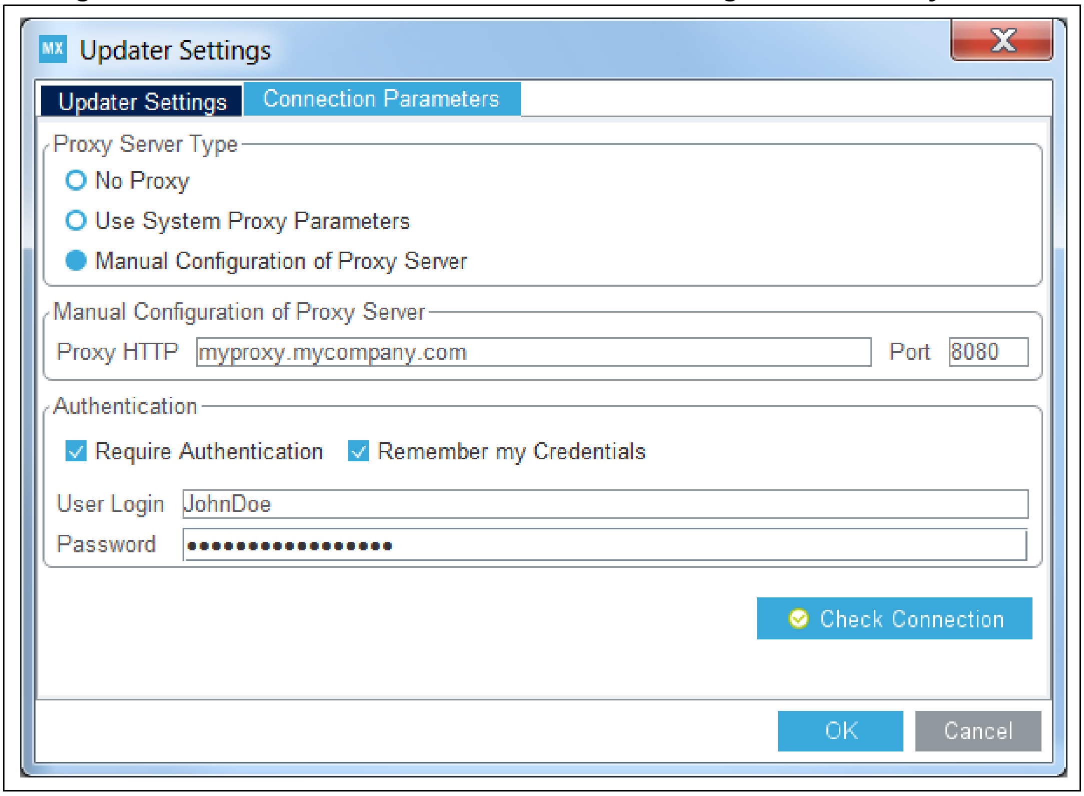
     On Windows, proxy parameters are retrieved from the PC system settings. Uncheck “Require Authentication” if a proxy server without login/password configuration is used.

– Manual Configuration of Proxy Server

Enter the Proxy server http address and port number. Enter login/password information or uncheck “Require Authentication” if a proxy server without login/password configuration is used.

4. Optionally uncheck **Remember my credentials** to prevent STM32CubeMX to save encrypted login/password information in a file. This implies reentering login/password information each time STM32CubeMX is launched.
5. Click the **Check Connection** button to verify if the connection works. A green check mark appears to confirm that the connection operates correctly 

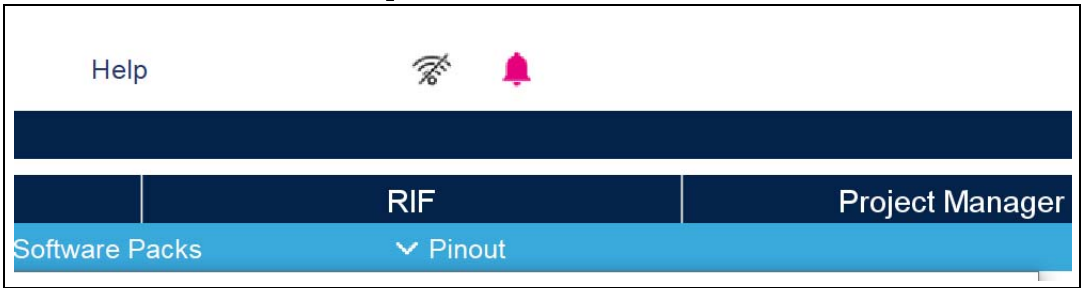
Figure 19. Connection Parameters tab - Manual Configuration of Proxy Server



6. Select **Help > Install New Libraries** submenu to select among a list of possible packages to install.
7. If the tool is configured for manual checks, select **Help > Check for Updates** to find out about new tool versions or firmware library patches available to install.

Note: If STM32Cube MX is not connected to the network, or if it detects a connection failure, an icon is displayed close to the myST menu item showing that there is no network connection. When the user clicks on that icon, “Configure network” menu is displayed, and by clicking on it, the “Updater Settings/Connection parameters” dialog pops up. Once the STM32CubeMX is connected to the network, the network icon disappears.

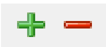
Figure 20. Connection failure



3.4.3 Installing STM32 MCU packages

To download new STM32 MCU packages, follow the steps below:

1. Select **Help** > **Manage embedded software packages** to open the **Embedded Software Packages Manager** (see *Figure 21*), or use Install/Remove button from the Home page.

   Expand/collapse buttons  expands/collapses the list of packages, respectively.

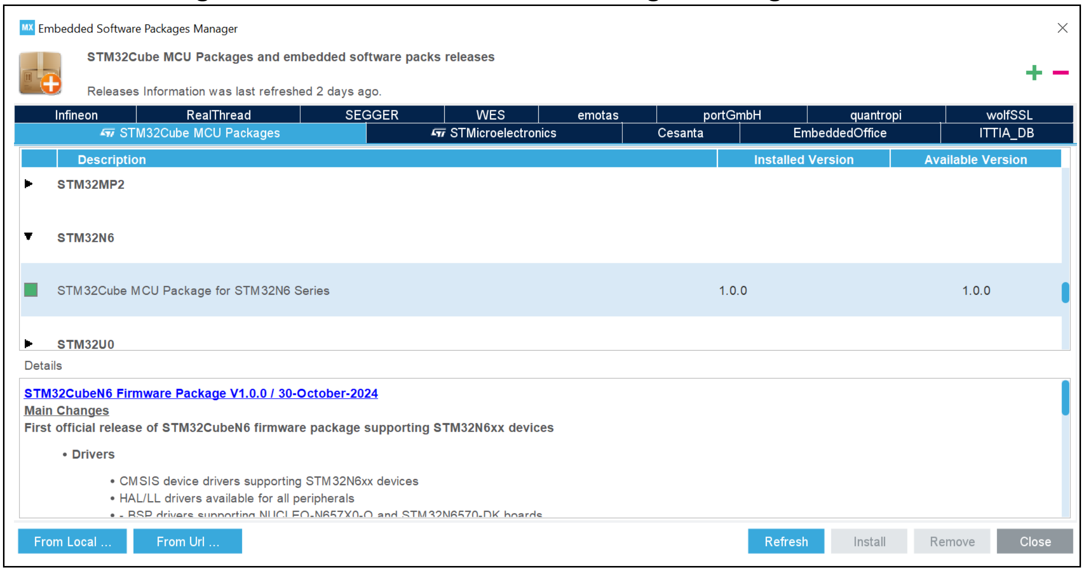
   If the installation was performed using STM32CubeMX, all the packages available for download are displayed along with their version including the version currently installed on the user PC (if any), and the latest version available from *www.st.com*.

   If no Internet access is available at that time, choose "From Local ...", then browse to select the zip file of the desired STM32Cube MCU package that has been previously downloaded. An integrity check is performed on the file to ensure that it is fully supported by STM32CubeMX.

   The package is marked in green when the version installed matches the latest version available from *www.st.com*.
2. Click the checkbox to select a package then "Install Now" to start the download.

See *Figure 21* for an example.

Figure 21. Embedded Software Packages Manager window



3.4.4 Installing STM32 MCU package patches

Use the procedure described in *Section 3.4.3* to download STM32 MCU package patches.

A library patch, such as STM32Cube_FW_F7_1.4.1, can be easily identified by the version number, whose third digit is non-null (e.g. ‘1’ for the 1.4.1 version).

The patch is not a complete library package but only the set of library files that need to be updated. The patched files go on top of the original package (e.g. STM32Cube_FW_F7_1.4.1 complements STM32Cube_FW_F7_1.4.0 package).

Prior to 4.17 version, STM32CubeMX copies the patches within the original baseline directory (e.g. STM32Cube_FW_F7_V1.4.1 patched files are copied within the directory called STM32Cube_FW_F7_V1.4.0).

Starting with STM32CubeMX 4.17, downloading a patch leads to the creation of a dedicated directory. As an example, downloading STM32Cube_FW_F7_V1.4.1 patch creates the STM32Cube_FW_F7_V1.4.1 directory that contains the original STM32Cube_FW_F7_V1.4.0 baseline plus the patched files contained in STM32Cube_FW_F7_V1.4.1 package.

Users can then choose to go on using the original package (without patches) for some projects and upgrade to a patched version for others projects.

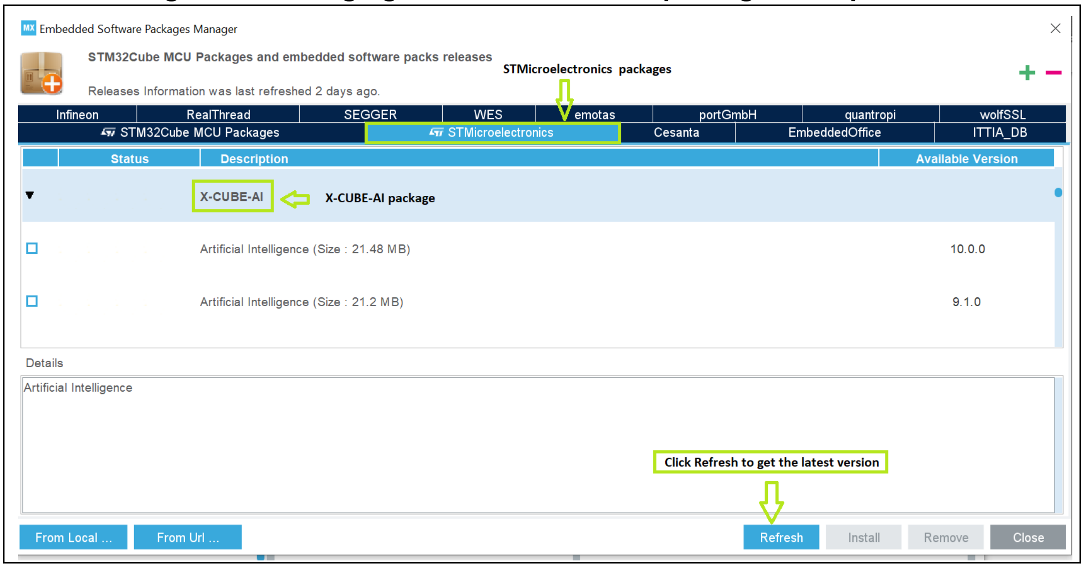
3.4.5 Installing embedded software packs

Starting from the release 4.24, STM32CubeMX offers the possibility to select third-party embedded software packages coming in the Arm® Keil™ CMSIS-Pack format (.pack), whose contents are described thanks to the pack description (.pdsc) file. Reference documentation is available from http://www.keil.com.

1. Select **Help > Manage embedded software packages** to open the **New Libraries Manager window** (see *Figure 22*), or use Install/Remove button from the Home page, or from the project **Pinout & Configuration** view (select **Software Packs > Manage Software Packs**).

   Use Expand/collapse buttons  to expand/collapse the list of packages, respectively.

Figure 22. Managing embedded software packages - Help menu



2. Click **From Local …** button to browse the computer filesystem and select an embedded software package. STM32Cube MCU packages come as .zip archives and embedded software packs come as .pack archives.

   This action is required in the following cases:

   – No Internet access is possible but the embedded software package is available locally on the computer.

   – The embedded software package is not public and hence not available on Internet. For such packages, STM32CubeMX cannot detect and propose updates.

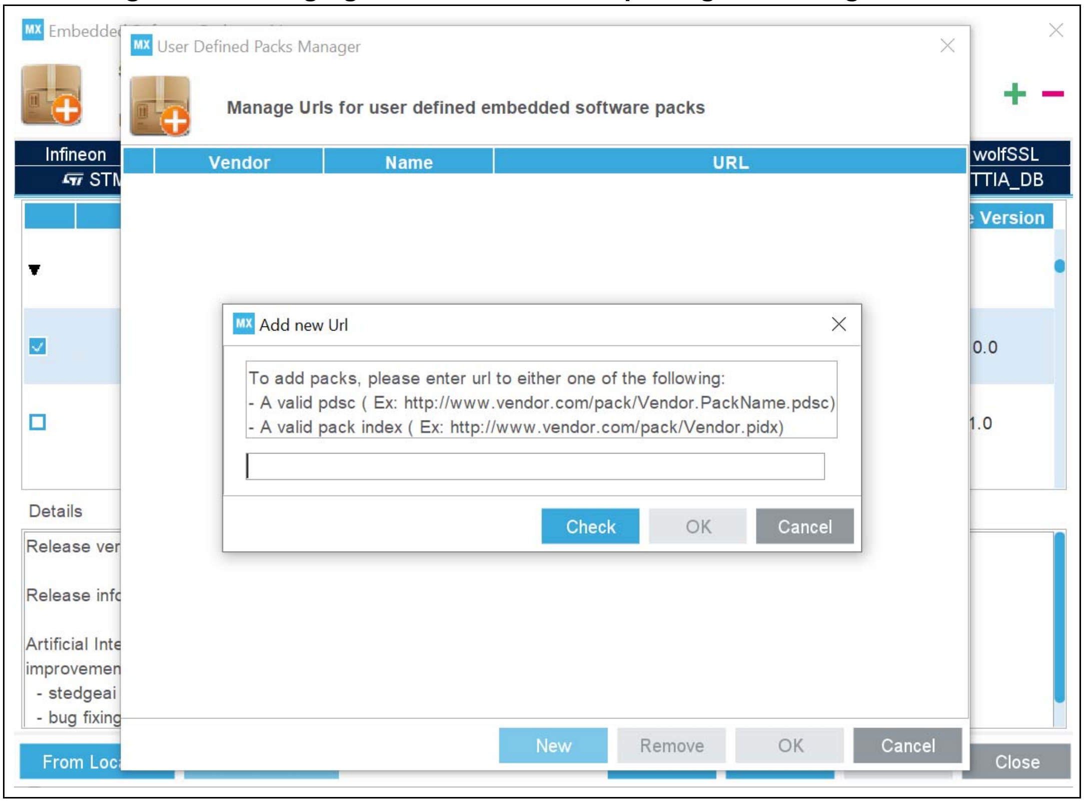
3. Click **From URL…** button to specify the download location from Internet for one of the pack .pdsc file or from the vendor pack index (.pidx).

   Proceed as follow:

   a) Choose **From URL …** and click **New** (see *Figure 23*).

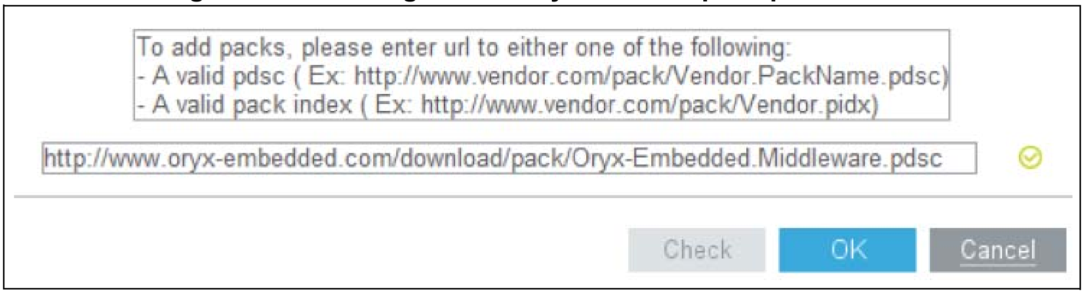
   b) Specify the .pdsc file url. As an example, the url of Oryx-Embedded middleware pack is https://www.oryx-embedded.com/download/pack/Oryx-Embedded.Middleware.pdsc (see *Figure 24*).

Figure 23. Managing embedded software packages - Adding a new url



c) Click the **Check** button to verify that the provided url is valid (see *Figure 24*).

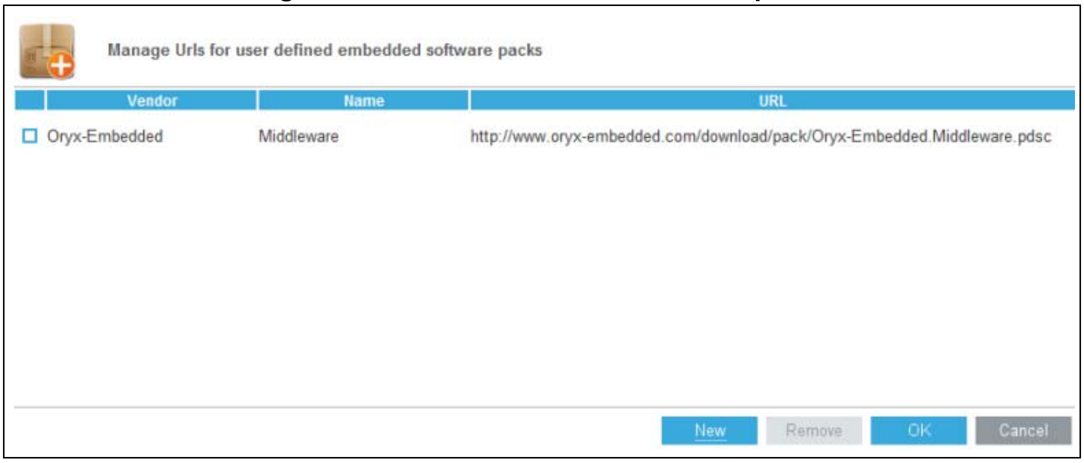
Figure 24. Checking the validity of vendor pack.pdsc file url



d) Click **OK**. The pack pdsc information is now available in the user defined pack list (see *Figure 25*).

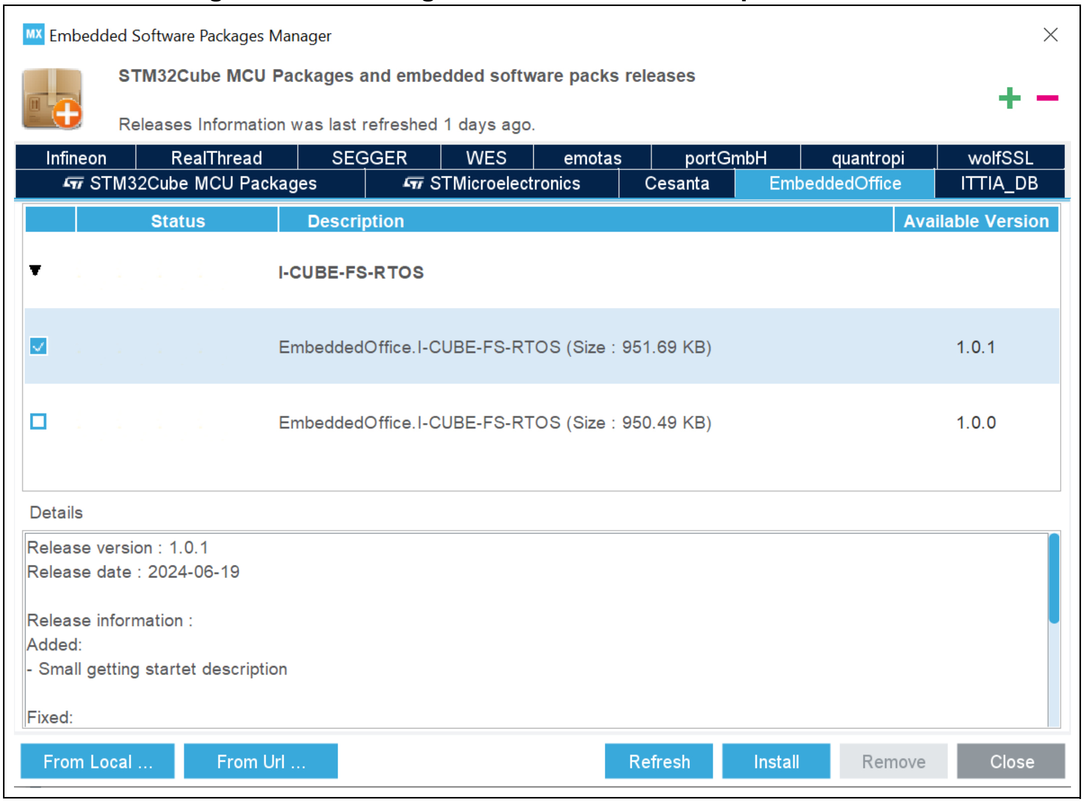
To delete a url from the list, select the url checkbox and click **Remove**.

Figure 25. User-defined list of software packs



e) Click **OK** to close the window and start retrieving psdc information. Upon successful completion, the available pack versions are shown in the list of libraries that can be installed. Use the corresponding checkbox to select a given release.

Figure 26. Selecting an embedded software pack release



f) Click **Install Now** to start downloading the software pack. A progress bar opens to indicate the installation progress. If the pack comes with a license agreement, a window pops up to ask for user's acceptance (see *Figure 27*). When the installation is successful, the check box turns green (see *Figure 28*).

The user can then add software components from this pack to its projects.

Figure 27. License agreement acceptance

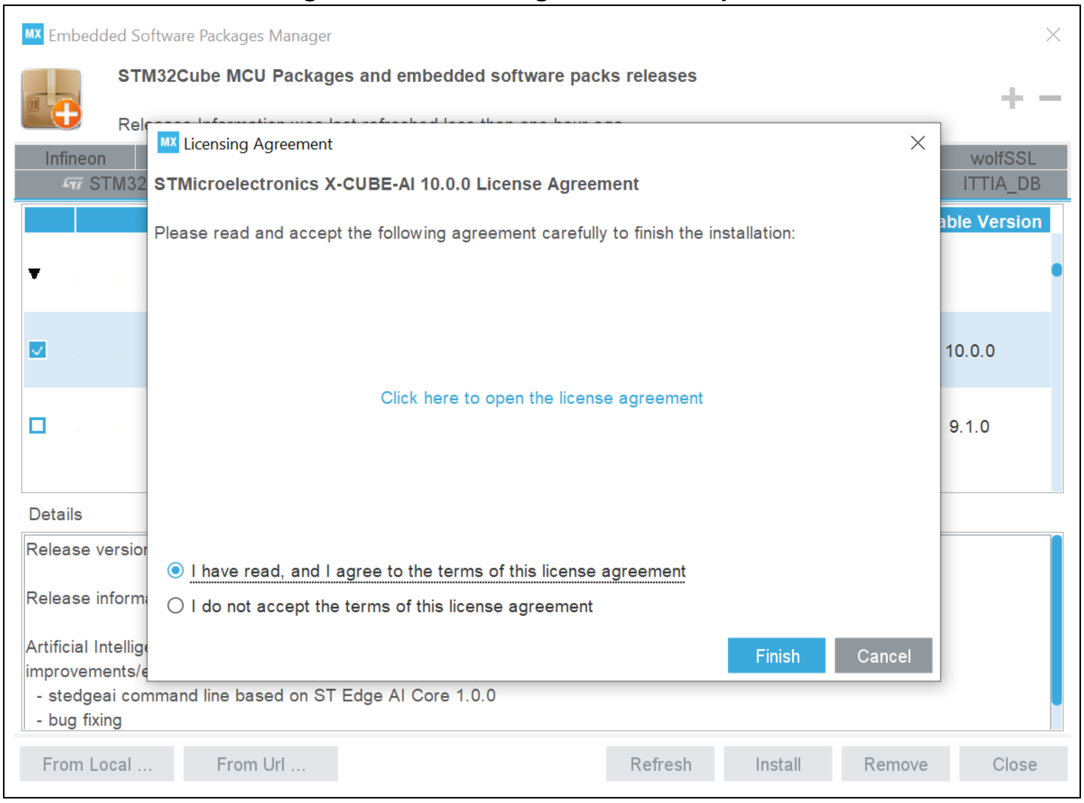
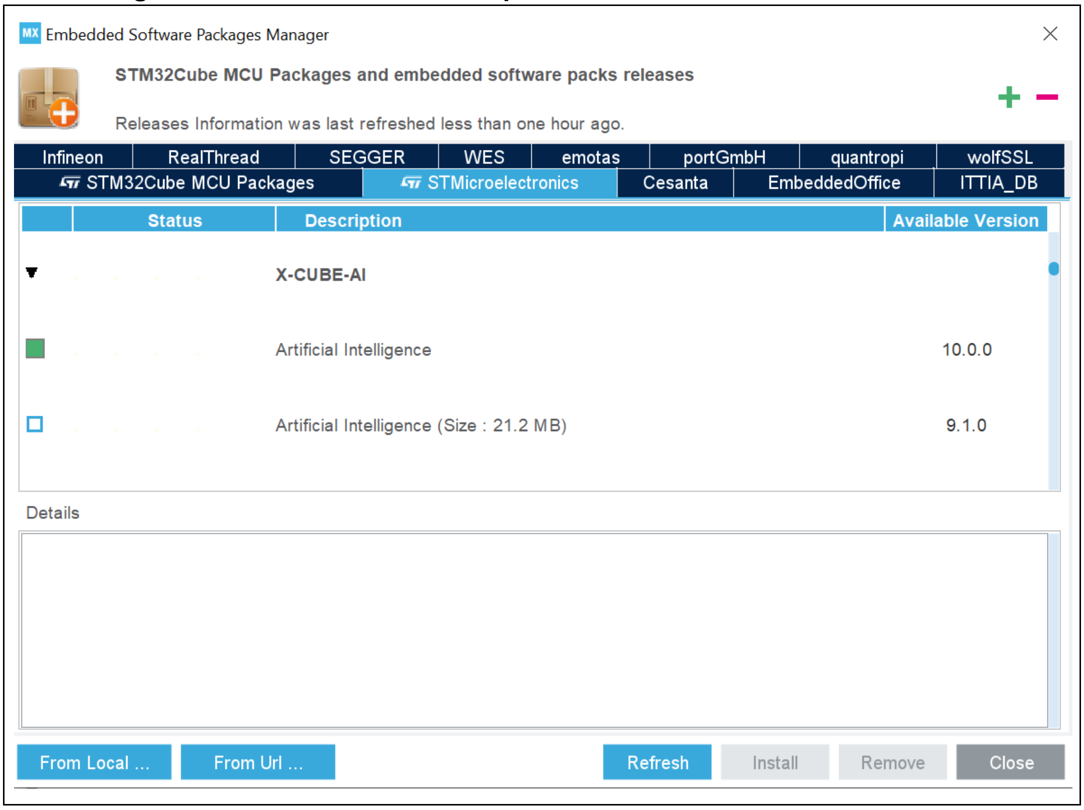


Figure 28. Embedded software pack release - Successful installation



3.4.6 Removing already installed embedded software packages

To clean up the repository from old library versions, thus saving disk space, proceed as follows (see figures *29* and *30*)

1. Select **Help > Manage embedded software packages** to open the **Embedded Software Packages Manager**, or use Install/Remove button from the Home page.
2. Click a green checkbox to select a package available in stm32cube repository.
3. Click the **Remove Now** button and confirm. A progress window opens to show the deletion status.

Figure 29. Removing a package

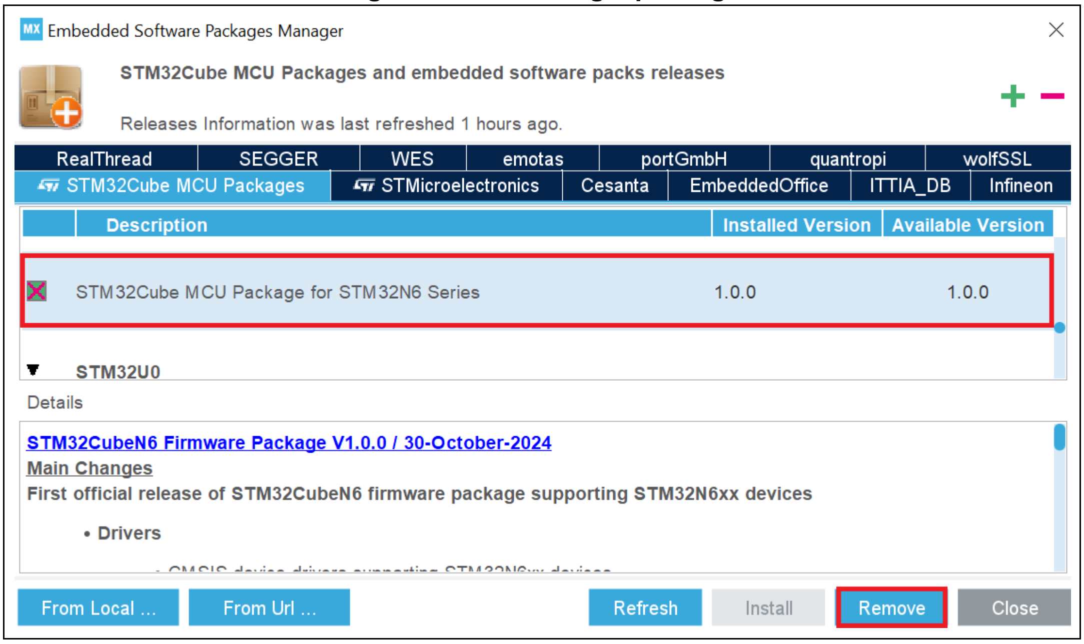
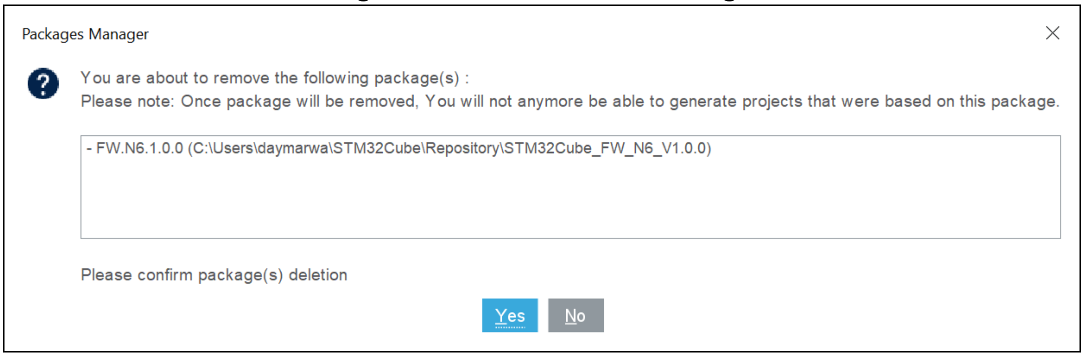


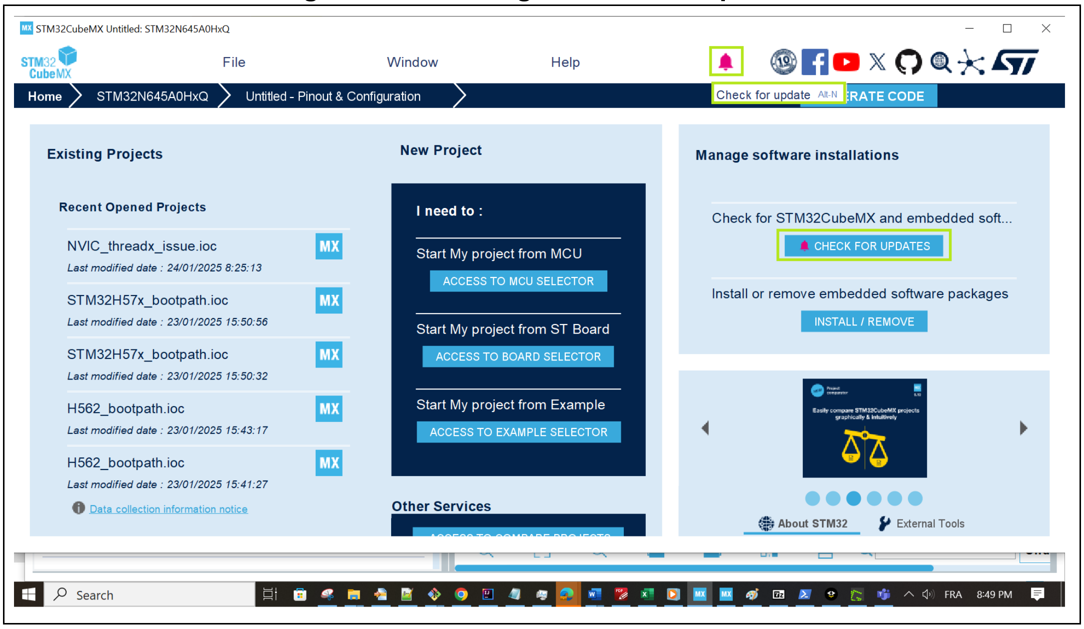
Figure 30. Confirmation message



3.4.7 Checking for updates

Starting with version V6.12.0, if there is a new CubeFW, X-Cube, or I-Cube available for update, an icon is displayed close to the myST menu. The same dedicated icon is displayed left to the “CHECK FOR UPDATES” button. When the user clicks on that icon, the Update Manager window opens.

Figure 31. Checking for available updates



When the updater is configured for automatic checks, it regularly verifies if updates are available.

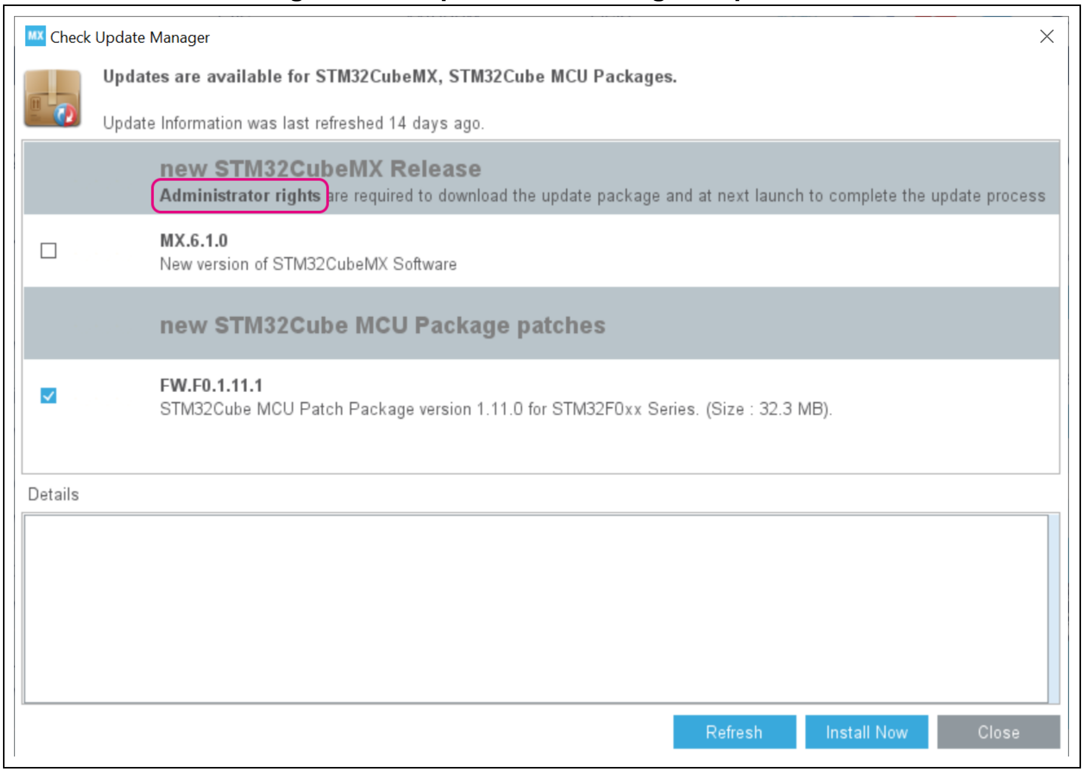
When automatic checks have been disabled in the updater settings window, the user can manually check if updates are available:

1. Click the icon to open the **Update Manager** window or Select **Help > Check for Updates**. All the updates available for the user current installation are listed.
2. Click the check box to select a package, and then Install Now to download the update.

Warning: When performing STM32CubeMX self-updates. administrator rights are required when downloading the self-update package and during the STM32CubeMX launch that completes the update process:
1. Launch STM32CubeMX with administrator account
2. Go to Help > Check for updates menu, select MX update package and click "Install now" to start the download
3. Re-launch STM32CubeMX with the administrator account to finish the update process

---

Figure 32. Help menu: checking for updates



4 STM32CubeMX user interface

STM32CubeMX user interface comes with three main views the user can navigate through using convenient breadcrumbs, namely the **Home** page, the **New project** window, and the project page. They come with panels, buttons and menus allowing users to take actions and make configuration choices with a single click.

The user interface is detailed in the following sections.

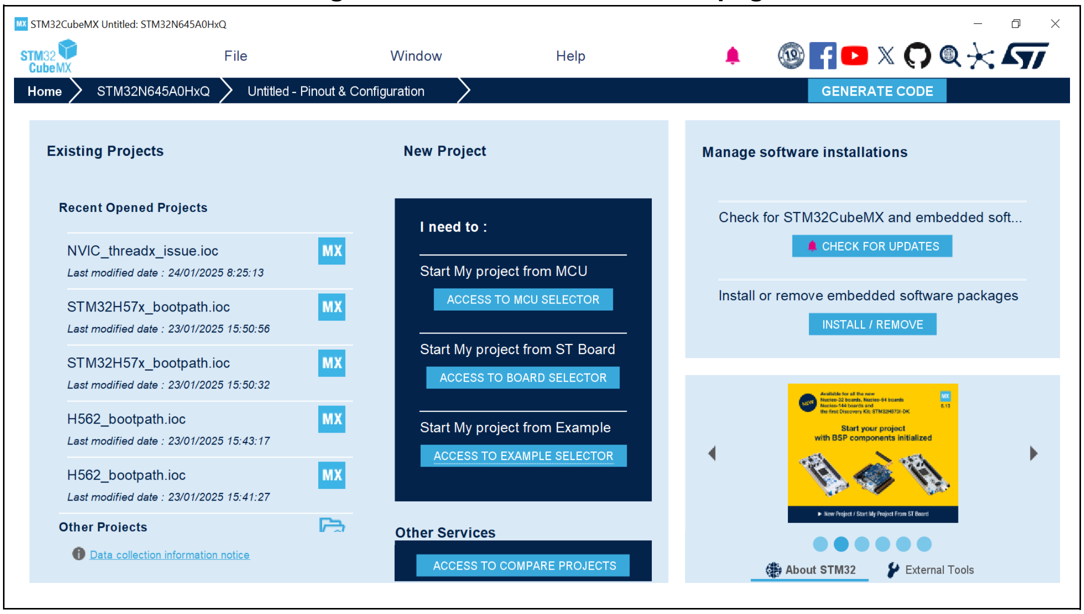
For C code generation, although the user can switch back and forth between the different configuration views, it is recommended to follow the sequence below:

1. From the **Project Manager** view, configure the project settings.
2. From the **Mode** panel in the **Pinout & Configuration** view, configure the RCC peripheral by enabling the external clocks, master output clocks, audio input clocks (when relevant for your application). This automatically displays more options on the **Clock configuration** view (see *Figure 183*). Then, select the features (peripherals, middlewares) and their operating modes relevant to the application.
3. If necessary, adjust the clock tree configuration from the clock configuration view.
4. From the Configuration panel in the **Pinout & Configuration** view configure the parameters required to initialize the peripherals and middleware operating modes.
5. Generate the initialization C code by clicking GENERATE CODE .

4.1 Home page

This is the first window that opens up when launching STM32CubeMX (see *Figure 33*). Closing it closes down the application. It offers shortcuts for some top level menus, an image carousel displaying STM32 latest news, as well as links to social network sites and external tools. Top-level menus and social network links remain accessible from the subsequent project page and are detailed in the following sections.

Figure 33. STM32CubeMX home page



4.1.1 File menu

Refer to *Table 2* for a description of the **File menu** and shortcuts.

Table 2. Home page shortcuts

Name *Keyboard shortcut*	**Description**	**Home page shortcut**
New Project... *Ctrl-N*	Opens a new project window showing all supported MCUs and a set of STMicroelectronics boards to choose from[1].	To create a new project starting from a board click  To create a new project starting from an MCU click 
Load Project... *Ctrl-L*	Loads an existing STM32CubeMX project configuration by selecting an STM32CubeMX configuration .ioc file (see *Caution:*).	Under Other project, click browse icon 
Import Project… *Ctrl-I*	Opens a new window to select the configuration file to be imported as well as the import settings. The import is possible only if you start from an empty MCU configuration. Otherwise, the menu is disabled[2].	None
Save Project *Ctrl-S*	Saves current project configuration (pinout, clock tree, peripherals, middlewares, Power Consumption Calculator) as a new project. This action creates a project folder including an .ioc file, according to user defined project settings.	None
Save Project as… *Ctrl-A*	Saves the current project.	None
Close Project *Ctrl-C*	Closes the current project and switches back to the welcome page.	None
Recent Projects *none*	Displays the list of the five most recently saved projects.	Under **Recent Project**, click  icon next to the project name.
Generate Report *Ctrl-R*	Saves the project current configuration as two documents (pdf and text formats).	None
Exit *Ctrl-X*	Proposes to save the project (if needed), then closes the application.	To close the window and the application click on . 

1. On **New project**: to avoid any popup error messages at this stage, make sure an Internet connection is available (Connection Parameters tab under Help > Updater settings menu) or that Data Auto-refresh settings are set to No Auto-Refresh at application start (Updater Settings tab under Help > Updater Settings menu).
2. On **Import**, a status window displays the warnings or errors detected when checking for import conflicts. The user can then decide to cancel the import.

Caution: On **project load**: STM32CubeMX detects if the project was created with an older version of the tool and if this is the case, it proposes the user to either migrate to use the latest STM32CubeMX database and STM32Cube firmware version, or to continue.
Prior to STM32CubeMX 4.17, clicking Continue still upgrades to the latest database "compatible" with the STM32Cube firmware version used by the project.
Starting from STM32CubeMX 4.17, clicking Continue keeps the database used to create the project untouched. If the required database version is not available on the computer, it is automatically downloaded.
When upgrading to a new version of STM32CubeMX, make sure to always backup your projects before loading the new project (especially when the project includes user code).

4.1.2 Window menu and Outputs tabs

The **Window menu** allows the user to access the **Outputs** function.

Table 3. Window menu

Name	Description
Outputs	Selecting/deselecting **Outputs** from the **Window** menu hides/shows the following Outputs tabs at the bottom of STM32CubeMX project page (see *Figure 34*) – **MCUs selection**: lists the MCUs of a given family matching the user criteria (series, peripherals, package,...) when an MCU was selected last(1). – **Outputs**: displays a non-exhaustive list of the actions performed, raised errors and warnings (see *Figure 35*) found upon user actions. **– IP assignment rules** **– MMT Output Log**
Font size	Makes possible to change font size settings. STM32CubeMX must be re-launched for changes to take effect.

1. Selecting a different MCU from the list resets the current project configuration and switches to the new MCU. The user is then prompted to confirm this action before proceeding.

Figure 34. Window menu

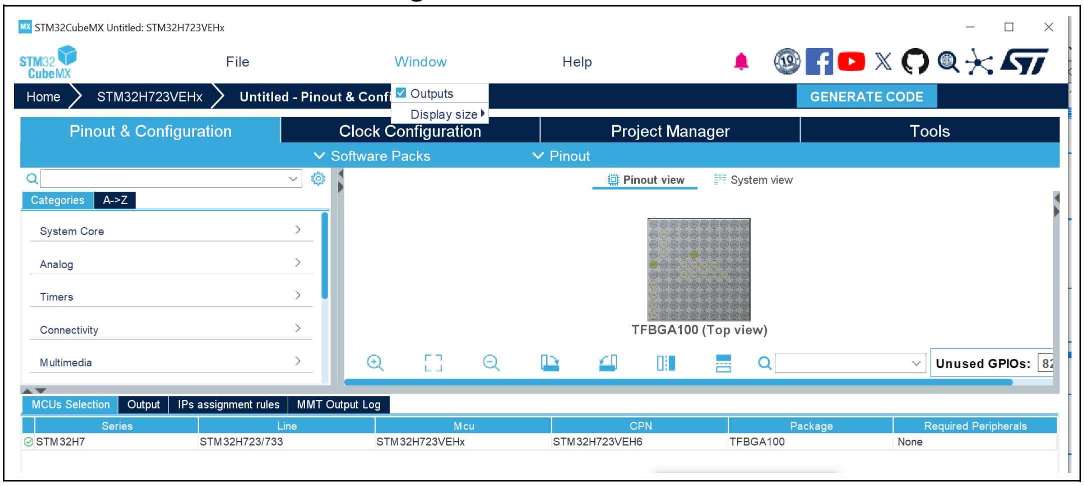
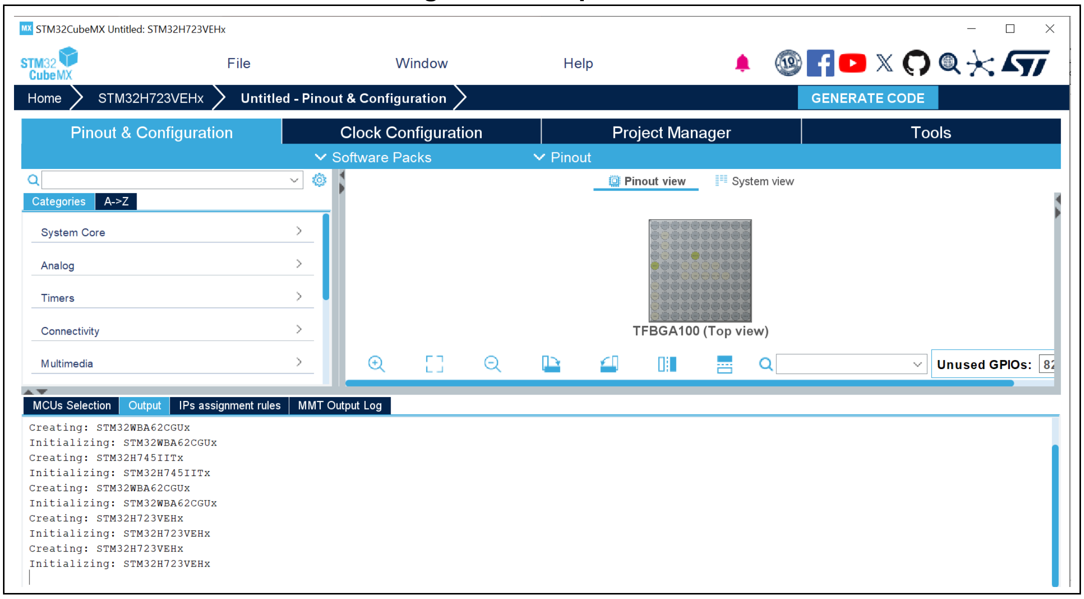


Figure 35. Output view



4.1.3 Help menu

Refer to *Table 4* for a description of the **Help** menu and shortcuts.

Table 4. Help menu shortcuts

Name *Keyboard shortcut*	**Description**	**Home page shortcut**
Help *F1*	Opens the STM32CubeMX user manual.	None
About *Alt-A*	Shows version information.	None
Docs & Resources *Alt-D*	Displays the official documentation available for the MCU used in the current project.	None
Video Tutorials *Alt-V*	Opens the Video Tutorial browser that proposes a list of videos and allows the user to launch a video in one click.	None
Refresh Data *Alt-R*	Opens a dialog window that proposes to refresh STM32CubeMX database with STM32 MCU latest information (description and list of official documents), and allows the user to download of all official documentation in one shot.	None
Check for Updates *Alt-C*	Shows the software and firmware release updates available for download.	Click CHECK FOR UPDATES
Manage embedded software packages *Alt-U*	Shows all the embedded software packages available for installation. A green check box indicates that the package is already installed in the user repository folder (the repository folder location is specified under Help > Updater Settings menu).	Click INSTALL/REMOVE

Table 4. Help menu shortcuts (continued)

Name *Keyboard shortcut*	Description	Home page shortcut
Connection & Updates *Alt-S*	Opens the Connection & Updates window to configure manual or automatic updates, proxy settings for Internet connections, and repository folder where the downloaded software and firmware releases are stored.	None
User Preferences	Opens the user preference window to enable or disable collect of features usage statistics.	None

4.1.4 Social links

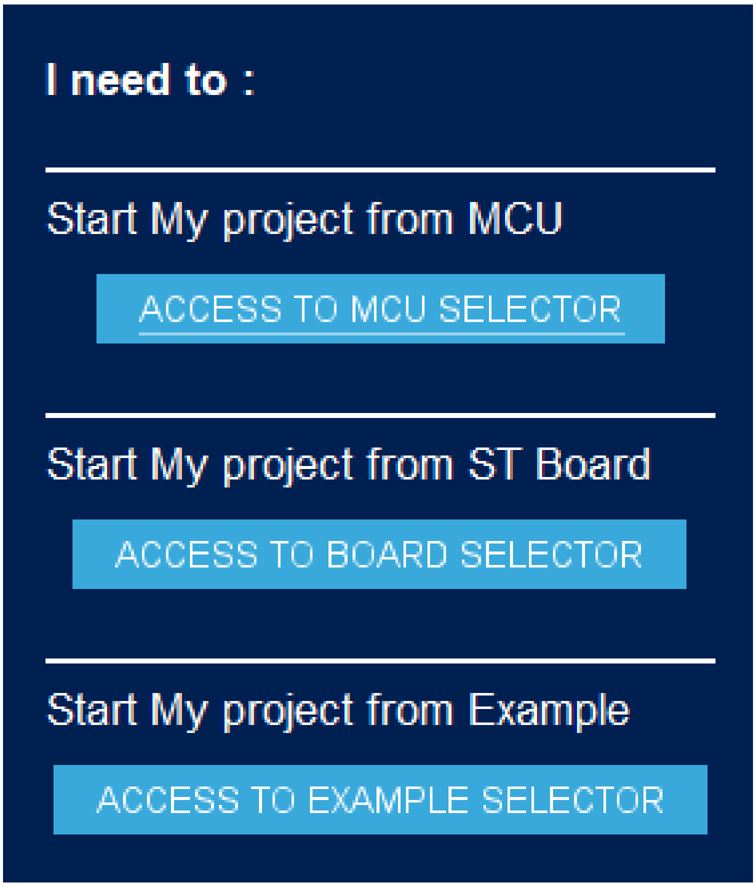
Developer communities on popular social platforms such as Facebook™, STM32 YouTube™ channel, as well as ST Community can be accessed from the STM32CubeMX toolbar (see *Figure 36*).

Figure 36. Link to social platforms

4.2 New Project window

The New Project window is accessible through the File Menu, or directly through shortcuts from the Home page (see *Figure 37*).

Figure 37. New Project window shortcuts



The main purpose is to select from the STM32 portfolio the microcontroller or board that best fits the user application needs, or simply to get started using an example project.

This window shows three tabs to choose from:

- an **MCU selector** tab (offering a list of target processors)
- a **Board selector** tab (showing a list of STMicroelectronics boards)
- an **Example selector** tab (allows the user to browse and open an example project)

The new project window also features a **Cross selector** tab (allows the user to find, for a given MCU/MPU part number and for a set of criteria, the best replacement within the STM32 portfolio)

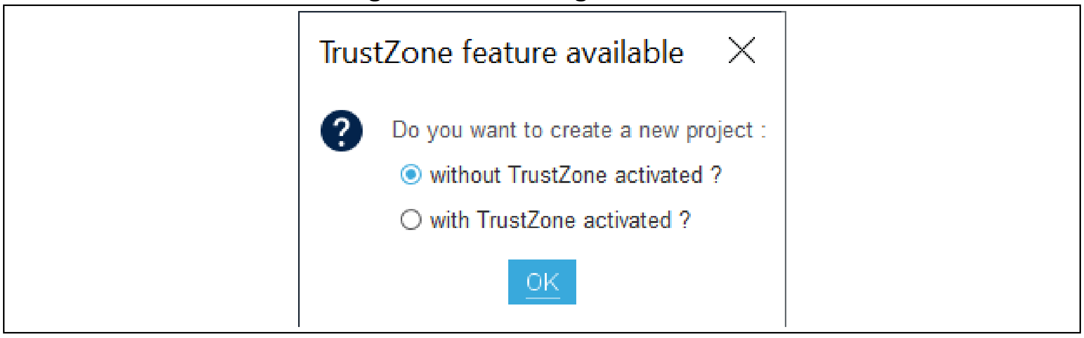
For the STM32L5 series the security features of the Arm Cortex-M33 processor and its Arm® TrustZone®[a] for Armv8-M are combined with ST security implementation. Selecting an STM32L5 MCU or board requires to choose whether to activate Arm® TrustZone® (hardware security) or not (see *Figure 38*). The project is adjusted accordingly:

- if Arm® TrustZone® is not activated, the solution is the same as for other STM32Lx series

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

- if Arm® TrustZone® is activated, the project configuration and the generated project shows specificities related to the security features (refer to dedicated sections in this manual).

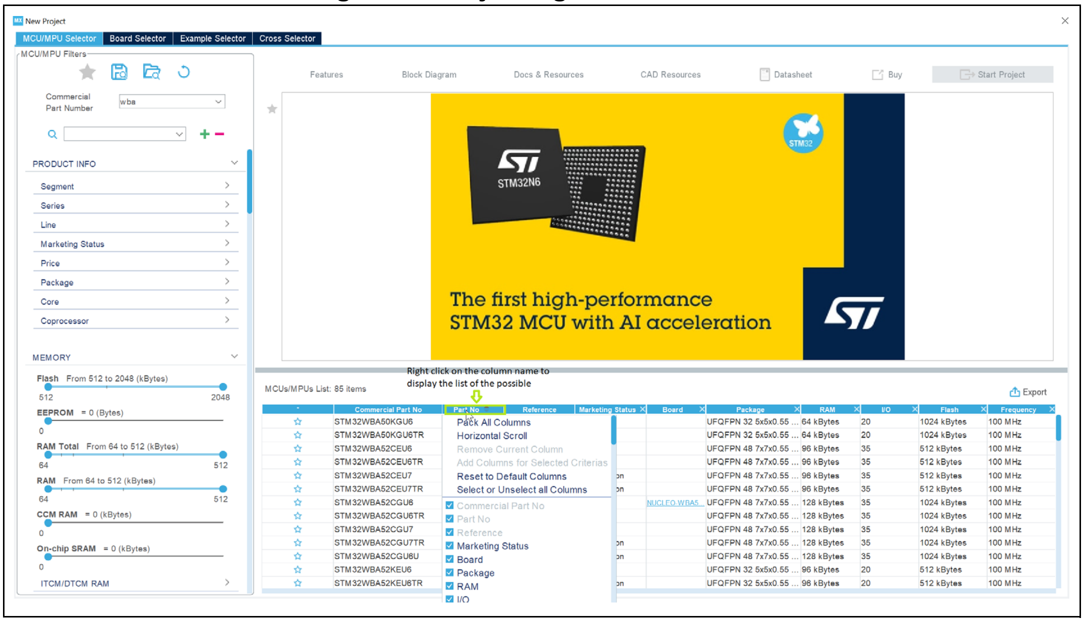
Figure 38. Enabling TrustZone®



The selectors result view can be adjusted (see *Figure 39*):

- Left click the column to sort
- Right click to add/remove columns

Figure 39. Adjusting selector results

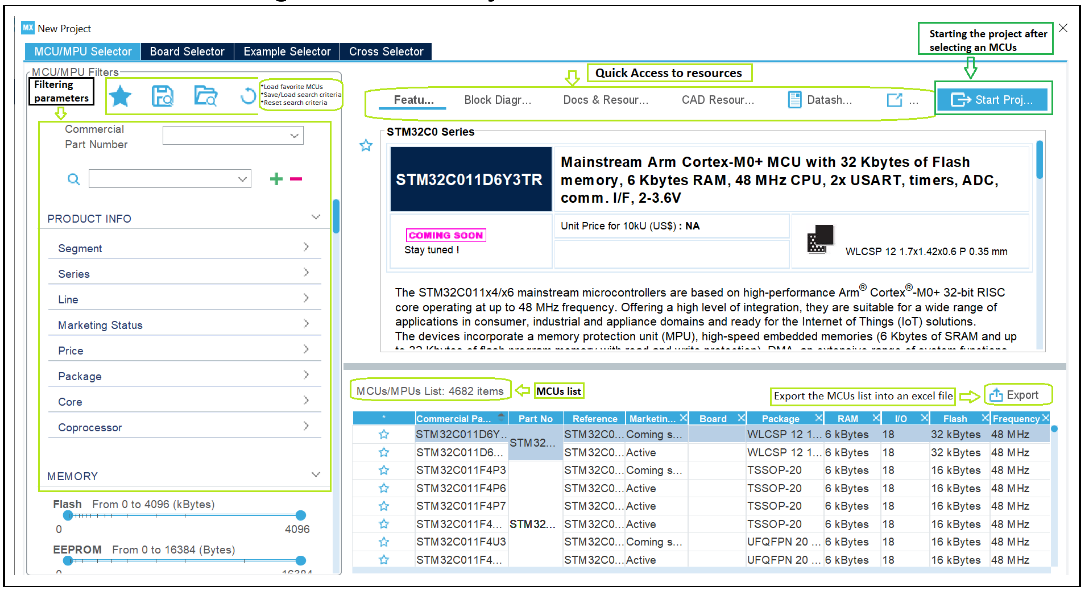


4.2.1 MCU selector

MCU selection

The MCU selector enables filtering on a combination of criteria: series, lines, packages, peripherals, or additional characteristics such as price, memory size, number of I/Os (see *Figure 40*), and on their graphics capabilities as well.

Figure 40. New Project window - MCU selector



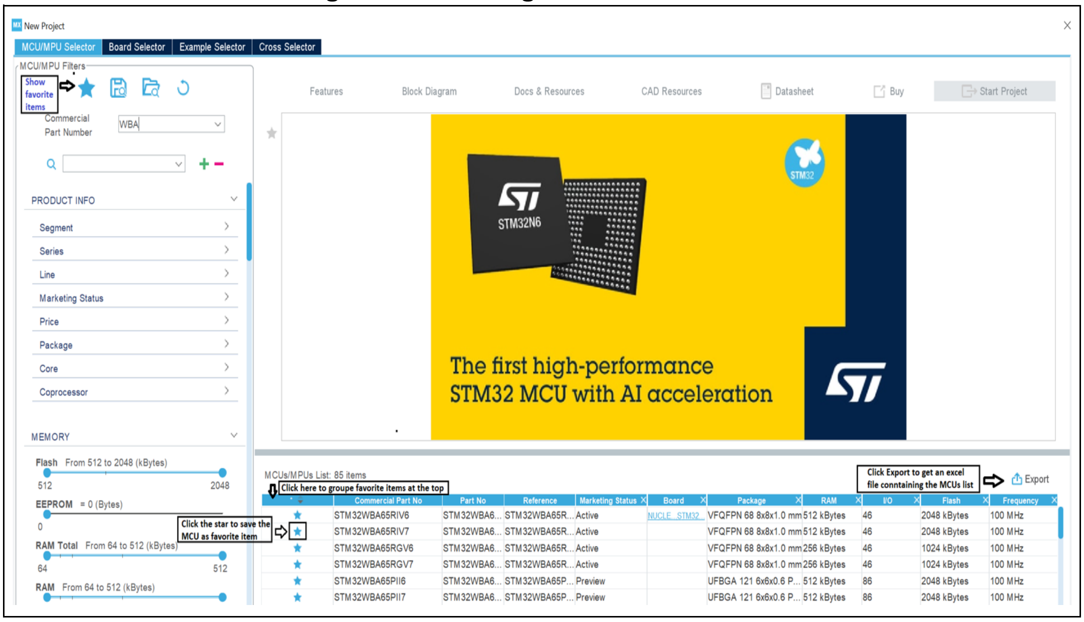
Export to Excel

By clicking on the Export icon, the user can save the MCU table information to an Excel file.

Show favorite MCUs

Clicking the ☆ icon for an MCU from the list marks it as favorite, see *Figure 41*.

Figure 41. Marking an MCU as favorite



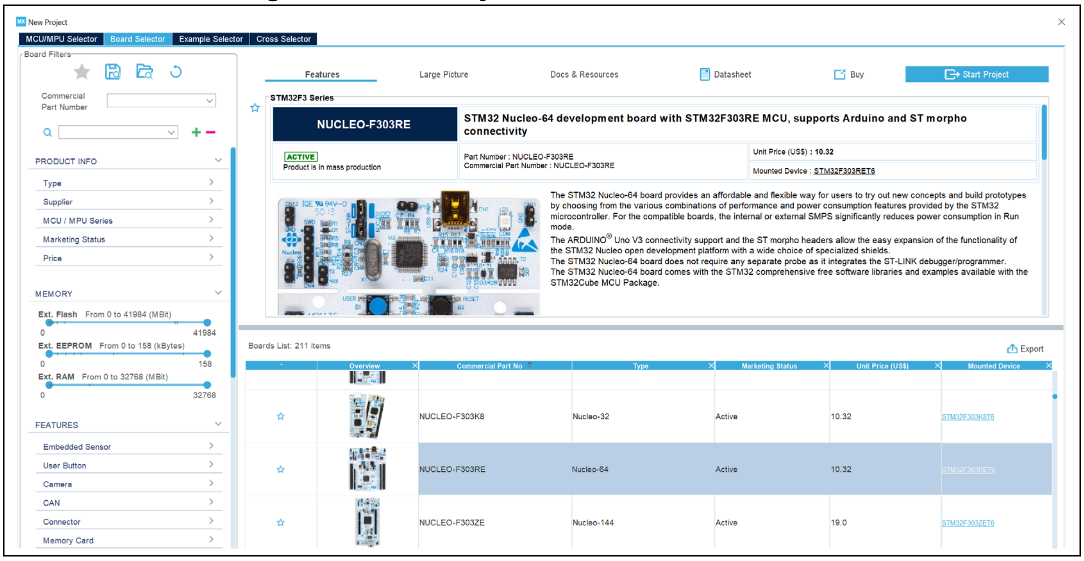
4.2.2 Board selector

The **Board selector** enables filtering on STM32 board types, series and peripherals (see *Figure 42*). Only the default board configuration is proposed. Alternative board configurations obtained by reconfiguring jumpers or by using solder bridges are not supported.

When a board is selected, the **Pinout** view is initialized with the relevant MCU part number along with the pin assignments for the LCD, buttons, communication interfaces, LEDs, and other functions. Optionally, the user can choose to initialize it with the default peripheral modes.

When a board configuration is selected, the signals change to “pinned”, that is, they cannot be moved automatically by STM32CubeMX constraint solver (an user action on the peripheral tree, such as the selection of a peripheral mode, does not move the signals). This ensures that the user configuration remains compatible with the board.

Figure 42. New Project window - Board selector



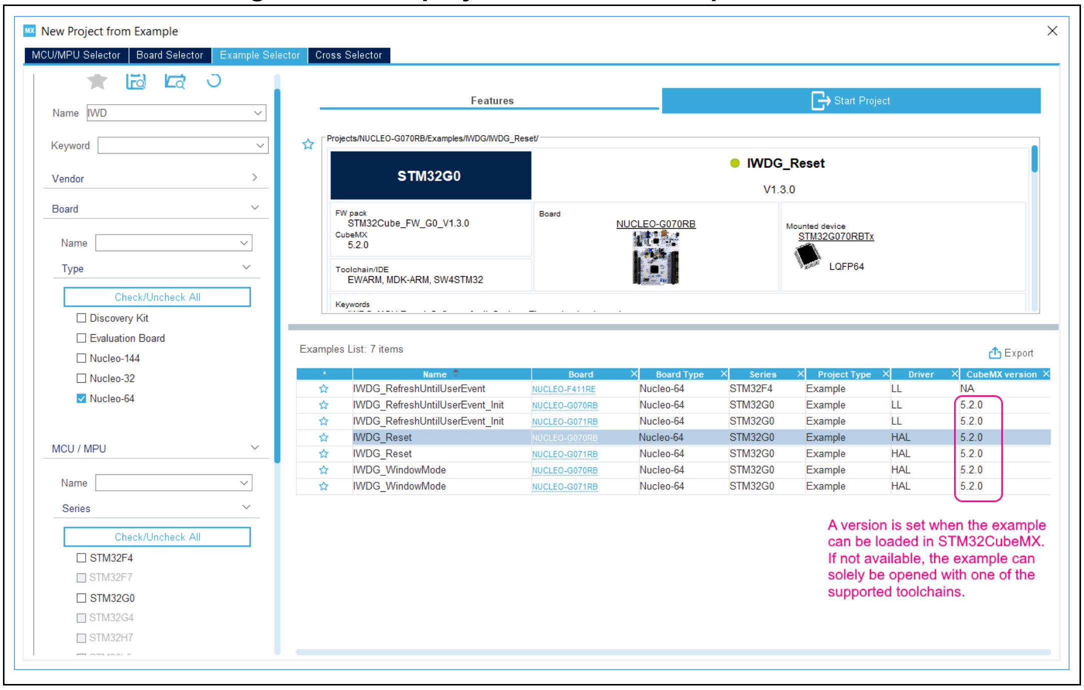
4.2.3 Example selector

The Example selector allows the user to browse a large set of examples and to start a new project from a selected example.

Note: An example is always related to a specific board, and, consequently, for the MCU available with that board.

Thanks to the filter panel it is possible to filter down the example list for a specific board type, series, peripheral or middleware as well as other characteristics (see *Figure 43*).

Figure 43. New project window - Example selector



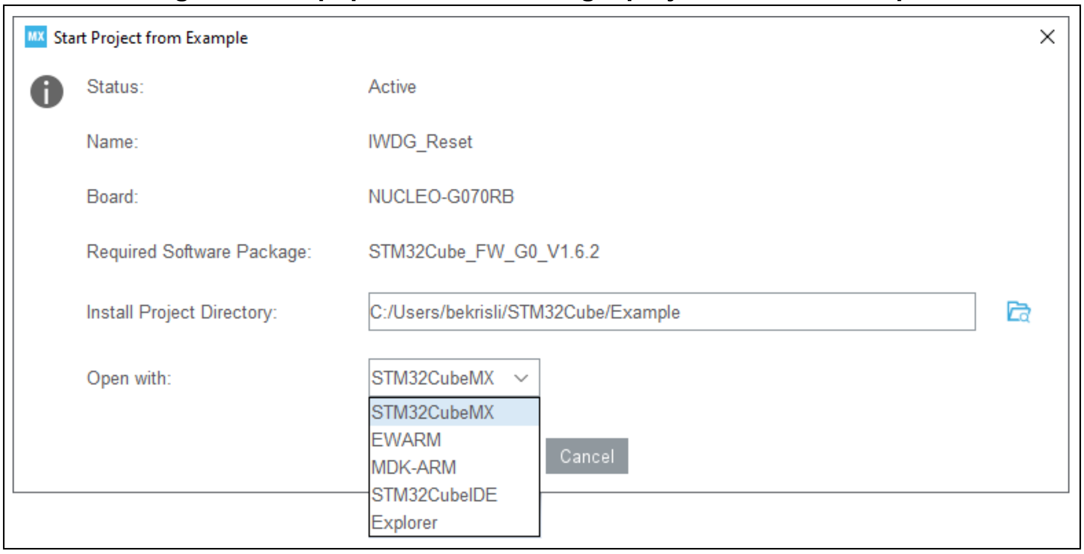
Selecting an example and clicking "Start project" allows STM32CubeMX to copy the example as a new project (the user can change the default location at this stage).

Warning: For some examples the “Start Project” button is shown with an “Under Development” warning icon. Projects created from these examples may be not functional (they do not compile). Fixes are in development.

Several options are available to open the newly created project (see *Figure 44*):

- with STM32CubeMX (available only for examples listed with an STM32CubeMX version set)
- with a File explorer
- with one of the supported toolchains (provided the toolchain is already installed on your computer)

Figure 44. Popup window - Starting a project from an example



Note: If the STM32Cube MCU package needed for the example is missing from the repository, STM32CubeMX automatically starts the download process.

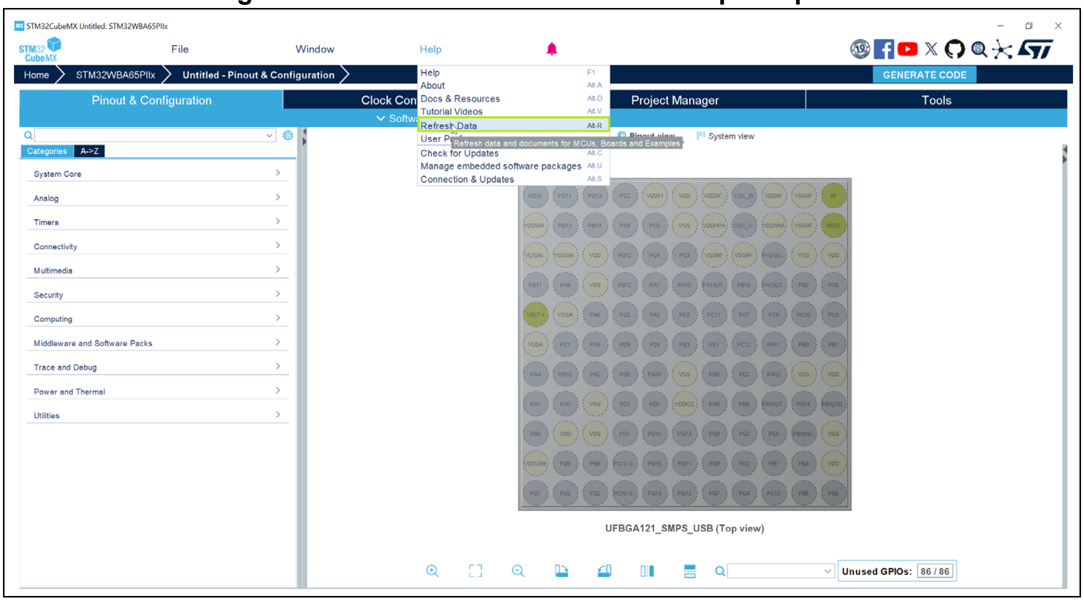
4.2.4 Cross selector

Part number selection

The Cross selector allows users to find the products that best replace the MCU or MPU they are currently using (from ST or other silicon vendors).

To access this functionality, STM32CubeMX data must be up to date. This is ensured using Refresh Data from the Help menu (see *Figure 45*).

Figure 45. Cross selector - Data refresh prerequisite



Clicking “ACCESS TO CROSS SELECTOR” under the “Start my project from Cross Selector” section of the main page opens the New Project window on the Cross selector tab.

Two drop downs menus allow the user to select the vendor and the part number of the product to be compared to (see *Figure 46*). A part number can also be entered partially: STM32CubeMX proposes a list of matching products (see *Figure 47*).

Figure 46. Cross selector - Part number selection per vendor

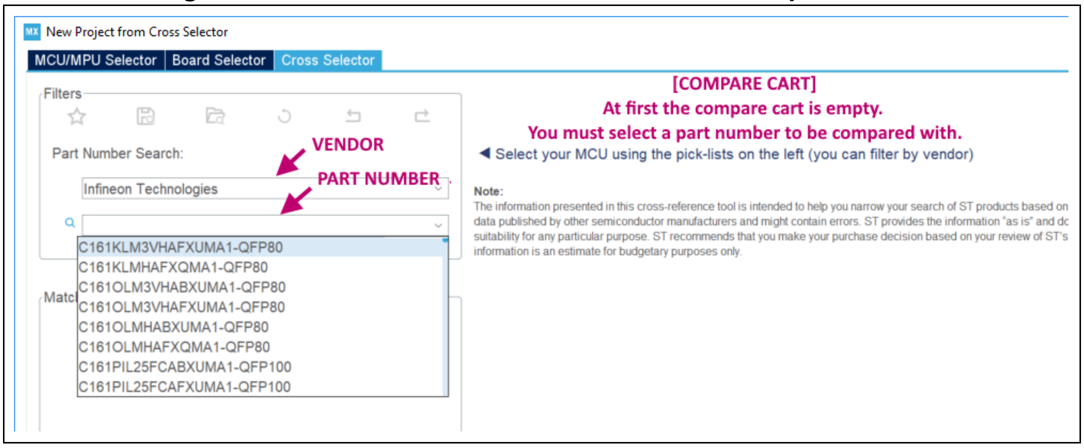
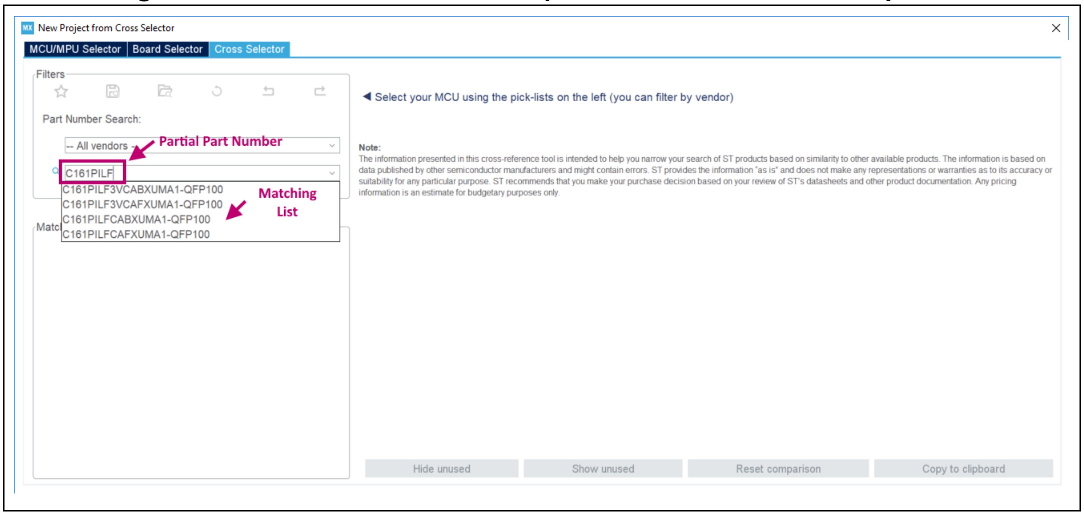


Figure 47. Cross selector - Partial part number selection completion

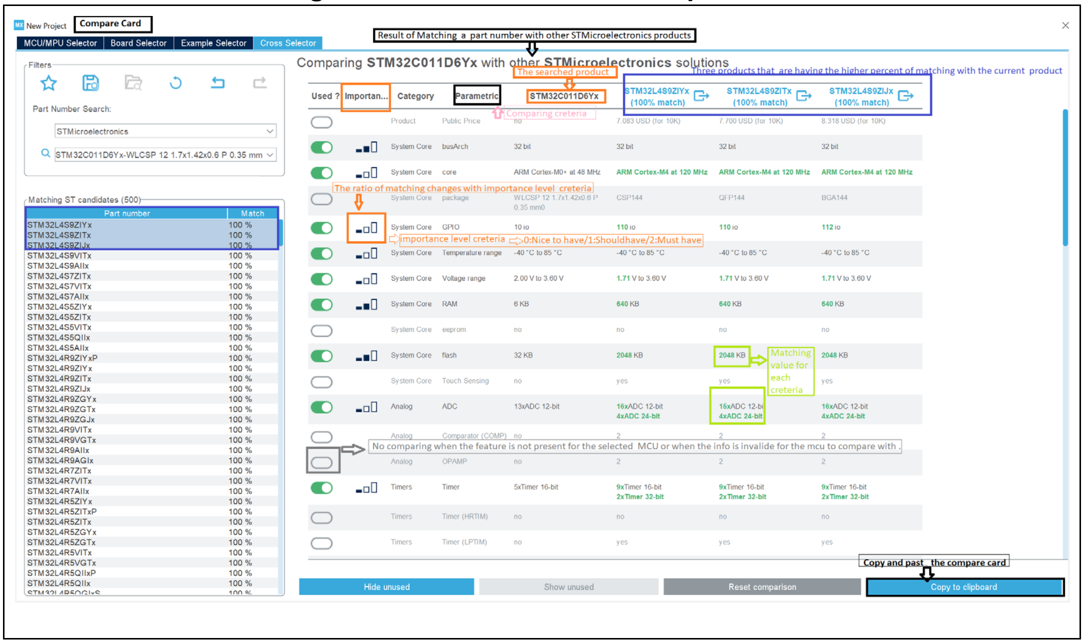


Compare cart

Once a part number is selected, a list of matching ST part number candidates is displayed along with their matching ratio in the Matching ST candidates panel.

By default, the three closest matches are selected and added to the compare cart along with the part number to be compared to (see *Figure 48*).

Figure 48. Cross selector - Compare cart



This selection can be changed anytime in the Matching ST candidates panel.

The comparison can be customized: the features to be used for comparison can be unselected when considered as irrelevant and their level of importance can be adjusted. These choices affect the computed matching ratio.

The comparison is disabled for features that are not supported on the part number to be compared with, or when the feature information is unavailable.

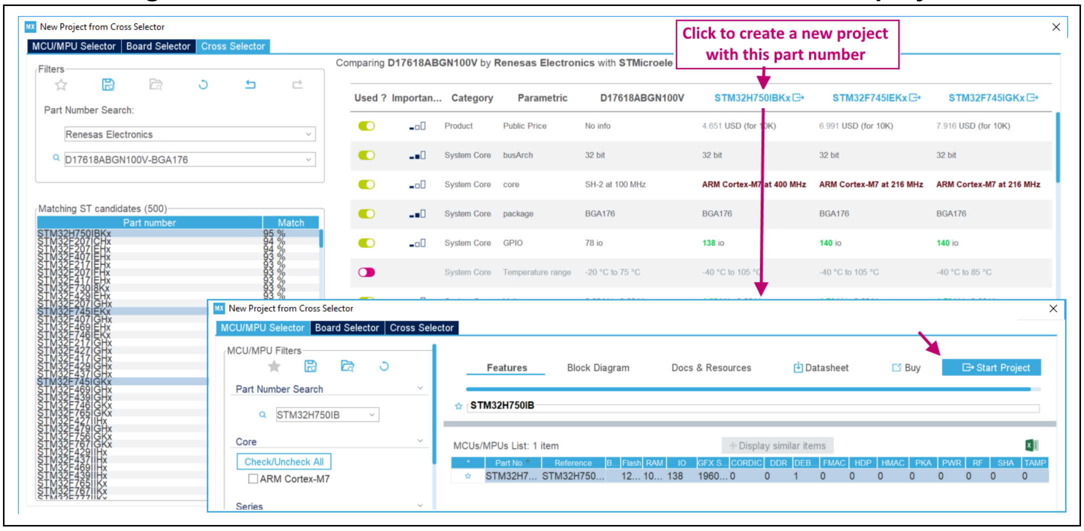
Buttons are available to manipulate and save a copy of the compare cart view:

- to hide criteria not used for the comparison, or show all of them
- to come back to default STM32CubeMX comparison settings
- to copy and paste the current cart view in a document or email.

MCU/MPU selection for a new project

Clicking an STM32 part number from the compare cart selects it in the MCU/MPU Selector tab, and clicking on  creates a new project for that part number (see *Figure 49*).

Figure 49. Cross selector - Part number selection for a new project



Clicking the Cross Selector Tab allows the user to go back to the cart and change the current selection for another part number.

4.3 Project page

Once an STM32 part number or a board has been selected or a previously saved project has been loaded, the project page opens, showing the following set of views (refer to dedicated sections for their detailed description):

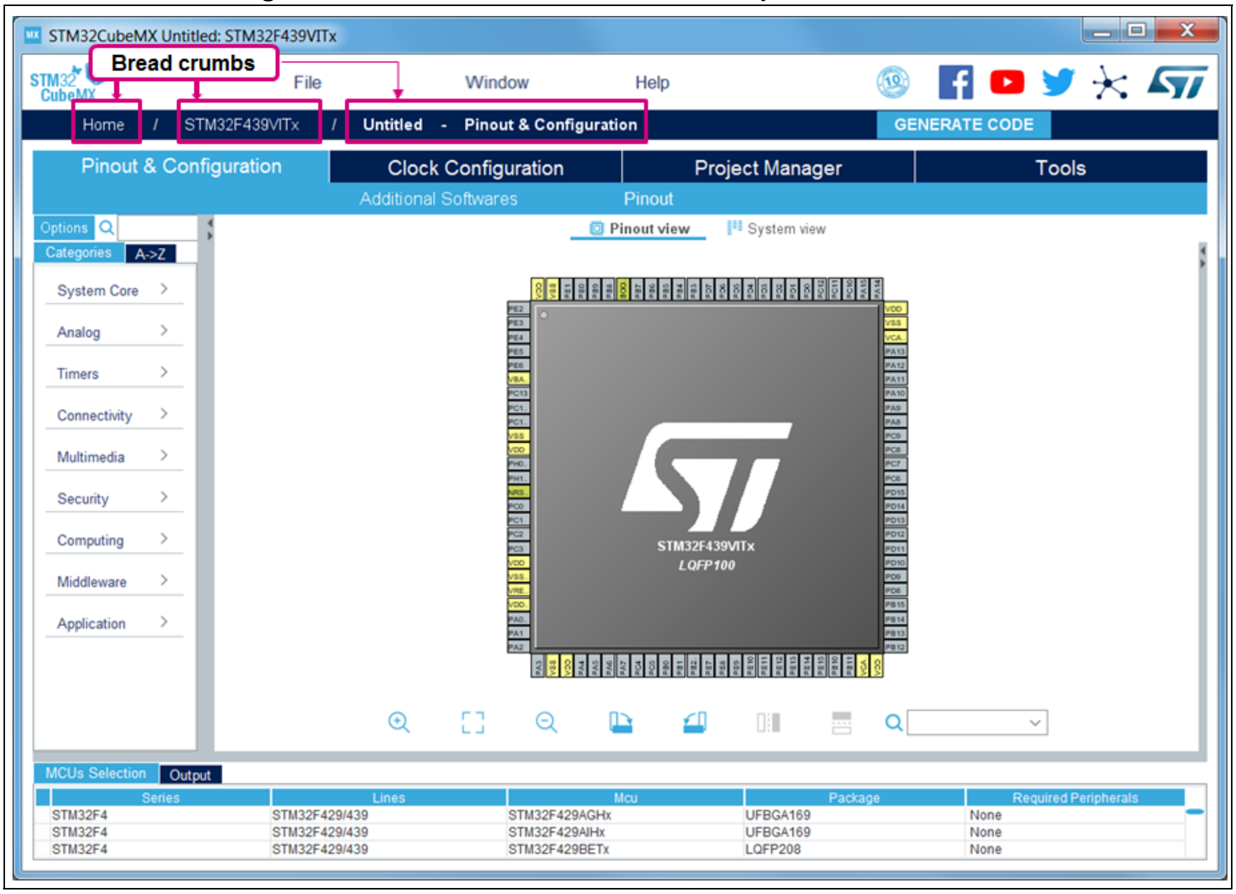
- **Pinout & Configuration**
- **Clock Configuration**
- **Project Manager**
- **Tools**

Users can move across the different views without impacting their project configuration.

A GENERATE CODE button is always accessible for the user to click and allows to generate the code corresponding to the current project configuration. Moreover, thanks to convenient navigation breadcrumbs (see *Figure 50*), the user can detect what its current location is in STM32CubeMX user interface, and can move to other locations:

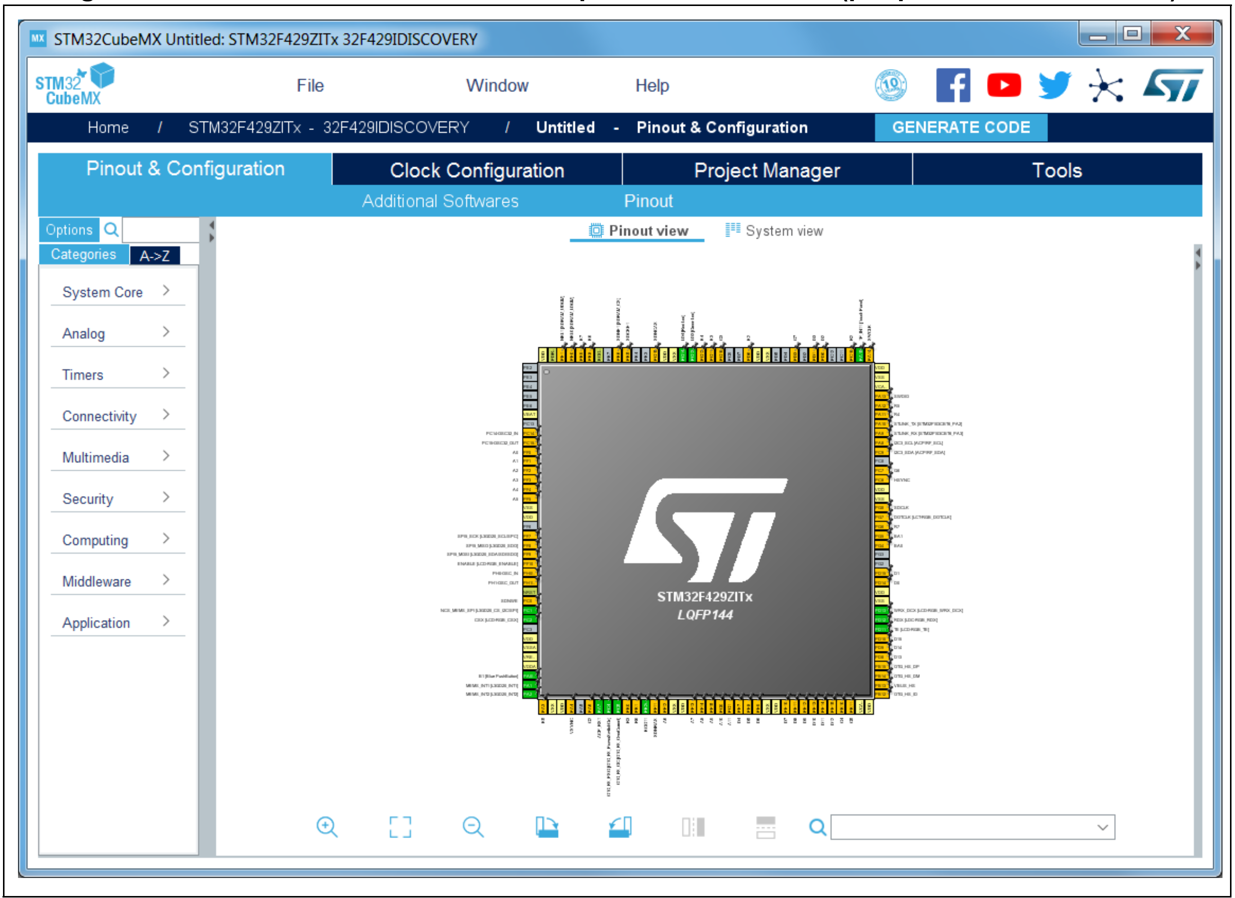
- to the home page by clicking the Home breadcrumb
- to the new project window by clicking the part number
- back to the project page by clicking the project name (or Untitled if the project does not have a name yet).

Figure 50. STM32CubeMX Main window upon MCU selection



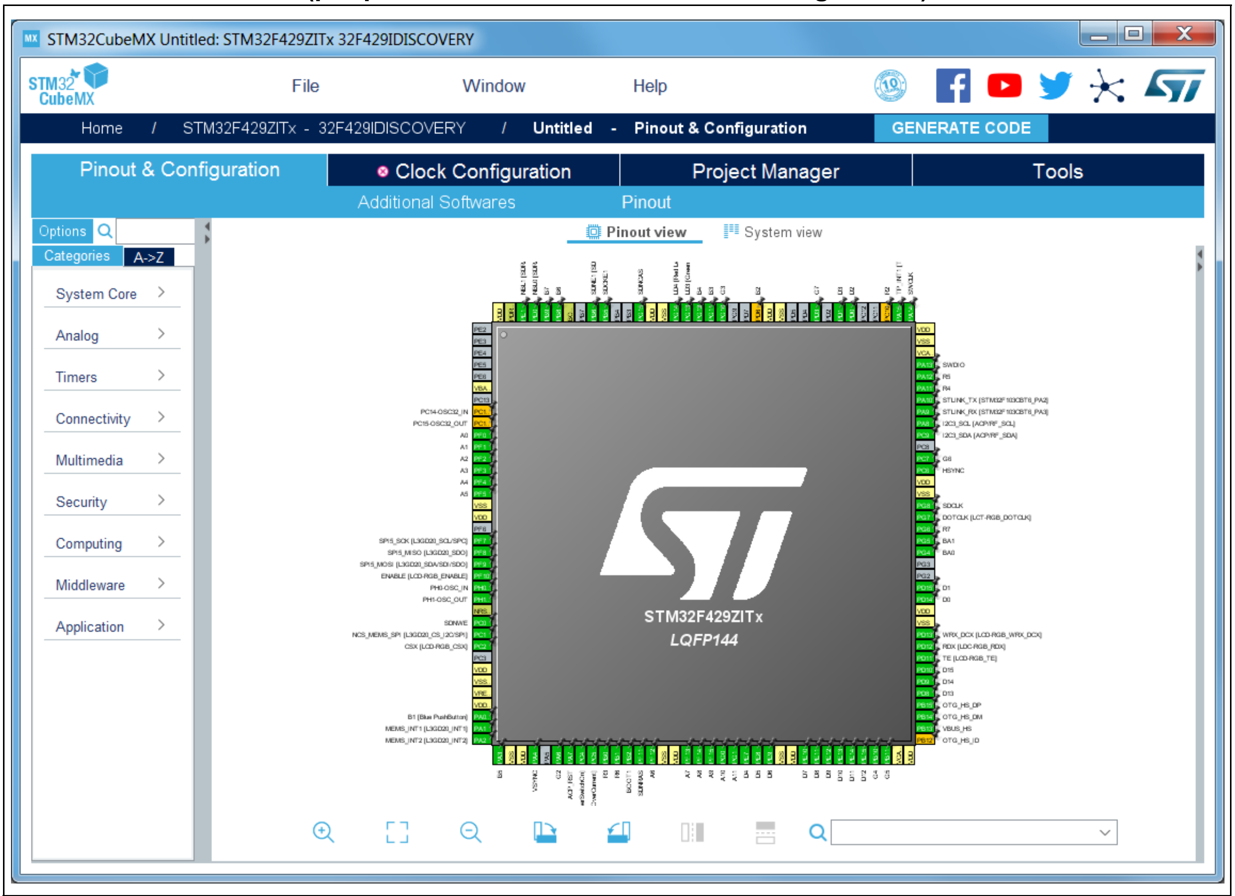
Selecting a board, then answering **No** in the dialog window requesting to initialize all peripherals to their default mode, automatically sets the pinout for this board. However, only the pins set as GPIOs are marked as configured, i.e. highlighted in green, while no peripheral mode is set. The user can then manually select from the peripheral tree the peripheral modes required for its application (see *Figure 51*).

Figure 51. STM32CubeMX Main window upon board selection (peripherals not initialized)



Selecting a board and accepting to initialize all peripherals to their default mode automatically sets both the pinout and the default modes for the peripherals available on the board. This means that STM32CubeMX generates the C initialization code for all the peripherals available on the board and not only for those relevant to the user application (see *Figure 52*).

Figure 52. STM32CubeMX Main window upon board selection (peripherals initialized with default configuration)



4.4 Boot chain (STM32 MPUs)

4.4.1 Boot mode configuration

ST embedded software can support complex architectures (such as OpenSTLinux), which require a complex boot chain, involving several processors, firmware, and a complex boot sequence. An overview is given in the STM32MPU Wiki portal.

The boot mode defines the processor that starts the software, defines the boot sequence scheme, and which software services can be started (such as secure services, also known as TrustZone®).

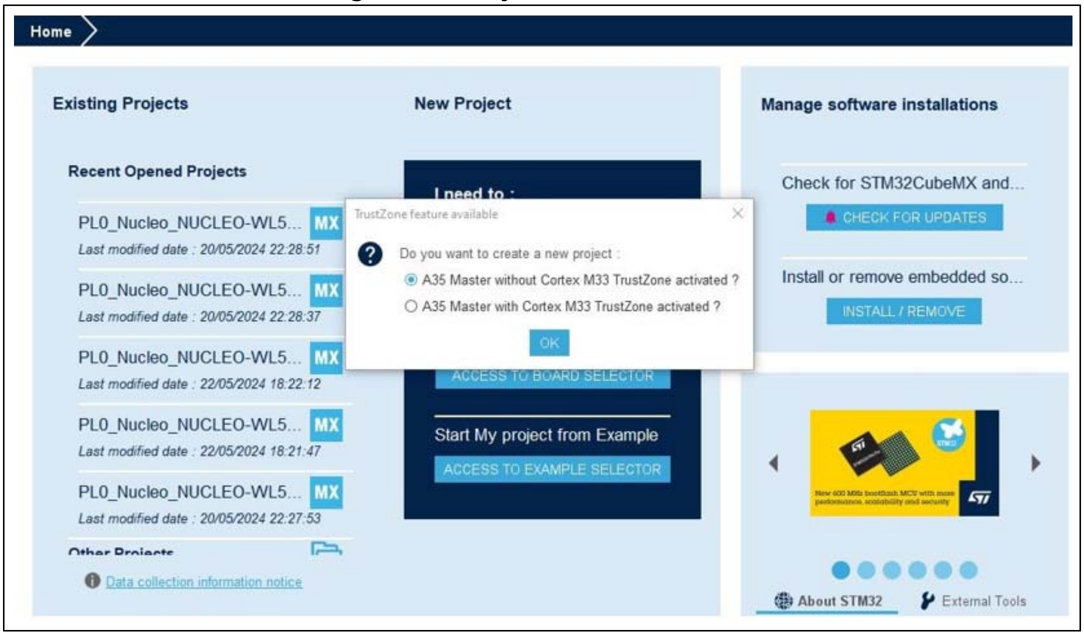
Creating a project for a dual core (Cortex-A35 and Cortex-M33) MPU

The first example uses the following boot mode: Cortex-A35 is the master processor, Cortex-M33 is the secondary one, in non-secure mode.

The master always runs in a secure mode.

- Select an STM32MP257x MPU
- Select the option “with A35 Master without Cortex M33 TrustZone activated?” on the popup window (see *Figure 53*)

Figure 53. Project choice interface

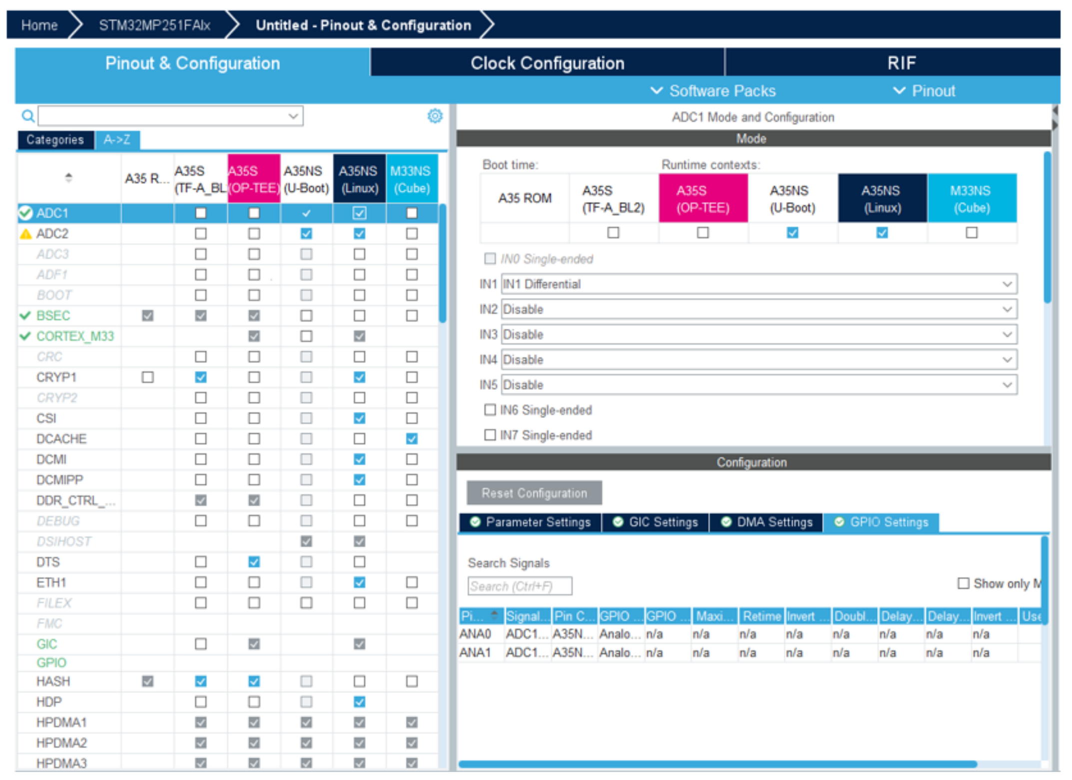


- Six contexts are created in the configuration panel (see *Figure 54*)

Figure 54. Contexts

- The Cortex-A35 runs under the OpenSTLinux operating system. It uses the following firmware:
  - TF-A BL2
  - OP-TEE
  - U-Boot
  - Linux
- The Cortex-M33 is configured using Cube firmware: M33NS Cube FW (HAL & LL)

Figure 55. IPs interface assignment



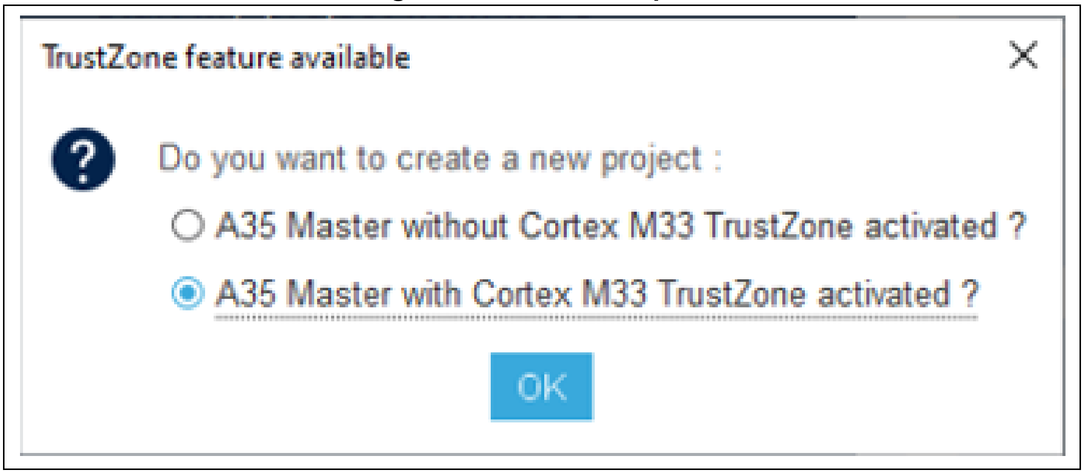
After assigning the IPs context go to "Project Manager" view, save the project, and generate the code.

The second example uses the following boot mode: Cortex-A35 is the master processor, Cortex-M33 core is the secondary one, in secure mode.

The master always runs in a secure mode.

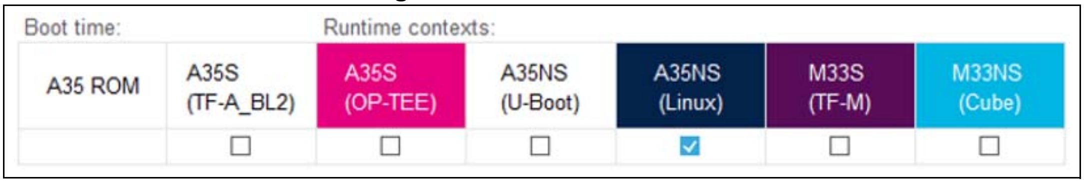
- Select an STM32MP257x MPU
- Select the option "with A35 Master with Cortex M33 TrustZone activated?" on the popup window (see *Figure 56*)

Figure 56. TrustZone option



- Six contexts created in the configuration panel (see *Figure 57*)

Figure 57. Selected context



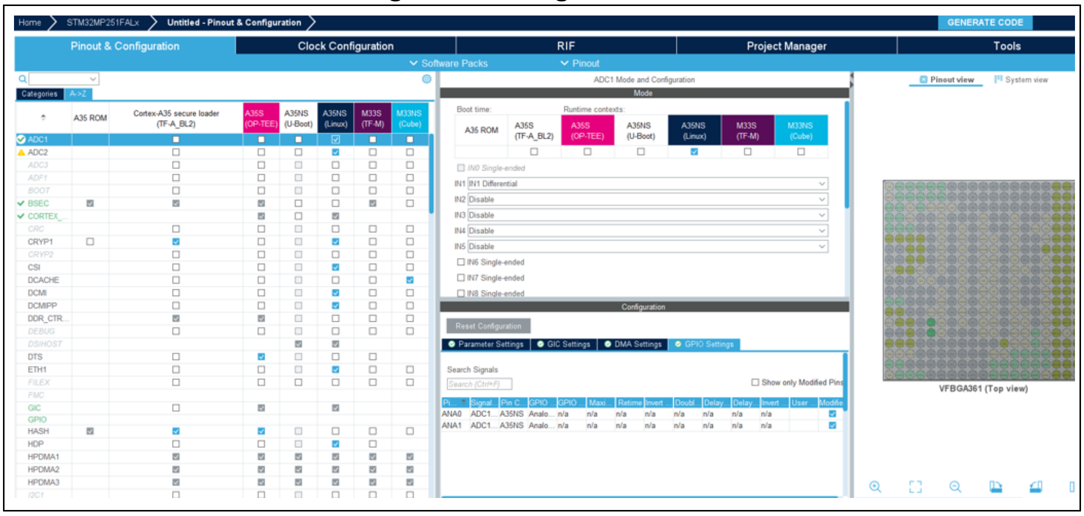
Cortex-A35 runs under the OpenSTLinux operating system. It uses the following firmware:

– TF-A BL2
– OP-TEE
– U-Boot
– Linux

Cortex-M33 secure is configured using Cube firmware: TF-M

To assign IPs context go to “Pinout & Configuration” and configure IPs.

Figure 58. Assign IP context



After assigning the IPs context go to “Project Manager” view, save the project, and then generate code.

4.4.2 Coprocessor initializers (STM32MP2x)

The STM32MP2xx comes with two possible coprocessors (Cortex-M33 or Cortex-M0+). STM32CubeMX manages only Cortex-M33.

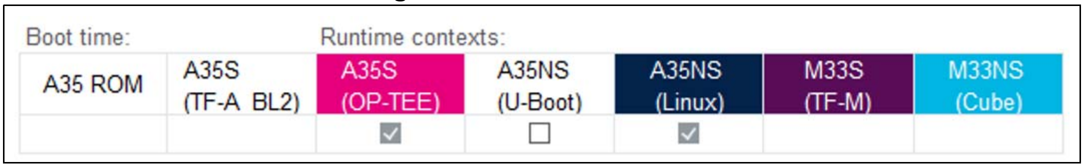
The STM32CubeMX tool indicates which programs running on the main processor can be started, or if to use the secondary processor.

When the system source code is generated, the settings that determine how the main processor can use the coprocessor are included in the device tree. These settings are found in the “rproc” sections (nodes) for each software component that can interact with the coprocessor. This ensures that, when the system is running, it knows how to handle the coprocessor according to the predefined configuration.

As an example:

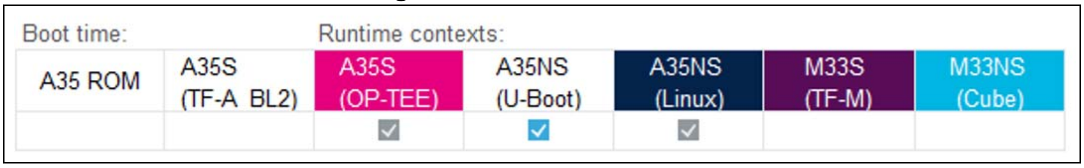
- OP-TEE is eligible to load the main processor.

Figure 59. OP-TEE selected



- Linux Kernel is eligible to load for the main processor.
- U-Boot will be available when Linux is selected.

Figure 60. U-Boot selection



4.4.3 Boot device selection (STM32MP25)

The term boot device refers to any storage device from which a microcontroller can load the initial software used to boot up the system. This initial software is part of the boot process that starts the computer and loads the operating system.

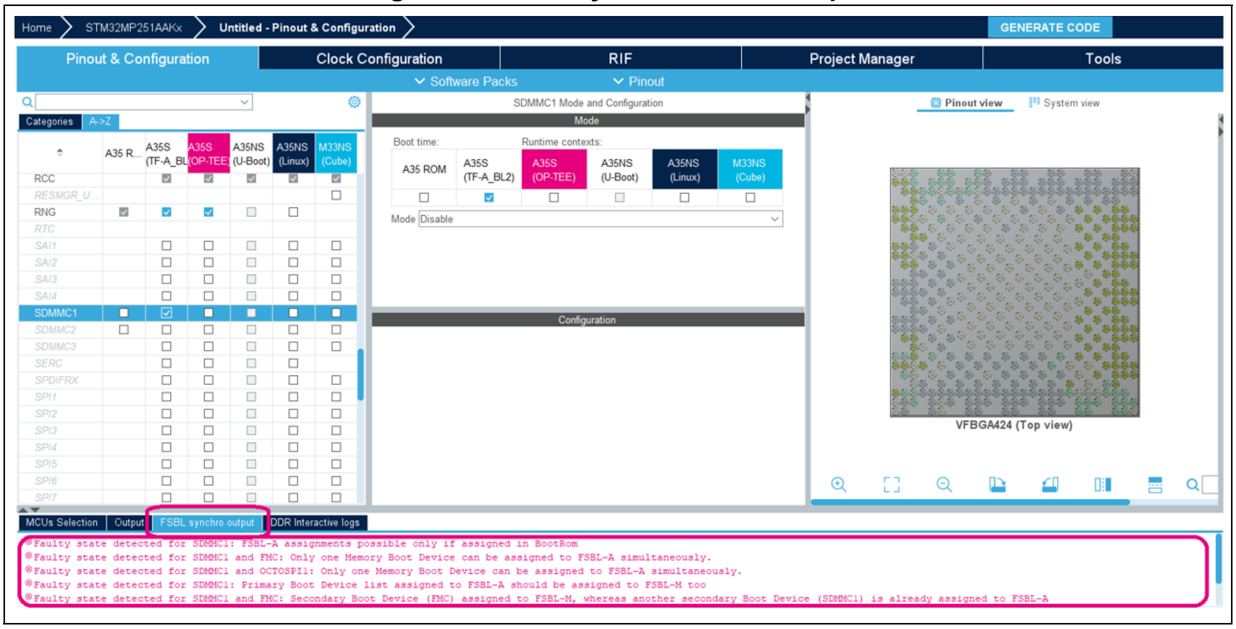
STM32CubeMX does not handle the configuration of the pins used by STM32 devices to select the boot source. To configure a correct boot, ensure that the boot device settings align with the boot pins configuration, programmed in the MCU hardware. This requires checking the datasheet or reference manual, to understand the boot pin settings, and then manually configuring the system to match those settings.

A boot device must be assigned to the ROM firmware and the early-stage Boot Loader (such as TF-A BL2 for OpenSTLinux).

When configuring a microcontroller, consider the constraints that affect the choice of boot devices, and their dependency upon the selected boot mode. STM32CubeMX checks the boot configuration of against a set of constraints to ensure that the system boots properly. This service is called Flexible Software Loader synchronization verification. The results of this verification are displayed in a dedicated output window (FSBL synchro output), providing developers with important diagnostic information.

The “FSBL synchro output” panel is displayed with the rule “Faulty state detected for SDMMC1: FSBL-A assignments possible only if assigned in BootRom”. Users can refer to this panel to align any misconfigurations.

Figure 61. FSBL synchronization output



4.5 Pinout & Configuration view

The **Pinout & Configuration** view comes with the following main panels, function and menu:

- A **Component list** that can be visualized in alphabetical order and per categories. By default, it consists of the list of peripheral and middleware that the selected MCU supports. Selecting a component from that list opens two additional panels (**Mode** and **Configuration**) that allow the user to set its functional mode and configure the initialization parameters that will be included in the generated code.
- A **Pinout view** that shows a graphic representation of the pinout for the selected package (e.g. BGA, QFP) where each pin is represented with its name (e.g. PC4) and its current alternate function assignment, if any.
- A **System view** that gives an overview of all the software configurable components: GPIOs, peripherals, middleware and additional software components. Clickable buttons allow opening the configuration options for the given component (Mode and Configuration panels). The button icon color reflects the status of the configuration status.
- A **Software Packs** menu with two sub-menus:
  – **Select Components** to select, for the current project, software components not available by default. This selection updates the **Pinout & Configuration** view accordingly
  – **Manage Software Packs** to install/uninstall software packs.
- An **Additional Software** function that allows to select, for the current project, software components that are not available by default. Selecting an additional software component updates the **Pinout & Configuration** view accordingly.
- A **Pinout** menu that allows the user to perform pinout related actions such as clear pinout configuration or export pinout configuration as csv file.

Tips

- You can resize the panels: hovering the mouse over a panel border displays a two-ended arrow: right-click and pull in a direction to extend or reduce the panel.
- You can show/hide the Configuration, Mode, Pinout and System views using the open and close arrows.

4.5.1 Component list

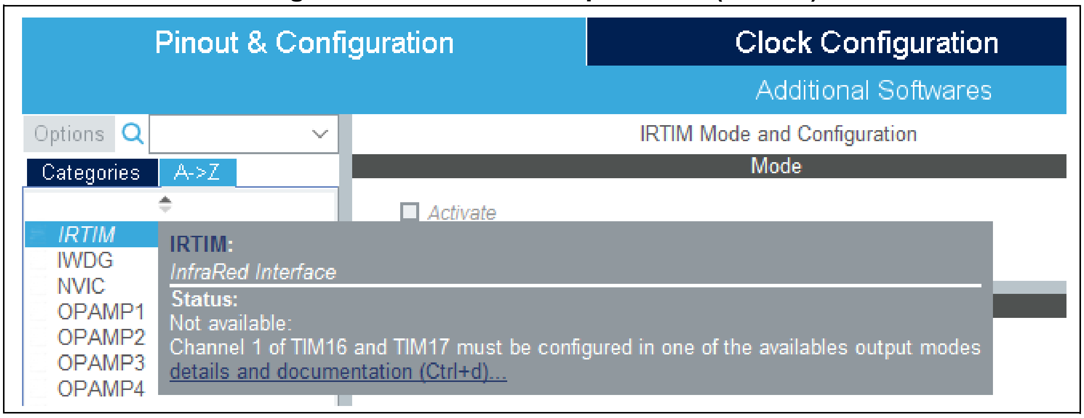
The component list shows all the components available for the project. Selecting a component from the component list, opens the Mode and Configuration panels.

Contextual help

The **Contextual Help** window is displayed when hovering the mouse over a peripheral or a middleware short name.

By default, the window displays the extended name and source of configuration conflicts if any (see *Figure 62*).

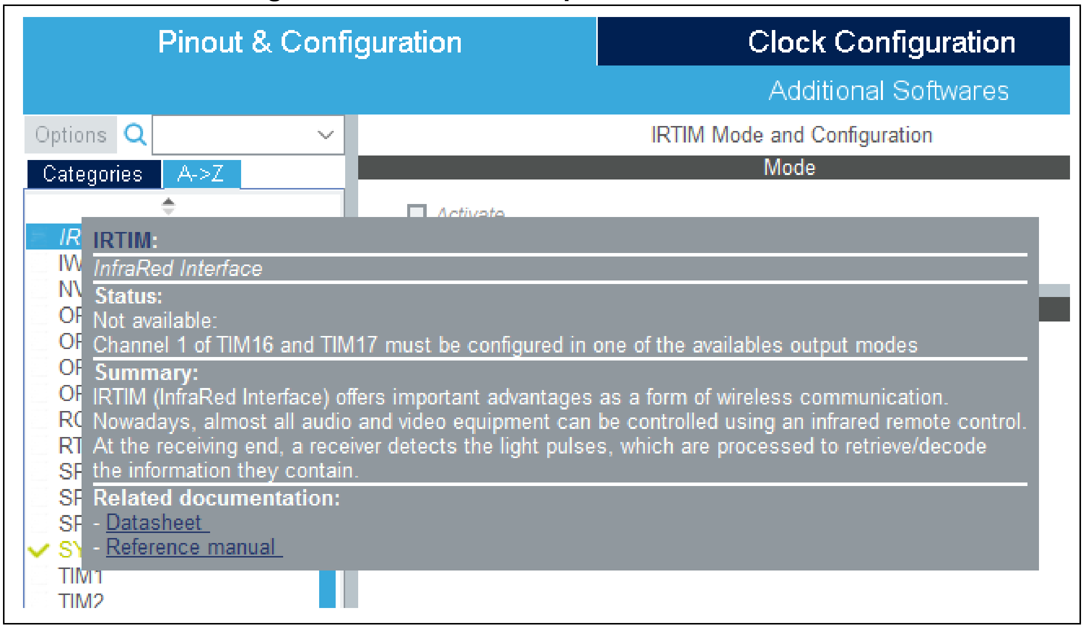
Figure 62. Contextual Help window (default)



Clicking the *details and documentation* link (or CTRL+d) provides additional information such as summary and reference documentation links (see *Figure 63*). For a given peripheral, clicking *Datasheet* or *Reference manual* opens the corresponding document, stored in STM32CubeMX repository folder, at the relevant chapter. Since microcontrollers datasheets and reference manuals are downloaded to STM32CubeMX repository only upon user request, a functional Internet connection is required:

- To check your Internet connection, open the **Connection** tab from the **Help > Updater Settings** menu.
- To request the download of reference documentation for the currently selected microcontroller, click **Refresh** from the **Help > Refresh Data** menu window.

Figure 63. Contextual Help detailed information



Icons and color schemes

Table 5 shows the icons and color scheme used in the component list view and the corresponding color scheme in the Mode panel.

Table 5. Component list, mode icons and color schemes

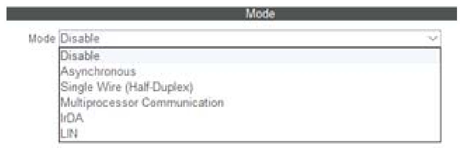
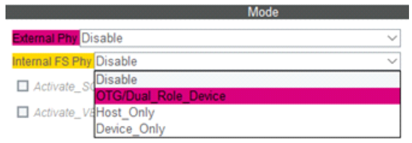
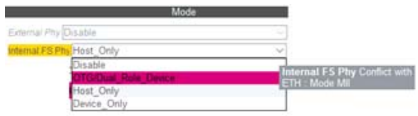
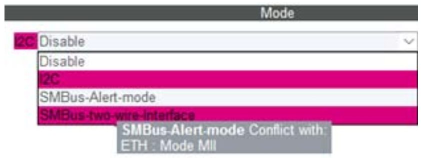
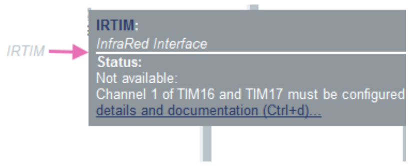
Display	Component status	Corresponding Mode view / Tooltips
Plain black text Example: UART5	The peripheral is not configured (no mode is set) and all modes are available.	
Gray italic text Example: *LWIP*	Peripheral is not available because some constraints are not solved. See tooltip.	
 Example:: ETH	The peripheral is configured (at least one mode is set) and all other modes are available. The green check mark indicates that all parameters are properly configured, a cross indicates they are not.	

Table 5. Component list, mode icons and color schemes (continued)

Display	Component status	Corresponding Mode view / Tooltips
Example: USB_OTG_HS	The peripheral is not configured (no mode is set) and at least one of its modes is unavailable.	
Example: USB_OTG_HS	The peripheral is configured (one mode is set) and at least one of its other modes is unavailable.	
Example: I2C2	The peripheral is not configured (no mode is set) and no mode is available. Move the mouse over the peripheral name to display the tooltip describing the conflict.	
Example: IRTIM	Peripheral is not available because of constraints.	

4.5.2 Component Mode panel

Select a component from the component list on the left panel to open the **Mode** panel.

The **Mode** panel helps the user configuring the MCU pins based on a selection of peripherals and of their operating modes. Since STM32 MCUs allow a same pin to be used by different peripherals and for several functions (alternate functions), the tool searches for the pinout configuration that best fits the set of peripherals selected by the user. STM32CubeMX highlights the conflicts that cannot be solved automatically (see *Table 5*).

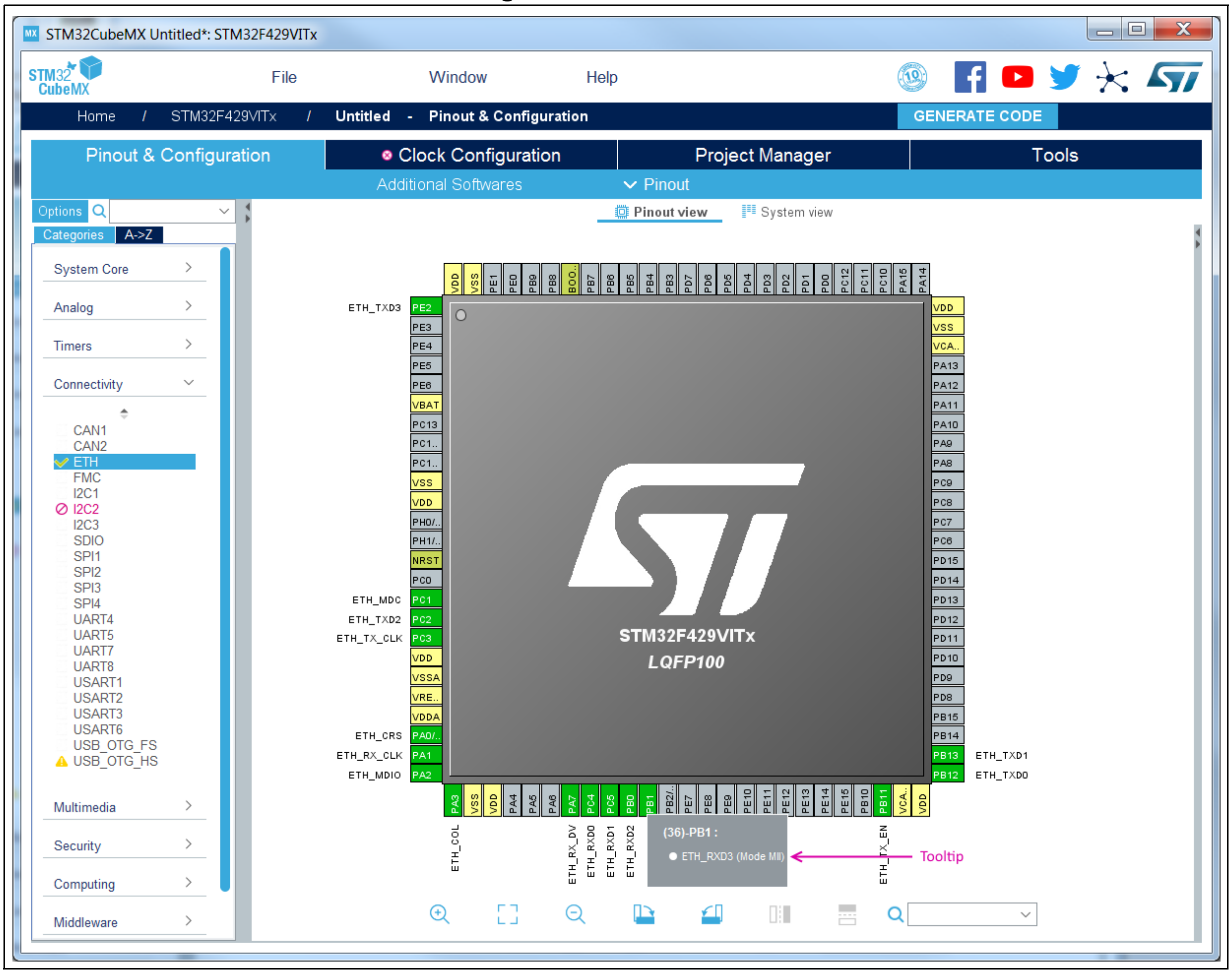
The **Mode** panel also allows to enable middleware and other software components for the project.

Note: For some middleware (USB, FATS, LwIP), a peripheral mode must be enabled before activating the middleware mode. Tooltips guide the user through the configuration. For FatFs, a user-defined mode has been introduced. This allows STM32CubeMX to generate FatFs code without a predefined peripheral mode. Then, it is up to the user to connect the middleware with a user-defined peripheral by updating the generated user_diskio.c/.h driver files with the necessary code.

4.5.3 Pinout view

Select **Pinout view** to show for the selected part number, a graphic representation of the pinout for the selected package (such as. BGA, QFP), where each pin is represented with its name (such as PC4), its configuration state and its current alternate function assignment, if any (such as ETH_MII_RXD0). See *Figure 64* for an example.

Figure 64. Pinout view



The **Pinout** view is automatically refreshed to match the user's component configuration performed in the **Mode** panel.

Assigning pins directly through the **Pinout** view instead of the **Mode** panel requires a good knowledge of the MCU since each individual pin can be assigned to a specific function.

Tips and tricks

See *Table 2* for list of menus and shortcuts.

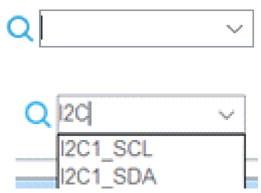
- Use the mouse wheel to zoom in and out.
- Click and drag the chip diagram to move it.
- Click best fit to reset it to best suited position and size.
- Use Pinout > Export pinout menus to export the pinout configuration as .csv text format.
- Some basic controls, such as insuring consistency for blocks of pins, are built-in. See *Appendix A* for details.

4.5.4 Pinout menu and shortcuts

Table 6. Pinout menu and shortcuts

Name or Icon	Shortcut	Description
Keep Current Signals Placement	*Ctrl-K*	Prevents moving pin assignments to match a new peripheral operating mode. It is recommended to use the new pinning feature that can block each pin assignment individually and leave this checkbox unchecked.
Show User Label	None	Displays user defined labels in the Pinout view.
Undo Mode and pinout	*Ctrl-Z*	Undoes last configuration steps (one by one).
Redo Mode and pinout	*Ctrl-Y*	Redoes steps that have been undone (one by one). **Warning** (limitation): configurations in the platform settings tabs are not restored.
Disable All Modes	*Ctrl-D*	Resets to “Disabled” all peripherals and middleware modes that have been enabled. The pins configured in these modes (green color) are consequently reset to “Unused” (gray color). Peripheral and middleware labels change from green to black (when unused) or gray (when not available).
Clear Pinouts	*Ctrl-P*	Clears user pinout configuration in the Pinout view. Note that this action puts all configured pins back to their reset state and disables all the peripheral and middleware modes previously enabled (whether they were using signals on pins or not).
Pins/Signals Option	*Ctrl-O*	Opens a window showing the list of all the configured pins together with the name of the signal on the pin and a Label field allowing the user to specify a label name for each pin of the list. For this menu to be active, at least one pin must have been configured. Click the pin icon to pin/unpin signals individually. Select multiple rows then right click to open contextual menu and select action to pin or unpin all selected signals at once. Click column header names to sort alphabetically by name or according to placement on MCU.
Clear Single Mapped Signals	*Ctrl-M*	Clears signal assignments to pins for signals that have no associated mode (highlighted in orange and not pinned).

Table 6. Pinout menu and shortcuts (continued)

Name or Icon	Shortcut	Description
List Pinout Compatible MCUs	*Alt-L*	Provides a list of MCUs that best match the pin configuration of the current project. The matching can be: – An exact match – A partial match with hardware compatibility: pin locations are the same, pin names may have been changed – A partial match without hardware compatibility: all signals can be mapped but not all at the same pin location Refer to *Section 15*.
Export pinout with Alternate functions	-	Generates pin configuration as a .csv text file including alternate functions information.
Export pinout without Alternate functions	*Ctrl-U*	Generates pin configuration as a .csv text file excluding alternate functions information.
Reset used GPIOs	*Alt-G*	Opens a window to specify the number of GPIOs to be freed among the total number of GPIO pins that are configured.
Set unused GPIOs	*Ctrl-G*	Opens a window to specify the number of GPIOs to be configured among the total number of GPIO pins that are not used yet. Specify their mode: Input, Output or Analog (recommended configuration to optimize power consumption). **Caution:** Before using this menu, make sure that debug pins (available under SYS peripheral) are set to access microcontroller debug facilities.
Layout reset	-	-
	-	Zooms-in the pinout view.
	-	Adjusts the chip pinout diagram to the best fit size.
	-	Zooms-out the pinout view.
	-	Rotates 90 degrees clock wise.
	-	Rotate 90 degrees counter-clock wise.
	-	Flips horizontally between bottom view and top view.
	-	Flips vertically between bottom view and top view.
	-	This Search field allows the user to search the Pinout view for a pin name, a signal name, a signal label or an alternate pin name When it is found, the pin or set of pins matching the search criteria blinks on the Pinout view. Click the Pinout view to stop blinking.

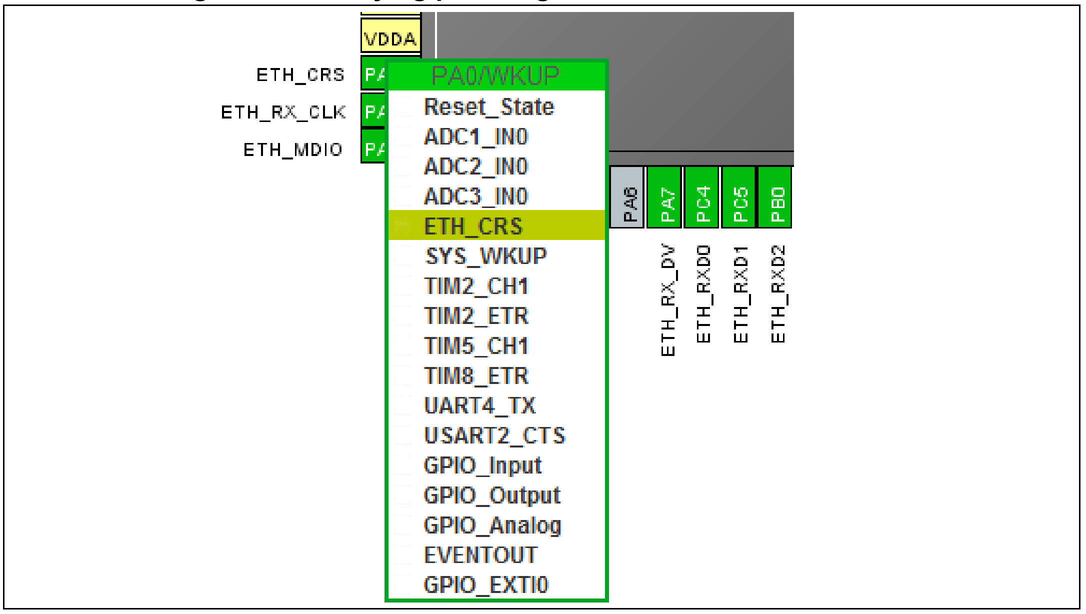
4.5.5 Pinout view advanced actions

Manually modifying pin assignments

To manually modify a pin assignment, follow the sequence below:

1. Click the pin in the **Pinout** view to display the list of all other possible alternate functions together with the current assignment highlighted in blue (see *Figure 65*).
2. Click to select the new function to assign to the pin.

Figure 65. Modifying pin assignments from the Pinout view



Manually remapping a function to another pin

To manually remap a function to another pin, follow the sequence below:

1. From the **Pinout** view, hold down the CTRL key then left-click on the pin and hold: if any pins are possible for relocation, they are highlighted in blue and blinking.
2. Drag the function to the target pin.

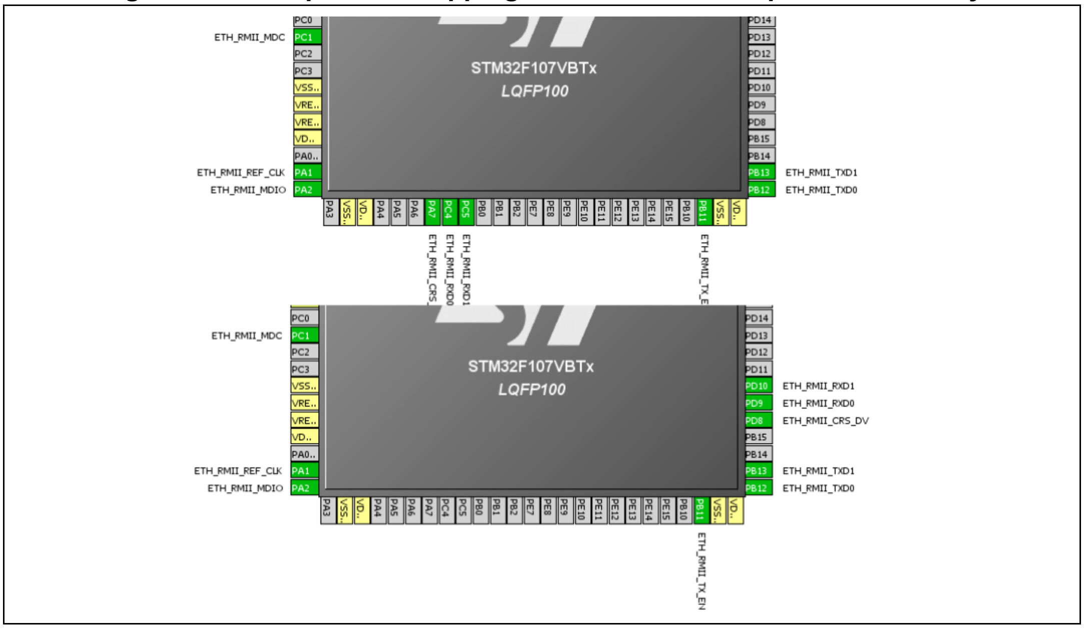
Caution: A pin assignment performed from the **Pinout** view overwrites any previous assignment.

Manual remapping with destination pin ambiguity

For MCUs with block of pins consistency (STM32F100x / F101x / F102x / F103x and STM32F105x / F107x), the destination pin can be ambiguous, e.g. there can be more than one destination block including the destination pin. To display all the possible alternative remapping blocks, move the mouse over the target pin.

Note: A "block of pins" is a group of pins that must be assigned together to achieve a given peripheral mode. As shown in Figure 66, two blocks of pins are available on STM32F107xx MCUs to configure the Ethernet peripheral in RMII synchronous mode: {PC1, PA1, PA2, PA7, PC4, PC5, PB11, PB12, PB13, PB5} and {PC1, PA1, PA2, PD10, PD9, PD8, PB11, PB12, PB13, PB5}.

Figure 66. Example of remapping in case of block of pins consistency



Resolving pin conflicts

To resolve the pin conflicts that may occur when some peripheral modes use the same pins, STM32CubeMX attempts to reassign the peripheral mode functions to other pins. The peripherals for which pin conflicts cannot be solved are highlighted in fuchsia with a tooltip describing the conflict.

If the conflict cannot be solved by remapping the modes, the user can try the following:

- If the ☑ Keep Current Signals Placement box is checked, try to select the peripherals in a different sequence.
- Uncheck the **Keep Current Signals Placement** box and let STM32CubeMX try all the remap combinations to find a solution.
- **Manually remap** a mode of a peripheral when you cannot use it because there is no pin available for one of the signals of that mode.

4.5.6 Keep Current Signals Placement

This checkbox is available from the **Pinout** menu. It can be selected or deselected at any time during the configuration. It is unselected by default.

It is recommended to keep the checkbox unchecked for an optimized placement of the peripherals (maximum number of peripherals concurrently used).

The **Keep Current Signals Placement** checkbox should be selected when the objective is to match a board design.

Keep Current Signals Placement is unchecked

This allows STM32CubeMX to remap previously mapped blocks to other pins in order to serve a new request (selection of a new peripheral mode or a new peripheral mode function) which conflicts with the current pinout configuration.

Keep Current Signals Placement is checked

This ensures that all the functions corresponding to a given peripheral mode remain allocated (mapped) to a given pin. Once the allocation is done, STM32CubeMX cannot move a peripheral mode function from one pin to another. New configuration requests are served if feasible within current pin configuration.

This functionality is useful to:

- lock all the pins corresponding to peripherals that have been configured using the **Peripherals** panel
- maintain a function mapped to a pin while doing manual remapping from the **Pinout** view.

Tip

If a mode becomes unavailable (highlighted in fuchsia), try to find another pin remapping configuration for this mode by following the steps below:

1. From the Pinout view, deselect the assigned functions one by one until the mode becomes available again.
2. Then, select the mode again and continue the pinout configuration with the new sequence (see *Appendix A: STM32CubeMX pin assignment rules* for a remapping example). This operation being time consuming, it is recommended to deselect the **Keep Current Signals Placement** checkbox.

Note: Even if Keep Current Signals Placement is unchecked, GPIO_ functions (excepted GPIO_EXTI functions) are not moved by STM32CubeMX.

4.5.7 Pinning and labeling signals on pins

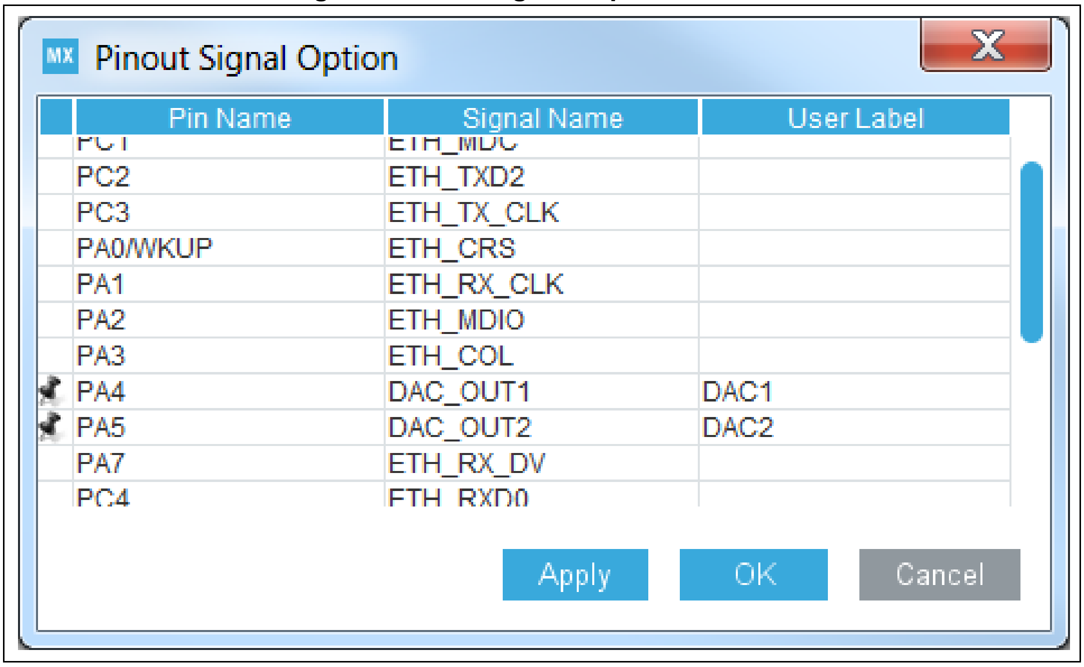
STM32CubeMX comes with a feature allowing the user to selectively lock (or pin) signals to pins. This prevents STM32CubeMX from automatically moving pinned signals to other pins when resolving conflicts. Labels, that are used for code generation, can also be assigned to the signals (see *Section 6.1* for details).

There are several ways to pin, unpin and label the signals:

1. From the **Pinout** view, right-click a pin with a signal assignment. This opens a contextual menu:
   a) For unpinned signals, select **Signal Pinning** to pin the signal. A pin icon is then displayed on the relevant pin. The signal can no longer be moved automatically (for example when resolving pin assignment conflicts).
   b) For pinned signals, select **Signal Unpinning** to unpin the signal. The pin icon is removed. From now on, to resolve a conflict (such as peripheral mode conflict), this signal can be moved to another pin, provided the Keep user placement option is unchecked.
   c) Select **Enter User Label** to specify a user defined label for this signal. The new label replaces the default signal name in the **Pinout** view.

2. From the **Pinout** menu, select **Pins/Signals Options**
   The Pins/Signals Options window (see *Figure 67*) lists all configured pins.

Figure 67. Pins/Signals Options window



a) Click the first column to individually pin/unpin signals.
b) Select multiple rows and right-click to open the contextual menu and select **Signal(s) Pinning** or **Unpinning**.
c) Select the User Label field to edit the field and enter a user-defined label.
d) Order list alphabetically by Pin or Signal name by clicking the column header. Click once more to go back to default i.e. to list ordered according to pin placement on MCU.

Note: Even if a signal is pinned, it is still possible however to manually change the pin signal assignment from the ***Pinout*** *view: click the pin to display other possible signals for this pin and select the relevant one.*

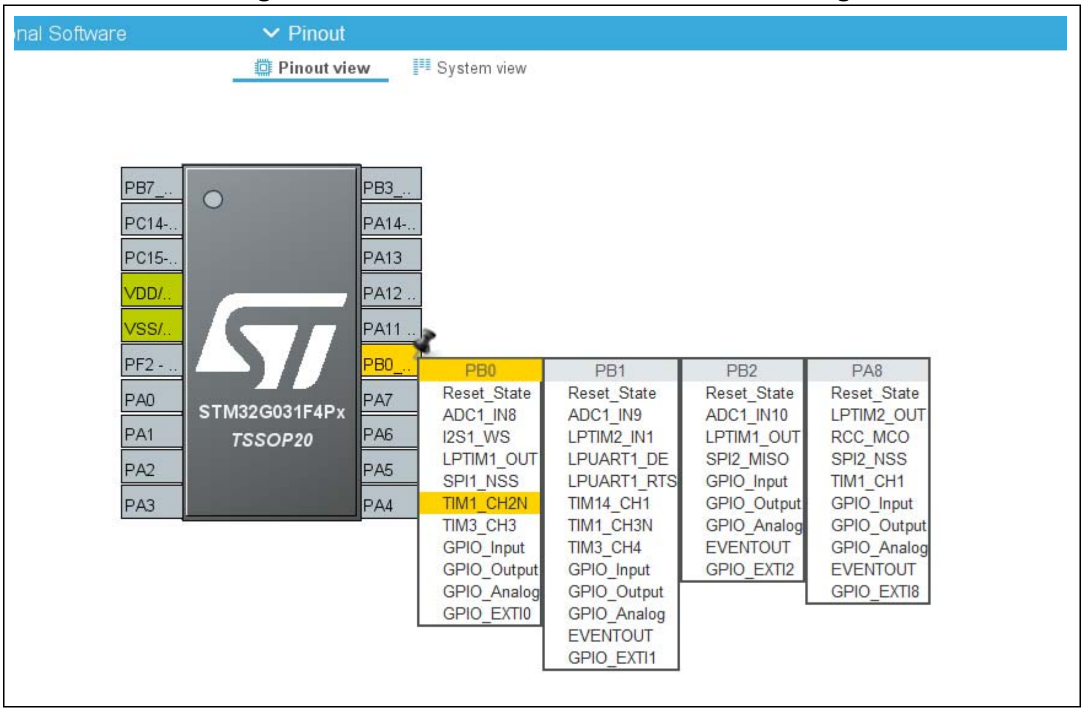
4.5.8 Pinout for multi-bonding packages

Multi-bonding has been introduced for packages with low pin counts (less than 20 pins) such as SO8N, TSSOP20 and WLCSP18 packages. it consists of having several MCU pads share a same pin on the package.

Multi-bonding has been introduced on the STM32G0 series for the STM32G031/G041 MCUs.

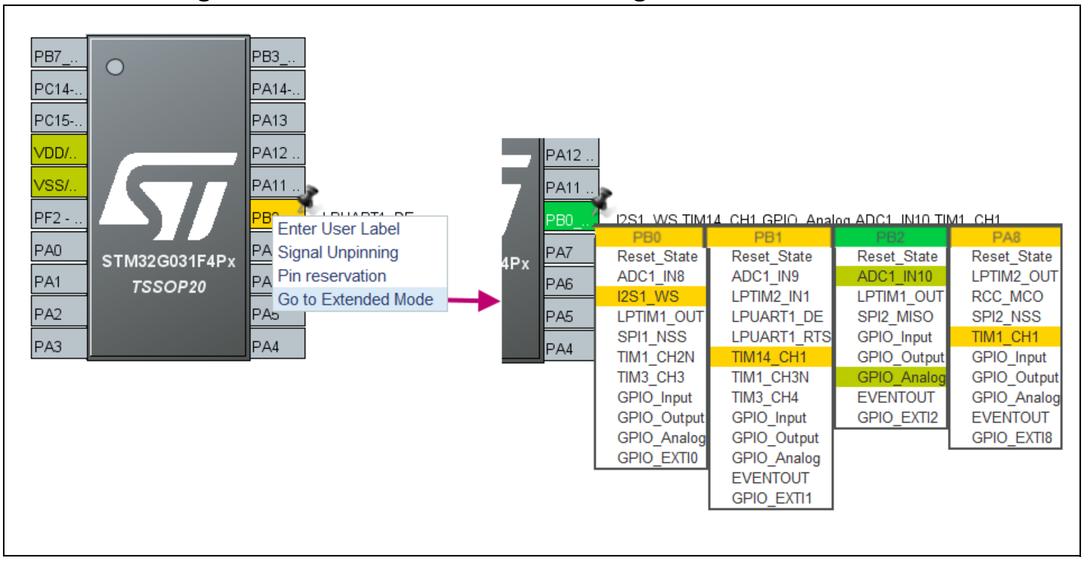
STM32CubeMX pinout view allows to displays all signals arriving on the pin and allows to select only one per pin, except for analog signals that can be combined with other analog GPIOs.

Figure 68. Pinout view: MCUs with multi-bonding



STM32CUbeMX offers also an extended mode selected by right-clicking the pin: it allows to select more than one signal per pin. This mode is meant for test purposes such as loopback tests. It is to be used with caution as it can lead to electrical conflicts or increased power consumption that can damage the device.

Figure 69. Pinout view: multi-bonding with extended mode



4.5.9 System view

Select System view to show all the software configurable components: GPIOs, peripherals and middleware. Clickable buttons allow the user to open the mode and configuration options of the component. The button icon reflects the component configuration status (see *Table 7* for configuration states and Figure System view).

When the user changes the component configuration from the Configuration panel, the system view is automatically refreshed with the new configuration state.

If the user disables the component from the Mode panel, the system view is automatically refreshed and there is no longer a button showing for that component.

Figure 70. System view

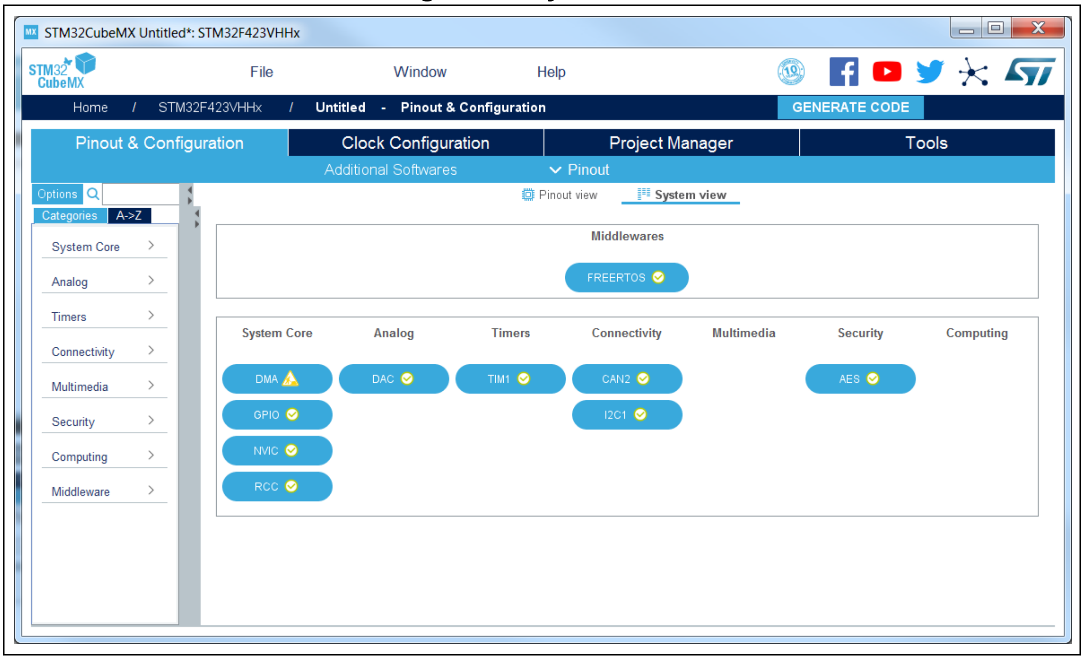
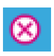
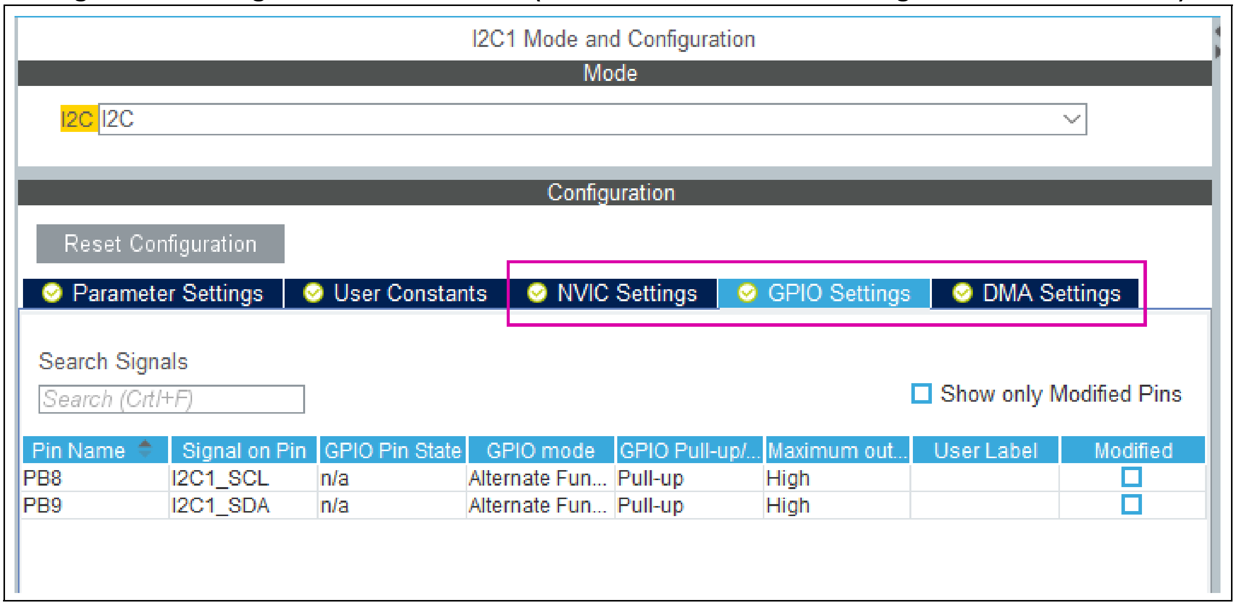


Table 7. Configuration states

Icon	Description
	Configuration is complete and correct.
	Configuration is correct but some parts remain to be configured (optional).
	Configuration is invalid and must be fixed for the generated C project to be functional.

GPIO, DMA and NVIC settings can be accessed either via a dedicated button (like other peripherals, or via a tab in the Configuration panel (see *Figure 71*).

Figure 71. Configuration window tabs (GPIO, DMA, and NVIC settings for STM32F4 series)



4.5.10 Component configuration panel

This panel appears when clicking on a component name in the left panel. It allows the user to configure the functional parameters required to initialize the peripheral or the middleware in the selected operating mode (see *Figure 72*). STM32CubeMX uses these settings to generate the corresponding initialization C code.

The configuration window includes several tabs:

- **Parameter settings** to configure library dedicated parameters for the selected peripheral or middleware,
- **NVIC, GPIO and DMA settings** to set the parameters for the selected peripheral (see *Section 4.5.14*, *Section 4.5.12* and *Section 4.5.13*).
- **User constants** to create one or several user defined constants, common to the whole project (see *Section 4.5.11*).

Invalid settings are detected and are:

- reset to minimum / maximum valid value if user choice is, respectively, smaller / larger than minimum / maximum threshold
- reset to the previous valid value if the previous one is neither a maximum nor a minimum threshold value
- highlighted in fuchsia.

Figure 72. Peripheral mode and Configuration view

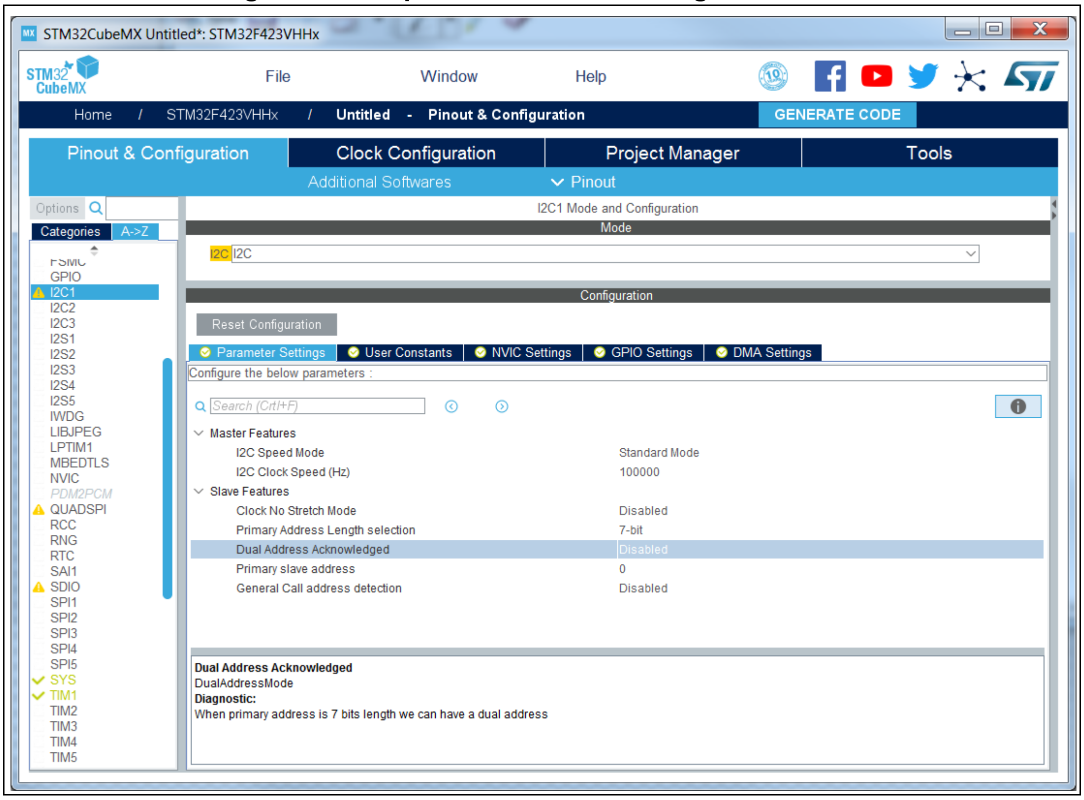


Table 8 describes peripheral and middleware configuration buttons and messages.

Table 8. Peripheral and Middleware configuration window buttons and tooltips

Buttons and messages	Action
(i)	Shows / hides the description panel.
Tooltip Enabled Disabled Enabled Disabled I2C_DUALADDRESS_ENABLE	Guides the user through the settings of parameters with valid min-max range. To display it, move the mouse over a parameter value from a list of possible values.
I2C Clock Speed (Hz) 100000 ✔ Decimal Hexadecimal No check	Clicking on the gear icon allows to select whether to display hexadecimal or decimal values, or any value unchecked (No check option).
Search (Crtl+F)	Search
Reset Configuration	Resets the component back to its default configuration (initial settings from STM32CubeMX).

No check option

By default, STM32CubeMX checks that the parameter values entered by the user are valid. This check can be bypassed by selecting the option No Check for a given parameter. This allows entering you any value (such as a constant) that might not be known by STM32CubeMX configuration.

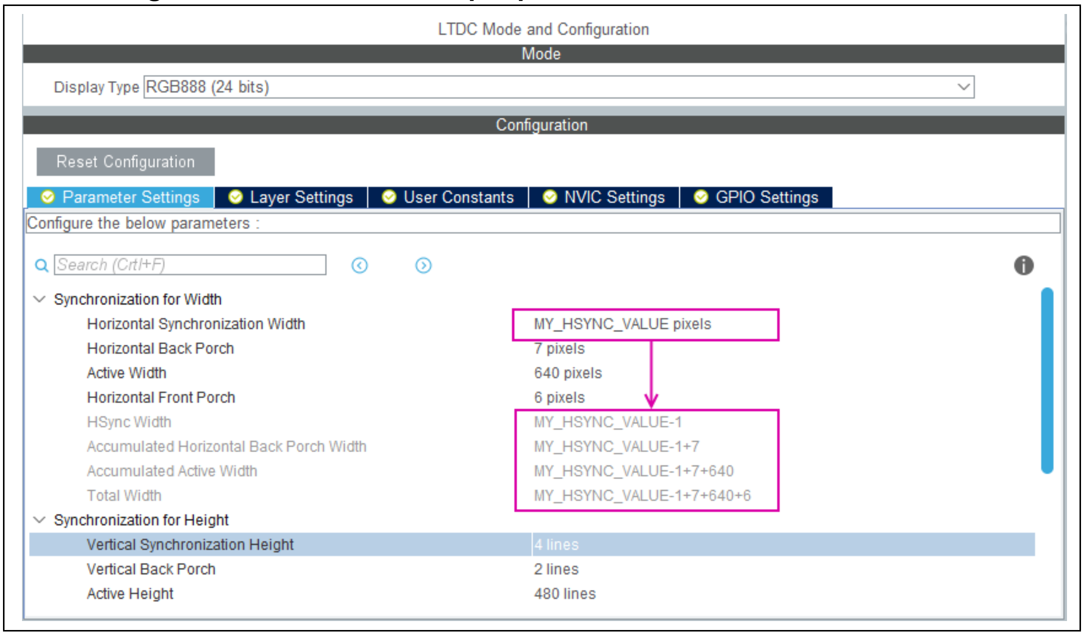
The validity check can be bypassed only on parameters whose values are of integer type (either hexadecimal or decimal). It cannot be bypassed on parameters coming from a predefined list of possible values or on those which are of non-integer or text type.

To go back to the default mode (decimal or hexadecimal values with validity check enabled), enter a decimal or hexadecimal value and check the relevant option (hexadecimal or decimal check).

Caution: When a parameter depends upon another parameter that is set to No Check:

- Case of a parameter depending on another parameter for the evaluation of its minimum or maximum possible value: If the other parameter is set to No Check, the minimum or maximum value is no longer evaluated and checked.
- Case of a parameter depending on another parameter for the evaluation of its current value: If the other parameter is set to No Check, the value is no longer automatically derived. Instead, it is replaced with the formula text showing as variable the string of the parameter set to No check (see *Figure 73*).

Figure 73. Formula when input parameter is set in No Check mode



4.5.11 User Constants configuration window

The **User Constants** tab is available to define user constants (see *Figure 74*). Constants are automatically generated in the STM32CubeMX user project within the main.h file (see *Figure 75*). Once defined, they can be used to configure peripheral and middleware parameters (see *Figure 76*).

Figure 74. User Constants tab

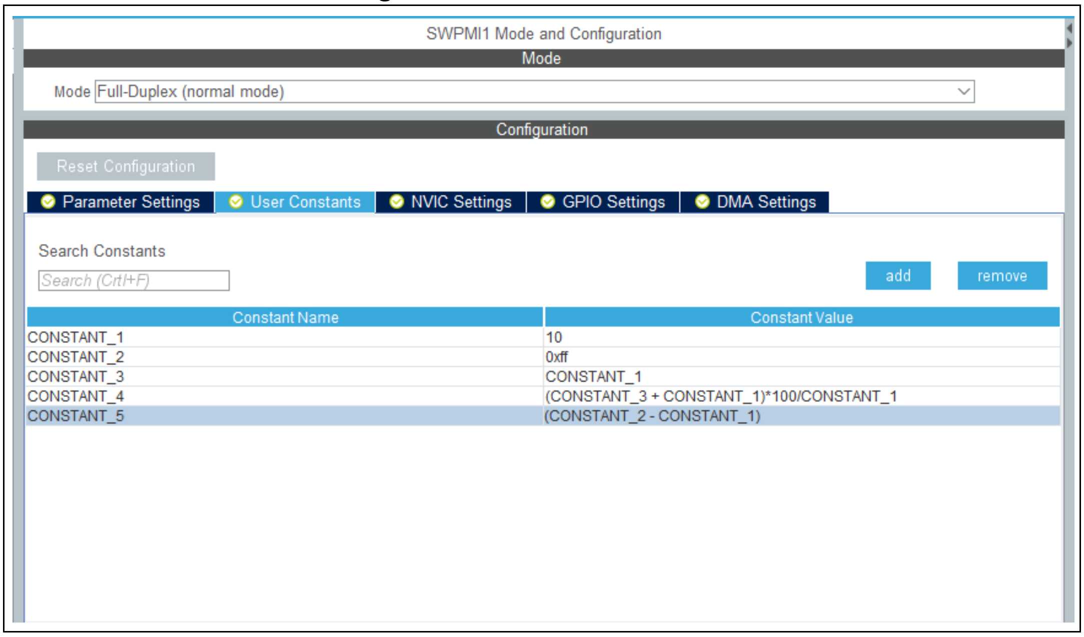
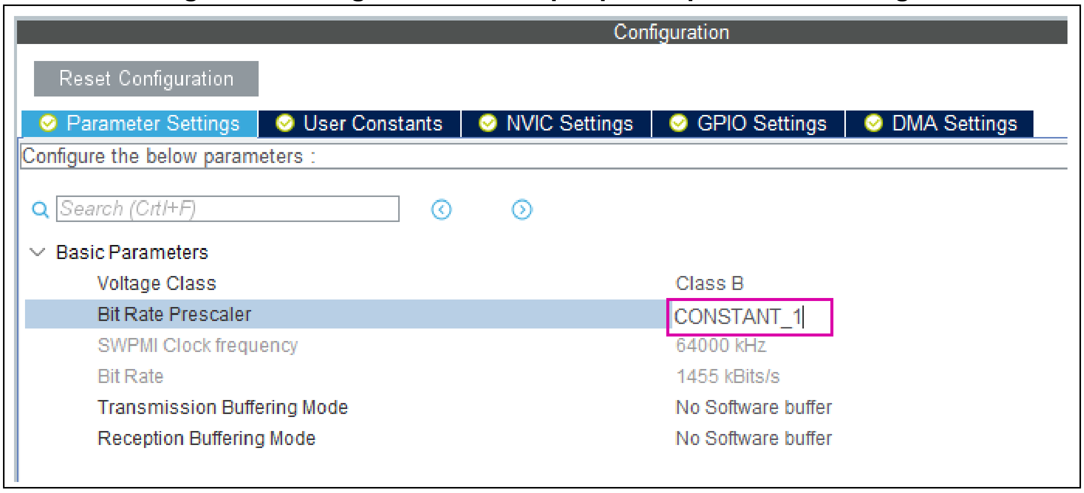


Figure 75. Extract of the generated main.h file

```
/* Includes ------------------------------------------------------------------*/

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private define ------------------------------------------------------------*/
#define CONSTANT_1 10
#define CONSTANT_2 0xff
#define CONSTANT_3 CONSTANT_1
#define CONSTANT_4 (CONSTANT_3+CONSTANT_1)*100/CONSTANT_1
#define CONSTANT_5 (CONSTANT_2 - CONSTANT_1)

/* USER CODE BEGIN Private defines */

/* USER CODE END Private defines */
```

Figure 76. Using constants for peripheral parameter settings



Creating/editing user constants

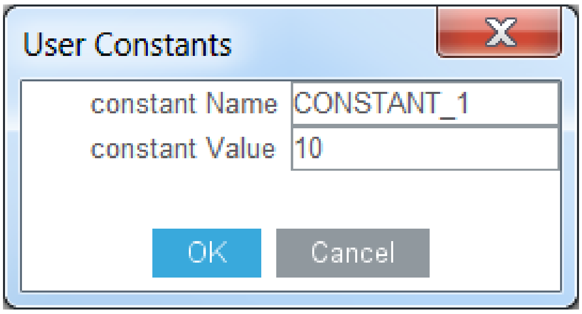
Click the **Add** button to open the **User Constants** tab and create a new user-defined constant (see *Figure 77*).

A constant consists of:

- A name that must comply with the following rules:
  - It must be unique.
  - It must not be a C/C++ keyword.
  - It must not contain a space.
  - It must not start with digits.
- A value, which can be (see *Figure 74* for examples):
  - a simple decimal or hexadecimal value
  - a previously defined constant
  - a formula using arithmetic operators (subtraction, addition, division, multiplication, and remainder) and numeric value or user-defined numeric constants as operands
  - a character string: the string value must be between double quotes (example: “constant_for_usart”).

Once a constant is defined, its name and/or value can be changed: double-click the row that specifies the user constant to modify. This opens the **User Constants** tab for edition. The change of constant name is applied wherever the constant is used. This does not affect the peripheral or middleware configuration state. Changing the constant value impacts the parameters that use it and might result in invalid settings (such as exceeding a maximum threshold). Invalid parameter settings are highlighted in fuchsia.

Figure 77. Specifying user constant value and name



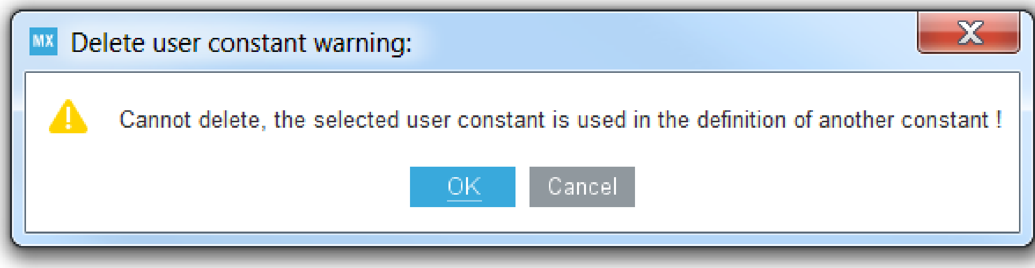
Deleting user constants

Click the **Remove** button to delete an existing user-defined constant.

The user constant is then automatically removed except in the following cases:

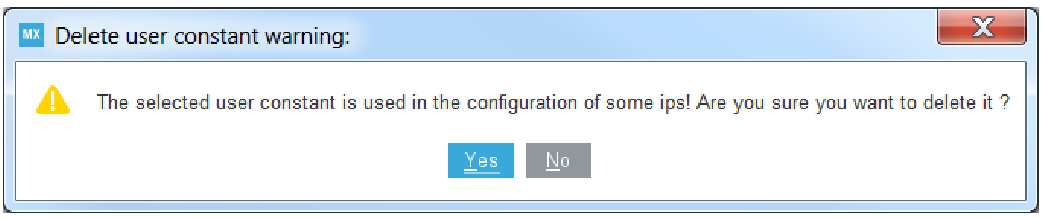
- When the constant is used for the definition of another constant. In this case, a popup window displays an explanatory message (see *Figure 78*).

Figure 78. Deleting an user constant is not allowed when it is already used for another constant definition



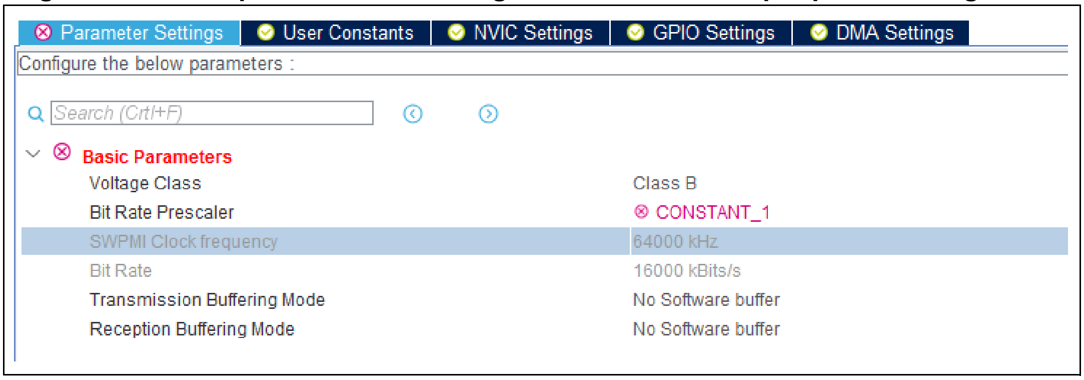
- When the constant is used for the configuration of a peripheral or middleware library parameter. In this case, the user is requested to confirm the deletion since the constant removal results in a invalid peripheral or middleware configuration (see *Figure 79*).

Figure 79. Confirmation request to delete a constant for parameter configuration



Clicking **Yes** leads to an invalid peripheral configuration (see *Figure 80*).

Figure 80. Consequence when deleting a user constant for peripheral configuration



Searching for user constants

The **Search Constants** field makes it possible the search of a constant name or value in the complete list of user constants (see *Figure 81* and *Figure 82*).

Figure 81. Searching for a name in a user constant list

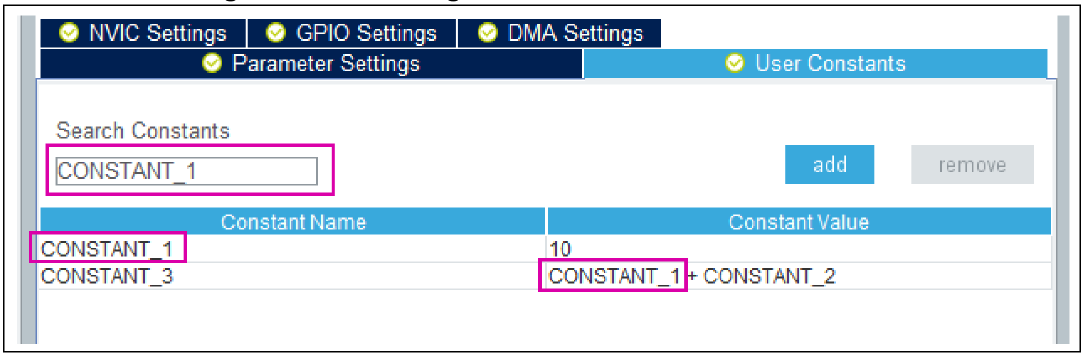
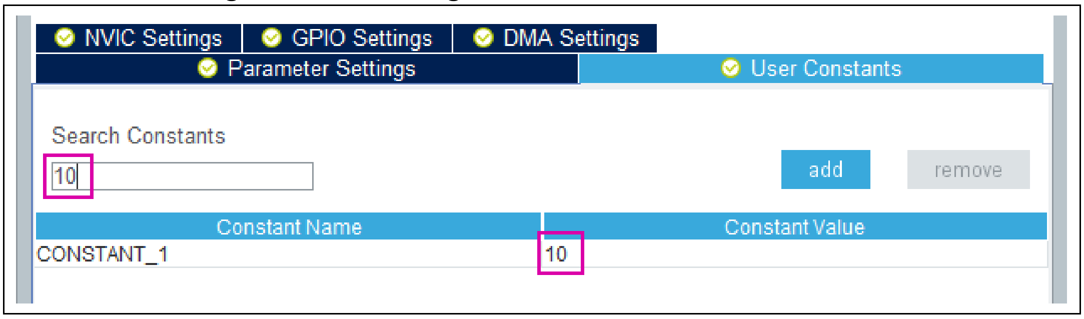


Figure 82. Searching for a value in a user constant list

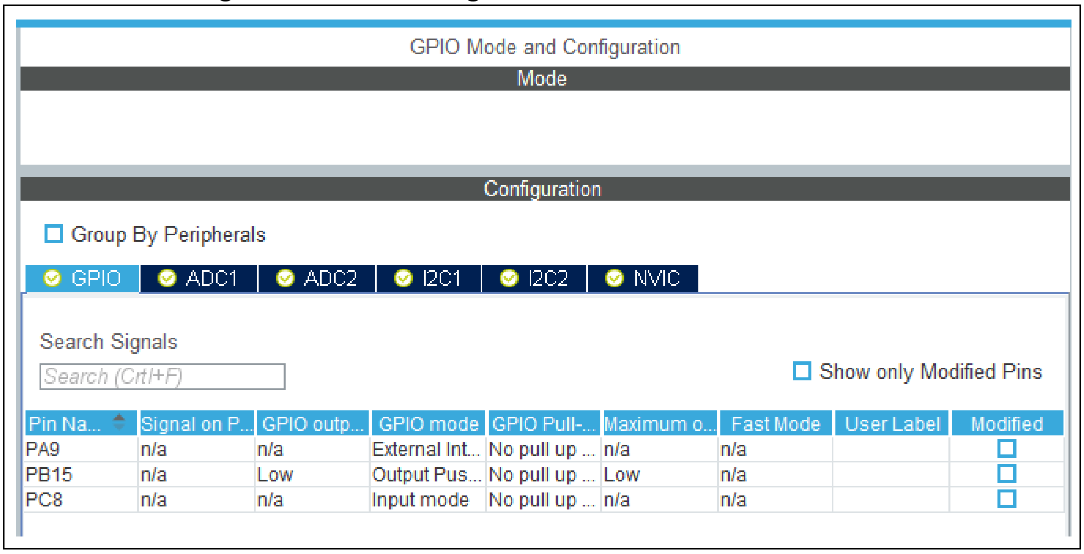


4.5.12 GPIO configuration window

Click **GPIO** in the **System view** panel to open the **GPIO configuration** window to configure the GPIO pin settings (see *Figure 83*). The configuration is populated with default values that might not be adequate for some peripheral configurations. In particular, check if the GPIO speed is sufficient for the peripheral communication speed, and select the internal pull-up whenever needed.

Note: GPIO settings can be accessed for a specific peripheral instance via the dedicated window in the peripheral instance configuration window. In addition, GPIOs can be configured in output mode (default output level). The generated code is updated accordingly.

Figure 83. GPIO configuration window - GPIO selection



Click on a row or select a set of rows to display the corresponding GPIO parameters:

- **GPIO PIN state**

  Changes the default value of the GPIO output level. It is set to low by default and can be changed to high.
- **GPIO mode** (analog, input, output, alternate function)

  Selecting a peripheral mode in the **Pinout** view automatically configures the pins with the relevant alternate function and GPIO mode.
- **GPIO pull-up/pull-down**

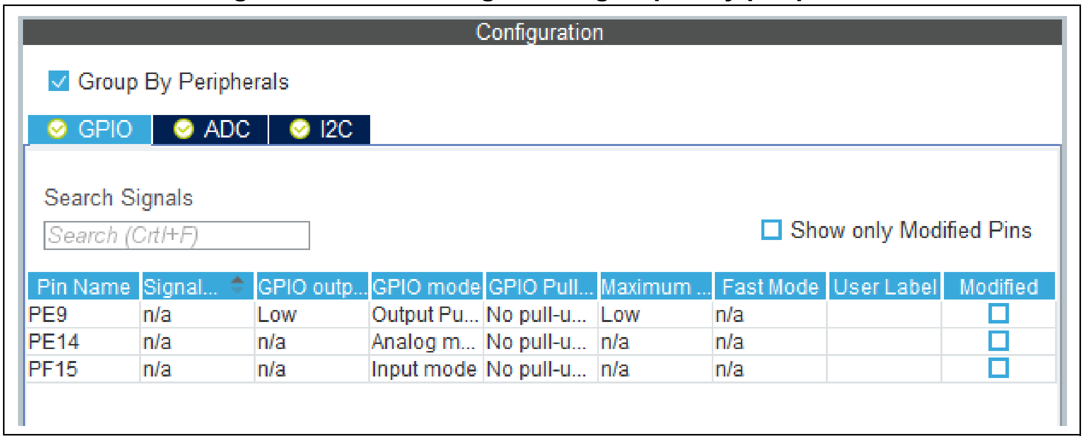
  Set to a default value, can be configured when other choices are possible.
- **GPIO maximum output speed** (for communication peripherals only)

  Set to Low by default for power consumption optimization, can be changed to a higher frequency to fit application requirements.
- **User Label**

  Changes the default name (such as GPIO_input) into a user defined name. The **Pinout** view is updated accordingly. The GPIO can be found under this new name via the **Find** menu.

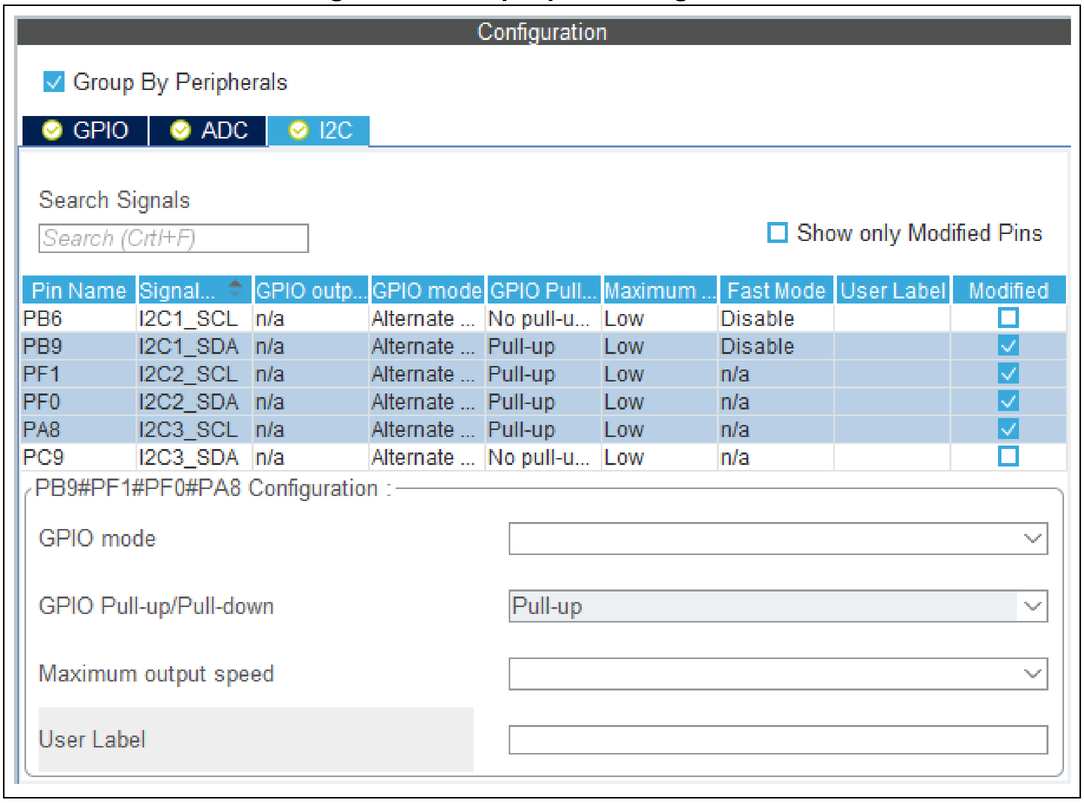
The **Group by Peripherals** checkbox allows the user to group all instances of a peripheral under the same window (see *Figure 84*).

Figure 84. GPIO configuration grouped by peripheral



As shown in *Figure 85*, row multi-selection can be performed to change a set of pins to a given configuration at the same time.

Figure 85. Multiple pins configuration



4.5.13 DMA configuration window

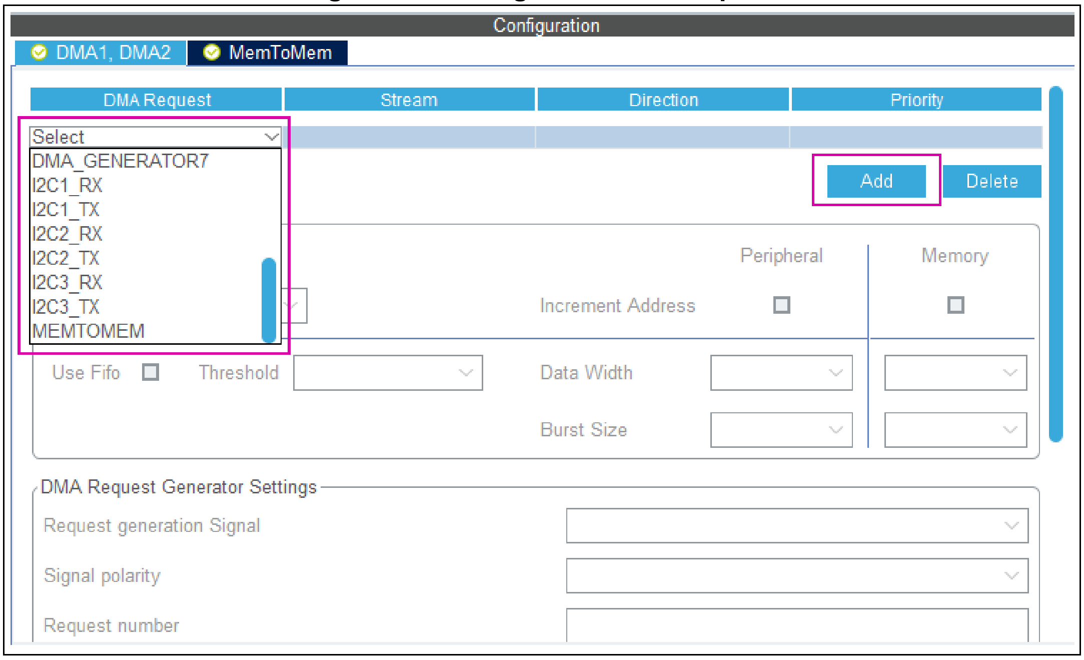
Click **DMA** in the **System** view to open the **DMA configuration** window.

This window is used to configure the generic DMA controllers available on the MCU. The DMA interfaces allow to perform data transfers between memories and peripherals while the CPU is running, and memory to memory transfers (if supported).

*Note: Some peripherals (such as **USB** or **Ethernet**) have their own DMA controller, which is enabled by default or via the Peripheral Configuration window.*

Clicking **Add** in the **DMA configuration** window adds a new line at the end of the DMA configuration window with a combo box proposing to choose between possible **DMA requests** to be mapped to peripherals signals (see *Figure 86*).

Figure 86. Adding a new DMA request

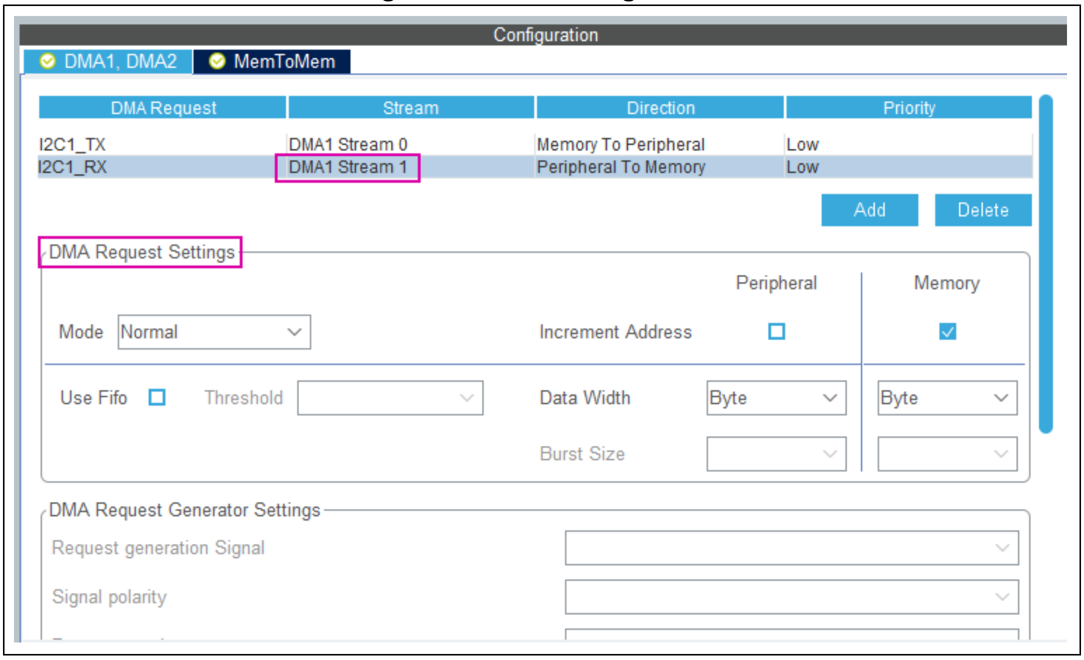


Selecting a DMA request automatically assigns a stream among all the streams available, a direction and a priority. When the DMA channel is configured, it is up to the application code to fully describe the DMA transfer run-time parameters such as the start address.

The DMA request (called channel for STM32F4 MCUs) is used to reserve a stream to transfer data between peripherals and memories (see *Figure 87*). The stream priority is used to decide which stream to select for the next DMA transfer.

DMA controllers support a dual priority system using the software priority first, and in case of equal software priorities, a hardware priority that is given by the stream number.

Figure 87. DMA configuration

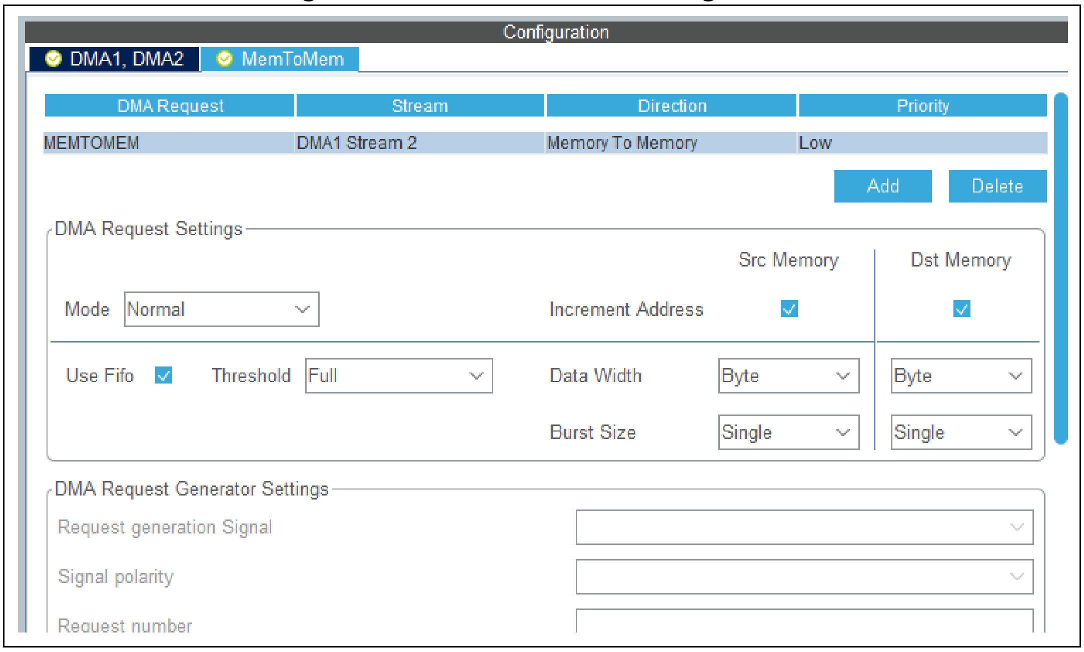


Additional DMA configuration settings can be done through the **DMA configuration** window:

- ***Mode***: regular mode, circular mode, or peripheral flow controller mode (only available for the SDIO peripheral).
- ***Increment Add:*** the type of peripheral address and memory address increment (fixed or postincremented, in which case the address is incremented after each transfer). Click the checkbox to enable the post-incremented mode.
- ***Peripheral data width***: 8, 16, or 32 bits
- Switching from the default direct mode to the *FIFO mode* with programmable *threshold*:
  a) Click the **Use FIFO** checkbox.
  b) Configure the ***peripheral and memory data width*** (8, 16, or 32 bits).
  c) Select between ***single transfer*** *and* ***burst transfer***. If you select burst transfer, choose a burst size (1, 4, 8, or 16).

In case of memory-to-memory transfer (MemToMem), the DMA configuration applies to a source memory and to a destination memory.

Figure 88. DMA MemToMem configuration



4.5.14 NVIC configuration window

Click **NVIC** in the **System** view to open the Nested Vector interrupt controller configuration window (see *Figure 89*).

Interrupt unmasking and interrupt handlers are managed within two tabs:

- **NVIC**, to enable peripheral interrupts in the NVIC controller and to set their priorities
- **Code generation**, to select options for interrupt related code generation

Enabling interruptions using the NVIC tab view

The **NVIC** view (see *Figure 89*) does not show all possible interrupts, but only the ones available for the peripherals selected in the **Pinout & Configuration** panels. System interrupts are displayed but can never be disabled.

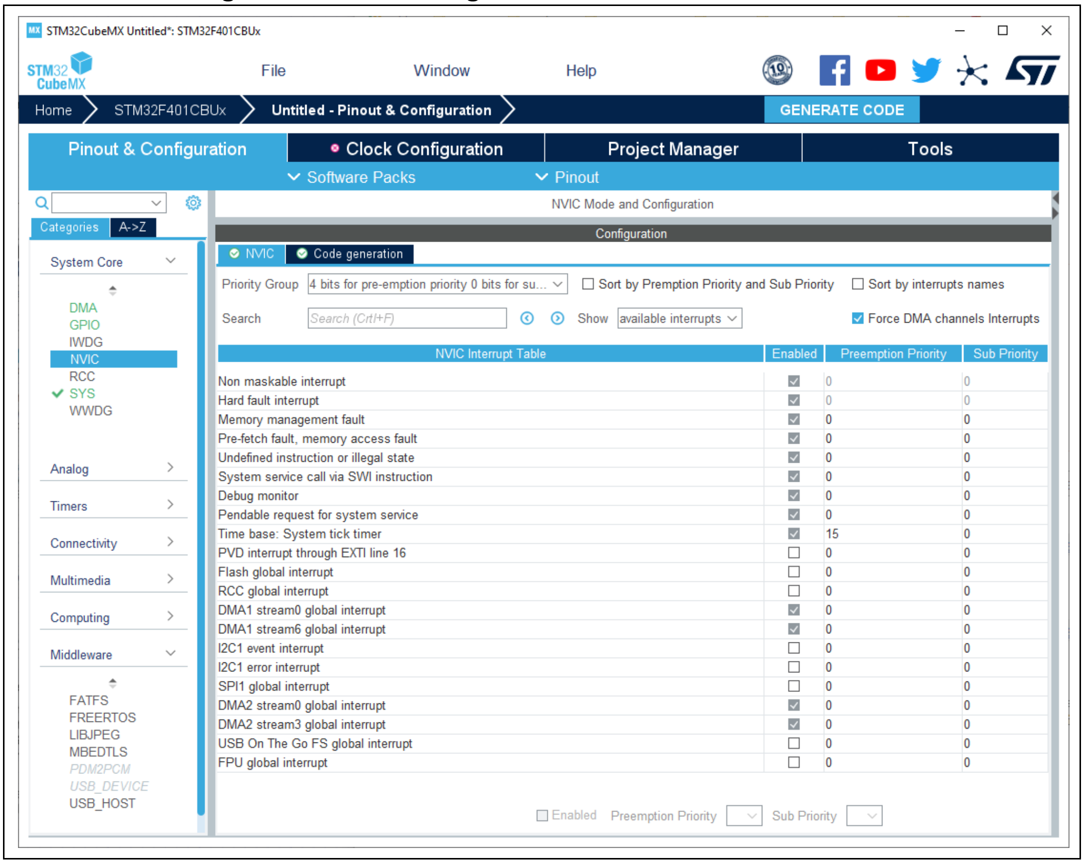
Check/uncheck the **Show only enabled interrupts** box to filter or not enabled interrupts.

When DMA channels are configured in the project, check/uncheck “Force DMA channels interrupts” to automatically enable/disable DMA channels interrupts in the generated code.

Use the **search field** to filter out the interrupt vector table according to a string value. As an example, after enabling UART peripherals from the **Pinout** panel, type UART in the NVIC search field and click the green arrow close to it: all UART interrupts are displayed.

Enabling a **peripheral interrupt** generates NVIC function calls **HAL_NVIC_SetPriority** and **HAL_NVIC_EnableIRQ** for this peripheral.

Figure 89. NVIC configuration tab - FreeRTOS disabled

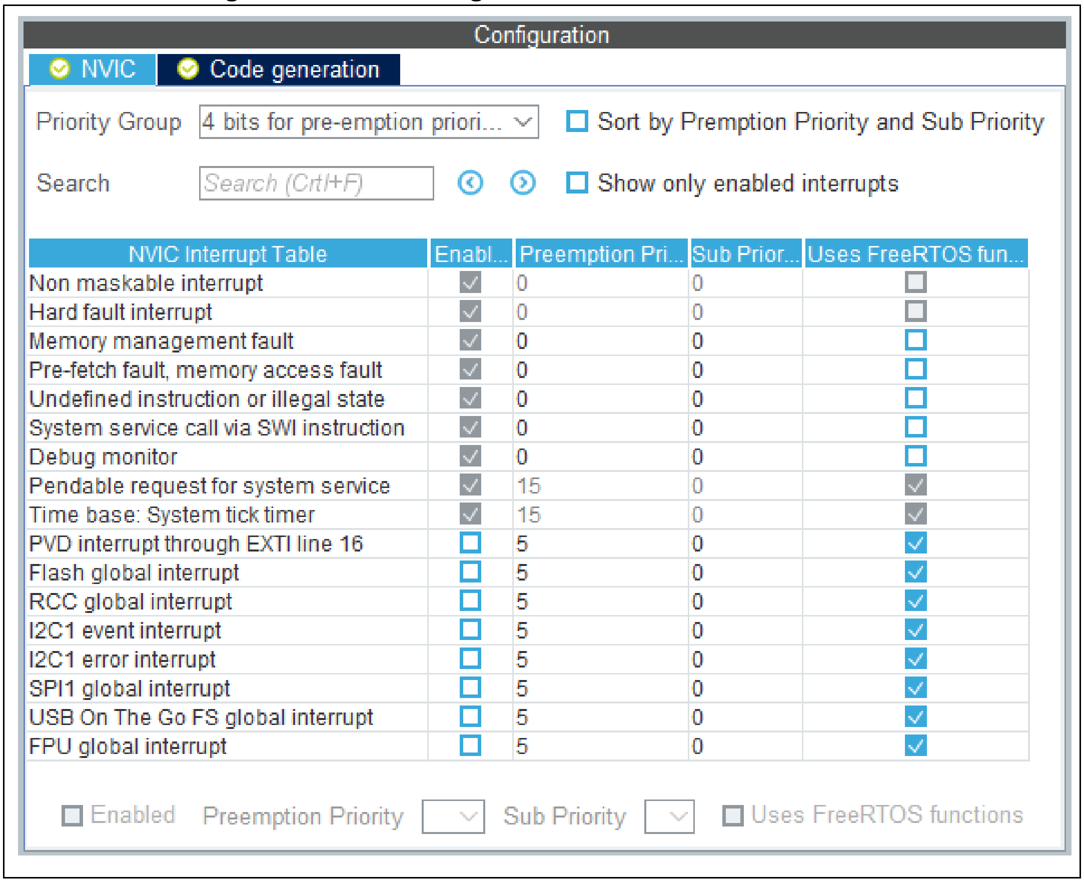


When FreeRTOS is enabled, an additional column is shown (see *Figure 90*).

In this case, all the interrupt service routines (ISRs) that are calling the interrupt safe FreeRTOS APIs must have a priority lower than the priority defined in the LIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY parameter (the highest the value, the lowest the priority). The check in the corresponding checkbox guarantees that the restriction is applied.

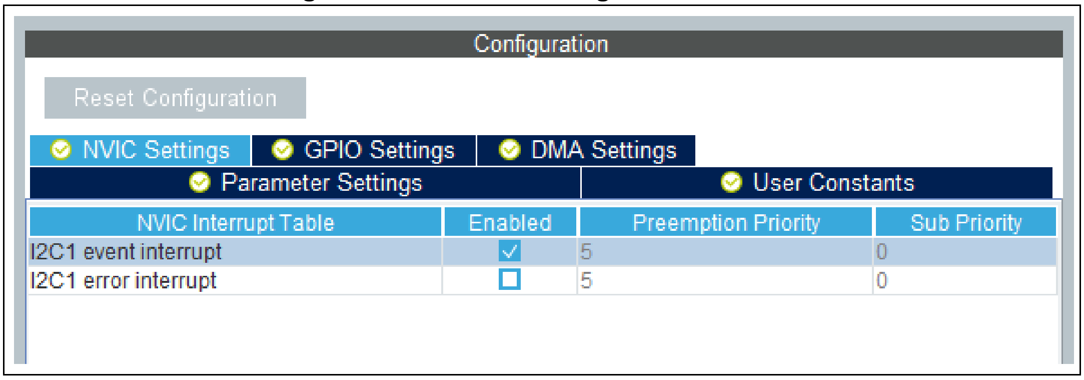
If an ISR does not use such functions, the checkbox can be unchecked and any priority level can be set. It is possible to check/uncheck multiple rows (see rows highlighted in blue in *Figure 90*).

Figure 90. NVIC configuration tab - FreeRTOS enabled



Peripheral dedicated interrupts can also be accessed through the NVIC window in the configuration window (see *Figure 91*).

Figure 91. I2C NVIC configuration window



STM32CubeMX NVIC configuration consists in selecting a priority group, enabling/disabling interrupts and configuring interrupts priority levels (preemption and sub-priority levels):

1. Select a **priority group**

   Several bits allow to define NVIC priority levels, they are divided in two groups, preemption priority and sub-priority. For example, in the case of STM32F4 MCUs, the NVIC priority group 0 corresponds to 0-bit preemption and 4-bit sub-priority.

2. In the interrupt table, click one or more rows to select one or more interrupt vectors. Use the widgets below the interrupt table to configure the vectors one by one or several at a time:
   - **Enable checkbox**: check/uncheck to enable/disable the interrupt.
   - **Preemption priority**: select a priority level. The preemption priority defines the ability of one interrupt to interrupt another.
   - **Sub-priority**: select a priority level. Defines the interrupt priority level.

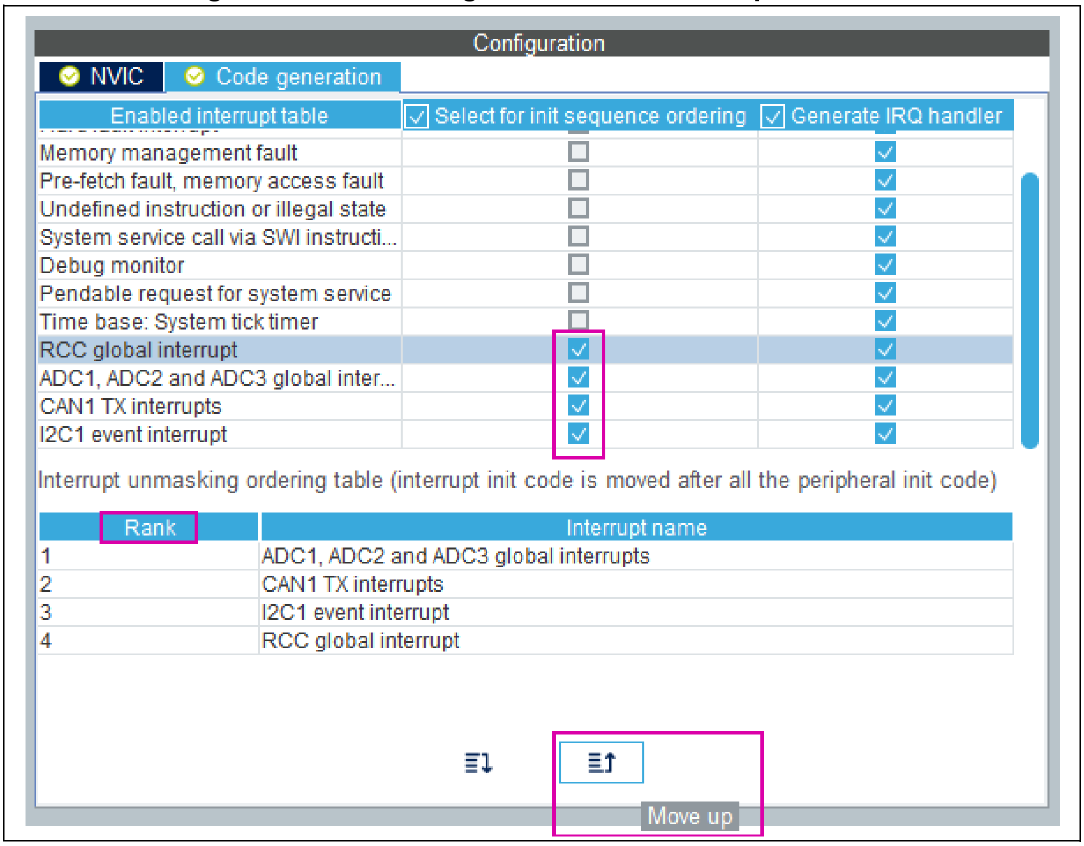
Code generation options for interrupt handling

The **Code Generation** view allows customizing the code generated for interrupt initialization and interrupt handlers:

- **Selection/Deselection of all interrupts for sequence ordering and IRQ handler code generation**

  Use the checkboxes in front of the column names to configure all interrupts at a time (see *Figure 92*). Note that system interrupts are not eligible for init sequence reordering as the software solution does not control it.

Figure 92. NVIC Code generation – All interrupts enabled



- **Default initialization sequence of interrupts**

  By default, the interrupts are enabled as part of the peripheral MSP initialization function, after the configuration of the GPIOs and the enabling of the peripheral clock.

  This is shown in the CAN example below, where *HAL_NVIC_SetPriority* and *HAL_NVIC_EnableIRQ* functions are called within stm32xxx_hal_msp.c file inside the peripheral *msp_init* function.

  Interrupt enabling code is shown in bold:

```
void HAL_CAN_MspInit(CAN_HandleTypeDef* hcan)
{
GPIO_InitTypeDef GPIO_InitStruct;
if(hcan->Instance==CAN1)
{
 /* Peripheral clock enable */
 __CAN1_CLK_ENABLE();
 /**CAN1 GPIO Configuration
 PD0     ------> CAN1_RX
 PD1     ------> CAN1_TX
 */
 GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1;
 GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
 GPIO_InitStruct.Alternate = GPIO_AF9_CAN1;
 HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

/* Peripheral interrupt init */
 HAL_NVIC_SetPriority(CAN1_TX_IRQn, 2, 2);
 HAL_NVIC_EnableIRQ(CAN1_TX_IRQn);
 }
}
```

  For **EXTI GPIOs** only, interrupts are enabled within the *MX_GPIO_Init* function:

```
/*Configure GPIO pin : MEMS_INT2_Pin */
GPIO_InitStruct.Pin = MEMS_INT2_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_EVT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(MEMS_INT2_GPIO_Port, &GPIO_InitStruct);

/* EXTI interrupt init*/
HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
```

  For some peripherals, the application still needs to call another function to actually activate the interruptions. Taking the timer peripheral as an example, the *HAL_TIM_IC_Start_IT* function needs to be called to start the Timer input capture (IC) measurement in interrupt mode.

- **Configuration of interrupts initialization sequence**

  Checking **Select for Init sequence ordering** for a set of peripherals moves the HAL_NVIC function calls for each peripheral to a same dedicated function, named **MX_NVIC_Init**, defined in the main.c. Moreover, the HAL_NVIC functions for each peripheral are called in the order specified in the **Code generation** view bottom part (see *Figure 93*).

  As an example, the configuration shown in *Figure 93* generates the following code:

```
/** NVIC Configuration
*/
void MX_NVIC_Init(void)
{
 /* CAN1_TX_IRQn interrupt configuration */
 HAL_NVIC_SetPriority(CAN1_TX_IRQn, 2, 2);
 HAL_NVIC_EnableIRQ(CAN1_TX_IRQn);
 /* PVD_IRQn interrupt configuration */
 HAL_NVIC_SetPriority(PVD_IRQn, 0, 0);
 HAL_NVIC_EnableIRQ(PVD_IRQn);
 /* FLASH_IRQn interrupt configuration */
 HAL_NVIC_SetPriority(FLASH_IRQn, 0, 0);
 HAL_NVIC_EnableIRQ(CAN1_IRQn);
 /* RCC_IRQn interrupt configuration */
 HAL_NVIC_SetPriority(RCC_IRQn, 0, 0);
 HAL_NVIC_EnableIRQ(CAN1_IRQn);
 /* ADC_IRQn interrupt configuration */
 HAL_NVIC_SetPriority(ADC_IRQn, 0, 0);
 HAL_NVIC_EnableIRQ(ADC_IRQn);
}
```

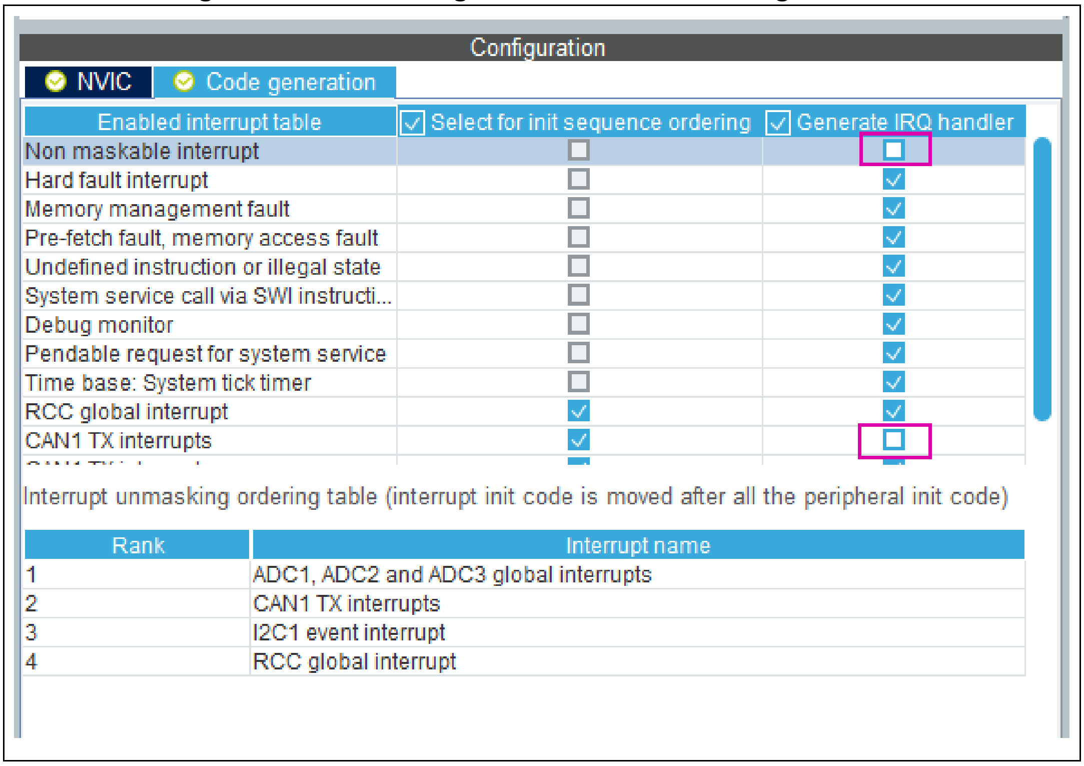
- **Interrupts handler code generation**

  By default, STM32CubeMX generates interrupt handlers within the stm32xxx_it.c file. As an example:

```
void NMI_Handler(void)
{
 HAL_RCC_NMI_IRQHandler();
}
void CAN1_TX_IRQHandler(void)
{
 HAL_CAN_IRQHandler(&hcan1);
}
```

  The column **Generate IRQ Handler** allows the user to control whether the interrupt handler function call can be generated or not. Deselecting CAN1_TX and NMI interrupts from the **Generate IRQ Handler** column as shown in *Figure 93* removes the code mentioned earlier from the stm32xxx_it.c file.

Figure 93. NVIC Code generation - IRQ Handler generation

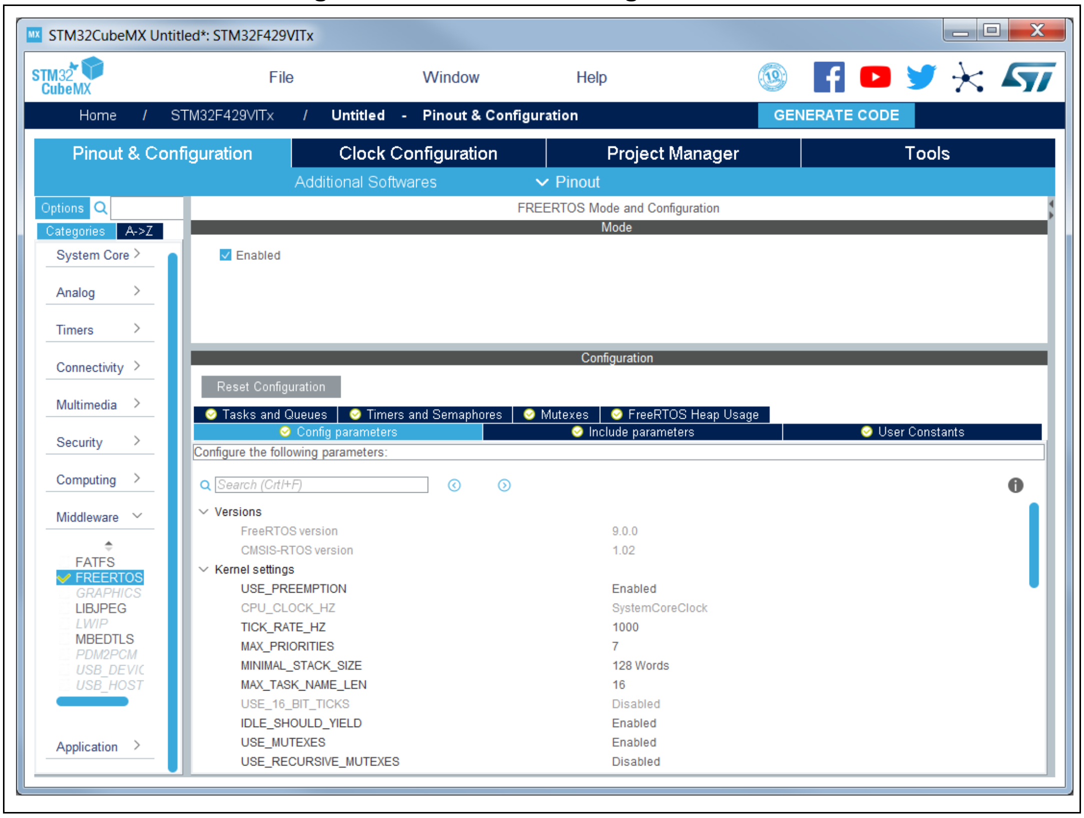


4.5.15 FreeRTOS configuration panel

Through STM32CubeMX FreeRTOS configuration window, the user can configure all the resources required for a real-time OS application, and reserve the corresponding heap. FreeRTOS elements are def/ined and created in the generated code using CMSIS-RTOS API functions. Follow the sequence below:

1. In the **Pinout & Configuration** tab, click FreeRTOS to reveal the Mode and Configuration panels (see *Figure 94*).
2. Enable freeRTOS in the Mode panel.
3. Go to the configuration panel to proceed with configuring FreeRTOS native parameters and objects, such as tasks, timers, queues, and semaphores. In the Config tab, configure Kernel and Software settings. In the Include parameters tab, select the API functions required by the application and this way, optimize the code size. Both Config and Include parameters are part of the FreeRTOSConfig.h file.

Figure 94. FreeRTOS configuration view



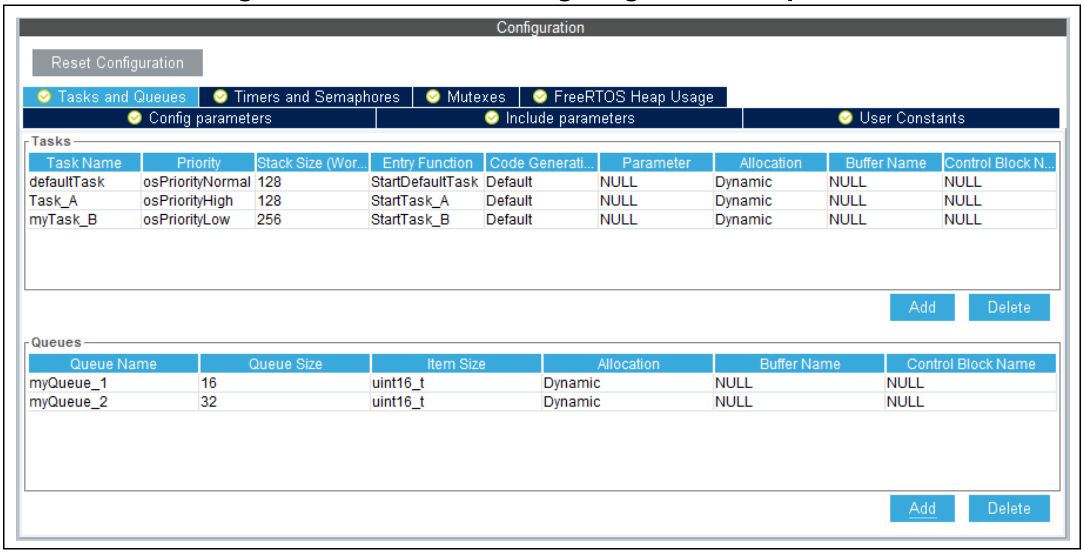
Tasks and Queues tab

As any RTOS, FreeRTOS allows structuring a real-time application into a set of independent tasks, with only one task being executed at a given time. Queues are meant for inter-task communications: they allow to exchange messages between tasks or between interrupts and tasks.

The **FreeRTOS Tasks and Queues** tab enables the creation and configuration of such tasks and queues (see *Figure 95*).

The corresponding initialization code is generated within main.c or freeRTOS.c if the option “generate code as pair of .c/.h files per peripherals and middleware” is set in the **Project Settings** menu, or within main.c by default, or within freeRTOS.c if the option “generate code as pair of .c/.h files per peripherals and middleware” is set in the **Project Manager** menu.

Figure 95. FreeRTOS: configuring tasks and queues



- Tasks

  Under the **Tasks** section, click the **Add** button to open the **New Task** window where task **name**, **priority**, **stack size** and **entry function** can be configured (see *Figure 96*). These settings can be updated at any time: double-clicking a task row opens again the new task window for editing.

  The entry function can be generated as weak or external:

  – When the task is generated as **weak**, the user can propose a definition different from the one generated by default.

  – When the task is **extern**, it is up to the user to provide its function definition.

  By default, the function definition is generated including user sections to allow customization.

- Queues

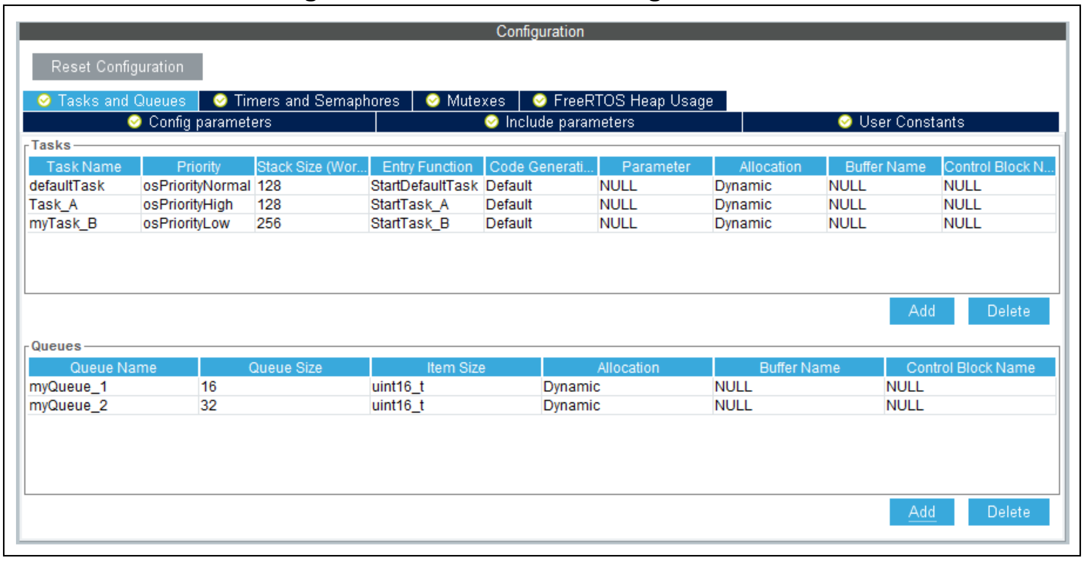
  Under the **Queues** section, click the **Add** button to open the **New Queue** window where the queue **name**, **size** and **item size** can be configured (see *Figure 96*). The queue size corresponds to the maximum number of items that the queue can hold at a time, while the item size is the size of each data item stored in the queue. The item size can be expressed either in number of bytes or as a data type:

- 1 byte for uint8_t, int8_t, char and portCHAR types
- 2 bytes for uint16_t, int16_t, short and portSHORT types
- 4 bytes for uint32_t, int32_t, int, long and float
- 8 bytes for uint64_t, int64_t and double

  By default, the FreeRTOS heap usage calculator uses four bytes when the item size cannot be automatically derived from user input.

  These settings can be updated at any time: double-clicking a queue row opens again the new queue window for editing.

Figure 96. FreeRTOS: creating a new task



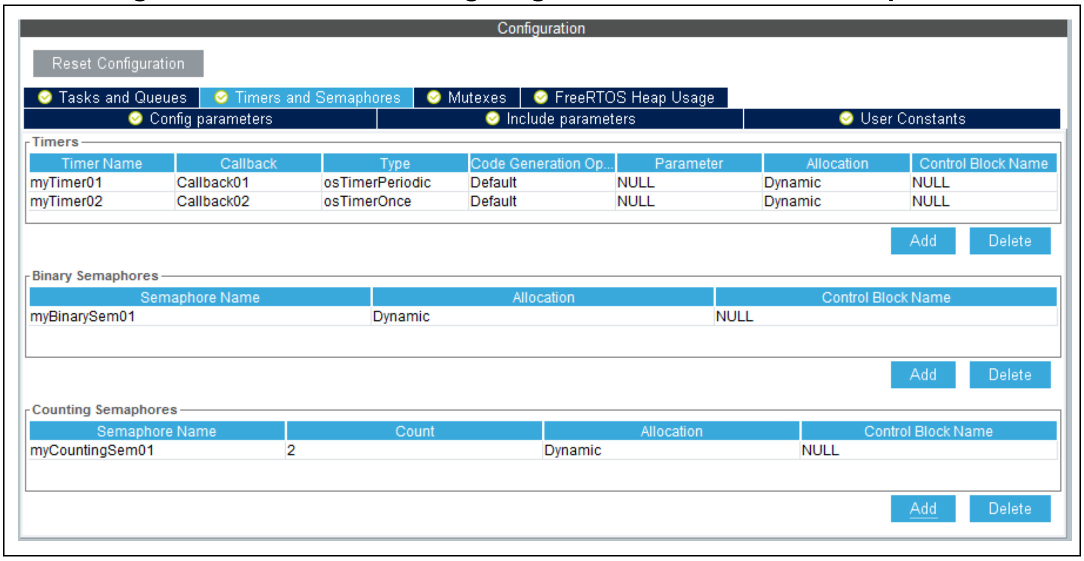
The following code snippet shows the generated code corresponding to *Figure 95*.

```
/* Create the thread(s) */
  /* definition and creation of defaultTask */
  osThreadDef(defaultTask, StartDefaultTask, osPriorityNormal, 0, 128);
  defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);

  /* definition and creation of Task_A */
  osThreadDef(Task_A, StartTask_A, osPriorityHigh, 0, 128);
  Task_AHandle = osThreadCreate(osThread(Task_A), NULL);

  /* definition and creation of Task_B */
  osThreadDef(Task_B, StartTask_B, osPriorityLow, 0, 256);
  Task_BHandle = osThreadCreate(osThread(Task_B), NULL);

  /* Create the queue(s) */
  /* definition and creation of myQueue_1 */
  osMessageQDef(myQueue_1, 16, 4);
  myQueue_1Handle = osMessageCreate(osMessageQ(myQueue_1), NULL);

  /* definition and creation of myQueue_2 */
  osMessageQDef(myQueue_2, 32, 2);
  myQueue_2Handle = osMessageCreate(osMessageQ(myQueue_2), NULL);
```

Timers, Mutexes and Semaphores

FreeRTOS timers, mutexes and semaphores can be created via the FreeRTOS **Timers and Semaphores** tab. They first need to be enabled from the Config tab (see *Figure 97*).

Figure 97. FreeRTOS - Configuring timers, mutexes and semaphores



Under each object dedicated section, clicking the **Add** button to open the corresponding **New <object>** window, where the object specific parameters can be specified. Object settings can be modified at any time: double- clicking the relevant row opens again the **New <object>** window for edition.

Note: Expand the window if the newly created objects are not visible.

- Timers

  Prior to creating timers, their usage (USE_TIMERS definition) must be enabled in the **software timer definitions section** of the **Configuration parameters** tab. In the same section, timer task priority, queue length and stack depth can be also configured.

  The timer can be created to be one-shot (run once) or auto-reload (periodic). The timer name and the corresponding callback function name must be specified. It is up to the user to fill the callback function code and to specify the timer period (time between the timer being started and its callback function being executed) when calling the CMSIS-RTOS osTimerStart function.

- Mutexes / Semaphores

  Prior to creating mutexes, recursive mutexes and counting semaphores, their usage (USE_ MUTEXES, USE_RECURSIVE_MUTEXES, USE_COUNTING_SEMAPHORES definitions) must be enabled within the **Kernel settings** section of the **Configuration parameters** tab.

  The following code snippet shows the generated code corresponding to *Figure 97*.

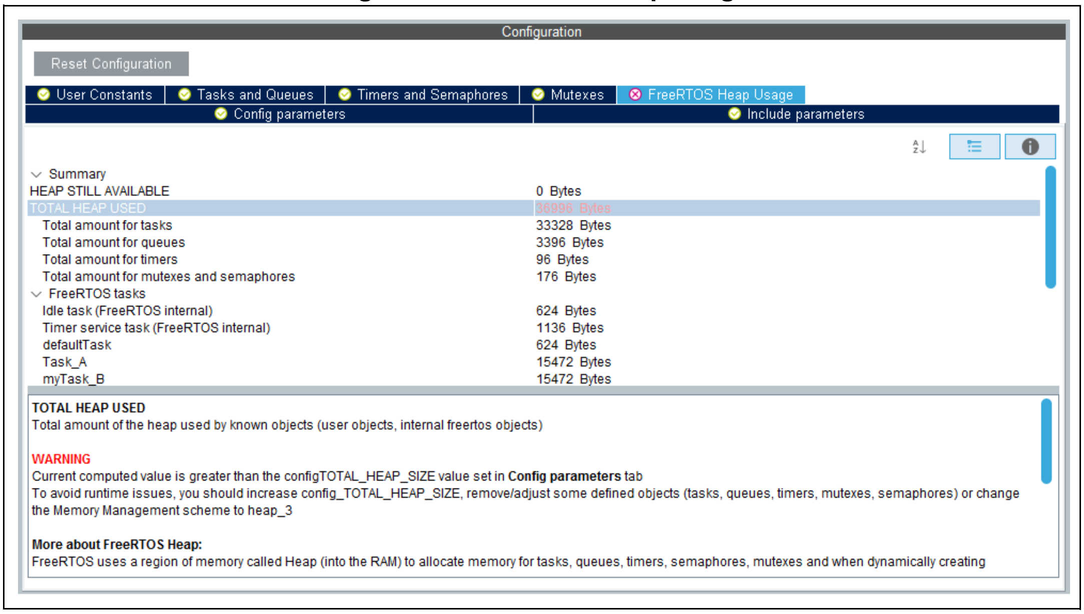
```
/* Create the semaphores(s) */
/* definition and creation of myBinarySem01 */
osSemaphoreDef(myBinarySem01);
myBinarySem01Handle = osSemaphoreCreate(osSemaphore(myBinarySem01), 1);

/* definition and creation of myCountingSem01 */
osSemaphoreDef(myCountingSem01);
myCountingSem01Handle = osSemaphoreCreate(osSemaphore(myCountingSem01),
7);
```

```
    /* Create the timer(s) */
  /* definition and creation of myTimer01 */
  osTimerDef(myTimer01, Callback01);
  myTimer01Handle = osTimerCreate(osTimer(myTimer01), osTimerPeriodic,
NULL);

  /* definition and creation of myTimer02 */
  osTimerDef(myTimer02, Callback02);
  myTimer02Handle = osTimerCreate(osTimer(myTimer02), osTimerOnce, NULL);

  /* Create the mutex(es) */
  /* definition and creation of myMutex01 */
  osMutexDef(myMutex01);
  myMutex01Handle = osMutexCreate(osMutex(myMutex01));

  /* Create the recursive mutex(es) */
  /* definition and creation of myRecursiveMutex01 */
  osMutexDef(myRecursiveMutex01);
  myRecursiveMutex01Handle =
osRecursiveMutexCreate(osMutex(myRecursiveMutex01));
```

FreeRTOS heap usage

The **FreeRTOS Heap usage** tab displays the heap currently used and compares it to the TOTAL_HEAP_SIZE parameter set in the **Config Parameters** tab. When the total heap used crosses the TOTAL_HEAP_SIZE maximum threshold, it is shown in fuchsia and a cross of the same color appears on the tab (see *Figure 98*).

Figure 98. FreeRTOS heap usage



4.5.16 Setting HAL timebase source

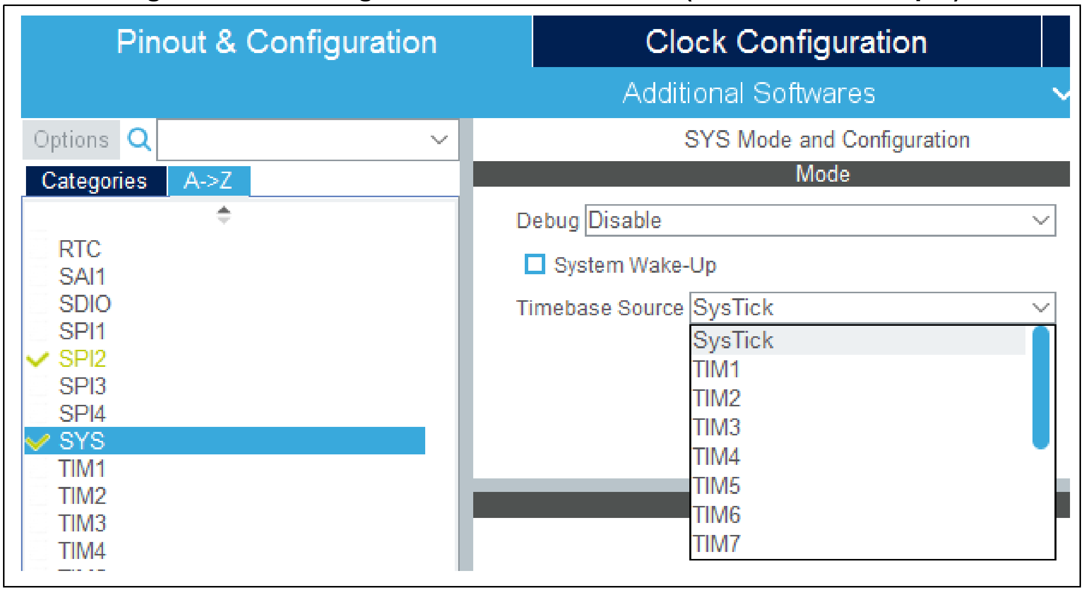
By default, the STM32Cube HAL is built around a unique timebase source, the Arm® Cortex® system timer (SysTick).

However, HAL-timebase related functions are defined as weak, so that they can be overloaded to use another hardware timebase source. This is strongly recommended when the application uses an RTOS, since this middleware has full control on the SysTick configuration (tick and priority) and most RTOSs force the SysTick priority to be the lowest.

Using the SysTick remains acceptable if the application respects the HAL programming model, that is, does not perform any call to HAL timebase services within an Interrupt Service Request context (no dead lock issue).

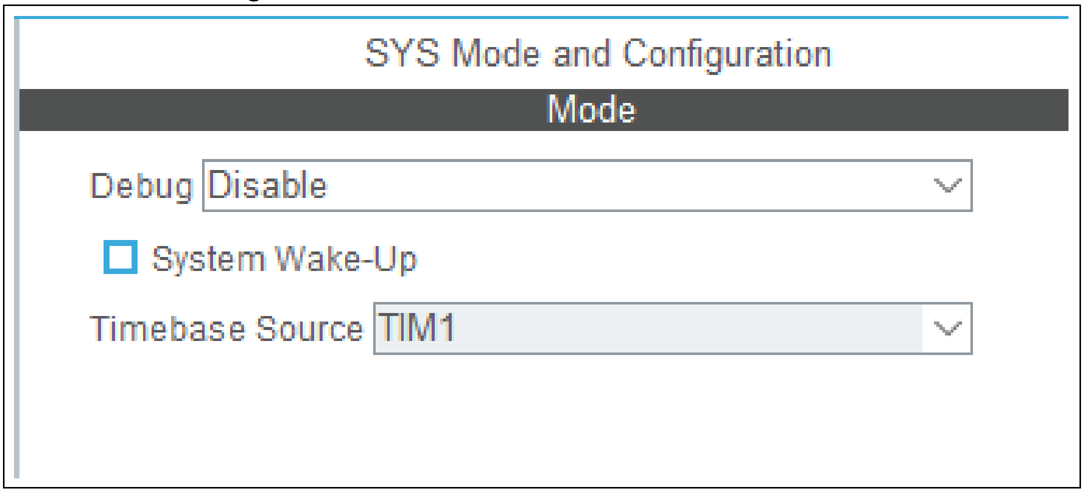
To change the HAL timebase source, go to the SYS peripheral in the **Component list** panel and select a clock among the available sources, such as SysTick, TIM1, TIM2 (see *Figure 99*).

Figure 99. Selecting a HAL timebase source (STM32F407 example)



When used as timebase source, a given peripheral is grayed and can no longer be selected (see *Figure 100*).

Figure 100. TIM1 selected as HAL timebase source

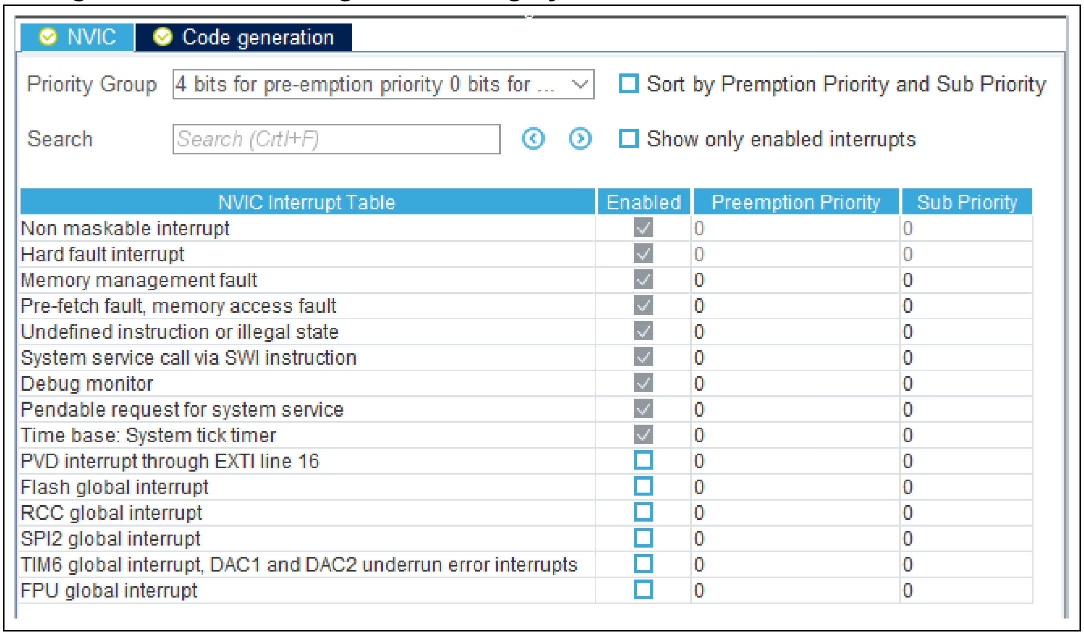


As illustrated in the following examples, the selection of the HAL timebase source and the use of FreeRTOS influence the generated code.

Example of configuration using SysTick without FreeRTOS

As illustrated in *Figure 101*, the SysTick priority is set to 0 (High) when using the SysTick without FreeRTOS.

Figure 101. NVIC settings when using SysTick as HAL timebase, no FreeRTOS



NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	☑	0	0
Hard fault interrupt	☑	0	0
Memory management fault	☑	0	0
Pre-fetch fault, memory access fault	☑	0	0
Undefined instruction or illegal state	☑	0	0
System service call via SWI instruction	☑	0	0
Debug monitor	☑	0	0
Pendable request for system service	☑	0	0
Time base: System tick timer	☑	0	0
PVD interrupt through EXTI line 16	☐	0	0
Flash global interrupt	☐	0	0
RCC global interrupt	☐	0	0
SPI2 global interrupt	☐	0	0
TIM6 global interrupt, DAC1 and DAC2 underrun error interrupts	☐	0	0
FPU global interrupt	☐	0	0

Interrupt priorities (in main.c) and handler code (in stm32f4xx_it.c) are generated accordingly:

- main.c file

```
/* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
```

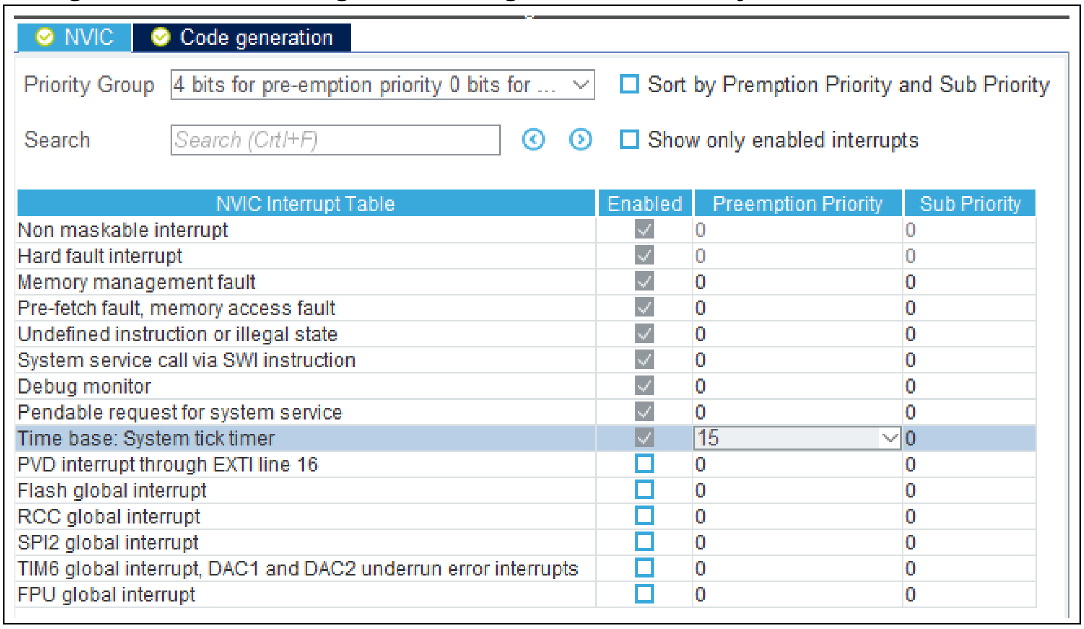
- stm32f4xx_it.c file

```
/**
* @brief This function handles System tick timer.
*/
void SysTick_Handler(void)
{
  /* USER CODE BEGIN SysTick_IRQn 0 */
 /* USER CODE END SysTick_IRQn 0 */
  HAL_IncTick();
  HAL_SYSTICK_IRQHandler();
  /* USER CODE BEGIN SysTick_IRQn 1 */

  /* USER CODE END SysTick_IRQn 1 */
}
```

Example of configuration using SysTick and FreeRTOS

As illustrated in *Figure 102*, the SysTick priority is set to 15 (Low) when using the SysTick with FreeRTOS.

Figure 102. NVIC settings when using FreeRTOS and SysTick as HAL timebase



As shown in the following code snippets, the SysTick interrupt handler is updated to use CMSIS-os osSystickHandler function.

- main.c file

```
  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 15, 0);
```

- stm32f4xx_it.c file

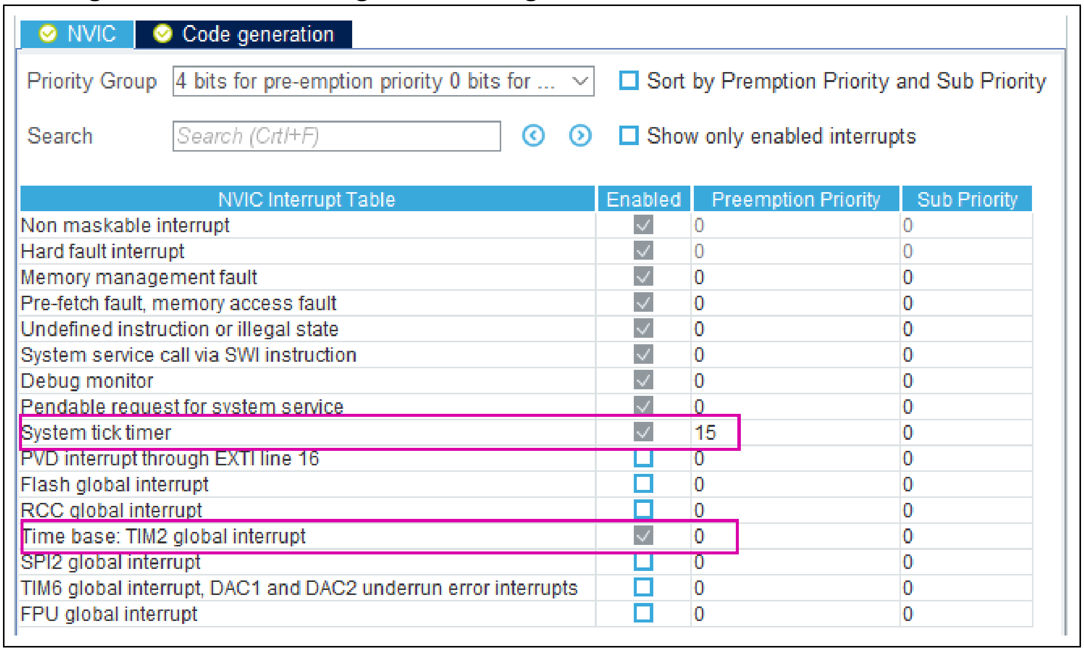
```
/**
* @brief This function handles System tick timer.
*/
void SysTick_Handler(void)
{
  /* USER CODE BEGIN SysTick_IRQn 0 */

  /* USER CODE END SysTick_IRQn 0 */
  HAL_IncTick();
  osSystickHandler();
  /* USER CODE BEGIN SysTick_IRQn 1 */

  /* USER CODE END SysTick_IRQn 1 */
}
```

Example of configuration using TIM2 as HAL timebase source

When TIM2 is used as HAL timebase source, a new stm32f4xx_hal_timebase_TIM.c file is generated to overload the HAL timebase related functions, including the *HAL_InitTick* function that configures the TIM2 as the HAL time-base source.

The priority of TIM2 timebase interrupts is set to 0 (High). The SysTick priority is set to 15 (Low) if FreeRTOS is used, otherwise is set to 0 (High).

Figure 103. NVIC settings when using FreeRTOS and TIM2 as HAL timebase



The stm32f4xx_it.c file is generated accordingly:

- SysTick_Handler calls osSystickHandler when FreeRTOS is used, otherwise it calls HAL_SYSTICK_IRQHandler.
- TIM2_IRQHandler is generated to handle TIM2 global interrupt.

4.6 Pinout & Configuration view for STM32 MPUs

For STM32MPUs the **Pinout & Configuration** view allows the user to:

- assign components to one or several run time contexts
- configure peripherals as boot devices
- select the peripherals to be managed by boot loaders
- assign GPIOs to one runtime (see *Figure 105*).

These possibilities are offered in two different panels (see *Figure 104*):

- from the component tree panel, listing all supported peripherals and middleware (the "Show contexts" option must be enabled)
- from each component mode panel, opened by clicking the component name.

Figure 104. STM32MPUs boot devices and runtime contexts

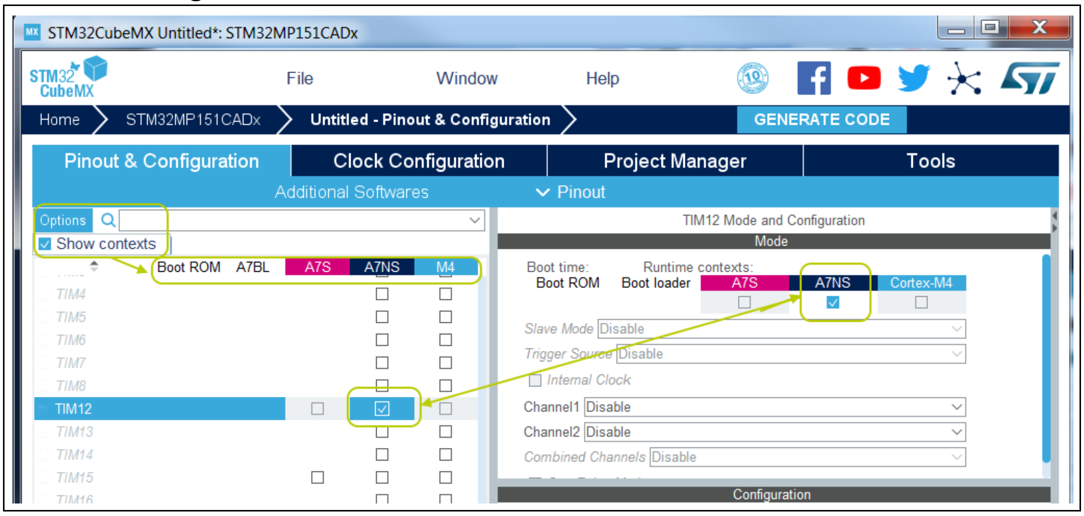
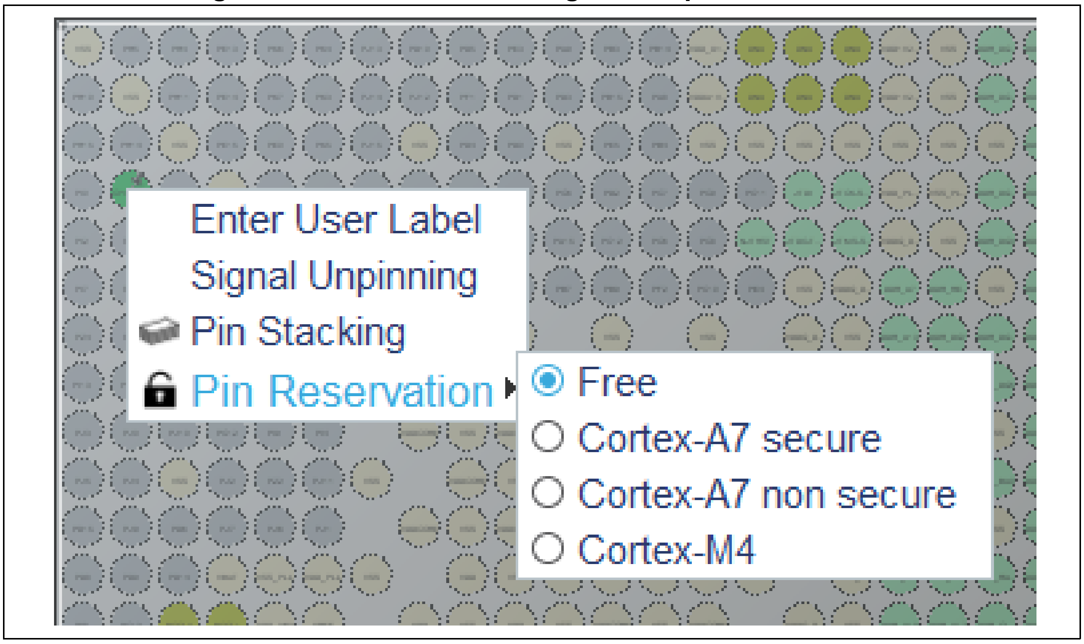


Figure 105. STM32MPUs: assignment options for GPIOs



4.6.1 Run time configuration

On these multi-core (Arm® Cortex®-A7 dual-core and Cortex-®M4) and multi-firmware devices, each firmware is executing on one of the cores. The association between firmware and core defines a runtime context. Three runtime contexts are available:

1. Cortex-A7 Non Secure running the Linux kernel
2. Cortex-A7 Secure running the SP_min
3. Cortex-M4 running the STM32Cube firmware.

Assigning a component to a runtime context means specifying which context(s) will control the component at runtime. Assignments to a Cortex-A7 context are reflected in the device tree code generation, while assignments to the Cortex-M4 context are reflected in STM32Cube based C code generation (refer to code generation sections for more details).

The component assignment to a context is done in the context dedicated column.

4.6.2 Boot stages configuration

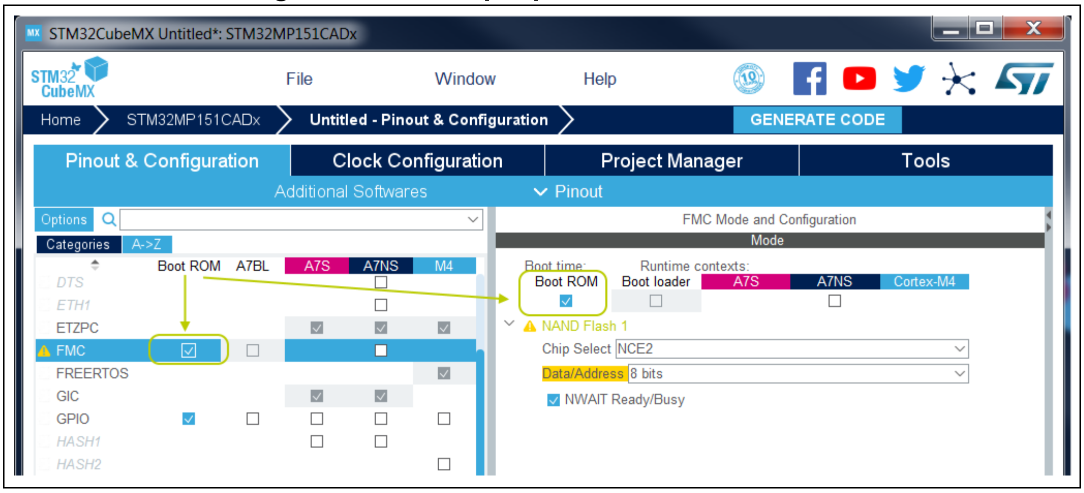
Boot ROM peripherals selection

Several execution stages are needed by the microprocessor to be up and running.

The binary code embedded in the ROM is the first to be executed. It uses a default configuration to initialize the clock tree and all peripherals involved in the boot detection.

The peripherals managed by the boot ROM program can be selected as boot devices. This choice is done in the Boot ROM column (see *Figure 106*).

Figure 106. Select peripherals as boot devices



When a peripheral is set as boot device, it imposes a specific pinout: some signals have to be mapped exclusively on pins visible by the boot ROM and only these signals/pins are taken into account by the boot ROM program.

When a functional mode of a ROM-bootable peripheral is set, the pinout linked to this mode is the same of that for a runtime context except for the signals imposed on specific pins by the boot ROM code.

During the boot step (boot ROM code execution), the peripheral is running only with the sub-set of bootable signals and pins. After boot, during runtime, the peripheral runs with all signals necessary to the selected functional mode.

Boot loader (A7 FSBL) peripherals selection

When the board starts, the launching of each of the Cortex-A7 runtime contexts (Secure and Non Secure) on which a firmware executes (for example Linux kernel for Cortex-A7 Non Secure) preceded by an early boot execution stage, that is before U-Boot relocation in DDR.

The Boot loader (A7 FSBL) column is used to define which devices can be managed during this Boot loader stage.

This assignment are reflected in the different device trees generated (refer to code generation sections for more details).

4.7 RIF configuration

Some STM32 products, like the STM32MP25x, have a special feature called RIF (resource isolation framework), used as a security guard for the their peripherals and memory. RIF decides which blocks the CPU can use, and manages the support systems for them. For details on how RIF operates, visit the STM32MPU Wiki website.

When the user sets up RIF in the STM32CubeMX program, the basic steps are the same, independently from the used device, even if there are several available options.

4.7.1 Configuration approach

In STM32CubeMX, the way the RIF keeps blocks safe is controlled by how user sets up them by software. When the settings change, STM32CubeMX checks them, translates what the user has done, and shows the updates in a special section called RIF panel.

User cannot set the access level or their special functions only by using software settings. This is managed by the main, trusted part of the software, with special access (Privileged mode). If there is need to use a setup where some blocks are used by less trusted software without special access (non-Privileged mode), user can make the changes in the RIF panel.

Blocks that user cannot set up with a software tool (like some memory areas in the STM32MP25), can still be protected by using the RIF panel.

The RIF panel is designed to display the security settings for the whole microcontroller (SOC level) in a way is similar to what detailed in the reference manual.

In the final steps:

- The system creates a set of rules (RIF configuration) that determine who is allowed to use different parts of the microcontroller. These rules are written out as source code.
- The code that sets up the microcontroller hardware blocks (like memory and peripherals) is made to match the software settings and the access rules user has set. This ensures that everything works together, without conflicts.

4.7.2 RIF global configurations

The RIF configuration panel can be conveniently accessed through the IP panel itself. This is because the RIF is integrated as a regular security IP within the STM32CubeMX system.

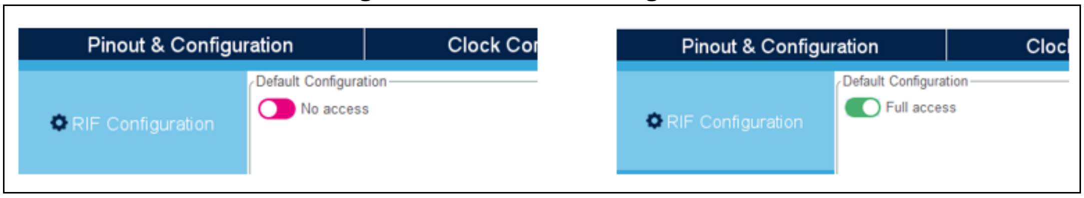
RIF global configurations for STM32MP2

The RIF configuration panel contains only one configuration, named Default configuration. The user can either lock down unused resources to prevent access, or leave them open for unrestricted use.

Two choices are proposed:

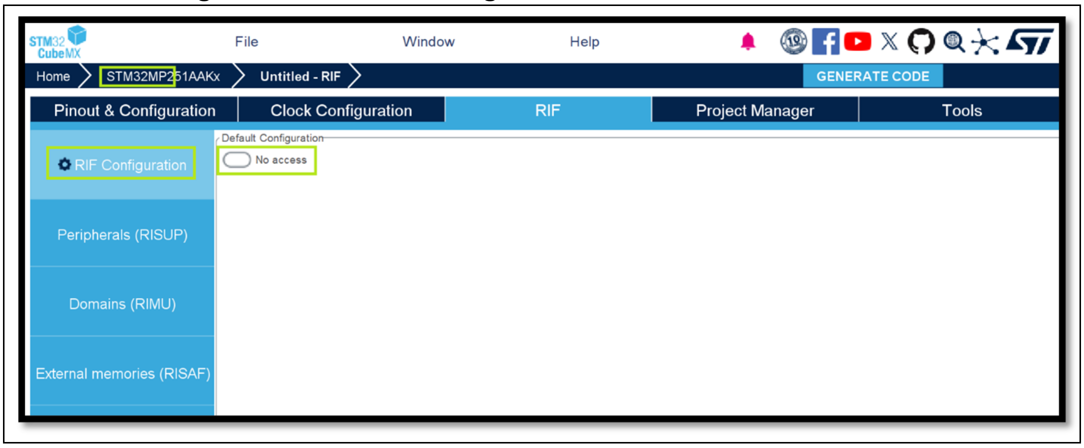
- No access: blocks the use of the resource. No one can read from it, write to it, or use it in any way.
- Full access: the resource can be used, it can be read and written without any restriction.

Figure 107. Default configuration



For the STM32MP2 series, there is a unique RIF configuration provided to the user. The radio button is disabled and indicates “No access” (see *Figure 108*): the user cannot read, write, or use it in any form.

Figure 108. Default configuration for the STM32MP2 series



RIF global configurations for STM32N6

For the STM32N6 series, the RIF default configuration is not supported.

Figure 109. RIF configuration extension in IPs panel for the STM32MP2 series

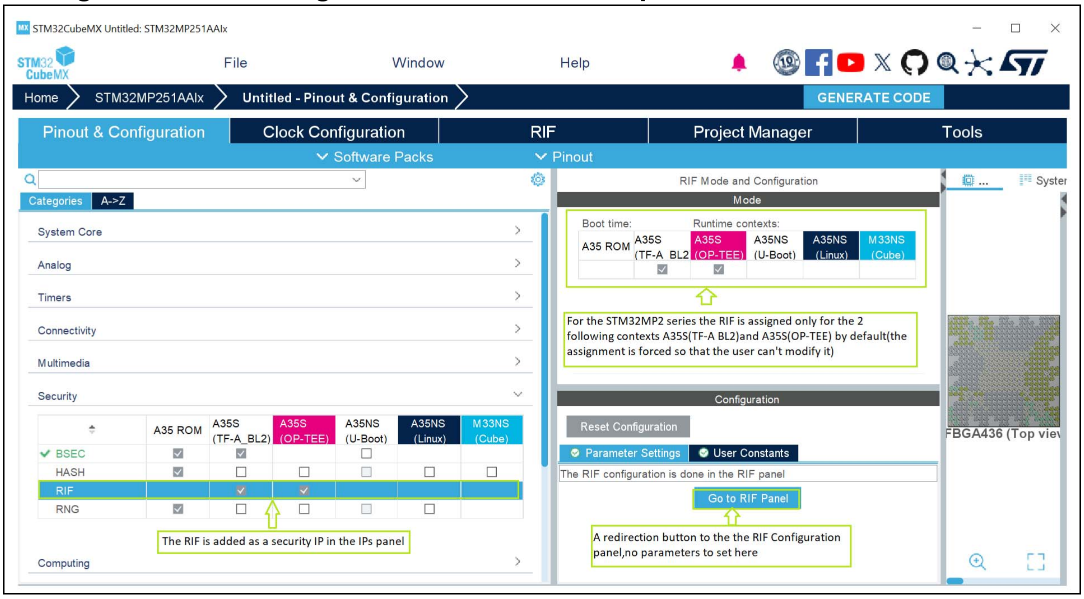
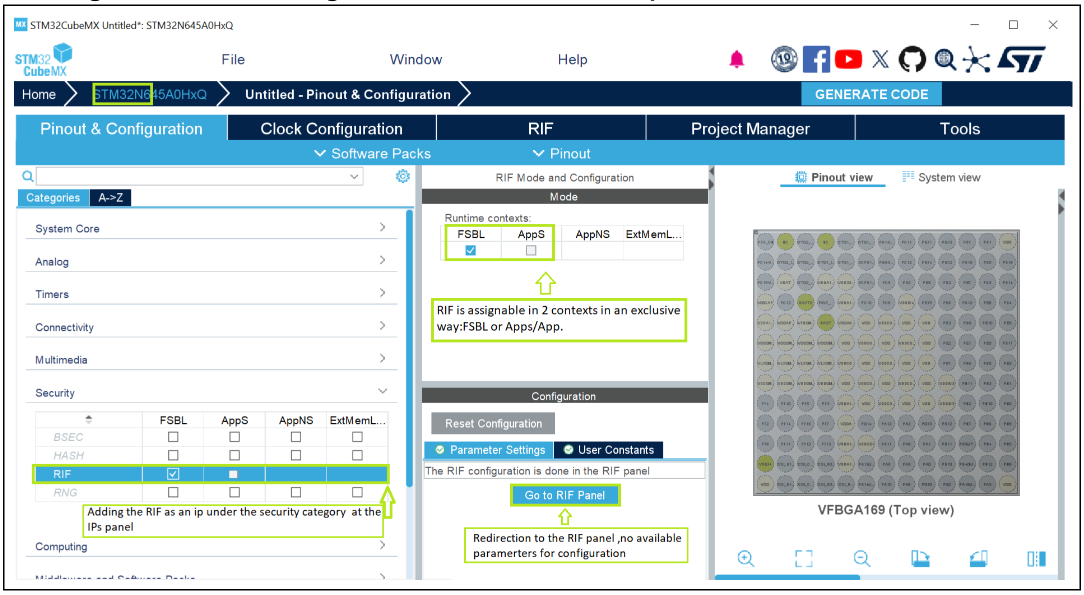


Figure 110. RIF configuration extension in IPs panel for the STM32N6 series



4.7.3 Peripherals protection

Microcontroller peripherals can be classified by their function or by how they are protected:

- Sorted by function:
  - Standard peripherals: do processing and can interact with other devices (such as I2C and UART).
  - Service peripherals: do processing but do not interact with other devices (such as CRYP and HASH).
  - System peripherals: provide services to other peripherals (such as RCC, GPIO, DMA).
- Sorted by protection scheme:
  - The whole peripheral is protected (non-RIF-aware IP). Access rules are set for the whole peripheral. The RISUP subsystem manages the protection.
  - Protection by specific function (RIF-aware and pseudo-RIF-aware IP). Access rules are based on specific functions/features. The peripheral itself controls the protection. For pseudo-RIF-aware IPs, although they are RIF-aware, their feature protection is managed by the RISUP.

In STM32CubeMX, the security for the microcontroller peripherals is set through the RIF, based on the software settings. The program figures out the security rules automatically, based on which parts (IP or IP features) are assigned to various parts of the software. When a part is assigned to a software area, it must be decided who can use, who can set it up, and what is allowed to do with it.

The configuration of access rights are available within the RIF panel:

- Non RIF-aware and pseudo RIF-aware IPs: access rights are managed through the RISUP panel.
- RIF-aware IPs: access rights for these IPs are configured in the RIF-aware IPs panel.

4.7.4 Peripheral instance protection

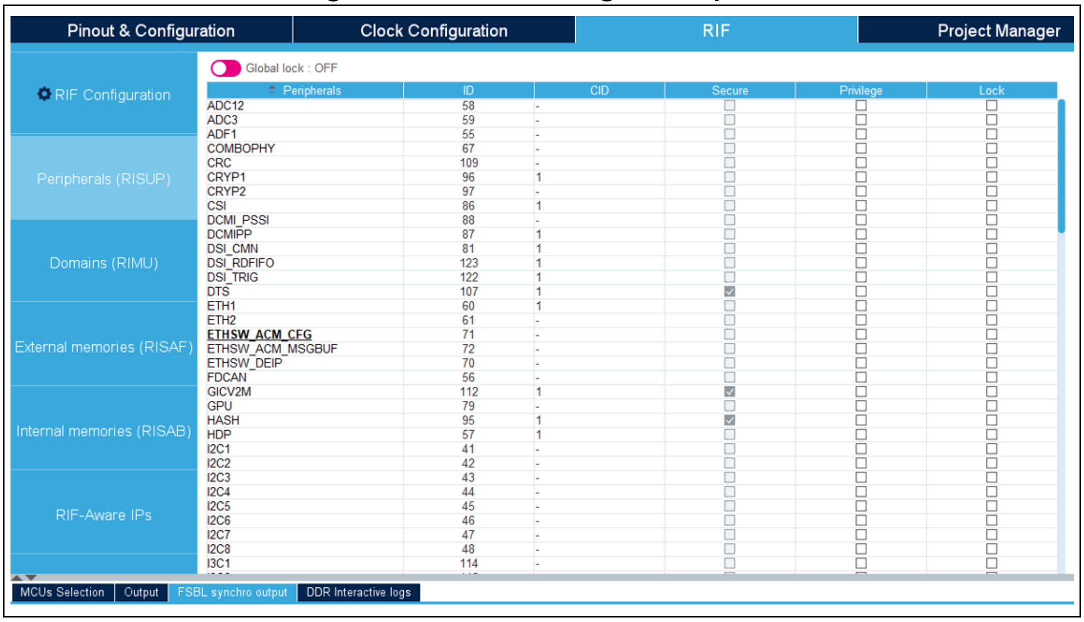
Peripheral instance protection for STM32MP2

The assignment of IPs (or IP features in the case of pseudo-RIF-aware IPs) to software contexts directly determines access rights. These rights are then displayed in the RIF RISUP configuration panel, which outlines the level of protection provided by the RIF, and where advanced configurations can be specified for each peripheral instance.

The RISUP configuration panel for STM32MP2 series is composed of:

- The list of IPs and features of pseudo-RIF-aware IPs
- IP identifiers (ID), as defined in the reference manual
- IP master owners compartment Identifiers (CID) and security states
- The RIF privilege level for each IP
- The lock state for each IP

Figure 111. RISUP configuration panel



The Lock blocks any change after boot (that is, after configuration in STM32CubeMX), to prevent software from subsequently making changes to the RIF elements.

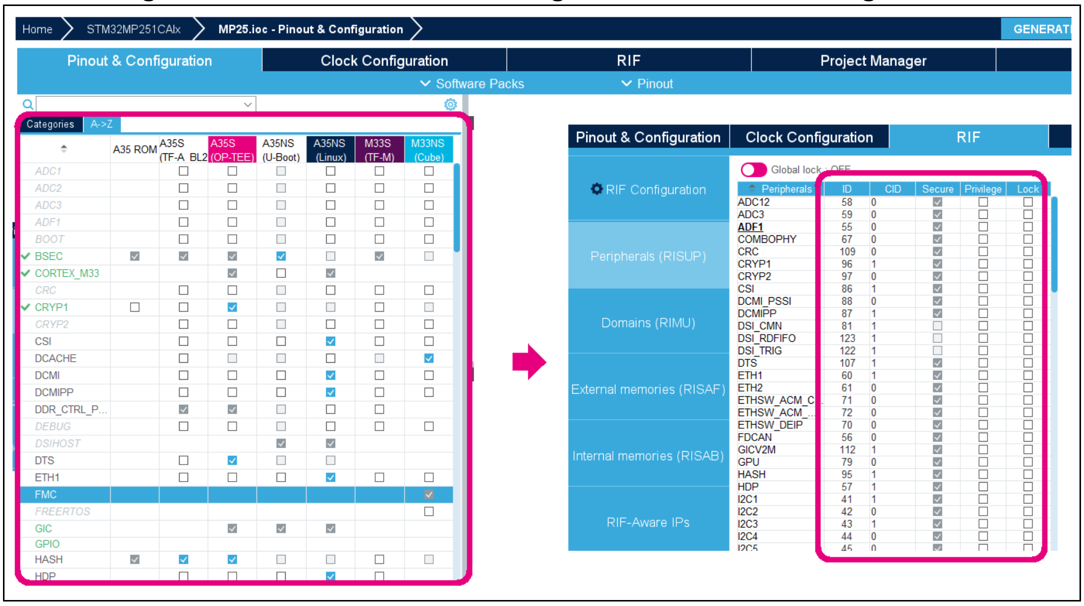
The Local Lock defines a Lock on independent elements.

Global Lock defines a Lock on a set of elements. By default, it is OFF. Global lock : OFF

Configuration example

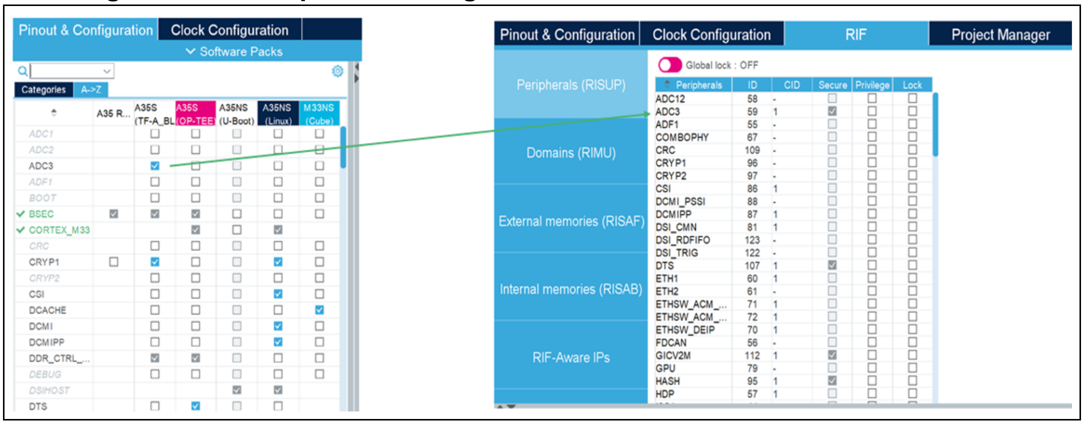
Figure 112 shows on left hand side the IP allocation per software context, and, on the right-hand side the equivalent in the RISUP configuration panel.

Figure 112. Software context configuration vs. RISUP configuration



For example, if the user sets ADC3 to Cortex-A35 secure context, on the RIF panel ADC3 is allocated to CID 1, and set secure. The user can then configure the privilege and the lock. If a peripheral is set in two contexts (Cortex-A35 and Cortex-M33), the allocated CID is 1&2.

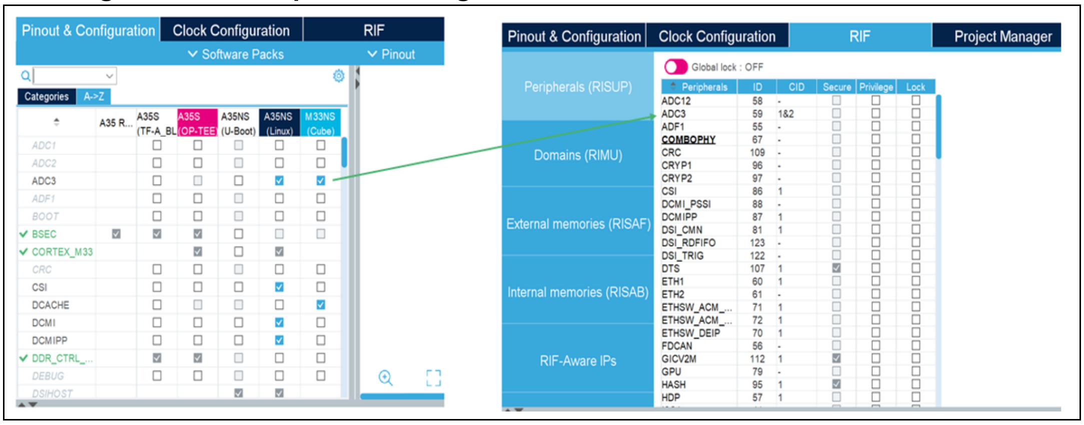
Figure 113. Example of IP assignment to one context and result in RISUP



If the user selects an IP in a Cortex-A35 Non Secure context and a Cortex-M33 Non Secure context, the CID is set to 1&2 and the Secure column is unticked, as shown in *Figure 114*.

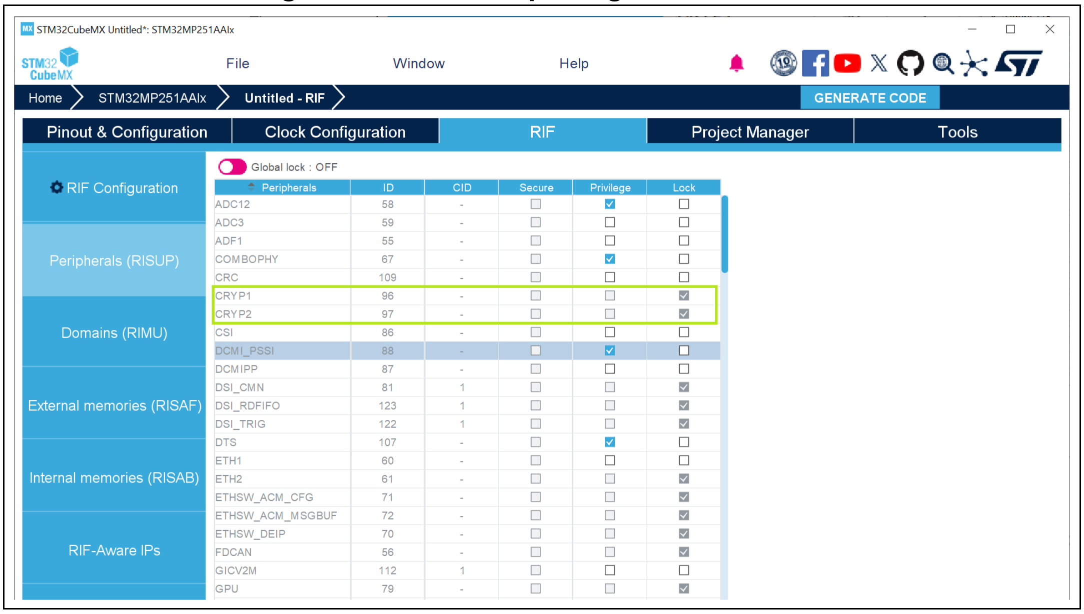
If the IP is not assigned to any software context, the CID column contain a –, and the Secure column is unticked (in the case of Full Access).

Figure 114. Example of IP assignment to two contexts and result in RISUP



The RISUP table contains entries for peripherals not supported by the MPU, such as CRYP1, CRYP2 on STM32MP251DAIx. These entries have the CID and lock cells permanently set and cannot be modified. The RISUP table is primarily read-only, with modifications limited to the MPU configuration.

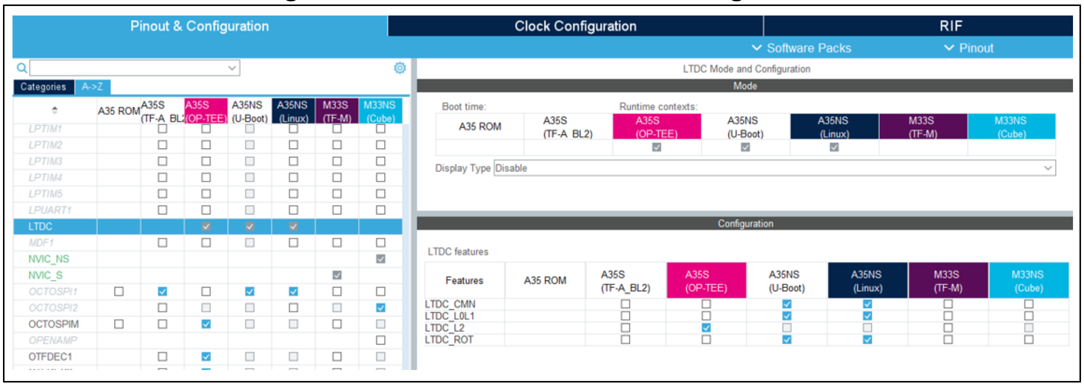
Figure 115. Lock and privilege in RISUP table



Note: Some IPs in RISUP do not exist in peripheral list, and some IPs are coupled. They show-up in the Peripheral column as one. As an example, ADC1 and ADC2 are shown as ADC12, ICACHE and DCACHE are shown as ICACHE_DCACHE.

The features of the pseudo RIF-aware IPs are also visible in the RISUP table, as shown in *Figure 116*.

Figure 116. Pseudo RIF-aware IP assignment



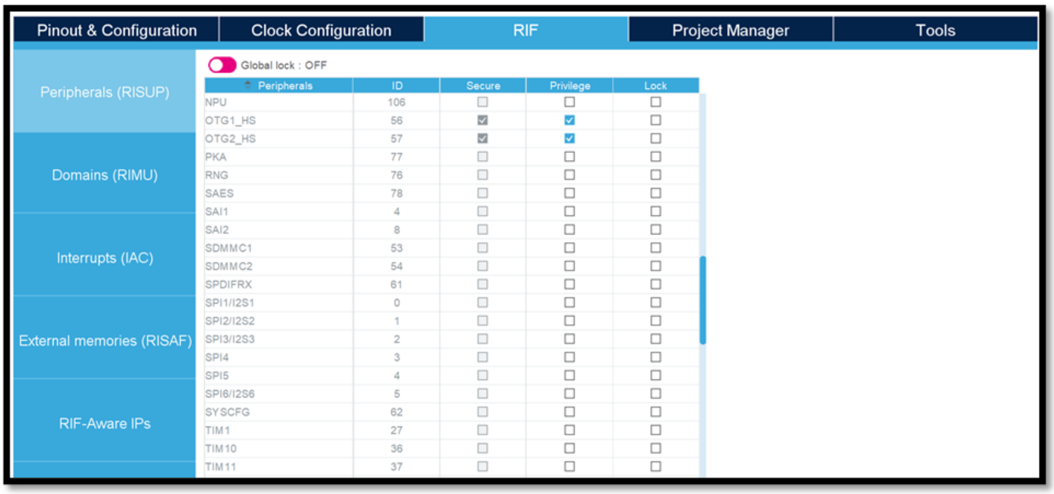
Peripheral instance protection for STM32N6

The RISUP panel for STM32N6 series (*Figure 117*) does not have the CID column because the STM32N6 contains a single M55 core. Consequently, the CID column is unnecessary as all peripherals in the STM32N6 are allocated by default to CID1, and the indication of RIF unused with a – is no longer available.

The panel is composed of five columns:

- Peripherals column: list of pseudo-RIF-aware IPs and features
- ID column: IP identifiers (as defined in the reference manual)
- Secure column: security state for each IP
- Privilege column: privilege level for each IP
- Lock column: lock state for each IP

Figure 117. Peripherals (RISUP) panel for the STM32N6 series



If the user chooses to create a new project as full secure (*Figure 118*), the column “Secure” is hidden, as all peripherals are working in secure mode (*Figure 119*).

Figure 118. Creation of a new project for the STM32N6 series - Secure projects

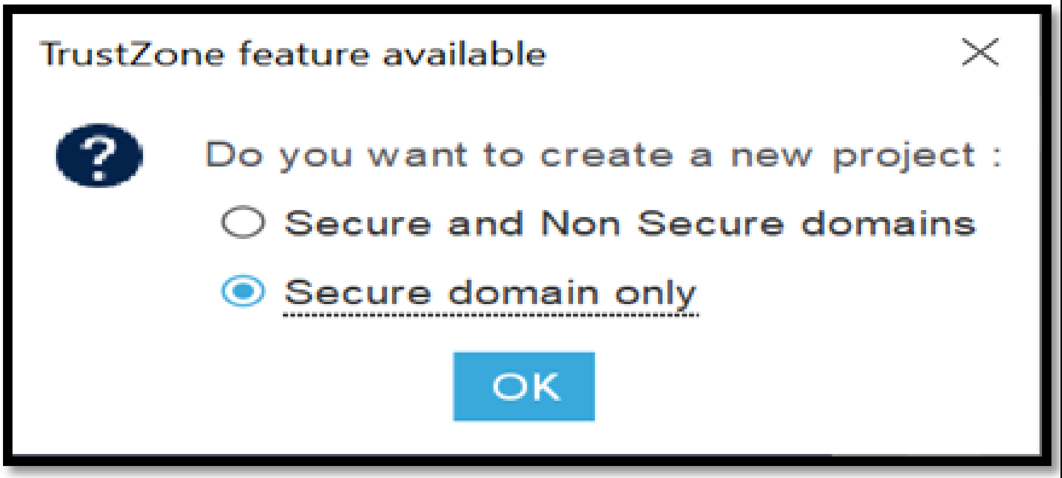
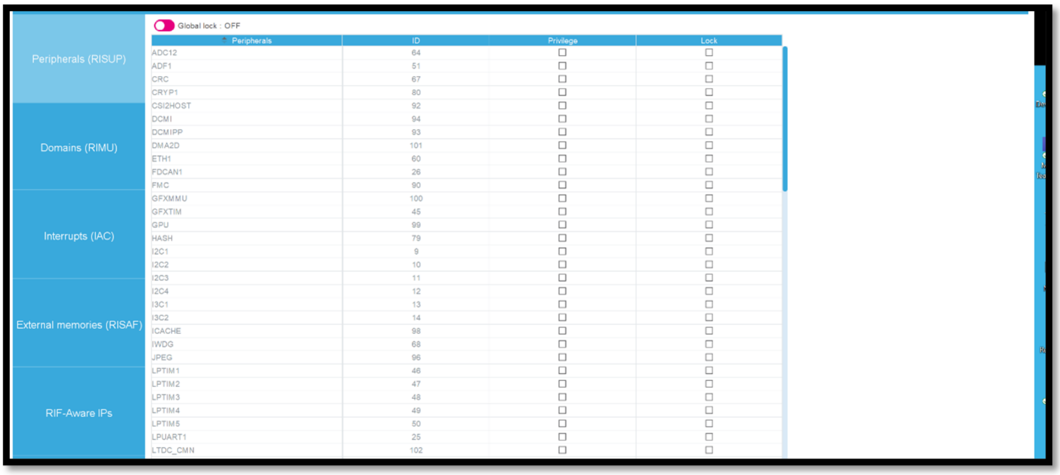


Figure 119. Peripherals (RISUP) panel for the STM32N6 series - Secure projects



4.7.5 IP feature protection

In certain scenarios, feature assignment can depend upon the feature assignment of another IP within the system.

Feature assignments are managed through the Features Configuration panel associated with each RIF-aware IP. For non-RIF-aware IPs, although access rights are inferred from the feature-to-software context assignments, they are documented in the IP sub-panel found within the RIF-aware IPs configuration panel.

The features assignment is combined with the IP modes:

- The features define which functionalities can be accessed, by which firmware
- The modes define which features are effectively used and initialized and open access to initialization parameters

The initialization parameters set depend on the corresponding feature assignment:

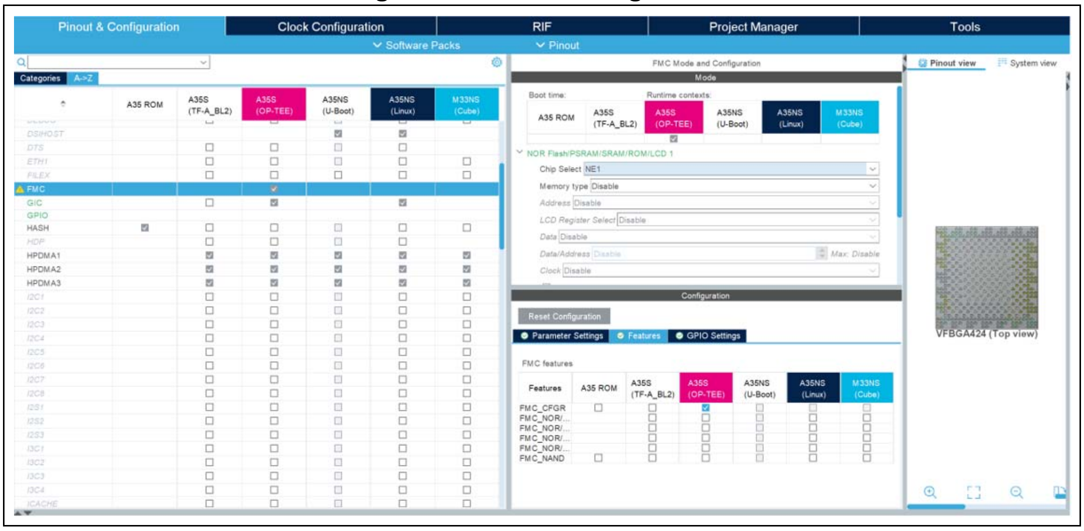
- HAL parameters when feature is assigned to a Cube firmware
- No parameters for firmware initialized via a device tree system (such as an OpenSTLinux firmware)

Configuration example

In the following example, the FMC IP is configured to work as a RIF-aware IP:

- Click on FMC IP in Pinout & Configuration panel
- The FMC related features is displayed on the configuration panel on the right-hand side
- Select A35S (OP-TEE) for the features FMC_CFGR
- In the FMC Mode and Configuration panel, pick “NE1” in the “Chip select” drop down
- In the Configuration panel, three tabs are displayed (Parameter Settings, Features, GPIO Settings)

Figure 120. FMC configuration



Configuring FMC in a RIF panel:

- Click on the RIF tab
- Select the RIF aware IP tab on the left-hand side
- Choose FMC

Each feature can be configured as secure or privileged.

- The CID column represents the hardware context
- The security column comes from the security of software context
- The privilege column is set to false by default

Figure 121. RIF FMC panel

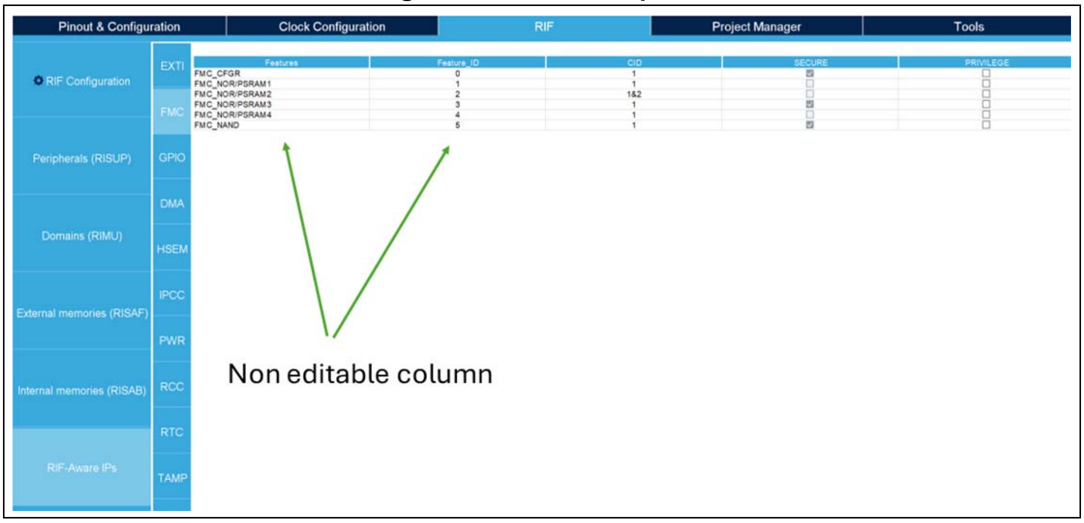


Figure 122. RTC features

Figure 123. RTC mode

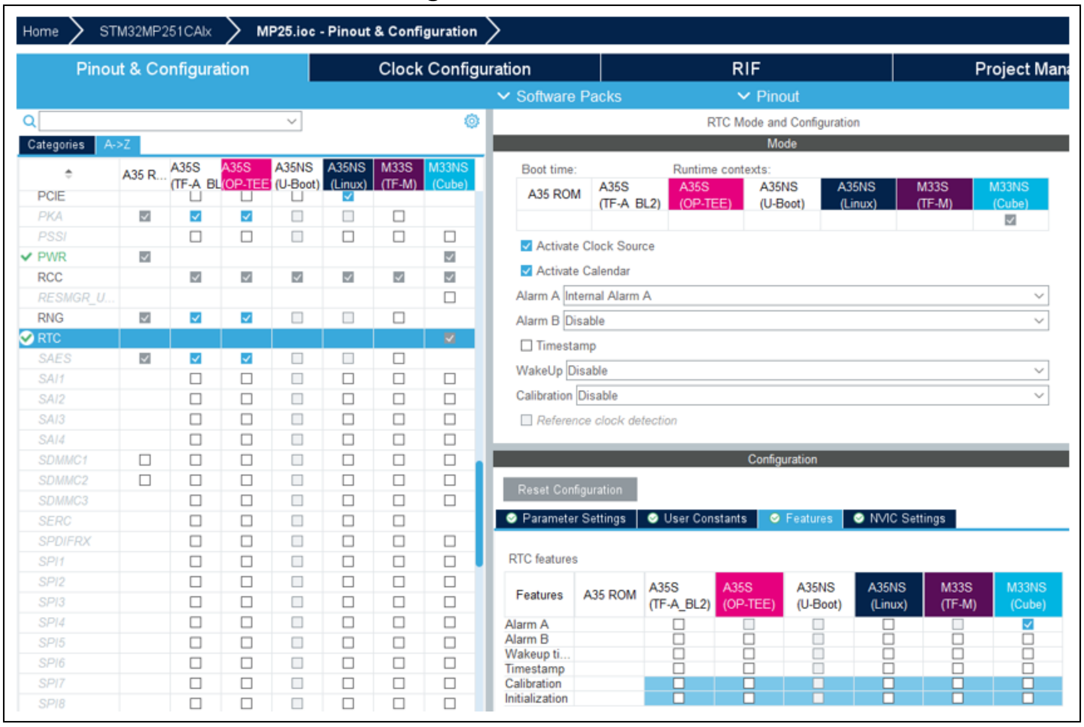
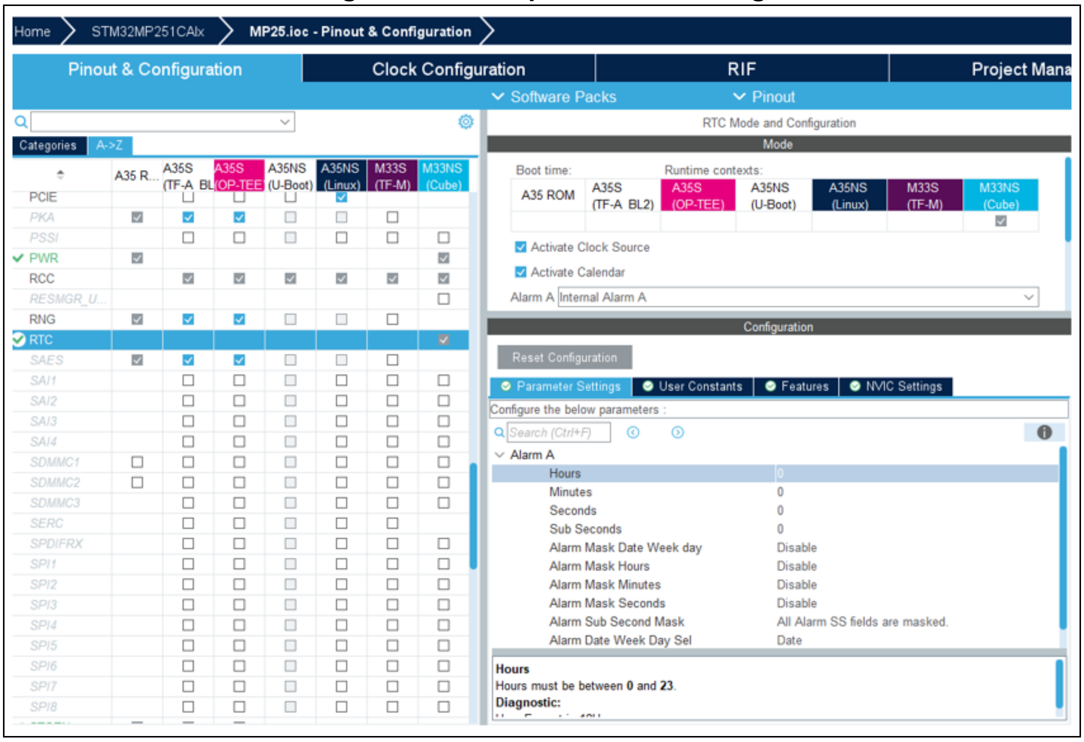


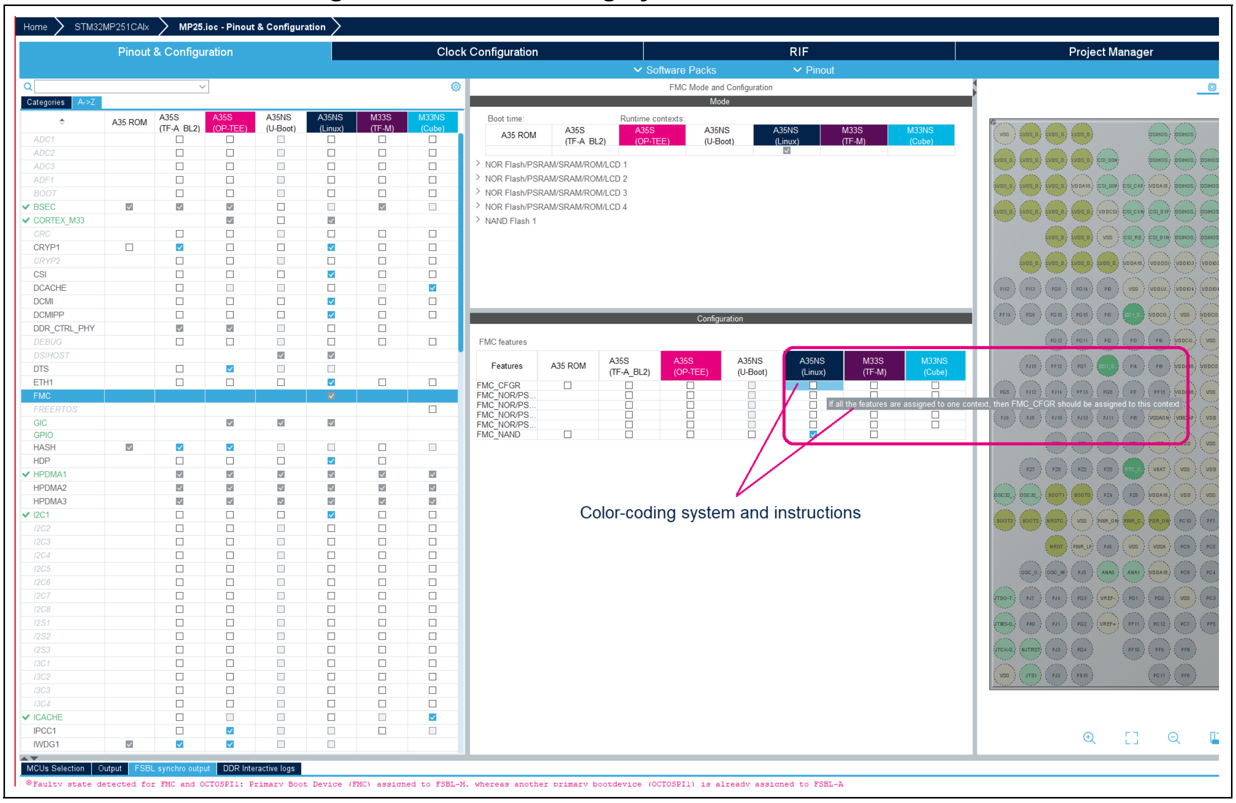
Figure 124. RTC parameters setting



4.7.6 Software constraints validation

When integrating software, there are specific guidelines for setting up access to integrated peripheral features, and to ensure that the software works correctly with the intended STMicroelectronics software architecture. These guidelines are recommendations, not mandatory requirements. STM32CubeMX provides a helpful feature in the Feature Assignment panel to follow these guidelines. It uses a color coding system and instructions to make the process easier.

Figure 125. Color coding system and instructions



4.7.7 Masters configuration

Masters configurations for STM32MP2

The RIMU is one of the core components of RIF. It allows some IPs (with data transfer capabilities) to be configured as a master. It can be used to assign an IP to a security domain by defining the secure, privilege, and compartment ID (CID).

RIMU allows the user to:

- See the list of master IPs (in their default configuration) for new domain creation
- Create new domain from each masters
- Configure the RIF security level of each master
- Configure the RIF privilege level of each master
- Configure global lock

Between RIMU and RISUP there is an inheritance relationship for common IPs. This relationship allows the IP to inherit the CID, security state, and privilege state from RISUP when the user does not define its own values.

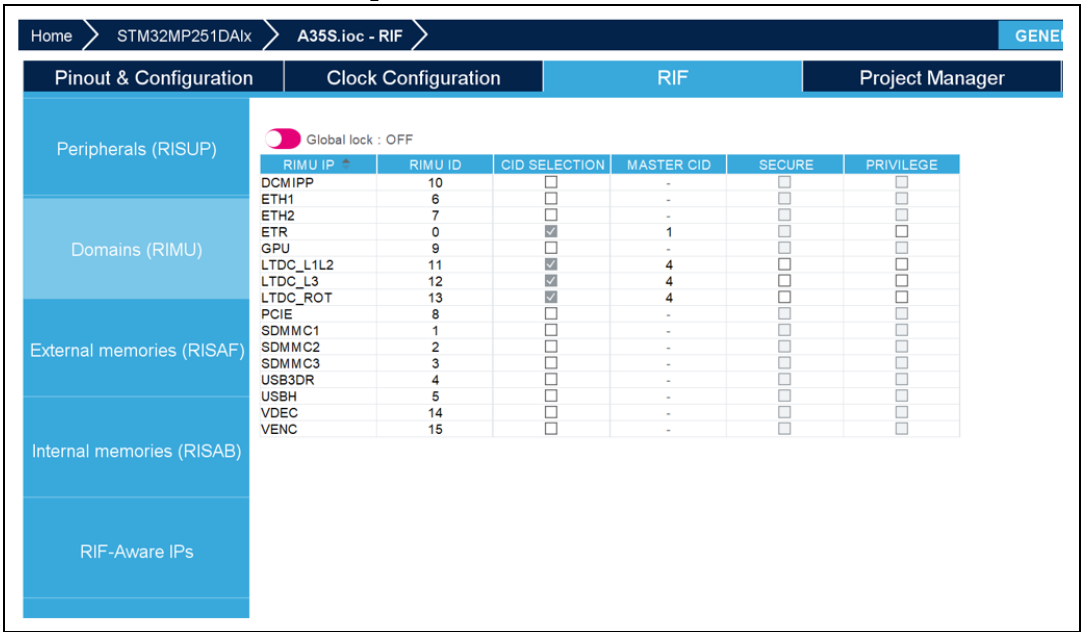
The user interface for STM32MP2 is composed of a table containing six columns:

1. IP name
2. IP id, which are unique
3. CID SELECTION, to select CID
4. Master CID, to change the CID value
5. Secure state, inherited from RISUP
6. PRIVILEGE state, when enabled

A global lock button on the top of RIMU table can be used to lock the RIMU.

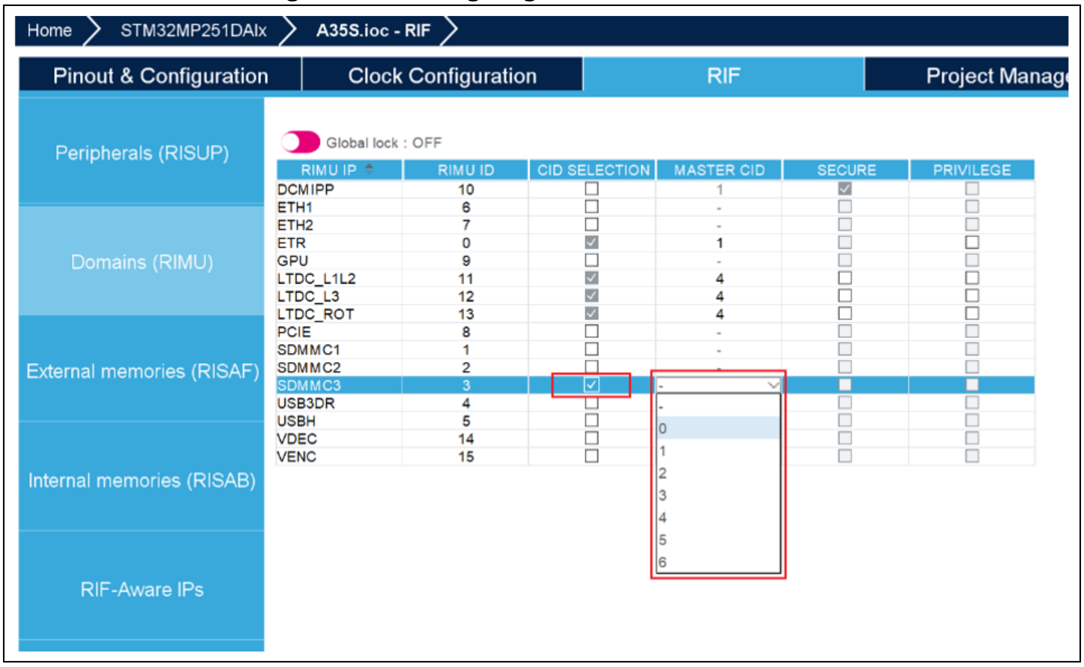
Note: The RIMU table may contain entries for peripherals not supported by the MPU. The configuration of these unsupported peripherals is possible, but the code generation process excludes any code related to them. This restriction is applicable only to MPU configurations within the RIMU table.

Figure 126. RIMU user interface



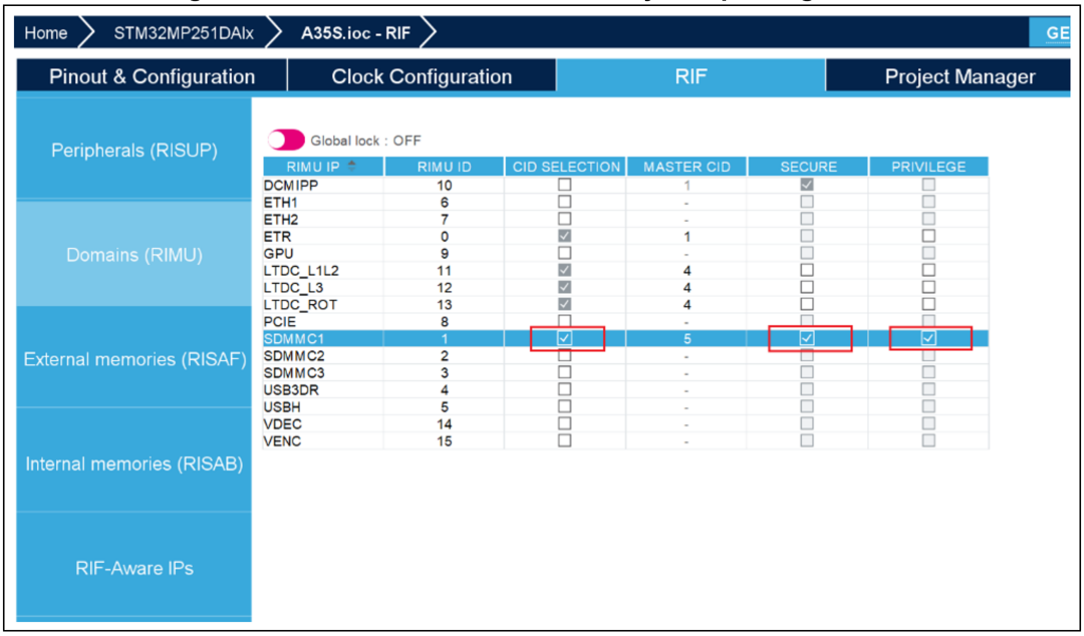
To define a CID for an IP, activate CID SELECTION, and then choose a value from 0 to 6, as shown in *Figure 127*.

Figure 127. Assigning a CID to an IP in RIMU



To change the security or privilege value for an IP, activate the appropriate CID SELECTION checkbox, as shown in *Figure 128*.

Figure 128. Modification of the security and privilege values



The inheritance relationship between RISUP and RIMU is established and valid only if the IP is assigned to a context in the Pinout & Configuration panel.

In the context of inheritance relationships, the user cannot change the value of security and privilege if they are false in RISUP, it can only change them from true to false if they are true.

Figure 129. IP assignment to a context

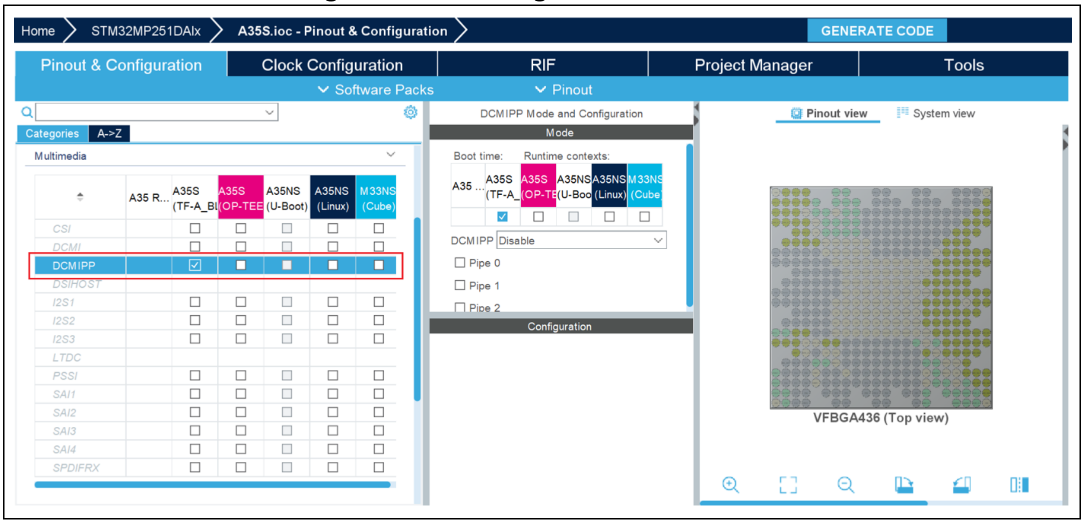


Figure 130. Result in RISUP of an IP assignment to a context

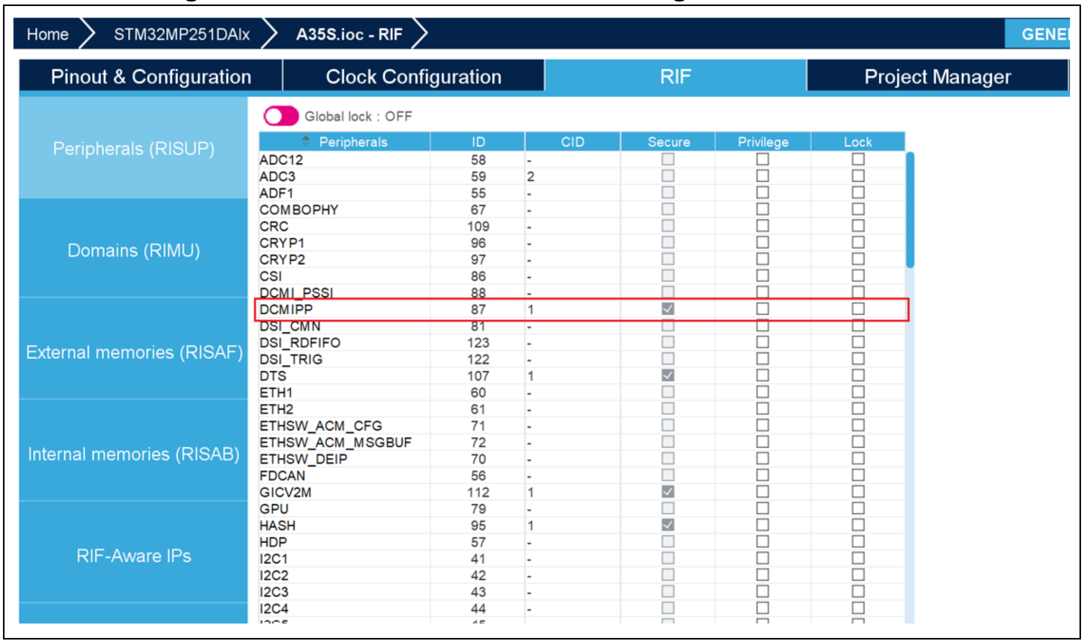
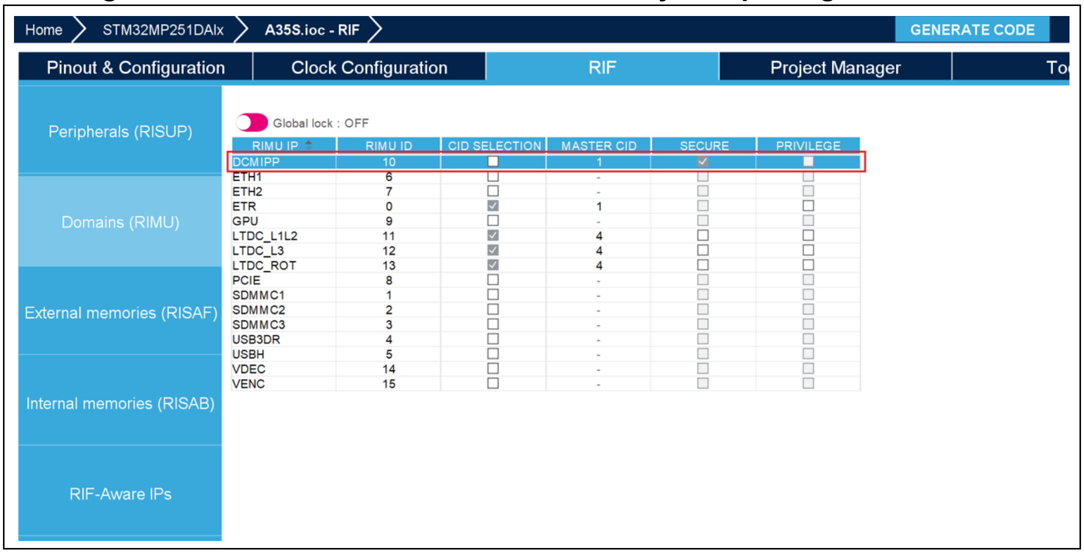


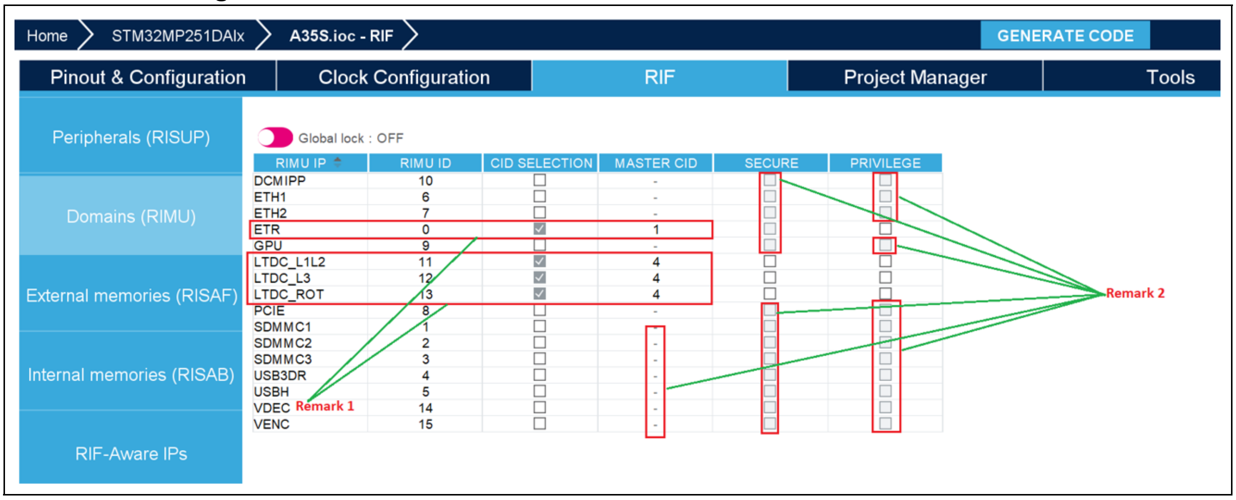
Figure 131. Inheritance of CID, state of security, and privilege from RISUP



Note that:

- Some IPs have default values
- If the user does not have the right to change values in the RIMU, these cells are greyed out
- If the user creates an STM32MP25xx project and selects the Cortex-M33 as the master, M33S (secured) the global lock button is activated by default
- When an IP is not used (CID = -) and the user checks the related CID SELECTION, a default value is assigned to the CID of the IP equal to the value of the TD CID

Figure 132. Default values for IPs and user modification restrictions

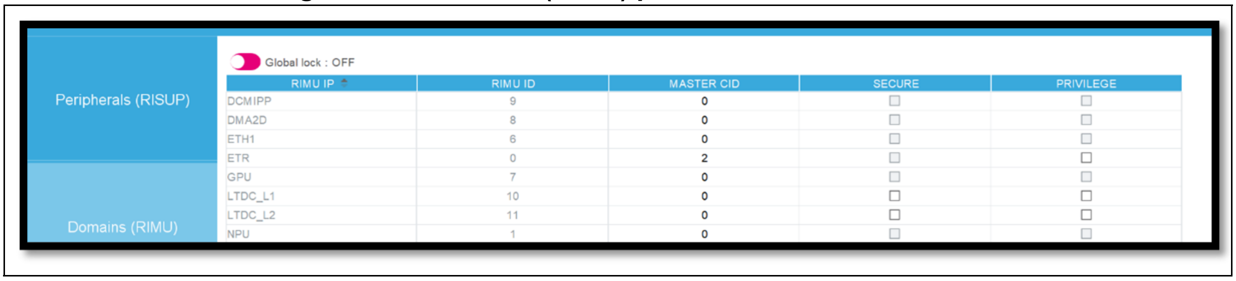


Masters configurations for STM32N6 series

The Domains (RIMU) panel for these series is composed of a table with five columns:

- RIMU IP: the name for each IP
- RIMU ID: unique for each IP
- MASTER CID: to set CID value for each IP
- SECURE: inherited from RISUP
- PRIVILEGE: privilege state if the IP is activated

Figure 133. Domains (RIMU) panel for STM32N6 series



4.7.8 Service peripherals protection

Service peripherals are special components that perform tasks or provide data for other parts of the system, and they do not have input/output ports (IOs). These peripherals are known as RIF-aware IPs, which means they are aware of the security and access framework RIF.

The user can set up these peripherals by configuring their features and modes to ensure that they are secure (protected) and ready to be used (enabled). The security settings are based on the assigned features, these settings are shown in a special area of the software called the RIF-aware IPs panel. Each type of RIF-aware IP has its own unique panel, different from the standard setup mentioned earlier. The peripherals available on STM32MP25 devices are detailed below.

HSEM

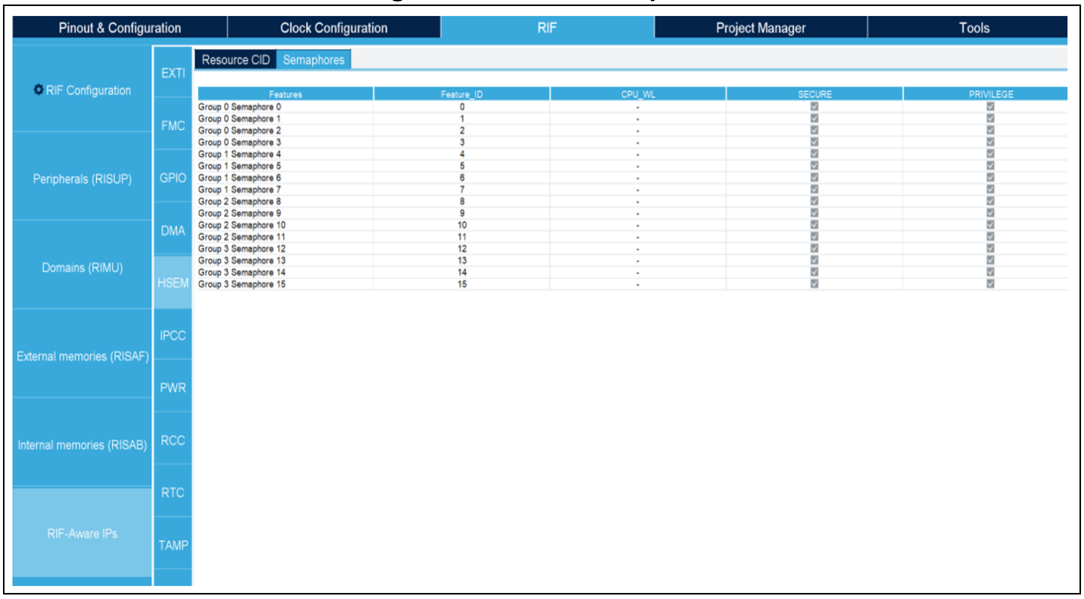
HSEM is not configurable in STM32CubeMX, but it is visible in RIF Panel to show and generate a default protection. It defines the filter access (secure and privilege) to HSEM features (16 HSEM semaphores).

As the IP is not used in the system, the protection is not configurable and forced to the “Default configuration”.

HSEM contains two tables, the first represents the CPU allocation per context, the second contains the features, their “CPU Whitelist” (CPU_WL), the security states and privileges.

All the features (semaphores in case of HSEM IP) are secured and privileged, as shown in *Figure 134*.

Figure 134. RIF HSEM panel



Note: For STM32MP21 products HSM is not present among RIF-aware IPs.

TAMP protection

TAMP for the STM32MP2 devices (*Figure 135*) contains two tabs:

- In the first, the user can configure the available resources, making them secure or privileged.
- In the second, the user can configure the memory zone area storing critical applications data.
  - Each zone can be resized using a dedicated panel available in the RIF configuration panel
  - Each zone is associated to a resource: the resource assignment defines the firmwares that can access a zone, and the access rights

For STM32N6 devices, the TAMP UI includes only a table that lists features along with their security and privilege statuses (see *Figure 136*). Users can select a value between 0 and 32 to define the security level (in SECURE column) for the two features backup registers protection offset (write, read/write) of the TAMP.

Figure 135. RIF TAMP panel (STM32MP2 devices)

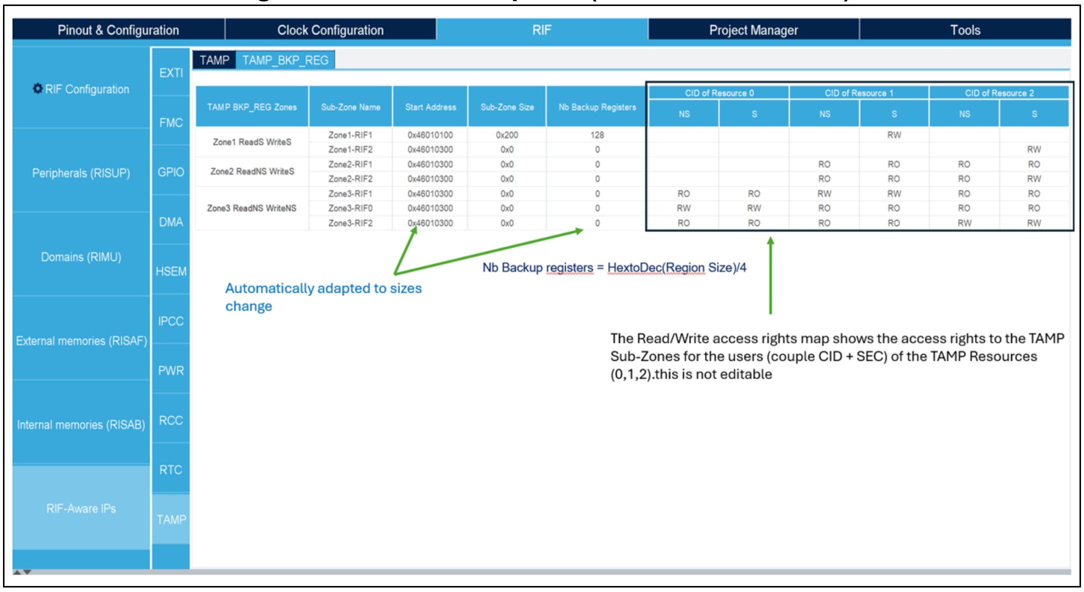
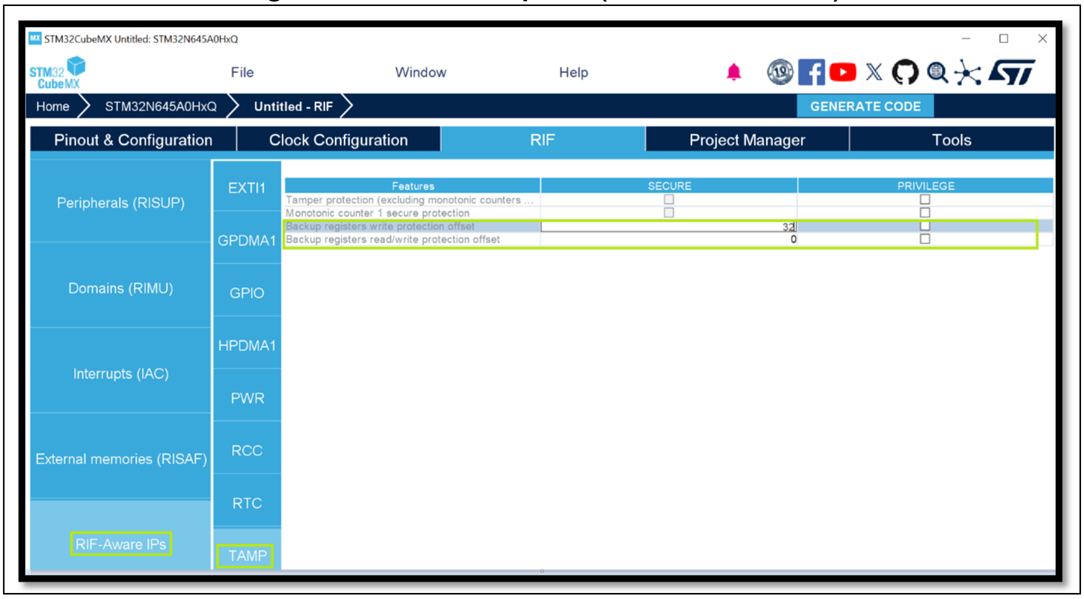


Figure 136. RIF TAMP panel (STM32N6 devices)



IPCC configuration

In the IPCC tab, the user can configure available resources, such as Resource features 0, 1 and 2, by setting their security levels or assigning privileged status.

PWR configuration

The PWR tab allows the user to manage settings for Resource 0, Resource 1, and Resource 2, providing options to secure these resources, or grant them special privileges.

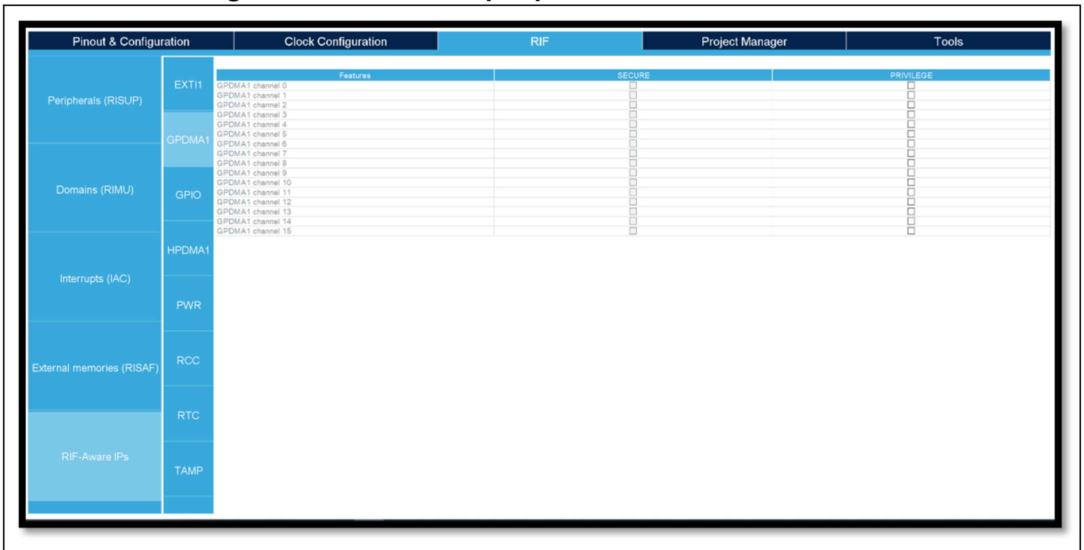
4.7.9 System peripherals (STM32MP2 and STM32N6 series)

System peripherals are components that share their functions and resources with other integrated peripherals (IPs). These system peripherals are designed to be RIF-aware, which means they are compatible with a certain security and access control system.

While these system peripherals generally use the same security setup as other RIF-aware IPs, they also have some unique features. The specific RIF configurations and what makes them different are described in the following subsections.

The RIF-aware IPs for STM32N6 are fewer than for STM32MP2, namely: EXTI1, GPDMA1, GPIO, HPDMA1, PWR, RCC, RTC, and TAMP. There is no need to display CID, as these MCUs are based on a single core.

Figure 137. RIF-aware peripherals for STM32N6 MCUs



IO configuration

There are two main types of IO (input/output) configurations:

- Alternate function IO (AF IO): used to transmit signals that the peripherals process.
- General purpose IO (GPIO) and external Interrupt IO (EXTI IO): serve general input/output functions and manage external interrupts.

For both types, security settings are automatically determined, based on their connections:

- For non-RIF-aware IPs, the security comes from the IP they are connected to
- For RIF-aware IPs, it is based on the specific features of the IP they are linked with

For GPIO and EXTI, the IO sets the security.

The assignments of IO to software contexts are displayed in the features panel specific to the GPIO IP. Additionally, the security settings (RIF protection) for these IO configurations can be found in the RIF-aware IP panel, under the GPIO sub-section

Figure 138. IO protection inheritance for a non-RIF-aware IP (I2C)

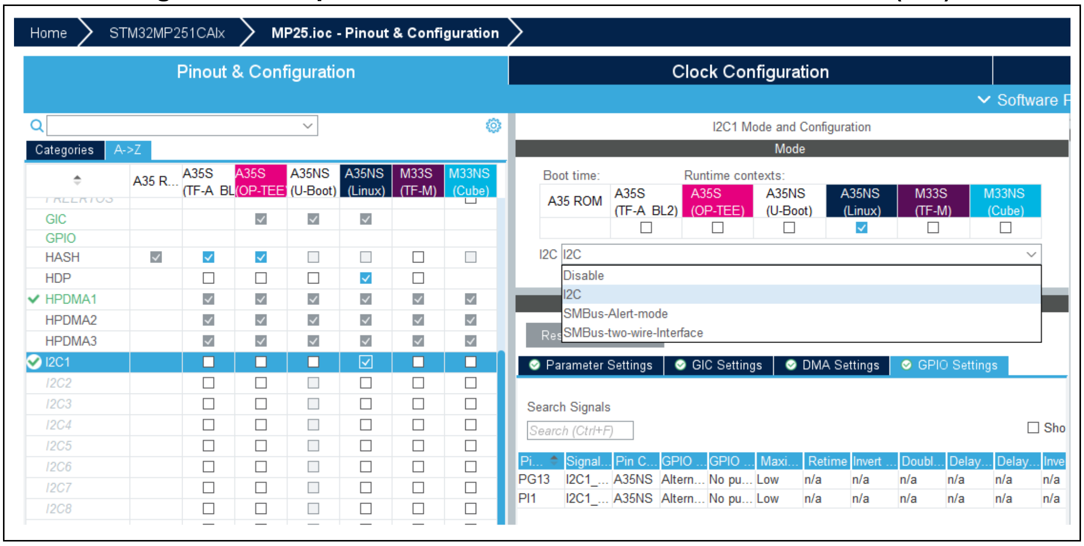


Figure 139. GPIO IP panel

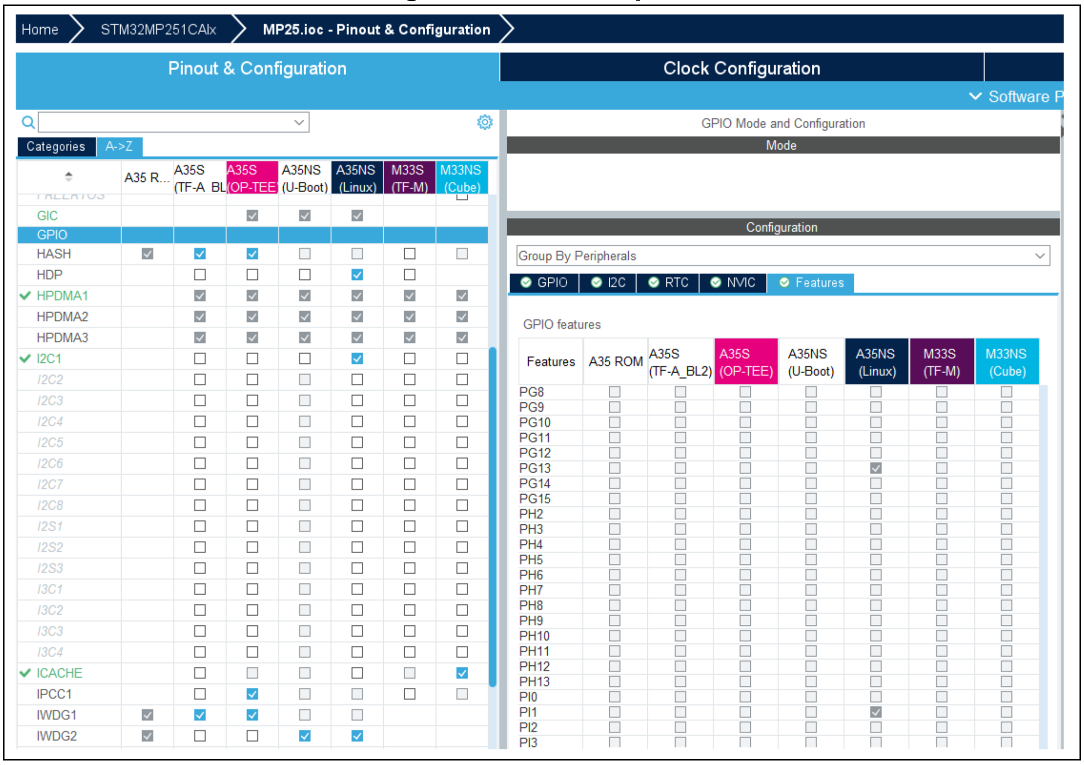


Figure 140. Inheritance in RIF GPIO panel

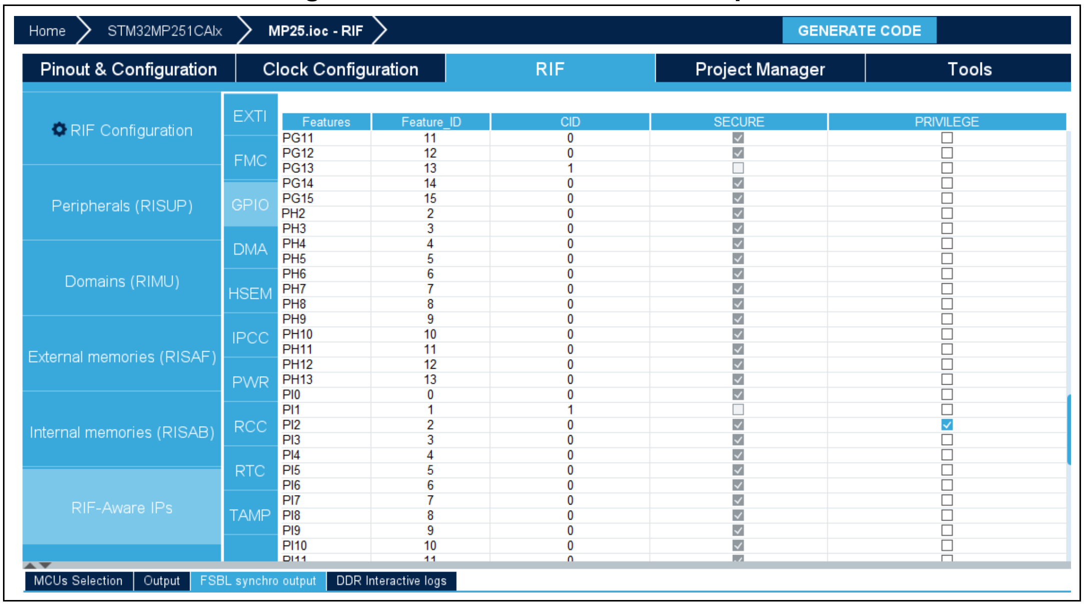
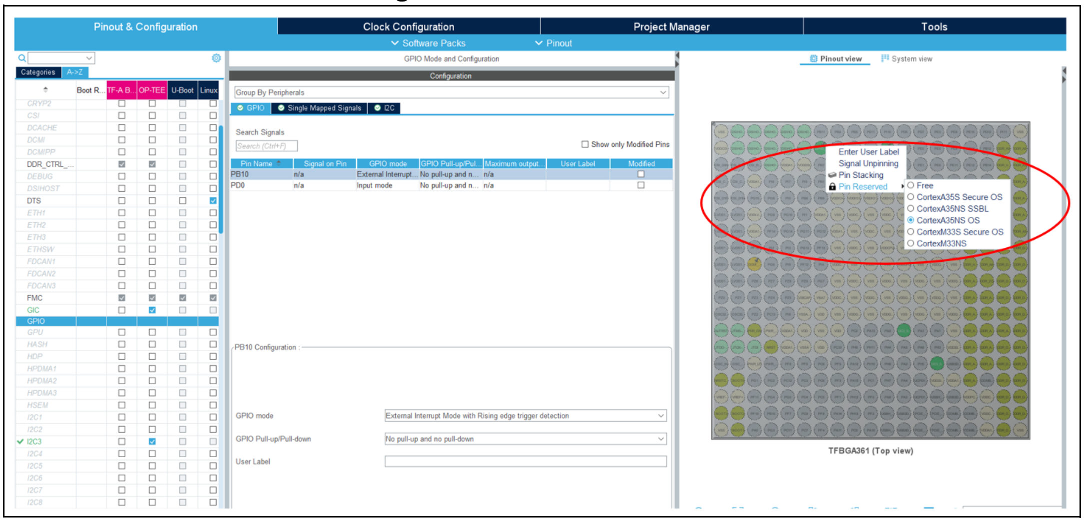


Figure 141. PIN reservation



DMA configuration

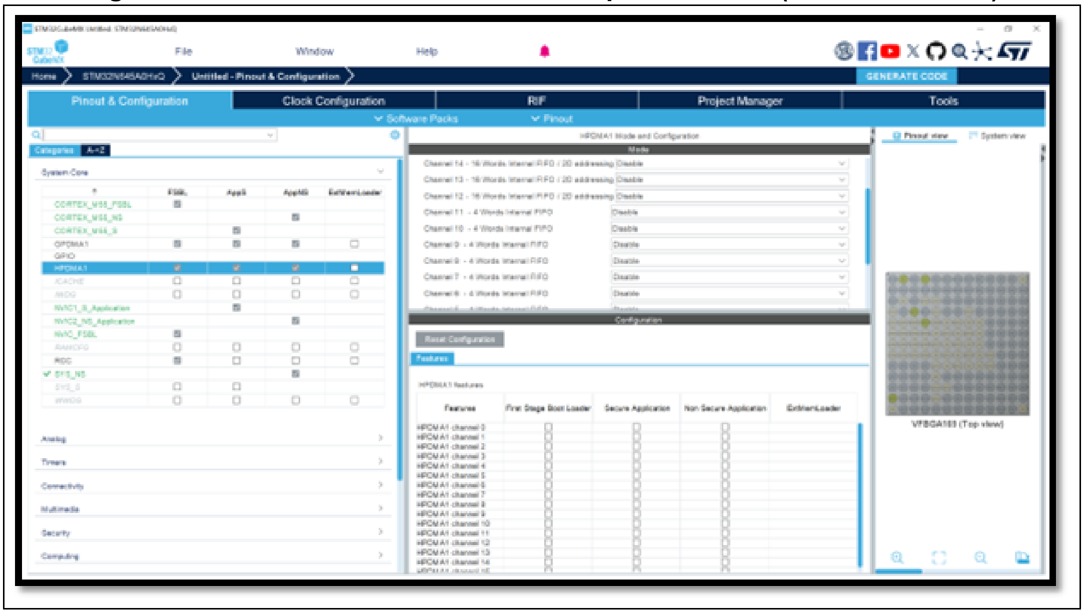
For STM32MP2 MPUs, DMA channels can be secured to prevent unauthorized access. Each channel is treated as a security feature within the DMA IP (integrated peripheral).

The approach to protecting DMA channels is similar to how IO protection is handled:

- The settings for which software contexts can use a DMA channel are determined by the peripheral that needs the DMA service. These settings are then shown in the DMA feature panel.
- The specific protection for each DMA channel is based on these settings and is displayed in the RIF-aware IP panel, under the DMA section.

For STM32N6 MCUs, the security of DMA channels is user-defined and not automatically inherited from the IPs, see *Figure 142*.

Figure 142. HPDMA1 features with RIF implementation (STM32N6 MCUs)



An example (based on STM32MP2 MPUs) is given for the I2C peripheral.

Figure 143. I2C IP panel

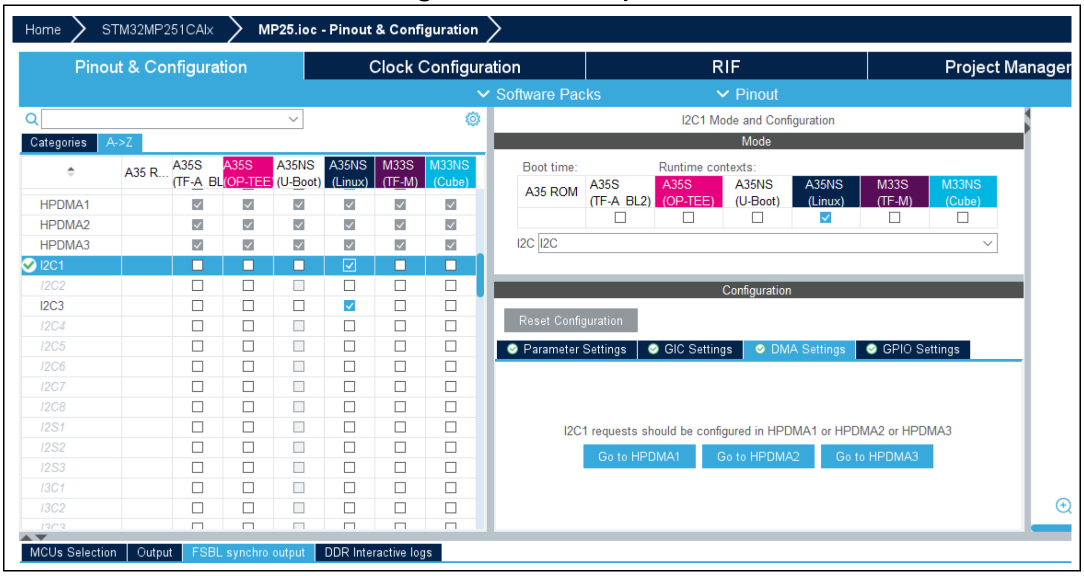


Figure 144. I2C mode panel

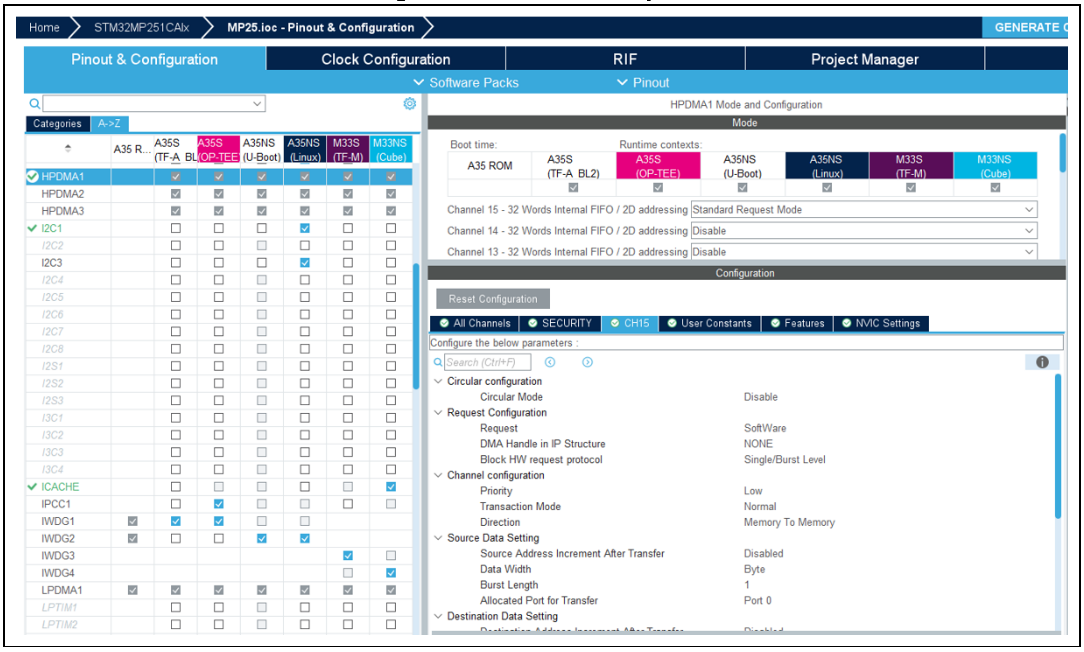


Figure 145. I2C features panel

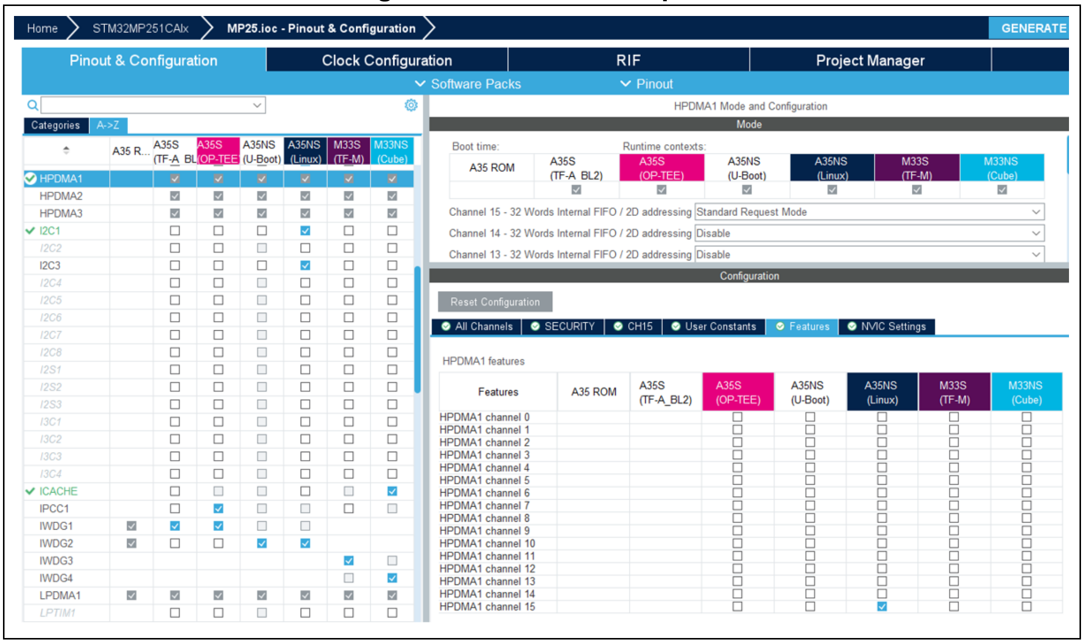
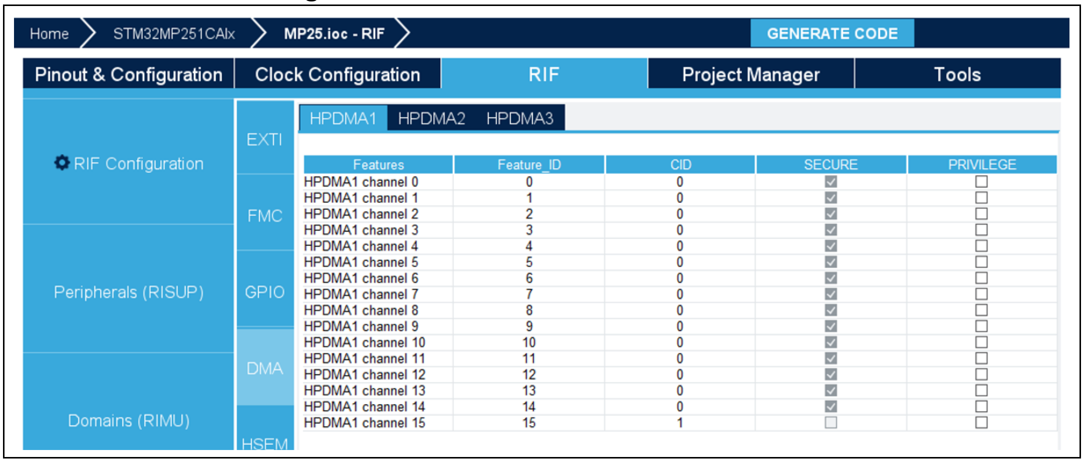


Figure 146. DMA RIF-aware IP inheritance



Clock configuration

Clock is a RIF-aware IP. Each clock is a RIF feature that can be protected thanks to the software context assignments of the feature.

The feature protection is then reported in the RIF-aware IP RCC panel.

The RCC feature assignment follows a different scheme, dependent on its type.

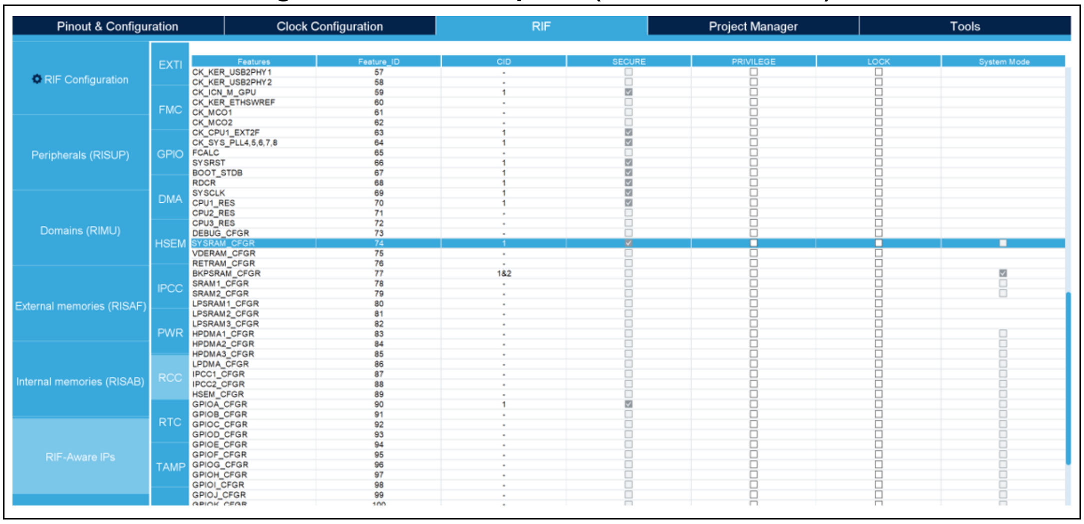
Three clocks feature types exist:

- The root clocks:
  - Their assignment is SOC family dependent.
  - On STM32MP25 devices, these features are fixedly assigned to a unique context.
- The HW resource clocks (RAM or peripherals clocks)
  - Their assignments are inherited from the HW resource assignments it clocks.
  - These clocks may be associated to an additional configuration (the System Mode) allowing to correctly protect the clock when it is shared between several CPU. This is the case for STM32MP25 devices.
- The system resource clocks:
  - These are the remaining clocks.
  - Their configuration should be done manually from the feature panel of RCC IP.

Example of the clock protection of the HW resource BKPSRAM:

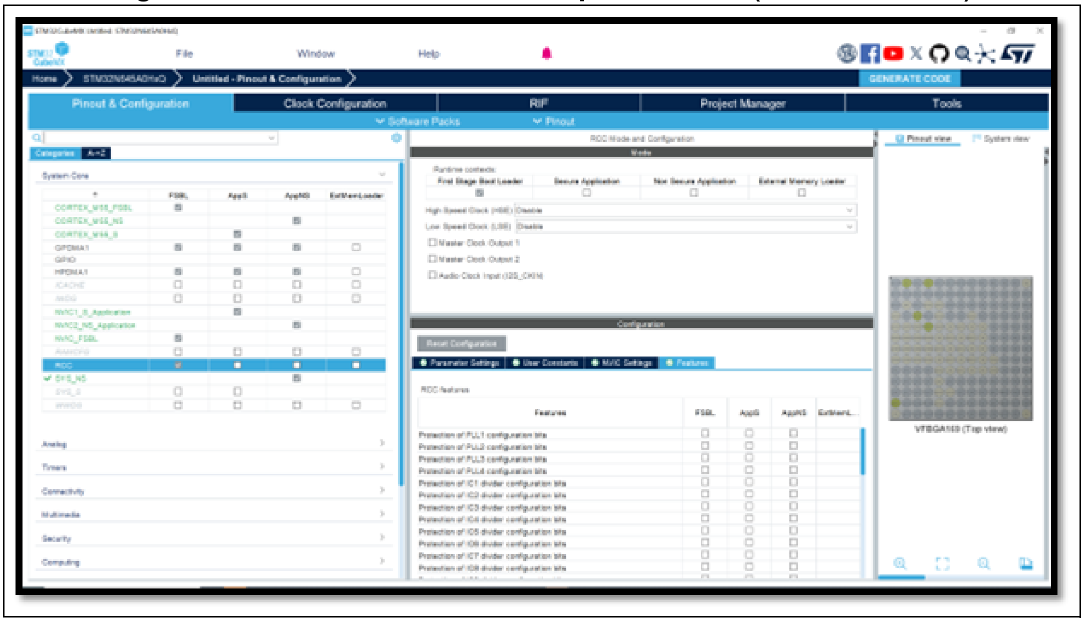
- For RCC the user can lock the features.
- Some features have a system mode, it is enabled if the feature has a CID equal to “1&2” as we can see in case of BKPSRAM_CFGR feature above (*Figure 147*).

Figure 147. RIF RCC panel (STM32MP2 MPUs)



For STM32N6 devices, features security are not automatically configured, but are defined by the user.

Figure 148. RCC features with RIF implementation (STM32N6 MCUs)



External interrupts protection

STM32CubeMX does not display a dedicated EXTI IP in the Pinout & Configuration section. However, EXTI can be secured in two ways:

1. From peripherals: the security for the interrupts is automatically taken from the peripheral that creates them. For example:
   - EXTI to wake up from peripheral: when assigning an IP, STM32CubeMX identifies and assigns the same security level and context ID to the pins connected to this IP. The security level and context ID are determined by the software context chosen, without any need to adjust settings in the Pinout View or GPIO configuration. The RIF configuration panel uses the IP software context to set the security level and context ID.
   - EXTI for PWR_WKUP: for power wake-up lines associated with a peripheral, the software context assignment is managed through the PWR configuration panel.
2. From system resources: these must be set up manually. To do this, adjust the settings in the EXTI sub-panel found within the RIF-aware IPs panel.

Any other EXTI not mentioned has a preset security configuration, which can viewed in the EXTI sub-panel of the RIF-aware IPs panel.

For STM32N6 MCUs, the security settings for EXTI are not automatically assigned but are instead defined by the user, who can set the privilege level.

Figure 149. RIF panel for EXTI1 (STM32N6 MCUs)

4.7.10 Memory protection for STM32MP2 series

The memory protection is configured through two RIF controllers:

- RISAF (resource isolation slave unit for address space protection full) acts as a firewall, allowing to define access rights for memory regions of DDR and external mapped flash memories
- RISAB (resource isolation slave unit for address space protection block-based) acts as a firewall, allowing to define access rights for memory regions of the internal SRAM.

In the next we will cover only the RISAF, but the process is the same for RISAB. The first is for managing internal memory and the second is for external memory.

RISAF configuration

RISAF is a mechanism allowing the user to configure memory access. Each memory is divided into zones. Each zone can be configured to be read-only or read/write.

The user can also specify if privileges are required, if the memory zone should be secured or encrypted.

The configuration happens at a compartment level.

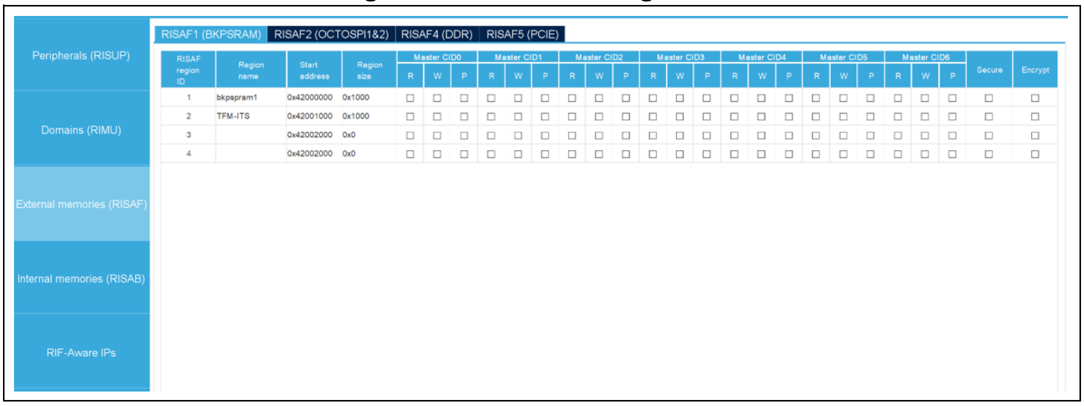
Through RISAF registers, a trusted application (or the application to which the configuration has been delegated) assigns memory regions and subregions to one or more security domains (secure, privilege, compartment). RISAF includes the DDR memory.

Through RISAF the user can:

- See the list of the different memories
- Access the memory configuration
- Configure the parameters of the memory regions (Start address, region size, Master CID, Read-Write-Privilege)
- Protect memory regions of DDR and external memories by clicking on the dedicated memory.

RISAF includes four memories, namely RISAF1 (BKPSRAM), RISAF2 (OCTOSPI 1&2), RISAF4 (DDR), and RISAF5 (PCIE).

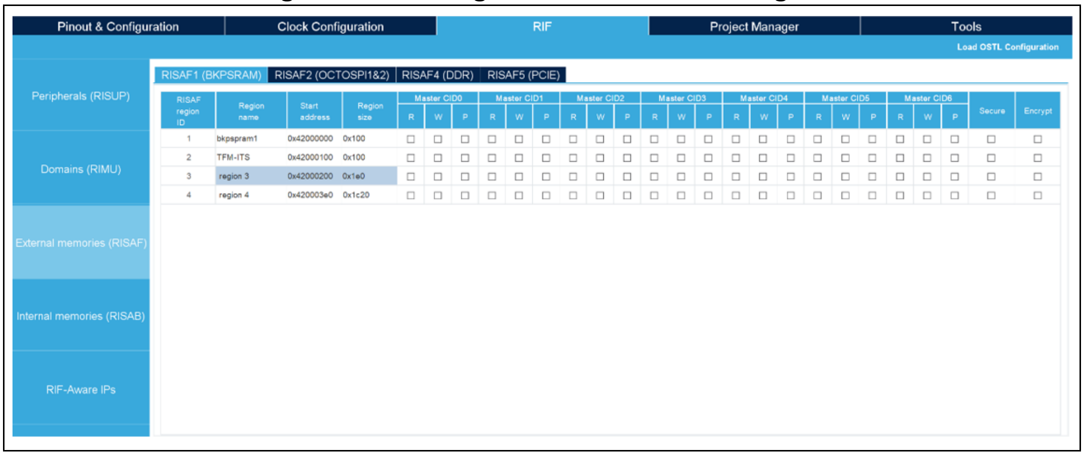
Figure 150. RISAF configuration



Each memory table contains several columns, such as region ID, region name, start address, region size in hexadecimal, seven groups for Master CID 0 to 6, secure and encrypt.

For each subregion, the user can change the region name and the region size. Each memory has its default configuration.

Figure 151. Configuration of a new subregion



RISAF1: backup static random access memory (BKPSRAM)

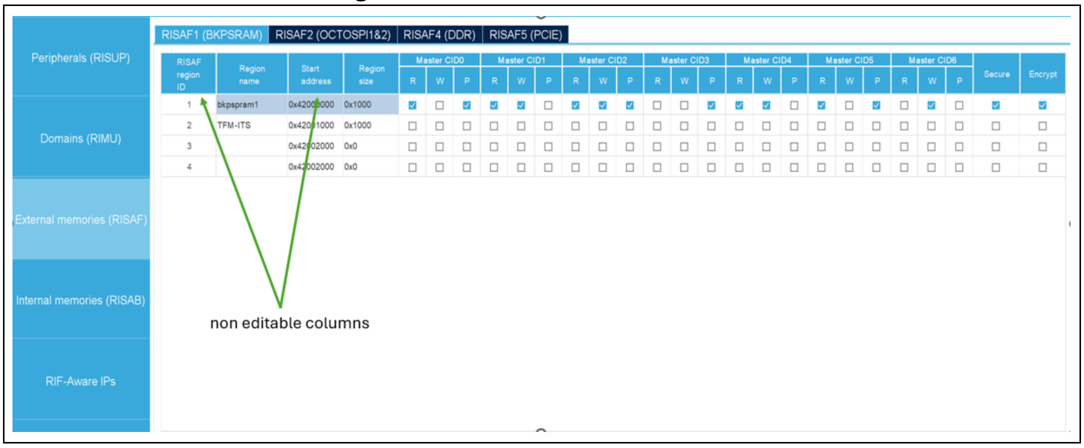
BKPSRAM is divided into four regions with id 1 to 4 by default. The memory is divided into two equal subregions. The user cannot add or remove regions.

To remove a region, the user must increase the size of another. To add a region, the user must decrease the size of another region.

Two columns are not editable: RISAF region id and start address. The user can change the name of subregion. If the name is empty or the region size is equal to 0, this subregion is not generated.

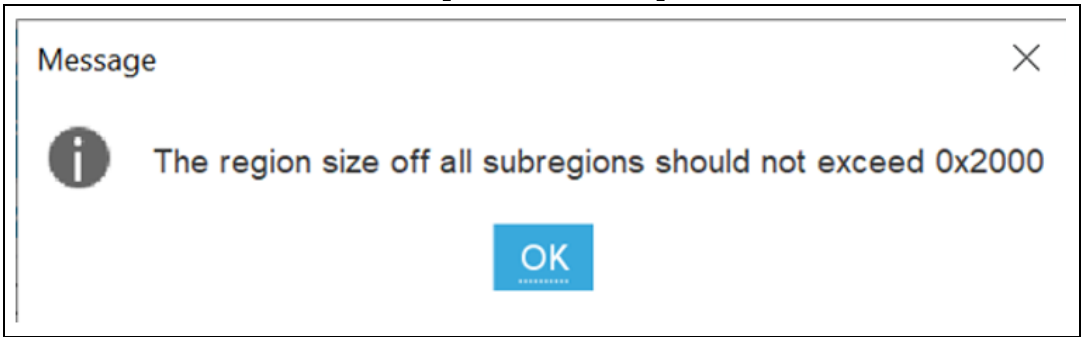
The start address and the ID column are not editable.

Figure 152. Non editable columns



The region size should not exceed the total region memory, or a warning is displayed.

Figure 153. Warning

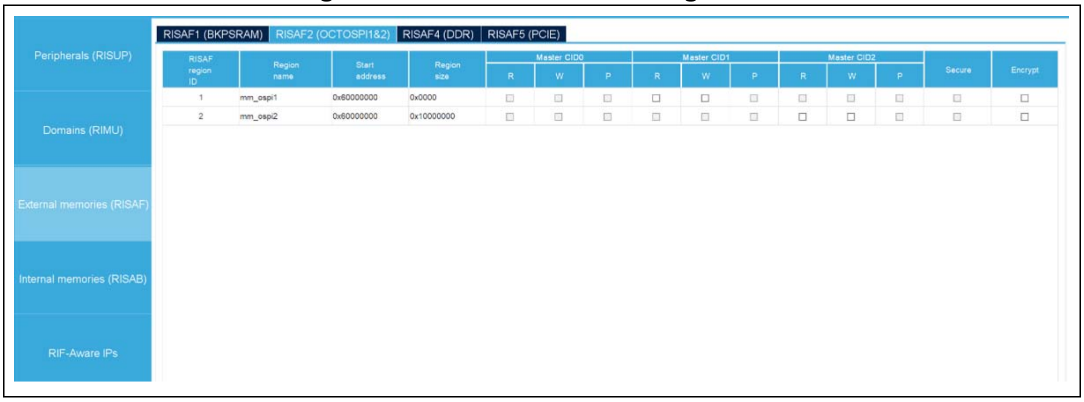


The user can assign a subregion to a master CID 0 to 6. CID 7 (not configurable in the UI) is reserved for debugging.

RISAF2: OCTOSPI1&2 memory configuration

The OCTOSPI1&2 Master CID group column is inherited from RISUP peripherals OCTOSPI1 and OCTOSPI2. By default, in OCTOSPI1&2 memory there are two subregions, mm_ospi1 and mm_ospi2.

Figure 154. OCTOSPI1&2 configuration



OCTOSPI1&2 use the Memory mapped mode: the two controllers are sharing the same 256 MB memory region.

By default, OCTOSPI2 takes the whole region. By clicking on the region size cell of mm_ospi1, a list appears, allowing the user to select the region size. Possible configurations are 0/256, 64/192, 128/128, 192/64, and 256/0 MB (see *Figure 155*). The start address changes automatically.

Figure 155. OCTOSPI1&2 memory mapping

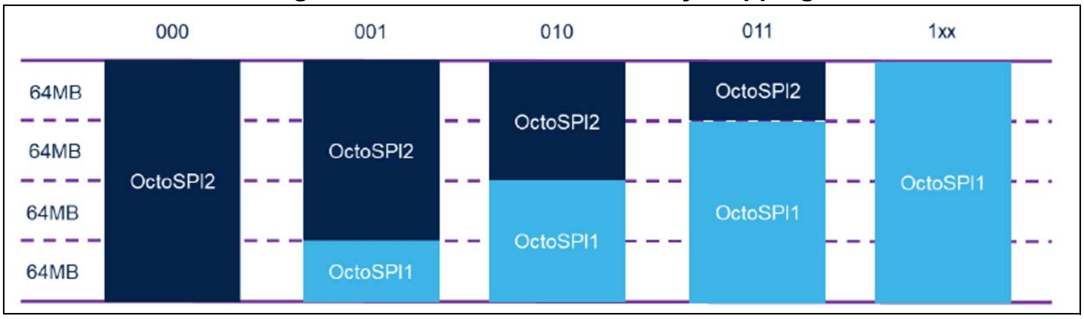
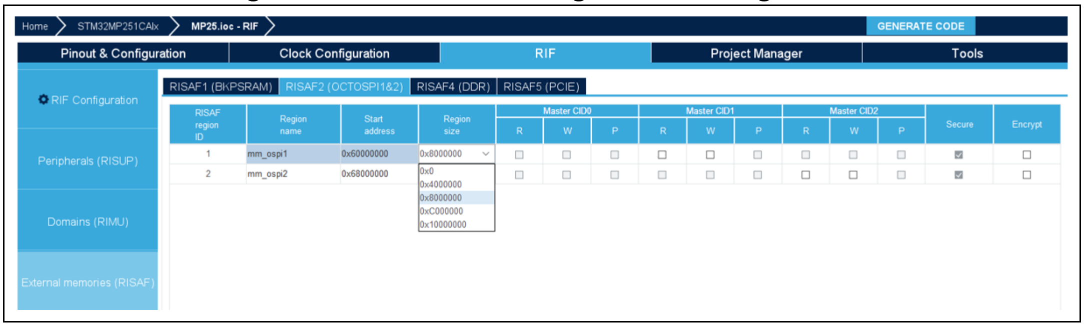


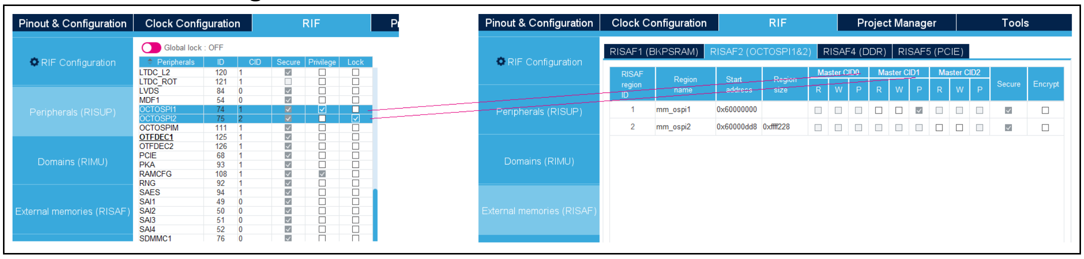
Figure 156. OCTOSPI1&2 region size configuration



The master CID column group read/write/privilege are inherited from the RISUP table.

If the OCTOSPI1 peripheral in RISUP is assigned to CID1, Master CID group column 1 is accessible, and the other CIDs are grayed out. If it is privileged in RISUP, it is privileged in Master CID1 privilege column, as shown in *Figure 157*.

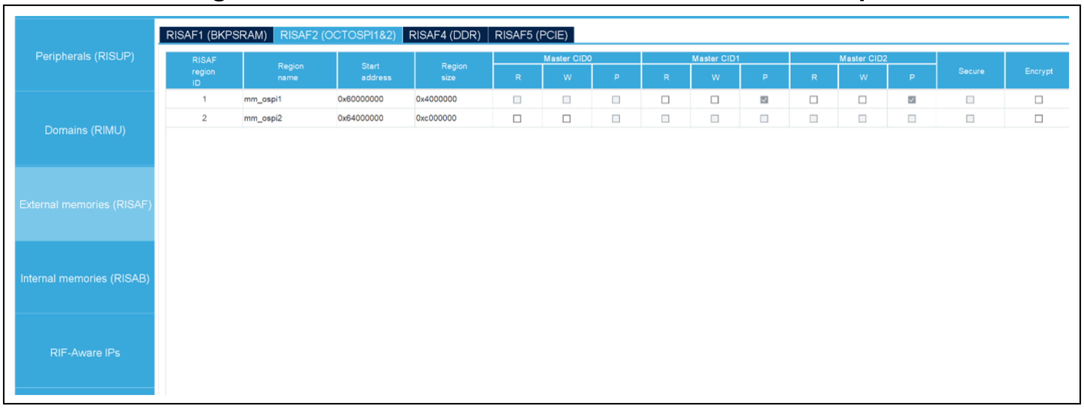
Figure 157. OCTOSPI1&2 inheritances from RISUP



If OCTOSPI1 is secure in RISUP, it is secure and grayed out in RISAF2. The checkboxes inherit their values from RISUP. Changes to the secure or to the privilege state must be performed in the RISUP table.

If OCTOSPI1 is CID1&2 in RISUP, the two Master CID 1 and CID 2 are activated in RISAF2.

Figure 158. OCTOSPI1&2 Master CID activation example

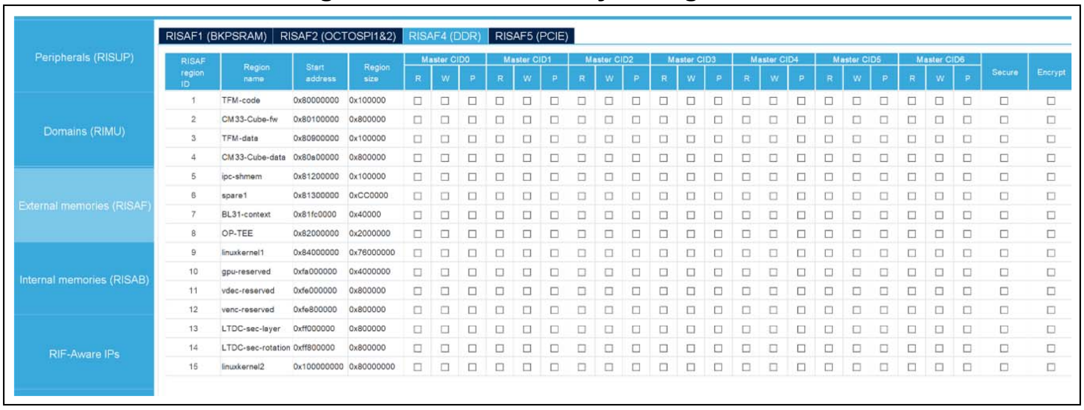


OCTOSPI2 is not assigned to any CID in RISUP, so the Master CID 0 is activated by default.

RISAF4: DDR memory configuration

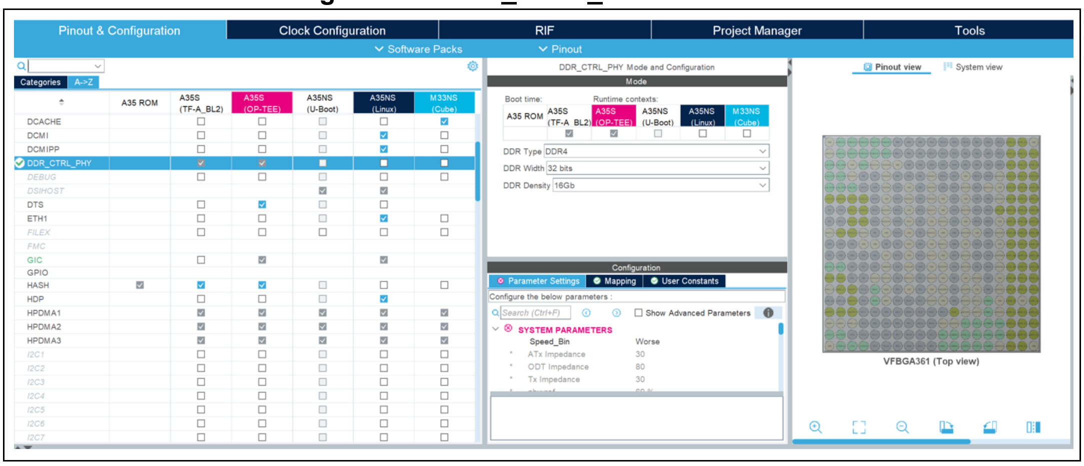
By default, the DDR is configured to handle 4 Gbytes of RAM divided into 15 subregions.

Figure 159. DDR memory configuration



To change the memory size, go to the Pinout & Configuration tab, select the DDR_CTRL_PHY, and choose the desired memory size.

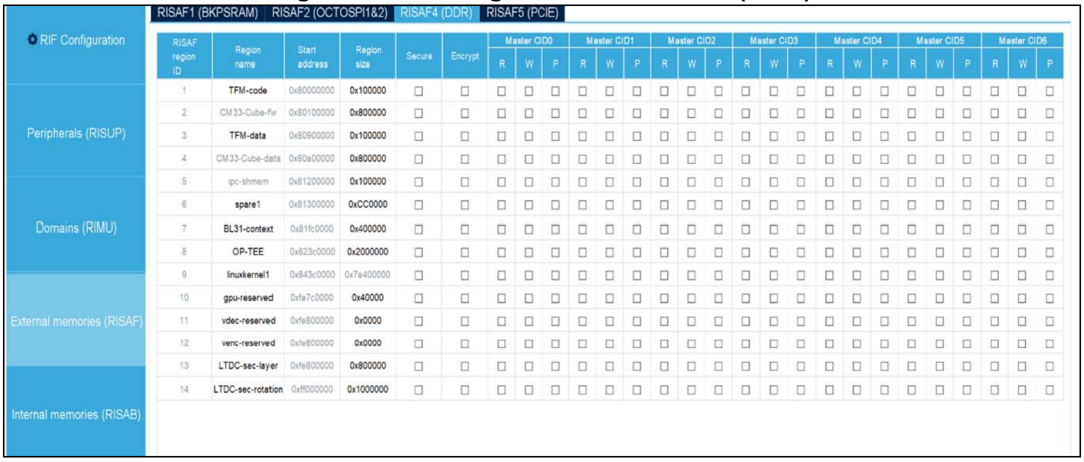
Figure 160. DDR_CTRL_PHY activation



When returning to RISAF4 (DDR) panel, a new configuration of the DDR memory appears in table:

- The region sizes can be one of the following values: 521 Mbytes, 1 Gbyte, or 2 Gbytes, depending on the user's choices.
- The number of regions decreases from 15 to 14.
- If the user decreases the size of a region, the decreased value is added to the size of the linuxkernel1.
- There is no empty region in the new implementation.
- For the DDR 4 GBytes, if the user decreases the size of a region, the decreased value is added to the size of the linuxkernel2.

Figure 161. Configuration of RISAF4 (DDR)

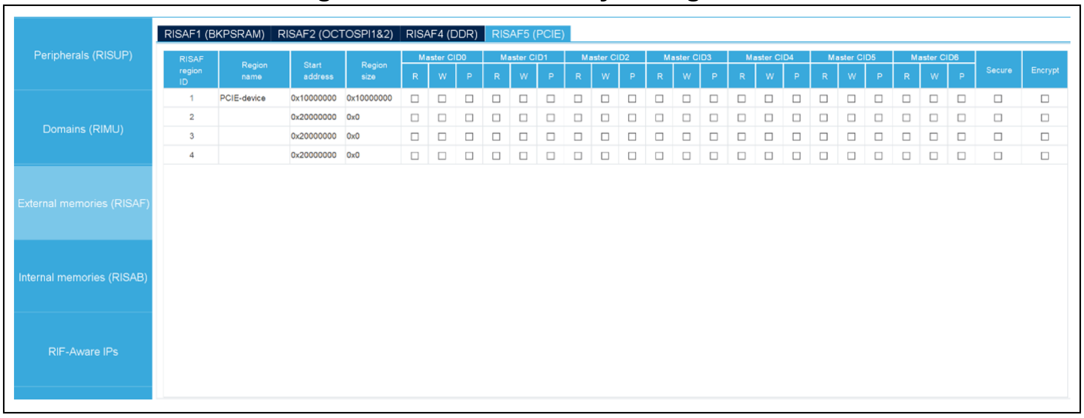


As BKPSRAM memory, the user can assign a subregion to a master CID 0 to 6, set the region as read / write, privilege, secure and encrypt.

RISAF 5: PCIE memory configuration

The PCIE memory is similar to BKPSRAM, except by default it has one subregion that takes the whole memory region size, and the user can add maximum three other regions, set the name, the read/write access rights, the privilege, secure and encrypt.

Figure 162. PCIE memory configuration



For STM32MP21 products:

- The scope of RISAF2 table scope is limited to OCTOSPI1, the unique OCTOSPI instance supported. Additionally, there is no RISAF5 (PCIE) table on these devices.
- When the cortex M33 is chosen, there is more than one memory region.

Figure 163. RISAF table for STM32MP21 products

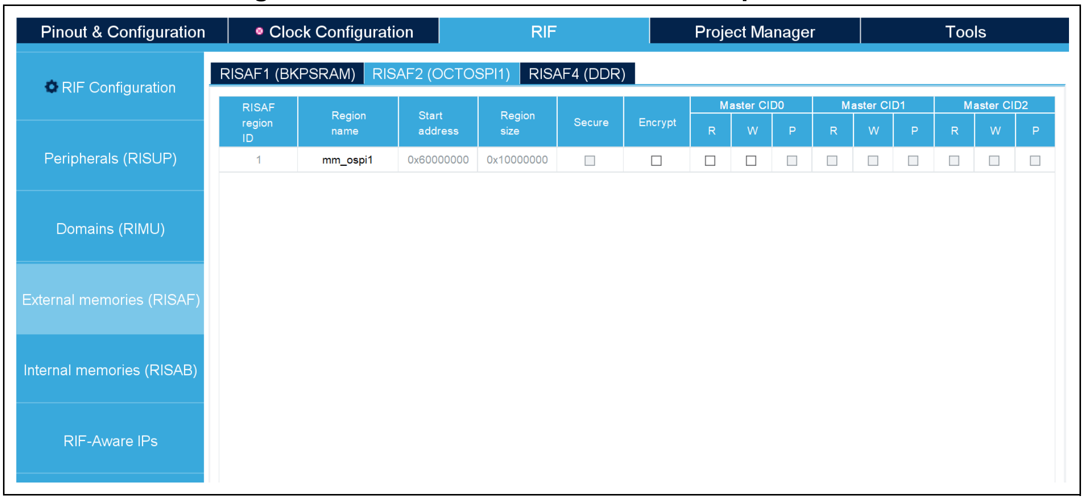


Figure 164. RISAF table for STM32MP2x devices in the Cortex M33 master

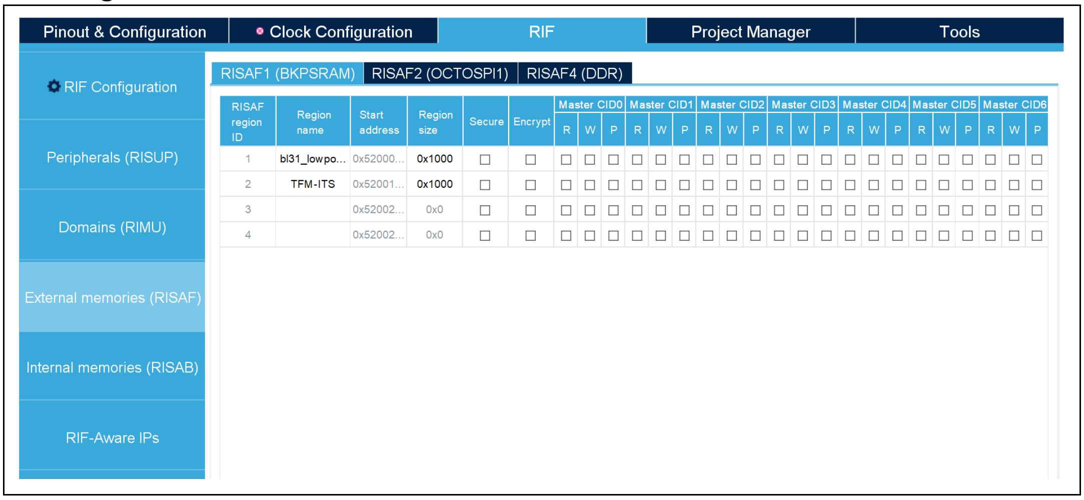
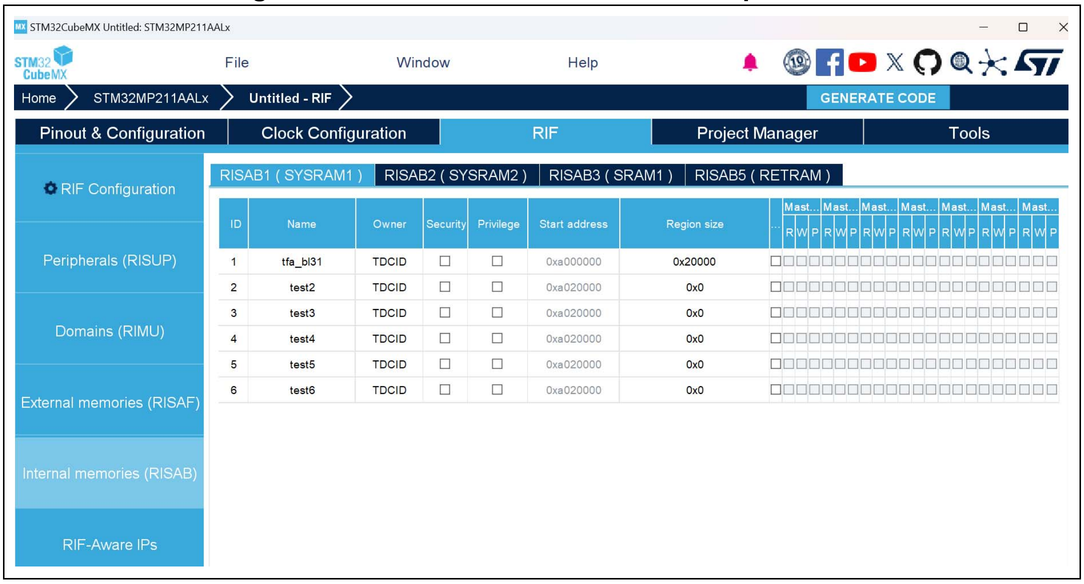


Figure 165. RISAB table for STM32MP21 products



Default memory protection

When starting a new project using the RISAB / RISAF configuration panels, there is a default memory protection scheme already in place. This default setup includes predefined region names, start addresses, and sizes that correspond to a memory map designed for the ST software architecture user is targeting.

The configuration can be modified, even if some areas are reserved. The user can modify this default memory map to suit the needs of a new application. For instance, when working with the STM32MP25 hardware, the OpenSTLinux software configuration for the 4-GByte DDR memory is always pre-loaded as the default setting. However, there are certain regions, specifically those related to the Cortex-M33NS, where the names of the memory regions are fixed and cannot be altered.

Memory mapping generation (MPUs only)

This section discusses the generation of memory mappings for an MPU designed for use with the OpenSTLinux architecture, and supporting the RIF. Note that this MPU does not support the Memory Management Tool.

STM32CubeMX utilizes the RISAF/RISAB configuration as a basis for generating the memory mapping. This mapping is specifically for the master secured firmware associated with the MPU.

The memory mapping created by STM32CubeMX includes only the base-memory regions. These are the regions predefined in the RISAF/RISAB configuration panels.

If an application requires additional memory sub-regions beyond the base-memory regions, these must be defined manually. The definitions go into the User Sections within the initialization code of the firmware that necessitates these extra sub-regions.

For the STM32MP25 hardware, when it is set to A35TD boot mode, a specific memory mapping is produced for the OP-TEE firmware.

This mapping is saved in a file named <project name>-mx-resmem.dtsi, where <project name> is a placeholder for the actual name of the project.

4.7.11 Memory protection for STM32N6 series

The memory protection is configured through one RIF controller. RISAF (resource isolation slave unit for address space protection full) acts as a firewall, allowing to define access rights for memory regions of DDR and external mapped flash memories

RISAF configuration

RISAF is a mechanism allowing the user to configure memory access. Each memory is divided into zones. Each zone can be configured to be read-only or read/write.

The user can also specify if privileges are required, if the memory zone should be secured or encrypted.

The configuration happens at a compartment level.

Through RISAF registers, a trusted application (or the application to which the configuration has been delegated) assigns memory regions and subregions to one or more security domains (secure, privilege, compartment). RISAF includes the DDR memory.

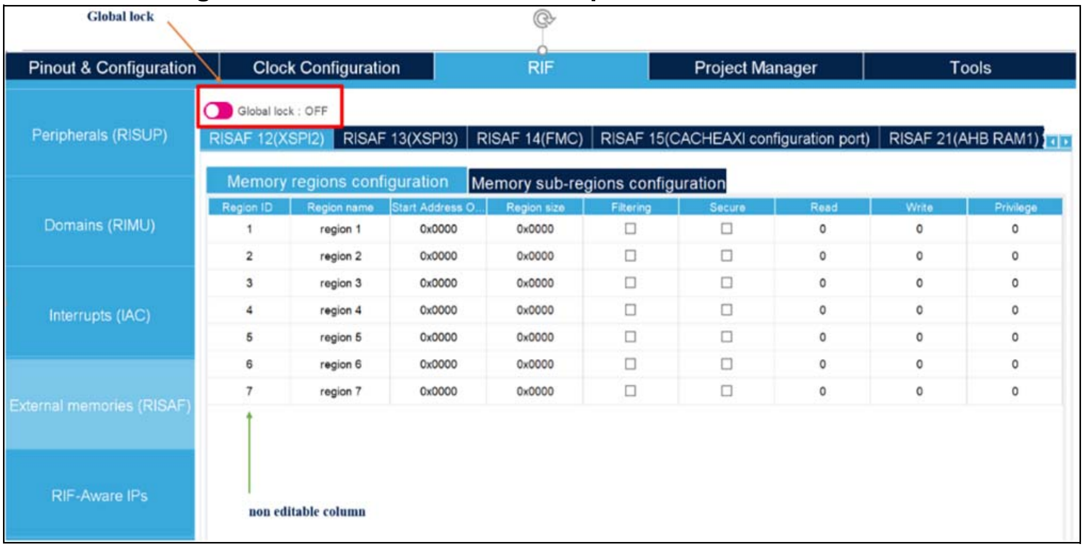
Through RISAF the user can:

- See the list of the different memories
- Access the memory configuration
- Configure the parameters of the memory regions (Start address, region size, Master CID, Read-Write-Privilege)
- Protect memory regions of DDR and external memories by clicking on the dedicated memory.

Configure memory access with RISAF for STM32N6 MCUs

The STM32N6 RISAF panel has a global lock, unlike the STM32MP2.

Figure 166. Global lock in RISAF panel for STM32N6 MCUs



The RISAF is divided into subcontrollers for 17 memory zones, assigned to security domains through the RISAF subcontrollers listed below:

- RISAF1 (TCM)
- RISAF2 (CPU AXI RAM0)

- RISAF3 (CPU AXI RAM1)
- RISAF4 (NPU master 0)
- RISAF5 (NPU master 1)
- RISAF6 (CPU master)
- RISAF7 (FLEXRAM)
- RISAF8 (CACHE AXI RAM)
- RISAF9 (VENCRAM)
- RISAF11 (XSPI1)
- RISAF12 (XSPI2)
- RISAF13 (XSPI3)
- RISAF14 (FMC)
- RISAF15 (CACHEAXI configuration port)
- RISAF21 (AHB RAM1)
- RISAF22 (AHB RAM2)
- RISAF23 (Backup RAM)

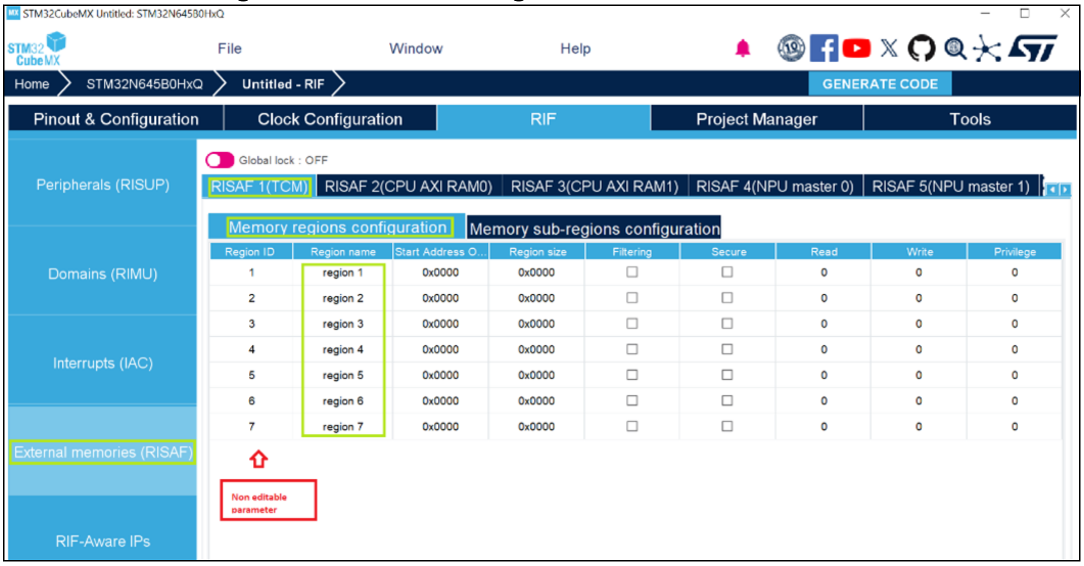
Each RISAF subcontroller manages a specific number of memory regions. By default, it controls seven regions, but some subcontrollers manage 11 regions, while others handle only two regions.

In the user interface Each RISAF sub controller is represented by 2 tables: Memory regions configuration and Memory sub-regions configuration.

Memory regions configuration table contains the following columns:

- Region ID: this column is non editable.
- Region name
- Start Address Offset and Region size: this value must be within a specific range for each region. If the user sets an incorrect value, a popup appears indicating that the value is out of range.
- Filtering
- Secure
- Read, Write, and Privilege: values range from 0 to 255.

Figure 167. RISAF configuration for STM32N6 series



RISAF also covers the configuration of the memory sub-regions in the Memory sub-regions configuration table. Each regions have two dedicated sub-regions (see *Figure 168*).

Subregions are not accessible by default (grayed). To activate a given subregion, activate the related filtering parameter in the Memory regions configuration table (see *Figure 169*).

Memory sub-regions configuration table contains the following columns:

- RISAF region ID
- SubRegion name
- Start Address Offset and Region size: this value must be within a specific range for each region. If the user sets an incorrect value, a popup appears indicating that the value is out of range.
- SubRegion CID: values range from 1 to 7.
- Filtering
- Delegated CID
- Delegation enabled
- Read
- Write
- Secure
- Privilege
- Lock

Figure 168. Sub-regions activation in RISAF (showing activated subregions)

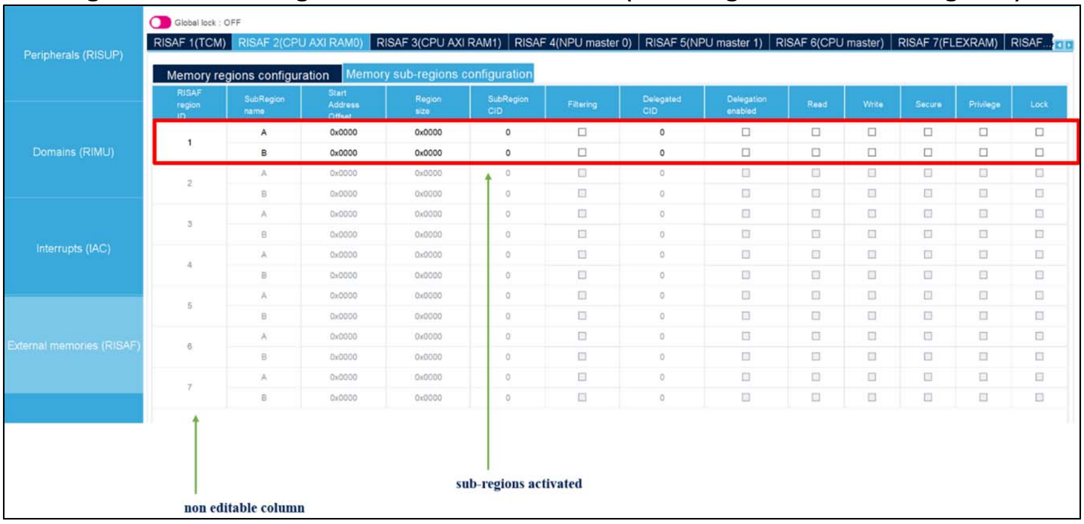
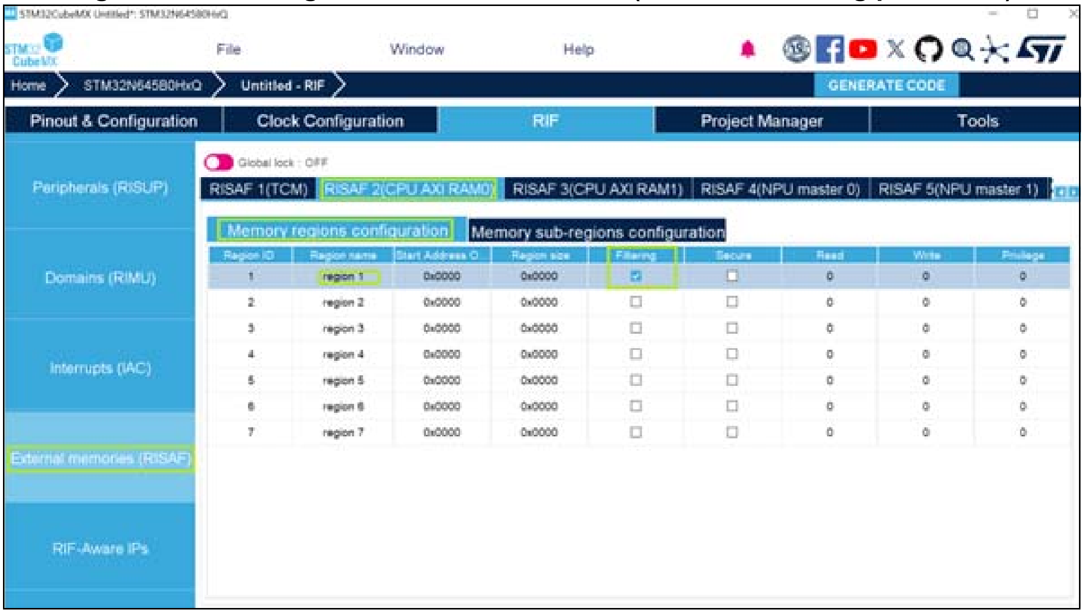


Figure 169. Sub-regions activation in RISAF (check the filtering parameter)



4.7.12 RIF code generation

The RIF configuration code generation is handled by the STM32CubeMX, which incorporates it into the initialization code of the project. The format of the generated code depends upon the type of driver used to manage the RIF. The options include HAL (Hardware Abstraction Layer) code for the Cube driver and dts-v1 (Device Tree Source version 1) code for the OpenSTLinux driver.

Note: Only dts-v1 code generation is supported.

In the context of the STM32MPU OpenSTLinux (OSTL), RIF configuration code is generated using the dts-v1 format.

The code generation adheres to the generic principles are outlined in *Section 9*.

The generated code is placed in a file named <project name>-mx-rif.dtsi, which is part of the master Secured firmware. Additionally, code relevant to the First Stage Bootloader (FSBL) firmware is generated in a file named <project name>-mx-fw-config.dts.

The specific syntax and semantic rules for the generated code are detailed in the RIF binding file. For more information, refer to the STM32MPU Wiki portal.

The next section details examples, including user interface screenshots and the corresponding generated code snippets. The procedure is straightforward: configure the RIF panels, click the GENERATE CODE button, and the code is produced in two files, with the RIF configuration landing in a file named <project name>-mx-rif.dtsi.

Figure 170. Example: RISUP configuration and generated code

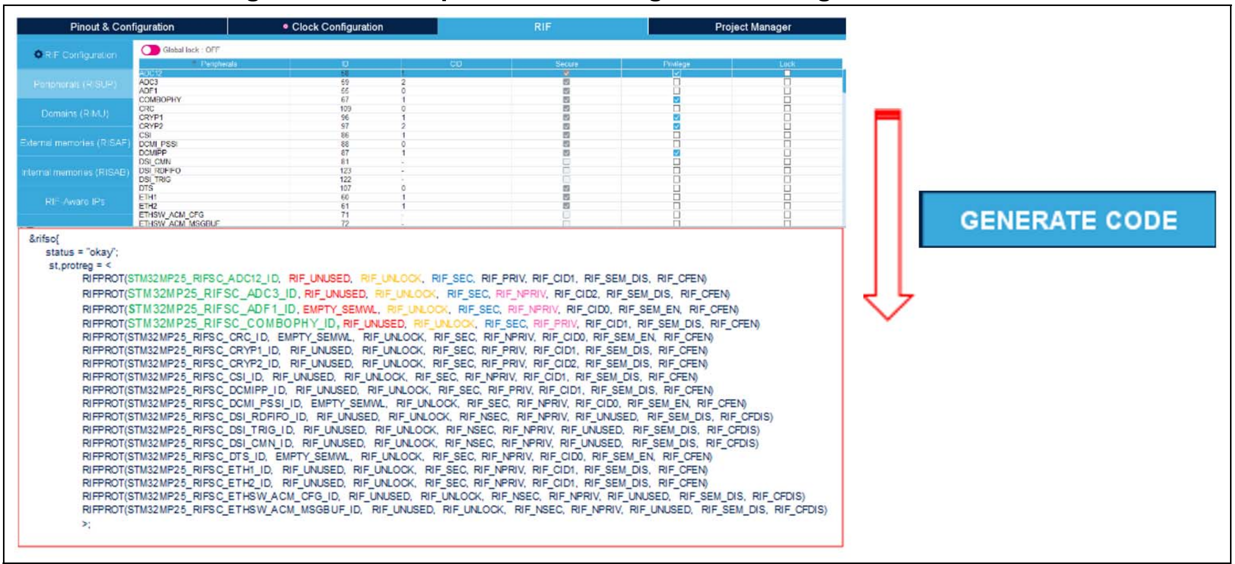
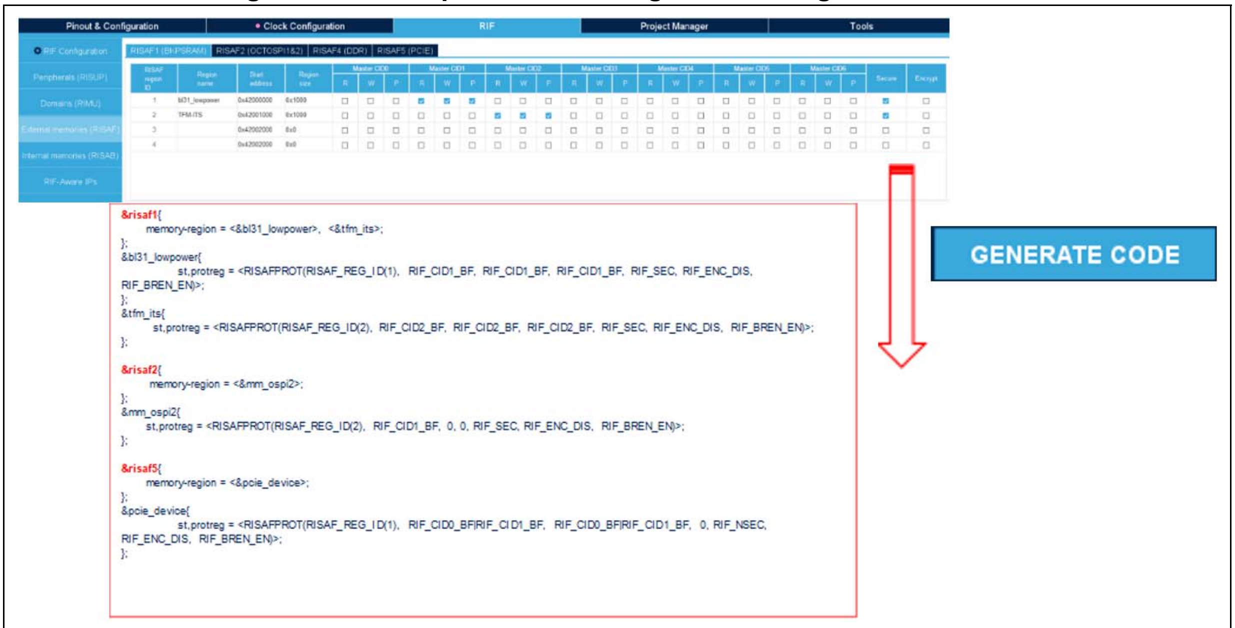


Figure 171. Example: RISAF configuration and generated code



Additionally, as described in *Memory mapping generation (MPUs only)*, a partial memory mapping is generated.

4.7.13 Implementation of illegal access controller (IAC) feature on STM32N6 series

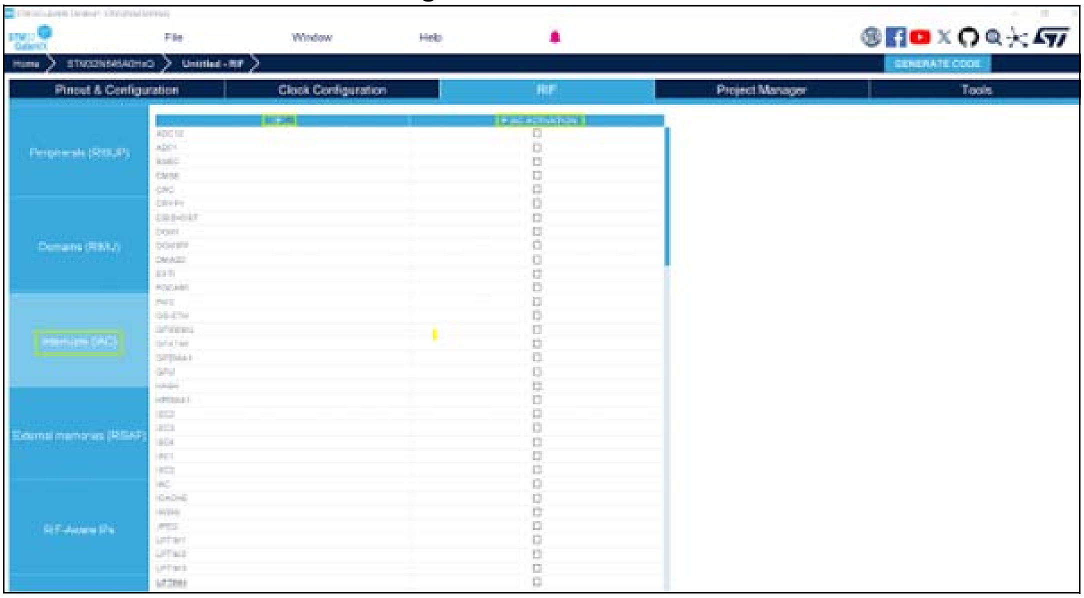
The STM32N6 MCUs support a new feature, the illegal access controller (IAC).

The IAC manages all the interrupts, and the RIF utilizes the IAC feature to centralize the detection of any illegal access related to the RIF, which is managed by a secure application.

The Interrupts (IAC) panel is composed of two columns:

- IP: The name of the IP.
- IP IAC Activation: Enables the user to activate or deactivate the interrupt related to a given IP.

Figure 172. IAC feature



4.8 Pinout & Configuration view for STM32H7 dual-core products

Some STM32H7 products come with an Arm Cortex-M7 core, an Arm Cortex-M4 core, and three power domains.

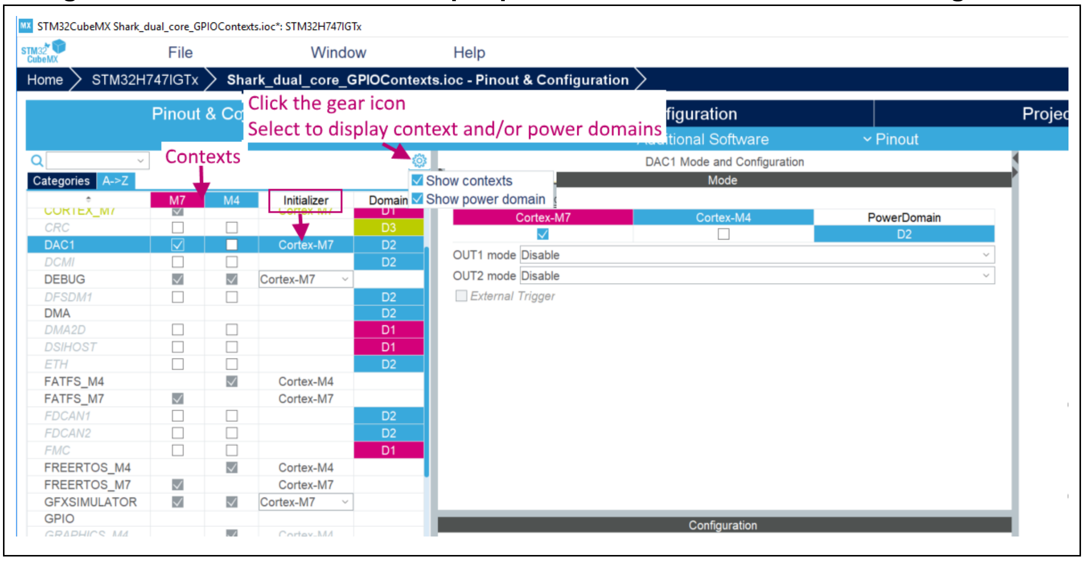
For such products, the **Pinout & Configuration** view allows the user to:

- For each peripheral and middleware: assign it to one core context or both, whenever possible. in case both contexts are selected, assign an “initializer” core to indicate on which core the peripheral or middleware initialization function shall be called.
- For each peripheral: view the power domain it belongs to.
- For GPIOs: assign it to a core or leave it free for other components that may require it. In this last case the GPIO initialization are performed on the same core as the component reserving it (code is generated accordingly).

For peripherals and middleware, these possibilities are offered in two different panels:

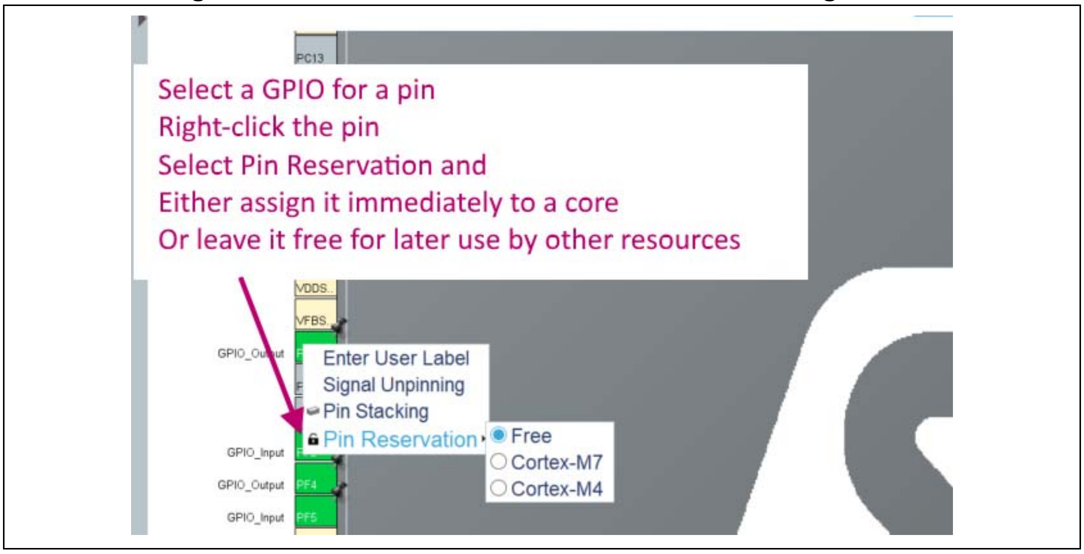
1. From the component tree panel, which lists all supported peripherals and middleware (clicking the gear icon enables the “Show contexts” option), see *Figure 173*
2. From each component mode panel, opened by clicking the component name.

Figure 173. STM32H7 dual-core: peripheral and middleware context assignment



For GPIOs (see *Figure 174*), assignment is done through the **Pinout** view directly or later and automatically through its selection in the platform settings panel of a middleware.

Figure 174. STM32H7 dual-core: GPIOs context assignment



4.9 Enabling security in Pinout & Configuration view (STM32L5 and STM32U5 series only)

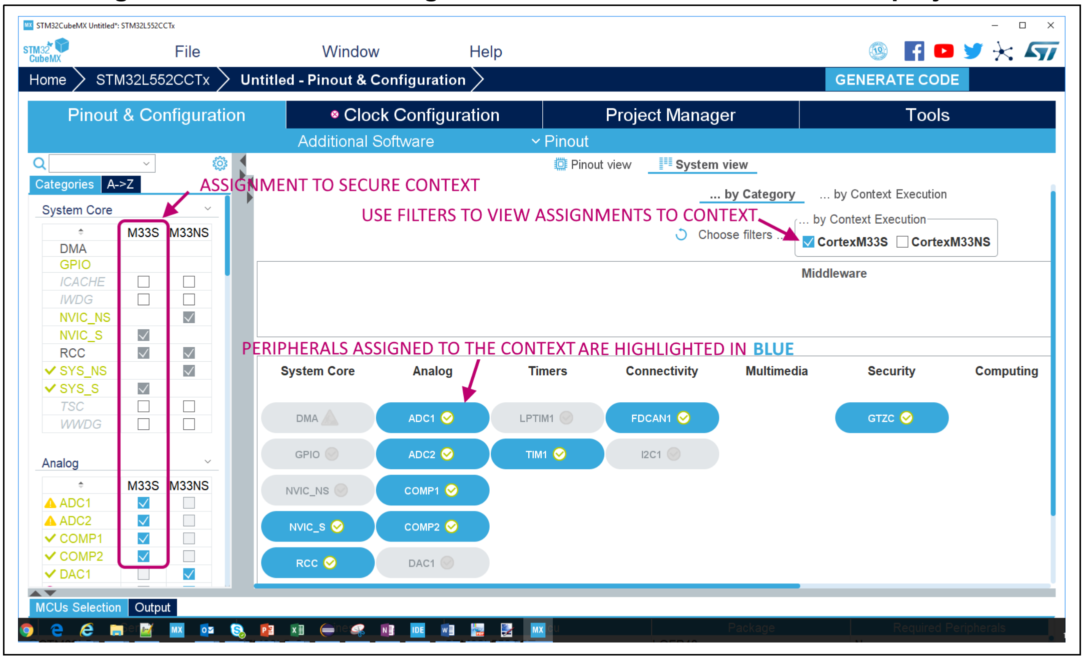
The STM32L5 MCU series harnesses the security features of the Arm Cortex-M33 processor and its TrustZone for Armv8-M combined with ST security implementation.

STM32L5 MCUs support

- two levels of privilege
  - unprivileged: software has limited access to system resources
  - privileged: software has full access to system resources, subject to security restrictions
- two security states, Secure and Nonsecure: TrustZone security is activated when the TZEN option bit is set in the FLASH_OPTR register. Security states are orthogonal to mode and privilege, therefore, each security state supports execution in both modes and both levels of privilege.

In STM32CubeMX the choice to activate TrustZone is made at project creation (see *Section 4.2*). When TrustZone is enabled, STM32CubeMX Pinout & Configuration view is adjusted accordingly, with a split between secure (M33S) and nonsecure context (M33NS), and more security-related configuration options (see *Figure 175*).

Figure 175. Pinout & Configuration view for TrustZone-enabled projects



4.9.1 Privilege access for peripherals, GPIO EXTIs and DMA requests

Independently of TrustZone, STM32CubeMX enables privilege access:

- for each peripheral: in the GTZC configuration panel (see *Section 4.9.5*), as shown in *Figure 176*
- for each GPIO EXTI: in the GPIO configuration panel, as shown in *Figure 177*
- for each DMA channel: in the DMA configuration panel (see *Section 4.9.4*), as shown in *Figure 178*.

Note: When TrustZone is active, either all or none of the RCC registers can be put in privilege mode. In STM32CubeMX, this is done by selecting “Privileged-only attribute” check box from RCC mode panel (see Figure 179). In privilege mode, all RCC registers configuration are reserved for the privilege application through the PWR_CR_PRIVEN bit, which is secured when Trustzone is activated.

Figure 176. Setting privileges for peripherals

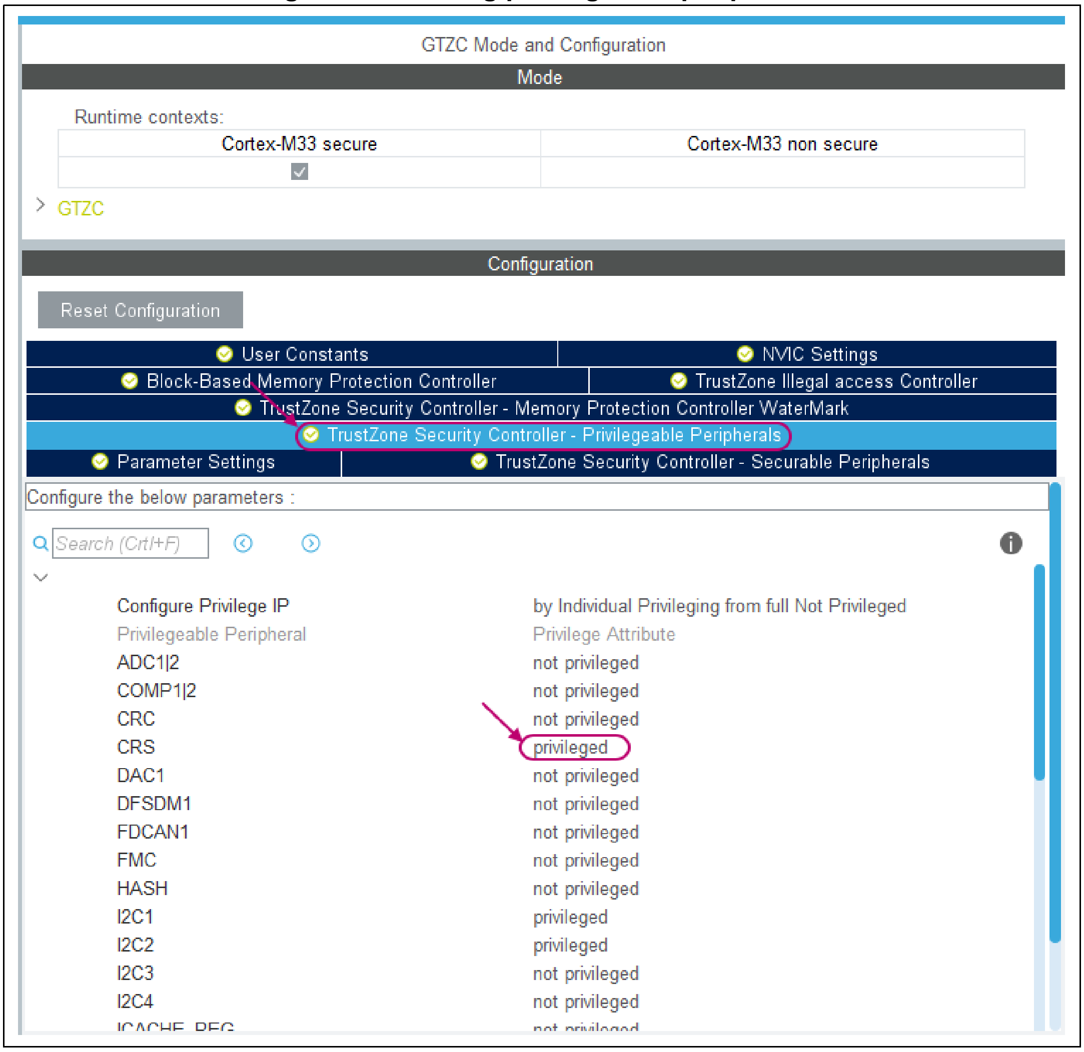


Figure 177. Setting privileges for GPIO EXTIs

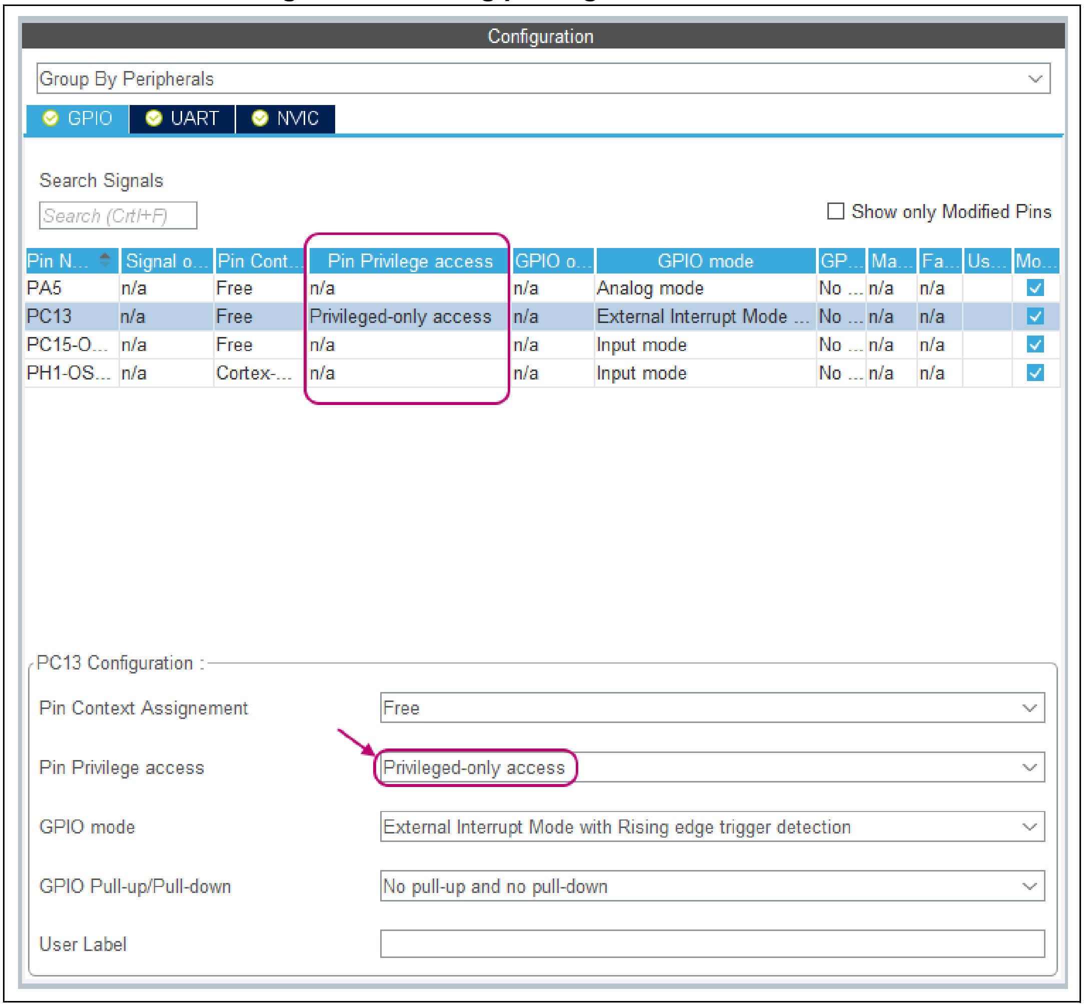


Figure 178. Configuring security and privilege of DMA requests

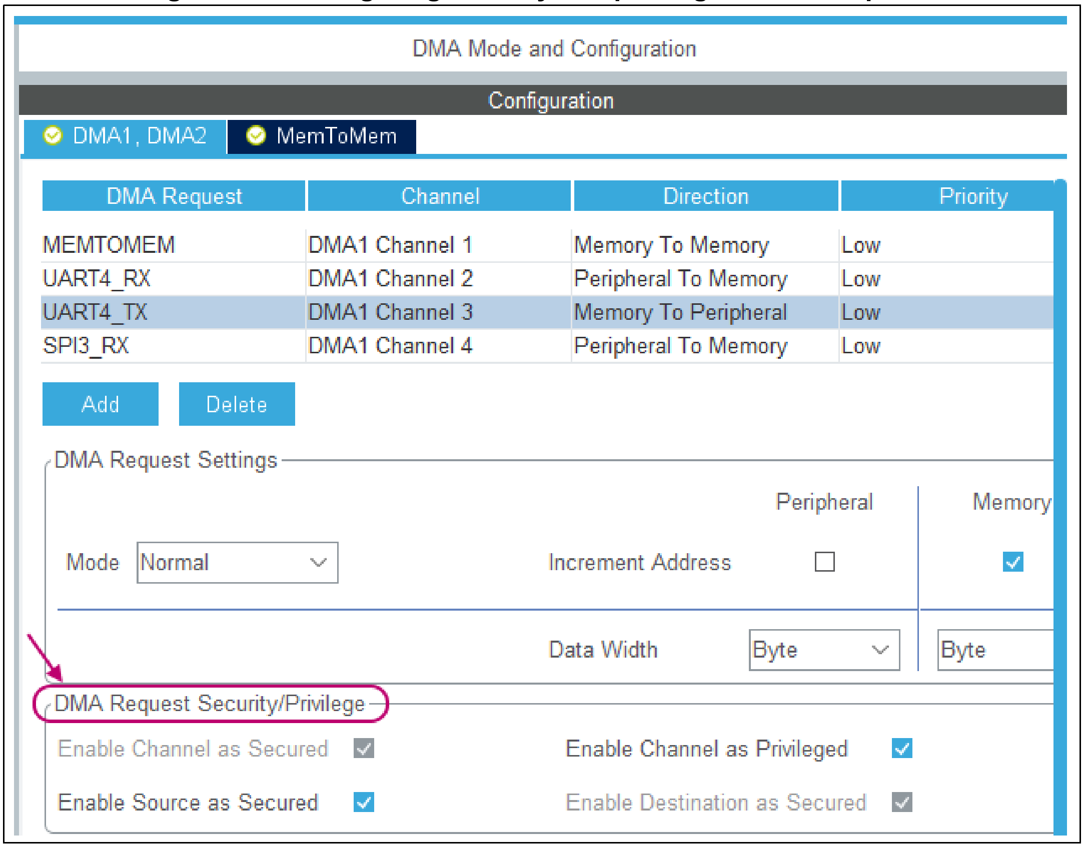
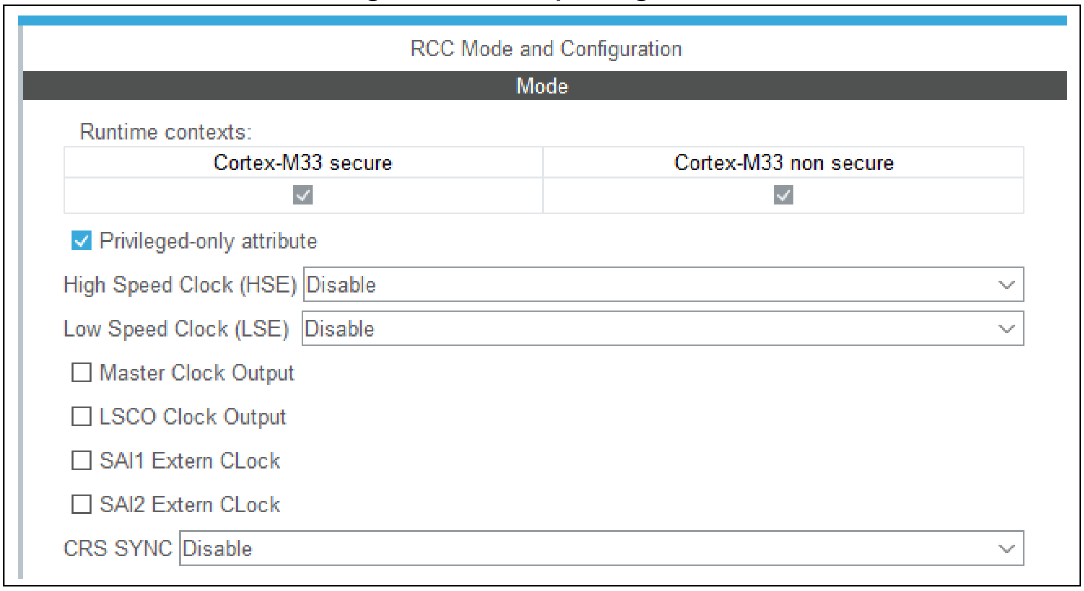


Figure 179. RCC privilege mode



4.9.2 Secure/nonsecure context assignment for GPIO/peripherals/middleware

STM32CubeMX allows the user

- to assign each peripheral and middleware to one of the contexts
- to assign a GPIO input or output to one of the context or to leave it free for other components that may require it. In this last case the GPIO assignment is in the same context as the component reserving it. By default all IOs are secured.

The assignment is done in different panels:

- For peripherals and middleware only: from the component tree panel when “Show contexts” option is enabled (clicking the gear icon) or from the mode panel.
- For peripherals only: from the GTZC configuration panel (peripherals only).
- For GPIOs only: from the configuration panel or from the Pinout view, through a right-click on the GPIO pin and by selecting “Pin Reservation”.
- For DMA requests: from the DMA configuration panel.

Note: RCC resources can be secured through the Clock configuration view (see Section 4.10.2).

Note: For middleware requiring a peripheral the middleware can only be assigned to the context the peripheral is already assigned to.

4.9.3 NVIC and context assignment for peripherals interrupts

When TrustZone is enabled, the interrupt controller is split into NVIC_NS for the nonsecure context and NVIC_S for the secure context. Two SysTick instances are available as well, one for each context: they are visible, respectively, under SYS_NS and SYS_S.

By default, all interrupts are secured.

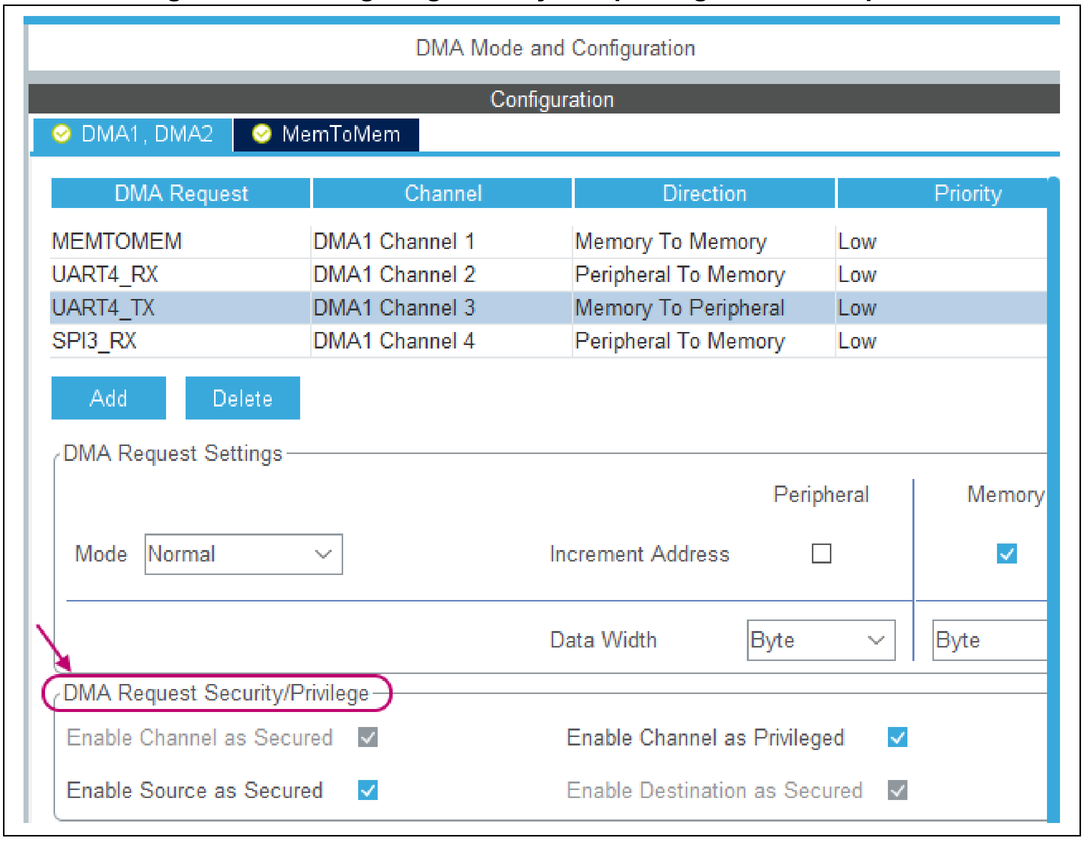
Peripherals interrupts are automatically assigned to the interrupt controller relevant to the context:

- For peripherals assigned to the nonsecure context, interrupts are enabled on NVIC_NS.
- For peripherals assigned to the secure context, interrupts are enabled on NVIC_S.

4.9.4 DMA (context assignment and privilege access settings)

STM32CubeMX allows the user to set as privileged the DMA channel and in some cases, to secure the DMA channel, source and destination see *Figure 180*.

Figure 180. Configuring security and privilege of DMA requests



The DMA channel is set to non-privileged by default. The choice to set it as privileged is always available.

The choice to secure the DMA channel, source, and destination depends on the request characteristics.

There are four cases:

- The request is either a memory to memory transfer request or a DMA generator request: the channel is not secure by default but can be secured. The source and destination can be secured only when the channel is secure.
- The request is for a peripheral assigned to the nonsecure context: channel, source and destination cannot be secured (checkboxes are disabled) and so they are forced to the nonsecure context.
- The request is a peripheral to memory request for a peripheral assigned to the secure context: channel and source are automatically secured (checkboxes enabled, cannot be disabled), while there is a choice to secure or not the destination.
- The request is a memory to peripheral request for a peripheral assigned to the secure context: channel and destination are automatically secured (checkboxes enabled, cannot be disabled), while there is a choice to secure or not the source.

4.9.5 GTZC

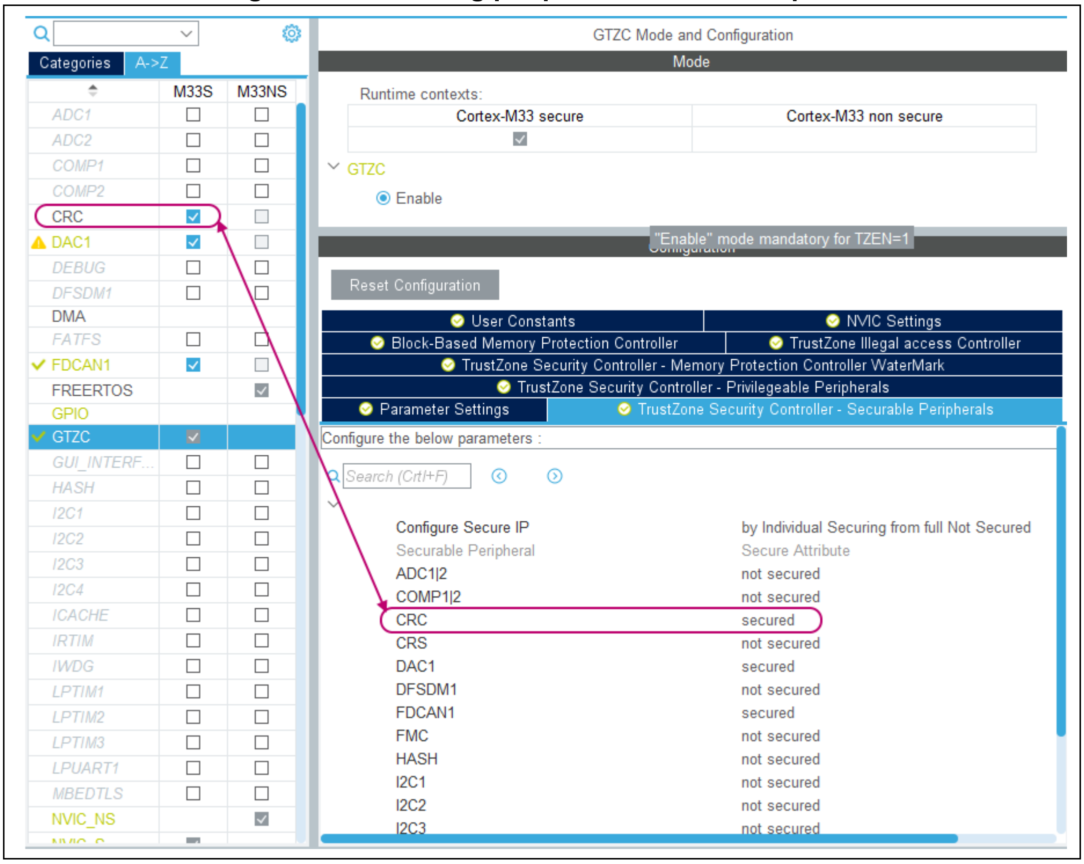
To configure TrustZone system security, STM32L5 series come with a Global TrustZone® security controller (GTZC). Refer to RM0438 "*STM32L552xx and STM32L562xx advanced Arm®-based 32-bit MCUs*" for more details.

In STM32CubeMX, for projects with TrustZone activated, GTZC is enabled by default and cannot be disabled. For projects without Trustzone active, GTZC can be enabled and gives only the possibility to set privileges.

GTZC is made up of three blocks that can be configured through STM32CubeMX using dedicated tabs in GTZC configuration panel:

- TZSC (TrustZone security controller)
  - Defines which peripherals are secured and/or privileged, and controls the nonsecure area size for the watermark memory peripheral controller (MPCWM). The TZSC block informs some peripherals (such as RCC or GPIOs) about the secure status of each securable peripheral, by sharing with RCC and I/O logic.
  - The privileges are set in the TrustZone® Security Controller - Privilegeable Peripherals tab.
  - The secure states are set in TrustZone® Security Controller - Securable Peripherals tab (they match the assignment to context (M33S or M33NS) done on the Tree view or in the Mode panel).
  - The MPCWM configuration is done through the TrustZone® Security Controller - Memory Protection Controller Watermark tab.
- MPCBB (block-based memory protection controller)
  - Controls secure states of all blocks (256-byte pages) of the associated SRAM. It is configured through the Block-based Memory Protection Controller tab.
- TZIC (TrustZone illegal access controller)
  - Gathers all illegal access events in the system and generates a secure interrupt towards NVIC. It is configured through the TrustZone® Illegal Access Controller tab.

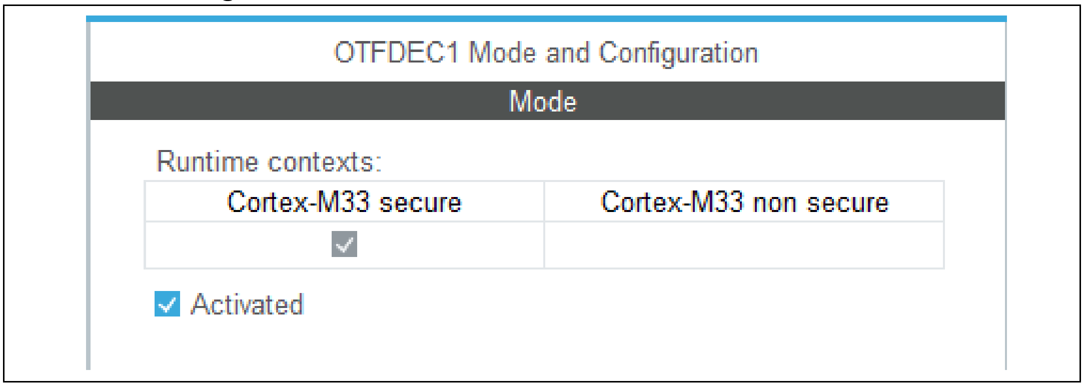
Figure 181. Securing peripherals from GTZC panel



4.9.6 OTFDEC

On-the-fly decryption engine (OTFDEC) allows the user to decrypt on-the-fly AHB traffic based on the read request address information. When security is enabled in the product OTFDEC can be programmed only by a secure host.

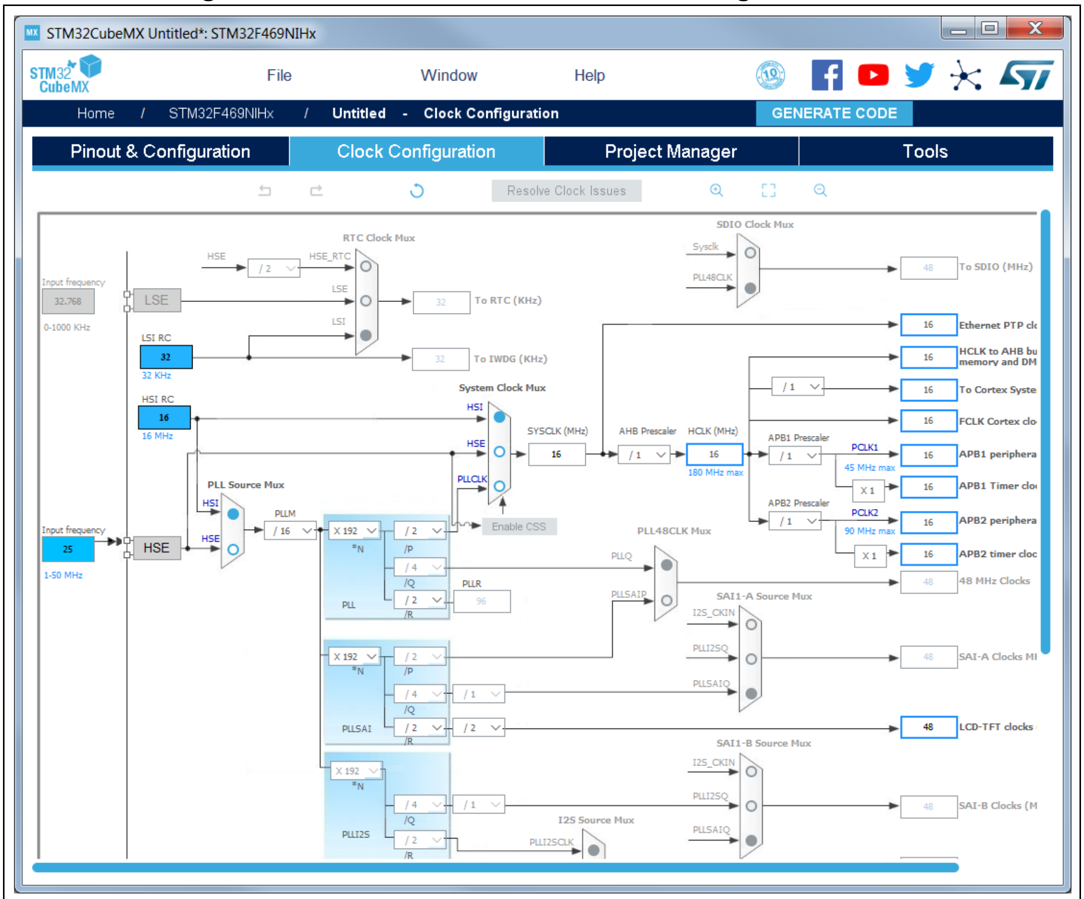
Figure 182. OTFDEC secured when TrustZone is active



4.10 Clock Configuration view

STM32CubeMX **Clock Configuration** window (see *Figure 183*) provides a schematic overview of the clock paths, clock sources, dividers, and multipliers. Drop-down menus and buttons can be used to modify the actual clock tree configuration, to meet the application requirements.

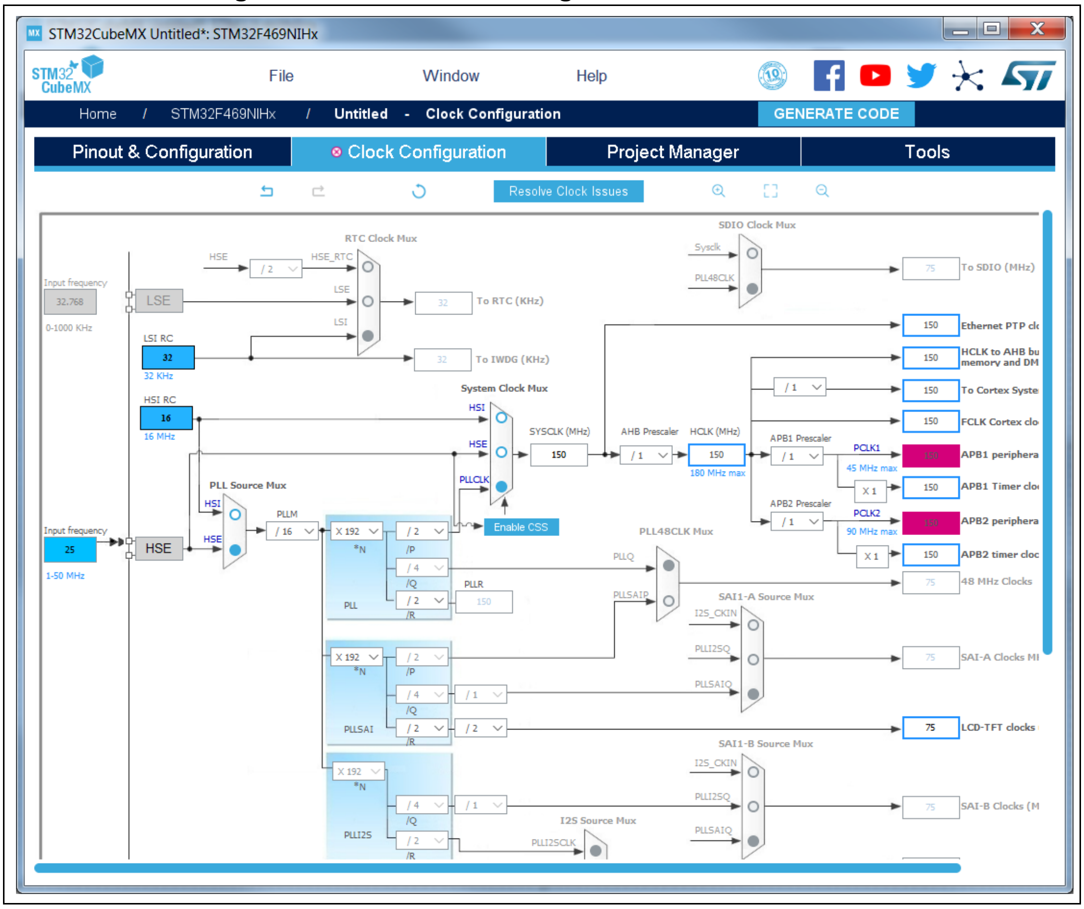
Figure 183. STM32F469NIHx clock tree configuration view



Actual clock speeds are displayed and active. The used clock signals are highlighted in blue.

Out-of-range configured values are highlighted (as shown in *Figure 184*) to flag potential issues. A solver feature is proposed to automatically resolve such configuration issues.

Figure 184. Clock tree configuration view with errors



Reverse path is supported: just enter the required clock speed in the blue filed and STM32CubeMX attempts to reconfigure multipliers and dividers to provide the requested value. The resulting clock value can then be locked by right clicking the field to prevent modifications.

STM32CubeMX generates the corresponding initialization code:

- main.c with relevant HAL_RCC structure initializations and function calls
- stm32xxxx_hal_conf.h for oscillator frequencies and V_{DD} values.

4.10.1 Clock tree configuration functions

External clock sources

When external clock sources are used, the user must previously enable them from the **Pinout** view available under the RCC peripheral.

Peripheral clock configuration options

Other paths, corresponding to clock peripherals, are grayed out. To become active, the peripheral must be properly configured in the **Pinout** view. This view allows the user to:

- **Enter a frequency value for the CPU clock (HCLK), buses or peripheral clocks**

  STM32CubeMX tries to propose a clock tree configuration that reaches the desired frequency while adjusting prescalers and dividers and taking into account other peripheral constraints (such as USB clock minimum value). If no solution can be found, STM32CubeMX proposes to switch to a different clock source or can even conclude that no solution matches the desired frequency.
- **Lock the frequency fields for which the current value should be preserved**

  Right click a frequency field and select **Lock** to preserve the value currently assigned when STM32CubeMX searches for a new clock configuration solution.

  The user can unlock the locked frequency fields when the preservation is no longer necessary.
- **Select the clock source that will drive the system clock (SYSCLK)**
  - External oscillator clock (HSE) for a user defined frequency.
  - Internal oscillator clock (HSI) for the defined fixed frequency.
  - Main PLL clock
- **Select secondary sources (as available for the product)**
  - Low-speed internal (LSI) or external (LSE) clock
  - I2S input clock
  - Other sources
- **Select prescalers, dividers and multipliers values**
- **Enable the Clock Security system (CSS) on HSE when it is supported by the MCU**

  This feature is available only when the HSE clock is used as the system clock source directly or indirectly through the PLL. It allows detecting HSE failure and inform the software about it, thus allowing the MCU to perform rescue operations.
- **Enable the CSS on LSE when it is supported by the MCU**

  This feature is available only when the LSE and LSI are enabled and after the RTC or LCD clock sources have been selected to be either LSE or LSI.
- **Reset the Clock tree default settings by using the toolbar Reset button**

  This feature reloads STM32CubeMX default clock tree configuration.
- **Undo/Redo user configuration steps by using the toolbar Undo/Redo buttons**
- **Detect and resolve configuration issues**

  Erroneous clock tree configurations are detected prior to code generation. Errors are highlighted in fuchsia and the **Clock Configuration** view is marked with a fuchsia cross (see *Figure 184*).

  Issues can be resolved manually or automatically by clicking the **Resolve Clock Issue** button that is enabled only if issues have been detected.

  The underlying resolution process follows a specific sequence:

  a) Setting HSE frequency to its maximum value (optional).

  b) Setting HCLK frequency then peripheral frequencies to a maximum or minimum value (optional).

  c) Changing multiplexers inputs (optional).

d) Finally, iterating through multiplier/dividers values to fix the issue. The clock tree is cleared from fuchsia highlights if a solution is found, otherwise an error message is displayed.

Note: To be available from the clock tree, external clocks, I2S input clock, and master clocks must be enabled in RCC configuration in the ***Pinout*** *view. This information is also available as tooltips.*

The tool automatically performs the following operations:

- Adjust bus frequencies, timers, peripherals and master output clocks according to user selection of clock sources, clock frequencies and prescalers/multipliers/dividers values.
- Check the validity of user settings.
- Highlight invalid settings in fuchsia and provide tooltips to guide the user to achieve a valid configuration.

The **Clock Configuration** view is adjusted according to the RCC settings (configured in RCC **Pinout & Configuration** views) and vice versa:

- If in RCC **Pinout** view, the external and output clocks are enabled, they become configurable in the **Clock Configuration** view.
- If in RCC Configuration view, the Timer prescaler is enabled, the choice of Timer clocks multipliers is adjusted.

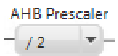
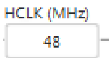
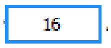
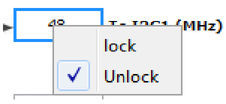
Conversely, the clock tree configuration may affect some RCC parameters in the configuration view:

- Flash latency: number of wait states automatically derived from V_{DD} voltage, HCLK frequency, and power over-drive state.
- Power regulator voltage scale: automatically derived from HCLK frequency.
- Power over-drive is enabled automatically according to HCLK frequency. When the power drive is enabled, the maximum possible frequency values for AHB and APB domains are increased. They are displayed in the **Clock Configuration** view.

The default optimal system settings that is used at startup are defined in the *system_stm32f4xx.c* file. This file is copied by STM32CubeMX from the STM32CubeF4 MCU package. The switch to user defined clock settings is done afterwards in the main function.

Figure 183 gives an example of Clock tree configuration for an STM32F429x MCU, and *Table 9* describes the widgets that can be used to configure each clock.

Table 9. Clock configuration view widgets

Format	Configuration status of the Peripheral Instance
	Active clock sources
	Unavailable settings are blurred or grayed out (clock sources, dividers,…)
	Gray drop down lists for prescalers, dividers, multipliers selection.
	Multiplier selection
	User defined frequency values
	Automatically derived frequency values
	User-modifiable frequency field
	Right click blue border rectangles to lock/unlock a frequency field. Lock to preserve the frequency value during clock tree configuration updates.

4.10.2 Securing clock resources (STM32L5 series only)

When the TrustZone security is activated, the RCC is able, through the security configuration register, to prevent nonsecure access to system clock resources.

Accordingly, STM32CubeMX allows the user to configure as secure:

- system clock sources with a fixed frequency: HSI, LSI, and RC48
- system clock sources with a configurable frequency: HSE (+CSS), MSI and LSE (+CSS)
- two multiplexers: CLK48 clock multiplexer, System Clock (+MCO source) multiplexer
- other system configurations: PLLSYS, PLLSAI1, PLLSAI2 phase-locked loops and AHB/APB1/APB2 bus pre-scalers

In the Clock Configuration view, these securable resources are highlighted with a key icon. Security is enabled using the Secure checkbox accessed through a right-click on the resource. Once the resource is secure, it is highlighted with a green square.

Configurable resources can be locked to prevent further configuration changes: this is done by selecting the Lock checkbox accessed through a right-click on the resource.

There is also a shortcut button to lock/unlock in one click all resources that are both securable and configurable.

When a peripheral is configured as secure, its related clock, reset, clock source and clock enable are also secure. In STM32CubeMX the peripheral is configured as secure in the Pinout & Configuration view and its clock source is automatically highlighted as secure using a green square in the Clock configuration view.

Table 10. Clock Configuration security settings

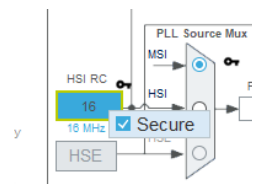
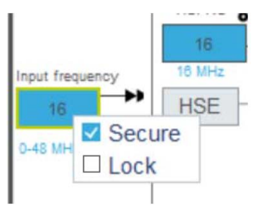
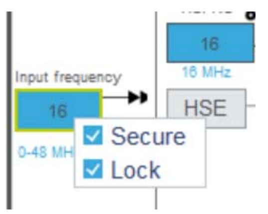
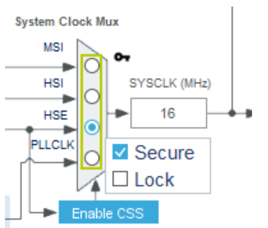
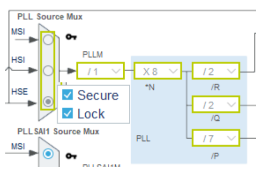
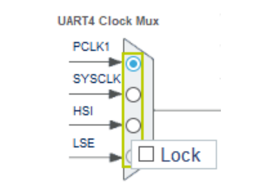
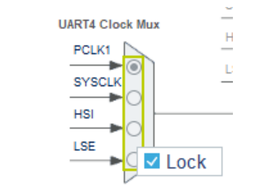
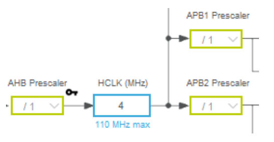
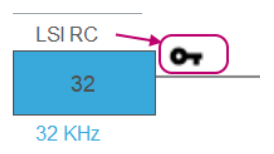
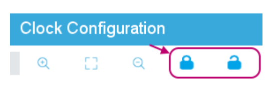
View	Description
	Example of non-configurable system clock resource that is secured.
	Example of the system clock HSE clock source that is secured and remains open for editing: the frequency value can be changed.
	Example of the system clock HSE clock source that is secured and has been locked for editing: the frequency value cannot be modified.
	Example of the system clock multiplexer that is secured and unlocked: the clock source can be changed.
	Example of the main PLL multiplexer that is secured and locked. The clock source is HSE and cannot be changed. PLLxxM, PLLxxN, PLLxxP, PLLxxQ and PLLxxR are secured and locked for editing as well.

Table 10. Clock Configuration security settings (continued)

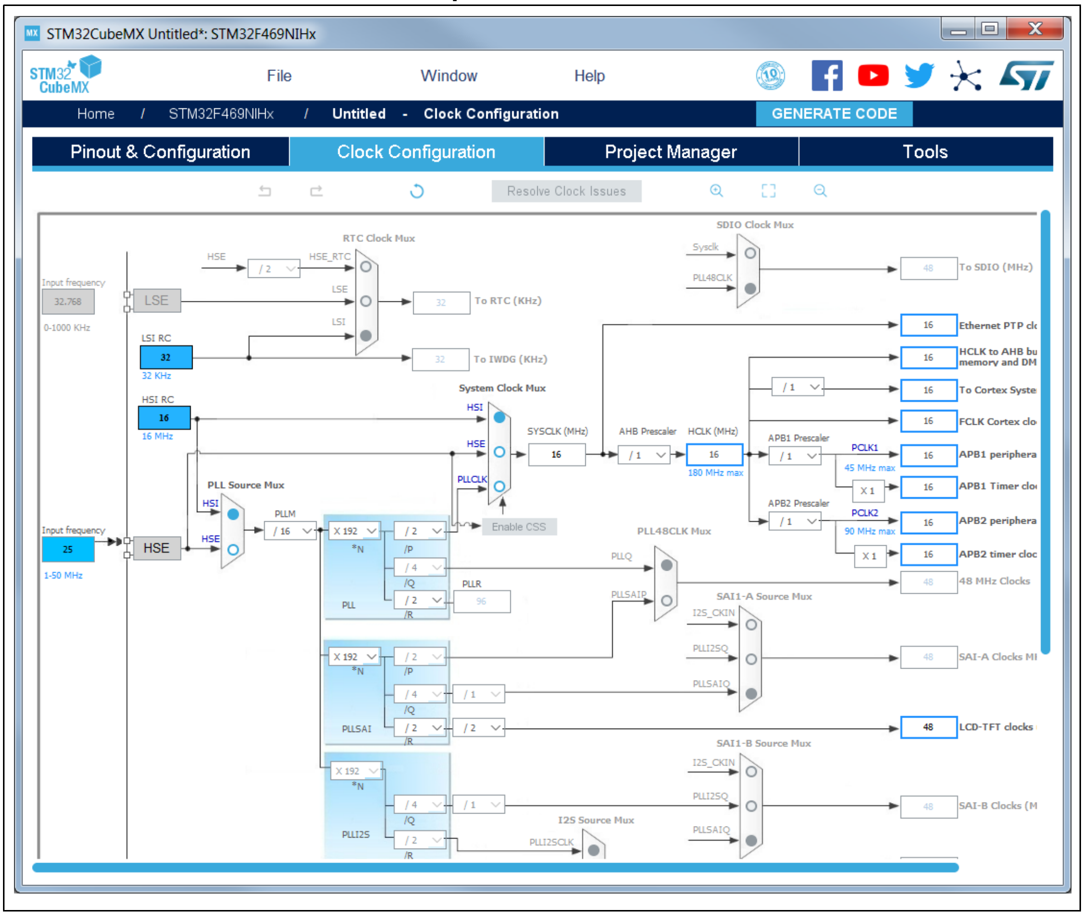
View	Description
	Example of the UART4 clock source multiplexer: the clock source is secured because the UART4 peripheral is configured as secure in the Pinout & Configuration view. It is set to PCLK1 and can be changed as the Lock checkbox is unchecked.
	Example of the UART4 clock source multiplexer: the clock source is secured because the UART4 peripheral is configured as secure in the Pinout & Configuration view. It is set to PCLK1 and can no longer be changed as Lock is on.
	Example of securing and locking the access to AHB prescaler. APB1 and APB2 prescalers are locked as well.
	Example of LSI highlighted as a securable resource using the key icon.
	Lock/Unlock All button (active only for secure and configurable resources).

4.10.3 Recommendations

The **Clock Configuration** view is not the only entry for clock configuration, RCC and RTC peripherals can also be configured.

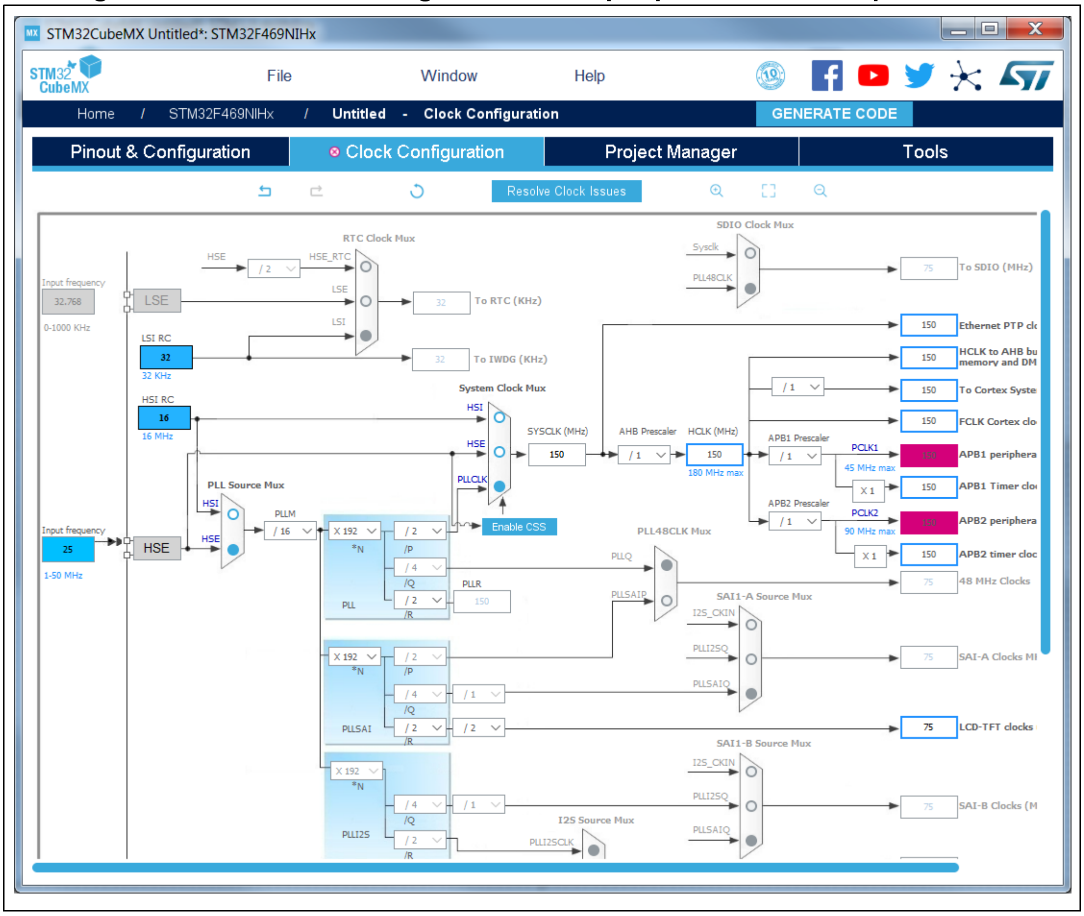
1. From the **Pinout & Configuration** view, go to the RCC mode panel to enable the clocks as needed: external clocks, master output clocks and Audio I2S input clock when available. Then go to the RCC configuration panel, and adjust the default settings if needed. Changes are reflected in the **Clock Configuration** view. The defined settings may change the settings in the RCC configuration as well (see *Figure 185*).

Figure 185. Clock tree configuration: enabling RTC, RCC clock source and outputs from Pinout view



2. Go to the **RCC configuration** in the **Pinout & Configuration** view. The settings defined there for advanced configurations are reflected in the **Clock configuration** view. The defined settings may change the settings in the RCC configuration.

Figure 186. Clock tree configuration: RCC peripheral advanced parameters



4.10.4 STM32F43x/42x power overdrive feature

STM32F42x/43x MCUs implement a power overdrive feature that allows them to work at the maximum AHB/APB bus frequencies (for example, 180 MHz for HCLK) when a sufficient V_{DD} supply voltage is applied (for example, V_{DD} > 2.1 V).

Table 11 lists the different parameters linked to the power overdrive feature and their availability in STM32CubeMX user interface.

Table 11. Voltage scaling versus power overdrive and HCLK frequency

Parameter	STM32CubeMX panel	Value
V_{DD} voltage	Configuration (RCC)	User-defined within a predefined range. Impacts power over-drive.
Power regulator voltage scaling		Automatically derived from HCLK frequency and power over-drive (see *Table 12*).
Power over-drive		This value is conditioned by HCLK and V_{DD} values (see *Table 12*). It can be enabled only if $V_{DD} \geq 2.2$ V. When $V_{DD} \geq 2.2$ V it is automatically derived from HCLK, or can be configured by the user if multiple choices are possible (as an example, HCLK = 130 MHz)
HCLK/AHB clock maximum frequency value	Clock Configuration	Displayed in blue to indicate the maximum possible value. For example: maximum value is 168 MHz for HCLK when power overdrive cannot be activated (when $V_{DD} \leq 2.1$ V), otherwise it is 180 MHz.
APB1/APB2 clock maximum frequency value		Displayed in blue to indicate the maximum possible value.

Table 12 gives the relations between power-over drive mode and HCLK frequency.

Table 12. Relations between power over-drive and HCLK frequency

HCLK frequency range: V_{DD} > 2.1 V required to enable power over-drive (POD)	Corresponding voltage scaling and power over-drive (POD)
≤120 MHz	– Scale 3 – POD is disabled
120 to 144 MHz	– Scale 2 – POD can be enabled or disabled
144 to 168 MHz	– Scale 1 when POD is disabled – Scale 2 when POD is enabled
168 to 180 MHz	– POD must be enabled – Scale 1 (otherwise frequency range not supported)

4.10.5 Clock tree glossary

Table 13. Glossary

Acronym	Definition
HSI	High speed Internal oscillator: enabled after reset, lower accuracy than HSE
HSE	High speed external oscillator: requires an external clock circuit
PLL	Phase locked loop: used to multiply above clock sources
LSI	Low speed Internal clock: low power clocks usually used for watchdog timers

Table 13. Glossary (continued)

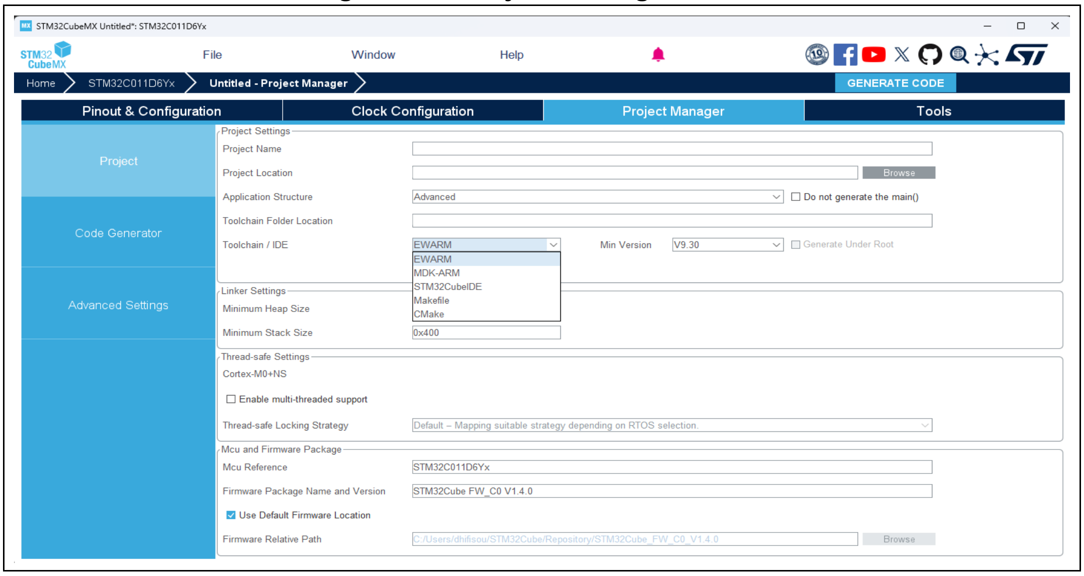
Acronym	Definition
LSE	Low speed external clock: powered by an external clock
SYSCLK	System clock
HCLK	Internal AHB clock frequency
FCLK	Cortex free running clock
AHB	Advanced high performance bus
APB1	Low speed advanced peripheral bus
APB2	High speed advanced peripheral bus

4.11 Project Manager view

This view (see *Figure 187*) comes with three tabs:

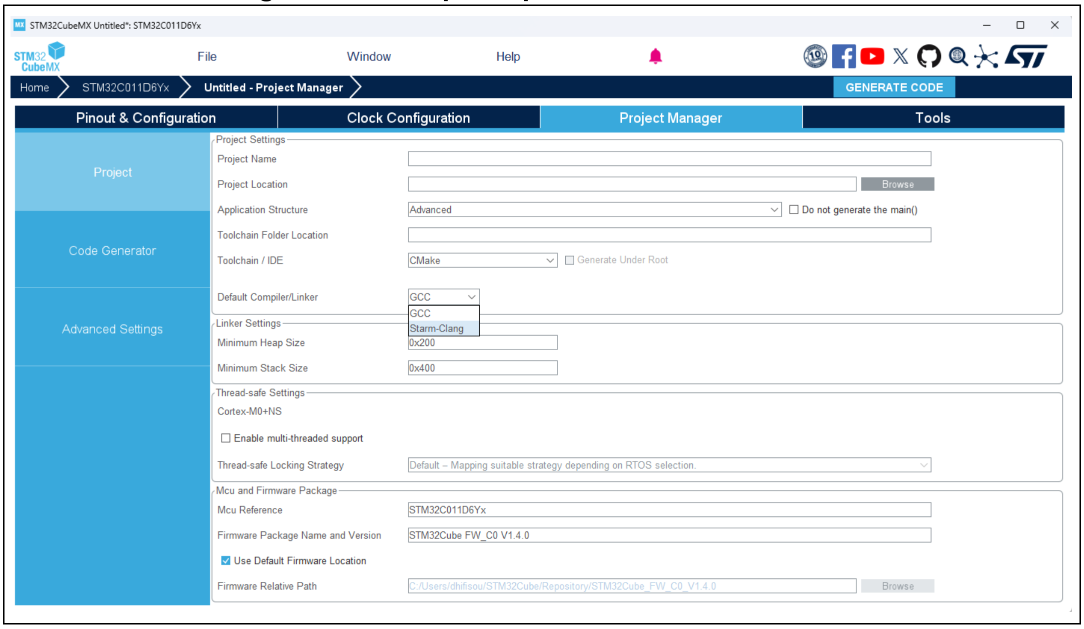
- Project: to specify the project name, location, toolchain, and firmware version.
- Code Generation: to set code generation options, such as the location of peripheral initialization code, library copy/link options, and to select templates for customized code.
- Advanced Settings: dedicated to ordering STM32CubeMX initialization function calls.

Figure 187. Project Settings window



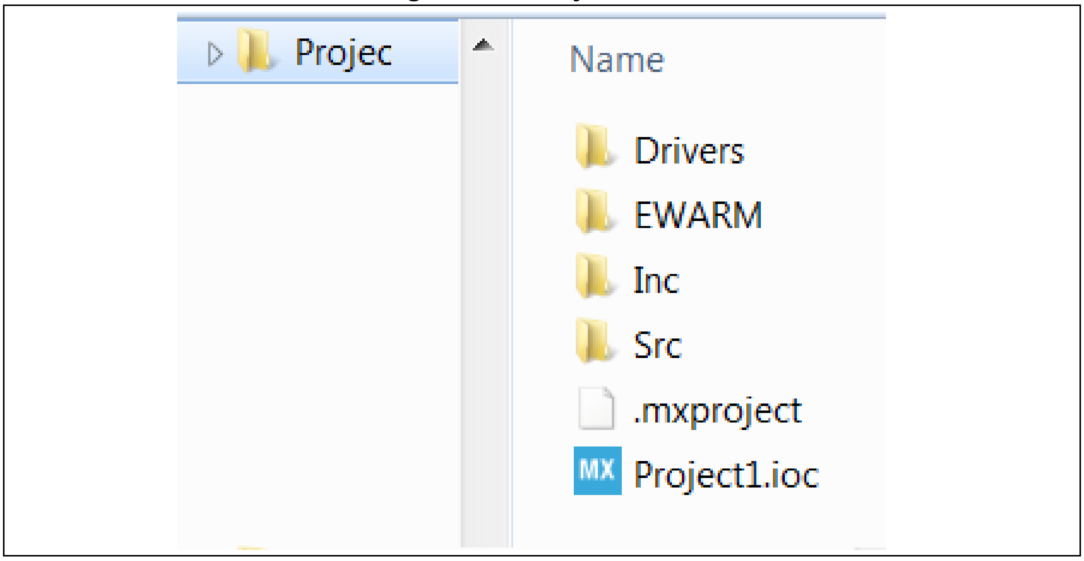
If the user chooses the CMake toolchain, a new field named “Default Compiler/Linker” appears, to enable the selection of the desired compiler (GCC/Starm-Clang).

Figure 188. Compiler option for CMake toolchain



The code is generated in the project folder tree shown in *Figure 189*.

Figure 189. Project folder



Note: Some project setting options become read-only once the project is saved. To modify these options, the project must be saved as new, using the ***File > Save Project as*** *menu.*

Caution: STM32CubeMX uses reserved folder names. User cannot create new folder named *Middlewares* or *Utilities* inside project folder generated by STM32CubeMX, because, after code regeneration, those folders are deleted or modified.

4.11.1 Project tab

The **Project** tab of the **Project Settings** window allows configuring the following options (see *Figure 187*):

- Project settings:
  - Project name: name used to create the project folder and the .ioc file name at a given project location
  - Project location: directory where the project folder is stored.
  - Application structure: select between Basic and Advanced options.

    Basic structure: recommended for projects using one or no middleware. This structure consists in placing the IDE configuration folder at the same level as the sources, organized in sources and includes subfolders (see *Figure 190*)

    Advanced structure: recommended when several middleware components are used in the project, makes the integration of middleware applications easier (see *Figure 191*)
  - Toolchain folder location: by default, it is located in the project folder at the same level as the .ioc file.
  - Toolchain/IDE: selected toolchain
  - For the STM32MPUs, OpenSTLinux settings: location of generated device tree and manifest version and contents for current project (see *Figure 192*). These information enable the synchronization of the right SW components versions with STM32CubeMP1 for Cortex® M and Linux, tf-a, u-boot for Cortex® A. It is important to take them into account especially to ensure one Cube firmware version is aligned with SW components for Cortex® A around OpenAMP / RPM link and resource management API.

  Selecting *Makefile* under Toolchain/IDE leads to the generation of a generic gcc-based makefile.
- Additional project settings for STM32CubeIDE toolchain:

  Select the optional **Generate under root** checkbox to generate the toolchain project files in STM32CubeMX user project root folder or deselect it to generate them under a dedicated toolchain folder.

  STM32CubeMX project generation under the root folder allows the user to benefit from the following Eclipse features:
  - Optional copy of the project into the Eclipse workspace when importing a project.
  - Use of source control systems such as GIT or SVN from the Eclipse workspace.

  Choosing to copy the project into workspace prevents any further synchronization between changes done in Eclipse and changes done in STM32CubeMX, as there will be two different copies of the project.

- Linker settings: value of minimum heap and stack sizes to allocate for the application. The default values are 0x200 and 0x400 for heap and stack sizes, respectively. These values may need to be increased when the application uses middleware stacks.
- Firmware package selection when more than one version is available (this is the case when successive versions implement the same API and support the same MCUs). By default, the latest available version is used.
- Firmware location selection option

  The default location is the location specified under the **Help > Updater Settings** menu.

  Deselecting the **Use Default Firmware Location** checkbox allows the user to specify a different path for the firmware that will be used for the project (see *Figure 193*).

Figure 190. Selecting a basic application structure

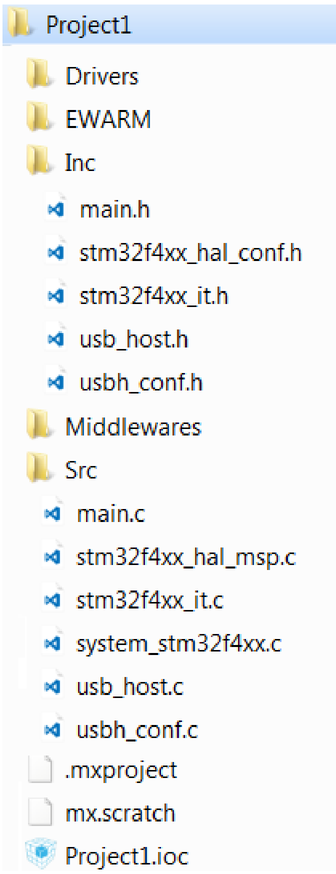


Figure 191. Selecting an advanced application structure

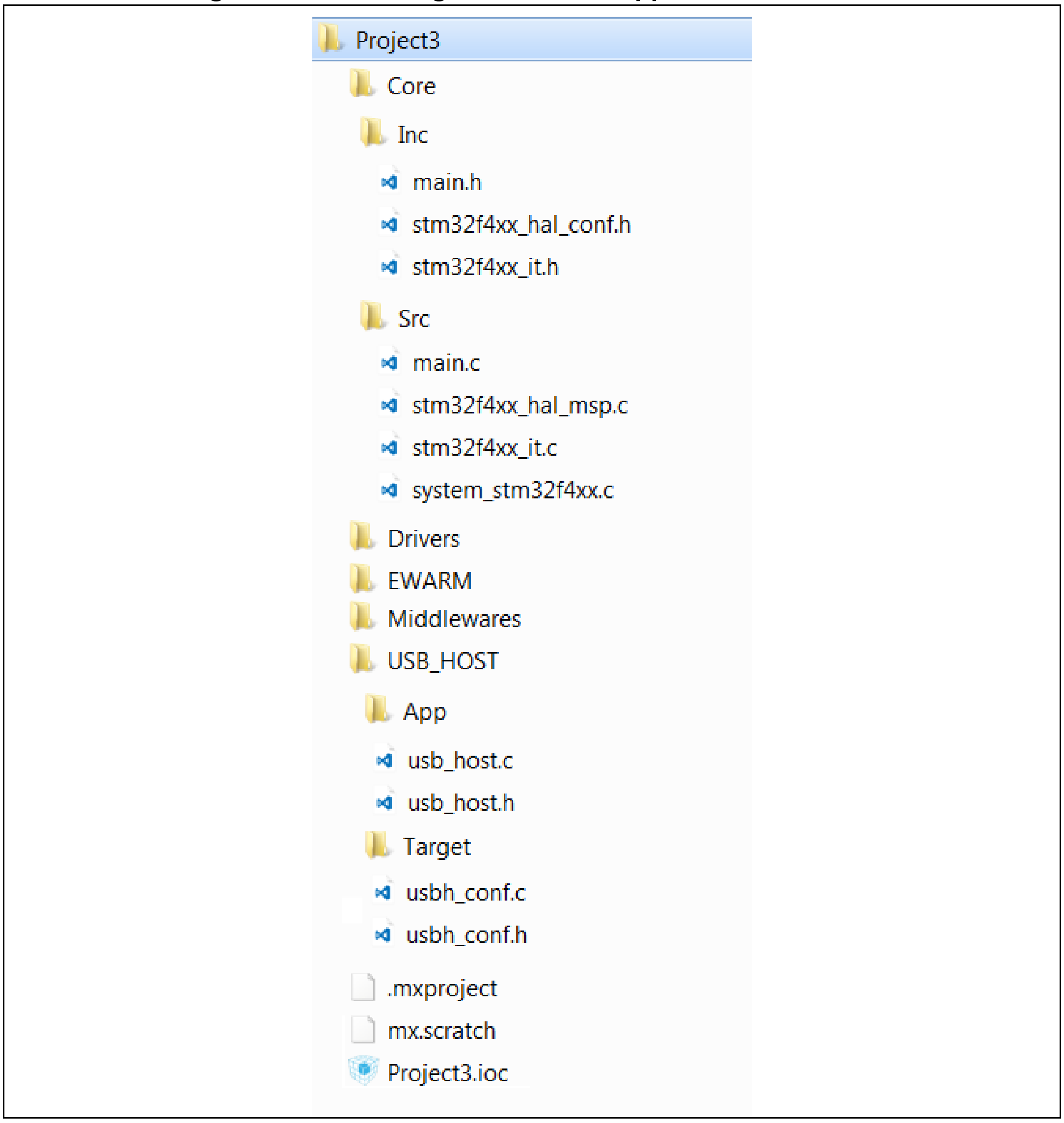


Figure 192. OpenSTLinux settings (STM32MPUs only)

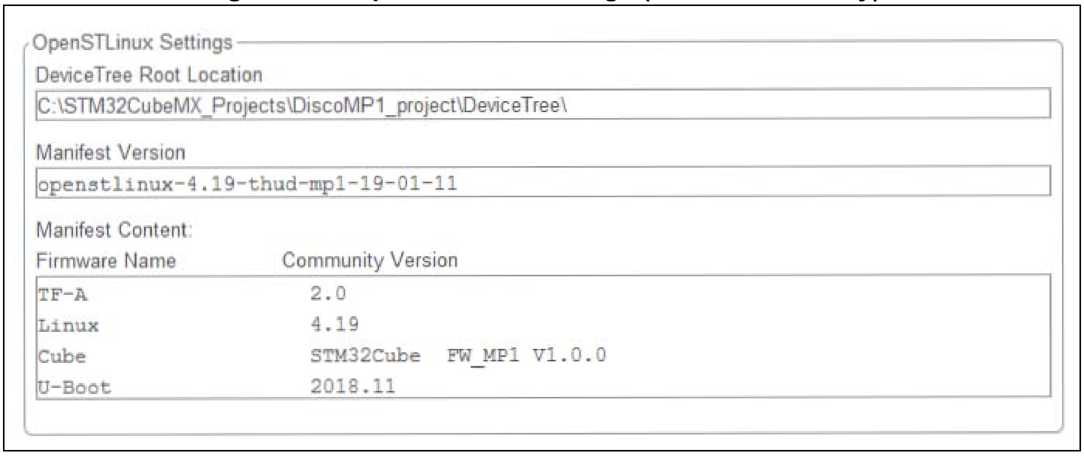
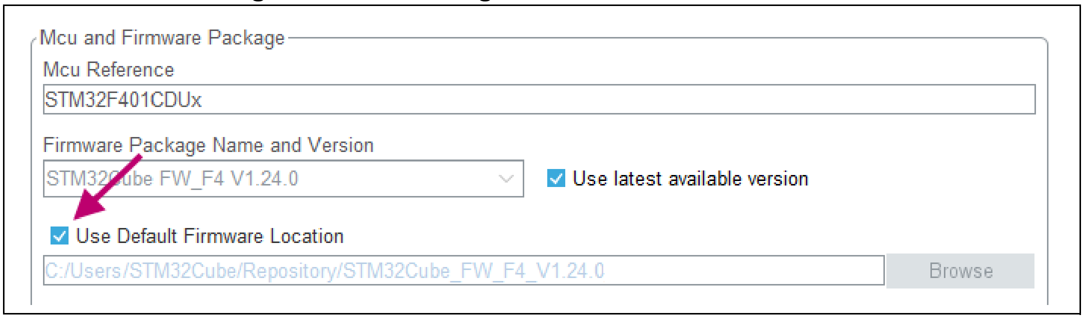
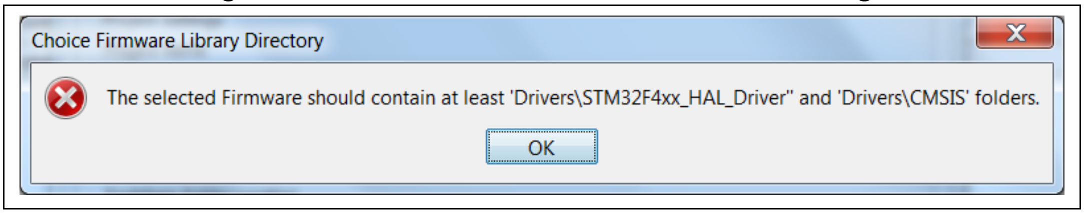


Figure 193. Selecting a different firmware location



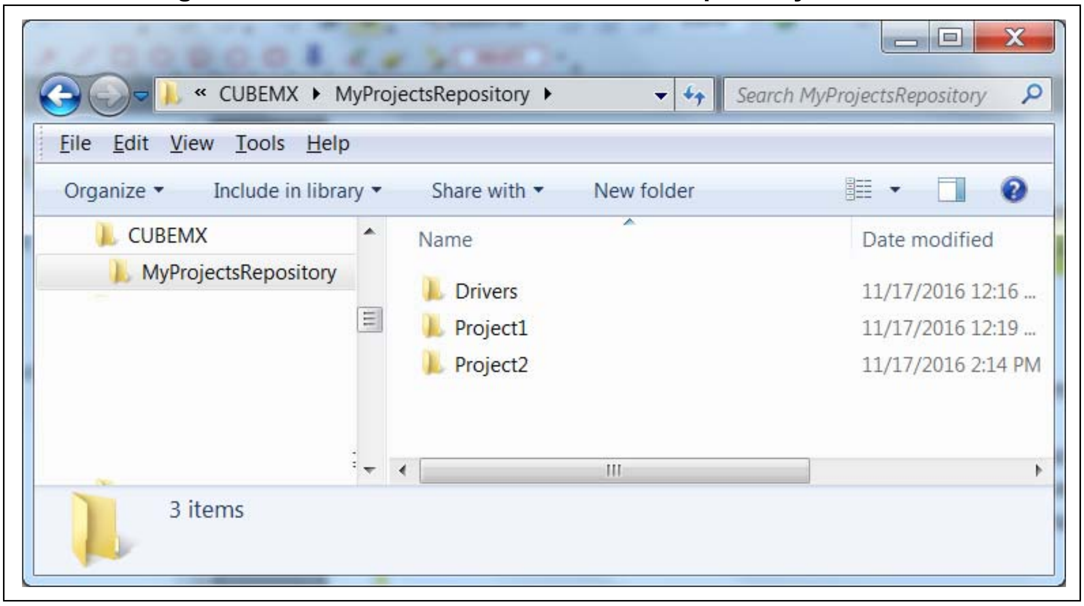
The new location must contain at least a *Drivers* directory containing the HAL and CMSIS drivers from the relevant STM32Cube MCU package. An error message pops up if the folders are not found (see *Figure 194*).

Figure 194. Firmware location selection error message



To reuse the same *Drivers* folder across all projects that use the same firmware location, select the **Add the library files as reference** from the **Code generator** tab (see *Figure 195*).

Figure 195. Recommended new firmware repository structure



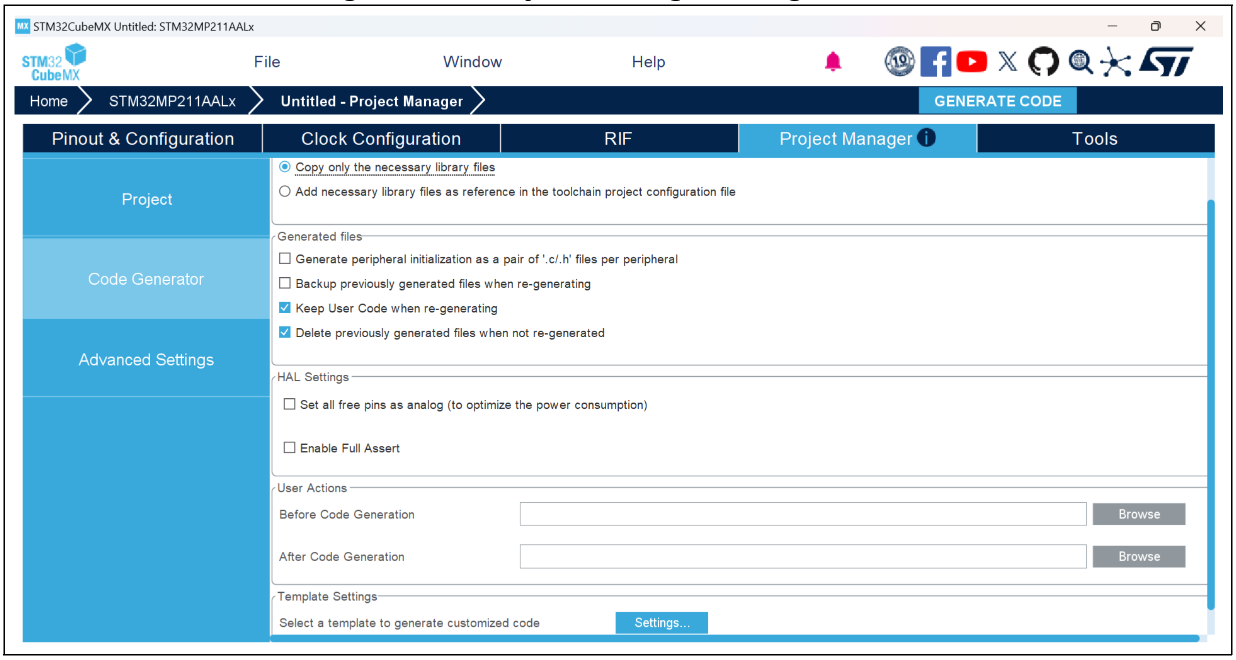
Caution: STM32CubeMX manages firmware updates only for this default location. Choosing another location prevents the user from benefiting from automatic updates. The user must manually copy new driver versions to its project folder.

4.11.2 Code Generator tab

The **Code Generator** tab allows specifying the following code generation options (see *Figure 196*):

- STM32Cube Firmware Library Package option
- Generated files options
- HAL settings options
- Custom code template options

Figure 196. Project Settings code generator



STM32Cube Firmware Library Package option

The following actions are possible:

- Copy all used libraries into the project folder

  STM32CubeMX copies to the user project folder the drivers libraries (HAL, CMSIS) and the middleware libraries relevant to the user configuration (such as FatFs, USB).
- Copy only the necessary library files:

  STM32CubeMX copies to the user project folder only the library files relevant to the user configuration (e.g., SDIO HAL driver from the HAL library).
- Add the required library as referenced in the toolchain project configuration file

  By default, the required library files are copied to the user project. Select this option for the configuration file to point to files in STM32CubeMX repository instead: the user project folder will not hold a copy of the library files but only a reference to the files in STM32CubeMX repository.

Generated files options

This area allows the user to define the following options:

- Generate peripheral initialization as a pair of .c/.h files or keep all peripheral initializations in the main.c file.
- Backup previously generated files in a backup directory

  The .bak extension is added to previously generated .c/.h files.

  Keep user code when regenerating the C code.

  This option applies only to user sections within STM32CubeMX generated files. It does not apply to the user files that might have been added manually or generated via ftl templates.
- Delete previously generated files when these files are no longer needed by the current configuration. For example, uart.c/.h file are deleted if the UART peripheral, that was enabled in previous code generation, is now disabled in current configuration.

HAL settings options

This area allows selection one HAL settings options among the following:

- Set all free pins as analog to optimize power consumption
- Enable/disable Use the *Full Assert* function: the Define statement in the stm32xx_hal_conf.h configuration file is commented or uncommented, respectively.

Custom code template options

To generate custom code, click the **Settings** button under **Template Settings**, to open the Template Settings window (see *Figure 197*).

The user is then prompted to choose a source directory to select the code templates from, and a destination directory where the corresponding code will be generated.

The default source directory points to the extra_template directory, within the installation folder, to use for storing all user defined templates. The default destination folder is located in the user project folder. STM32CubeMX then uses the selected templates to generate user custom code (see *Section 6.3*).

Figure 198 shows the result of the template configuration of *Figure 197*: a sample.h file is generated according to sample_h.ftl template definition.

Figure 197. Template Settings window

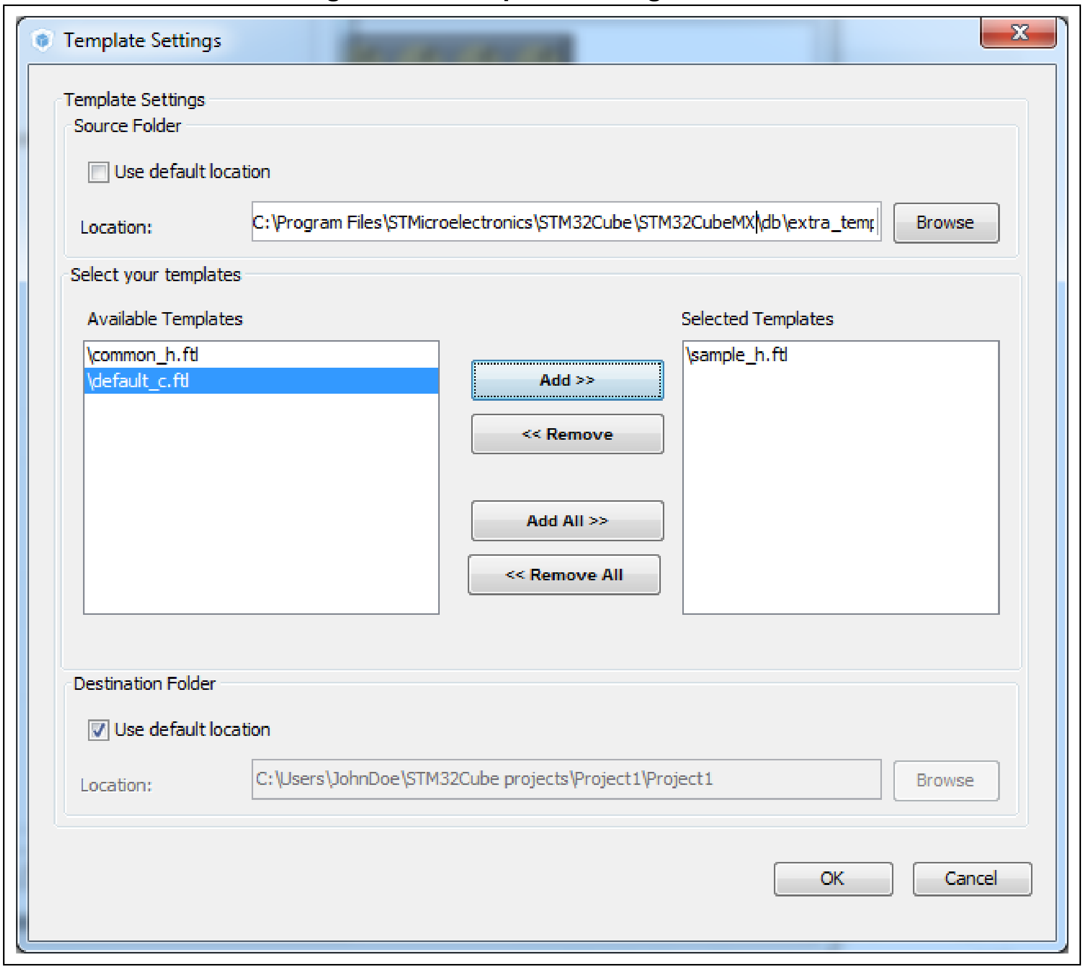
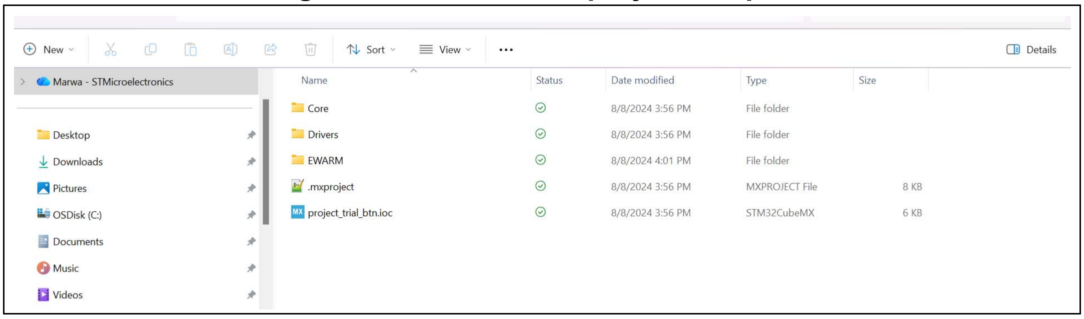


Figure 198. Generated project template



4.11.3 Advanced Settings tab

This tab comes with three panels (see *Figure 199*):

- The **Driver selector** panel, to select the driver (HAL or LL) to be used when generating the initialization code of a peripheral instance.
- The **Generated Function Calls** panel, to choose whether the function calls must be generated or not, generated as static or not and in which order.
- The **Register callback** panel, to select the peripherals for which the register callback define must be generated as part of the stm32xxxx_hal_conf.h file.

As an example, when ADC is enabled in the register callback panel, STM32CubeMX generates

```
#define USE_HAL_ADC_REGISTER_CALLBACKS 1U
```

Choosing not to generate code for some peripherals or middlewares

By default, STM32CubeMX generates initialization code. This automatic generation can be disabled per peripheral or middleware in the Generate code column.

Ordering initialization function calls

By default, the generated code calls the peripheral/middleware initialization functions in the order in which peripherals and middleware have been enabled in STM32CubeMX. The user can then choose to re-order them by modifying the Rank number, using the up and down arrow buttons.

The reset button allows the user to switch back to alphabetical order.

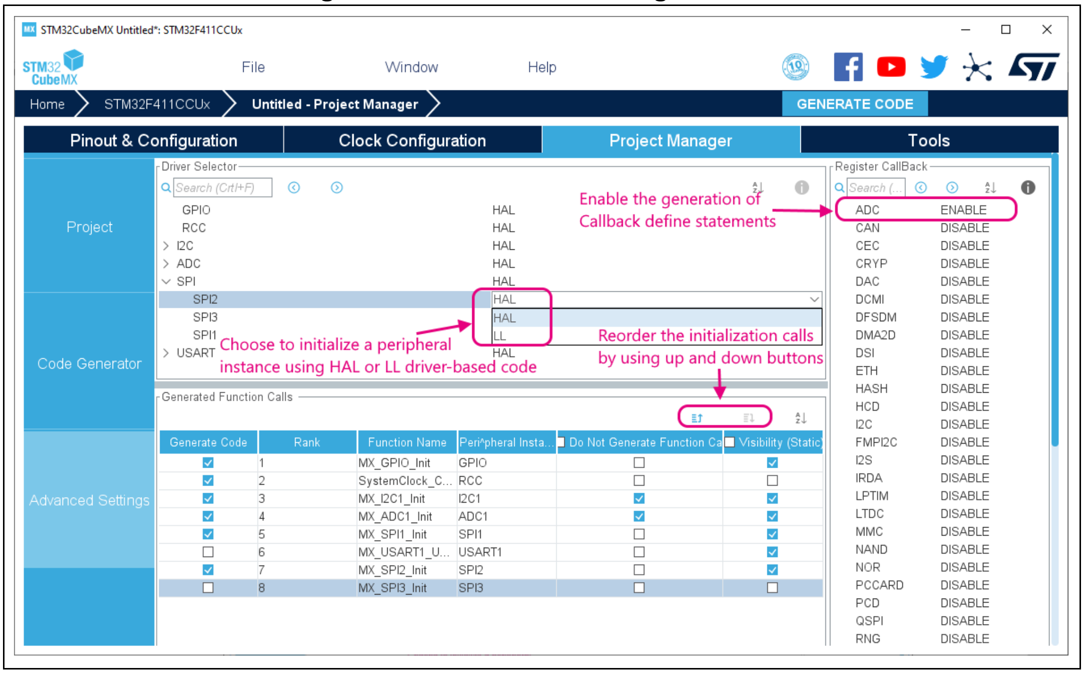
Disabling calls to initialization functions

If the “**Not to be generated**” checkbox is checked, STM32CubeMX does not generate the call to the corresponding peripheral initialization function. It is up to the user code to do it.

Choosing between HAL and LL based code generation for a given peripheral instance

Starting from STM32CubeMX 4.17 and STM32L4 series, STM32CubeMX offers the possibility for some peripherals to generate initialization code based on Low Layer (LL) drivers instead of HAL drivers: the user can choose between LL and HAL driver in the **Driver Selector** section. The code is generated accordingly (see *Section 6.2*).

Figure 199. Advanced Settings window



Unselecting the **Visibility (Static)** option, as shown for MX_I2C1_init function in *Figure 199*, allows the generation of the function definition without the static keyword, and hence extends its visibility outside the current file (see *Figure 200*).

Figure 200. Generated init functions without C language “static” keyword

```
/* Private function prototypes --------
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_LPTIM1_Init(void);
static void MX_LPTIM2_Init(void);
void MX_I2C1_Init(void);
static void MX_I2C2_Init(void);
static void MX_SPI1_Init(void);
static void MX_SPI2_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_USART2_Init(void);
```

Caution: For the STM32MPUs
By default the SystemClock_Config function is called in STM32Cube Cube firmware *main()* function, as the “Not generate Function call” box in Project Manager/Advanced Settings panel is not activated by default (see *Figure 199*).
This configuration is valid for running STM32Cube firmware in engineering (Cortex-M4 stand-alone) mode, and is not valid for running STM32Cube firmware in production mode: the “Not generate Function call” box must be checked under Project Manager/Advanced Settings panel, so that there is no call to *SystemClock_Config()* in the *main()* function.

4.12 Import Project window

4.12.1 Import Project feature for STM32MCU projects

The **Import Project** menu eases the porting of a previously-saved configuration to another MCU. By default the following settings are imported:

- **Pinout** tab: MCU pins and corresponding peripheral modes.The import fails if the same peripheral instances are not available in the target MCU.
- **Clock configuration** tab: clock tree parameters.
- **Configuration** tab: peripherals and middleware libraries initialization parameters.
- **Project settings**: choice of toolchain and code generation options.

To import a project, proceed as follows:

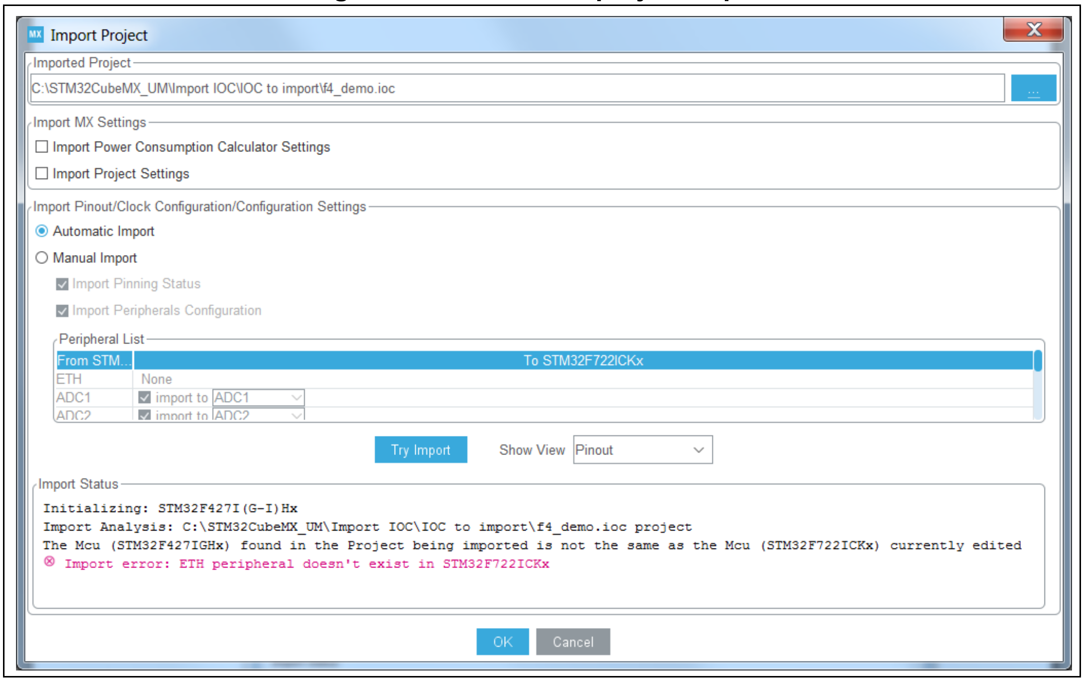
1. Select the **Import project** icon that appears under the **File** menu after starting a New Project and once an MCU has been selected.

   The menu remains active as long as no user configuration settings are defined for the new project, that is just after the MCU selection. It is disabled as soon as a user action is performed on the project configuration.
2. Select **File > Import Project** for the dedicated Import project window to open. This window allows to specify the following options:
   – The STM32CubeMX configuration file (.ioc) pathname of the project to import on top of current empty project.
   – Whether to import the configuration defined in the **Power Consumption Calculator** tab or not.
   – Whether to import the project settings defined through the **Project > Settings** menu: IDE selection, code generation options and advanced settings.
   – Whether to import the project settings defined through the **Project > Settings** menu: IDE selection and code generation options.

– Whether to attempt to import the whole configuration (automatic import) or only a subset (manual import).

a) Automatic project import (see *Figure 201*)

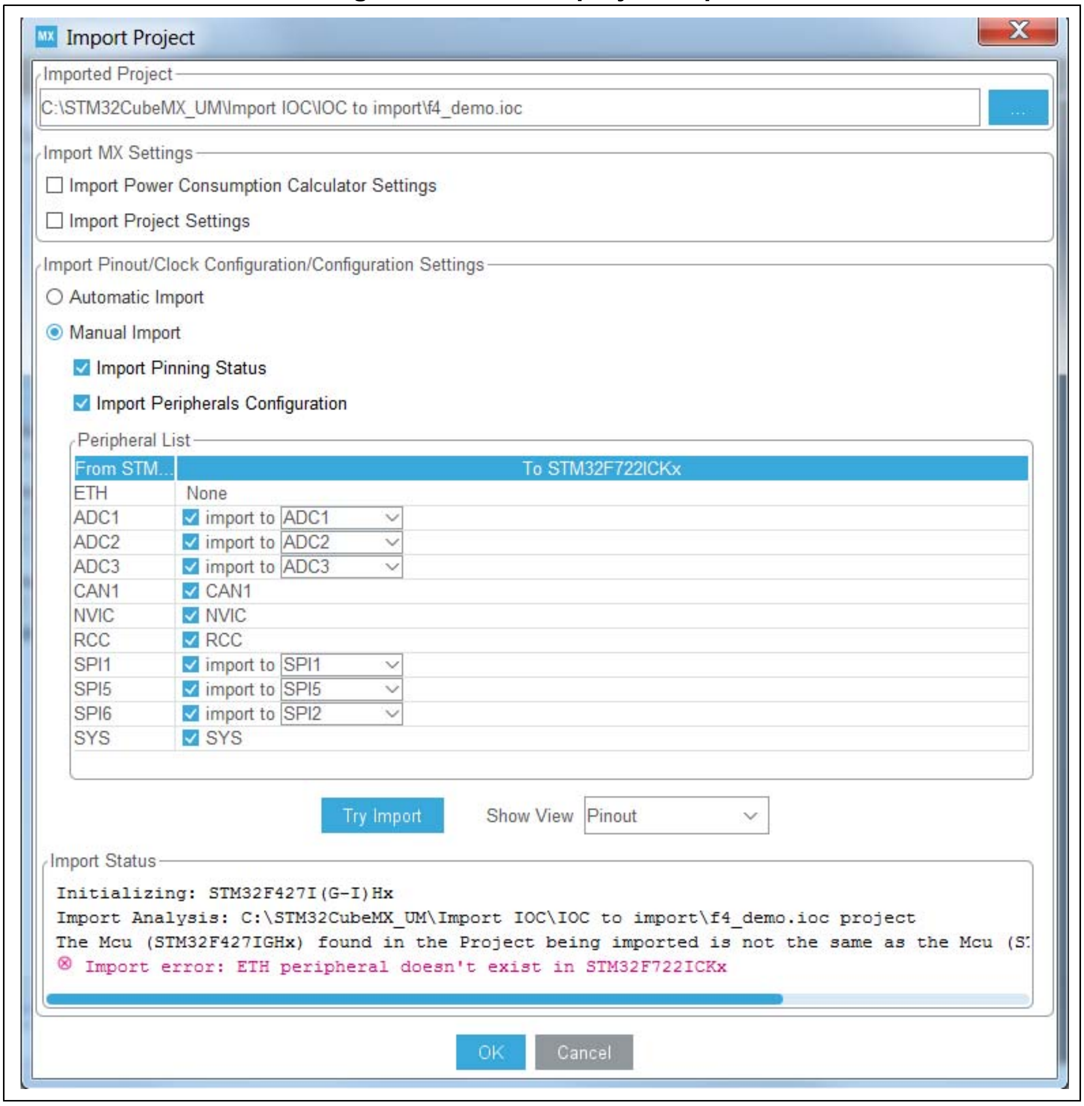
Figure 201. Automatic project import



b) Manual project import

In this case, checkboxes allow the user to select manually the set of peripherals (see *Figure 202*). Select the **Try Import** option to attempt importing.

Figure 202. Manual project import



The Peripheral List indicates:

– The instances configured in the project to be imported

– The instances, if any exist for the MCU currently selected, to whom the configuration must be imported. If several instances are candidate for the import, the user must choose one.

Conflicts can occur when importing a smaller package with less pins or a lower-end MCU with less peripheral options.

Click the **Try Import** button to check for such conflicts: the Import Status window and the Peripheral list get refreshed to indicate errors (see *Figure 203*), warnings, and whether the import has been successful or not:

– Warning icons indicate that the user has selected a peripheral instance more than once, and that one of the import requests will not be performed.

– A cross sign indicates that there is a pinout conflict, and that the configuration cannot be imported as such.

The manual import can be used to refine import choices and resolve the issues raised by the import trial. *Figure 204* gives an example of a successful trial, obtained by deselecting the import request for some peripherals.

The **Show View** function allows switching between the different configuration tabs (pinout, clock tree, peripheral configuration) to check the impact of the “Try Import” action before actual deployment on the current project (see *Figure 204*).

Figure 203. Import Project menu - Try Import with errors

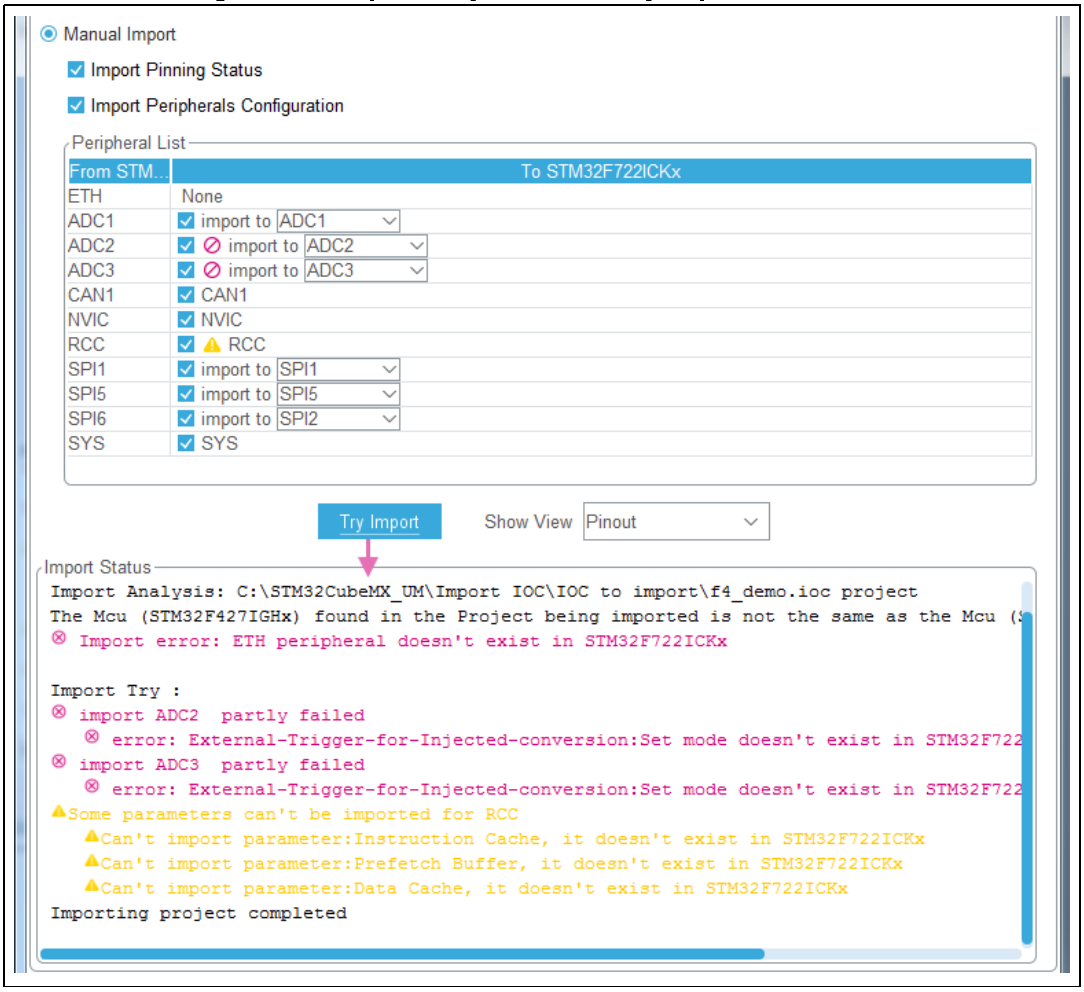
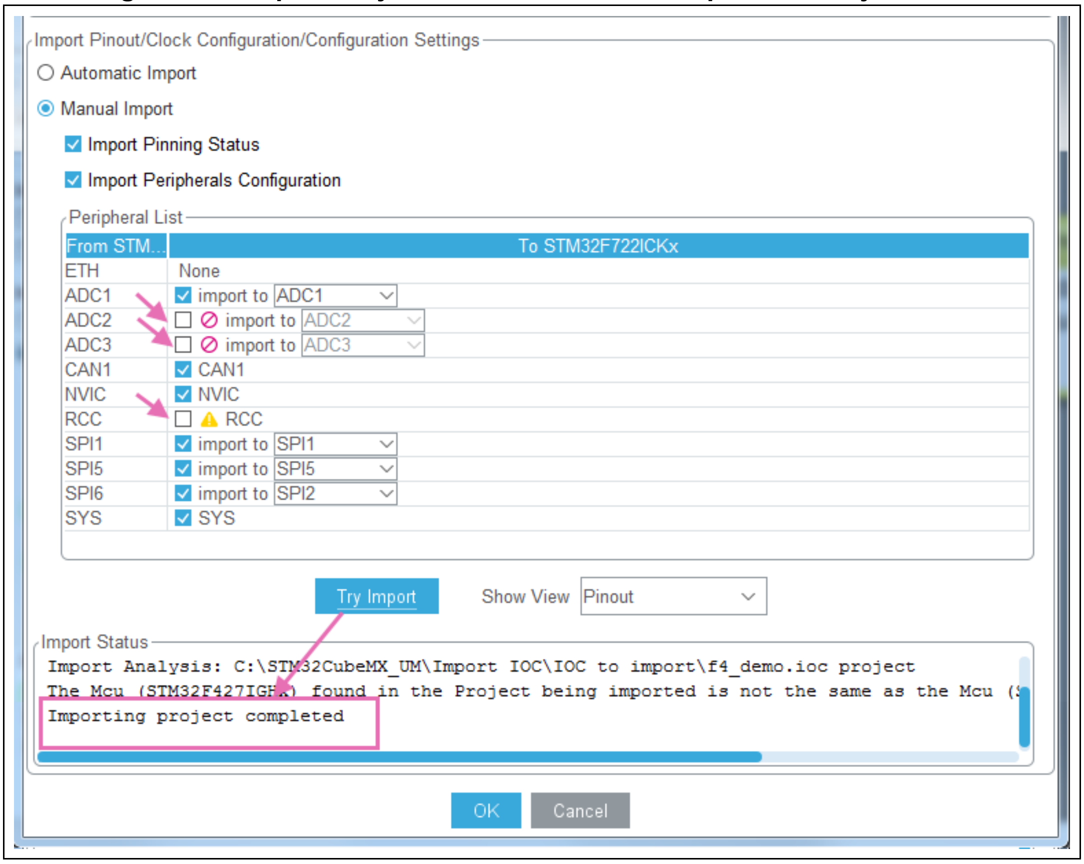


Figure 204. Import Project menu - Successful import after adjustments



3. Choose **OK** to import with the current status or **Cancel** to go back to the empty project without importing.

   Upon import, the Import icon is grayed while the MCU is configured, and it is no longer possible to import a non-empty configuration.

4.12.2 Import Project feature for STM32MPU projects

The pin-to-pin feature is a powerful tool for managing pin configurations across STM32MP2x designs. It is possible to transfer a complete pinout setup from one project (source MPU) to another (target MPU). This feature is tailored for STM32MP2x products supporting the VFBGA 361 10x10 mm package (part numbers ending with ALx).

Devices must share the same third digit after “MP”: as an example if source is STM32MP211AAL3, target third digit must be 1.

The supported bidirectional pin-to-pin compatibilities (under the above conditions) are STM32MP21 ↔ STM32MP25 and STM32MP21 ↔ STM32MP23. This is because these devices are pin-to-pin compatible in both directions. They can be interchanged without modifications to the pin connections (except some particular cases).

Compatibility is strictly limited to the explicitly mentioned conversions and conditions. The boot mode must be the same between the source and target MPUs. For more check the following table.

Table 14. Pin-to-pin compatibility

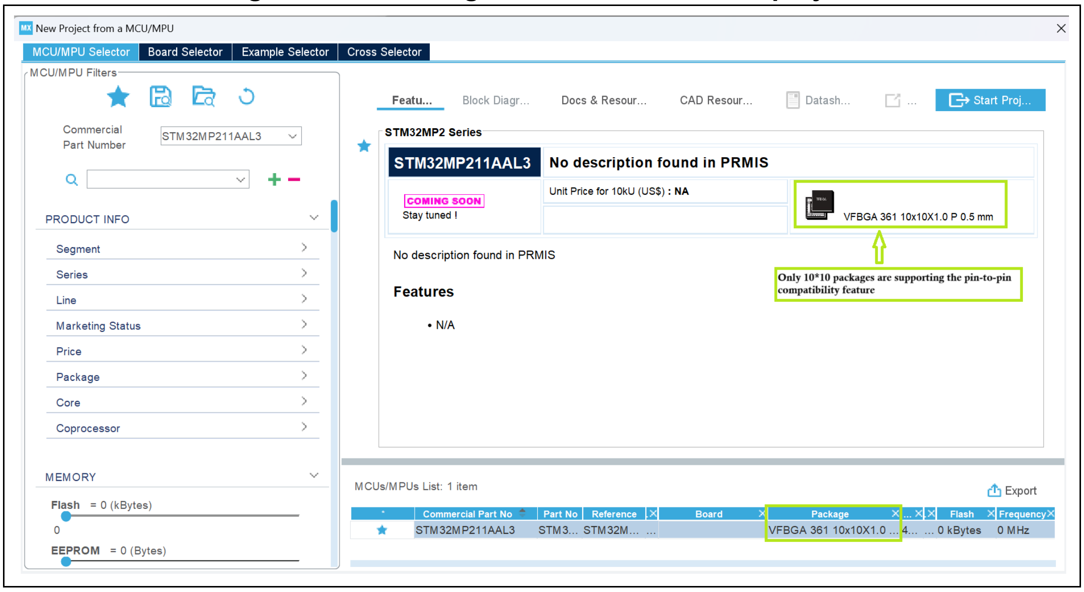
STM32MP23/5 (source)	STM32MP21 (target)
STM32MP257xx	NA
STM32MP255xx	STM32MP215xx
STM32MP253xx	STM32MP213xx
STM32MP251xx	STM32MP211xx
STM32MP237xx	NA
STM32MP235xx	STM32MP215xx
STM32MP233xx	STM32MP213xx
STM32MP231xx	STM32MP211xx

The pin-to-pin compatibility is accessible through **File** > **Import Project Menu**.

Example

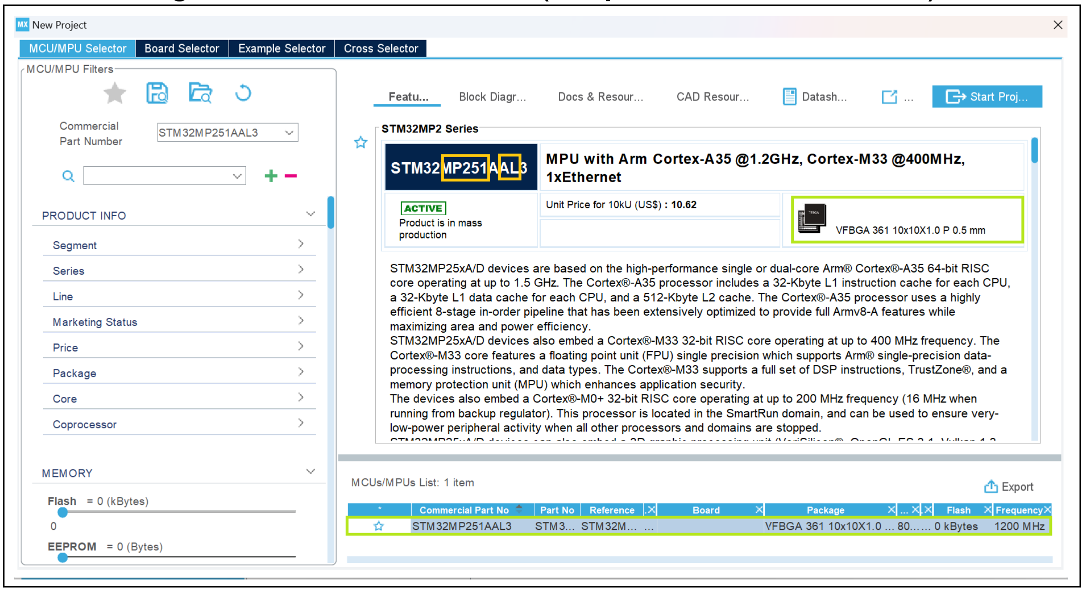
1. Create a new project using an STM32MP2 product supporting the pin-to-pin feature.

Figure 205. Creating an STM32MP211AAL3 project



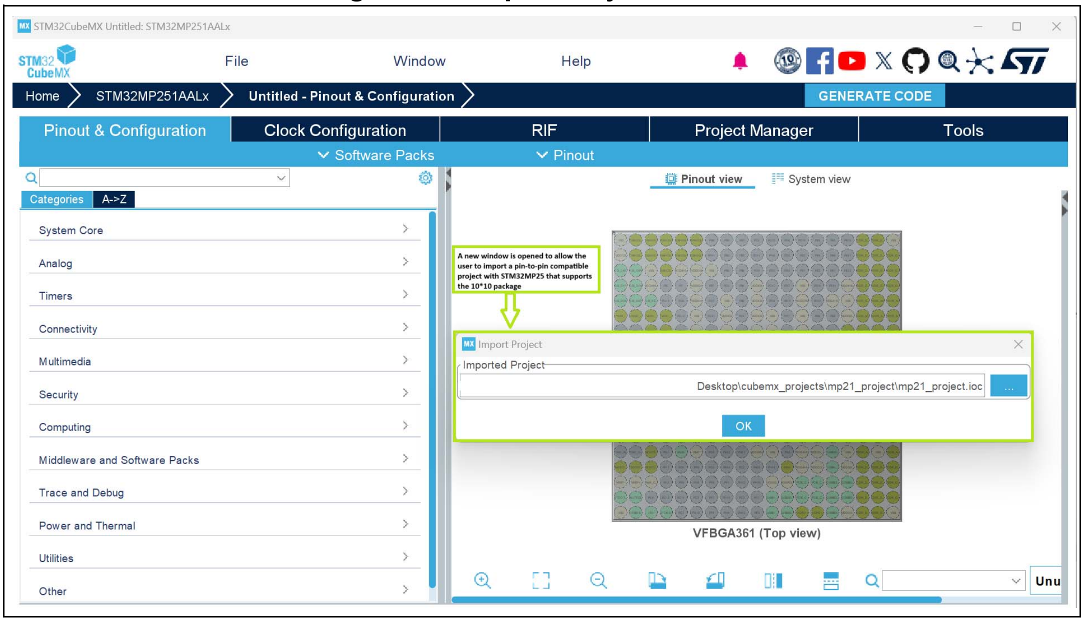
2. Perform configuration (such as activate an IP, set its mode and parameters), and save this initial project.
3. Start a second project using an STM32MP25 or STM32MP23 product associated with the initial product (refer to the above conversions rules and to *Table 14*).

Figure 206. STM32MP251 MPU (compatible with STM32MP211)



4. Navigate to **File > Import Project**, then click **Import Project**.

Figure 207. Import Project window



5. Select the previously saved project file from your local drive and click OK.

Figure 208. Importing the STM32MP211 project

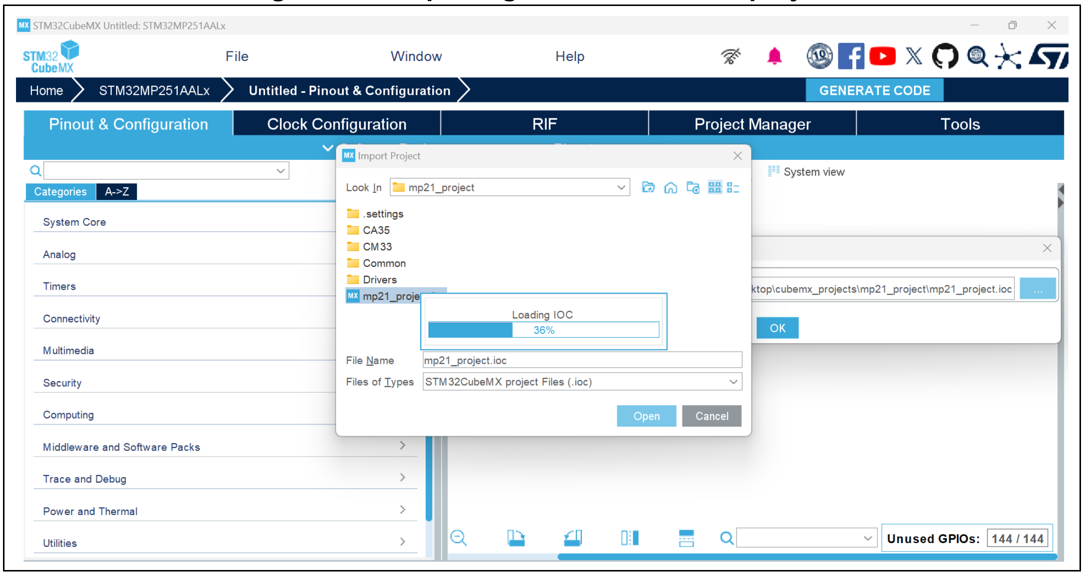
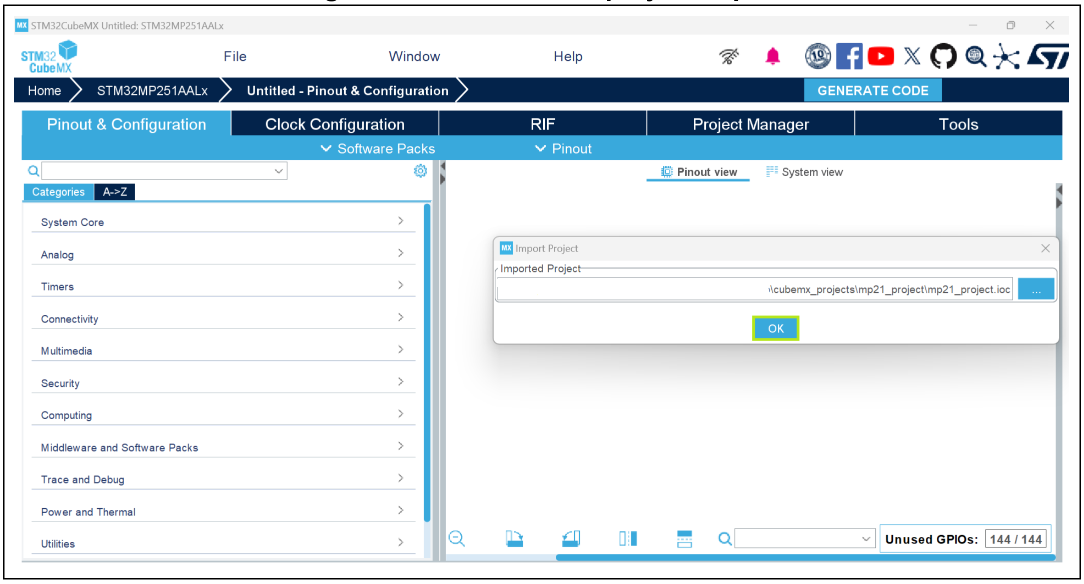


Figure 209. Confirm the project import



The pinout view is updated with the first project configuration, and the “Pin to pin compatibility” console appears, in case of warnings detection.

Figure 210. Pinout view after the project import

Usually IP names, signals, and modes match, but there can be some differences.

IP instances name, signals and pins

During conversion from STM32MP23/25 projects to STM32MP21 projects there can be a reduction in the IP instances. As an example, OCTOSPIM is removed, while, during conversion from STM32MP21 to STM32MP23/MP25, OCTOSPIM is set with a non-multiplexed configuration.

Table 15. Available IP instances

STM32MP25	STM32MP23	STM32MP21
FDCAN1, FDCAN2, FDCAN3		FDCAN1, FDCAN2
ADC1, 2, 3		ADC1, 2 (ADC3 renamed ADC2)
I2C1, 2, 3, 4, 5, 6, 7, 8		I2C1, 2, 3 (I2C8 renamed I2C3, removed I2C3, 4, 5, 6, 7)
I3C1, 2, 3, 4		I3C1, 2, 3 (I3C4 renamed I3C3, removed I3C3)
SPI1, 2, 3, 4, 5, 6, 7, 8		SPI1, 2, 3, 4, 5, 6 (removed SPI6, 7, SPI8 renamed SPI6)
Timers		No TIM20
UART4, 5, 7, 8, 9		UART4, 5, 7
USB3DR		USB_OTG_HS
PCIE		-
UCPD1		-

Table 15. Available IP instances (continued)

STM32MP25	STM32MP23	STM32MP21
DSIHOST	DSI	-
OCTOSPI1, OCTOSPI2, OCTOSPIM		OCTOSPI1
LVDS		-
ETHSW		-
VENC		-
VDEC		-
GPU		-
HASH1		HASH1, 2
OTFDEC1, 2		OTFDEC1
LPDMA1		-
ADF1		-
RNG1		RNG1, 2

When transitioning to STM32MP21, there can be a reduction in the number of accessible GPIOs and/or dedicated peripheral pins compared to the STM32MP23/25 products.

The associated signals are lost if an IP with existing signals in STM32MP25 is not found in STM32MP21. A warning appears in the Pin to pin console.

Table 16. Unavailable pins

STM32MP23/25	STM32MP21
PB0, PH6, PZ2, PB4, PI2, PZ4, PB8, PI3, PZ5, PF14, PI7, PZ6, PG6, PI9, PZ7, PH2, PI10, PZ8, PH3, PI11, PZ9	Not available
VDDA18COMBOPHY, VDDCOMBOPHY	
COMBOPHY_TX1P, COMBOPHY_RX1P, COMBOPHY_TX1N, COMBOPHY_RX1N	
COMBOPHY_REXT	
VDDCOMBOPHYTX	
PCIE_CLKINN, PCIE_CLKOUTN	
PCIE_CLKINP, PCIE_CLKOUTP	
VDDPCIECLK	

Modes

When migrating an existing project from the STM32MP23/25 series to an STM32MP21 device and vice versa, it is essential to account for differences in their operating modes. The following figures show differences for the OCTOSPI1 and USB_OTG_HS peripherals.

Figure 211. Modes matching: OCTOSPI1, STM32MP211 to STM32MP251

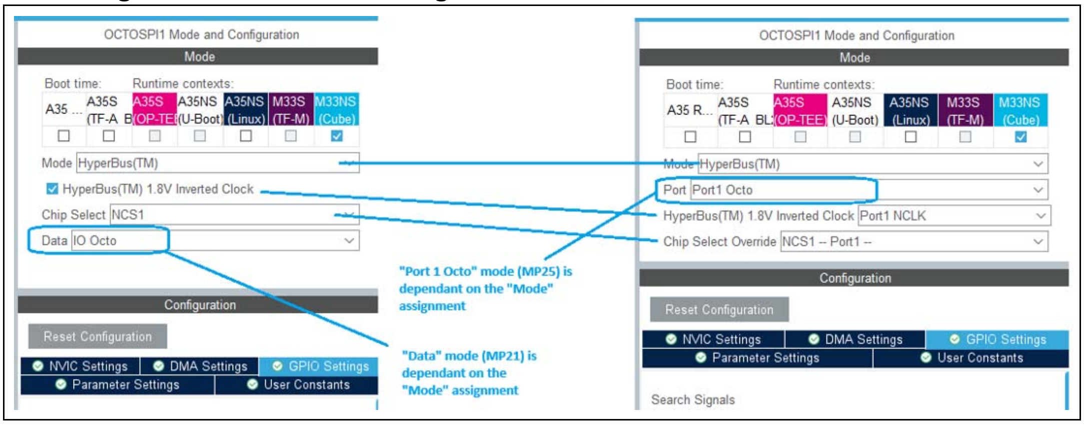


Figure 212. Modes matching: USB_OTG_HS, STM32MP211 to STM32MP251

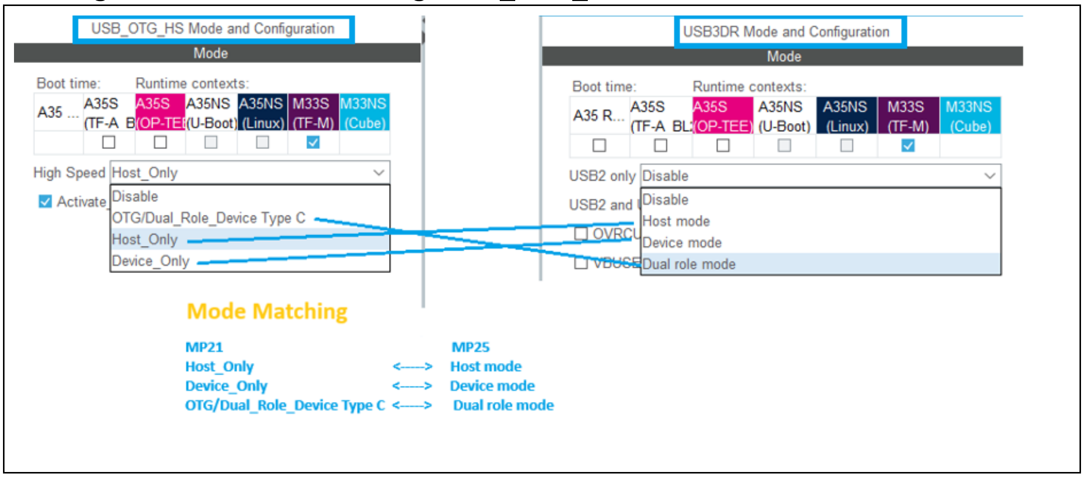


Figure 213. Modes matching: USB_OTG_HS, STM32MP211 to STM32MP251, mode TXRTUNE)

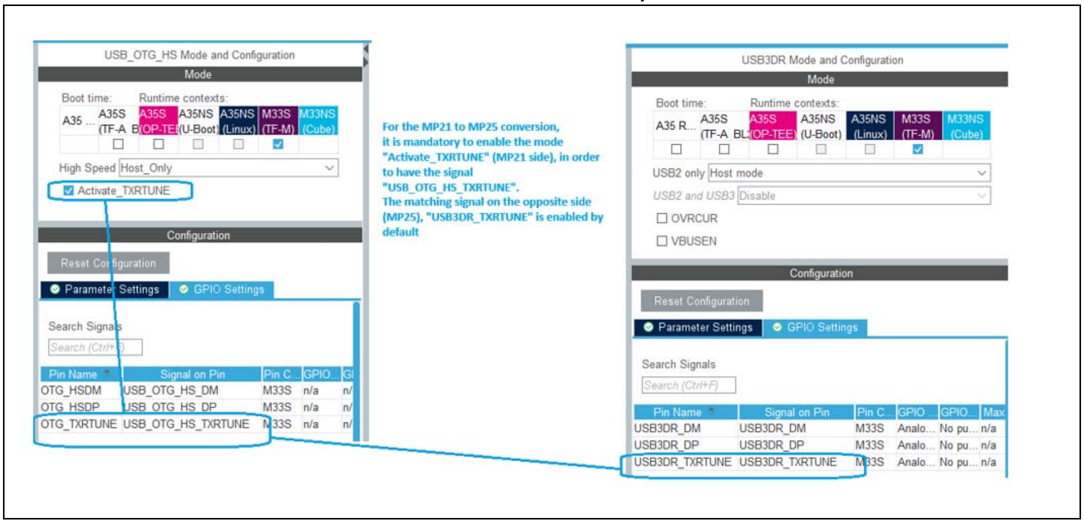
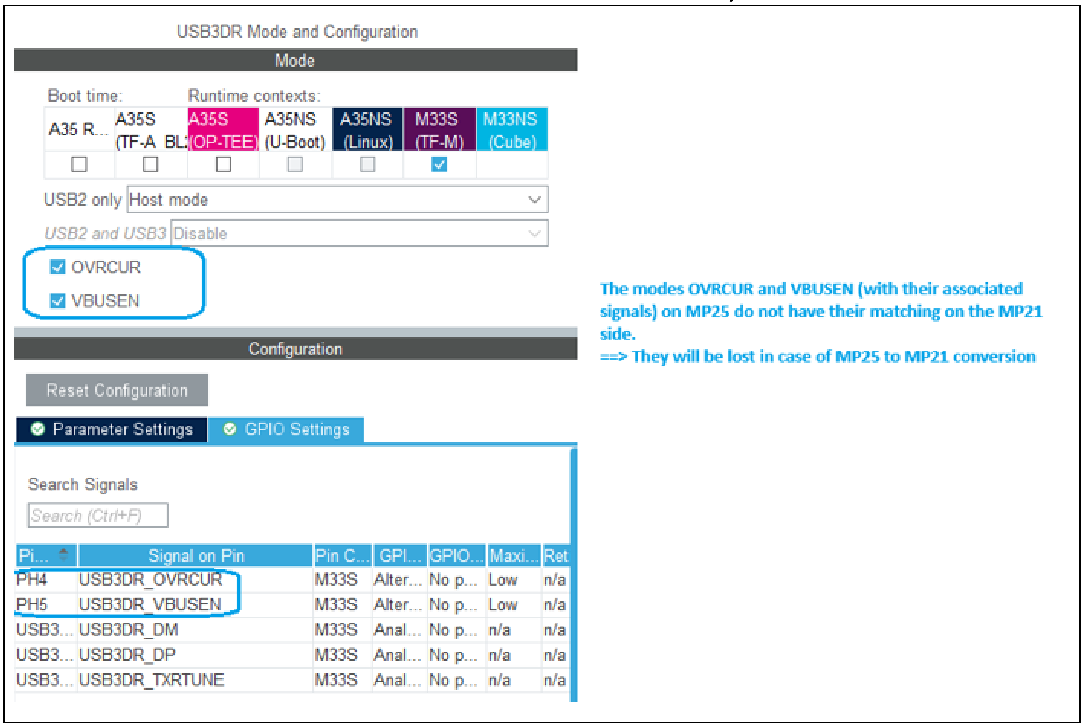


Figure 214. Modes matching: USB_OTG_HS, STM32MP251 to STM32MP211, modes OVRCUR and VBUSEN)



4.13 Set unused/reset used GPIOs windows

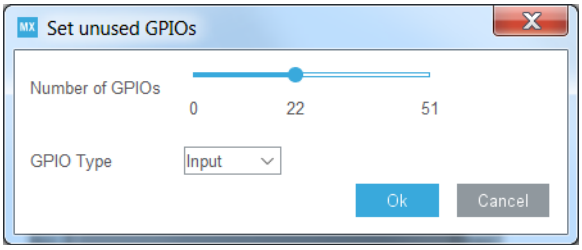
These windows are used to configure in the same GPIO mode several pins at the same time.

To open them:

- Select **Pinout > Set unused GPIOs** from the STM32CubeMX menu bar.

Note: The user selects the number of GPIOs and lets STM32CubeMX choose the actual pins to be configured or reset, among the available ones.

Figure 215. Set unused pins window



- Select **Pinout > Reset used GPIOs** from the STM32CubeMX menu bar.

  Depending whether the Keep Current Signals Placement option is checked or not on the toolbar, STM32CubeMX conflict solver is able to move or not the GPIO signals to other unused GPIOs:
  – When Keep Current Signals Placement is off (unchecked), STM32CubeMX conflict solver can move the GPIO signals to unused pins in order to fit in another peripheral mode.
  – When Keep Current Signals Placement is on (checked), GPIO signals is not moved and the number of possible peripheral modes is limited.

  Refer to *Figure 217* and *Figure 218* and check the limitation(s) in available peripheral modes.

Figure 216. Reset used pins window

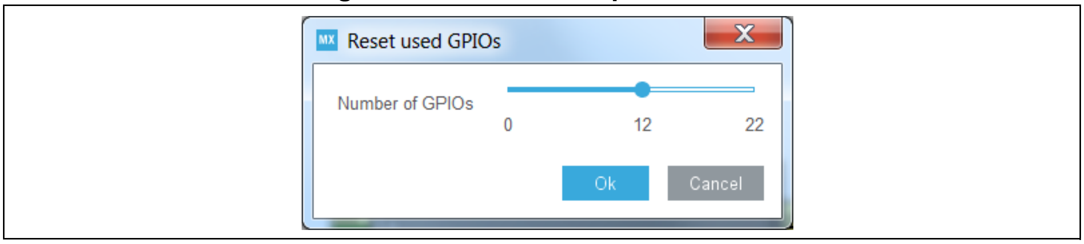


Figure 217. Set unused GPIO pins with Keep Current Signals Placement checked

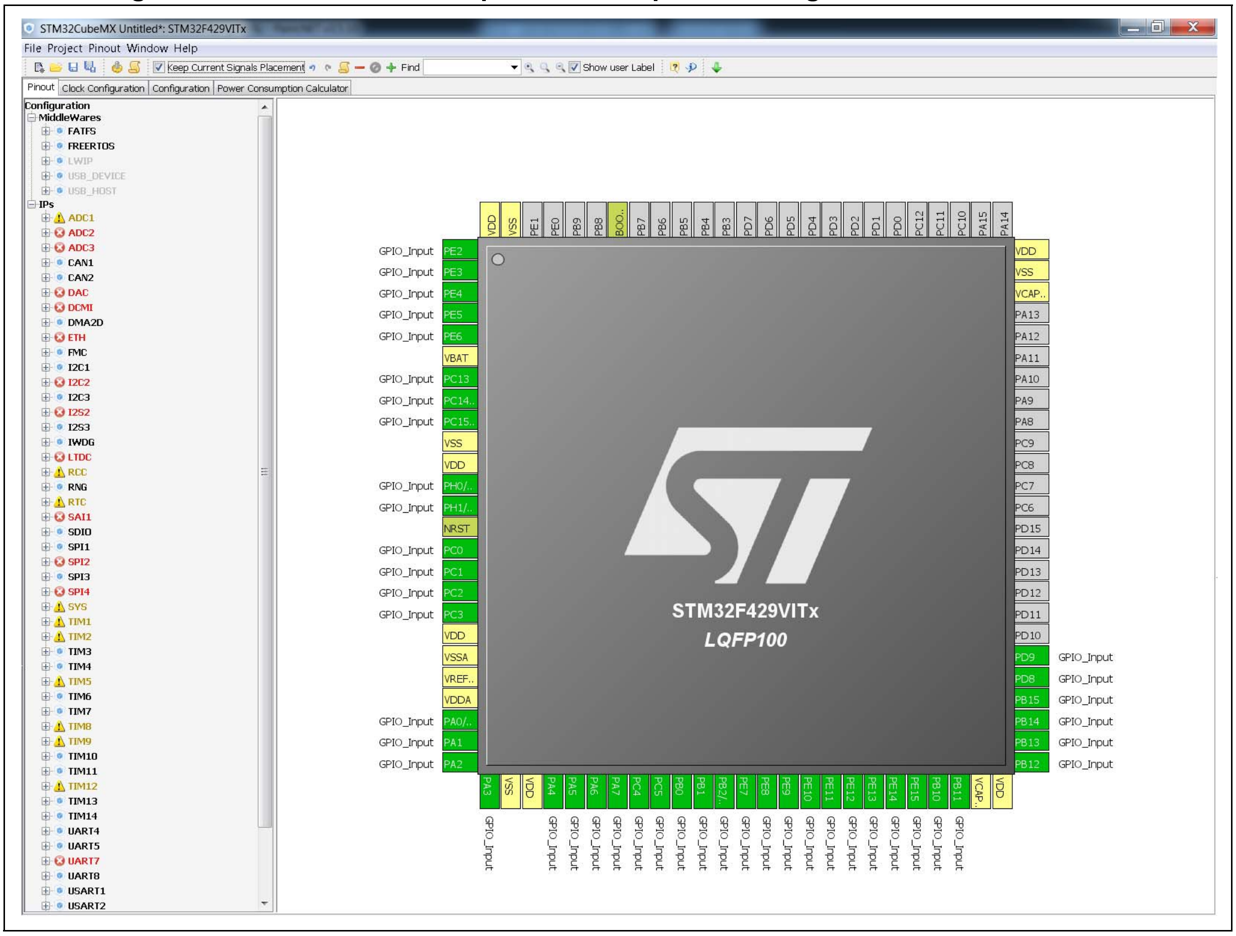
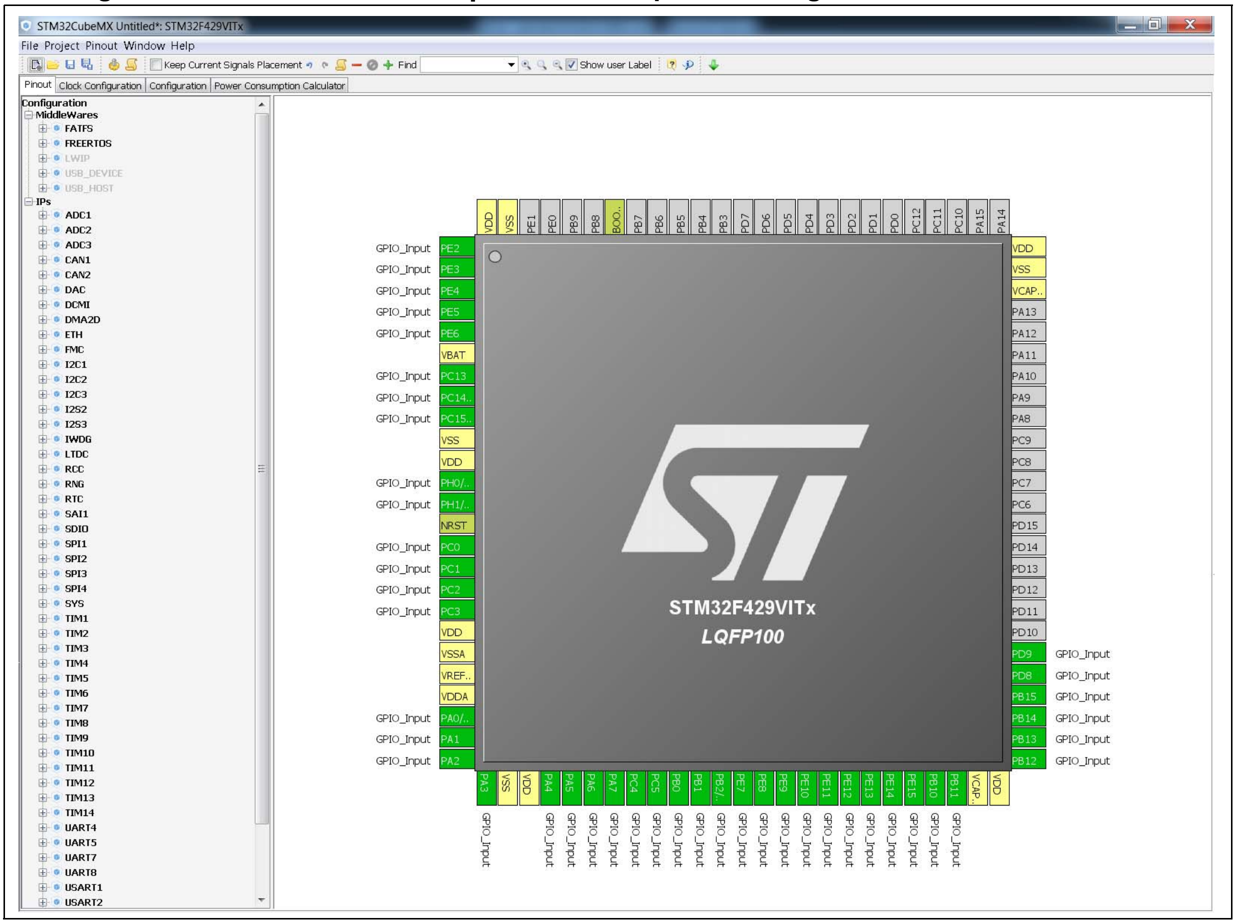


Figure 218. Set unused GPIO pins with Keep Current Signals Placement unchecked



4.14 Update Manager windows

Three windows can be accessed through the **Help** menu available from STM32CubeMX menu bar:

1. Select **Help > Check for updates** to open the **Check Update Manager** window and find out about the latest software versions available for download.
2. Select **Help > Manage embedded software packages** to open the **Embedded Software Package Manager** window and find out about the embedded software packages available for download. It also allows checking for package updates and removing previously installed software packages.
3. Select **Help > Updater settings** to open the **Updater settings** window and configure update mechanism settings (proxy settings, manual versus automatic updates, repository folder where embedded software packages are stored).

Refer to *Section 3.4* for a detailed description of these windows.

4.15 Software Packs component selection window

This window can be opened by clicking **Middleware and Software Packs** from the **Pinout & Configuration** tab, at any time when working on the project. It allows the user to select Software Packs components for the current project. It features four panels, as shown in *Figure 219*:

- **Filters** panel

  Can be hidden using the “Show/hide filters” button. It is located on the left side of the window and provides a set of criteria to filter the pack component list.
- **Packs** panel

  Main panel, displays the list of software components per pack that can be selected.
- **Component dependencies** panel

  Can be hidden using the “Show/hide dependencies” button. It displays dependencies, if any, for the component selected in the packs panel. It proposes solutions when any is found.

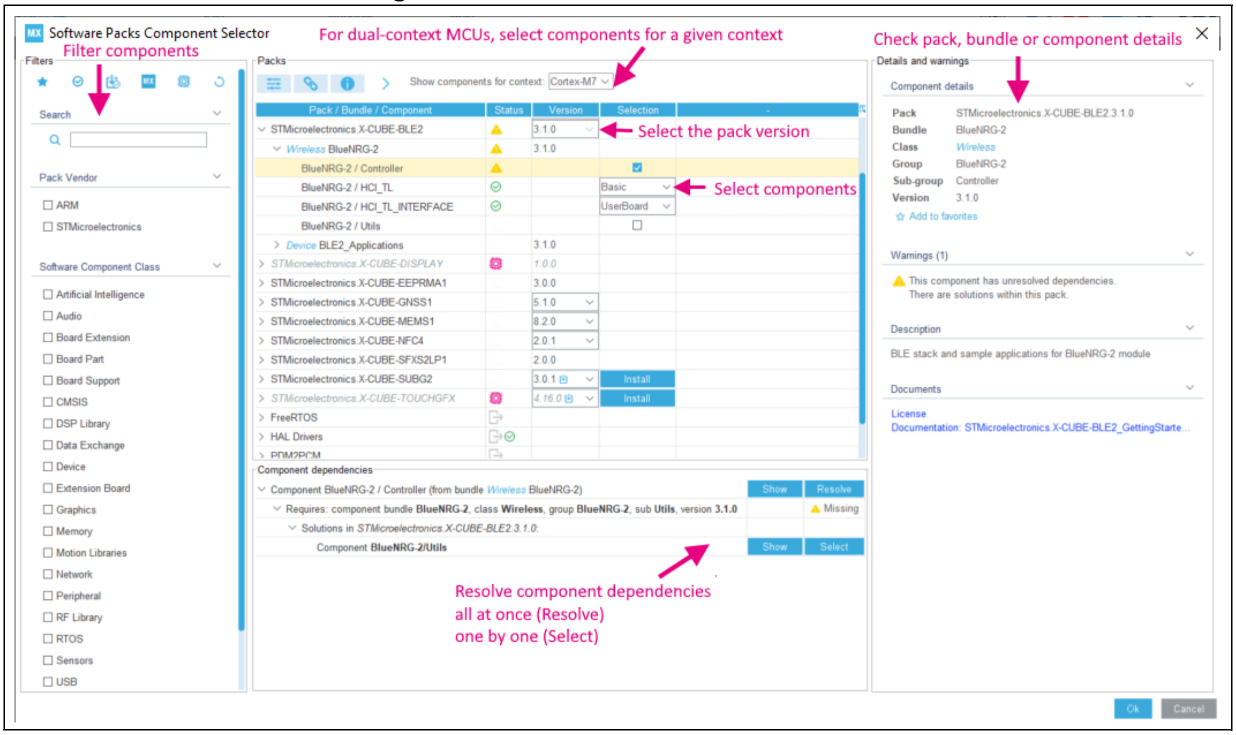
  Dependencies that are not solved are highlighted with fuchsia icons.

  Once the dependency is solved (by selecting a component among the solution candidates) it is highlighted with green icons.
- **Details and warnings** panel

  Can be hidden using the “Show/hide details” button. It is located on the right hand side. It provide informations for the element selected in the Pack panel.

  This element can be a pack, a bundle or a component. It offers the possibility to install a version of the pack available but not yet installed, and allows the user to migrate the current project to a newer version of the pack, raising incompatibilities that cannot be automatically resolved.

Figure 219. Additional software window



See *Section 10* for more details on how to handle additional software components through STM32CubeMX CMSIS-Pack integration.

4.15.1 Introduction on software components

Arm® Keil™ CMSIS-Pack standard defines the pack (*.pdsc) format for software components to be distributed as Software Packs. A Software pack is a zip file containing a *.pdsc description file.

STM32CubeMX parses the pack .pdsc file to extract the list of software components. This list is presented in the Packs panel.

Arm® Keil™ CMSIS-Pack standard defines a software component as a list of files. The component or each of the corresponding individual files can optionally refer to a condition that must resolve to true, otherwise the component or file is not applicable in the given context. These conditions are listed in the **Component dependencies** panel.

There are no component names. Instead, each component is uniquely identified for a given vendor pack by the combination of class name, group name and a version. Additional categories, such as sub-group and variant can be assigned. These details are listed in the **Details & Warnings** panel.

4.15.2 Filter panel

Click on to open the Filter panel

To filter the software component list, choose pack vendor names and software component classes or enter a text string in the search field.

The resulting software component table is collapsed. Click the left arrow to expand it and display all the components that match the filtering criteria.

Table 17. Additional software window - Filter icons

Icon	Description
	Show only favorite packs. A pack is set as favorite in the **Details and Warnings** panel by clicking Add to favorites
	Show only selected components. Components are selected in the Packs panel through checkboxes or variant selection when several implementation choices are available for the same component.
	Show only installed packs. Enables to show or hide not yet installed packs. Not yet installed packs are distinguished with the icon
MX	Show only packs compatible with this version of STM32CubeMX. Packs not compatible with this version are distinguished with the icon
	Show only packs compatible with the MCU used for the current project.
	Reset all filters

4.15.3 Packs panel

By default, the Packs panel shows a collapsed view: all known packs are displayed with their name and for one given version (latest version is the default). Icons are used only to highlight the status of a pack version or of a component (see Table Packs panel icons).

Details and warnings and **Component dependencies** panels are used to provide detailed information.

The default view can be expanded by clicking the left arrows, revealing the next level, which can be a Bundle or a top component. The lowest level is the component level.

From this panel, clicking an icon highlighting a limitation or an action opens the relevant secondary panel (Details & Warnings or Component Dependency resolution).

Note: Some packs can have conditions on Arm® cores or STM32 products, visible only when the selected MCU meets the criteria. For example, a pack stating the "<accept Dcore="Cortex-M4"/>" condition shows up, but is grayed for MCUs without Cortex®-M4 core.

Note: A pack may promote an API and be shown under the "exposed APIs" entry. Clicking the API name allows to display additional information in the Details & warnings panel. Selecting the component implementing the API selects the API itself. STM32CubeMX generates the project with both the API .h definition file and the API implementation .c file.

Note: Some components, highlighted in gray in the component panel, are shown as read-only. They are software components (HAL peripheral drivers or middleware offers) coming with STM32Cube MCU embedded software package and are natively available in STM32CubeMX.

Table 18. Additional Software window – Packs panel columns

Column name	Description
Pack/Bundle/Component	At pack level, shows the <name of the Software pack> At bundle level, shows the <Name of the Class>_<Bundle name, if any> At component level, shows the <Group name>/<Subgroup name, if any>. Class names are standardized by the Arm CMSIS standard[1]
Version	Shows the version that has been selected from a list of one or more available versions of a pack. Bundle and components can either inherit the version of the pack or have their own specific version. The version is shown in the **Details and Warning** panel.
Selection	Selects a component through a checkbox when only one implementation is available, or from a list if variants exist.

1. The Arm® Keil™ CMIS-Pack website, http://www.keil.com, lists the following classes:
- Data Exchange: Software components for data exchange
- File System: File drive support and file system
- Graphics: Graphic libraries for user interfaces
- Network: Network stack using Internet protocols
- RTOS: Real-time operating systems
- Safety: Components for testing application software against safety standards
- Security: Encryption for secure communication or storage
- USB: Universal serial bus stack
- Wireless: Communication stacks such as Bluetooth®, WiFi®, and ZigBee®.

Table 19. Additional Software window – Packs panel icons

Icon	Description
	The pack has been added to the user favorite list of packs. Use the **Details and Warnings** panel to add/remove packs from list of favorites.
	The pack version is not compatible with this STM32CubeMX version. Solution: select a compatible version.

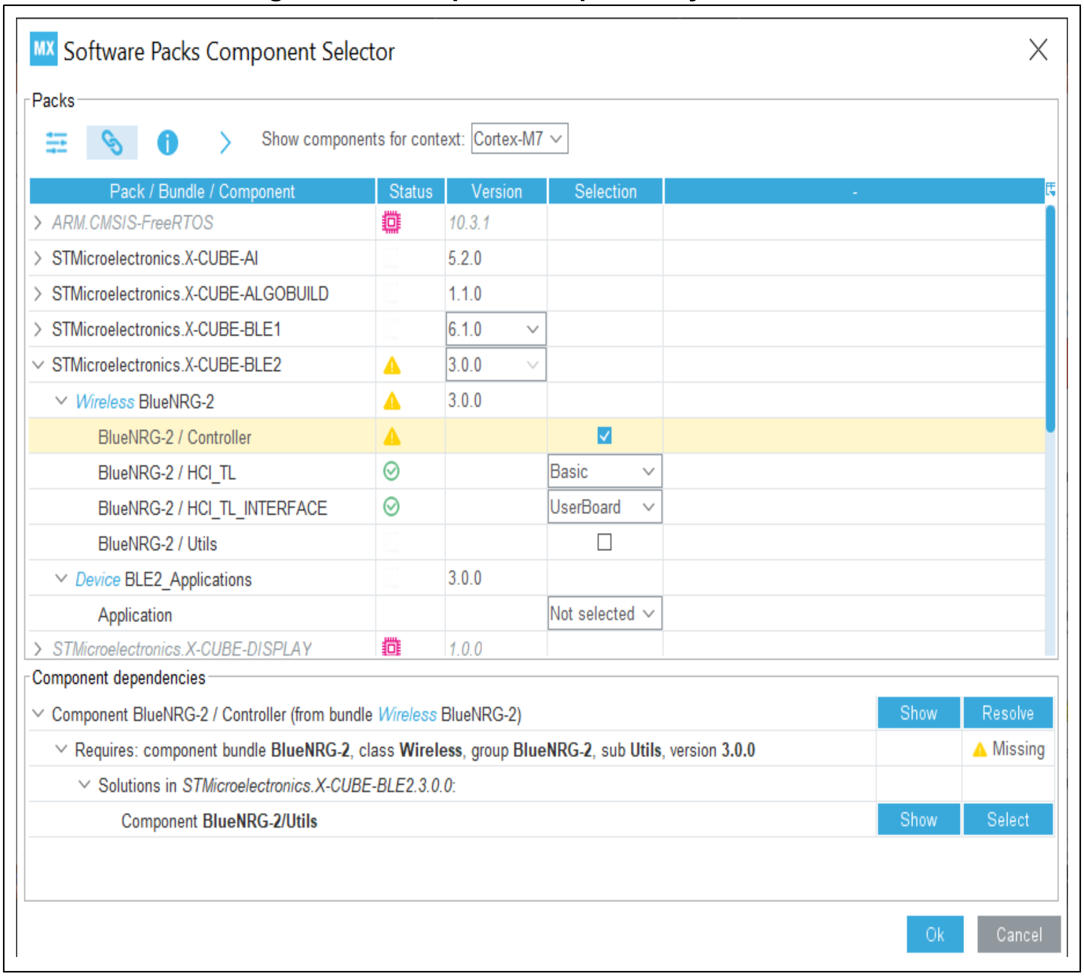
Table 19. Additional Software window – Packs panel icons (continued)

Icon	Description
	The pack version is not yet installed. Solution: go to the **Details and Warnings** panel to download the pack version to use it for a project.
	The component is not available for selection. Solution: download the pack this component belongs to.
	A component is selected and at least one condition remains to be solved. Select the line of the component with such icon to refresh the **Component dependencies** panel with the list of dependencies, status and solutions if any found.
	At least one component is selected and all conditions, if any, are met.
	Other pack versions are available to switch to. Solution: use the **Details and Warnings** panel to proceed with a change.
	Highlights the components natively available in STM32CubeMX for the currently selected MCU. They correspond to peripheral drivers and middleware stacks. For such components, the dependencies cannot be automatically resolved: go to the STM32CubeMX pinout view and enable the relevant peripheral instance or middleware in the mode panel. They will appear as selected (green checkbox) in the Component Selector.

4.15.4 Component dependencies panel

The conditions are dependency rules applying to a given software component. When a component is selected, it shows with a green icon if there is no dependency to resolve, with a warning icon otherwise. Click to open the dependency panel (see *Figure 220*).

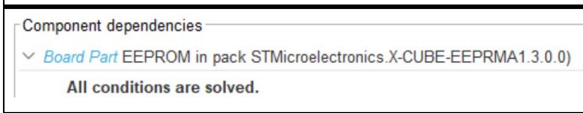
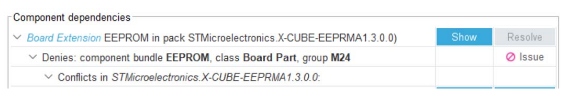
Figure 220. Component dependency resolution



The panel is refreshed when selecting a component, providing details on the dependencies to solve and the available solutions, if found (see *Table 20*):

- click the Show button to show the component solving the dependency
- click the Select button to select the component solving the dependency
- when available, click Resolve button to automatically resolve the dependencies.

Table 20. Component dependencies panel contextual help

Contextual help	Description
	No dependency to solve.
	Dependency to solve but issue encountered (no solution found or conflict).
	Dependency to solve and at least one solution found.

4.15.5 Details and Warnings panel

Click on  to show the panel (see *Figure 221*).

This panel is refreshed upon selecting a line from the **Packs** panel.

The following actions are possible from this panel:

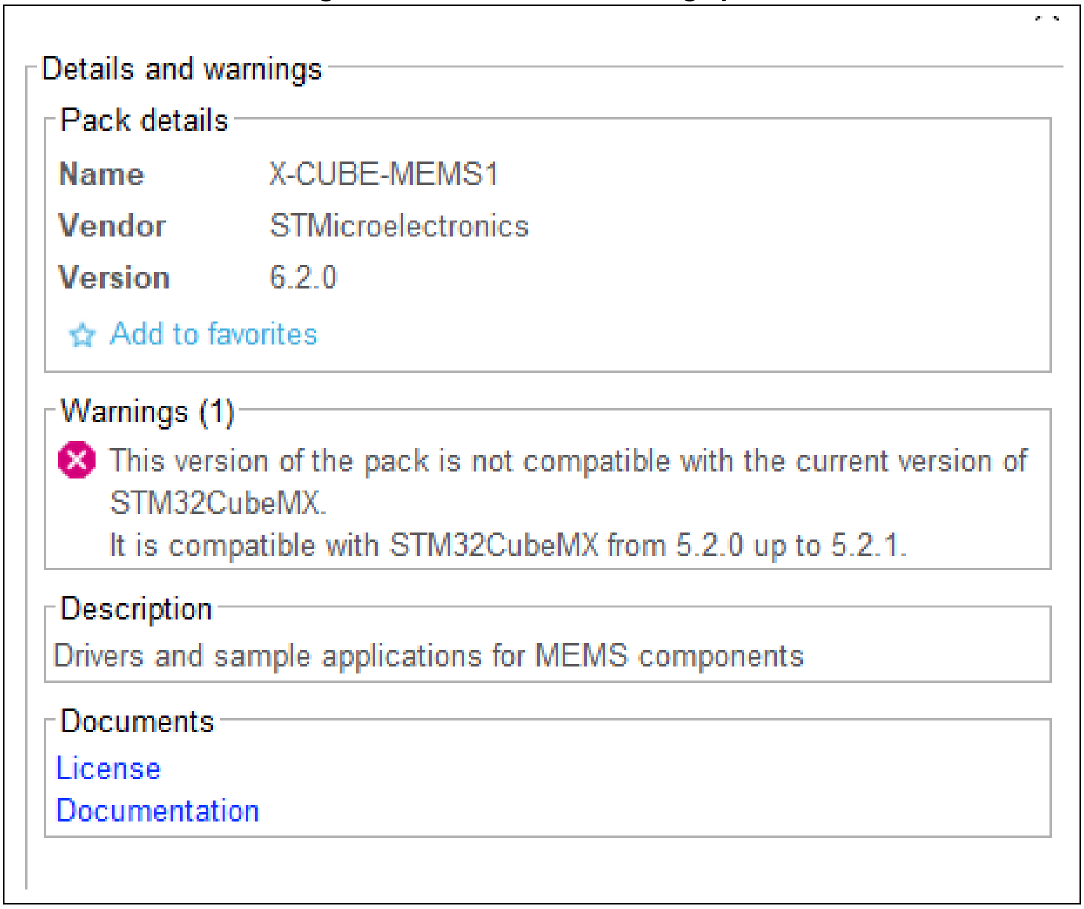
- Add/remove the pack from the list of favorite packs
- Install the pack
- Access the pack documentation through links
- Migrate the project to a new pack version

To migrate a project to a new software pack version:

1. Open the project
2. Migrate to the new pack version
3. Generate the code

Known issue: performing step *2* after step *3* (migrating after code generation) leads to errors (wrong file path generation and project compilation failure). To fix such issue, the project must be saved as new, and the code must be generated again. Actions are possible in this panel, namely adding/removing the pack to/from the list of favorite packs, installing a pack, accessing pack documentation through links.

Figure 221. Details and Warnings panel



4.15.6 Updating the tree view for additional software components

Once the selection of the software components required for the application is complete (see *Figure 222*), click **OK** to refresh STM32CubeMX window: the selected component appears in the tree view under Additional Software (*Figure 223*).

The current selection of additional software components appears in the tree view (see *Figure 223*). The software components must be enabled in the Mode panel and may be configured further if any parameter is proposed in the configuration panel. Hovering the mouse over the component name reveals contextual help with links to documentation.

Figure 222. Selection of additional software components

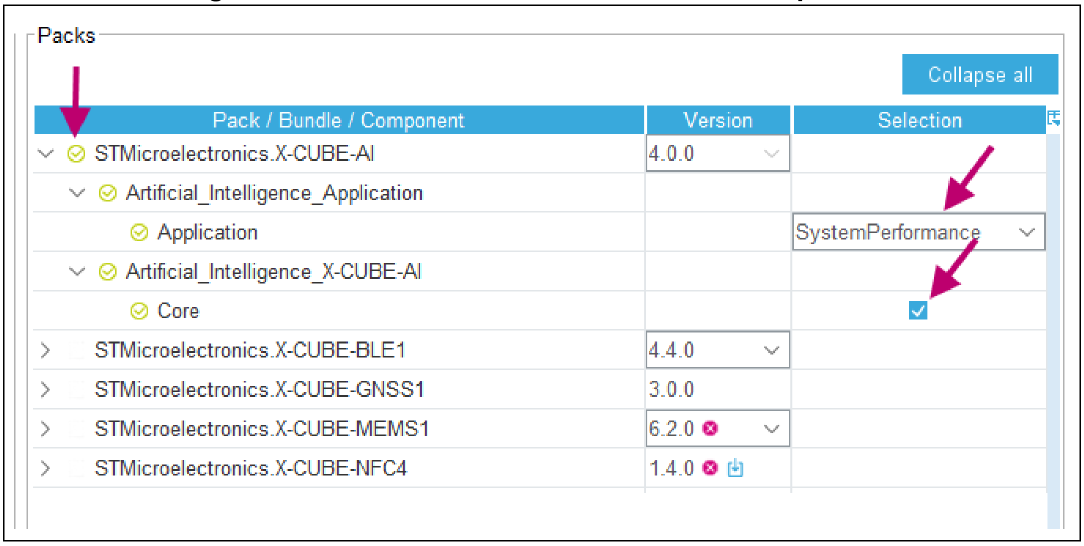
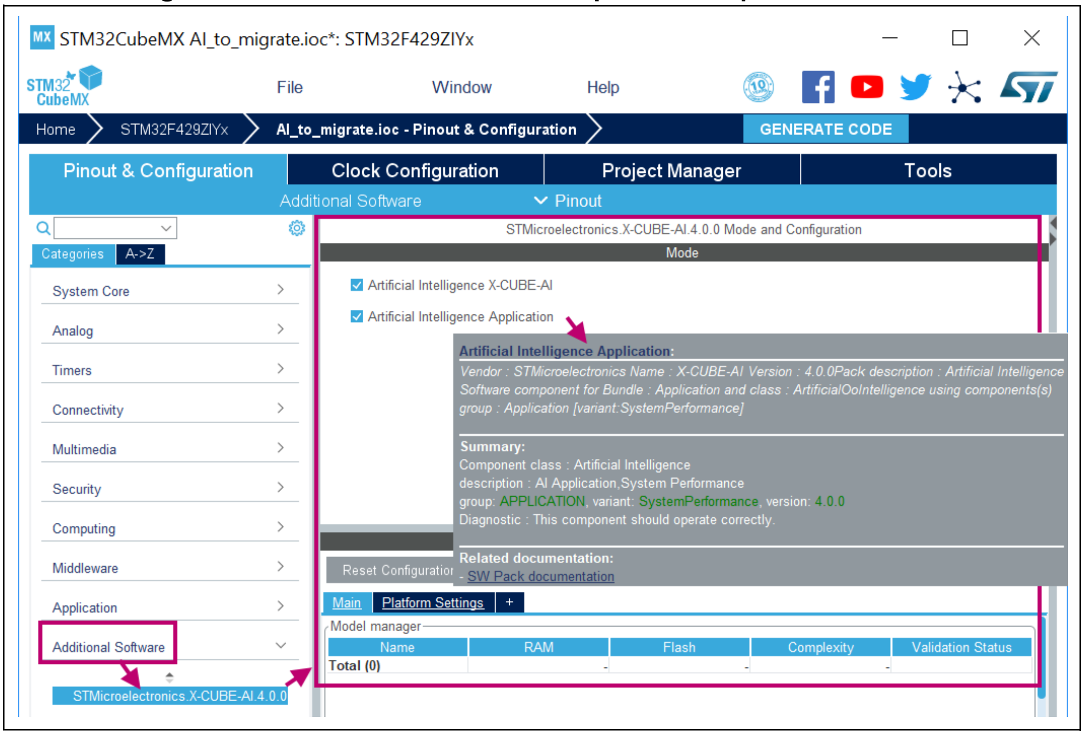


Figure 223. Additional software components - Updated tree view



4.16 LPBAM Scenario & Configuration view

Starting with STM32CubeMX 6.5.0, for projects without TrustZone activated and on the STM32U575/585 product line, users can optionally create LPBAM applications using the LPBAM Scenario & Configuration view (see *Figure 224*).

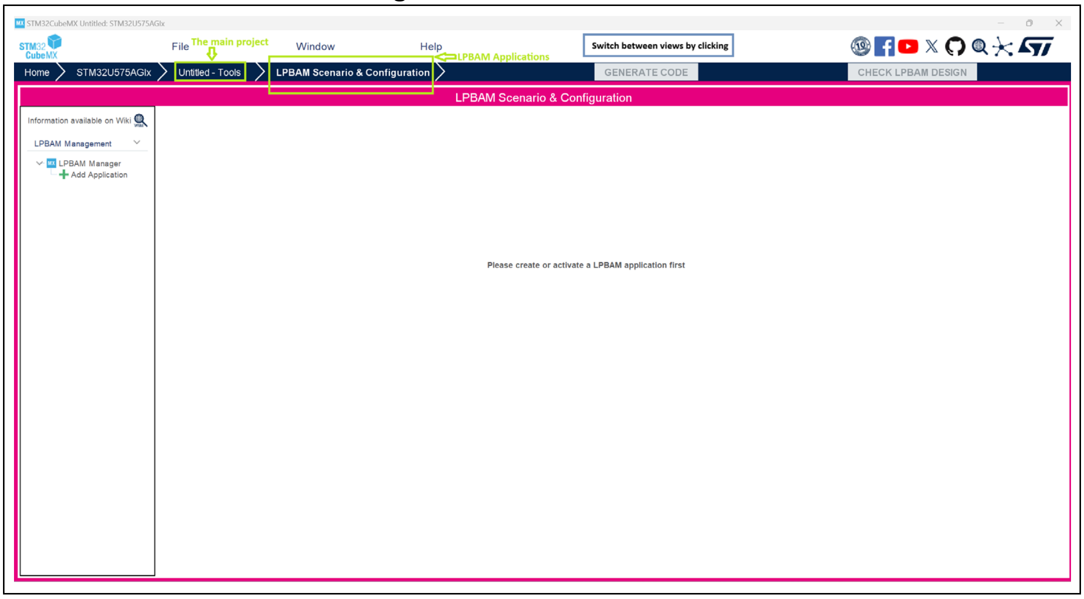
Starting with STM32CubeMX 6.6.0, users can create LPBAM applications for projects with TrustZone activated on the STM32U575/585 product lines.

Thanks to this view it is possible to:

- add/remove LPBAM applications
- for each LPBAM application, create queues
- for each queue, create functional nodes using the LPBAM firmware API available for peripherals on the Smart Run Domain
- for each LPBAM application, configure the pinout, the clock tree, and HAL-related configurations for the peripherals on the Smart Run Domain.

For details on how to work with this view, refer to *Section 18: Creating LPBAM projects*.

Figure 224. LPBAM window



4.17 CAD Resources view

STM32CubeMX CAD Resources view allows the user to quickly access and download schematic symbols, PCB footprints and 3D CAD models for one or more design toolchains. It requires STM32CubeMX to be connected to the Internet.

To configure and check the Internet connection select **Help > Updater settings** to open STM32CubeMX updater settings window.

CAD Resources can be accessed from the MCU Selector window and from STM32CubeMX project view.

Access from MCU selector

- Open the MCU selector from STM32CubeMX homepage
- Select an MCU commercial part number (Marketing status must not be “Coming soon”)
- Select the CAD Resources tab to see the CAD resources (see *Figure 225*).
- Use the slider to go down the panel and access the different resource views (Symbols, Footprint, and 3D models).

Note: For MCU commercial part numbers in “Coming Soon” Marketing status, there are no CAD resources available (see Figure 226).

To select the resources for download (see *Figure 227*)

- Select the design toolchain
- Select the CAD formats
- Accept terms and conditions
- Click to download
- Specify the download location

Figure 225. CAD Resources view

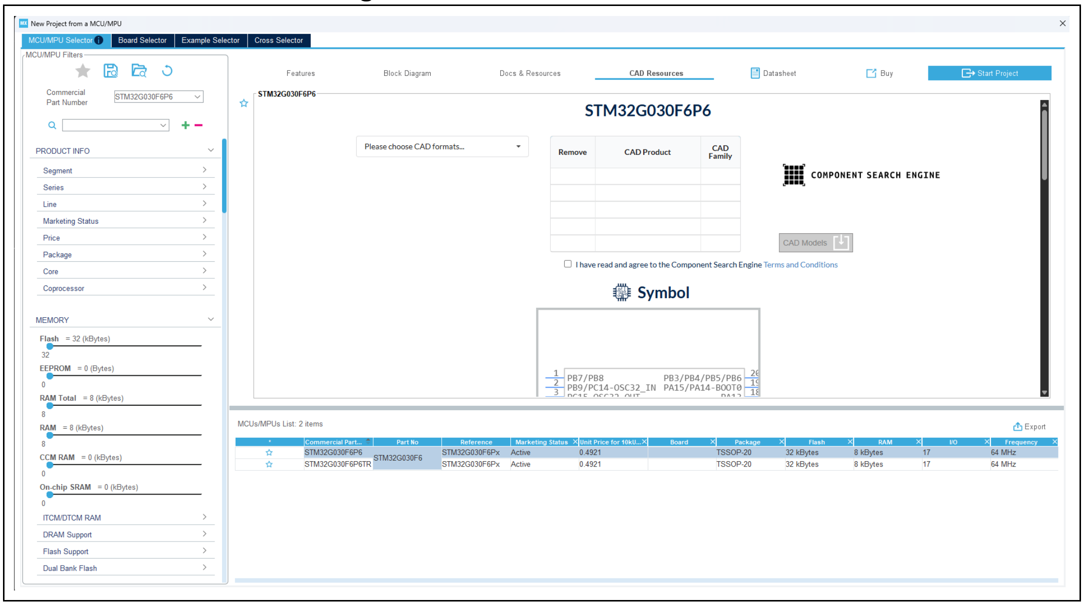


Figure 226. CAD Resources not available

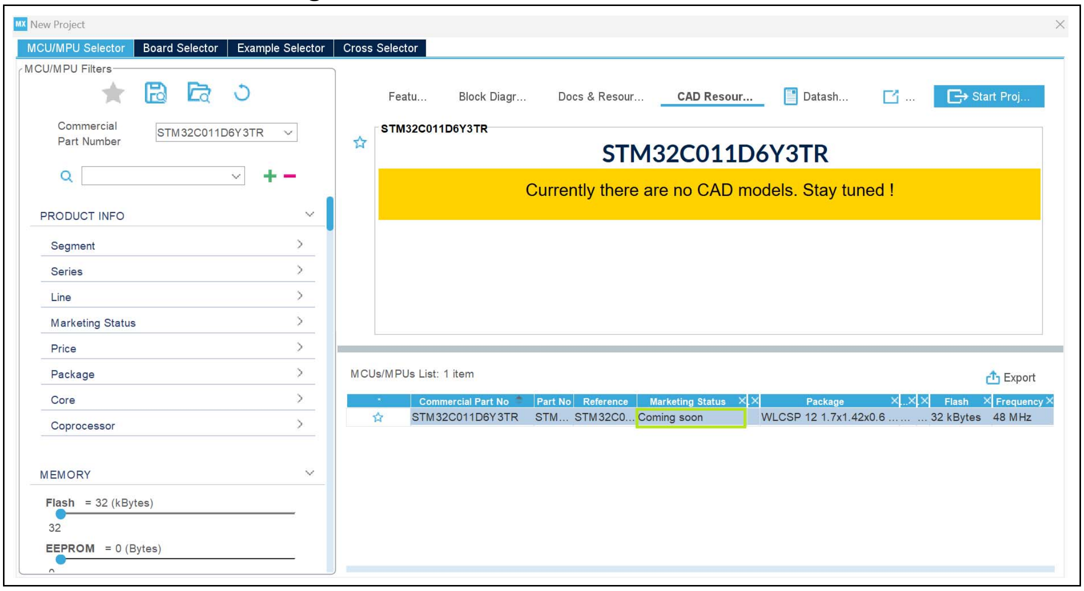
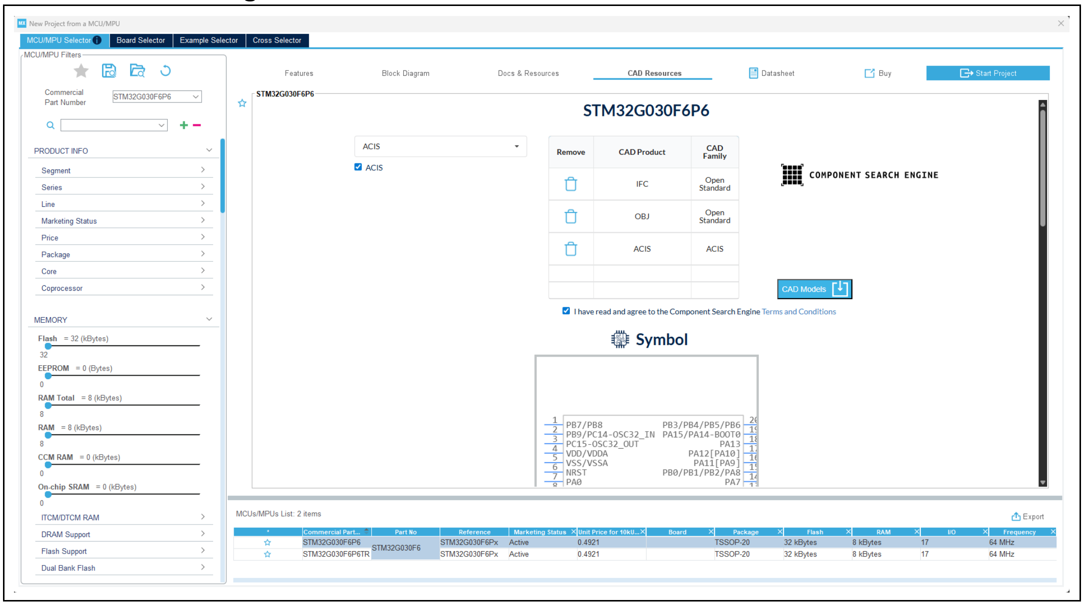


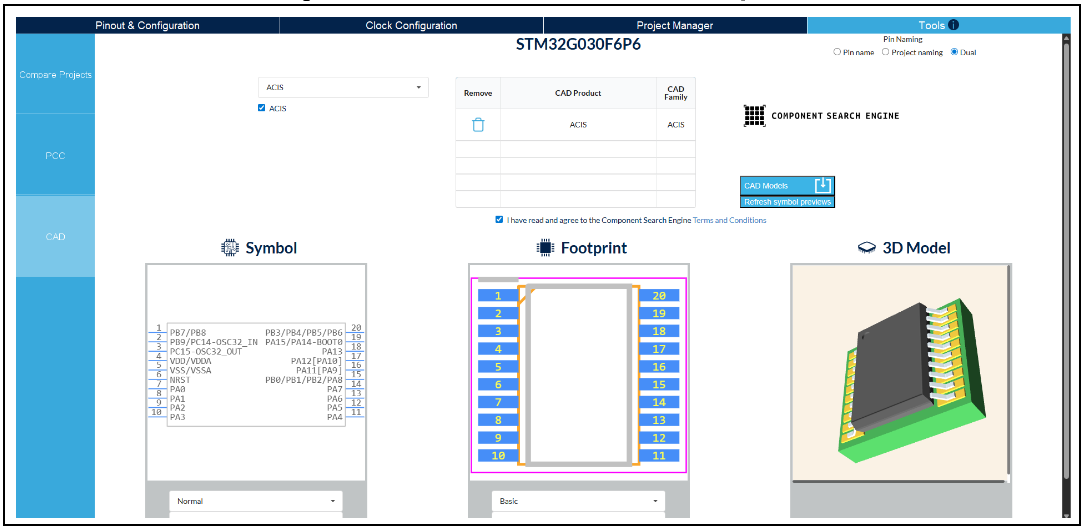
Figure 227. CAD Resources selection for download



Access from STM32CubeMX project view

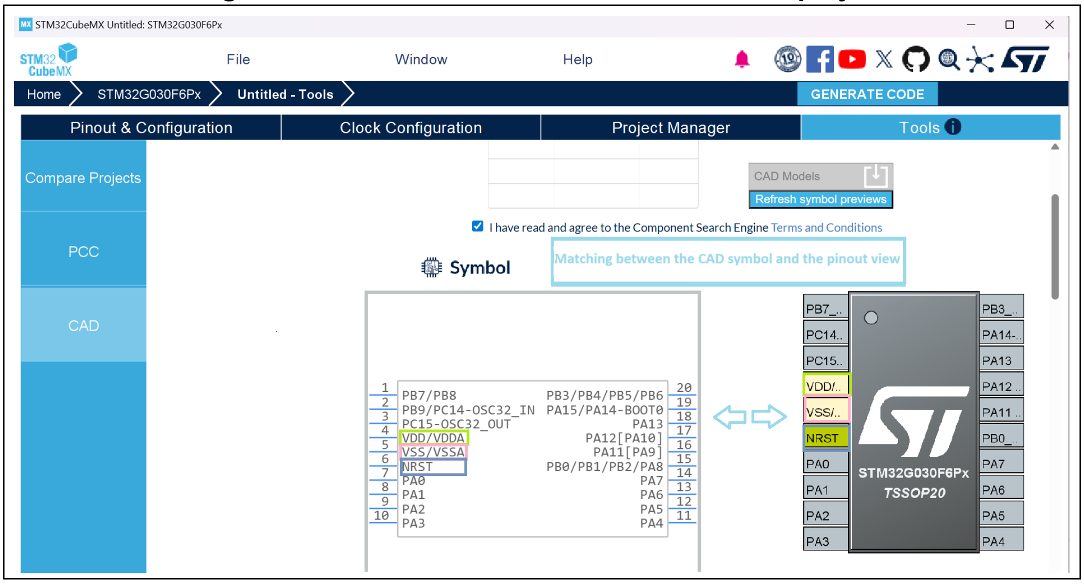
- Open an STM32CubeMX project (the MCU must not be in “Coming Soon” Marketing status)
- Select the CAD tab from the Tools panel to access CAD Resources (see *Figure 228*).

Figure 228. CAD Resources in Tools panel



The Symbol view reflects the STM32CubeMX project pinout configuration and, optionally, the labeling (see *Figure 229*). The downloaded CAD files are aligned with the pinout configuration and optionally, with the labels as well.

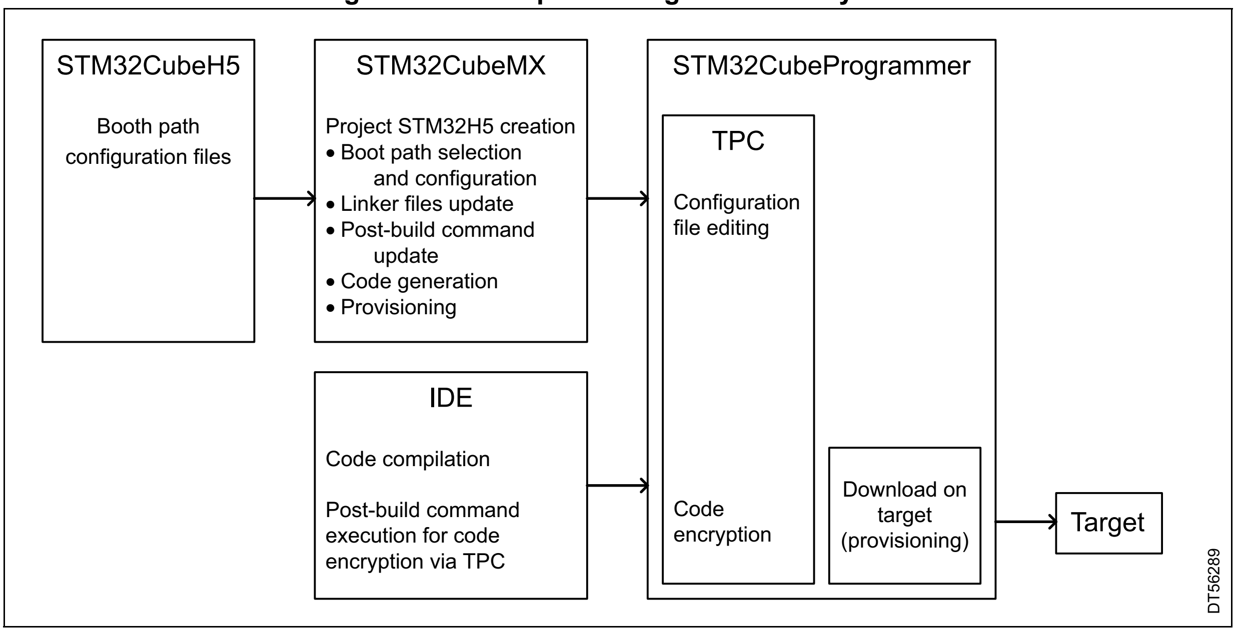
Figure 229. CAD Resources for STM32CubeMX project



4.18 Boot path

STM32CubeMX introduces the possibility to configure the boot path for the STM32H5 series.

Figure 230. Boot path configuration ecosystem



Note: STM32H56x and STM32H503 do not support cryptographic hardware accelerator (a feature needed for the ST-iROT and ST-uROT), therefore the full spectrum of boot paths is not available for these MCUs.

For details about boot path and its usage, read the wiki page available on *www.st.com*, and the guide located under the Utilities folder of the STM32Cube firmware package.

This section details, through examples, how to configure a boot path and generate the associated code. It includes compilation, encryption, and provisioning.

4.18.1 Available boot paths

The following tables give an overview of the different boot paths supported by STM32CubeMX, depending upon the device.

Table 21. Boot paths without TrustZone (TZEN = 0)

MCU	Application	OEM-iRoT → Application	OEM-iRoT→ uRoT → Application	ST-iRoT → Application	ST-iRoT → uRoT → Application
STM32H503x	√	√	-	-	-

Table 22. Boot paths with TrustZone (TZEN = 1)(1)

MCU	S/NS application	OEM-iRoT → S/NS application, and OEM-iRoT → S/NS application (assembled)	ST-iRoT → S application	ST-iRoT → uRoT S/NS application, and ST-iRoT→ S/NS application
STM32H56x	√	√	-	-
STM32H57x	√	√	√	√
STN32H523	√	√	-	-
STM32H533	√	√	√	-

1. S: secure, NS: nonsecure.

Table 23. Boot paths for STM32H7RS devices(1)

MCU	Application	OEM-iRoT → application	ST-iRoT → application	ST-iRoT → OEM-uRoT → application
STM32H7RSx	√	√	√	√

1. S: secure, NS: nonsecure.

The following figures indicate the boot paths that STM32CubeMX can configure, and the entry points after reset.

The related user option bytes are configured automatically (through Trusted Package Creator installed with STM32CubeMX), and programmed during the provisioning stage.

Figure 231. Boot paths for STM32H57x devices

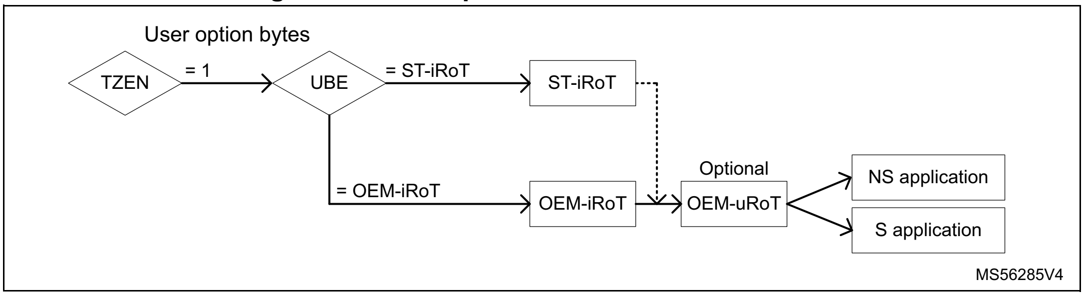


Figure 232. Boot paths for STM32H56x devices

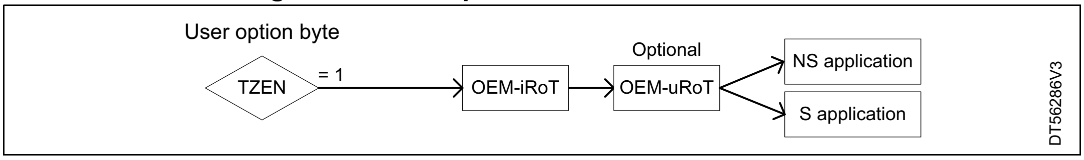


Figure 233. Application boot paths (legacy and ST-iRoT projects)

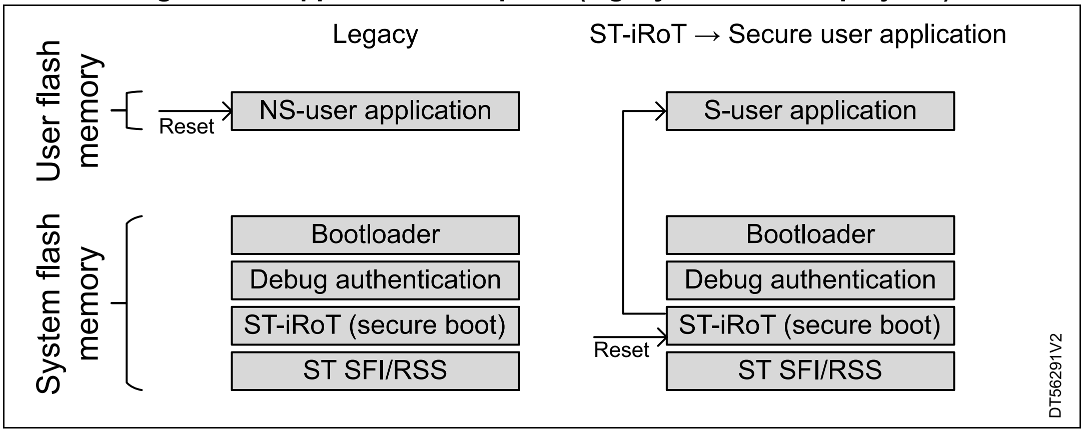


Figure 234. Application boot path (OEM-iRoT)

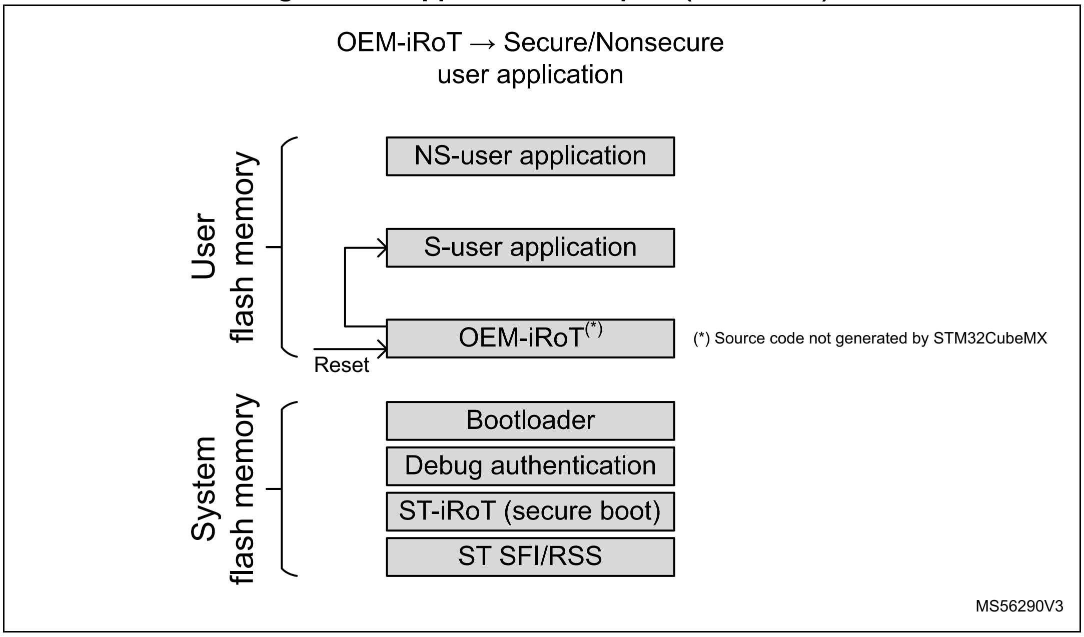


Figure 235. Application boot path (OEM-uRoT assembled)

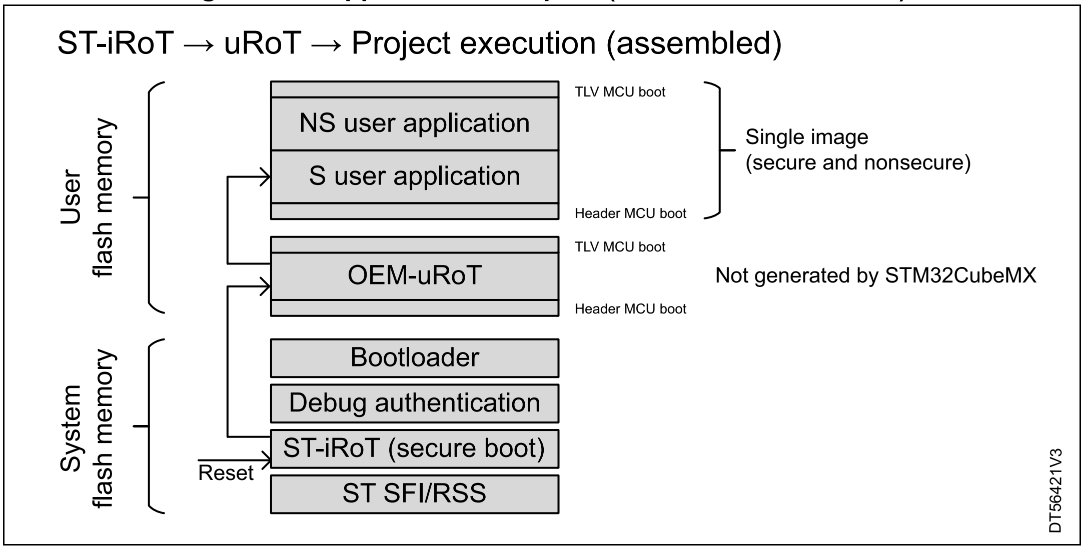


Figure 236. Application boot path: ST-iRoT and uRoT secure/nonsecure project

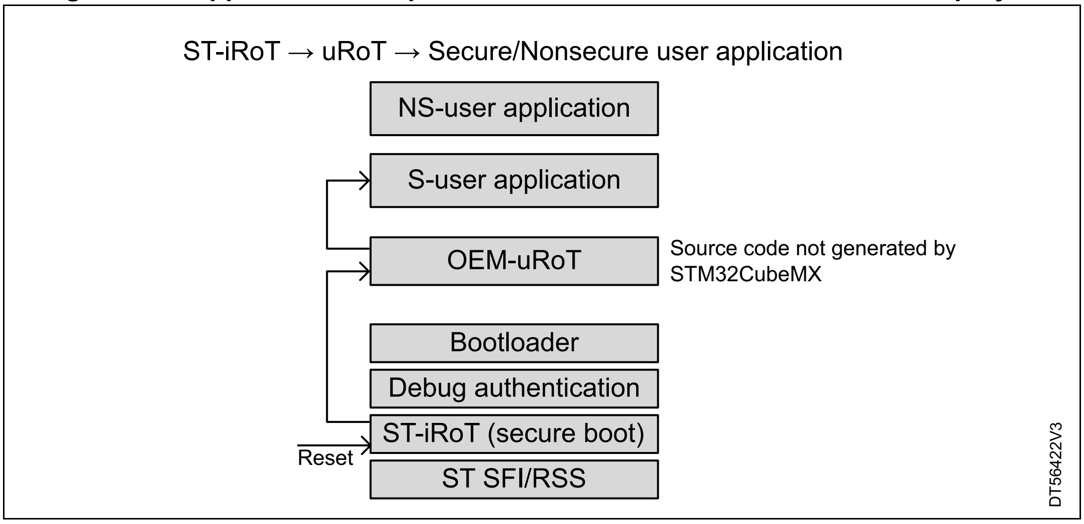


Figure 237. Application boot path: ST-iRoT and secure/nonsecure user application assembled

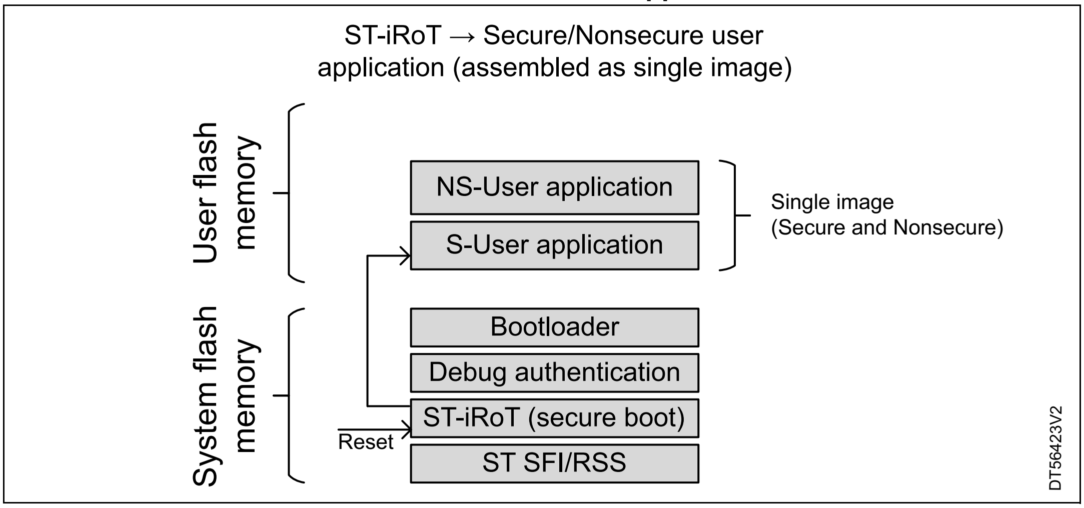


Figure 238. Application boot path: ST-iRoT dual figure

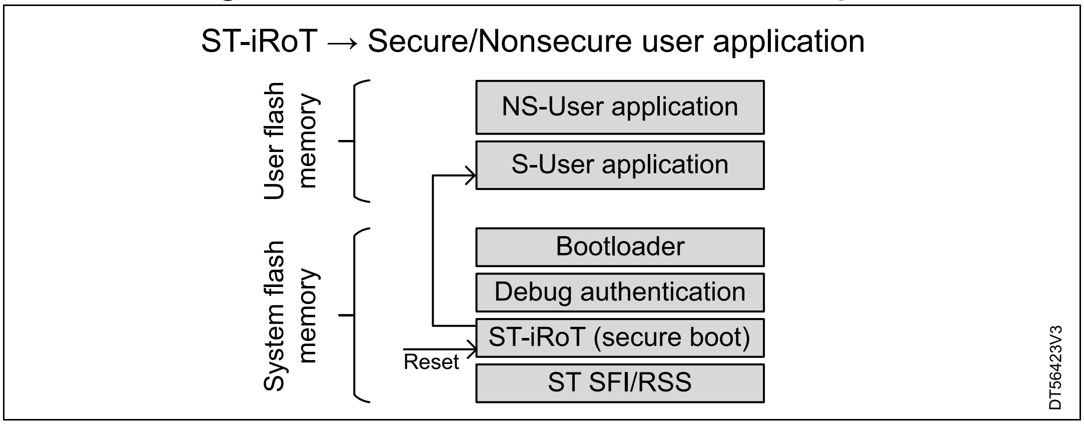
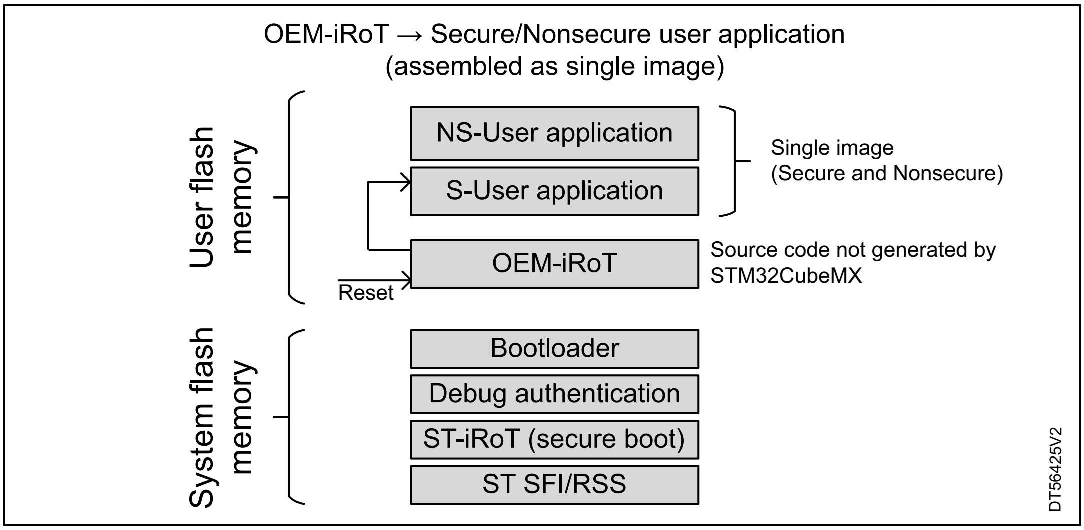


Figure 239. Application boot path: (OEM-iRoT and secure/nonsecure user application assembled)



4.18.2 Creating a boot path project: an example

Prerequisites

- Hardware: Discovery board STM32H573I-DK-REVC
- Tools
  - STM32CubeMX-6.8.0 or later
  - Trusted Package Creator (embedded in STM32CubeMX installation folder)
  - CubeFW must be installed through STM32CubeMX
  - IAR Embedded Workbench® rev 9.20.4 or later

4.18.3 How to configure an OEM-iRoT boot path

The following instructions describe how to generate an OEM immutable Root of Trust (OEM-iRoT) boot path. The procedure to generate other boot paths is similar, but the data required for the configuration can be different.

Step 1: Selecting the MCU

Figure 240. Select the device or board

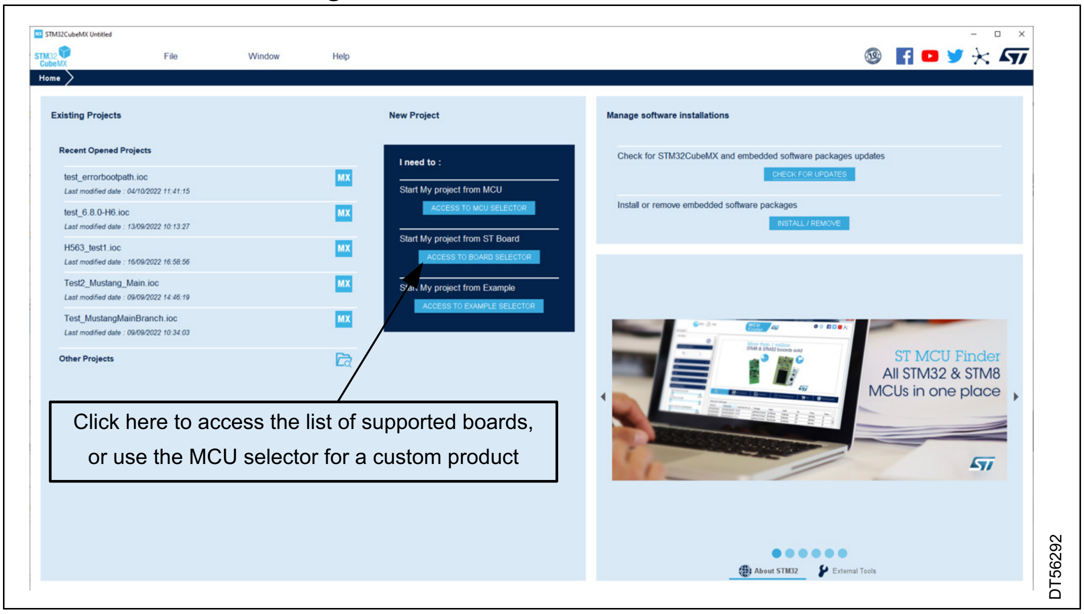


Figure 241. Select the STM32H5 device

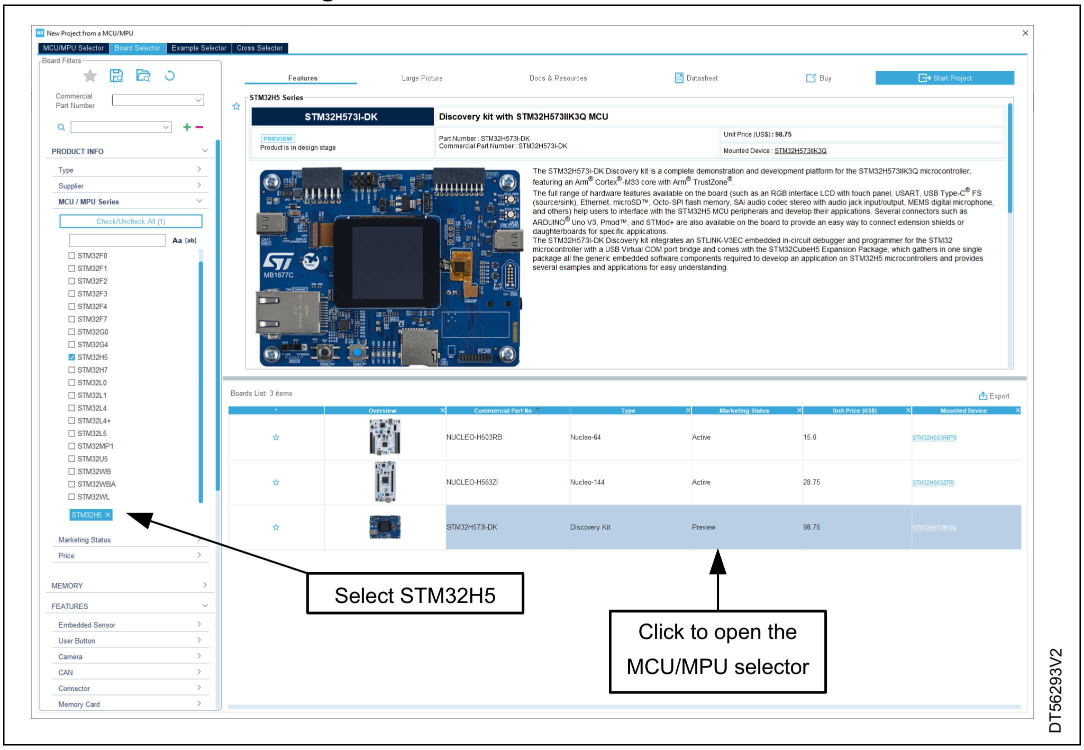
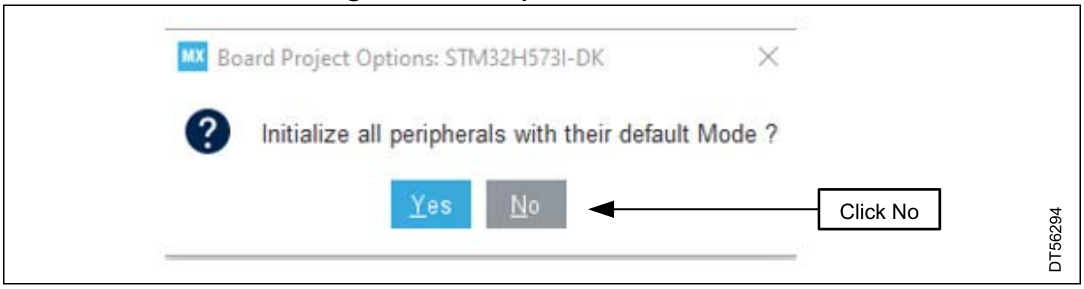


Figure 242. Peripheral initialization

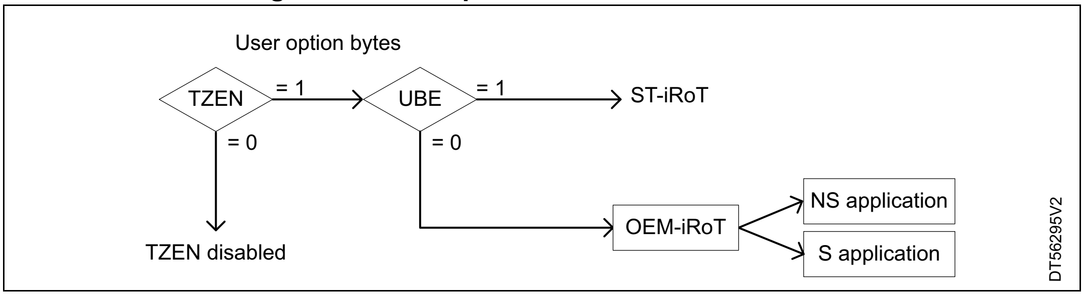


If you click yes, there will be an error during the secure code compilation. By default, all peripherals are set as secure, and the memory allocation for the secure code (defined through the OEM-iRoT_boot application) is too small.

Step 2: Project creation with OEM-iRoT boot path

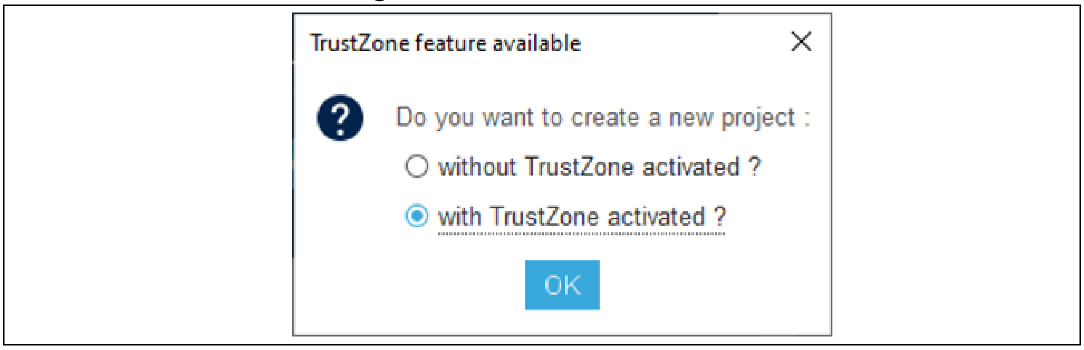
For this example, enable TrustZone (TZEN = 1).

Figure 243. Boot paths for STM32H56x devices



Select the option “with TrustZone activated ?” on the popup window, as shown below.

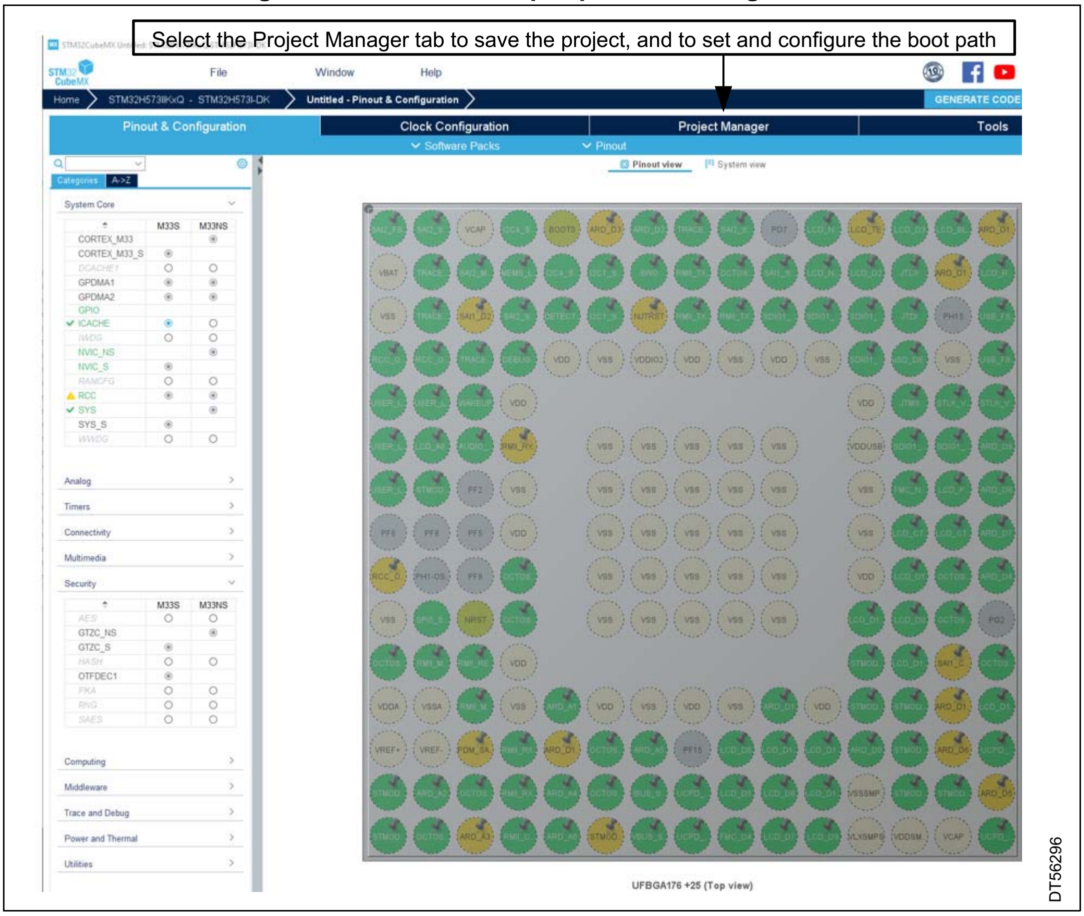
Figure 244. Activate TrustZone



Step 3: Device and peripherals configuration

The device and its peripherals can be configured. In this example, the default configuration is kept.

Figure 245. Device and peripherals configuration



Step 4: Overall configuration

Configure the application (*Figure 246*), then save the project (*Figure 247*).

Figure 246. Configuring the project

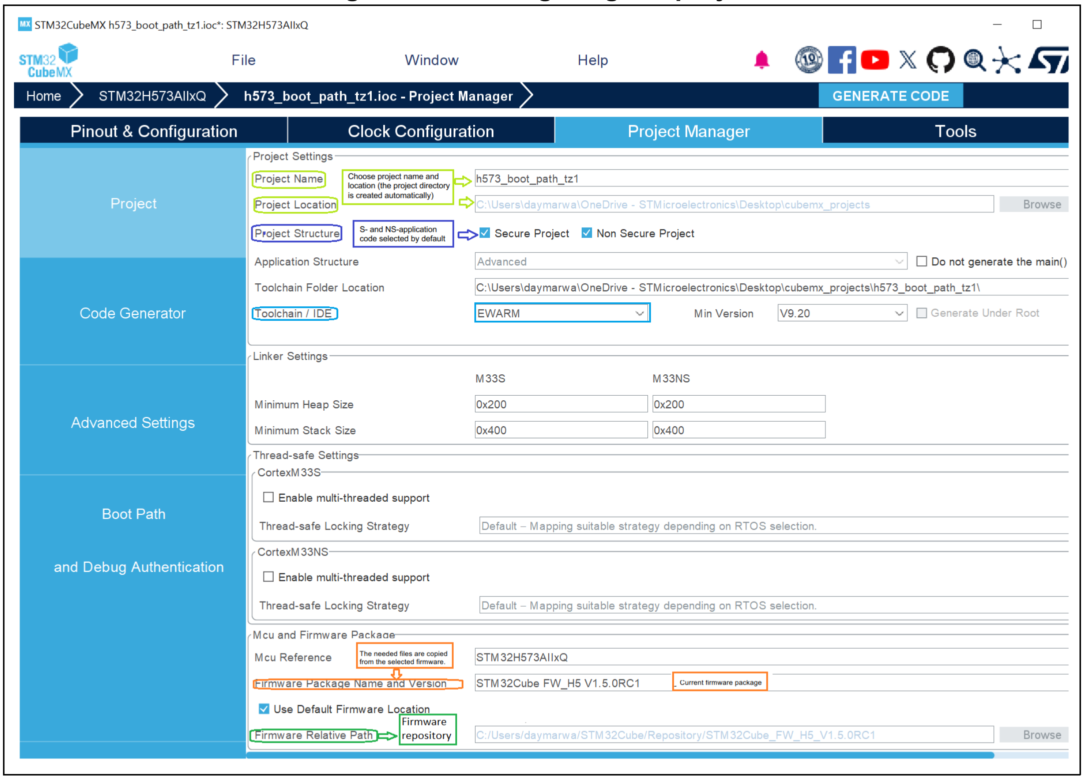
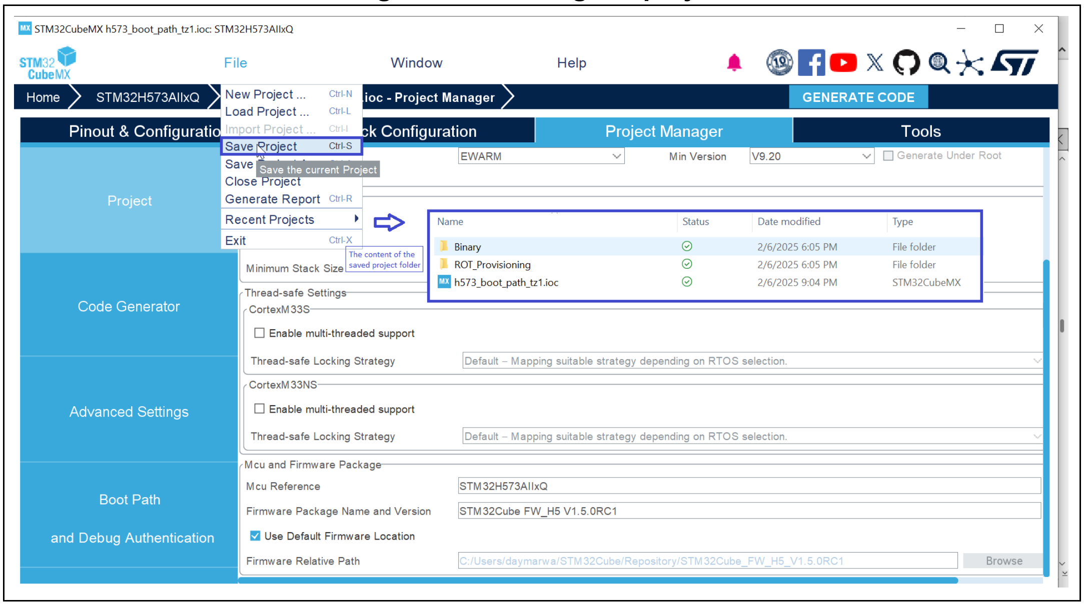


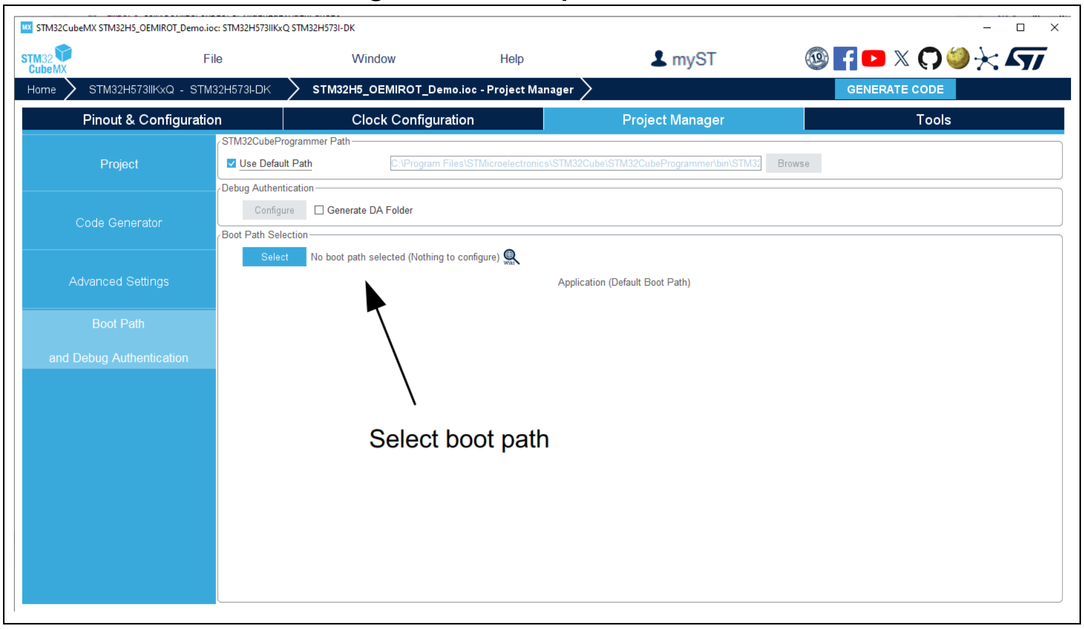
Figure 247. Saving the project



Step 5: Boot path selection

The possible first stages are proposed according to selected device and project structure.

Figure 248. Boot path selection



- Select OEM-iRoT for this example

Figure 249. Select OEM-iRoT

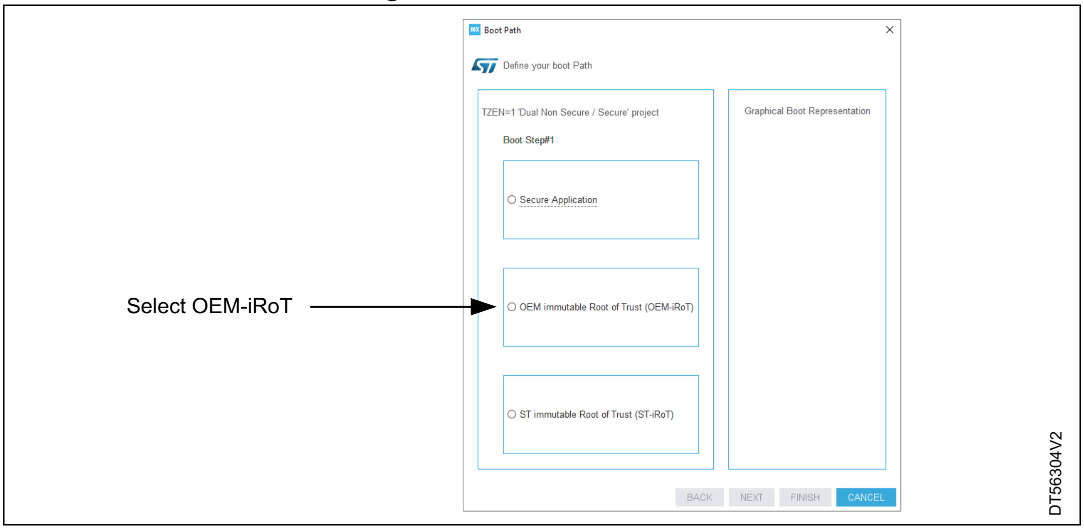
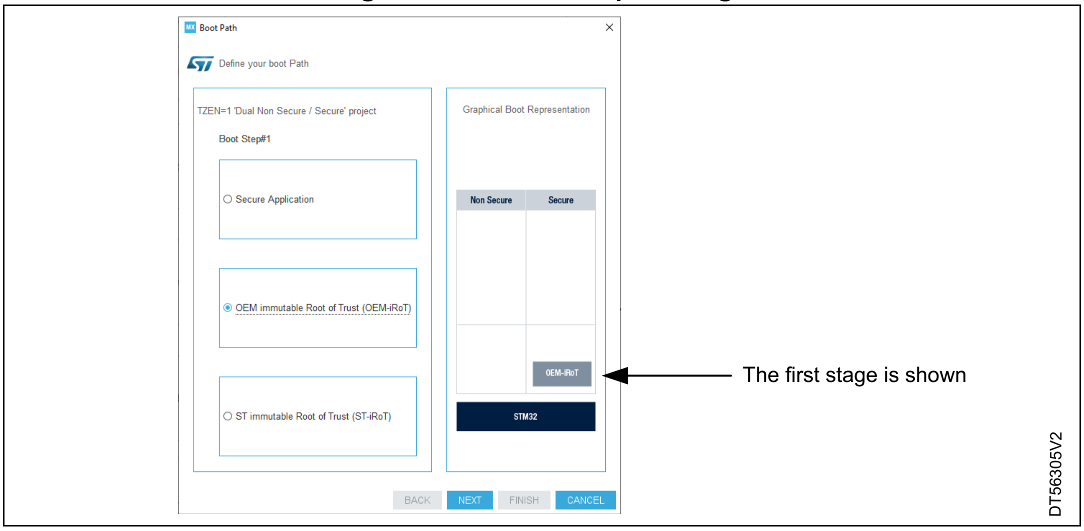
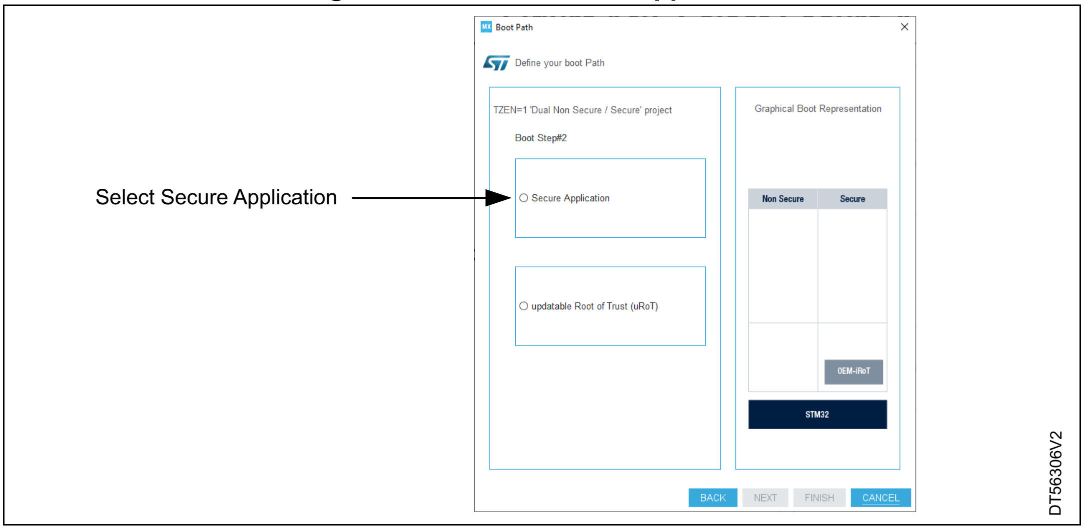


Figure 250. First boot path stage



- All possible boot paths for the second stage are proposed according to the selected device and project structure.
- Select “Secure Application”, it generates secure and nonsecure codes.

Figure 251. Select Secure Application



- Click on FINISH to generate the binary, RoT_Provisioning folder, and sub-folders.

Figure 252. Last boot path stage

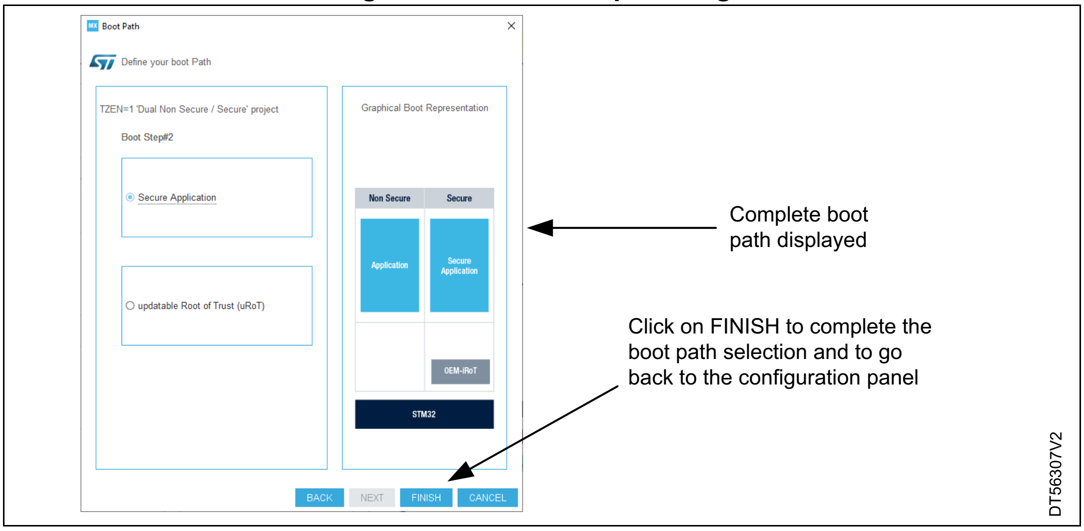
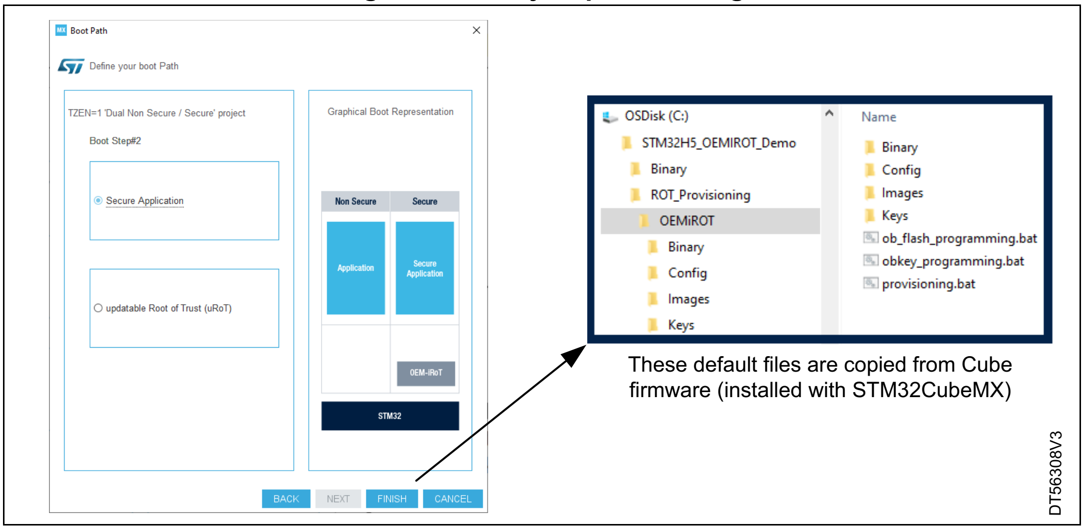


Figure 253. Project provisioning



Note: If a selected boot path is not supported, a warning message is displayed, and the "FINISH" button is grayed out.

Note: For STM32H56x and STM32H523x devices it is not possible to configure the OEM-iRoT boot path if the flash size of the current MCU is not aligned with the FLASH_SIZE entry in the map.properties file. A pop-up window (see Figure 254) is displayed.

Figure 254. Flash size not aligned

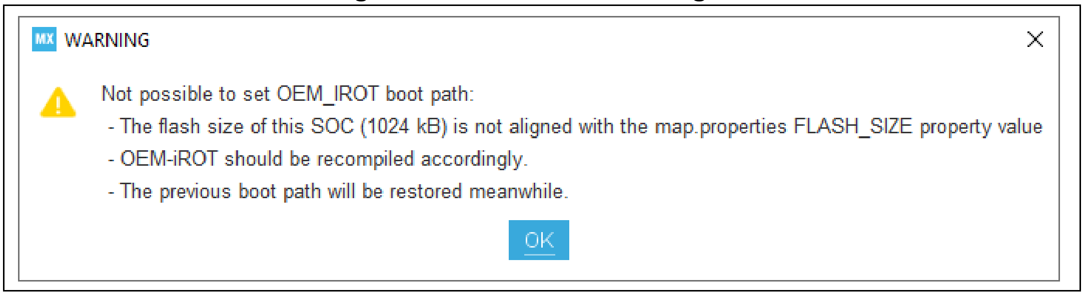
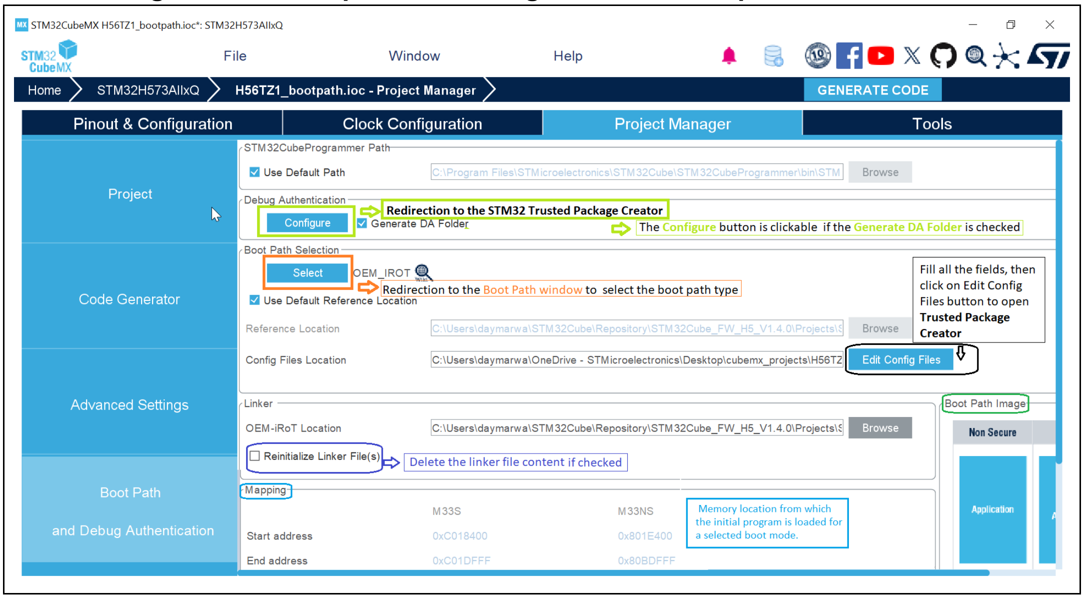


Figure 255. Boot path and debug authentication panel

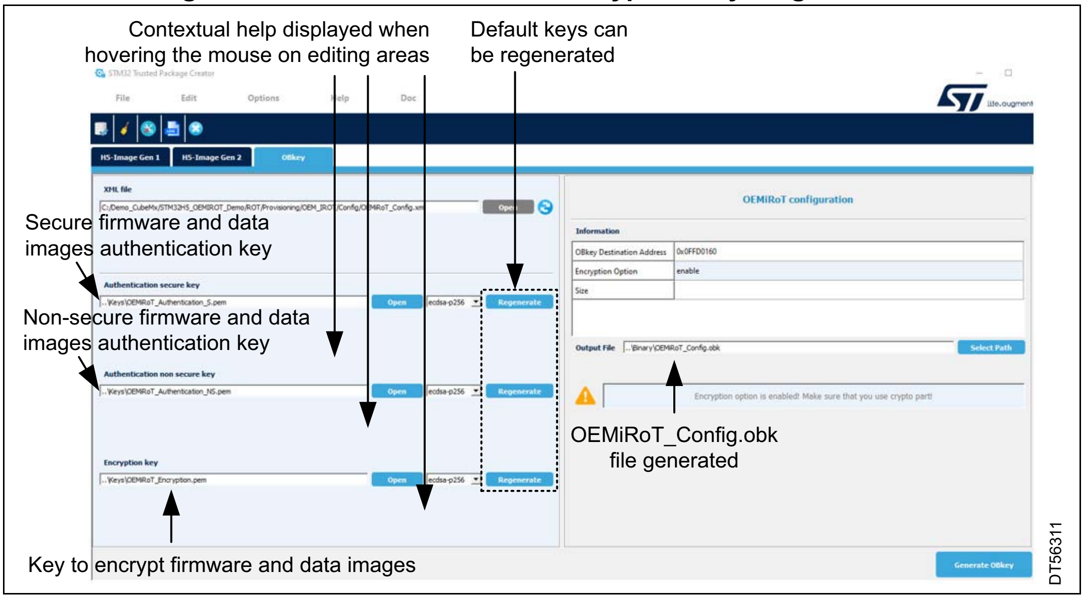


Step 6: Authentication and encryption keys regeneration, option byte file generation

Customization of OEM-iROT configuration file (OEMiROT_Config.obk):

- The default configuration file of CubeFW can be used, but the default keys must be regenerated or replaced
- To customize the configuration file, proceed as follow:
  - a) Launch Trusted Package Creator and select STM32H5 (click edit in Project Manager as indicated in *Figure 253*)
  - b) Open OBkey tab
  - c) The default keys can be regenerated
  - d) The OEMiROT_Config.obk file is generated. The modified parameters are saved in OEMiROT_Config

Figure 256. Authentication and encryption keys regeneration



- The H5-Image Gen1 and Gen2 tabs indicate the location of the image configuration files and the path of the binary input and output files. Keep the default settings.

Figure 257. Secure image configuration

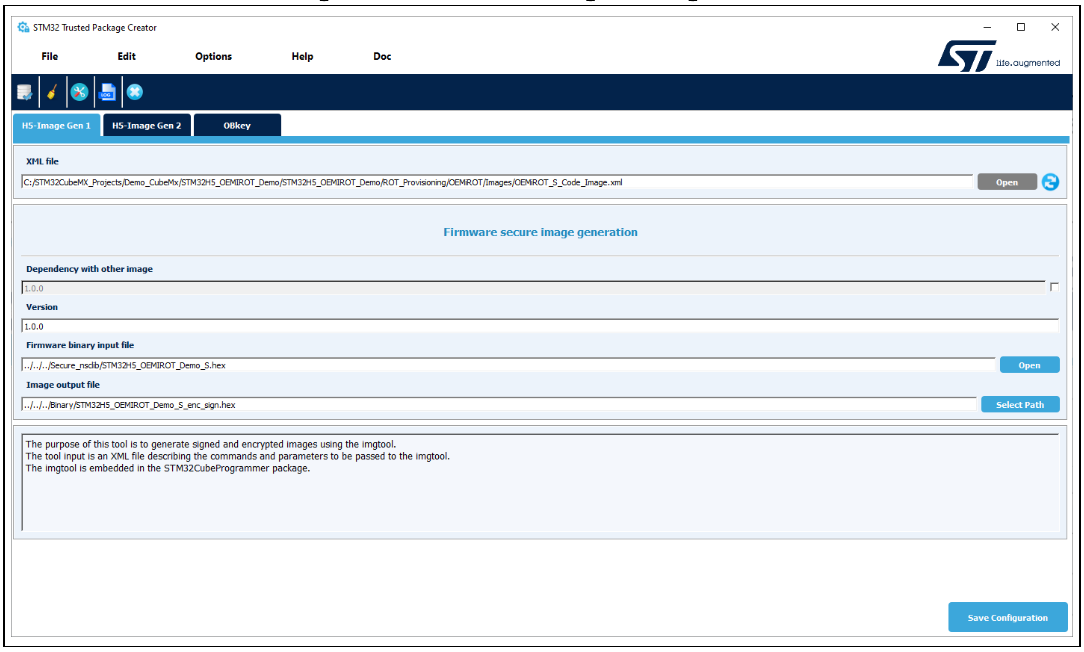
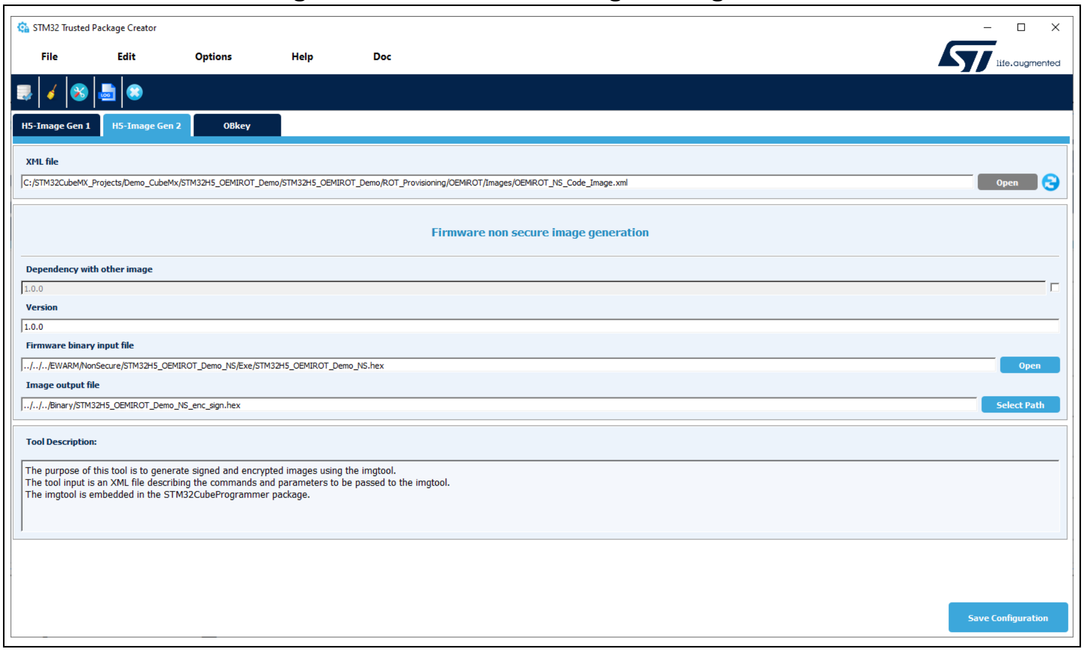


Figure 258. Nonsecure image configuration



Step 7: Code generation

Figure 259. Generate the code

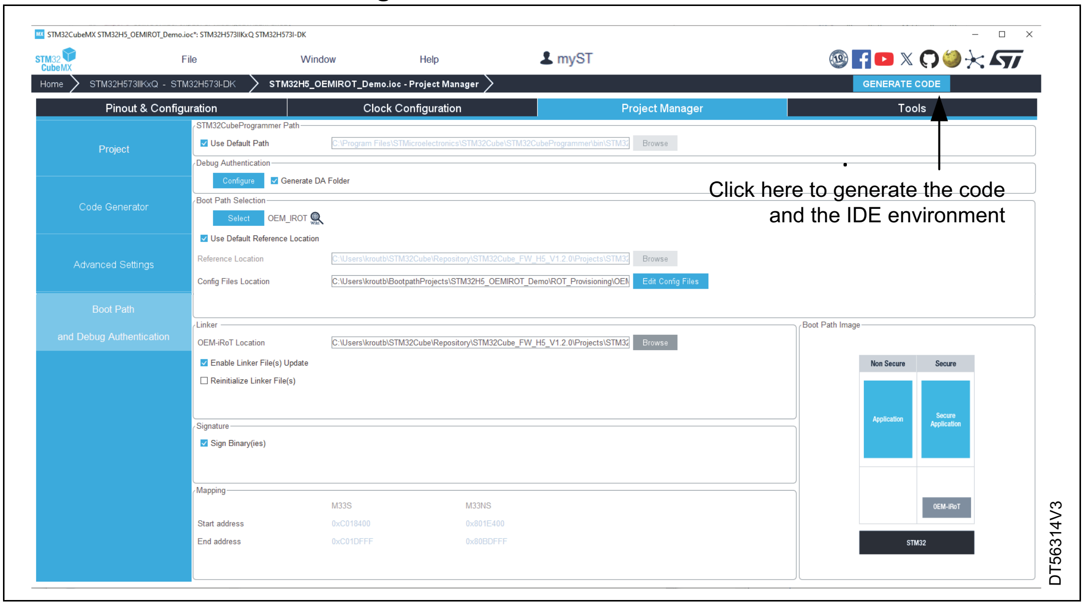
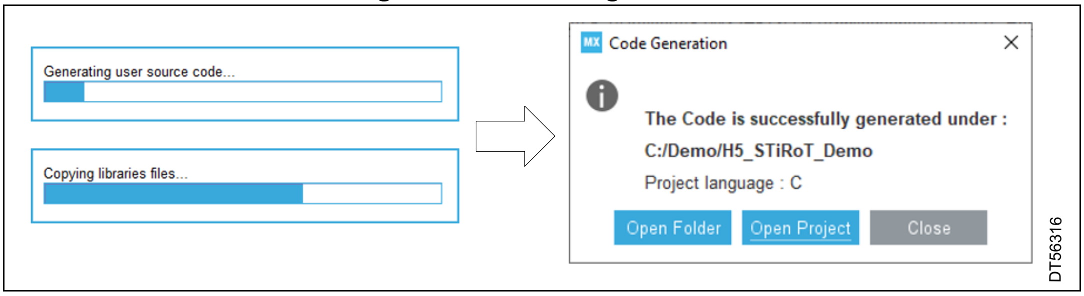
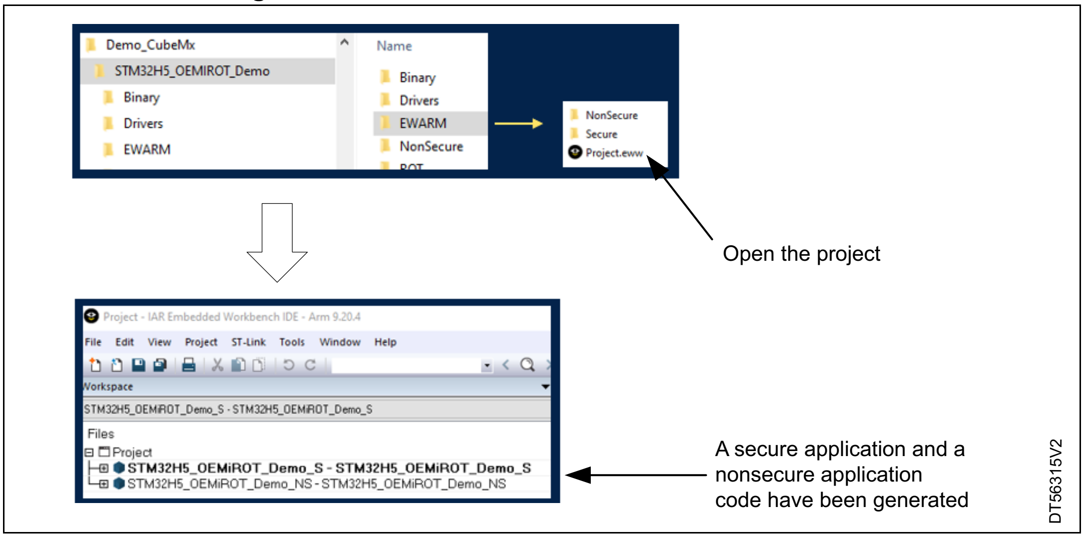


Figure 260. Code is generated



Additional directories, including the IDE environment, are created.

Figure 261. Secure and nonsecure IDE directories

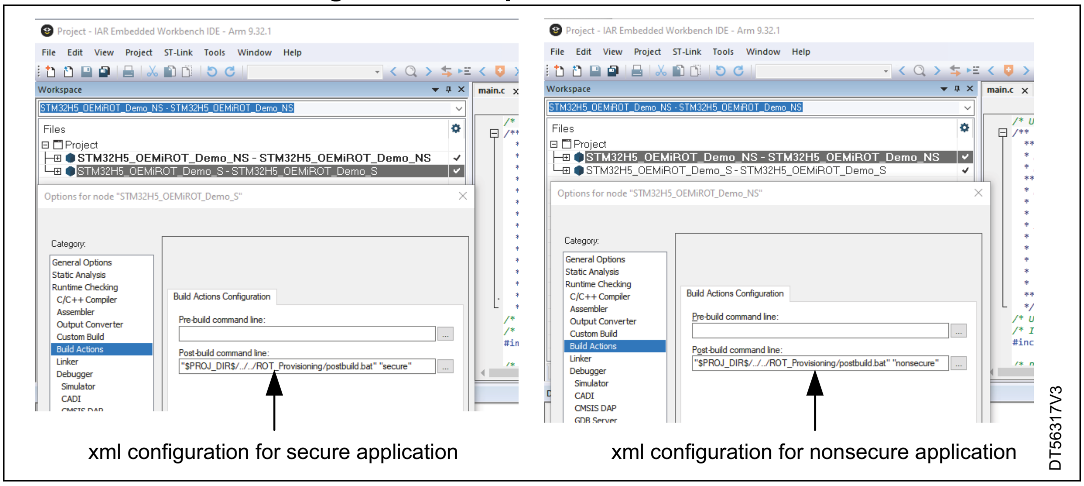


The S and NS applications can be developed using the generated code skeletons.

Step 8: Code compilation

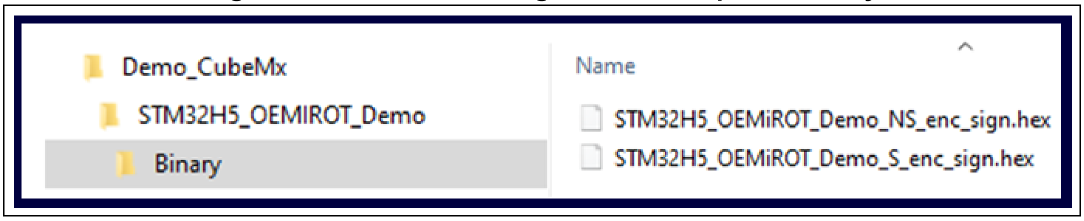
Select Project → Option → Build Actions. The links to the Trusted Package executable, and to the secure and nonsecure application xml files are filled automatically.

Figure 262. IDE post build commands



The secure code must be generated before the nonsecure one. Compile each code separately (right click on Project → Rebuild all). The secure and nonsecure signed and encrypted binaries are generated during the post build phase.

Figure 263. Trusted Package Creator output directory



Step 9: Provisioning of the board

The program cannot be flashed using an IDE. Use provisioning scripts found in the user environment, and double click on the provisioning.bat file (*Figure 264*). During provisioning, log files are generated to inform the user about the activity. Follow the on-screen instructions (*Figure 265*).

Figure 264. Board provisioning

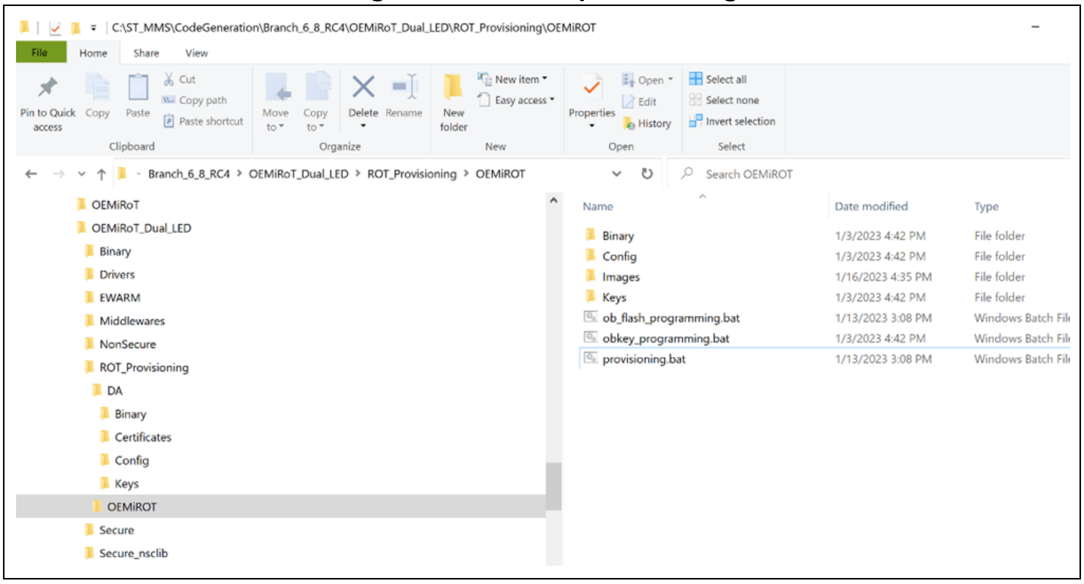
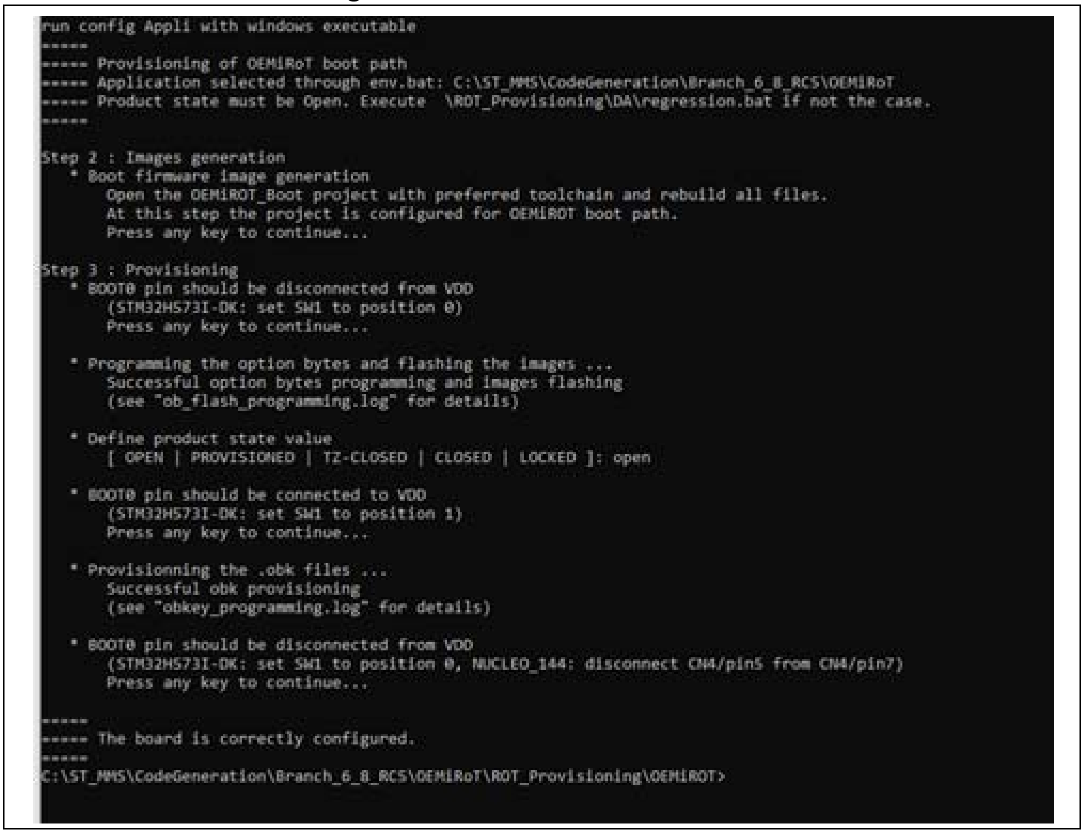


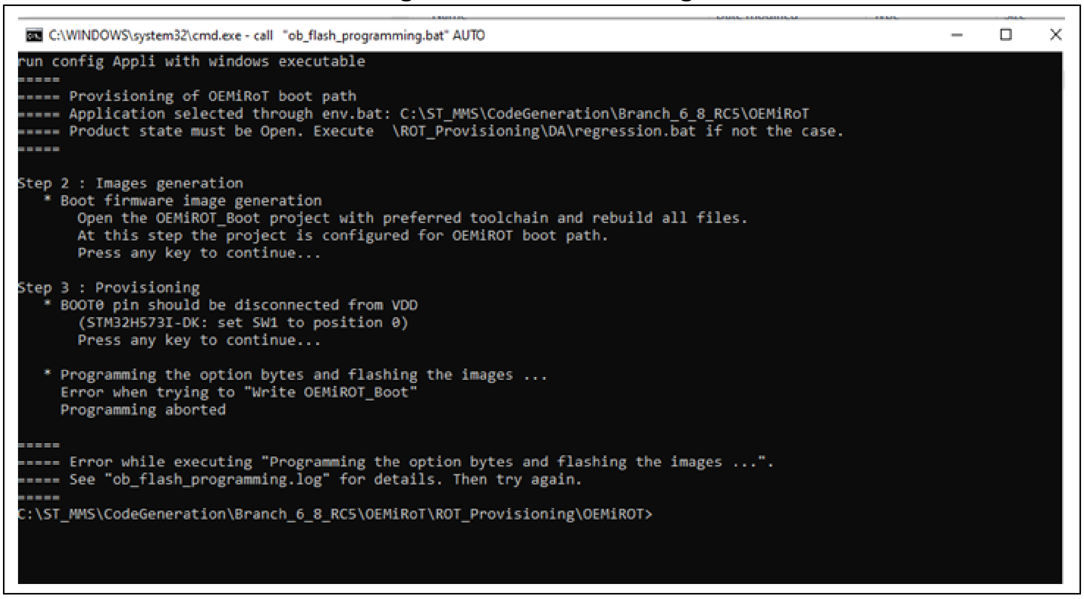
Figure 265. On-screen instructions



In the user environment, STM32CubeMX has generated an env.bat file, containing the information required for provisioning. Do not change this file.

A pop-up (see *Figure 266*) appears if you forget to compile the project OEMiRoT_Boot in the CubeFW.

Figure 266. Error message



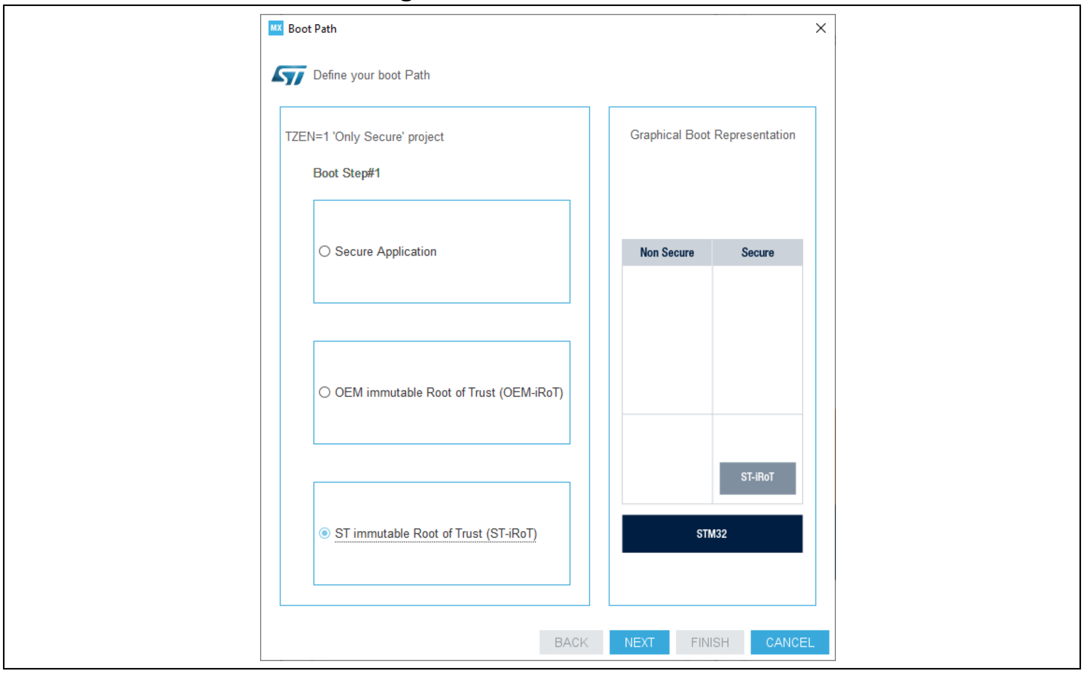
4.18.4 How to configure an ST-iRoT boot path

The configuration for an ST immutable Root of Trust (ST-iRoT) boot path. The requirements are the same of the previous example.

Step 1: Generating the code

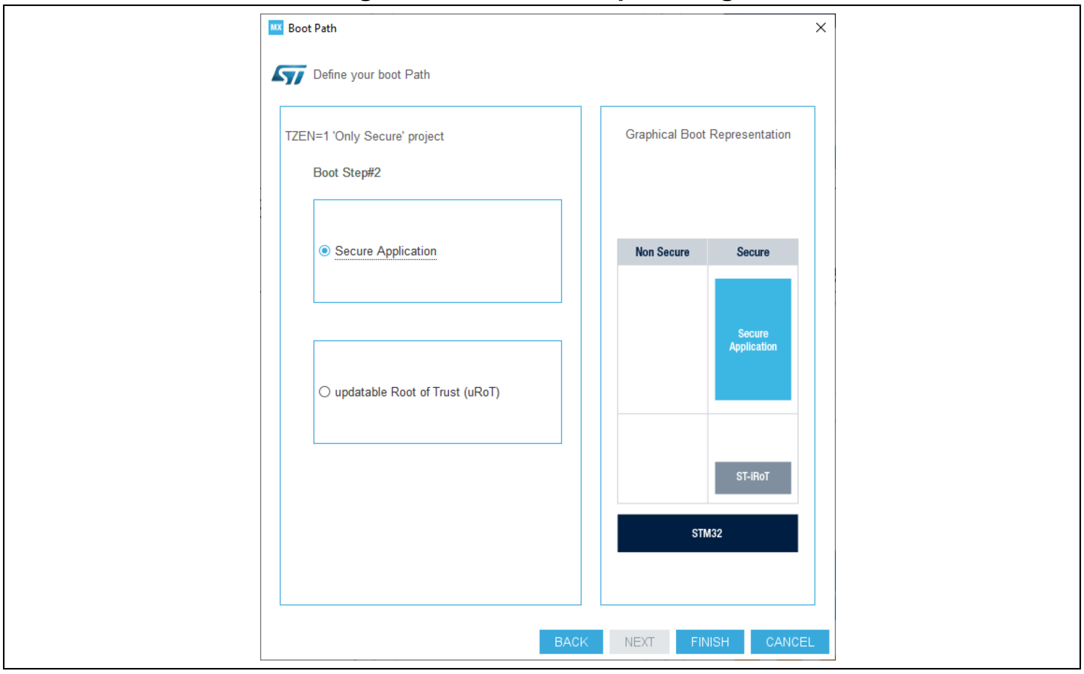
- Select an STM32H57x MCU
- Create a project with TrustZone activated (TZEN = 1)
- In Project Manager, choose “Secure Project”
- Save the project
- Go to “Boot Path and Debug Authentication” tab, and press the Select button
- Choose ST immutable Root of Trust (ST-iRoT)

Figure 267. Select ST-iRoT



- Select Secure Application

Figure 268. Final boot path stage



- Click “FINISH”, the boot path configuration panel is displayed (see *Figure 269*), use it to configure the application, then press the GENERATE CODE button to generate the code for the selected toolchain

Figure 269. Boot path and Debug Authentication tab

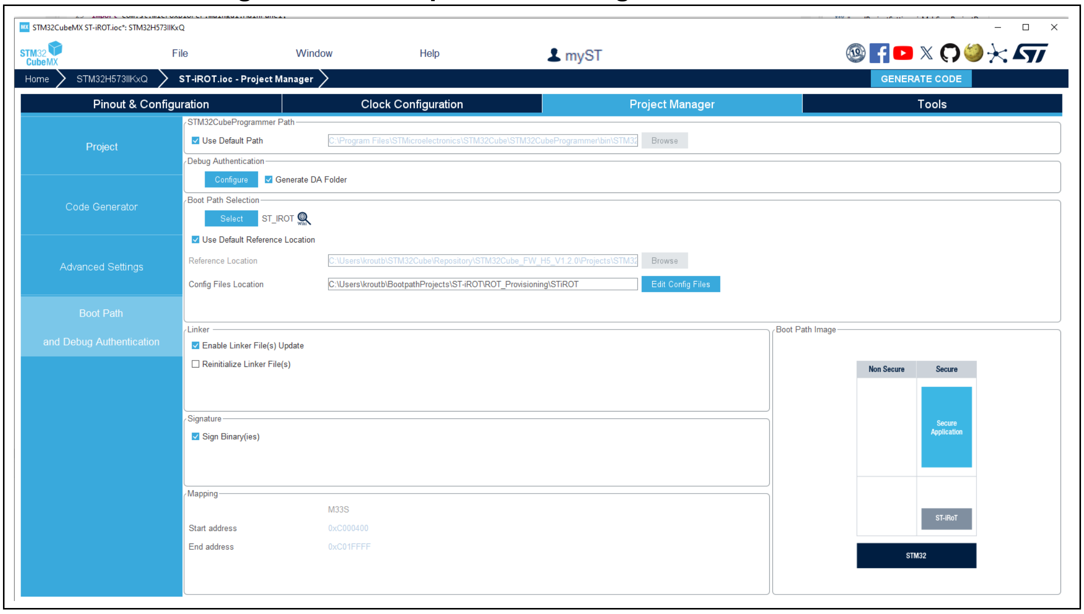
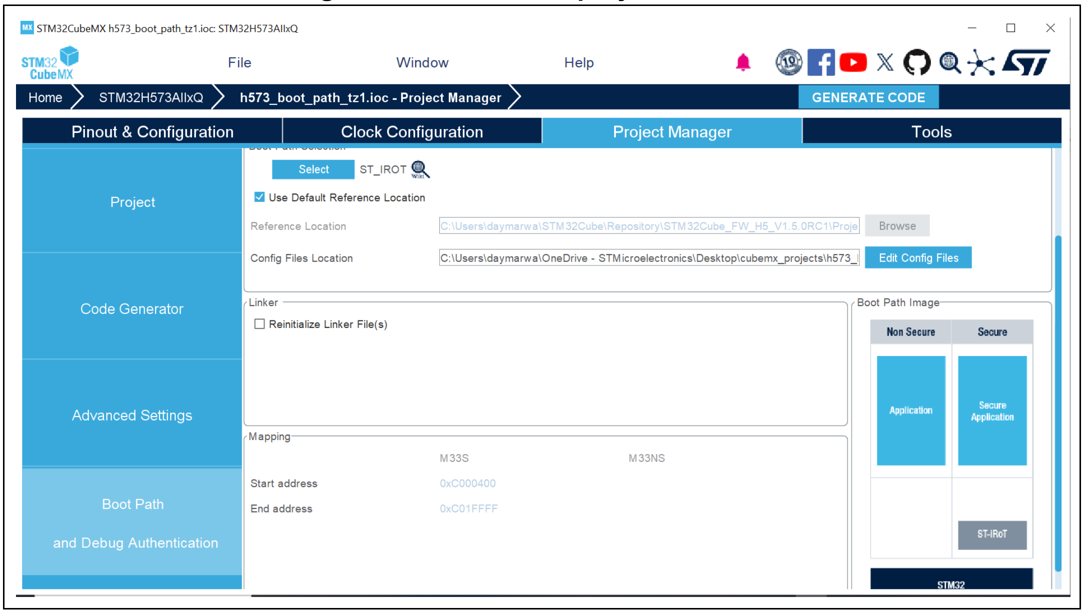
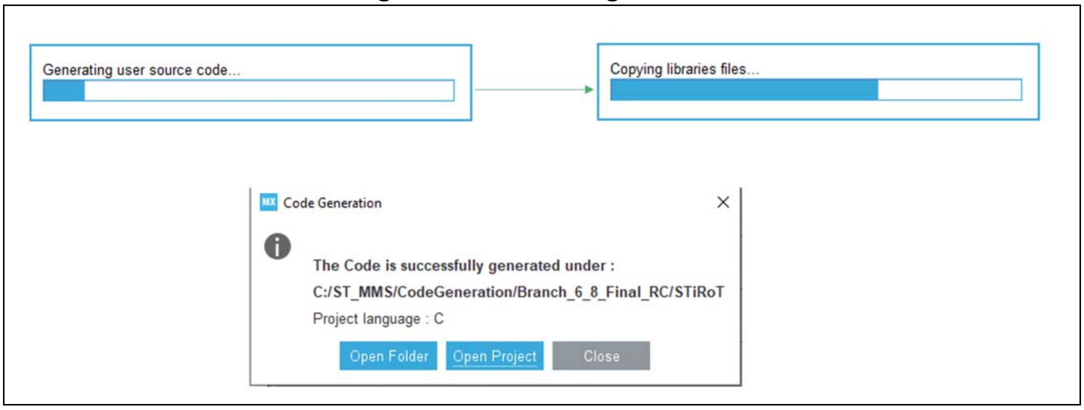


Figure 270. Select the project structure



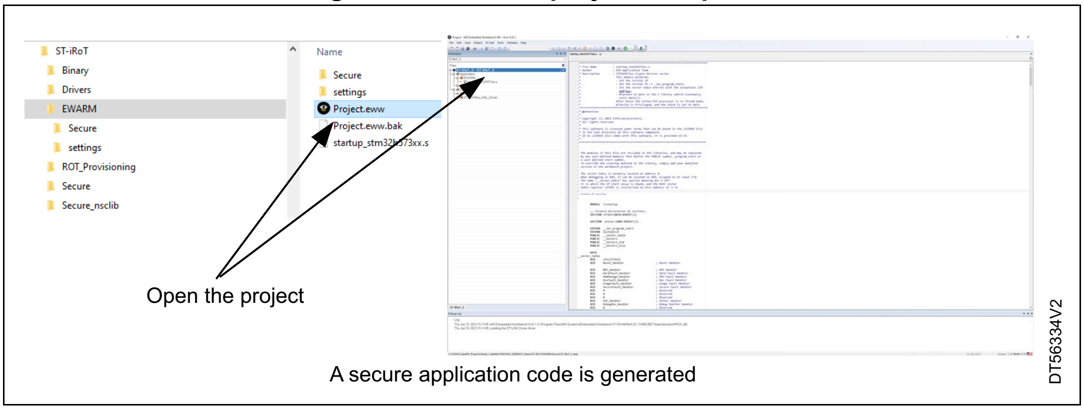
For this boot path, only the secure project is generated.

Figure 271. Code is generated



Additional directories, including the IDE environment are created.

Figure 272. Secure project completed



Secure applications can be developed using the generated code skeletons.

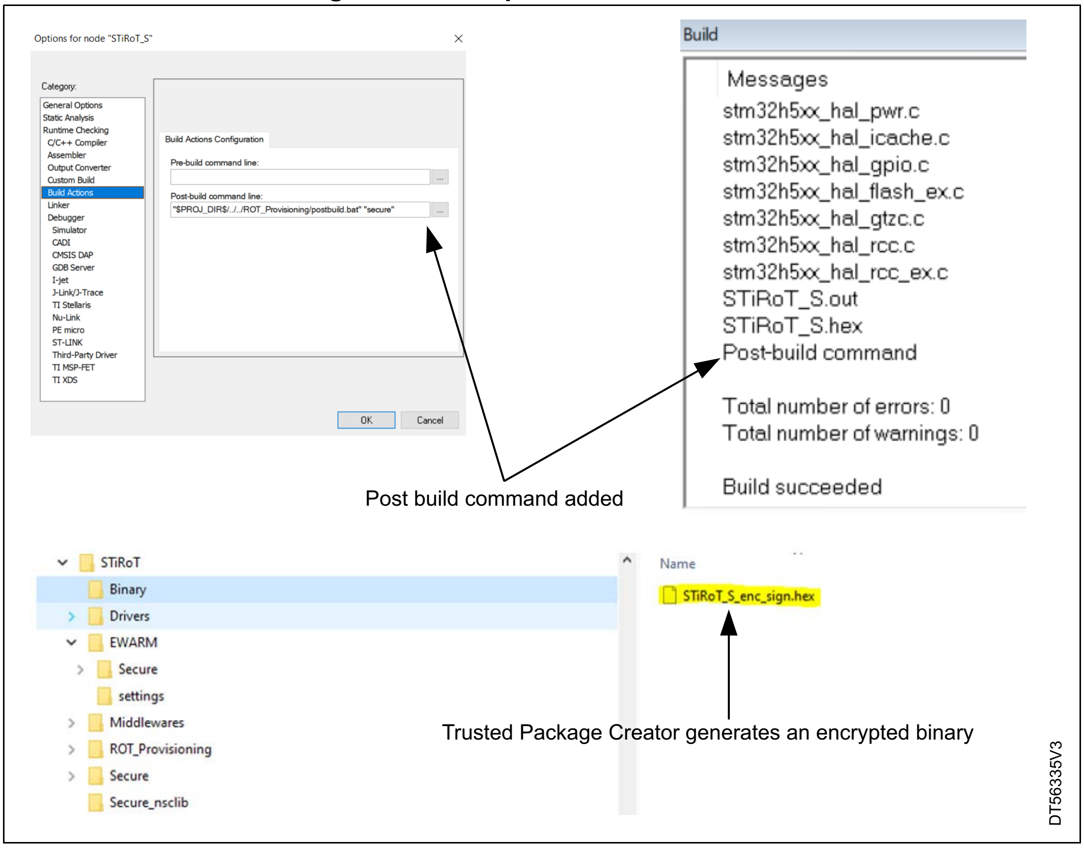
The Post build command creates a secure compiled encrypted code for the provisioning.

Step 2: Code compilation

The generated binaries are automatically encrypted

- Open the project in the selected toolchain, for example IAR
  - Select: Project → Option → Build Actions
  - The links to the Trusted Package executable and to the secure application xml are filled automatically
  - Compile secure (right click on Project → Rebuild all)

Figure 273. IDE post build commands

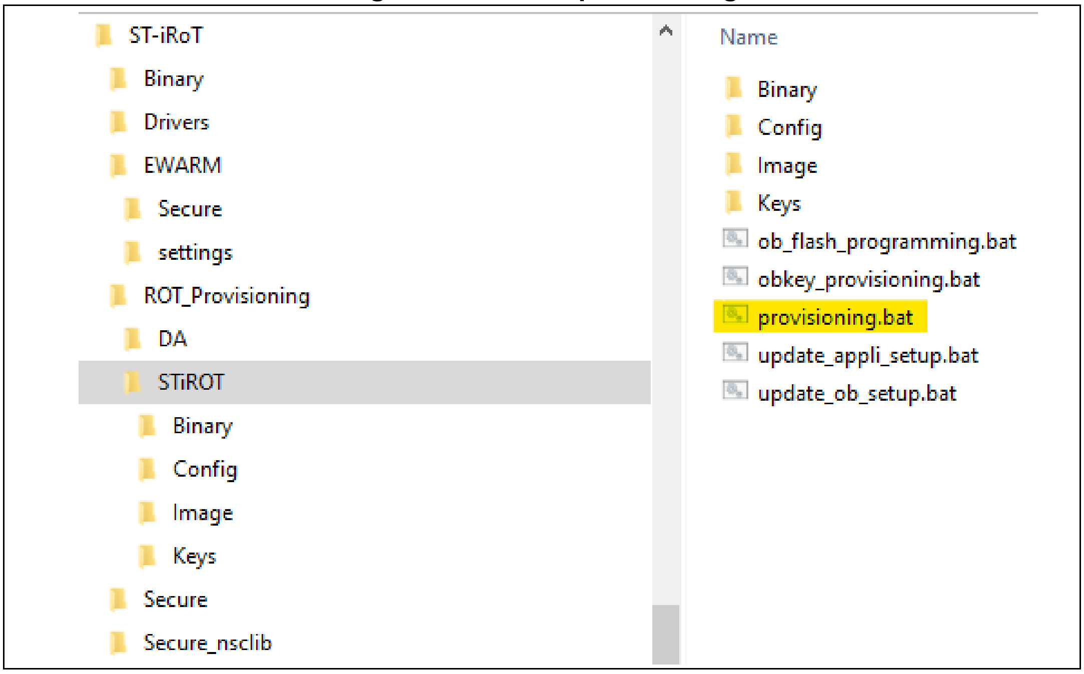


ST-iRoT board provisioning

The program cannot be flashed using an IDE, use the provisioning scripts found in the user environment.

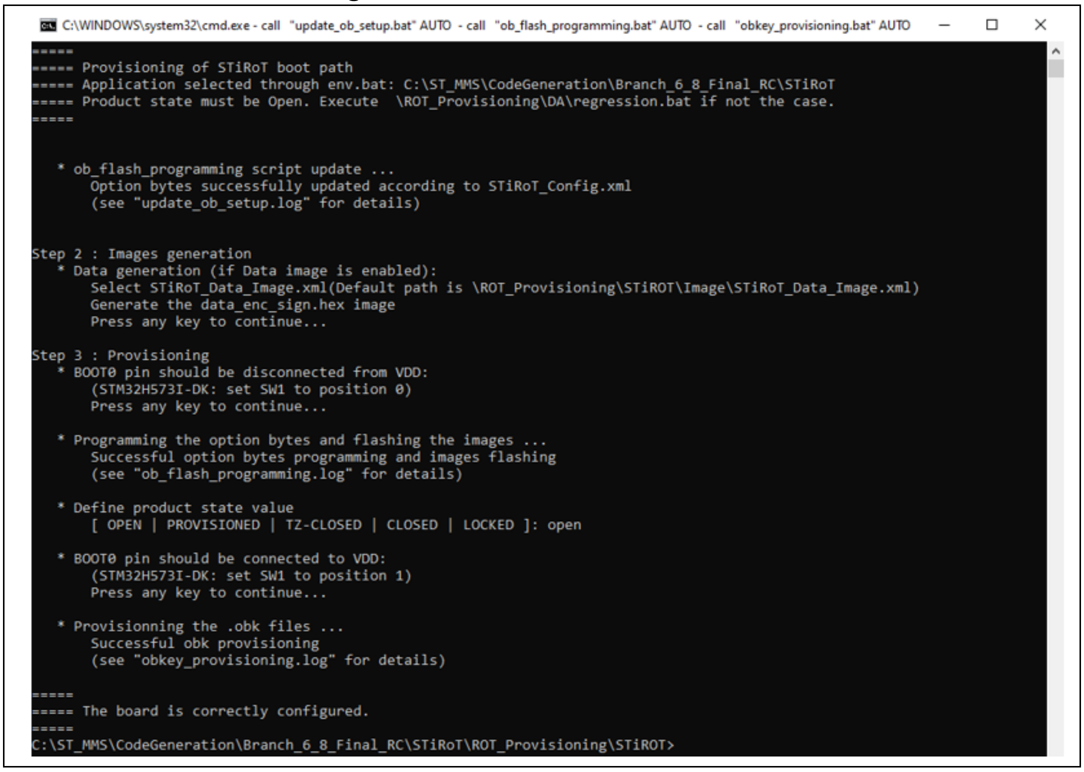
- Double click on the provisioning.bat file (*Figure 274*)

Figure 274. Board provisioning



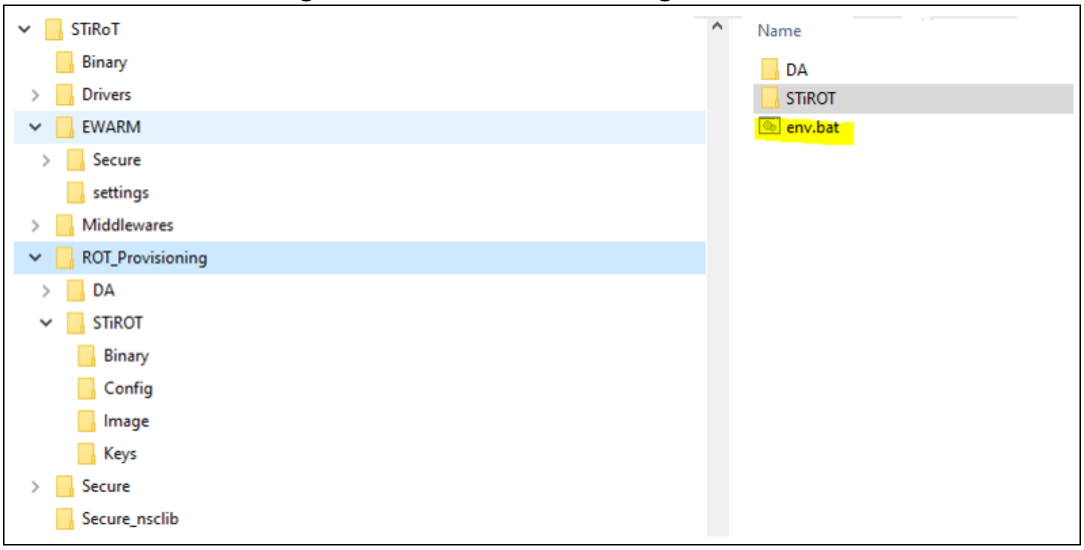
- During provisioning, log files are generated to inform the user about the activity
- Follow the on-screen instructions (*Figure 275*)

Figure 275. On-screen instructions



In the user environment STM32CubeMX has generated an env.bat file containing the required data for provisioning, do not change it.

Figure 276. Environment configuration file



4.18.5 How to configure an assembled boot path

The configuration described below is an example of an assembled boot path.

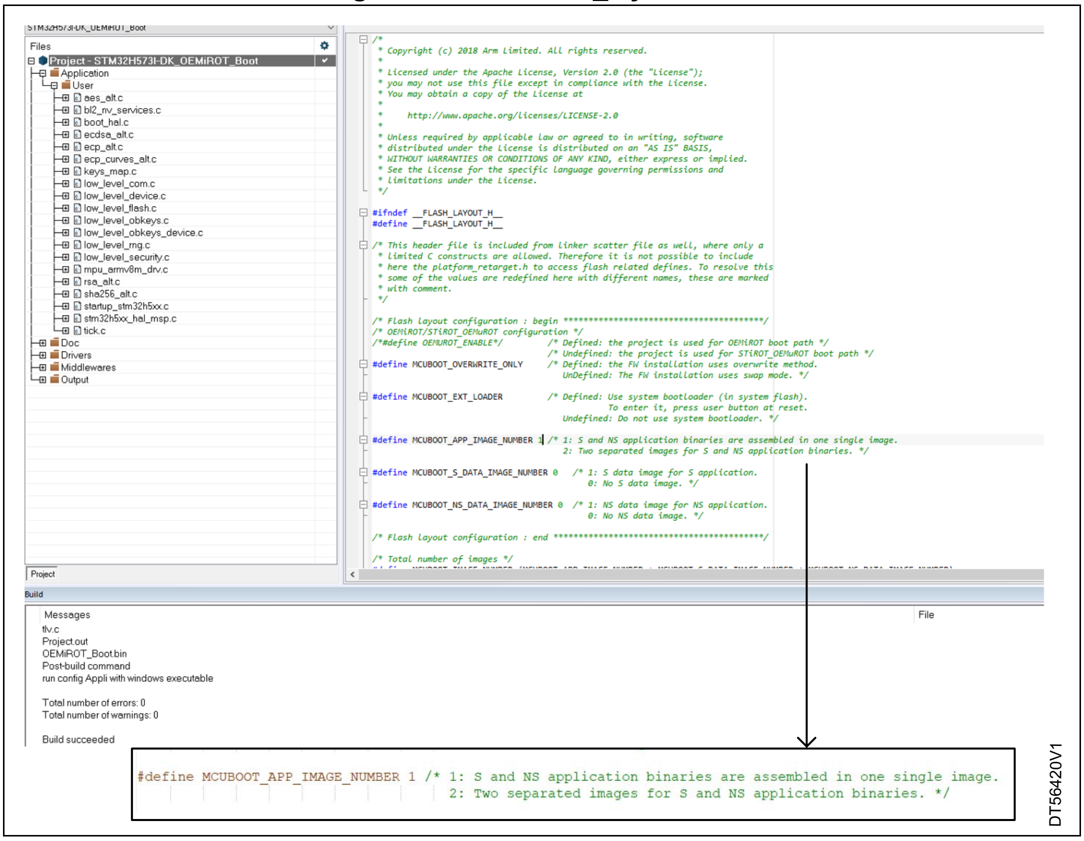
Prerequisites:

- Hardware: Discovery board STM32H573I-DK-REVC or later
- Required tools
  - Secure manager package, to be downloaded and installed from *www.st.com*
  - STM32CubeMX-6.9.0 or later
  - STM32 Trusted Package Creator (embedded in STM32CubeMX installation folder)
  - IAR Embedded Workbench rev 9.20.4 or later, and the patch in the STM32CubeH5 firmware (Version 1.1.0 or later), named EWARM/EWARMv8_STM32H5xx_Vx.x.x.zip.

Step 1: Configure flash_layout.h file

- Go to STM32Cube\Repository\STM32Cube_FW_H5_VX.X.X\Projects\STM32H573I-DK\Applications\ROT\OEMiROT_Boot\Inc
- Open flash_layout.h
- Set the value of this define to 1 to assemble the secure and nonsecure binaries into one: #define MCUBOOT_APP_IMAGE_NUMBER 1.

Figure 277. The flash_layout.h file



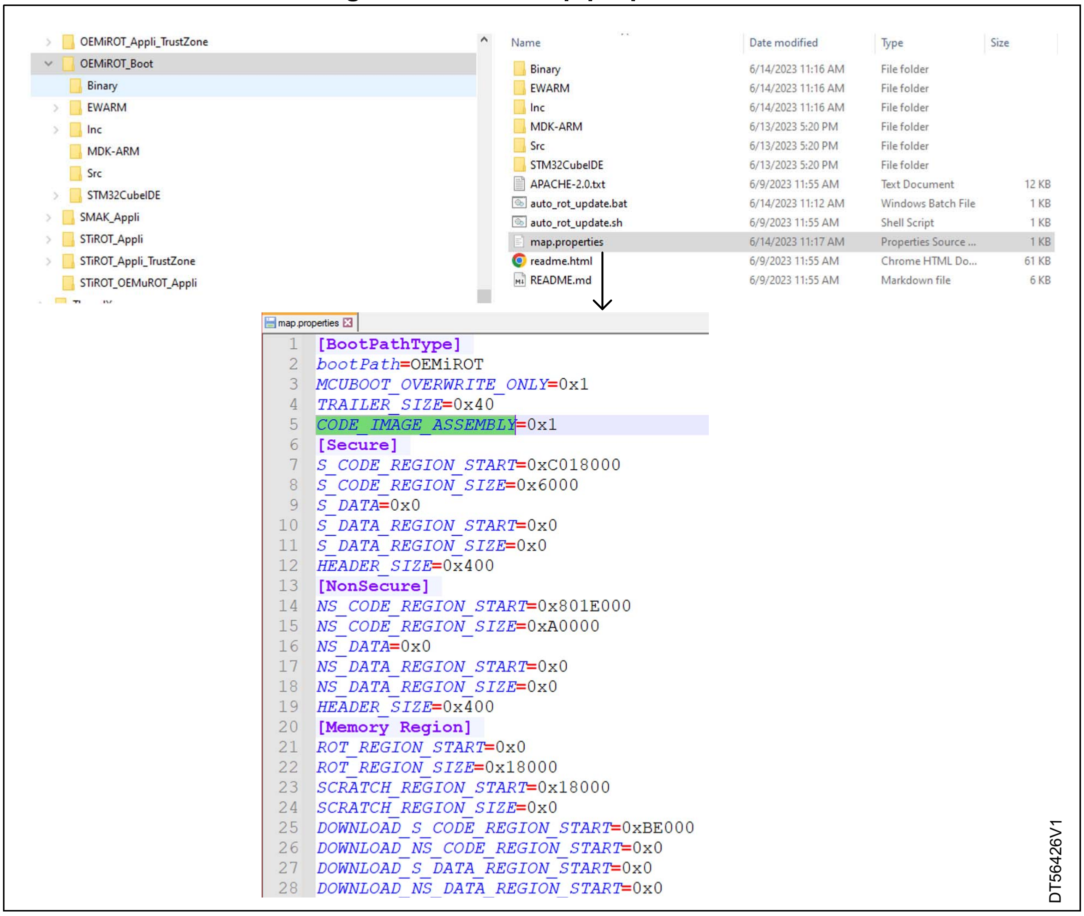
Step 2: Compile OEMiROT_Boot project

- Open OEMiROT_Boot with your preferred tool chain, and recompile the project.
  - The map.properties file is automatically updated (CODE_IMAGE_ASSEMBLY=0x01)
  - The image file (OEMiRoT_NS_Code_Image.xml) is automatically updated (firmware area size)

Step 3: OEMiROT (assembled) code generation

- Open STM32CubeMX application and create a new project with the H5 series (example: choose “STM32H573ZITxQ”)
- Go to Project Manager window, and select secure and nonsecure application
- Add a name for the project and save it
- Go to Boot Path and Debug Authentication Panel: in Boot path selection, click on Select button
- Select OEM-iRoT in the boot path wizard window, and click Next
- Select Secure application, and click Finish

Figure 278. The map.properties file



- Generate and build the project

Figure 279. Secure generated project

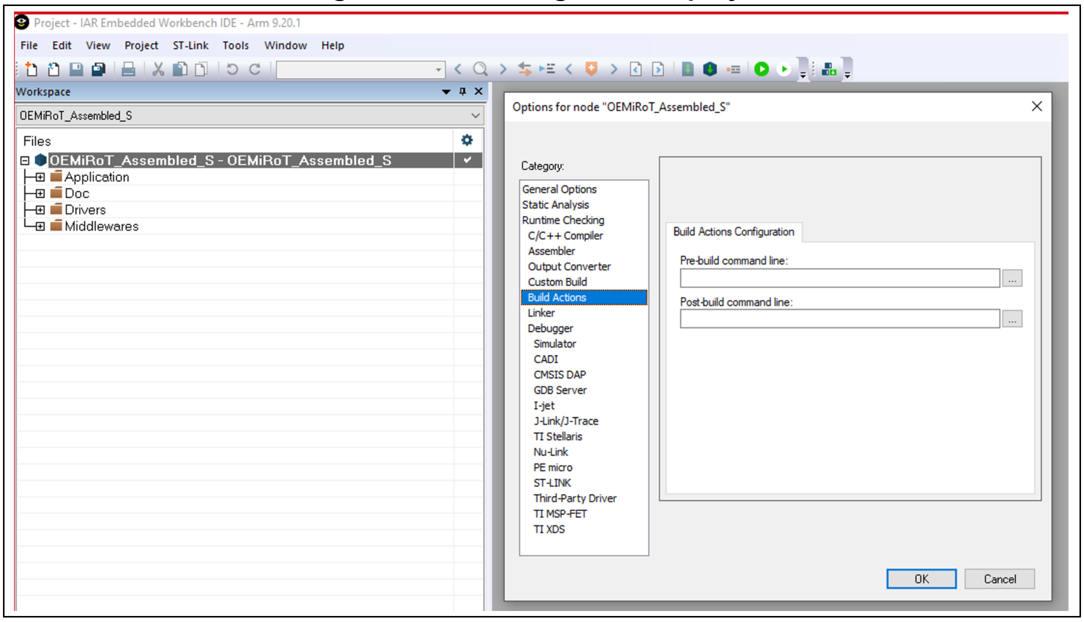


Figure 280. Nonsecure generated project

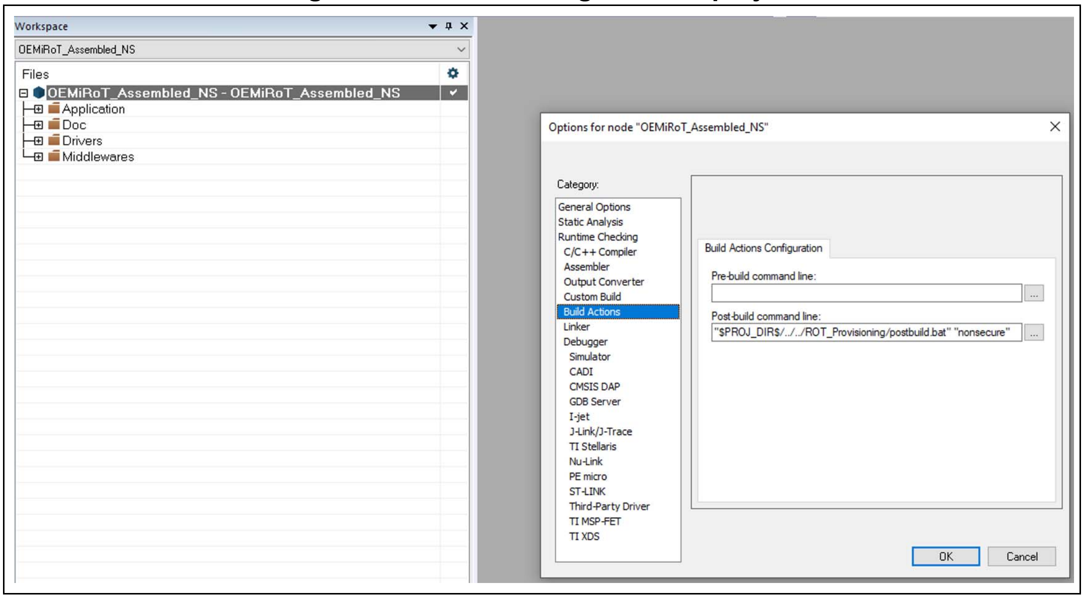
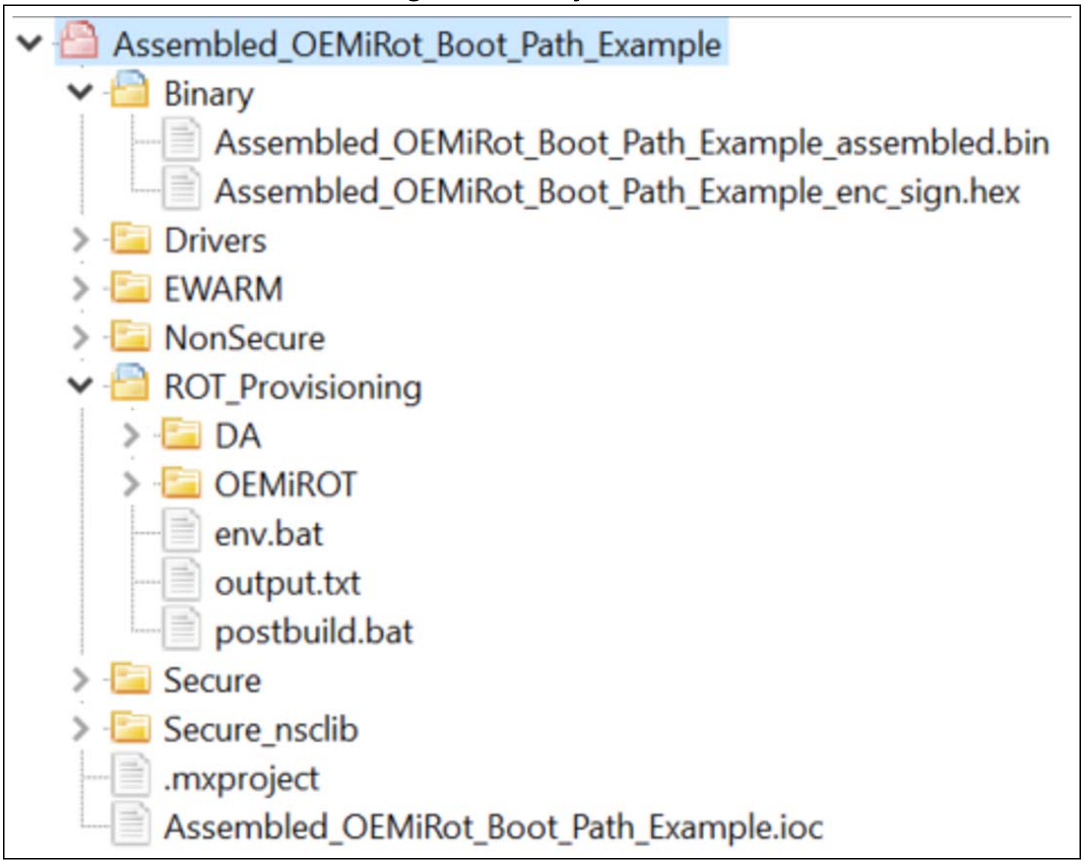


Figure 281. Compilation project

```
OEMiRoT_Assembled_NS.out
OEMiRoT_Assembled_NS.bin
Post-build command
Wed Jun 14 11:36:28 2023 : [INF] ############## Executing oneimage command ##############
Wed Jun 14 11:36:28 2023 : [INF] AppliCfg.exe oneimage -fb OEMiRoT_Assembled_S.bin -sb OEMiRoT_Assembled_NS.bin -o 0x6000 -ob OEMiRoT_Assembled_assembled.bin –vb
Wed Jun 14 11:36:28 2023 : [INF] First binary  : 'C:\ST_MMS\CodeGeneration\Branch_6_9_Beta_7\OEMiRoT_Assembled\ROT_Provisioning\/../Secure_nsclib\OEMiRoT_Assembled_S.bin'
Wed Jun 14 11:36:28 2023 : [INF] Second binary : 'C:\ST_MMS\CodeGeneration\Branch_6_9_Beta_7\OEMiRoT_Assembled\ROT_Provisioning\/../EWARM/NonSecure\OEMiRoT_Assembled_NS\Exe\OEM
iRoT_Assembled_NS.bin'
Wed Jun 14 11:36:28 2023 : [INF] Extra padding will be added at the end of the first binary
Wed Jun 14 11:36:28 2023 : [INF] Image assembly success
Wed Jun 14 11:36:28 2023 : [INF] Final image size '29966'
        STM32 Trusted Package Creator v2.14.0-B03

-pb C:\ST_MMS\CodeGeneration\Branch_6_9_Beta_7\OEMiRoT_Assembled\ROT_Provisioning\OEMiROT\Images\OEMiROT_NS_Code_Image.xml

Image generated successfully


Total number of errors: 0
Total number of warnings: 0

Build succeeded
```

- Open the project folder. A Python script assembles both binaries (Secure, Non Secure), then the TPC signs them:
  - Assembled_OEMiRot_Boot_Path_Example_assembled.bin → File assembled by the Python script
  - Assembled_OEMiRot_Boot_Path_Example_enc_sign.hex → File signed by the TPC

Figure 282. Project folder



- The post build command is added only for the Non Secure project.

4.18.6 How to configure OEM-uRoT (STiRot uROT) boot path

- Select an STM32H57x MCU
- Create a project with TrustZone activated (TZEN = 1), see *Figure 283*
- In Project Manager, save the project, see *Figure 284*
- Go to “Boot Path and Debug Authentication” tab, and press the Select button, see *Figure 285*
- Select “ST immutable Root of Trust (ST-iRot)”, then click “NEXT”, see *Figure 286*
- Select “OEM updatable Root of Trust (OEM-uRoT)”, then click “NEXT”, see *Figure 286*
- Select “Secure Application”, then click “FINISH”, see *Figure 287*
- The panel of boot path configuration is displayed, use it to configure the boot path in the “Boot Path and Debug Authentication” tab, see *Figure 288*
- Generate and build the project, see *Figure 291* and *Figure 292*

Figure 283. Project creation

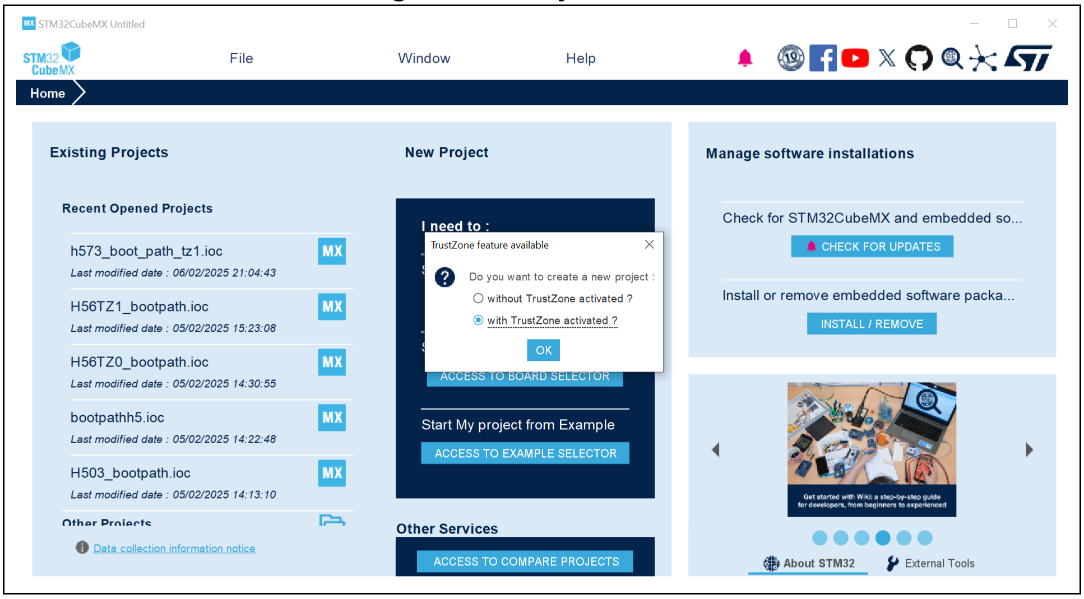


Figure 284. Save the project

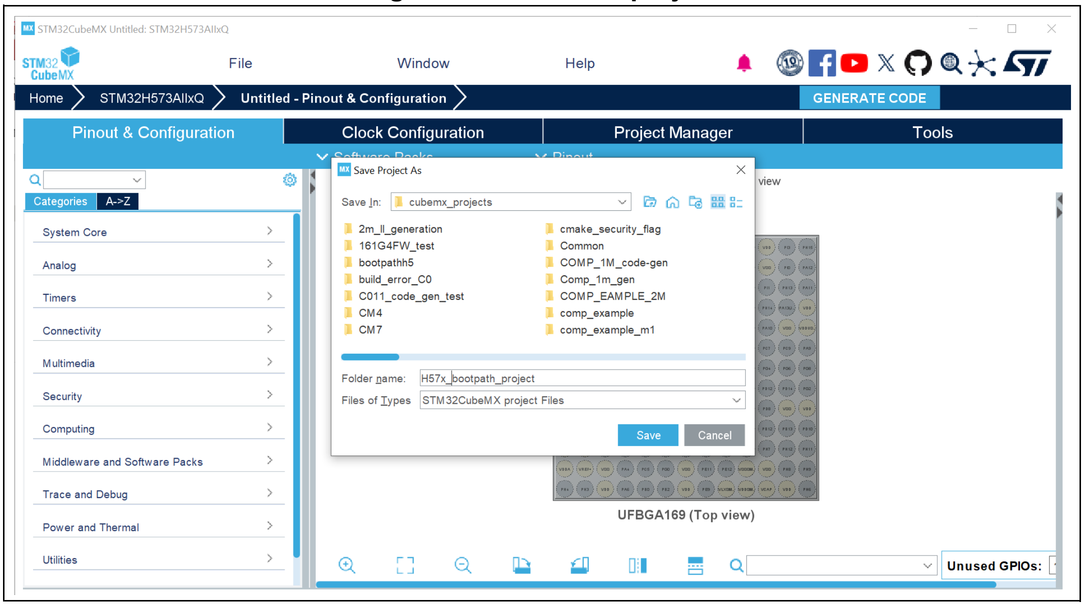


Figure 285. Boot path and debug authentication panel

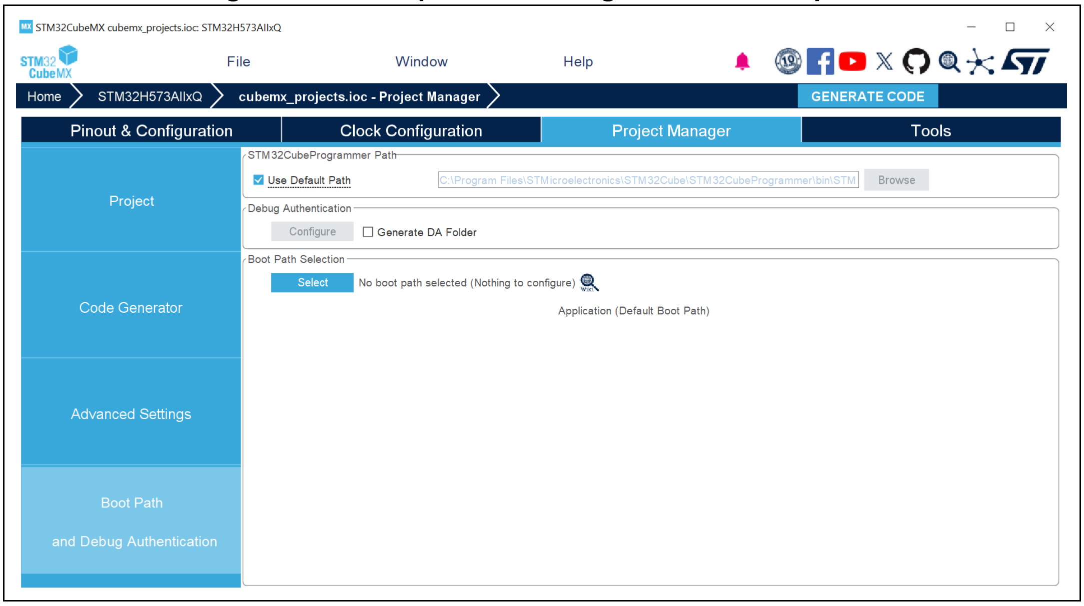


Figure 286. First (left) and second (right) boot path stage

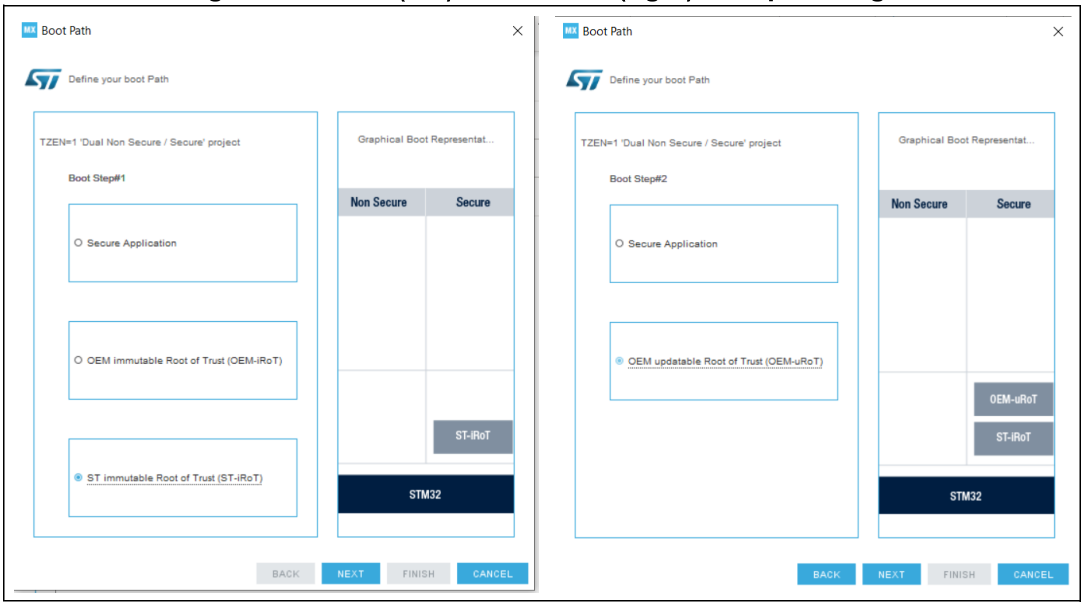


Figure 287. Final boot path stage

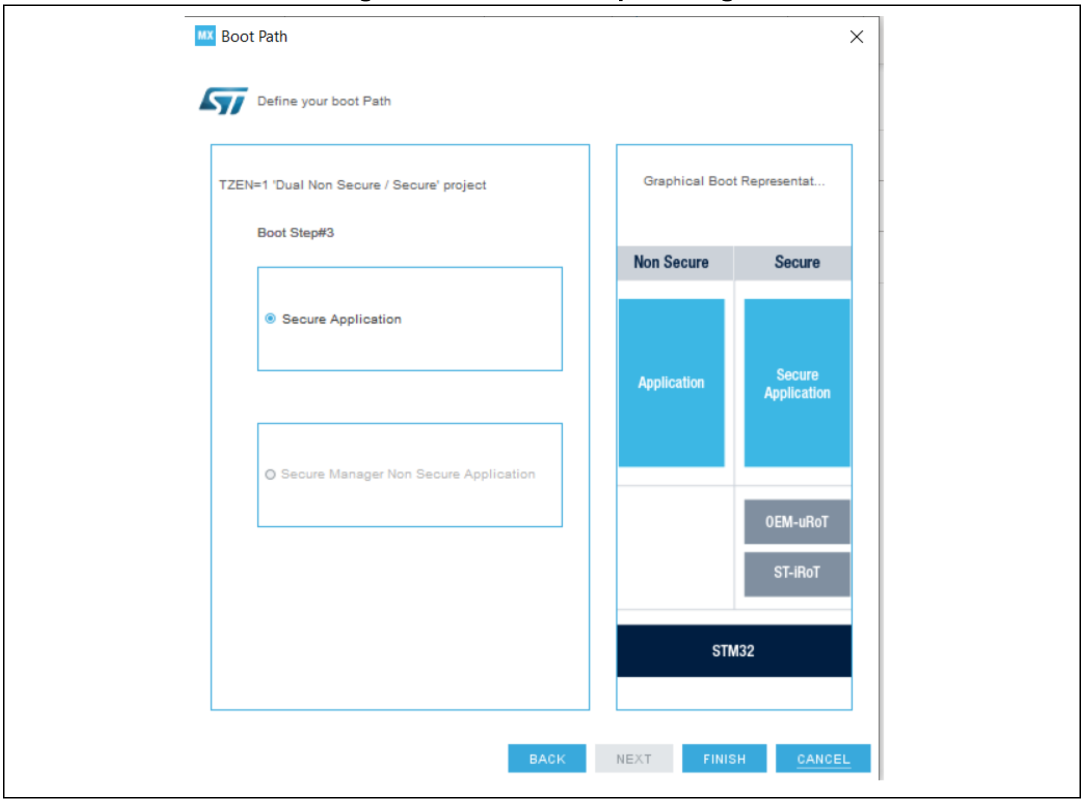


Figure 288. Boot path and debug authentication tab

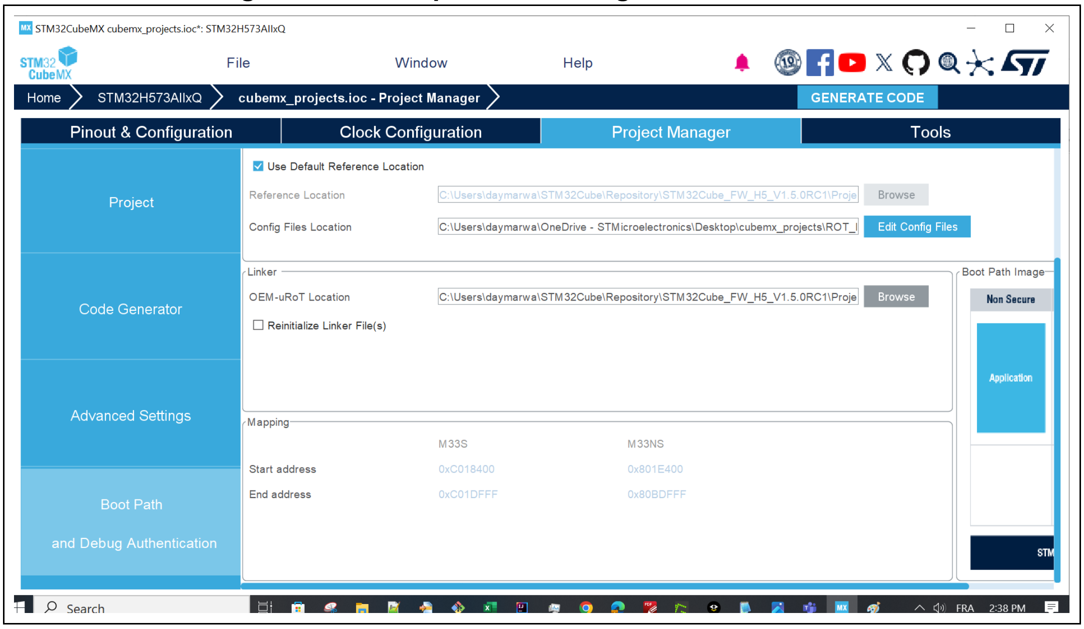


Figure 289. map.properties file

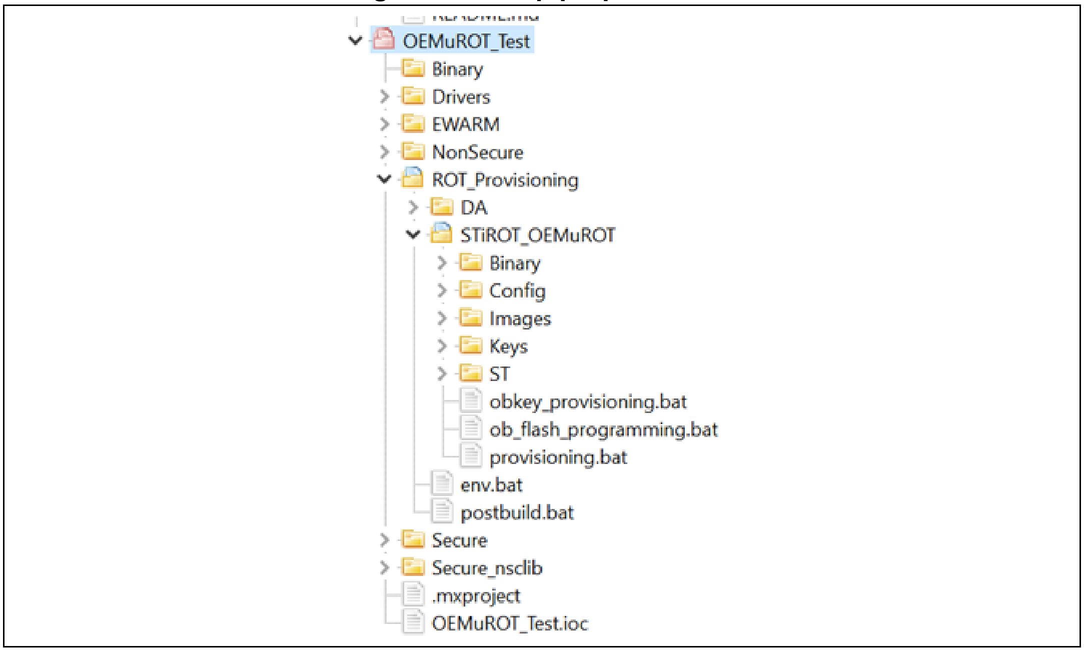


Figure 290. Code generation with EWARM

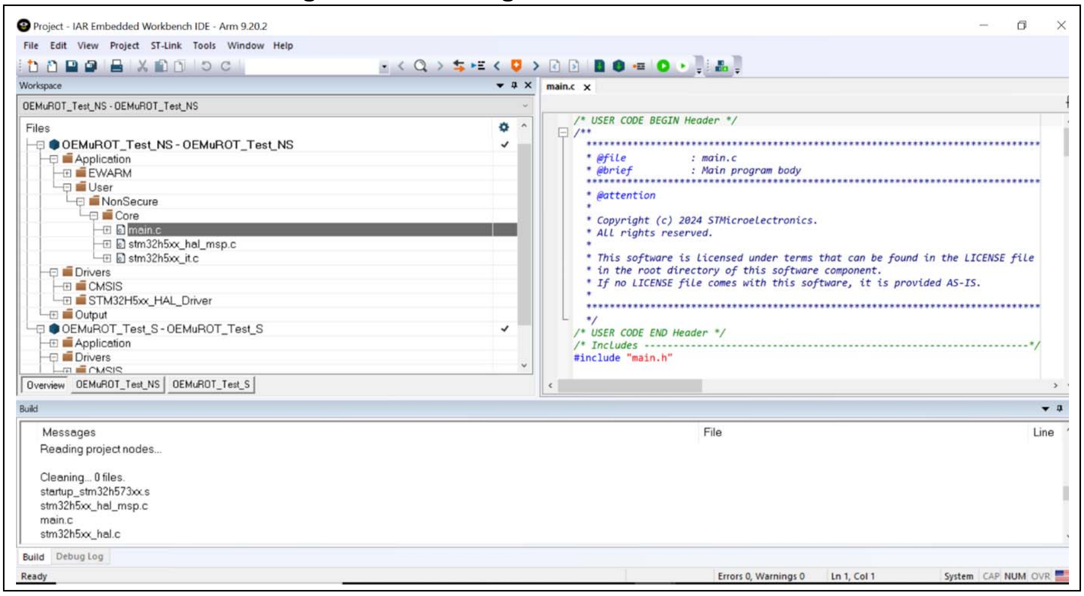


Figure 291. Nonsecure generated project

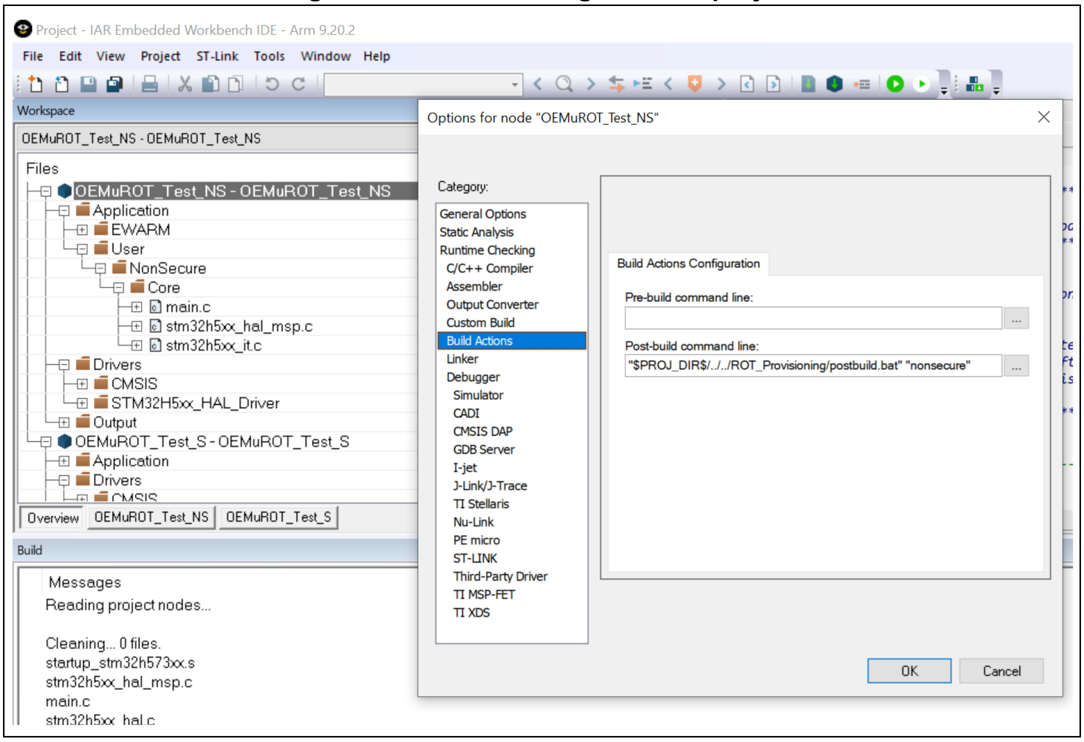
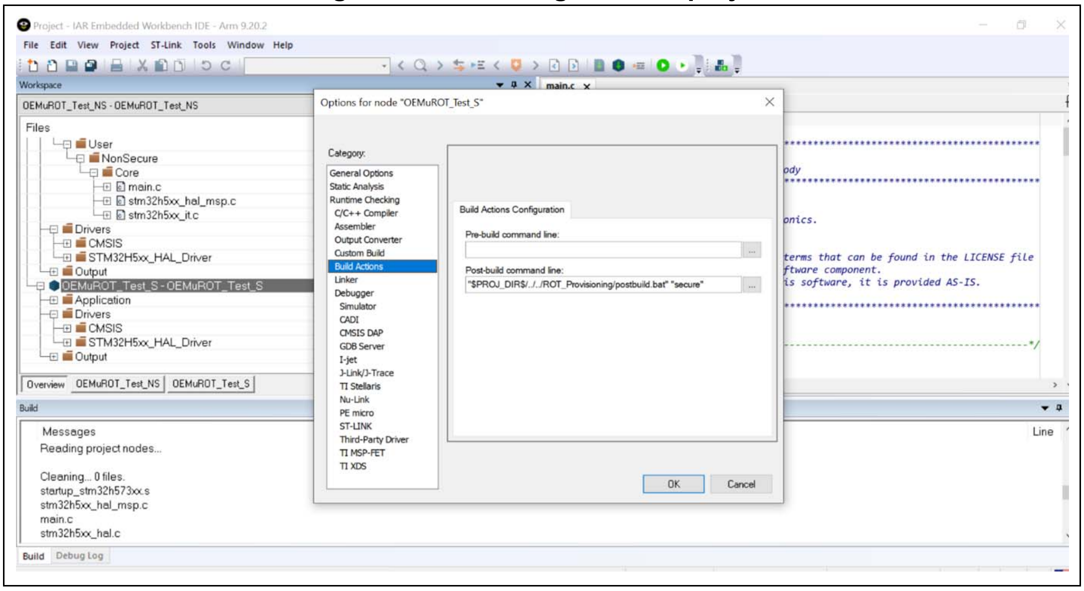


Figure 292. Secure generated project



4.18.7 How to configure ST-iRoT boot path with STM32H7RS devices

Go through the following steps:

1. Select an STM32H7S3Vx MCU (*Figure 293*)
2. A popup (see *Figure 294*) asks to preconfigure the Memory Protection Unit. It is recommended to optimize the speculative read access of the core. Select "Yes" to keep the default configuration.
3. In Project Manager Window, check only "Appli Project", name the project, and save it (*Figure 295*).
4. Go to "Boot Path and Debug Authentication" tab and press the Select button (*Figure 296*).
5. Select "ST immutable Root of Trust (ST-iRoT)", then click "NEXT" (*Figure 297*).
6. Select "Application", then click "FINISH" (*Figure 298*).
7. The panel of boot path configuration is displayed (see *Figure 299*), use it to configure the boot path in the "Boot Path and Debug Authentication" tab.
8. Generate and build the project (see *Figure 300*).

Figure 293. Boot path project

Figure 294. Use default configuration

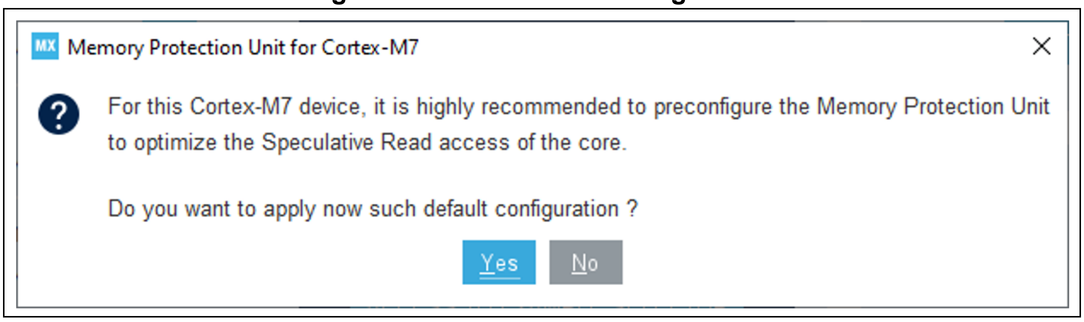


Figure 295. Configure the project

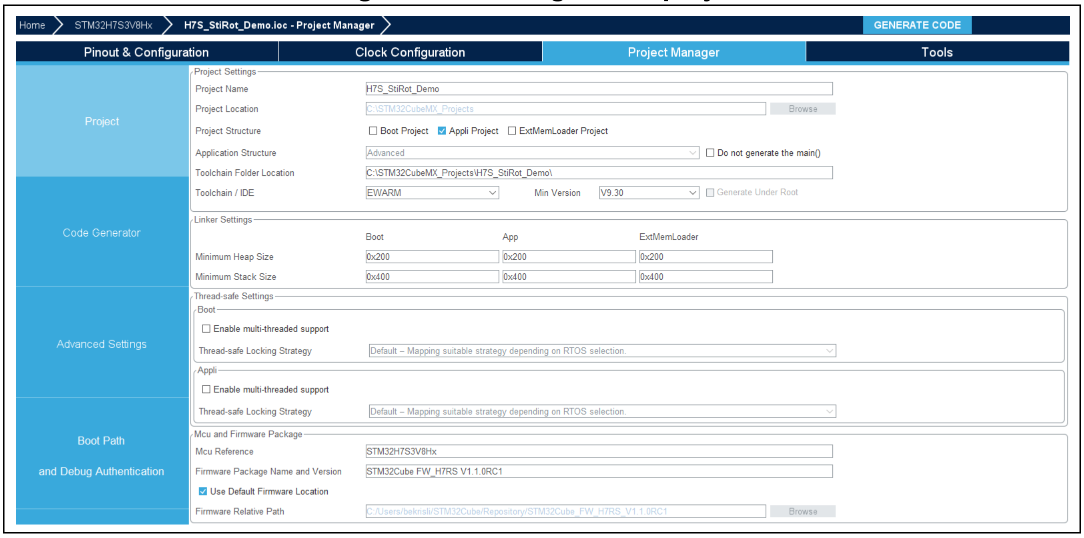


Figure 296. Select the project

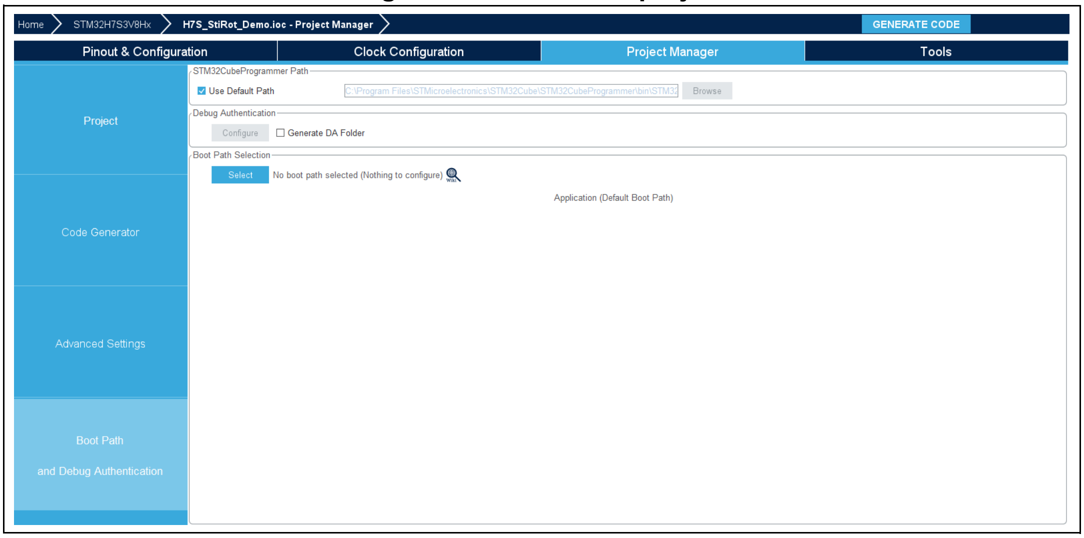


Figure 297. First boot path stage

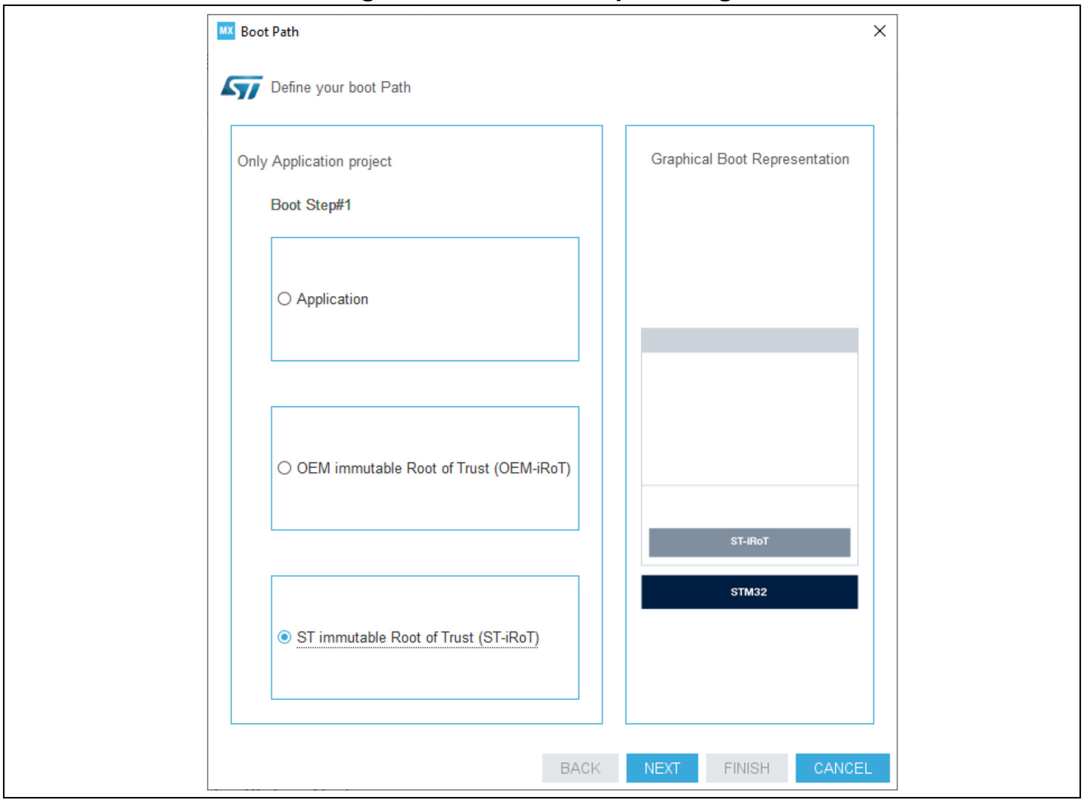


Figure 298. Final boot path stage

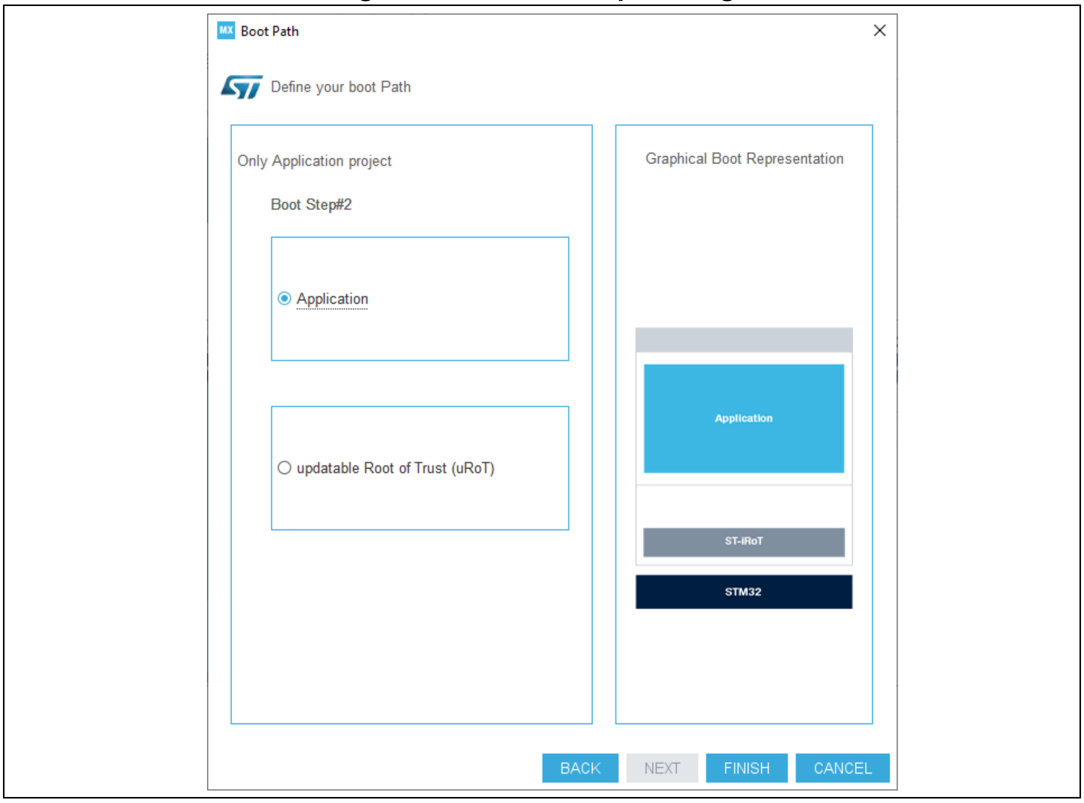


Figure 299. Boot path and debug authentication panel

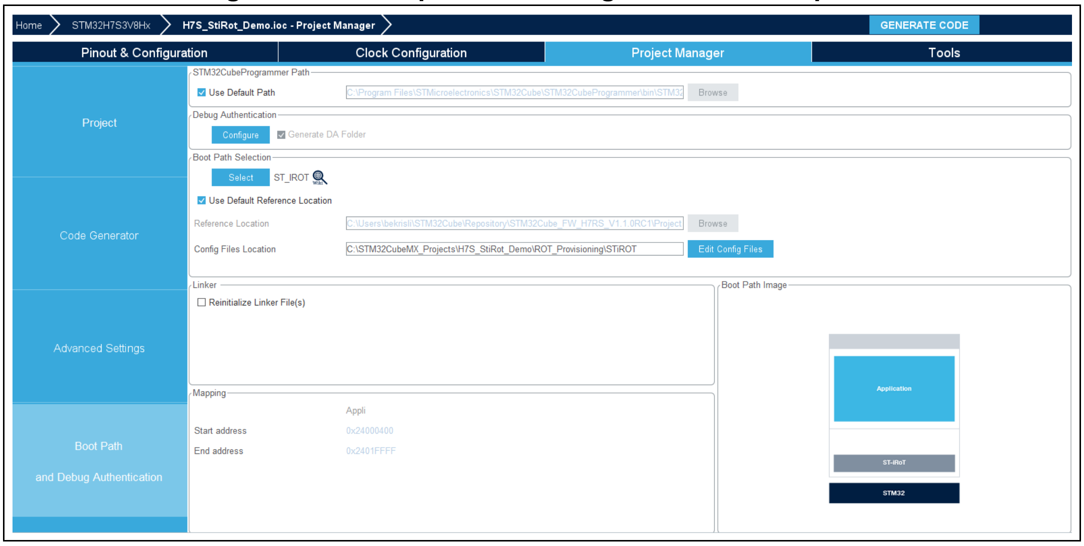


Figure 300. Generate the code

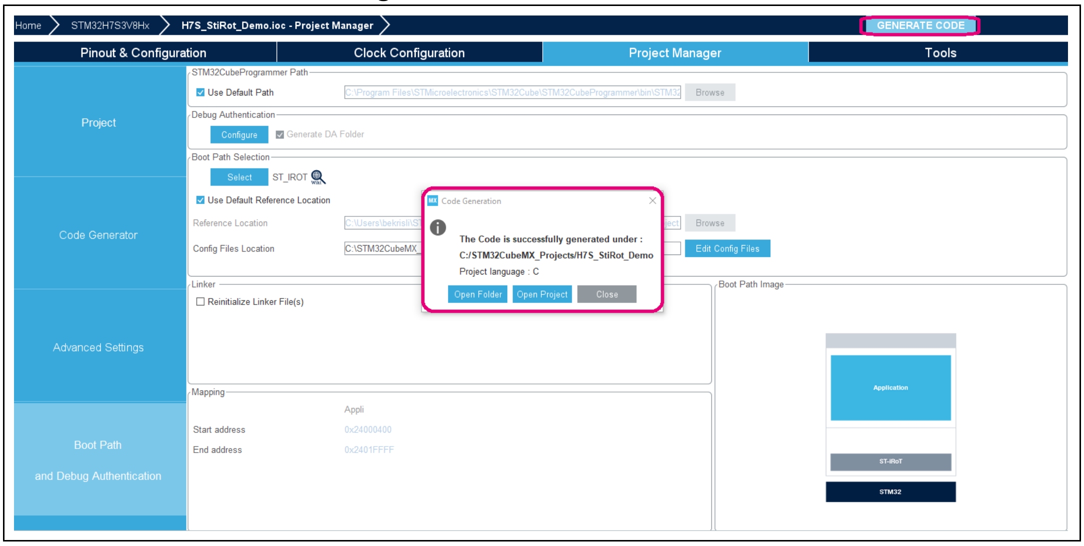
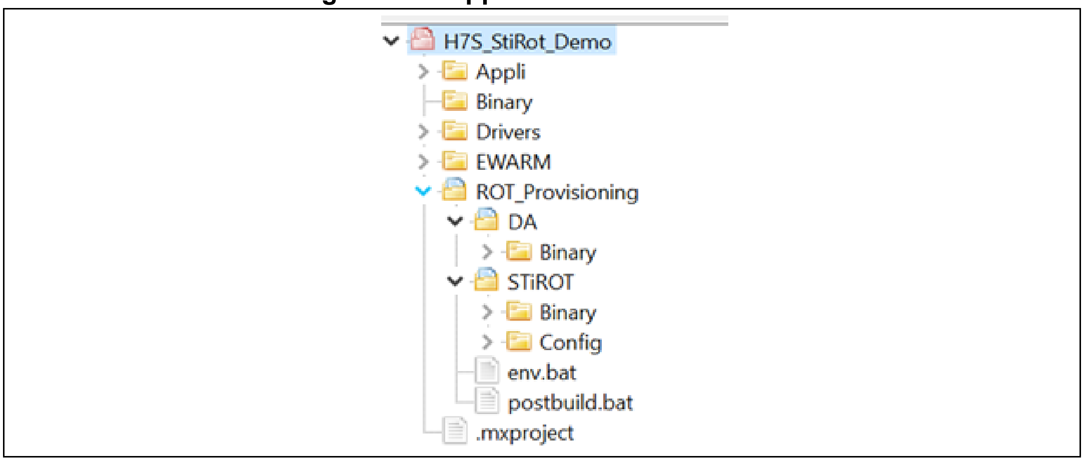


Figure 301. Application IDE directories



4.19 About window

This window displays STM32CubeMX version information. To open it, select **Help > About** from the STM32CubeMX menu bar.

Figure 302. About window

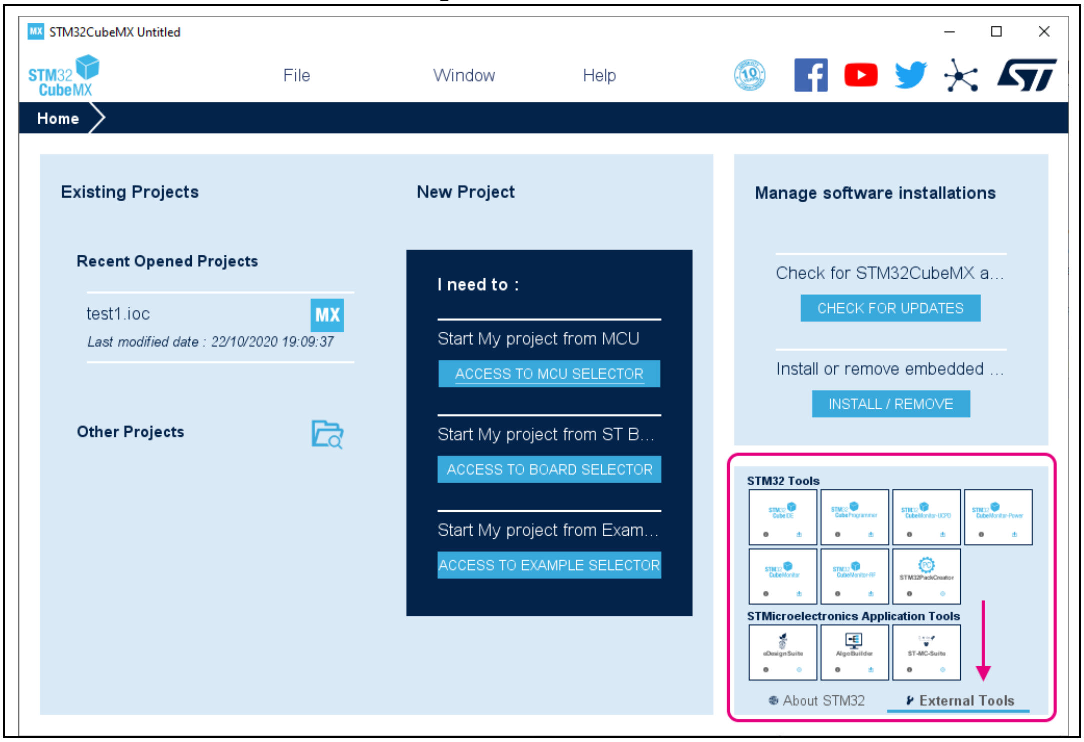
5 STM32CubeMX tools

5.1 External Tools

This panel is accessible from the home page. It provides an overview of the tools relevant for the STM32 product portfolio (see *Figure 303*):

- click to open the tool information note
- click to open the tool webpage on *www.st.com*
- click to launch the tool.

Figure 303. ST Tools



5.2 Compare Projects

This new feature is designed to enable the comparison of two projects, based on the same or on different microcontrollers. This tool allows users to efficiently analyze and correlate similarities and differences in IP configurations and project structure between two projects.

5.2.1 User interface of the Compare Projects tool

The user can activate this function from the Tools panel by clicking the **Compare Projects** field (see *Figure 304*), or from the home page (see *Figure 305*) by clicking ACCESS TO COMPARE PROJECT under **Other services** (even before creating any project).

Figure 304. Reaching Compare Project from the Tools panel

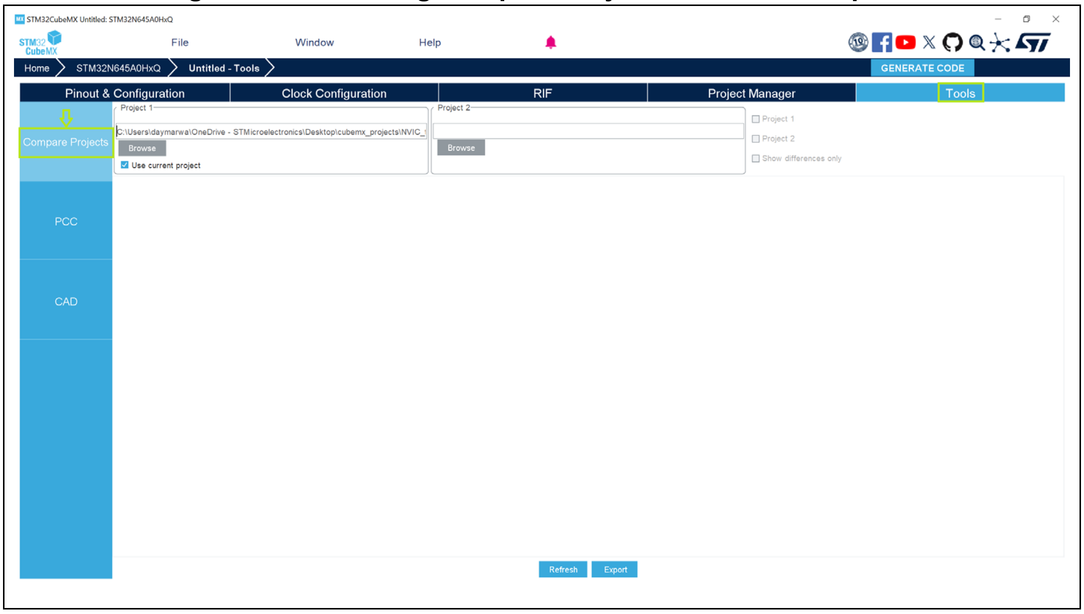
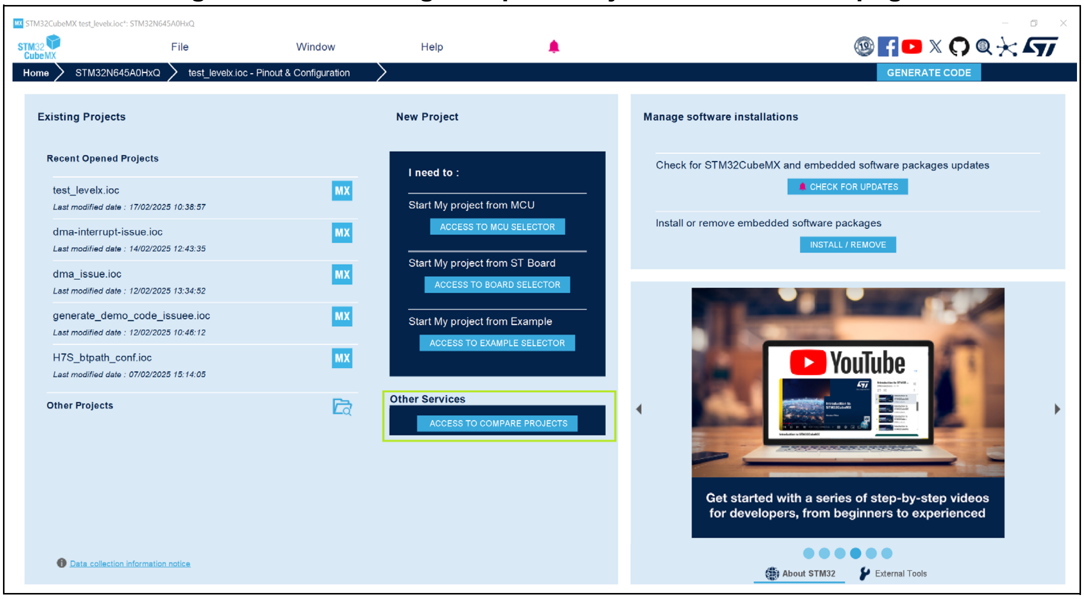


Figure 305. Reaching Compare Project from the home page



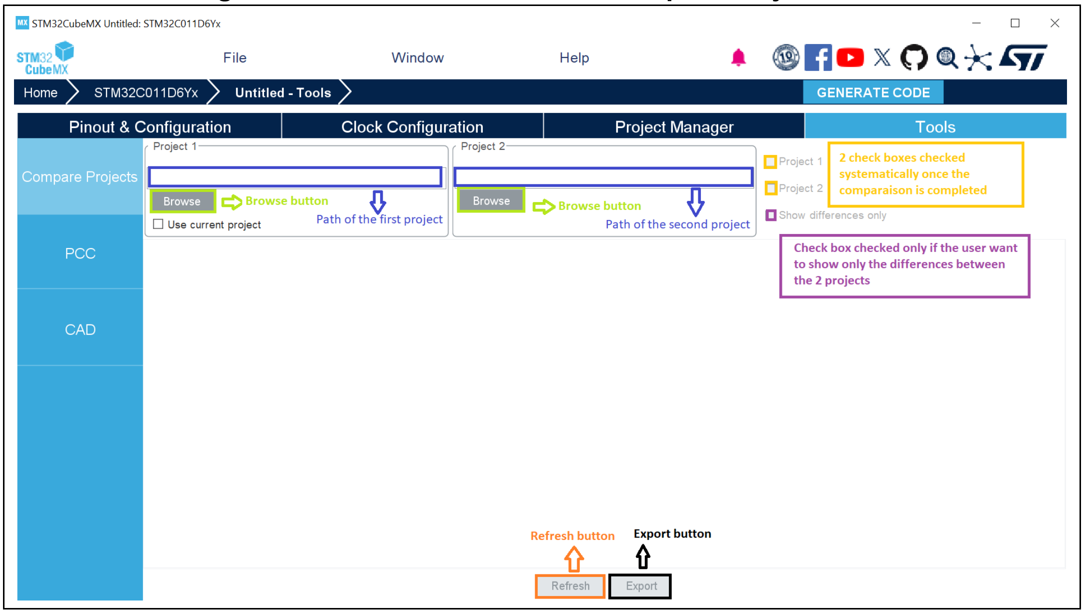
The tool is composed of:

- Two main panels, named Project 1 and Project 2.
  - Each field contains a “Browse” button from where we can load the desired .ioc file.
  - The check box “Use Current project” in the first panel is to use the opened project in STM32CubeMX instead of loading a saved one in the local device.
- Three check boxes, named Project 1, Project 2, and “Show differences only”:
  - Once the second project is loaded, the Project 1 and Project 2 check boxes are checked systematically, indicating that the output table contains all the parameters of Project 1 and Project 2 (whether they are different or not).

– If the user wants to see only the parameters that are different between the two projects, it can check “Show differences only”.

- Refresh button: once clicked, it performs an instantiated comparison.
- Export button: allows users to save or transfer the result of the comparison to an external file in Excel format.

Figure 306. User interface of the Compare Projects tool



5.2.2 Comparing two projects

1. Load the first project from the local device (*Figure 307*).
   – Once the first .ioc is loaded, a popup appears to indicate the need to upload the second .ioc.
2. Load the second project (*Figure 308*).
   – After uploading the second .ioc file, the output is displayed in the UI (*Figure 309*).
   – If the user uploads the same .ioc file (having the same path) into the two fields, a popup appears to indicate that it is irrelevant to perform a comparison (*Figure 310*).
   – If the two projects have the same structure and the user checks “Show differences only”, only the headers of the respective tables “Target”, “Peripherals & Middleware”, and “Project Settings” are shown, no data are displayed (*Figure 311*).

Figure 307. Load the first .ioc file

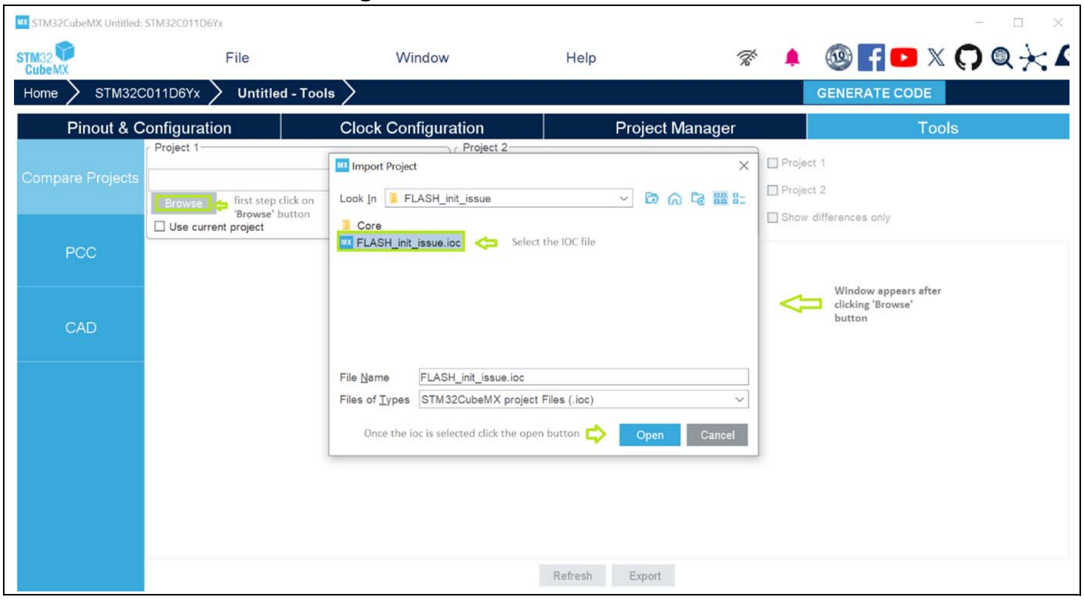


Figure 308. Starting the comparison

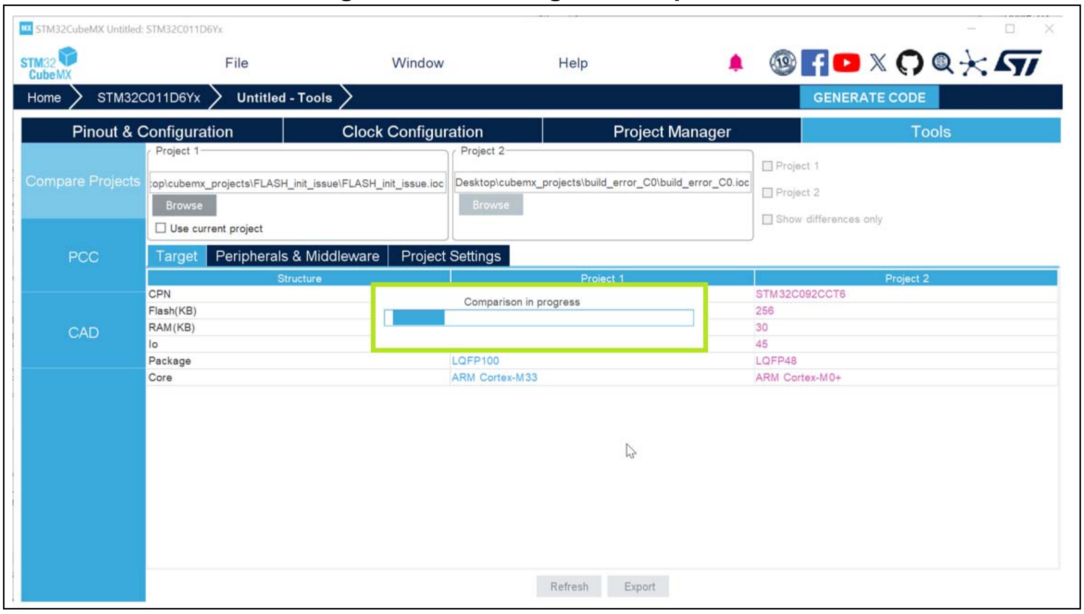


Figure 309. Result of the comparison

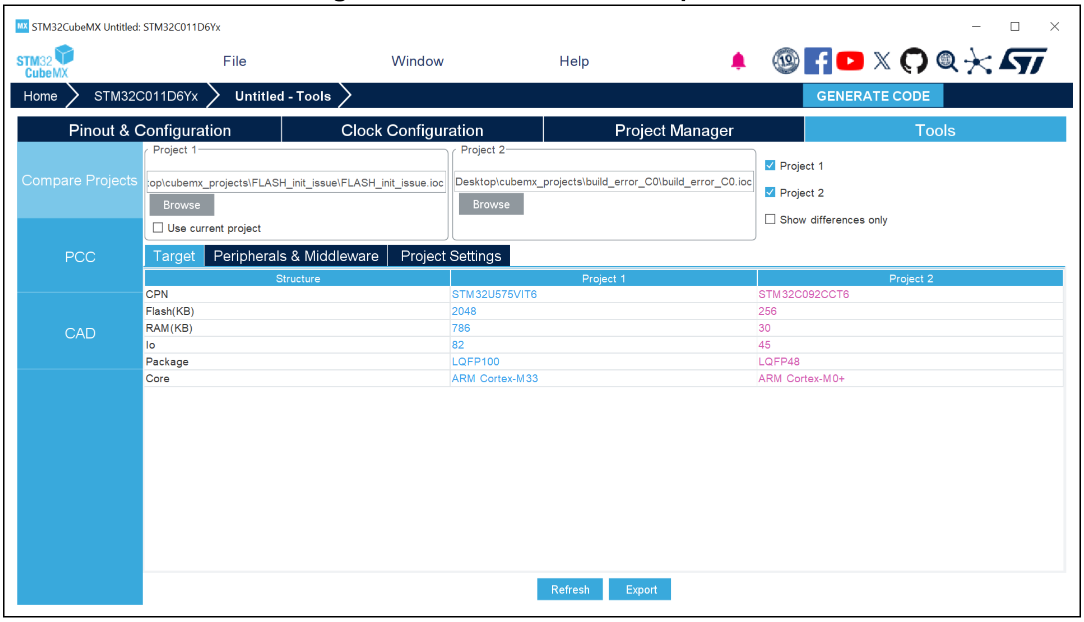


Figure 310. Loading the same project

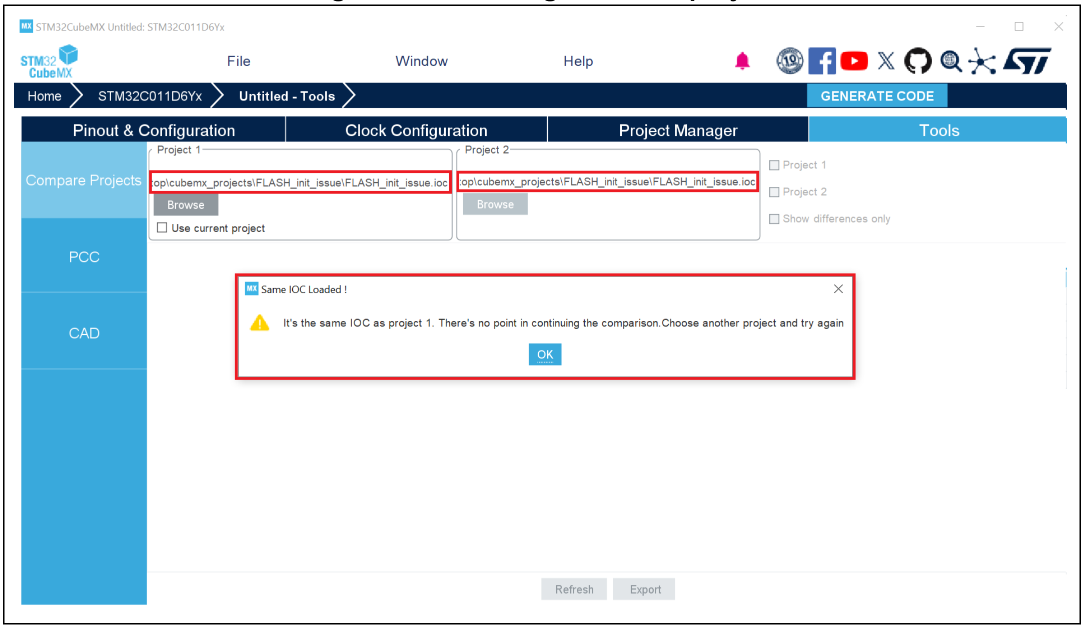
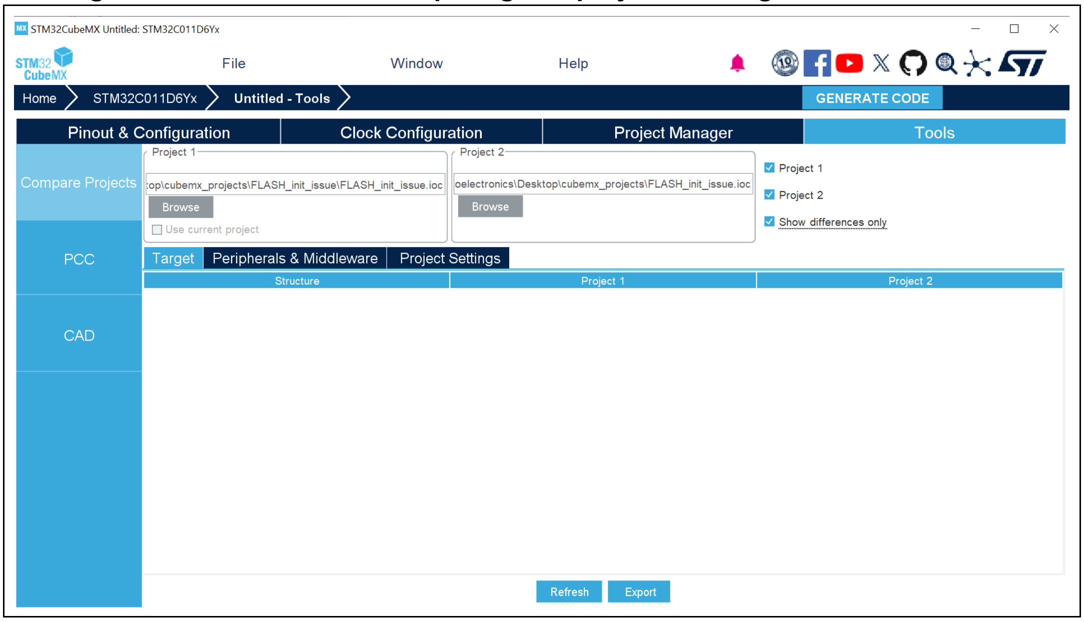


Figure 311. The result of comparing two projects having the same structure



The user has the following options for comparing a current project:

1. Load a project from local and start comparing it as the current project (Project 1).
2. Compare a newly created project (after configuration), even unsaved, with a saved project, see *Figure 312*.
3. Compare a currently open project with itself (the popup blocking the comparison of two similar projects does not appear), see *Figure 313*:
   a) Load a the first project saved on the local as the current project (current project automatically ticked and the configuration can be modified after loading)
   b) Load the same project file from the local in the Project 2 field.

Figure 312. Compare the current non saved project with another project

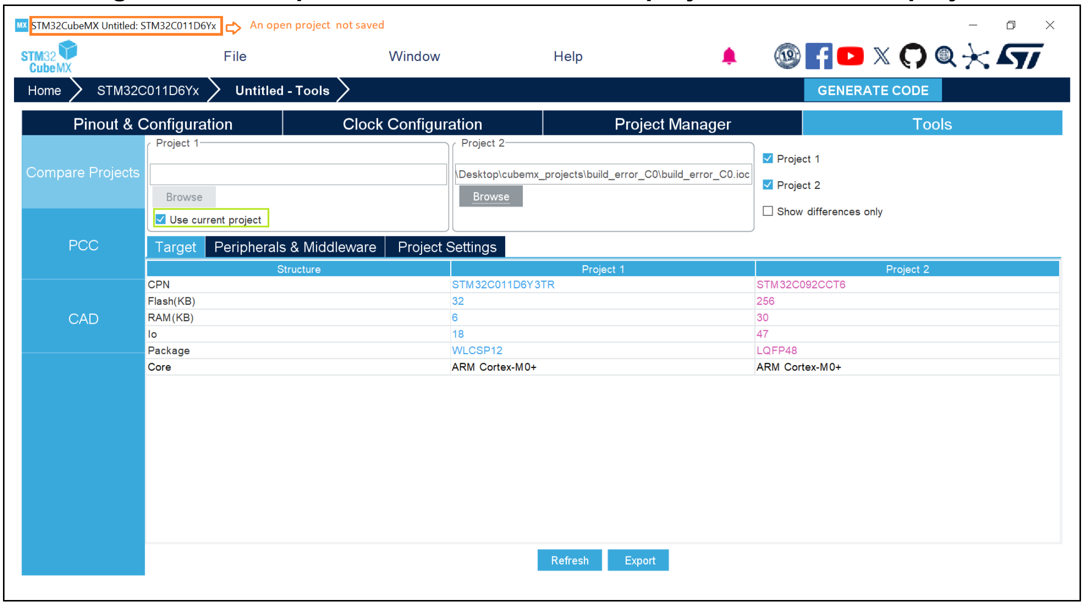
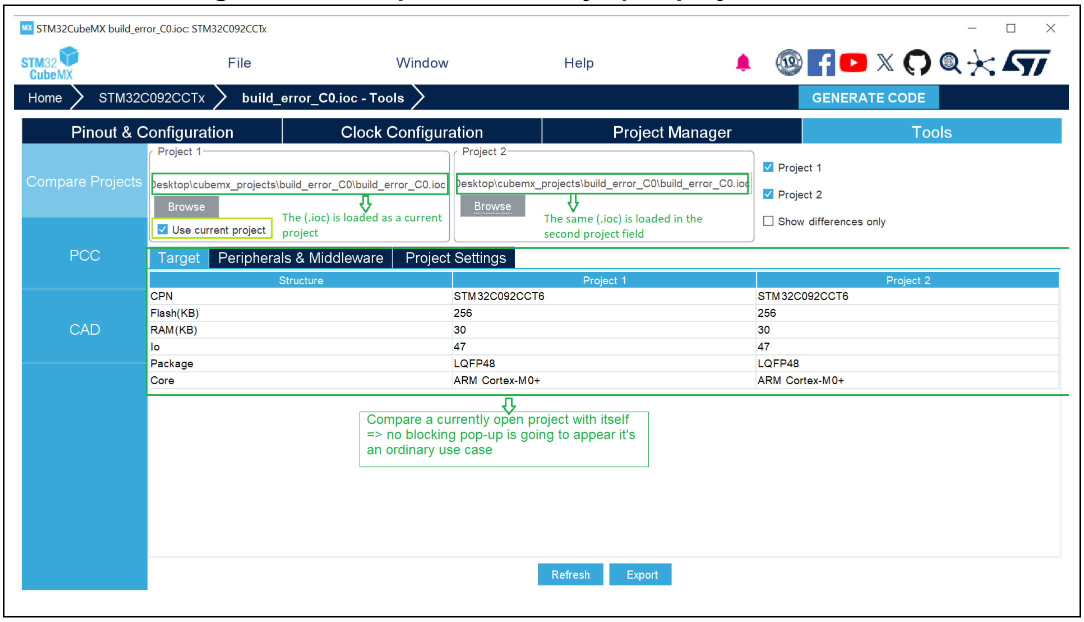


Figure 313. Compare a currently open project with itself



5.2.3 The output of the comparison

After starting the comparison, the following elements are added to the UI:

Target table (*Figure 314*): provides a comparison of the parameters used in each project. It clearly shows differences in:

- Part numbers
- Number of IOs
- Package types
- Core configurations
- Available flash memory sizes

The Target table is composed of 3 column:

- Structure: a listing of the MCU parameters.
- Project 1: the values corresponding to the MCU parameters of the first project.
- Project 2: the values corresponding to the MCU parameters of the second project.

The user has the option to:

- Side-by-side comparison: showing data from both projects simultaneously.
- Individual inspection: inspect each file separately by selecting the Project 1 or Project 2 checkbox.
- Focus on differences: exclusively view the differences between 2 projects by checking 'Show differences only'.

The Target table offers a quick and straightforward overview of key differences between the two projects. It represents an invaluable tool for project migrations and initial hardware evaluations.

Peripherals & Middleware table (*Figure 315*): displays the differences and similarities in the configuration of each peripheral or middleware used in the two projects (the sub-parameters and the corresponding values).

The data are presented by lines. It is composed of the following columns:

- Category name
- IP name
- Mode
- Parameters settings
- Project 1
- Project 2

The table includes a highlighting feature, which uses color coding to visually differentiate parameters:

- Specific parameters for project 1 are marked in blue color
- Specific parameters for project 2 are marked in pink color
- The common parameters for the two projects are marked in black color

All peripheral categories are displayed collectively, with the option of sorting in alphabetical order.

Project Settings table (*Figure 316*), helping the users to know the software environment required for each project, and to determine the necessary tools for ensuring project compatibility and facilitating migration. The table contains information about the firmware package used for each MCU and about the toolchain used for building each project.

It is composed of three columns:

- Settings:
  - CustomerFirmwarePackage
  - FirmewarePackage
  - ProjectStructure
  - TargetToolchain
- Project 1
- Project 2

Figure 314. Target table

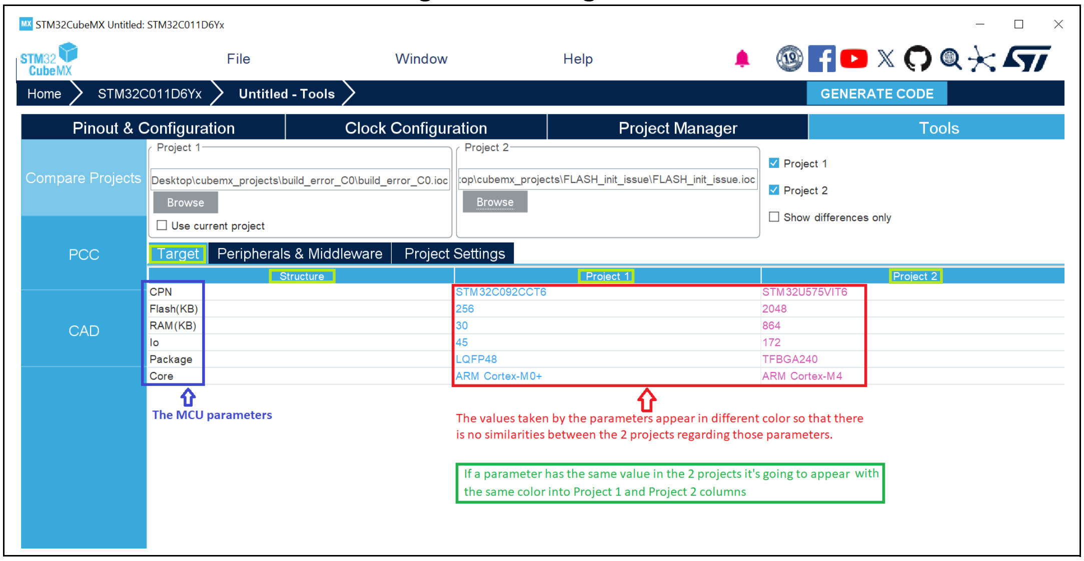


Figure 315. Peripherals & Middleware table

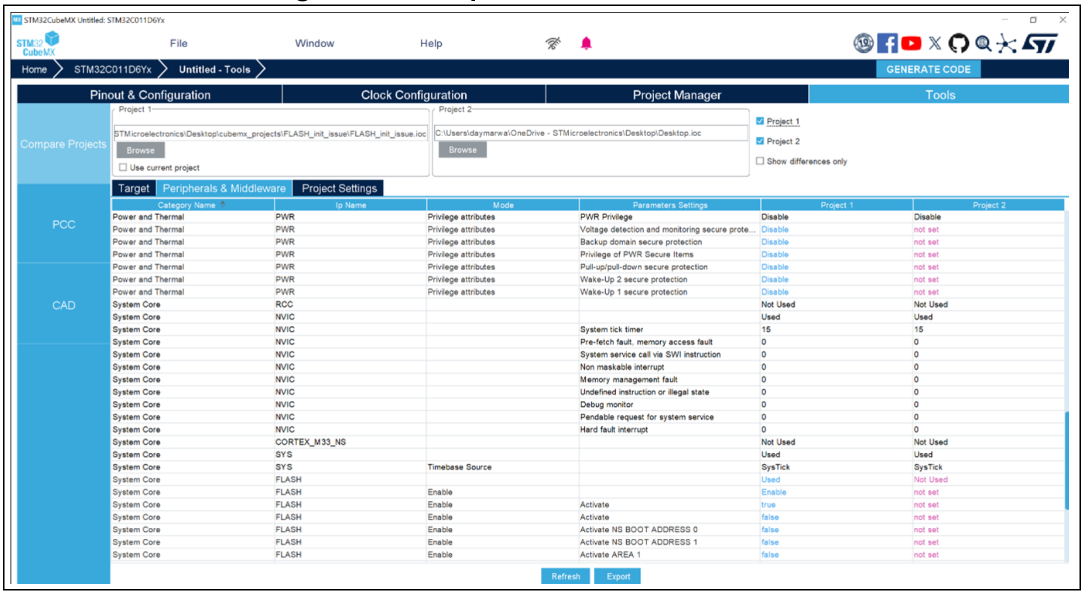
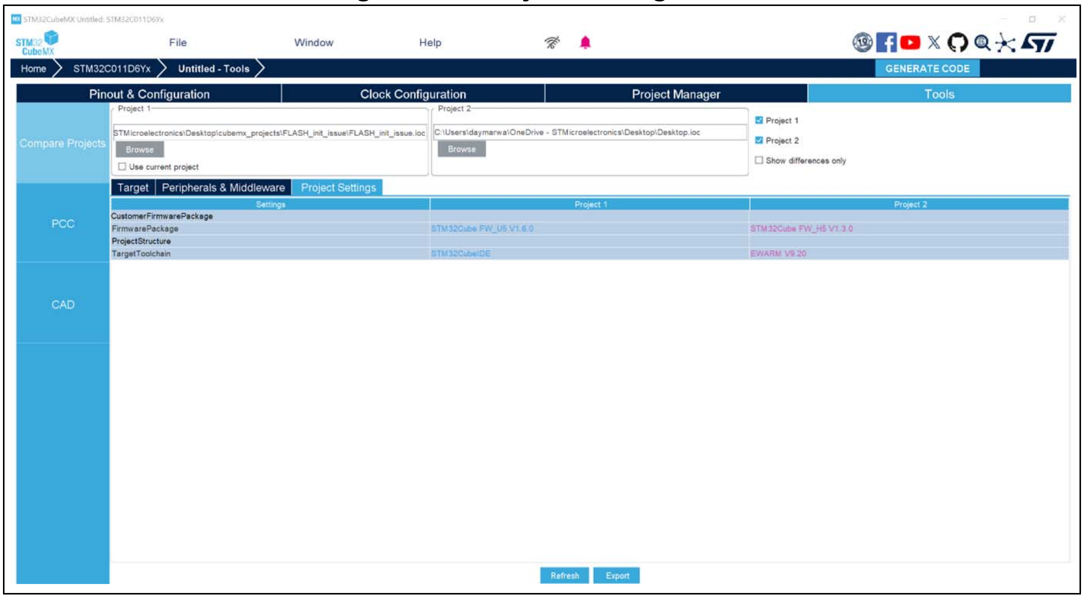


Figure 316. Project Settings table



5.2.4 Saving the comparison result of the two projects

In the user interface of the “Compare Projects” tool, there is an Export button that allows users to save the result of the comparison in an external Excel file.

By clicking the Export button, a window named Save appears to allow the user to choose a name for the resulting file and save it (*Figure 317*).

The available format:

- The result is exported into three sheets in an Excel format (*Figure 318*).
- Each sheet represents a table (Target, Peripherals & Middleware, Projects Settings).

If the user wants to get only the differences in the exported file, they should click on “Show differences only”.

Figure 317. Choosing the Excel format to save the comparison result

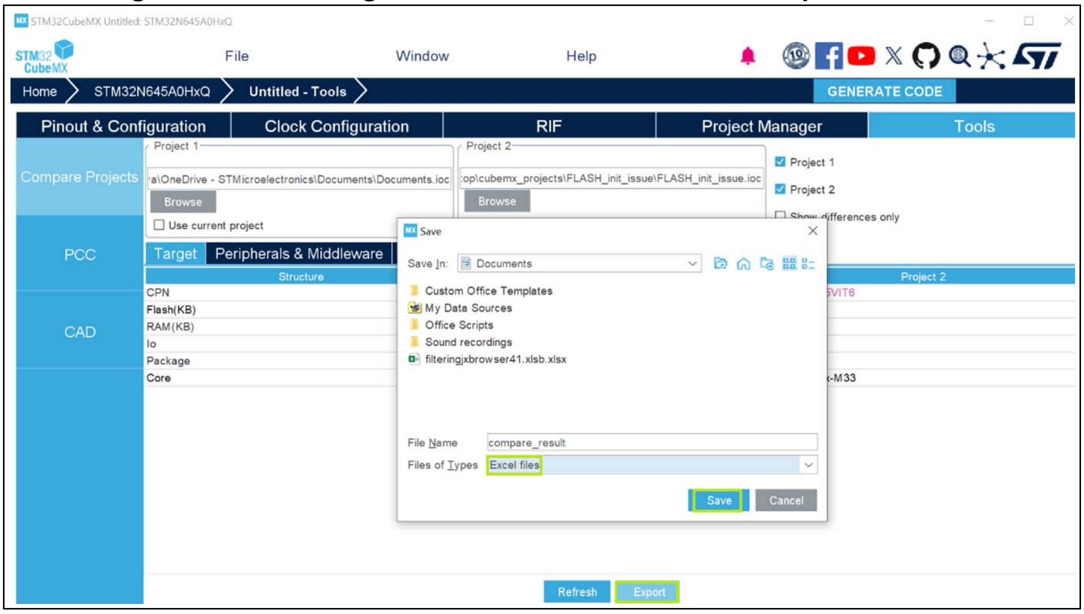


Figure 318. Comparison result in Excel format

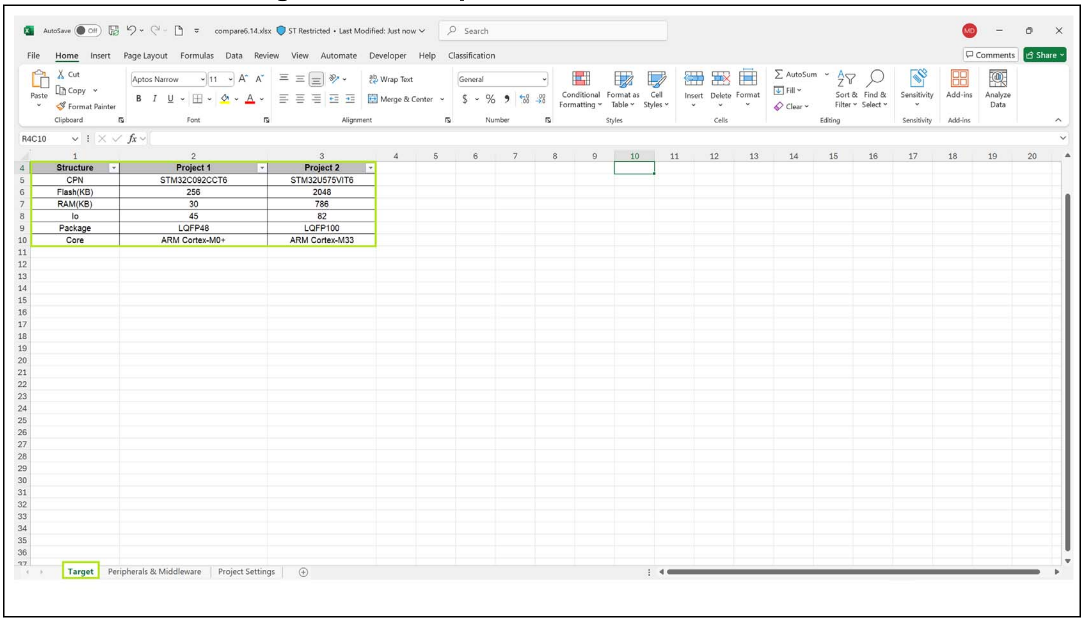
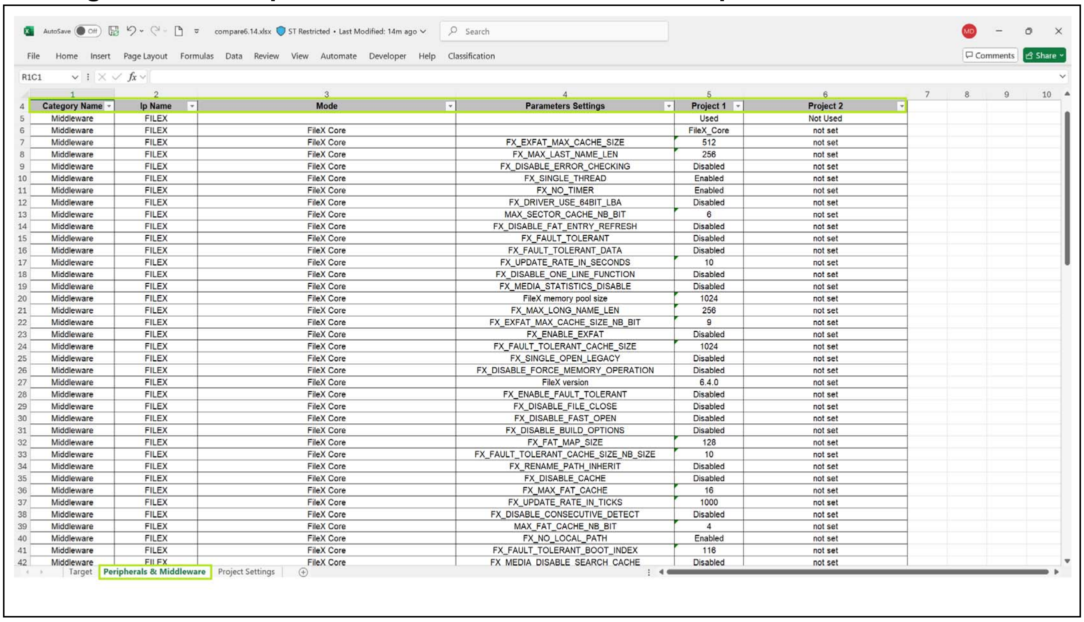
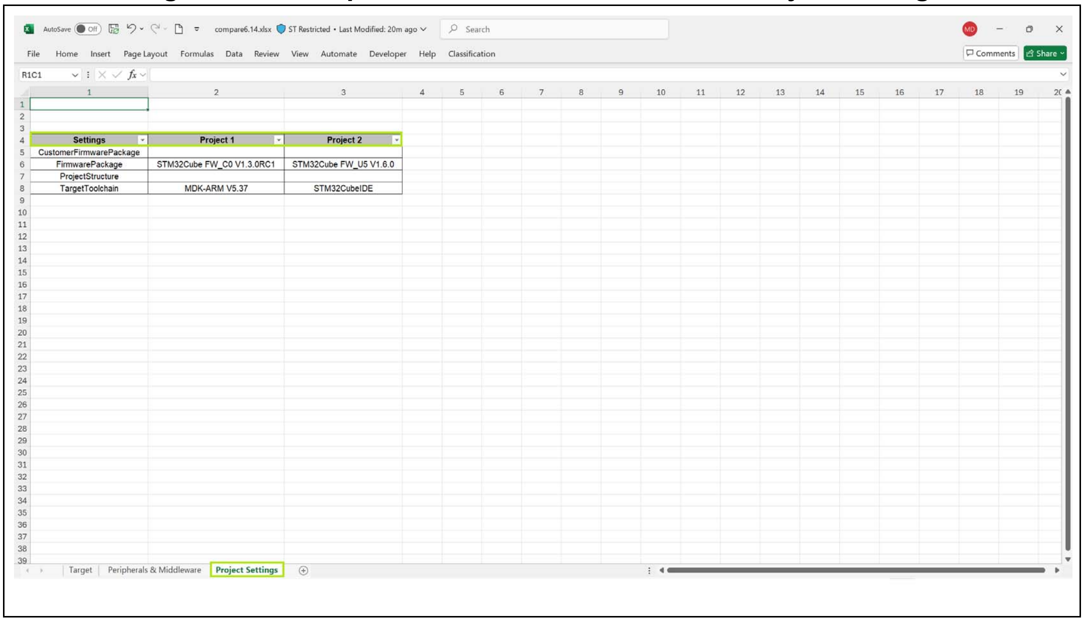


Figure 319. Comparison result in Excel format - Peripherals and middleware



Category Name	Ip Name	Mode	Parameters Settings	Project 1	Project 2
Middleware	FILEX			Used	Not Used
Middleware	FILEX	FileX Core		FileX_Core	not set
Middleware	FILEX	FileX Core	FX_EXFAT_MAX_CACHE_SIZE	512	not set
Middleware	FILEX	FileX Core	FX_MAX_LAST_NAME_LEN	256	not set
Middleware	FILEX	FileX Core	FX_DISABLE_ERROR_CHECKING	Disabled	not set
Middleware	FILEX	FileX Core	FX_SINGLE_THREAD	Enabled	not set
Middleware	FILEX	FileX Core	FX_NO_TIMER	Enabled	not set
Middleware	FILEX	FileX Core	FX_DRIVER_USE_64BIT_LBA	Disabled	not set
Middleware	FILEX	FileX Core	MAX_SECTOR_CACHE_NB_BIT	6	not set
Middleware	FILEX	FileX Core	FX_DISABLE_FAT_ENTRY_REFRESH	Disabled	not set
Middleware	FILEX	FileX Core	FX_FAULT_TOLERANT	Disabled	not set
Middleware	FILEX	FileX Core	FX_FAULT_TOLERANT_DATA	Disabled	not set
Middleware	FILEX	FileX Core	FX_UPDATE_RATE_IN_SECONDS	10	not set
Middleware	FILEX	FileX Core	FX_DISABLE_ONE_LINE_FUNCTION	Disabled	not set
Middleware	FILEX	FileX Core	FX_MEDIA_STATISTICS_DISABLE	Disabled	not set
Middleware	FILEX	FileX Core	FileX memory pool size	1024	not set
Middleware	FILEX	FileX Core	FX_MAX_LONG_NAME_LEN	256	not set
Middleware	FILEX	FileX Core	FX_EXFAT_MAX_CACHE_SIZE_NB_BIT	9	not set
Middleware	FILEX	FileX Core	FX_ENABLE_EXFAT	Disabled	not set
Middleware	FILEX	FileX Core	FX_FAULT_TOLERANT_CACHE_SIZE	1024	not set
Middleware	FILEX	FileX Core	FX_SINGLE_OPEN_LEGACY	Disabled	not set
Middleware	FILEX	FileX Core	FX_DISABLE_FORCE_MEMORY_OPERATION	Disabled	not set
Middleware	FILEX	FileX Core	FileX version	6.4.0	not set
Middleware	FILEX	FileX Core	FX_ENABLE_FAULT_TOLERANT	Disabled	not set
Middleware	FILEX	FileX Core	FX_DISABLE_FILE_CLOSE	Disabled	not set
Middleware	FILEX	FileX Core	FX_DISABLE_FAST_OPEN	Disabled	not set
Middleware	FILEX	FileX Core	FX_DISABLE_BUILD_OPTIONS	Disabled	not set
Middleware	FILEX	FileX Core	FX_FAT_MAP_SIZE	128	not set
Middleware	FILEX	FileX Core	FX_FAULT_TOLERANT_CACHE_SIZE_NB_SIZE	10	not set
Middleware	FILEX	FileX Core	FX_RENAME_PATH_INHERIT	Disabled	not set
Middleware	FILEX	FileX Core	FX_DISABLE_CACHE	Disabled	not set
Middleware	FILEX	FileX Core	FX_MAX_FAT_CACHE	16	not set
Middleware	FILEX	FileX Core	FX_UPDATE_RATE_IN_TICKS	1000	not set
Middleware	FILEX	FileX Core	FX_DISABLE_CONSECUTIVE_DETECT	Disabled	not set
Middleware	FILEX	FileX Core	MAX_FAT_CACHE_NB_BIT	4	not set
Middleware	FILEX	FileX Core	FX_NO_LOCAL_PATH	Enabled	not set
Middleware	FILEX	FileX Core	FX_FAULT_TOLERANT_BOOT_INDEX	116	not set
Middleware	FILEX	FileX Core	FX_MEDIA_DISABLE_SEARCH_CACHE	Disabled	not set

Figure 320. Comparison result in Excel format - Project settings



Settings	Project 1	Project 2
CustomerFirmwarePackage		
FirmwarePackage	STM32Cube FW_C0 V1.3.0RC1	STM32Cube FW_U5 V1.6.0
ProjectStructure		
TargetToolchain	MDK-ARM V5.37	STM32CubeIDE

5.3 Power Consumption Calculator view

For an ever-growing number of embedded systems applications, power consumption is a major concern. To help minimizing it, STM32CubeMX offers the **Power Consumption Calculator** tab (see *Figure 321*), which, given a microcontroller, a battery model and a user-defined power sequence, provides the following results:

- Average current consumption

  Power consumption values can be taken from the datasheet or interpolated from a user specified bus or core frequency.
- Battery life
- Average DMIPs

  DMIPs values are directly taken from the MCU datasheet and are neither interpolated nor extrapolated.
- Maximum ambient temperature (T_{AMAX})

  According to the chip internal power consumption, the package type, and a maximum junction temperature of 105 °C, the tool computes the maximum ambient temperature to ensure good operating conditions.

  Current T_{AMAX} implementation does not account for I/O consumption. For an accurate estimate, I/O consumption must be specified using the Additional Consumption field. The formula for I/O dynamic current consumption is specified in the microcontroller datasheet.

The **Power Consumption Calculator** view allows developers to visualize an estimate of the embedded application consumption and lower it further at each power sequence step:

- make use of low power modes when available
- adjust clock sources and frequencies based on the step requirements
- enable only the peripherals necessary for each phase.

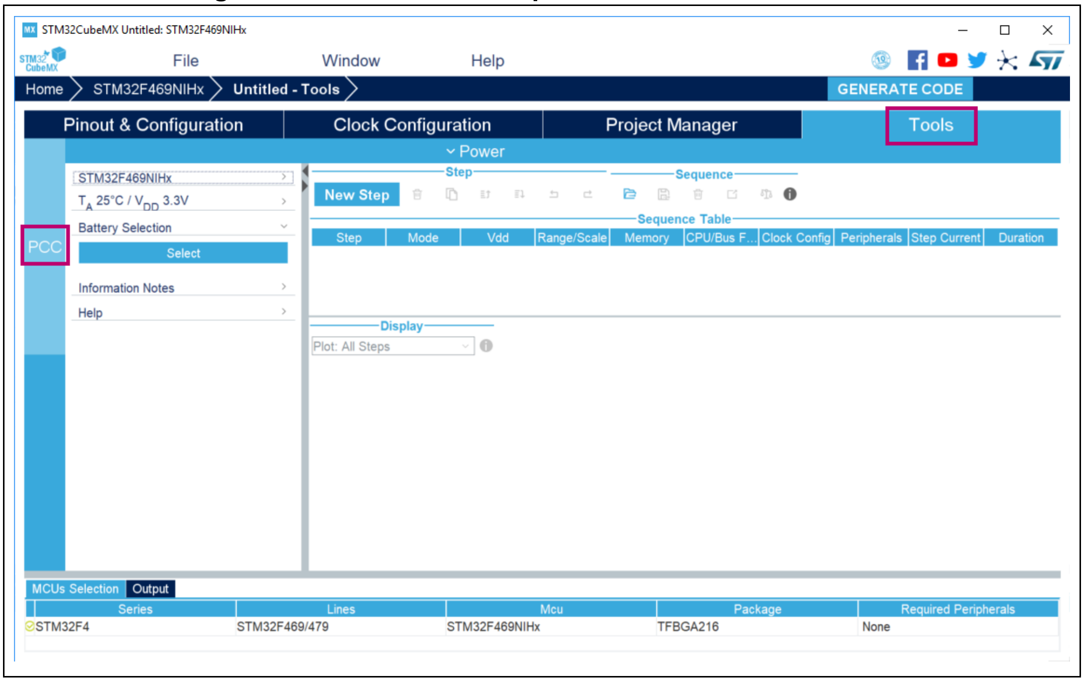
For each step the user can choose V_{BUS} as possible power source instead of the battery, impact battery life. If power consumption measurements are available at different voltage levels, STM32CubeMX also proposes a choice of voltage values (see *Figure 324*).

An additional option, the transition checker, is available for STM32L0, STM32L1, STM32L4, STM32L4+, STM32G0, STM32G4, STM32H7 and STM32WB series. When enabled, the transition checker detects invalid transitions within the currently configured sequence. It ensures that only possible transitions are proposed to the user when a new step is added.

5.3.1 Building a power consumption sequence

The default starting view is shown in *Figure 321*.

Figure 321. Power Consumption Calculator default view



Selecting a V_{DD} value

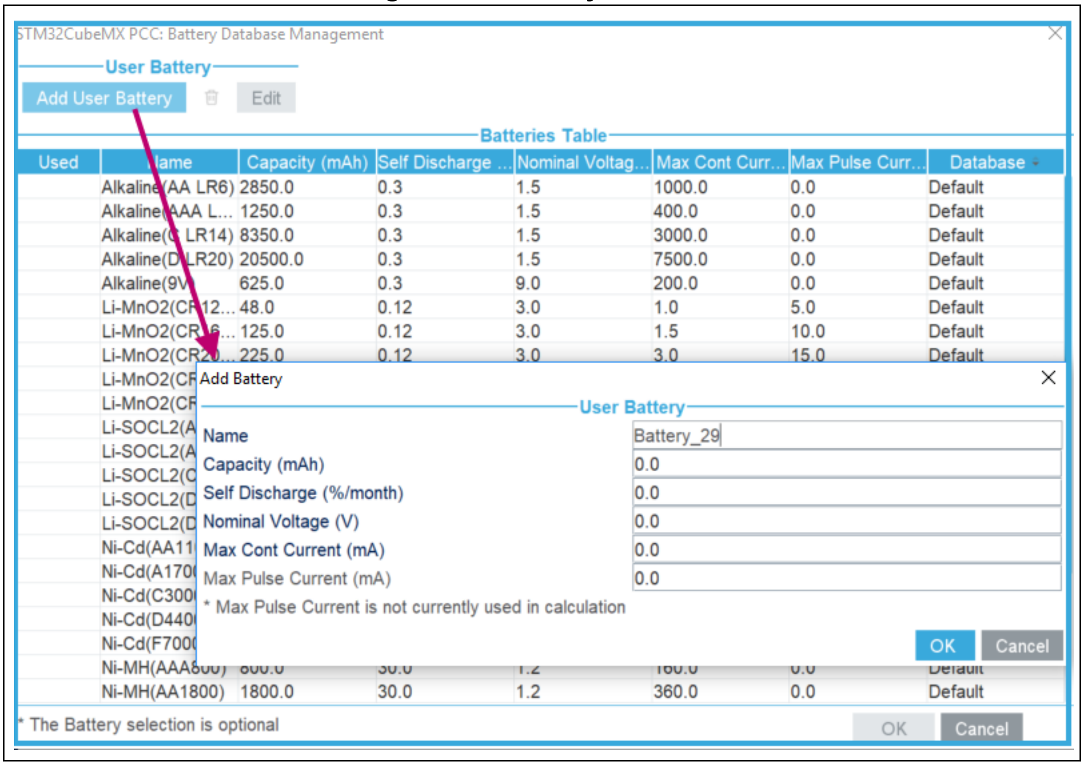
From this view and when multiple choices are available, the user must select a V_{DD} value.

Selecting a battery model (optional)

Optionally, the user can select a battery model. This can also be done once the power consumption sequence is configured.

The user can select a predefined battery or choose to specify a new battery that best matches its application (see *Figure 322*).

Figure 322. Battery selection



Power sequence default view

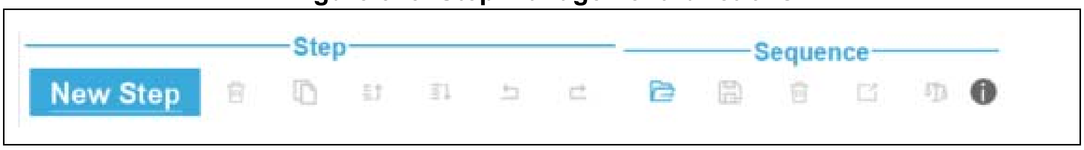
The user can now proceed and build a power sequence.

Managing sequence steps

Steps can be reorganized within a sequence (**Add** new, **Delete** a step, **Duplicate** a step, move **Up** or **Down** in the sequence) using the set of Step buttons (see *Figure 323*).

The user can undo or redo the last configuration actions by clicking the **Undo** button in the Power Consumption Calculator view or the Undo icon from the main toolbar

Figure 323. Step management functions

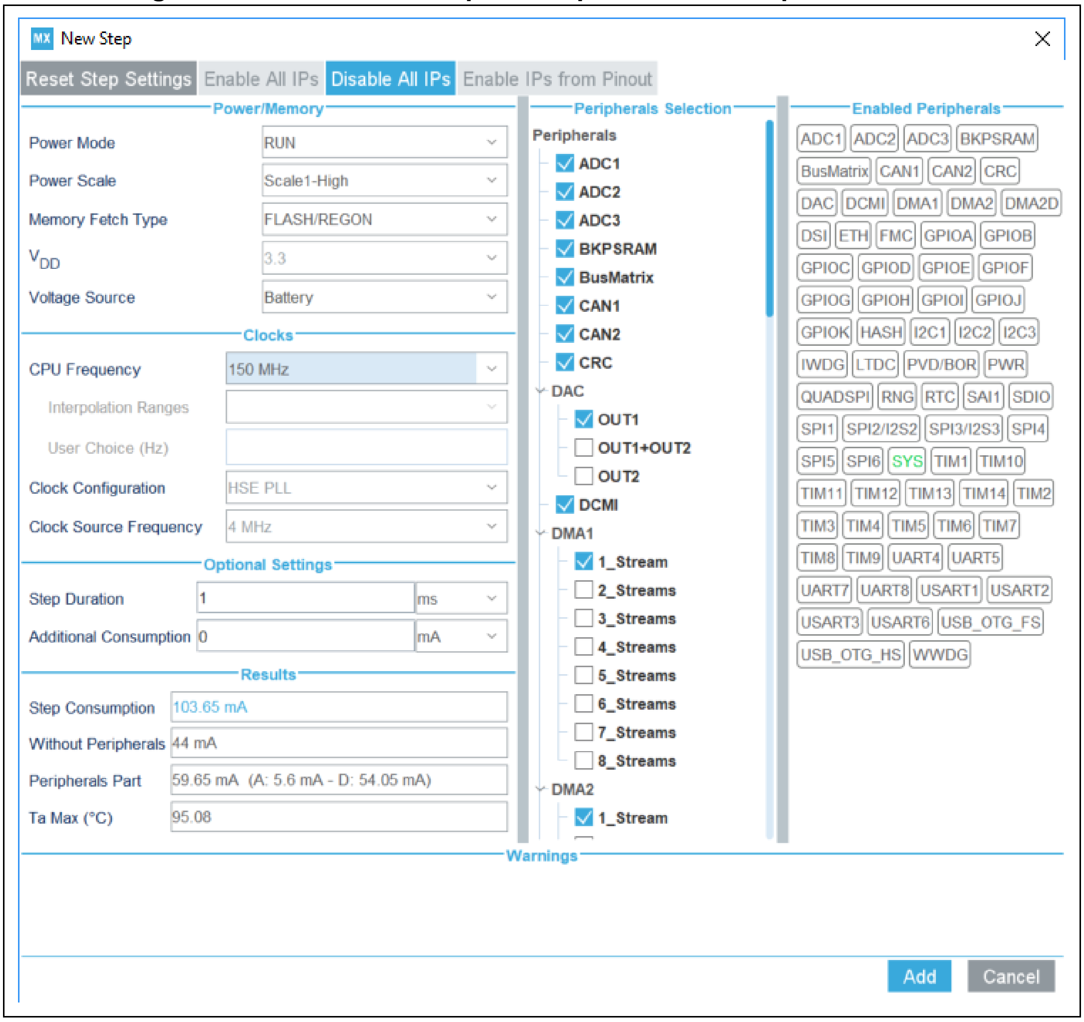


Adding a step

There are two ways to add a new step:

- Click **Add** in the Power Consumption panel. The **New Step** window opens with empty step settings.
- Or, select a step from the sequence table and click **Duplicate**. A **New Step** window opens duplicating the step settings (see *Figure 324*).

Figure 324. Power consumption sequence: New Step default view



Once a step is configured, resulting current consumption and T_{AMAX} values are provided in the window.

Editing a step

To edit a step, double-click it in the sequence table, this opens the **Edit Step** window.

Moving a step

By default, a new step is added at the end of a sequence. Click the step in the sequence table to select it and use the **Up** and **Down** buttons to move it elsewhere in the sequence.

Deleting a step

Select the step to be deleted and click the **Delete** button.

Using the transition checker

Not all transitions between power modes are possible. The Power Consumption Calculator power menu proposes a transition checker to detect invalid transitions or restrict the sequence configuration to only valid transitions.

Enabling the transition checker option prior to sequence configuration ensures that the user will be able to select only valid transition steps.

Enabling the transition checker option on an already configured sequence will highlight the sequence with a green frame if all transitions are valid (see *Figure 325*), or in fuchsia if at least one transition is invalid (fuchsia frame with description of invalid step highlighted in fuchsia, see *Figure 326*). In the latter case, the user can click the **Show log** button to find out how to solve the transition issue (see *Figure 327*).

Figure 325. Enabling the transition checker option on an already configured sequence - All transitions valid

Sequence Table

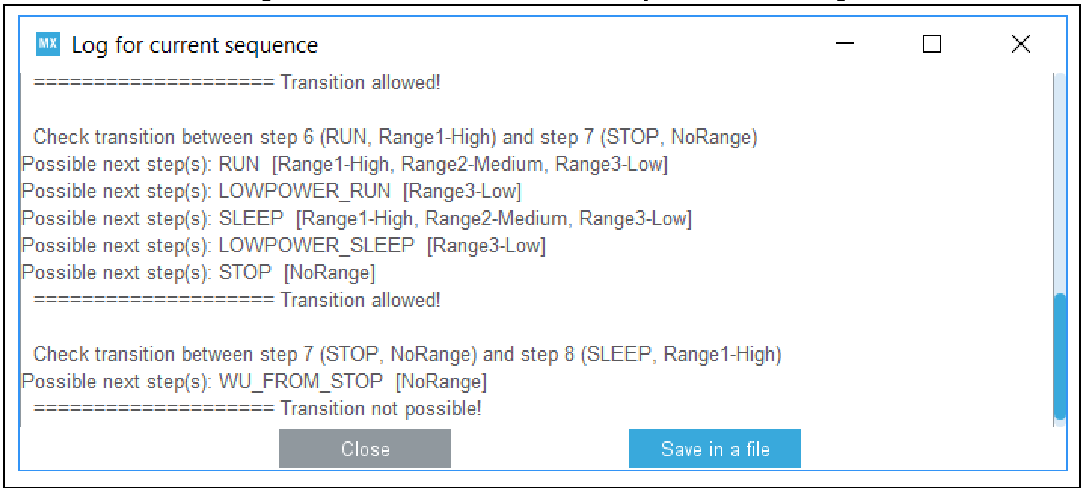
Step	Mode	Vdd	Range/Scale	Memory	CPU/Bus Freq	Clock Config	Peripherals	Step Current	Duration
1	RUN	3.0	Range3-Low	FLASH	1000000 Hz	MSI		166.9 µA	1 ms
2	RUN	3.0	Range2-Medi...	FLASH	8 MHz	HSEBYP		1.3 mA	1 ms
3	RUN	3.0	Range1-High	FLASH	8 MHz	HSEBYP	COMP1 COM...	1.55 mA	1 ms
4	SLEEP	3.0	Range1-High	FLASH	8 MHz	HSEBYP		380 µA	1 ms
5	RUN	3.0	Range3-Low	FLASH	4.2 MHz	MSI	COMP1 COM...	623.66 µA	1 ms
6	RUN	3.0	Range1-High	FLASH	8 MHz	HSEBYP		1.55 mA	1 ms
7	STOP	3.0	NoRange	n/a	0 Hz	ALL CLOCKS...		410 nA	1 ms

Figure 326. Enabling the transition checker option on an already configured sequence - At least one transition invalid

Sequence Table

Step	Mode	Vdd	Range/Scale	Memory	CPU/Bus Freq	Clock Config	Peripherals	Step Current	Duration
1	RUN	3.0	Range3-Low	FLASH	1000000 Hz	MSI		166.9 µA	1 ms
2	RUN	3.0	Range2-Medi...	FLASH	8 MHz	HSEBYP		1.3 mA	1 ms
3	RUN	3.0	Range1-High	FLASH	8 MHz	HSEBYP	COMP1 COM...	1.55 mA	1 ms
4	SLEEP	3.0	Range1-High	FLASH	8 MHz	HSEBYP		380 µA	1 ms
5	RUN	3.0	Range3-Low	FLASH	4.2 MHz	MSI	COMP1 COM...	623.66 µA	1 ms
6	RUN	3.0	Range1-High	FLASH	8 MHz	HSEBYP		1.55 mA	1 ms
7	STOP	3.0	NoRange	n/a	0 Hz	ALL CLOCKS...		410 nA	1 ms
8	SLEEP	3.0	Range1-High	FLASH	8 MHz	HSEBYP		380 µA	1 ms

Figure 327. Transition checker option - Show log



5.3.2 Configuring a step in the power sequence

The step configuration is performed from the **Edit Step** and **New Step** windows. The graphical interface guides the user by forcing a predefined order for setting parameters.

Their naming may differ according to the selected MCU series. For details on each parameter, refer to glossary in *Section 5.3.4* and to *Appendix C*, or to the electrical characteristics section of the datasheet.

The parameters are set automatically by the tool when there is only one possible value (in this case, the parameter cannot be modified and is grayed out). The tool proposes only the configuration choices relevant to the selected MCU.

To configure a new step:

1. Click **Add** or **Duplicate** to open the **New step** window or double-click a step from the sequence table to open the **Edit step** window.
2. Within the open step window, select in the following order:
   - The **Power Mode**

     Changing the Power Mode resets the whole step configuration.
   - The **Peripherals**

     Peripherals can be selected/deselected at any time after the Power Mode is configured.
   - The **Power scale**

     The power scale corresponds to the power consumption range (STM32L1) or the power scale (STM32F4).

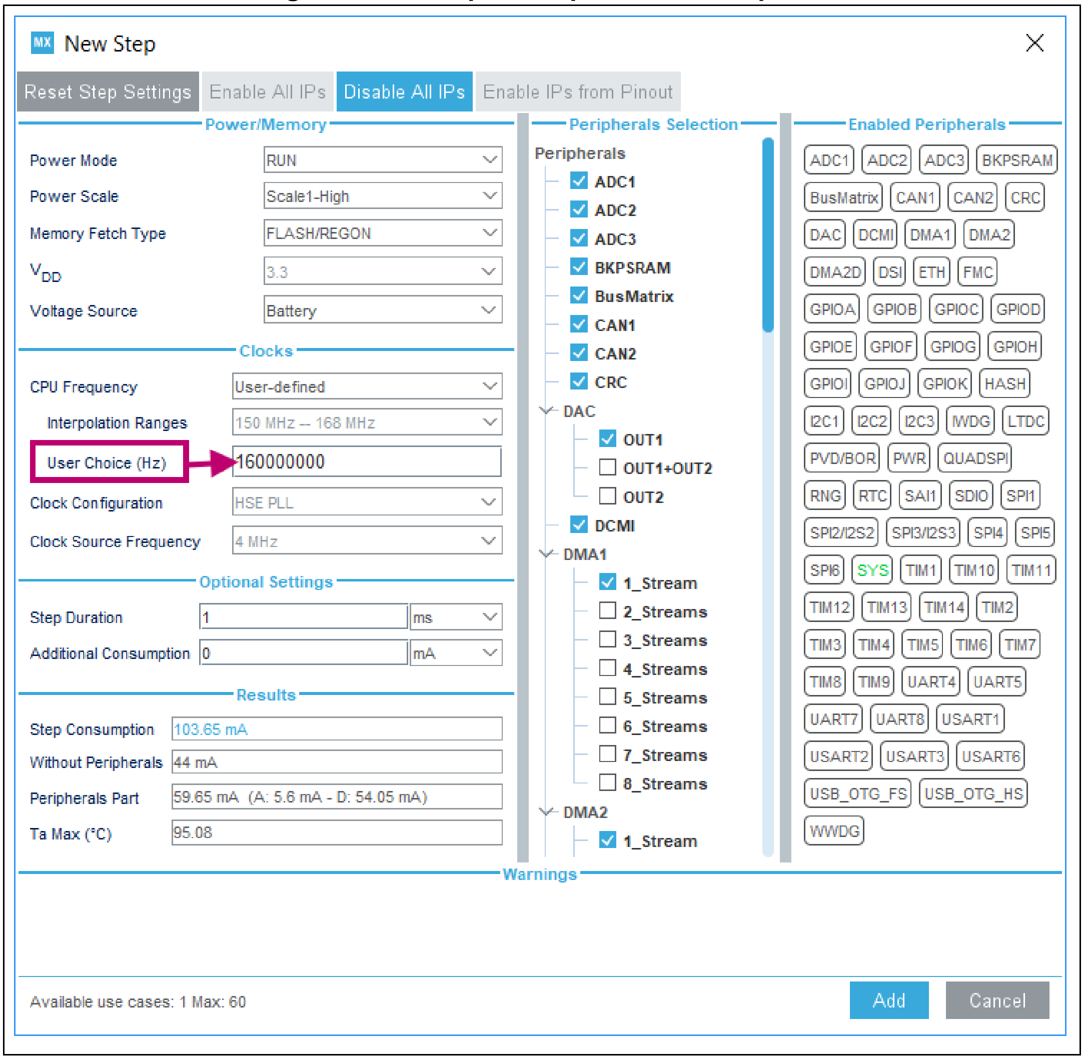
     Changing the Power Mode or the Power Consumption Range discards all subsequent configurations.
   - The **Memory Fetch Type**
   - The V_{DD} value if multiple choices available
   - The voltage source (battery or VBUS)
   - A **Clock Configuration**

     Changing the Clock Configuration resets the frequency choices further down.
   - When multiple choices are available, the **CPU Frequency** (STM32F4) and the **AHB Bus Frequency/CPU Frequency**(STM32L1) or, for active modes, a user specified frequency. In this case, the consumption value will be interpolated (see *Using interpolation*).
3. Optionally set
   - A **step duration** (1 ms is the default value)
   - An **additional consumption** value (expressed in mA) to reflect, for example, external components used by the application (external regulator, external pull-up, LEDs or other displays). This value added to the microcontroller power consumption will impact the step overall power consumption.
4. Once the configuration is complete, the **Add** button becomes active. Click it to create the step and add it to the sequence table.

Using interpolation

For steps configured for active modes (Run, Sleep), frequency interpolation is supported by selecting CPU frequency as User Defined and entering a frequency in Hz (see *Figure 328*).

Figure 328. Interpolated power consumption

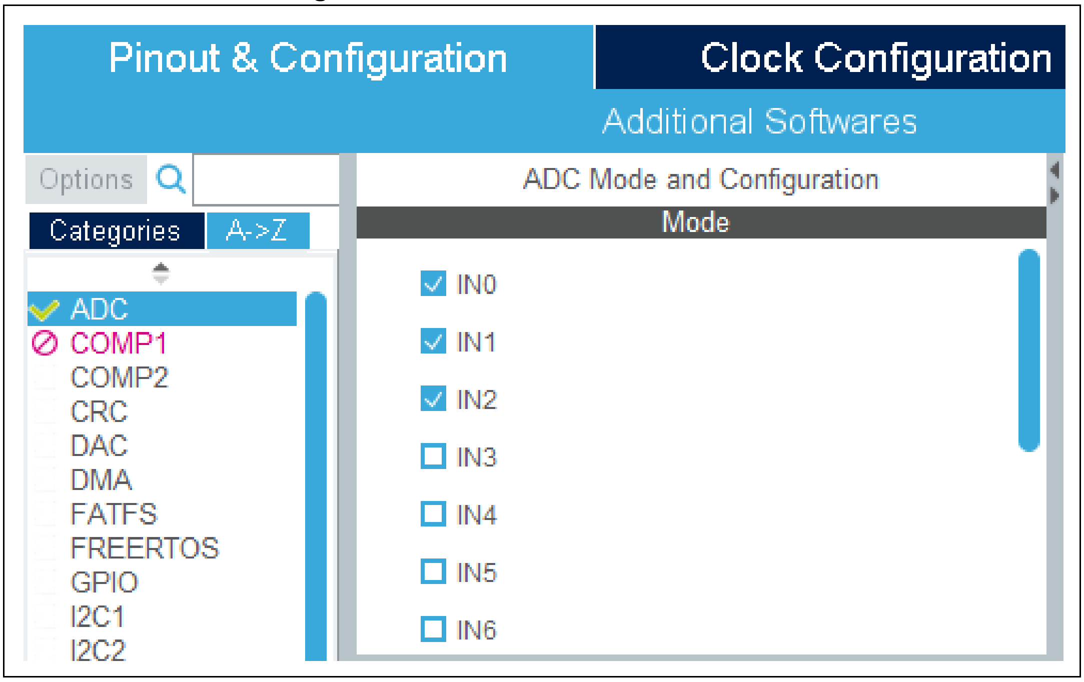


Importing pinout

Figure 329 illustrates the example of the ADC configuration in the **Pinout** view: clicking **Enable IPs from Pinout** in the Power Consumption Calculator view selects the ADC peripheral and GPIO A (*Figure 330*).

The **Enable IPs from Pinout** button allows the user to automatically select the peripherals that have been configured in the **Pinout** view.

Figure 329. ADC selected in Pinout view

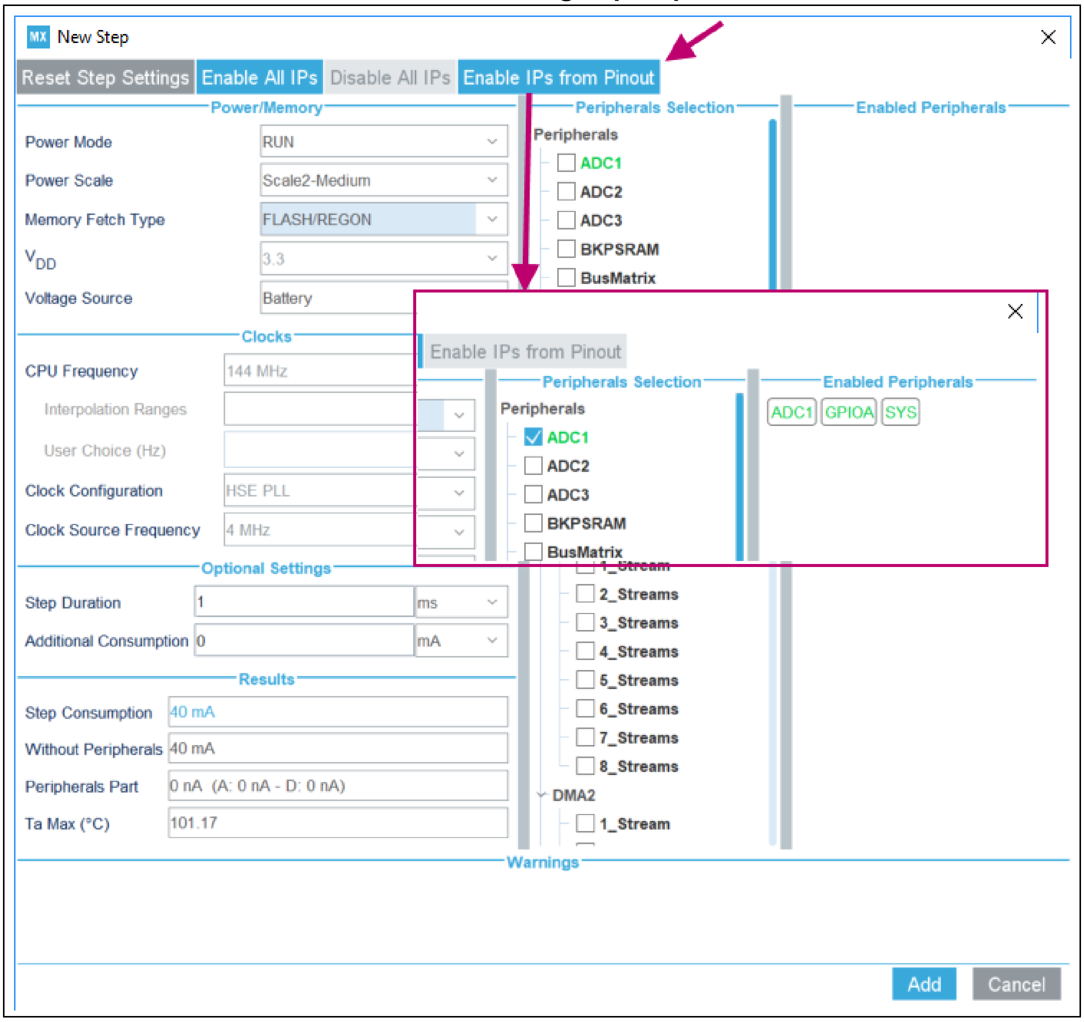


Selecting/deselecting all peripherals

Clicking **Enable All IPs** allows the user to select all peripherals at once.

Clicking **Disable All IPs** removes them as contributors to the consumption.

Figure 330. Power Consumption Calculator configuration window: ADC enabled using import pinout

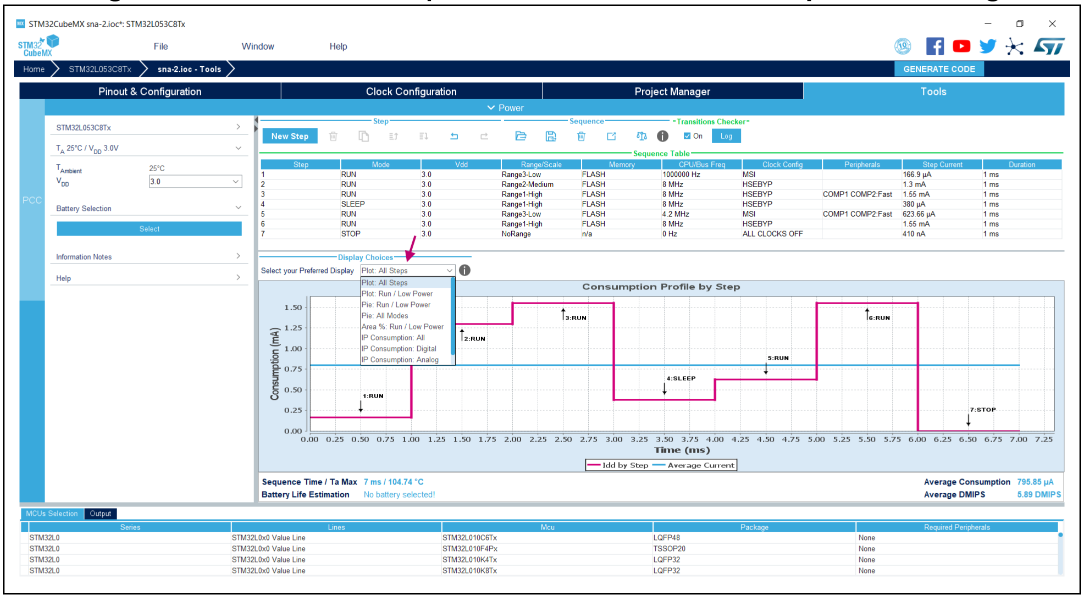


5.3.3 Managing user-defined power sequence and reviewing results

The configuration of a power sequence leads to an update of the Power Consumption Calculator view (see *Figure 331*):

- The sequence table shows all steps and step parameters values. A category column indicates whether the consumption values are taken from the datasheet or are interpolated.
- The sequence chart area shows different views of the power sequence according to a display type (e.g. plot all steps, plot low power versus run modes)
- The results summary provides the total sequence time, the maximum ambient temperature (T_{AMAX}), plus an estimate of the average power consumption, DMIPS, and battery lifetime provided a valid battery configuration has been selected.

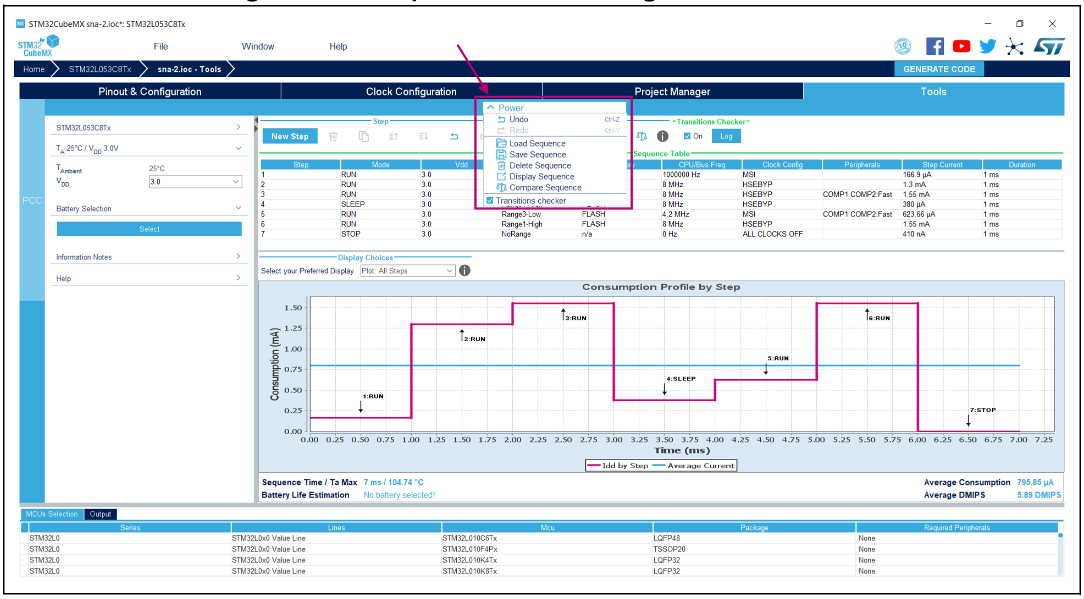
Figure 331. Power Consumption Calculator view after sequence building



Managing the whole sequence (load, save and compare)

From the power menu (see *Figure 332*), the current sequence can be saved, deleted or compared to a previously saved sequence that will be displayed in a dedicated popup window.

Figure 332. Sequence table management functions

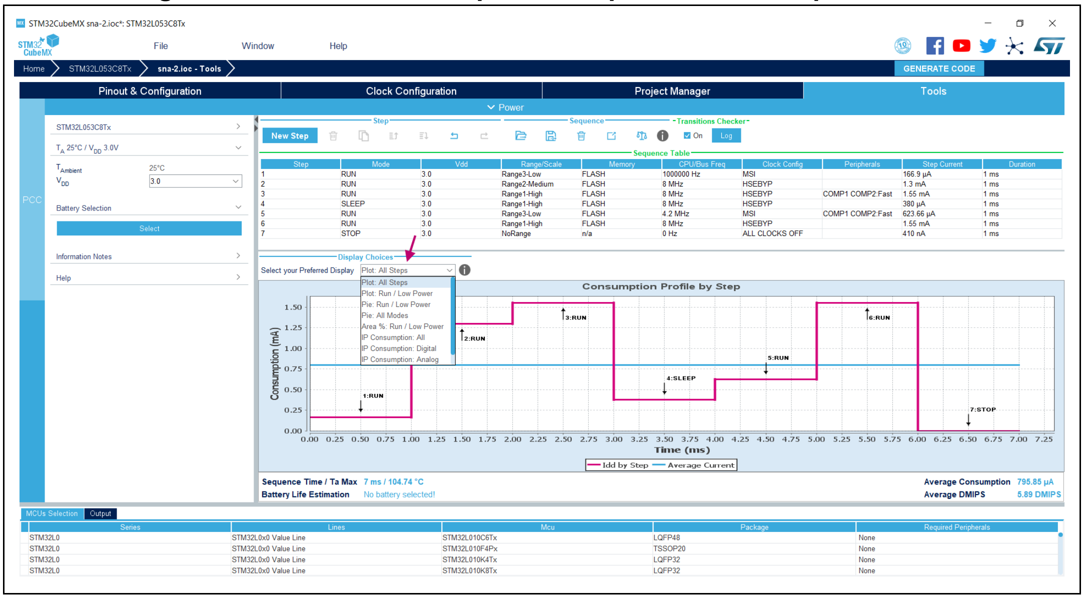


Managing the results charts and display options

In the Display area, select the type of chart to display (e.g. sequence steps, pie charts, consumption per peripherals). You can also click **External Display** to open the charts in dedicated windows (see *Figure 333*).

Right-click on the chart to access the contextual menus: **Properties**, **Copy**, **Save** as png picture file, **Print**, **Zoom** menus, and **Auto Range** to reset to the original view before zoom operations. **Zooming** can also be achieved by mouse selecting from left to right a zone in the chart and **Zoom reset** by clicking the chart and dragging the mouse to the left.

Figure 333. Power Consumption: Peripherals consumption chart

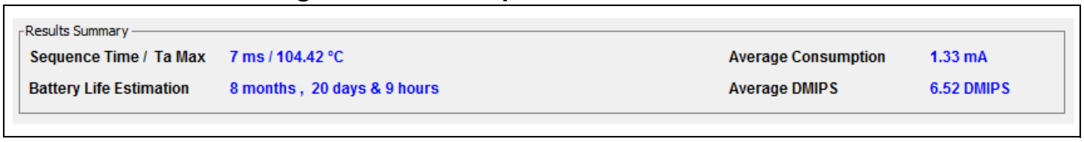


Overview of the Results summary area

This area provides the following information (see *Figure 334*):

- Total sequence time, as the sum of the sequence steps durations.
- Average consumption, as the sum of each step consumption weighed by the step duration.
- The average DMIPS (Dhrystone million instructions per second) based on Dhrystone benchmark, highlighting the CPU performance for the defined sequence.
- Battery life estimation for the selected battery model, based on the average power consumption and the battery self-discharge.
- T_{AMAX}: highest maximum ambient temperature value found during the sequence.

Figure 334. Description of the Results area



5.3.4 Power sequence step parameters glossary

The parameters that characterize power sequence steps are the following (refer to *Appendix C: STM32 microcontrollers power consumption parameters* for more details):

- Power modes

  To save energy, it is recommended to switch the microcontroller operating mode from running mode, where a maximum power is required, to a low-power mode requiring limited resources.
- V_{CORE} range (STM32L1) or Power scale (STM32F4)

  These parameters are set by software to control the power supply range for digital peripherals.
- Memory Fetch Type

  This field proposes the possible memory locations for application C code execution. It can be either RAM, FLASH or FLASH with ART ON or OFF (only for families that feature a proprietary Adaptive real-time (ART) memory accelerator which increases the program execution speed when executing from flash memory).

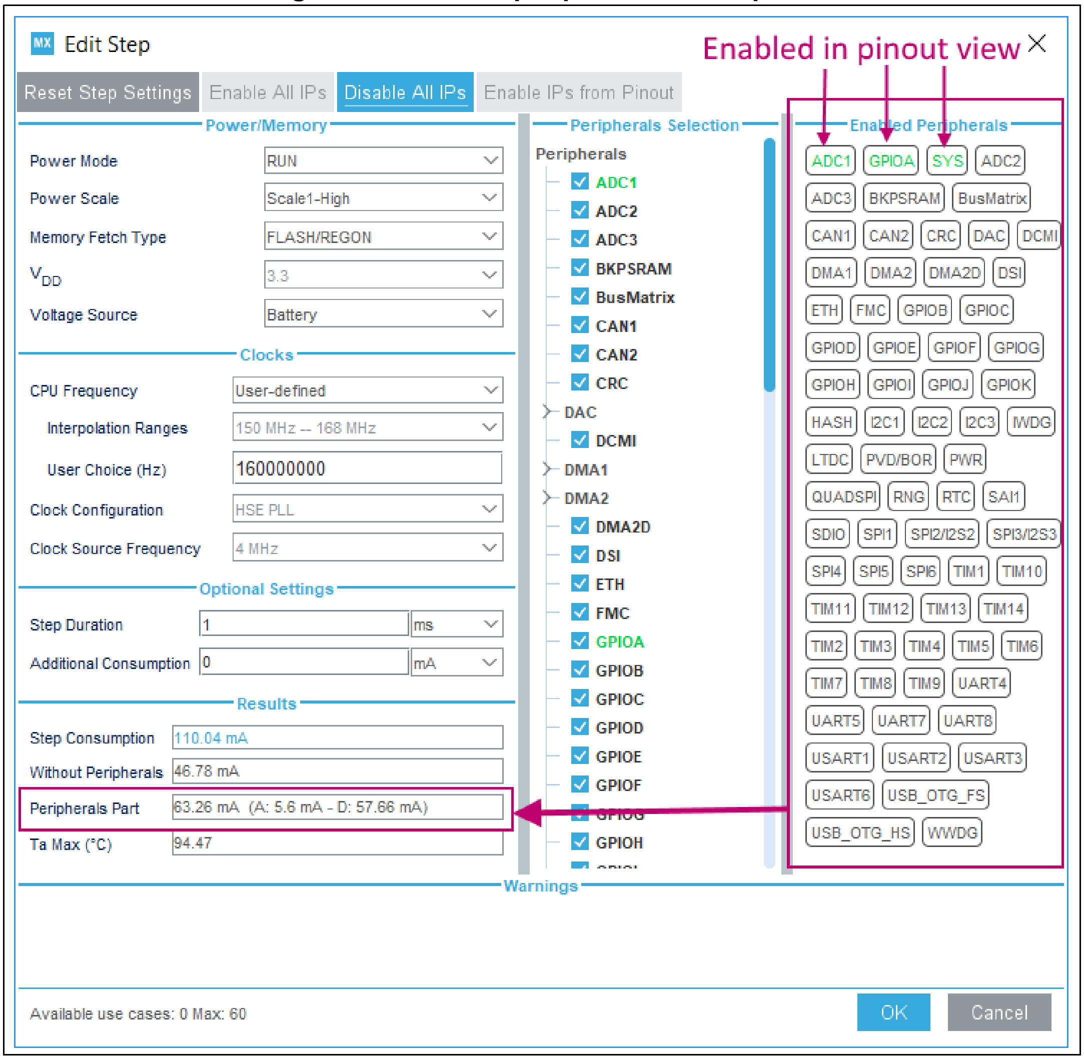
  The performance achieved thanks to the ART accelerator is equivalent to 0 wait state program execution from flash memory. In terms of power consumption, it is equivalent to program execution from RAM. In addition, STM32CubeMX uses the same selection choice to cover both settings, RAM and flash memory with ART ON.
- Clock Configuration

  This operation sets the AHB bus frequency or the CPU frequency that will be used for computing the microcontroller power consumption. When there is only one possible choice, the frequencies are automatically configured.

  The clock configuration drop-down list allows to configure the application clocks:
  - the internal or external oscillator sources: MSI, HSI, LSI, HSE or LSE
  - the oscillator frequency
  - other determining parameters, among them PLL ON, LSE Bypass, AHB prescaler value, LCD with duty
- Peripherals

  The peripheral list shows the peripherals available for the selected power mode. The power consumption is given assuming that peripherals are only clocked (e.g. not in use by a running program). Each peripheral can be enabled or disabled, their individual power consumption is displayed in a tooltip. An overall consumption due to peripheral analog and digital parts is provided in the step Results area (see *Figure 335*).

Figure 335. Overall peripheral consumption



The user can select the peripherals relevant for the application:

– none (**Disable All)**

– some (using peripheral dedicated checkbox)

– all (**Activate All**)

– or all from the previously defined pinout configuration (**Import Pinout**).

Only the selected and enabled peripherals are taken into account when computing the power consumption.

- Step duration

  The user can change the default step duration value. When building a sequence, the user can either create steps according to the application actual power sequence or define them as a percentage spent in each mode. For example, if an application spends 30% in Run mode, 20% in Sleep and 50% in Stop, the user must configure a 3-step sequence consisting in 30 ms in Run, 20 ms in Sleep and 50 ms in Stop.

- Additional consumption

  This field allows entering an additional consumption resulting from specific user configuration (e.g. MCU providing power supply to other connected devices).

5.3.5 Battery glossary

- Capacity (mAh)

  Amount of energy that can be delivered in a single battery discharge.
- Self-discharge (% / month)

  This percentage, over a specified period, represents the loss of battery capacity when the battery is not used (open-circuit conditions), as a result of internal leakage.
- Nominal voltage (V)

  Voltage supplied by a fully charged battery.
- Max. continuous current (mA)

  This current corresponds to the maximum current that can be delivered during the battery lifetime period without damaging the battery.
- Max. pulse current (mA)

  This is the maximum pulse current that can be delivered exceptionally, for instance when the application is switched on during the starting phase.

5.3.6 SMPS feature

Some microcontrollers (e.g. STM32L496xxxxP) allow the user to connect an external switched mode power supply (SMPS) to further reduce power consumption.

For such microcontrollers, the Power Consumption Calculator tool offers the following features:

- Selection of SMPS for the current project

  From the left panel, check the **Use SMPS** box to use SMPS (see *Figure 336*). By default, ST SMPS model is used.
- Selection of another SMPS model by clicking the **Change** button

  This opens the SMPS database management window in which the user can add a new SMPS model (see *Figure 337*). The user can then select a different SMPS model for the current sequence (see *Figure 338*, *Figure 339* and *Figure 340*)
- Check for invalid SMPS transitions in the current sequence by enabling the SMPS checker

  To do this, select the checkbox to enable the checker and click the **Help** button to open the reference state diagram (see *Figure 341*).
- Configuration of SMPS mode for each step (see *Figure 342*)

  If the SMPS checker is enabled, only the SMPS modes valid for the current step are proposed.

Figure 336. Selecting SMPS for the current project

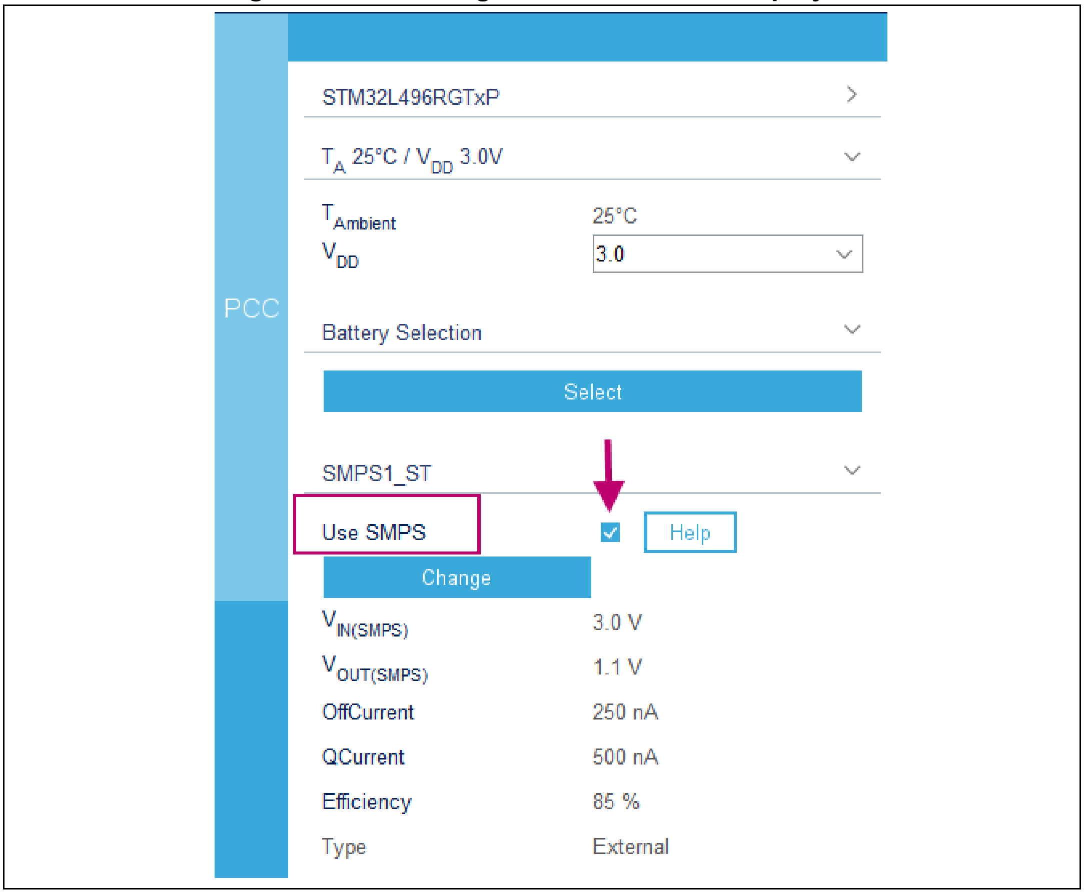


Figure 337. SMPS database - Adding new SMPS models

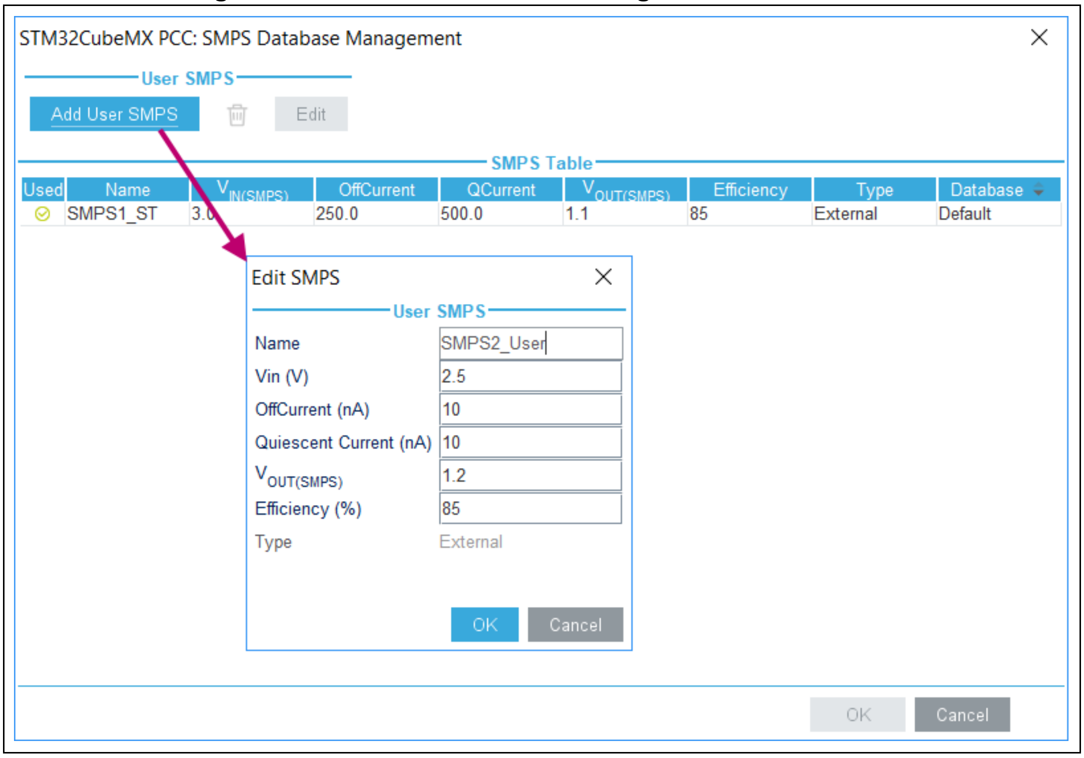


Figure 338. SMPS database - Selecting a different SMPS model

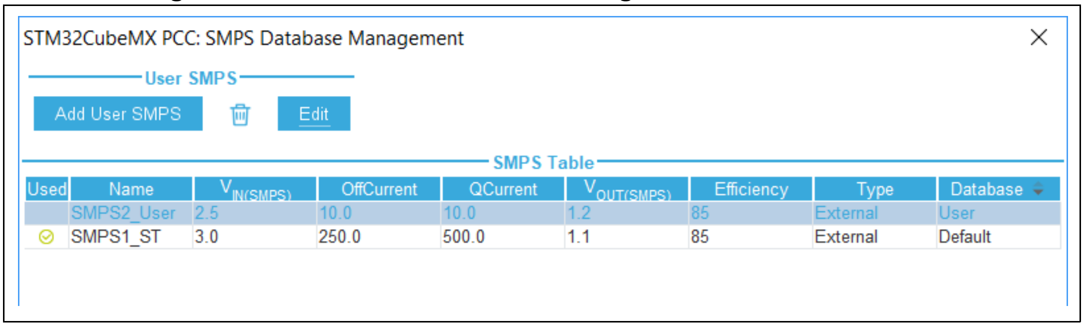


Figure 339. Current project configuration updated with new SMPS model

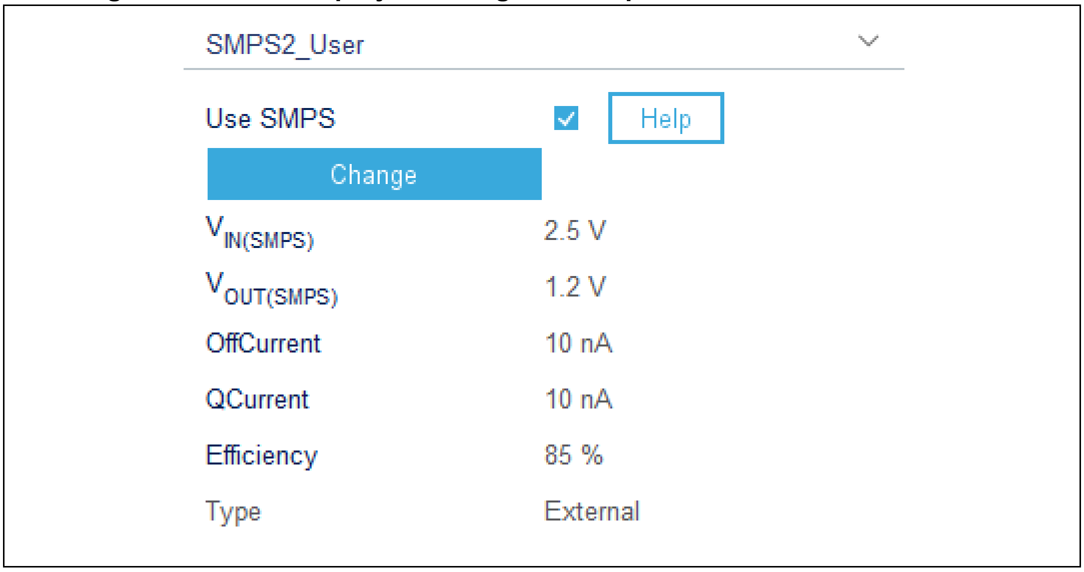


Figure 340. SMPS database management window with new model selected

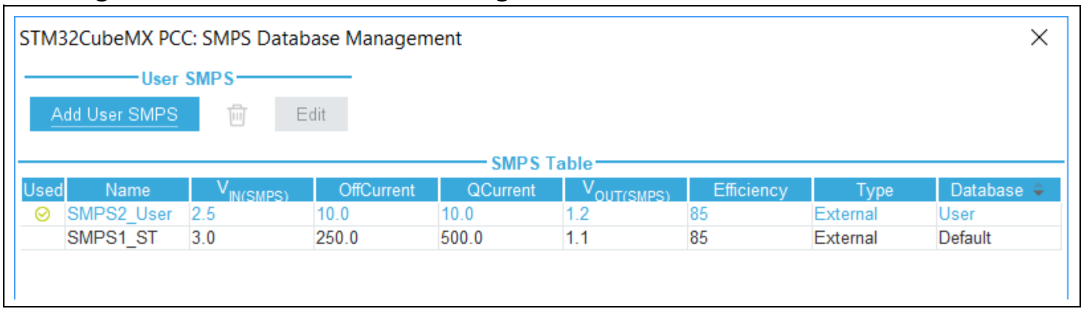


Figure 341. SMPS transition checker and state diagram helper window

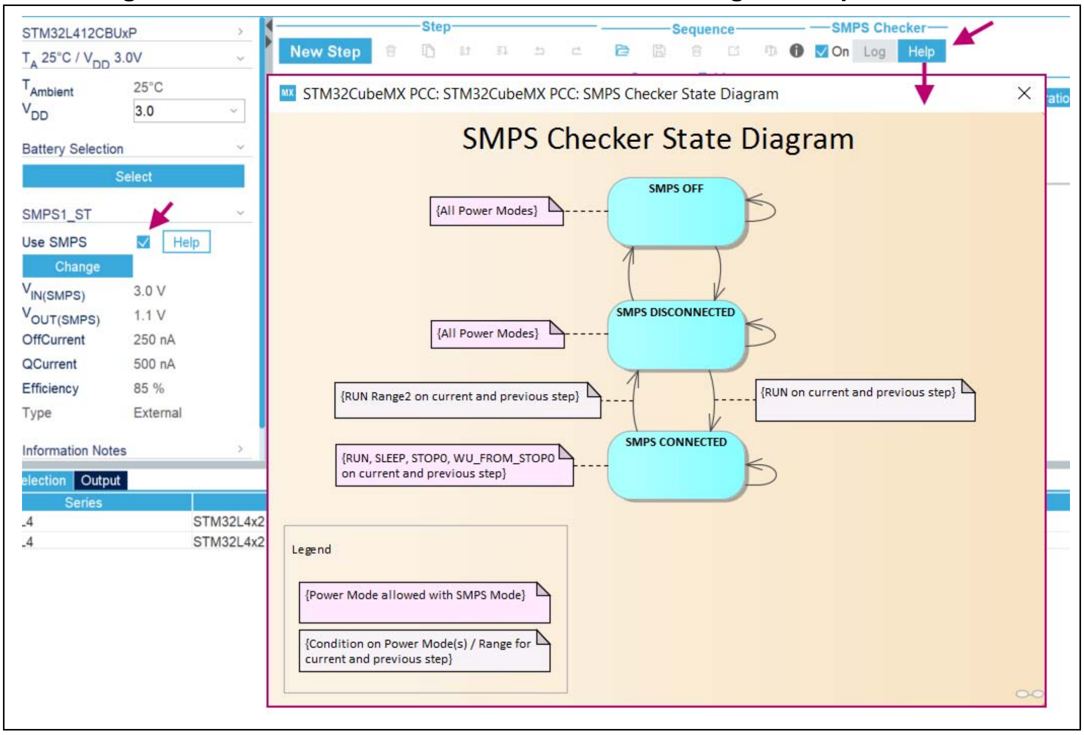
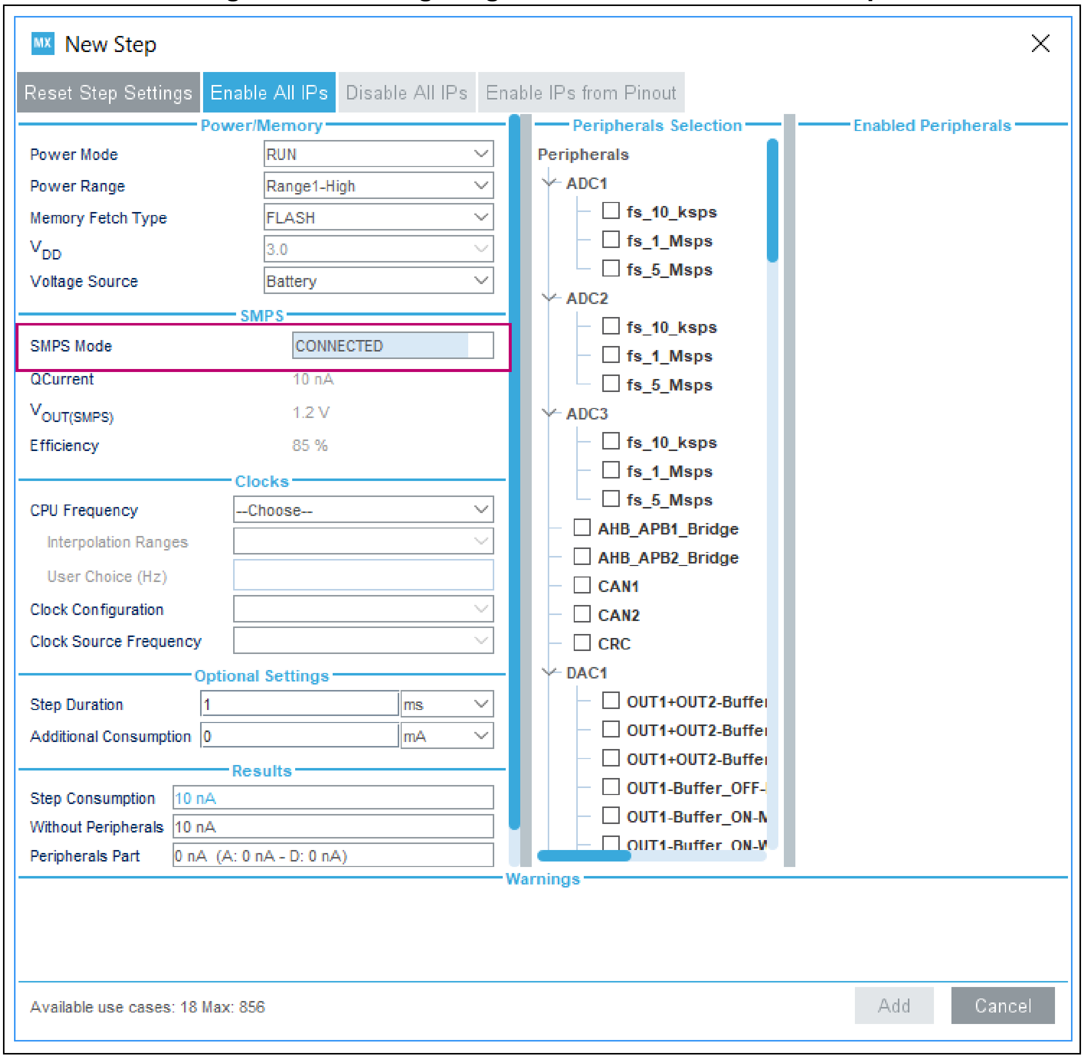


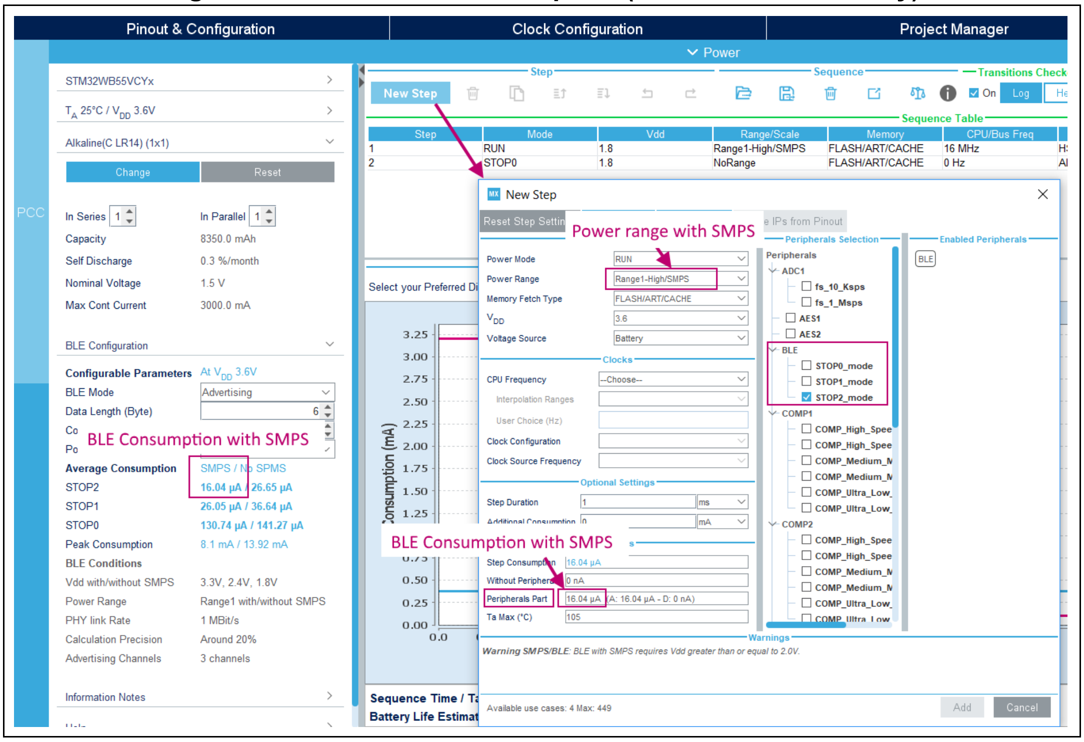
Figure 342. Configuring the SMPS mode for each step



5.3.7 Bluetooth Low-Energy®/ZigBee® support (STM32WB series only)

The Power Consumption tool allows the user to take into account the consumption related to the RF peripheral and corresponding Bluetooth Low-Energy functional mode, combined with the usage of the SMPS feature.

Figure 343. RF related consumption (STM32WB series only)



The Bluetooth Low-Energy mode can be selected from the left panel and configured to reflect the application relevant settings. For each new step enabling BLE, the peripheral consumption part is updated accordingly (see *Figure 344*). A similar approach is used for ZigBee (see *Figure 345*).

Figure 344. RF Bluetooth Low-Energy mode configuration (STM32WB series only)

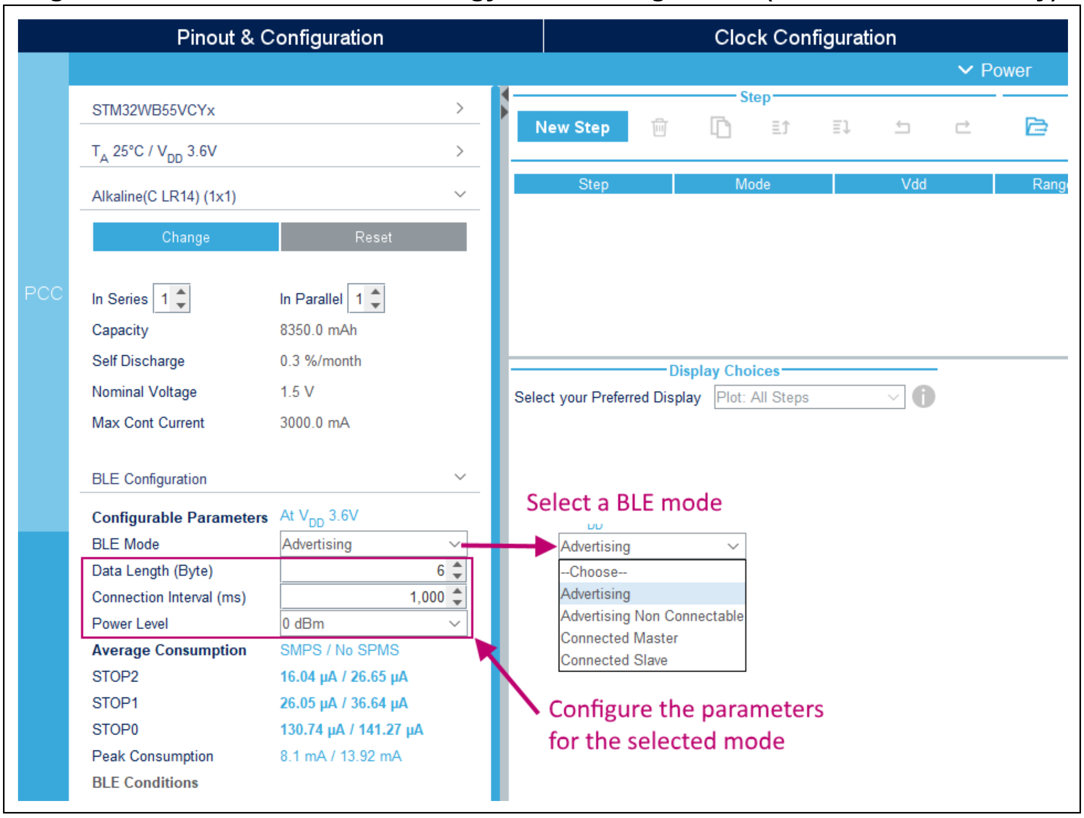
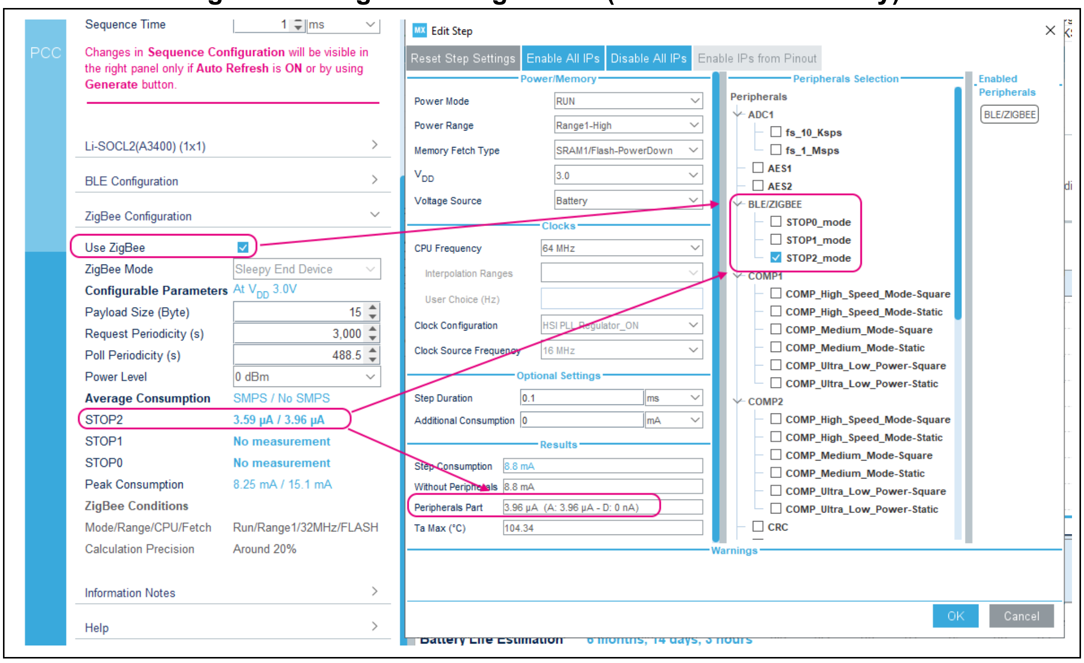


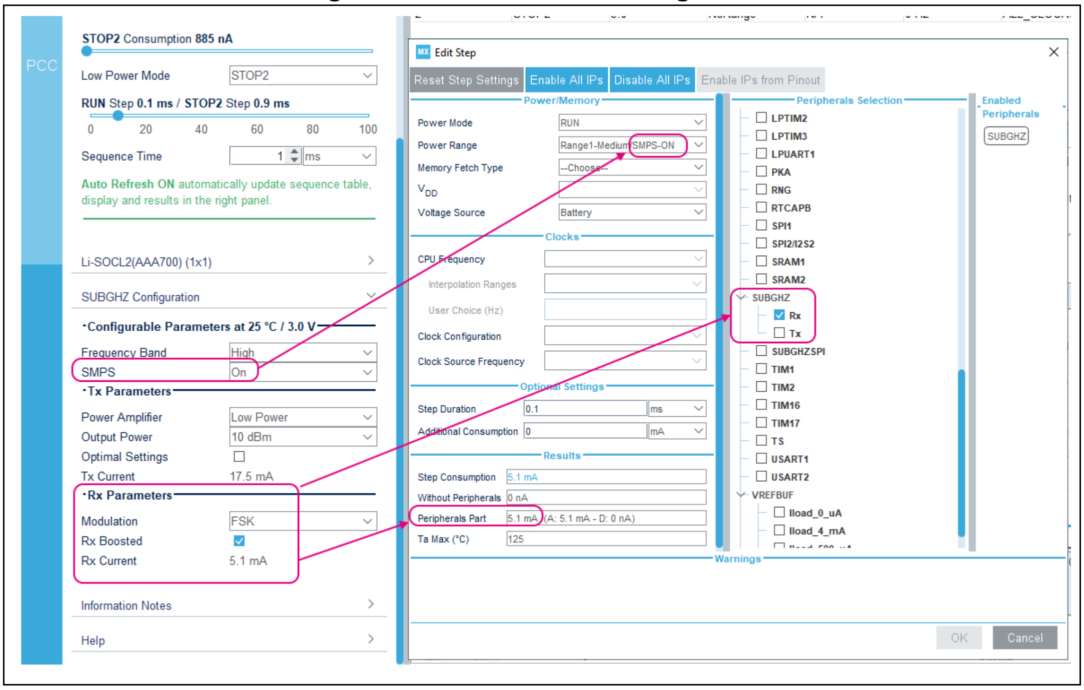
Figure 345. ZigBee configuration (STM32WB series only)



5.3.8 Sub-GHz support (STM32WL series only)

Sub-GHz usage can be enabled from the left panel and configured to reflect the application relevant settings. For each new step enabling ZigBee, the peripheral consumption part is updated accordingly (see *Figure 346*).

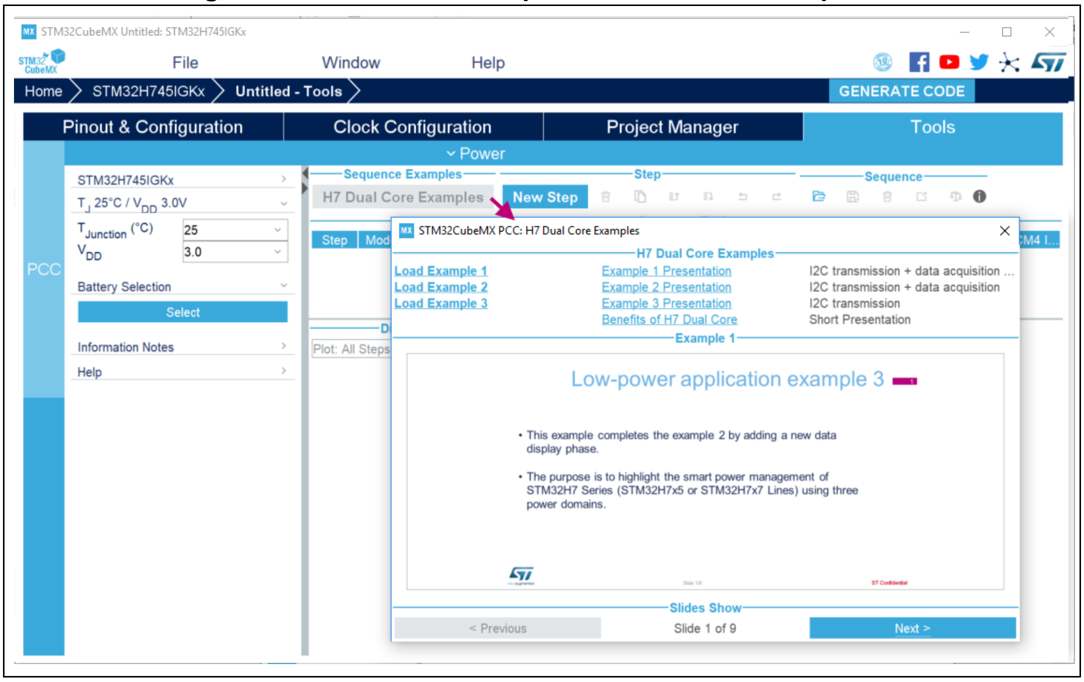
Figure 346. RF sub-GHz configuration



5.3.9 Example feature (STM32MPUs and STM32H7 dual-core only)

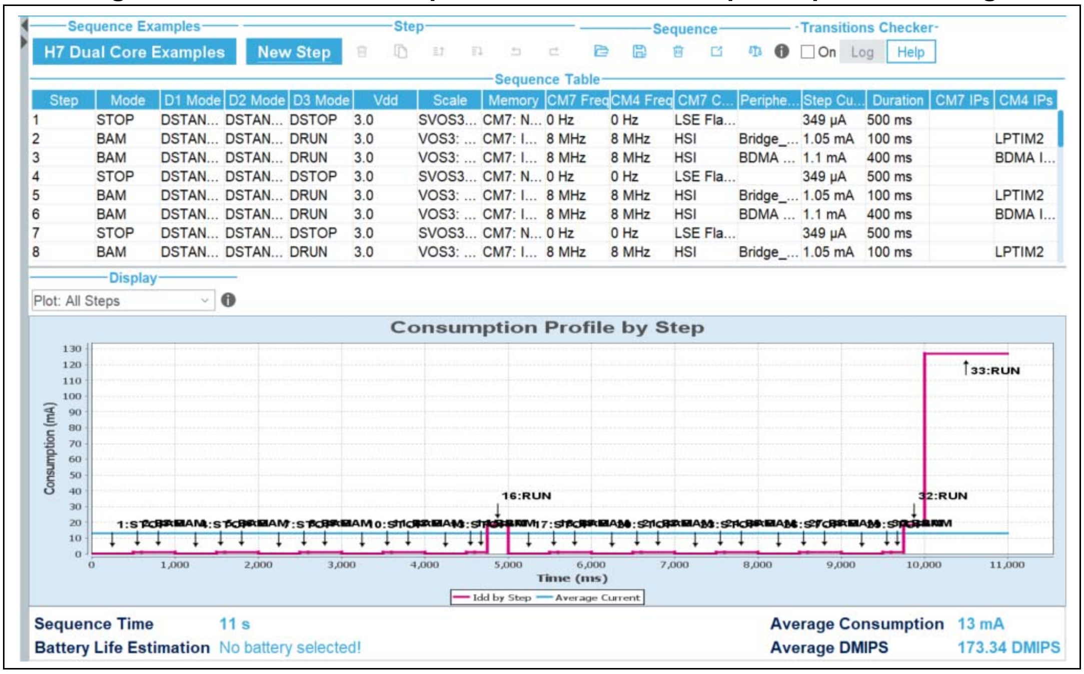
Under the section Sequence Examples, the PCC tool allows to access examples: each of them comes with an explanatory slide set and a ready-made sequence to load in PCC (see *Figure 347*).

Figure 347. Power Consumption Calculator - Example set



Clicking “Load Example N” loads the sequence corresponding to example N (see *Figure 348*).

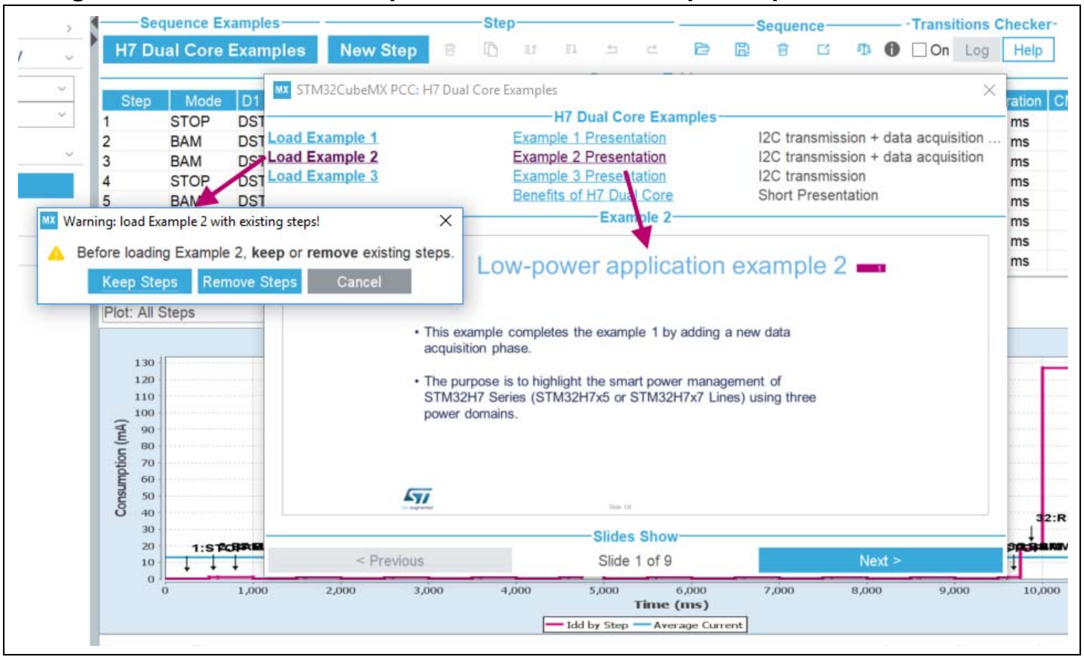
Figure 348. Power Consumption Calculator - Example sequence loading



Clicking “Example N Presentation” displays the explanations for that example.

The example can be changed anytime: the new sequence can be added to the current sequence, or replace it (see *Figure 349*).

Figure 349. Power Consumption Calculator - Example sequence new selection



Note: The examples are provided for a given part number and may require adjustments when used for a different part number. Also, after loading, it is recommended to edit each step and check settings.

5.4 DDR Suite (STM32MPUs only)

DDR SDRAMs are complex high speed devices that need careful PCB design.

The STM32MP15 devices support the following DDR types:

- LPDDR2
- LPDDR3
- DDR3 / DDR3L

They are specified by the JEDEC standard (standardization of interfaces, commands, timings, packages and ballout).

STM32CubeMX has been extended to provide an exhaustive tool suite for the DDR subsystem. It proposes the following key features.

- **Configuration of DDR** controller and PHY registers is managed automatically based on reduced set of editable parameters.
- **DDR testing** is offered based on a rich list. Tests go from basic to stress. User can also develop its own tests.

DDR configuration is accessible like the other peripherals in the **Pinout & Configuration** view: clicking the DDR from the component panel opens the mode and configuration panels.

DDR Test suite testing and tuning features are available from the **Tools** view.

The DDR suite relies on two important concepts:

- the **DDR timings** as key inputs for the configuration of the DDR Controller and PHY
- the tuning of DDR signals to compensate board design imperfections.

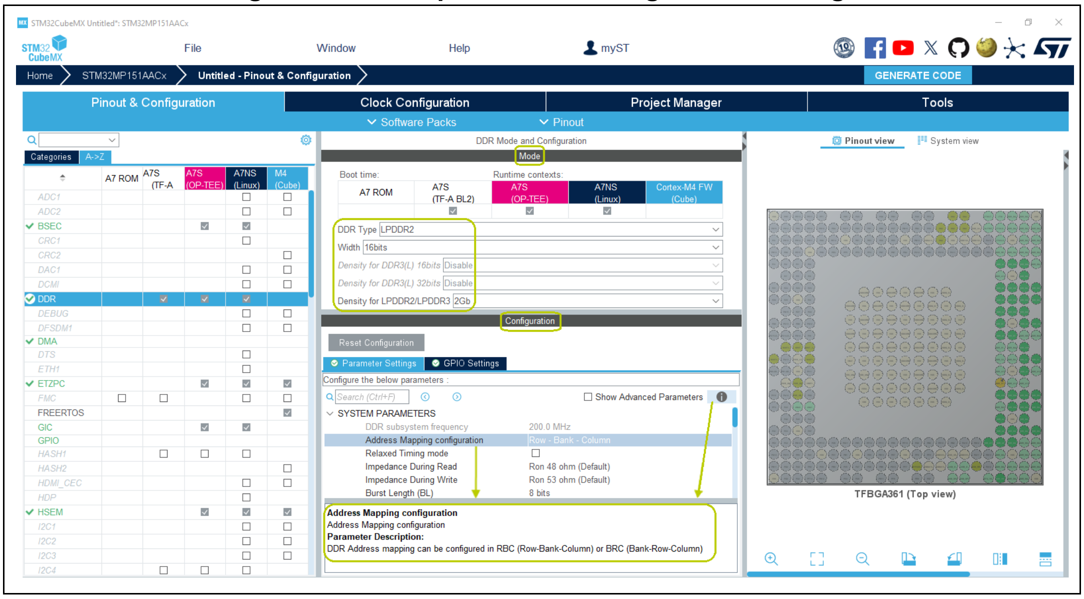
5.4.1 DDR configuration

STM32CubeMX allows to set DDR system parameters and JEDEC core timings. The timing parameters are available in the DDR datasheet.

DDR type, width, and density

The DDR type, width, and density parameters must be set to proceed with the DDR configuration. This can be done in the Mode panel after selecting the DDR in the **Pinout & Configuration** view. See *Figure 350* for an example of LPDDR2 settings.

Figure 350. DDR pinout and configuration settings



Another example: for a configuration with two “DDR3 16 bits 2 Gb” chips, settings are “DDR3/DDR3L”, “32 bits” and 4 Gb”.

Note: Contexts for DDR IP cannot be changed, DDR is tied to “Cortex-A7 nonsecure” identified as “Cortex-A7 NS” in the tool.

DDR configuration

Clicking on a parameter will show additional details in the DDR configuration footer.

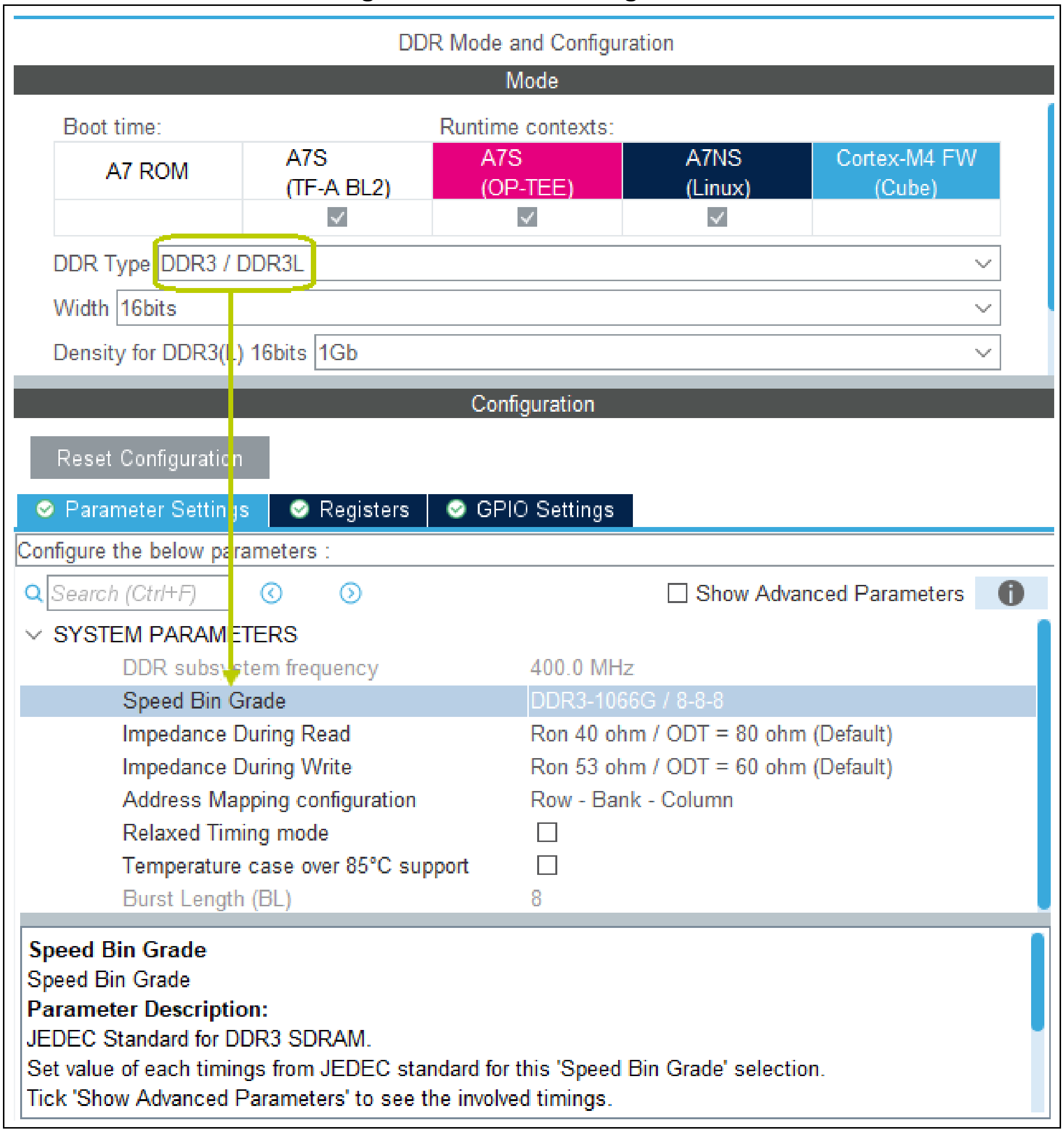
- The DDR frequency is taken from the ‘Clock configuration’ tab, it cannot be changed in the DDR configuration.
- The ‘Relaxed Timing’ mode is used during bring-up phase for trying relaxed key DDR timings value (one t_{CK} added to t_{RC}, t_{RCD} and t_{RP} timings)
- Other parameters must be retrieved from the user DDR datasheet.
- Some parameters are read-only: they are for information only and depend on the DDR type.

Clicking “generate code” automatically computes the DDR node of the device tree (DDR Controller and DDR PHY registers values) based on these parameters.

DDR3 configuration

For DDR3, the configuration is made easier with the selection of a **Speed Bin Grade** combination, instead of manually editing timing parameters.

Figure 351. DDR3 configuration



The Speed Bin Grade combination must match the selected DDR. If the exact combination is not in the pick-list, select “1066E / 6-6-6” for faster DDR Speed Bin Grade, or “1066G / 8-8-8” for a relaxed configuration.

Timing edition is optional, and reserved for advanced users: select Show Advanced parameters to display the list.

5.4.2 Connection to the target and DDR register loading

To manage DDR tests and tuning, STM32CubeMX must establish a connection with the target and more specifically with **U-Boot SPL** using the **DDR interactive protocol**:

- the DDR interactive protocol is only available in the **Basic boot scheme U-Boot SPL** binary and supported over the UART4 peripheral instance
- when U-Boot SPL detects a connection to STM32CubeMX on UART4, it stops its initialization process and accepts commands from STM32CubeMX.

There are two connection options:

1. the U-Boot SPL binary is available in flash memory
2. the U-Boot SPL needs to be loaded in SYSRAM because the DDR has not yet been tested nor tuned (and, consequently, is not fully functional yet).

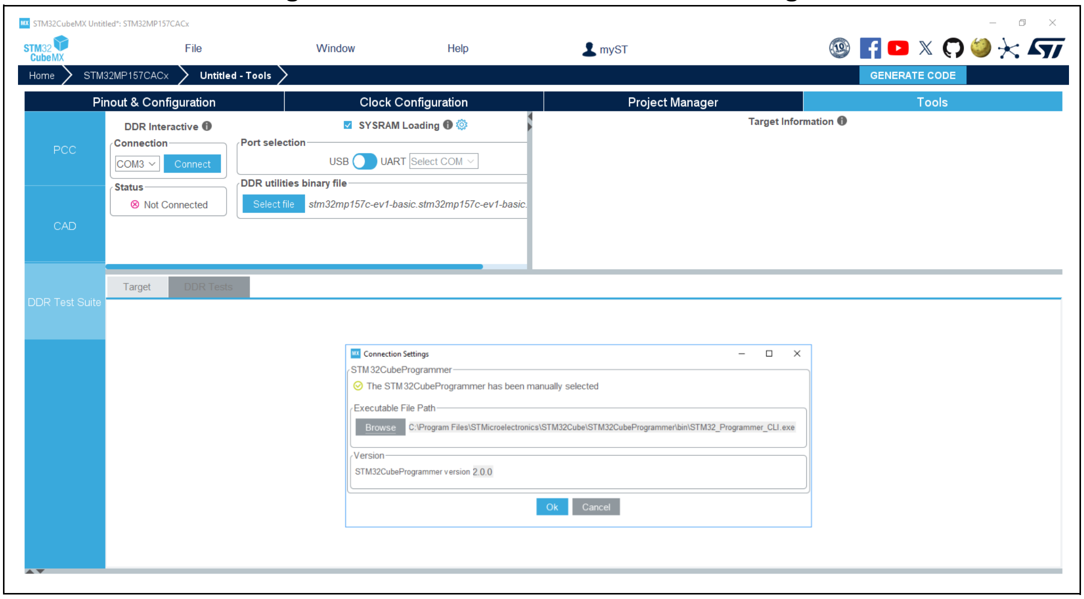
Prerequisites

- Installation of ST-Link USB driver to perform firmware upgrades: for Windows, latest version of STSW-LINK009, for Linux, use STSW-LINK007. Both can be downloaded from *www.st.com*.
- Installation of STM32CubeProgrammer (for SYSRAM loading only): installer can be downloaded from *www.st.com*.

Connection to the target

The COM port must be selected to connect to the target, as indicated in *Figure 352*.

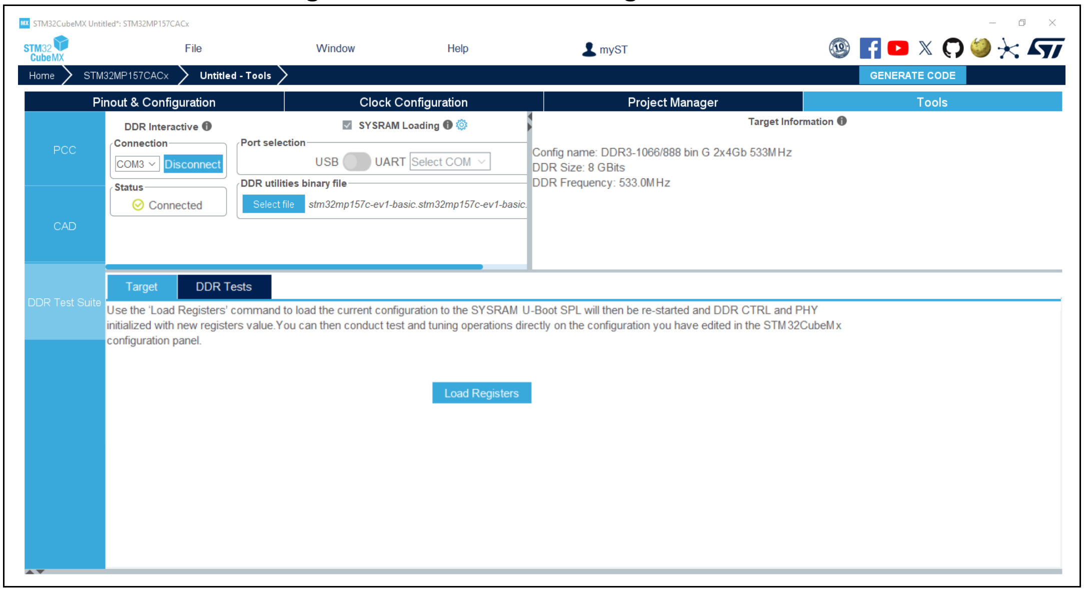
Figure 352. DDR Suite - Connection to target



If U-Boot SPL loading in SysRAM is required, it can be performed through UART or USB using the STM32CubeProgrammer tool. If not automatically detected by STM32CubeMX, the STM32CubeProgrammer tool location must be specified in the Connection settings window: click to open it. U-Boot SPL file must be manually selected in the build image folder.

Once up, the connection gives the various services and target information (see *Figure 353*).

Figure 353. DDR Suite - Target connected



Output/Log messages

STM32CubeMX outputs DDR suite related activity logs (see *Figure 354*) and interactive protocol communication logs (see *Figure 355*). They are displayed by enabling outputs from the Window menu.

Figure 354. DDR activity logs

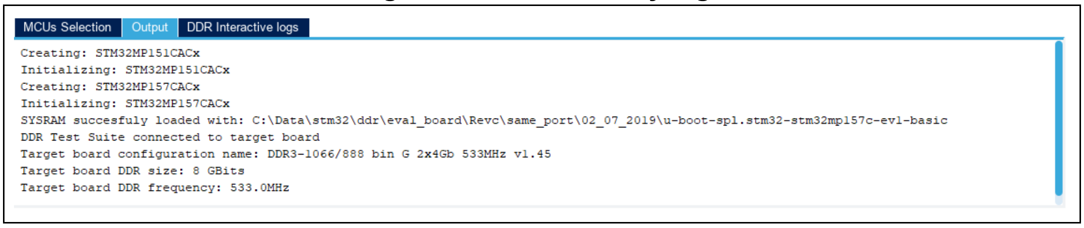
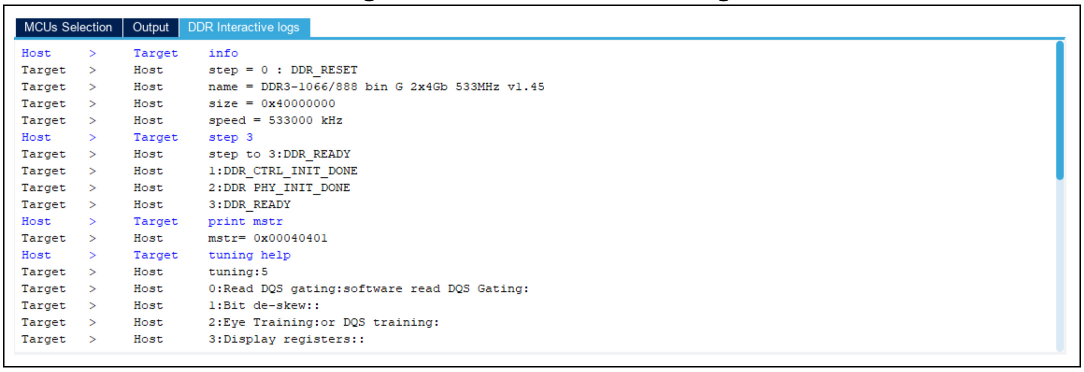


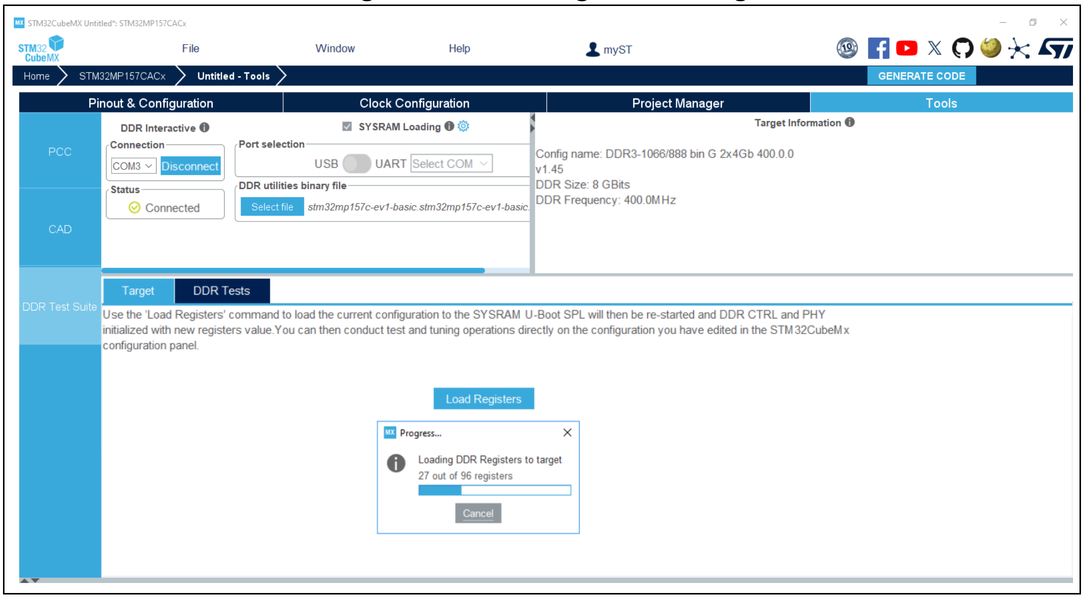
Figure 355. DDR interactive logs



DDR register loading (optional)

Once connected in DDR interactive mode, the current DDR configuration can be loaded in SYSRAM.

Figure 356. DDR register loading



This step is optional if the used U-Boot SPL already contains the required configuration. It triggers the DDR Controller and PHY initialization with those registers, and allows the user to quickly test a configuration without generating the device tree and dedicated U-Boot SPL binary file.

5.4.3 DDR testing

Prerequisites

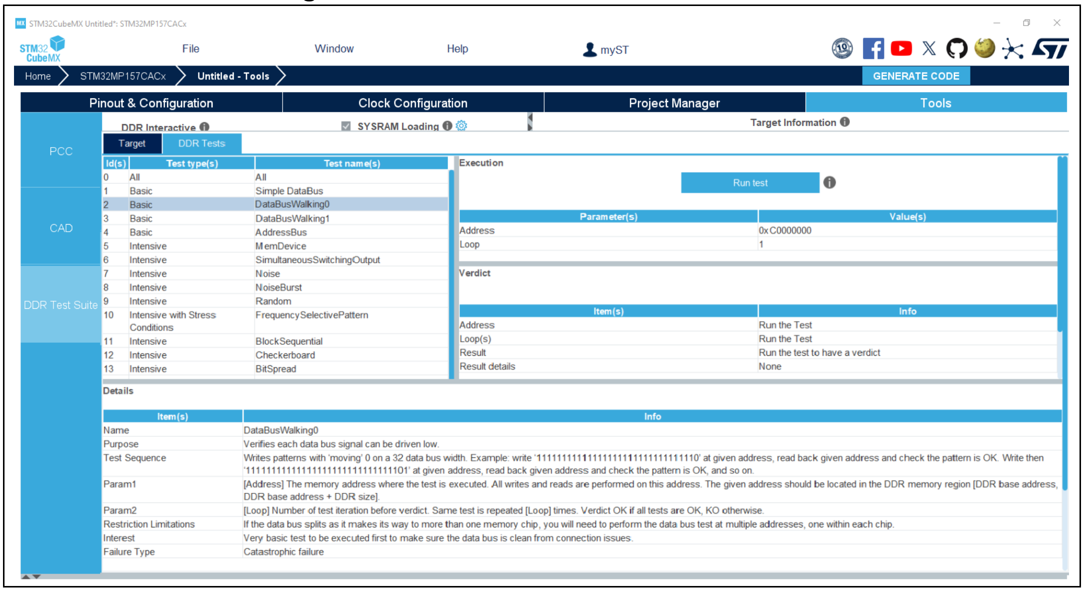
To proceed with DDR testing:

- The DDR suite must be in connected state
- The DDR configuration must be available in memory, either with the U-Boot SPL (with DDR register file in Device Tree) or in the DDR registers (see *Section 5.4.2*).

DDR test list

DDR tests are part of the U-Boot SPL (see *Figure 357*).

Figure 357. DDR test list from U-Boot SPL



New tests can be added by modifying the U-boot SPL.

Most of the tests come with parameters to be set prior to execution, such as:

- Address: the memory address where the test is executed. All writes and reads are performed on this address. The given address has to be located in the DDR memory region [DDR base address, DDR base address + DDR size].
- On STM32MP15, DDR base address is 0xC0000000 (as an example, DDR size for 4 Gbits is 0x20000000).
- Loop: number of test iterations before verdict. Same test is repeated [Loop] times. Verdict OK if all tests are OK, KO otherwise.
- Size: the byte size of the region to test. It must be a multiple of 4 (read/writes are performed on 32-bit unsigned integers), with minimal value equal to 4, and up to DDR size.
- Pattern: the 32-bit pattern to be used for read / write operations.

The DDR Suite embeds an auto-correction feature preventing users to specify wrong values.

All tests are performed with Data cache disabled and Instruction cache enabled.

DDR test results

The test verdict is reported by the U-Boot SPL: the parameters used for the tests are recalled, along with Pass/Fail status and results details (see *Figure 358*). The test history is available in the output and Logs panels (see *Figure 359*).

Figure 358. DDR test suite results

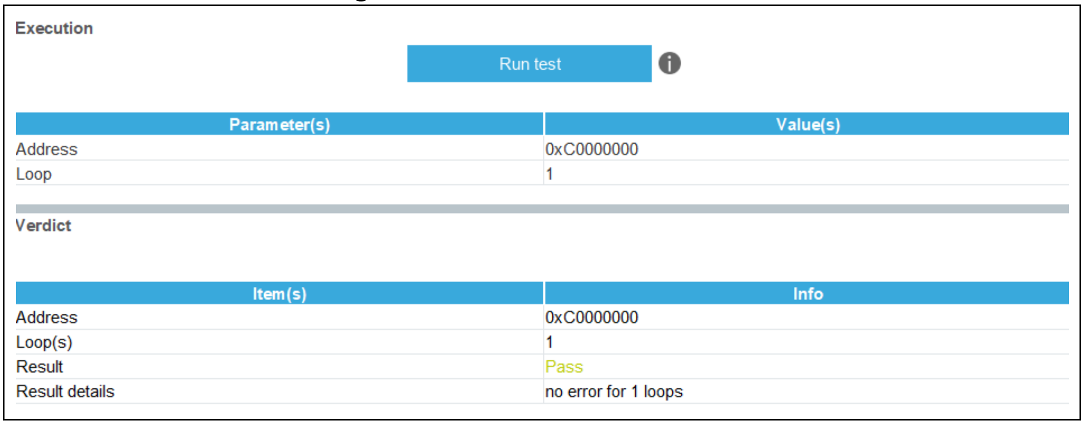
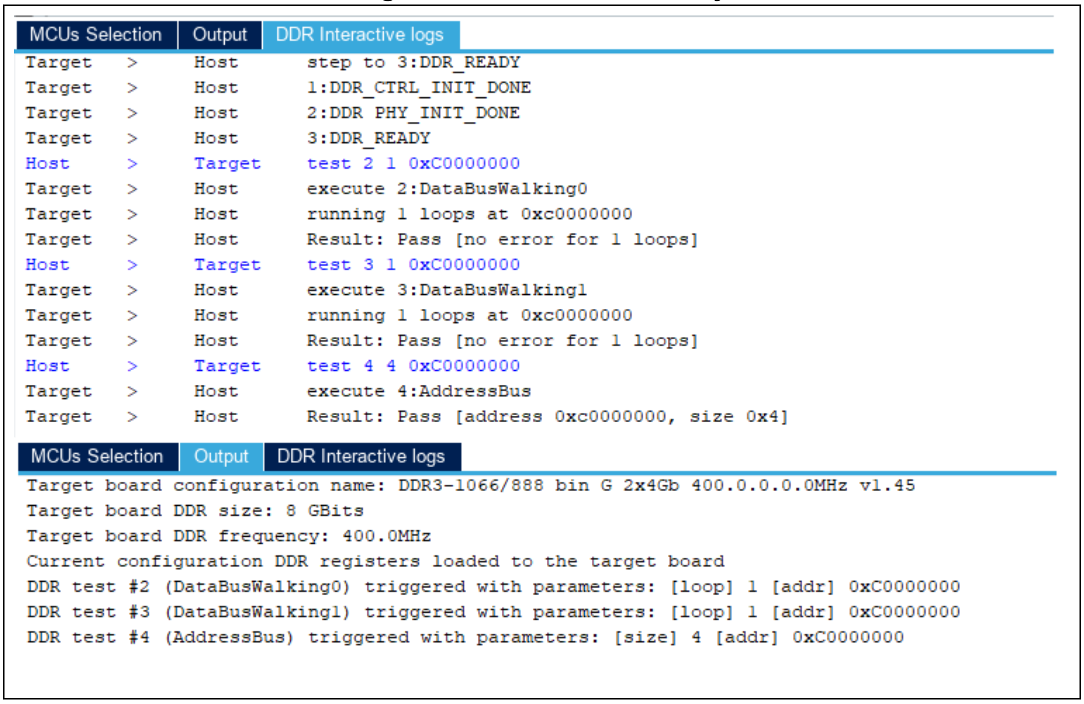


Figure 359. DDR tests history



5.5 STM32CubeMX Memory Management Tool

The Memory Management Tool (MMT) displays the memory map and defines memory attributes applied in user projects opened/created in STM32CubeMX.

The tool is located in the “Tools” tab. It allows the user to declare memory regions (referred to as application regions or AppReg) at application level.

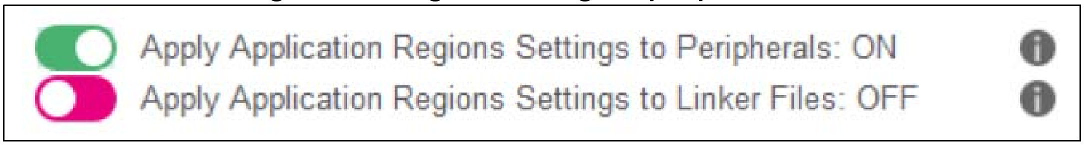
The HW constraints related to TrustZone, Memory Protection Unit, and the memory granularity are handled by MMT and made transparent to the user, so that the focus can be put on the memory regions. A linker file is generated according to the application regions declared and configured by the user.

The MMT key features are:
- Memory map display
- Application regions management
- Linker file generation

MMT interacts with peripherals starting from the moment the user enters its interface:
- Checks their settings
- Updates other peripherals involved in memory map configuration

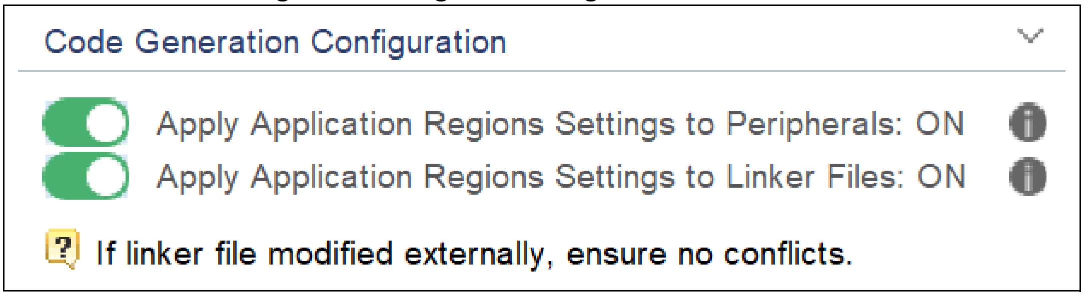
The peripherals are updated only when the first toggle button is ON.

Figure 360. Regions settings to peripherals ON



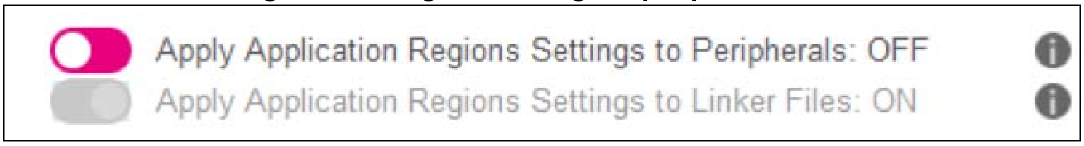
MMT updates the linker scripts only when the second toggle button is ON.

Figure 361. Regions settings to linker files ON



The applicative regions are saved into the user project even if the first toggle button is OFF.

Figure 362. Regions settings to peripherals OFF

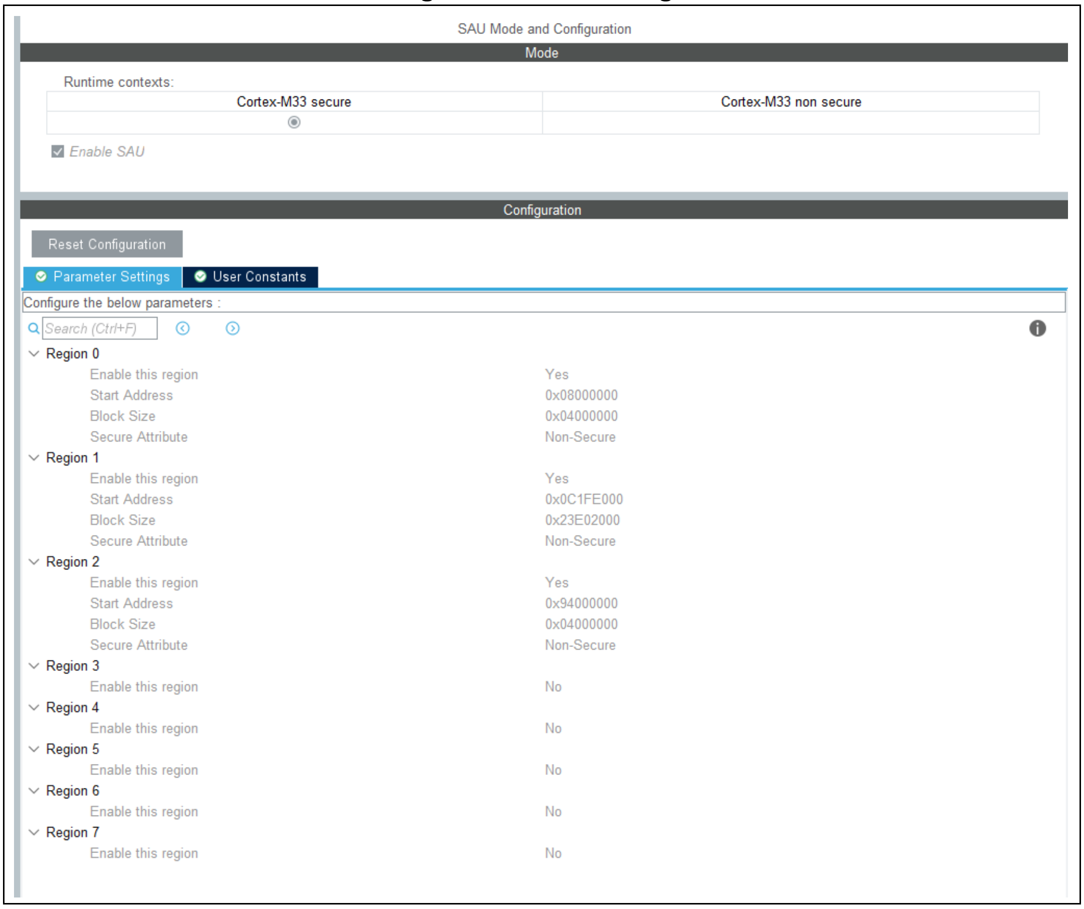


5.5.1 STM32H5, STM32U3, STM32U5, STM32WBA5, STM32WBA5M, and STM32WBA6 with TrustZone activated

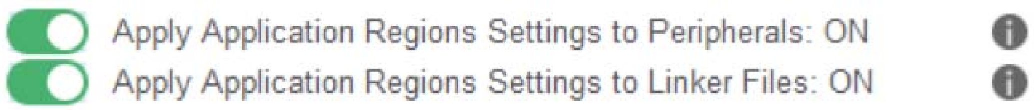
Feature: MMT usage, pinout, and configuration user interface

When the first toggle button is ON (see *Figure 360*), SAU, GTZC, Cortex-M33 (MPU), and FLASH configurations are under MMT control: their modes and parameters become read-only.

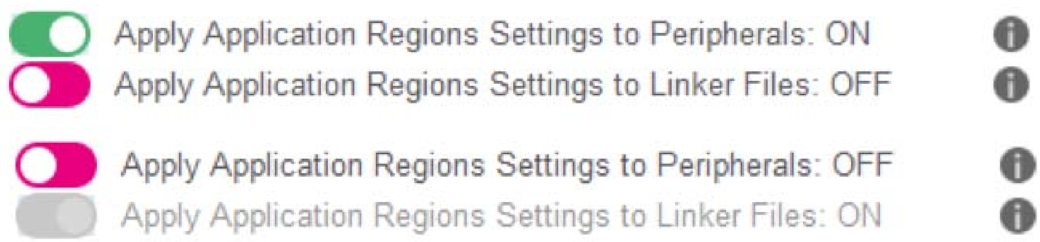
Figure 363. MMT usage



Feature: MMT usage and linker script

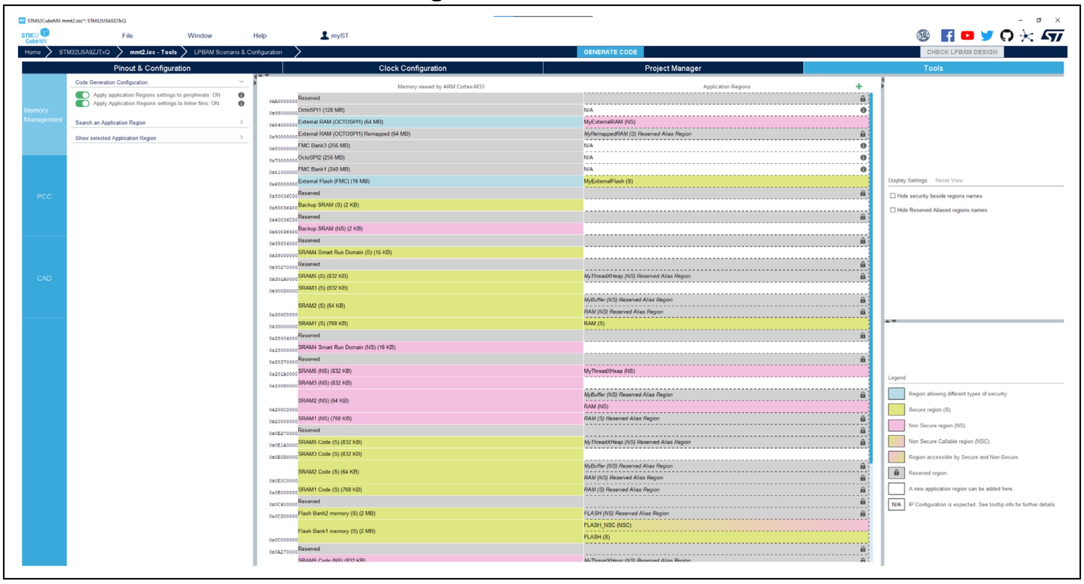


Linker files content is generated according to the configuration of application regions.



Linker files content is generated as if MMT is not used. SAU, GTZC, Cortex-M33 MPU, and FLASH are enabled, so that the user can modify the values supplied by MMT.

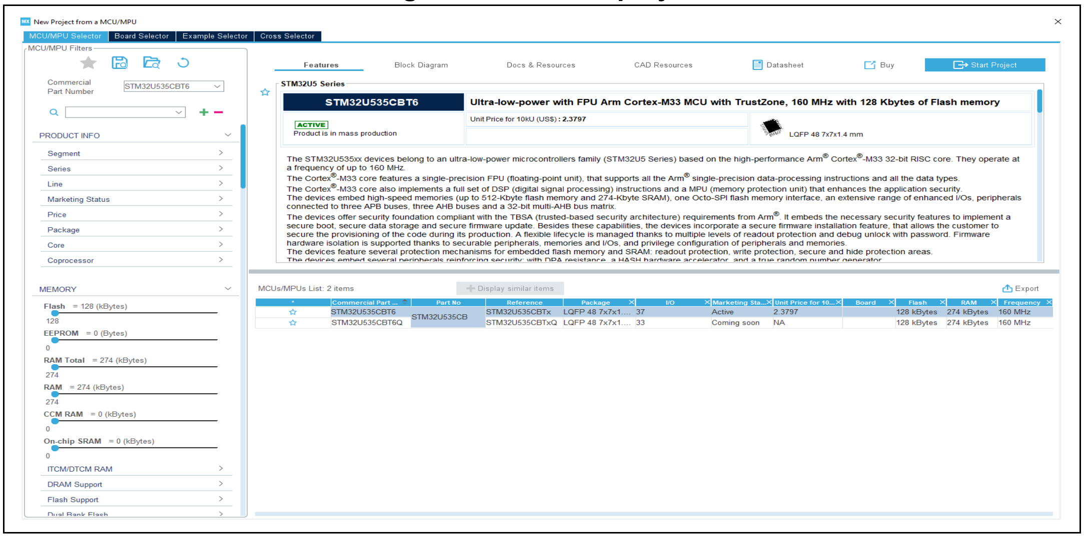
Figure 364. MMT view



5.5.2 An end-to-end usage example

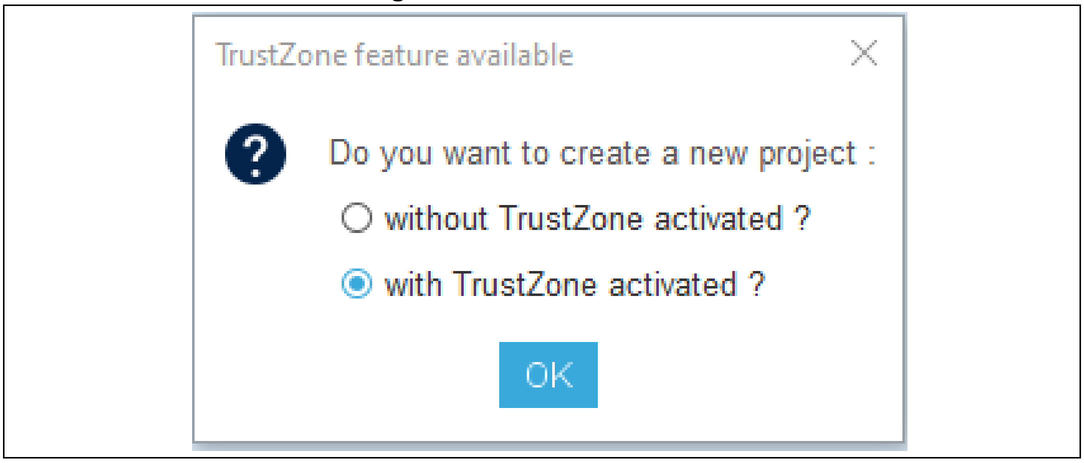
Choose a supported MCU (STM32U585x in this example).

Figure 365. Start a project



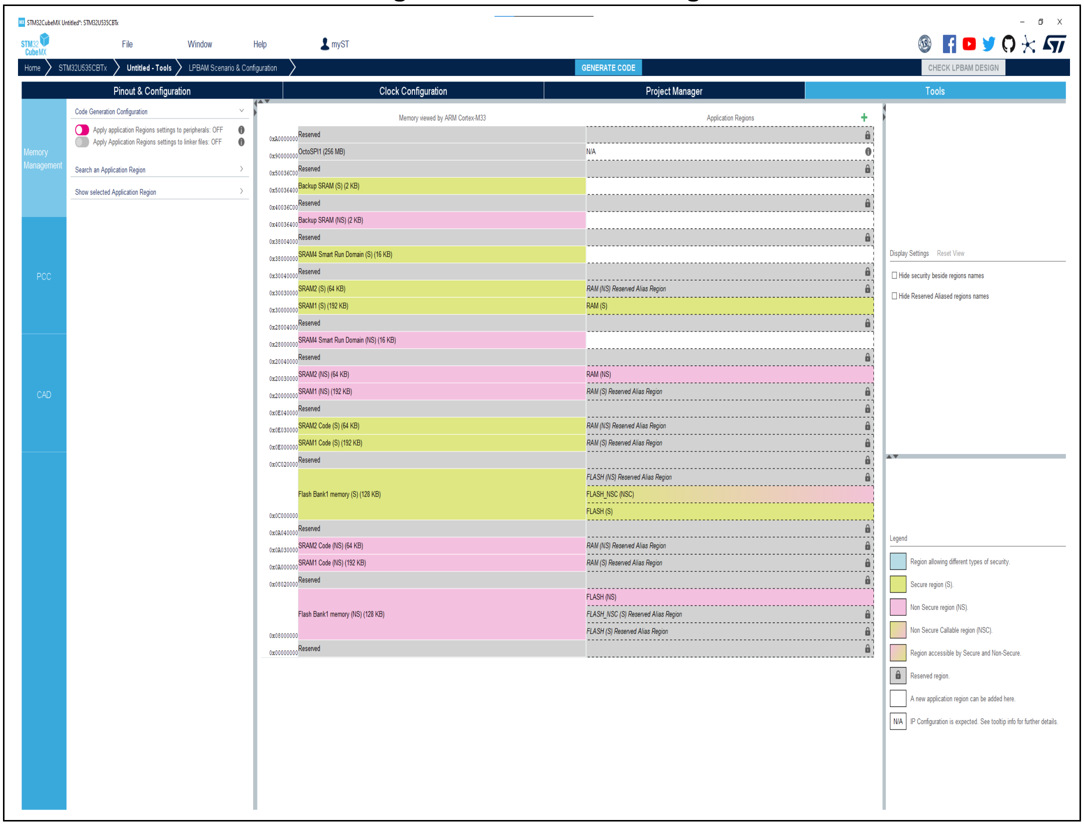
Press the “Start Project” button, and then choose the “with TrustZone activated ?” option.

Figure 366. Use TrustZone



Choose the “Tools” tab followed by the “Memory Management” option to display the Memory Management Tool (see *Figure 367*).

Figure 367. Default settings



The middle panel represents the memory, split into two columns: the left one is the memory seen by the core(s), the right one the memory set up for the application.

In this example there are two projects, a secure and a nonsecure one. The application region allocated to the secure project is green, the nonsecure application region is pink. The reserved memory regions are gray.

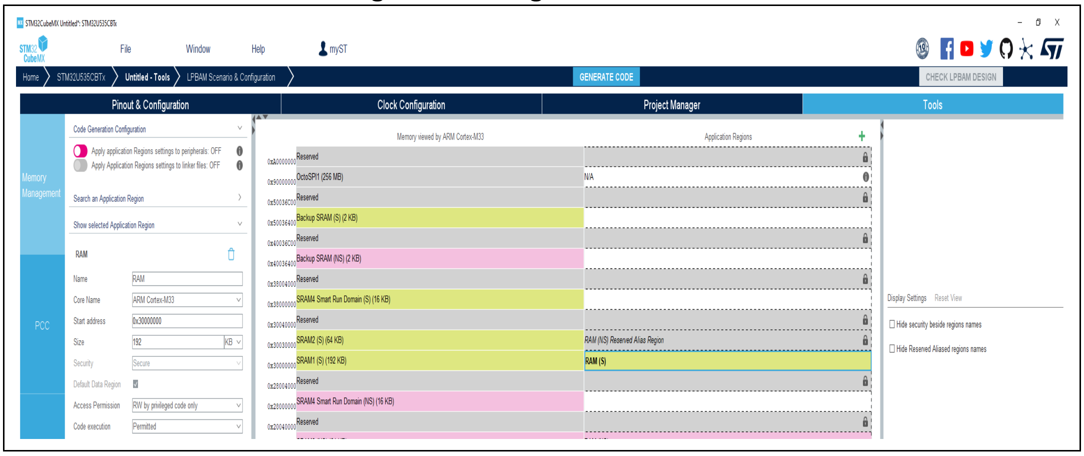
For the new project created under STM32CubeMX the tool creates the default application region to generate a valid project.

Region information

Clicking on a particular region in the Application Regions column shows the associated details on the left hand side.

You can choose to hide the name of the reserved region, or hide the Secure/Non Secure indication close to the region name (the secure/nonsecure indication is indicated by the color).

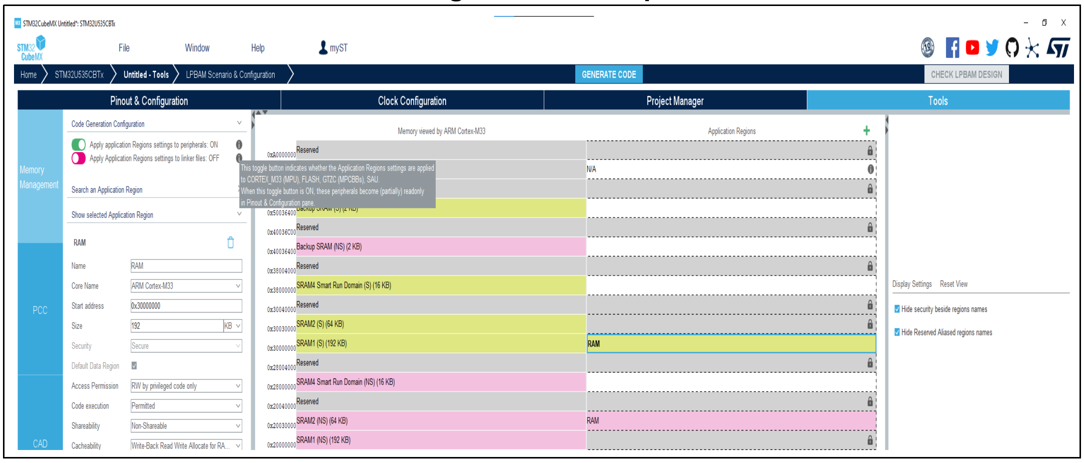
Figure 368. Region information



Code generation configuration

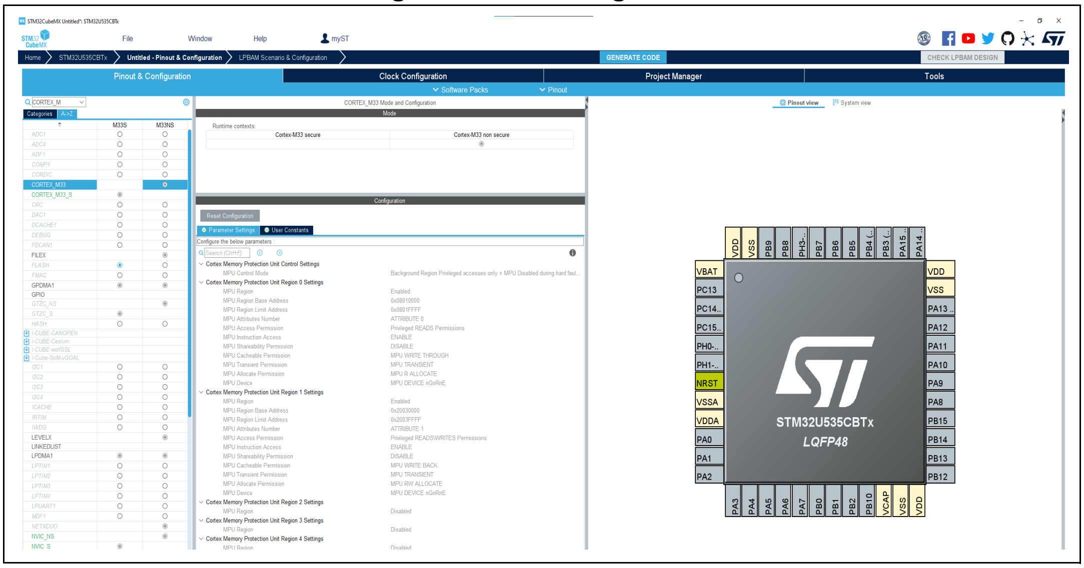
The application regions settings can be applied to peripherals on the left of the screen. The concerned peripherals are shown on the associated tooltip. This can impact their availability on the pinout screen configuration.

Figure 369. Tooltip



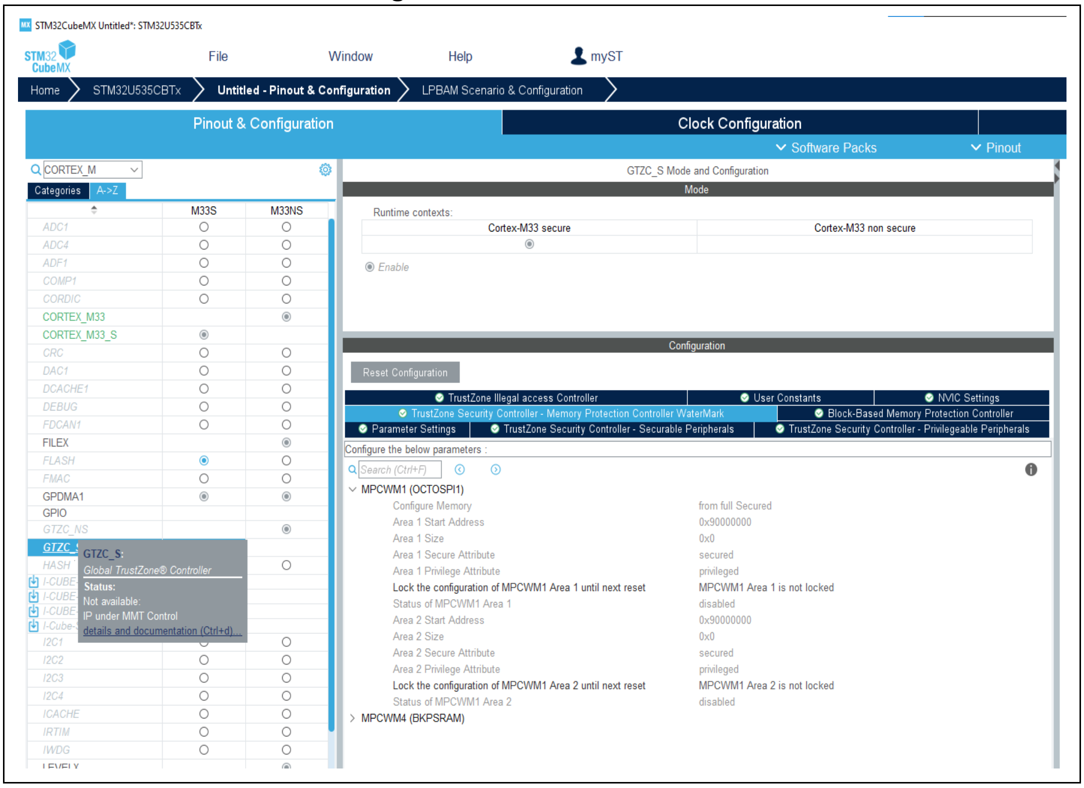
In this example, on the Pinout & Configuration panel, CORTEX_M33, FLASH, and GTZC are set, and correspond to the region configuration on the Memory Management Tool. They are grayed out, as they cannot be modified.

Figure 370. IP configuration



When an IP is under MMT control, a tooltip provides the info shown in *Figure 371*.

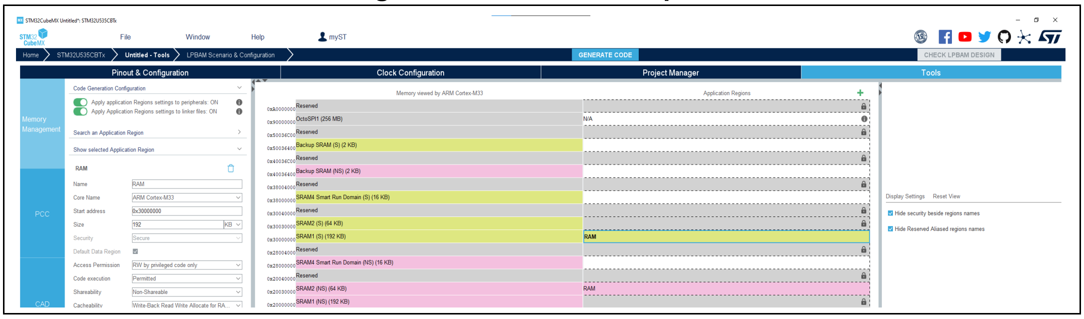
Figure 371. IP under control



Apply Application Regions settings to linker files

When this button is on, the linker scripts for the secure and non secure applications are generated, taking into account the configuration.

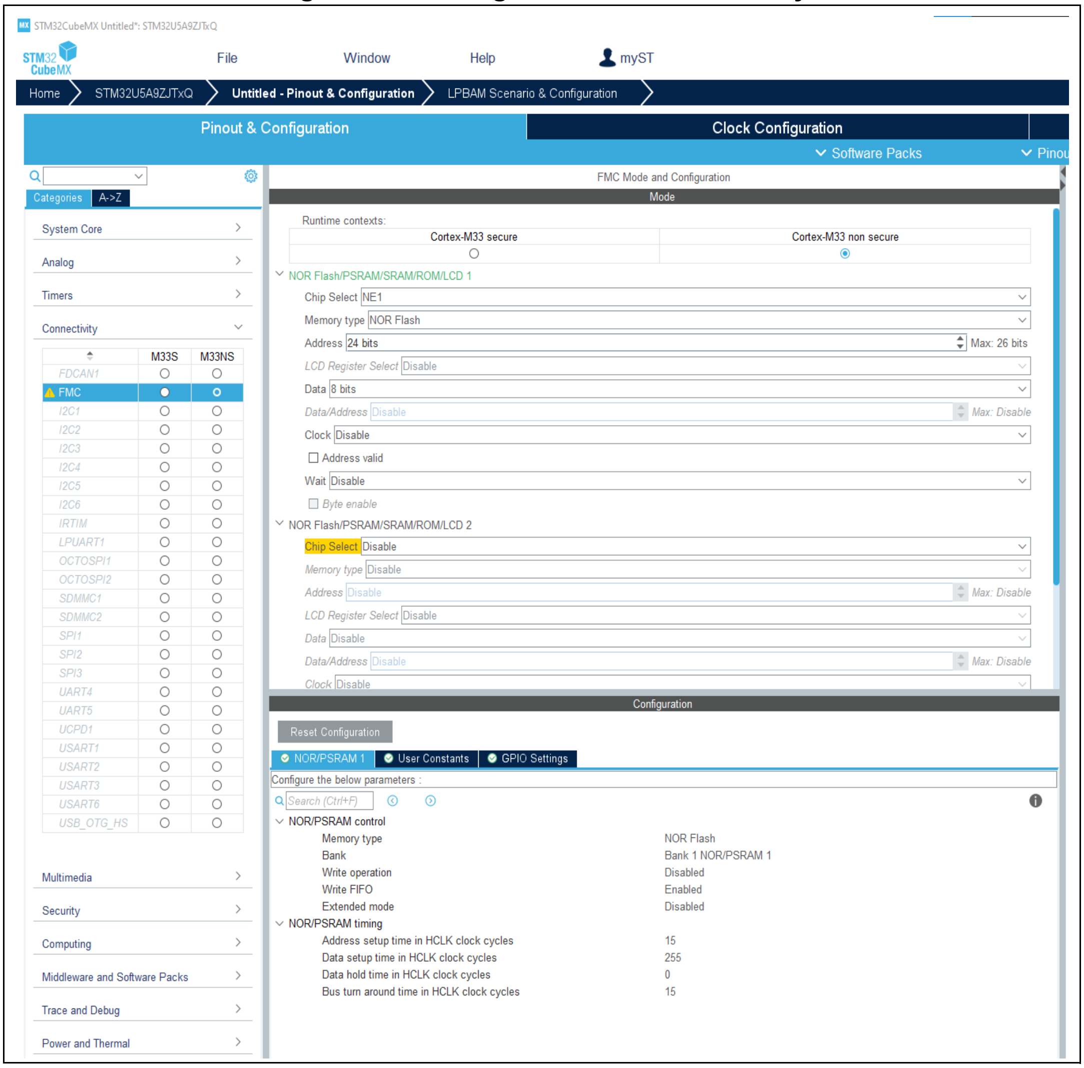
Figure 372. Linker files update



Configuring an external memory

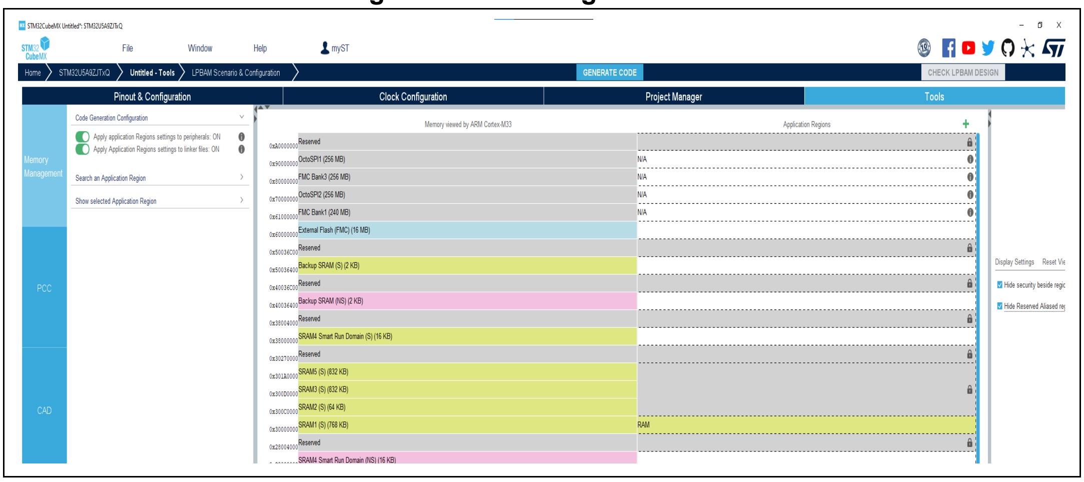
This example uses the FMC. Go to the Pinout & Configuration window (see *Figure 373*) and enable the IP.

Figure 373. Configure an external memory



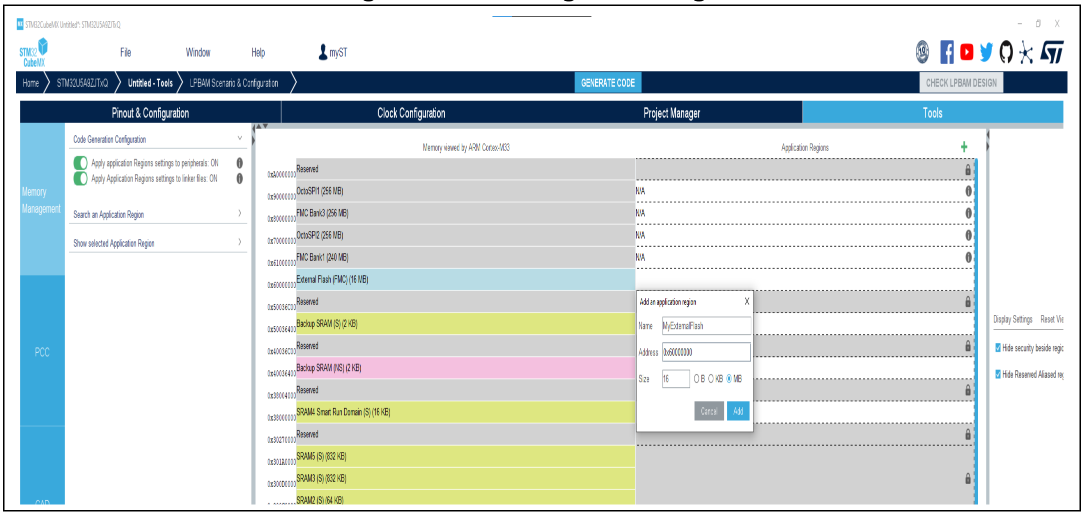
When going back to the MMT, a new region corresponding to the added FMC is created.

Figure 374. New region created



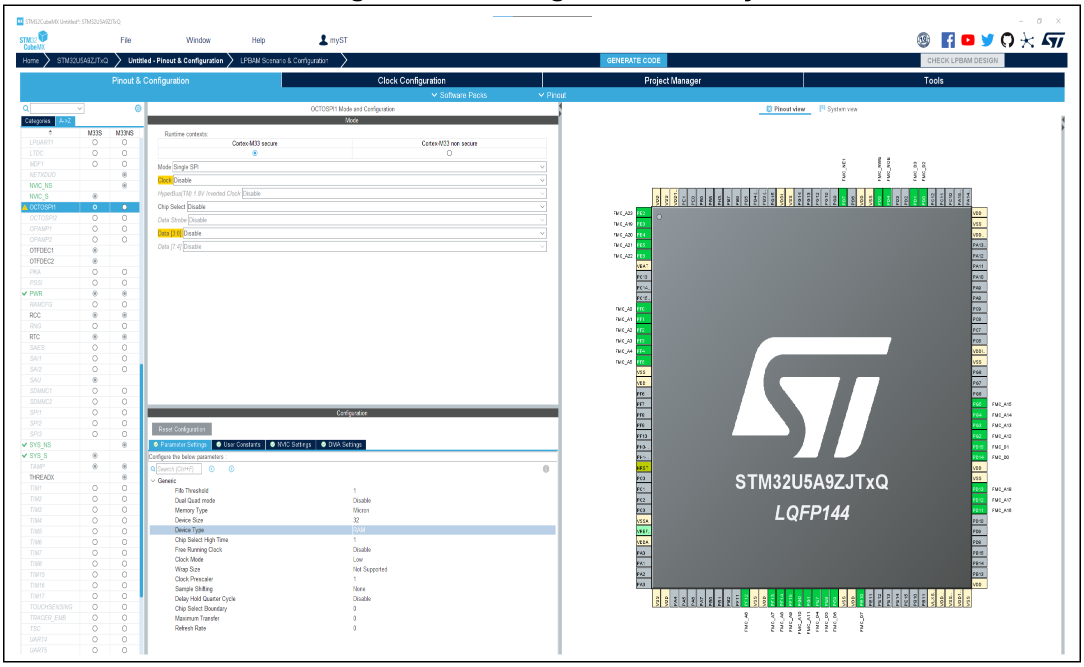
Add a new region by pressing the plus button appearing in the white space when hovering with the mouse.

Figure 375. Adding a new region



To add another external memory, go to the Pinout & Configuration view, and add OCTOSPI1 to Cortex-M33 Secure. Choose Single SPI, and specify Device Size and Device Type.

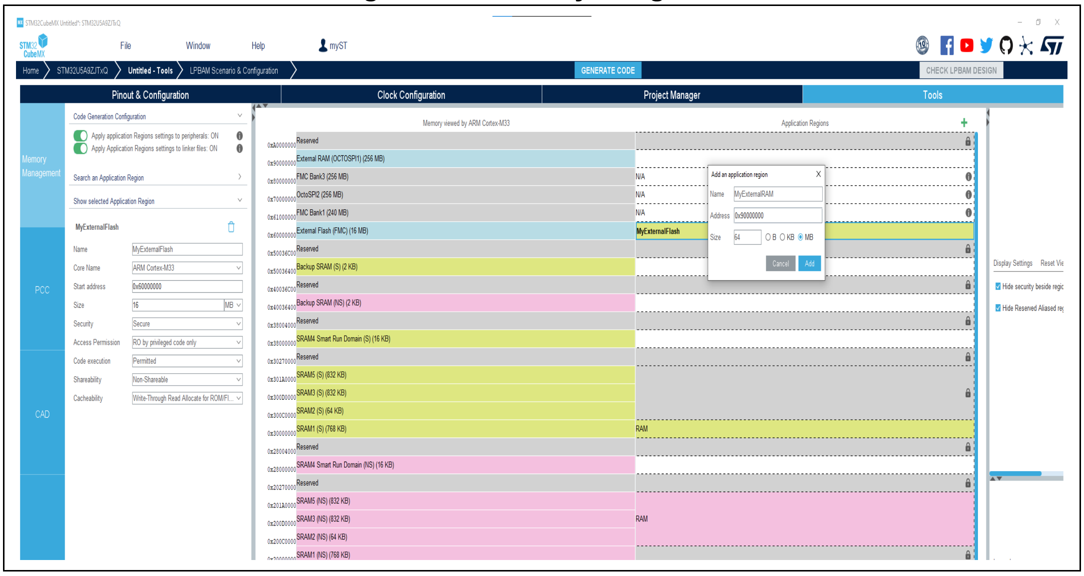
Figure 376. Adding a new memory



On the MMT there is now a new entry with OCTOSPI1.

- For our example, we need half of the available 128 Mbytes.
- Press the “+” button, set a name for the region (for instance: MyExternalRAM), and put 64 MB for its size.

Figure 377. Memory assignment



Configuring a memory region using the left panel

With the left panel (see *Figure 378*) you can adjust items such as starting position and size. In this example, the added region must be adjusted: we want it to be allocated to the non secure project, and to start in the middle of the RAM. By adjusting those values, the expected results appear (see *Figure 379*). The color is now pink (nonsecure), and the region starts in the middle of the RAM (OctoSPI1).

Figure 378. Left panel configuration

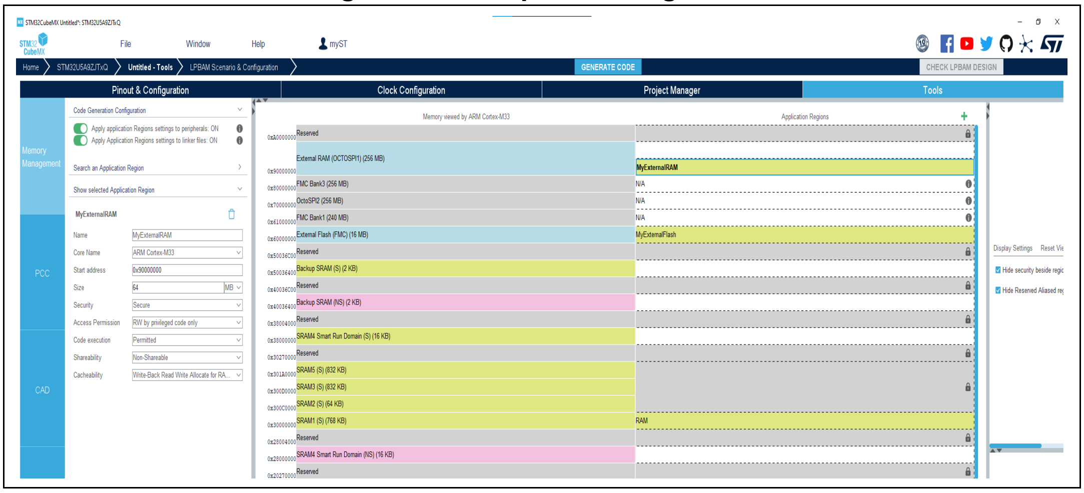
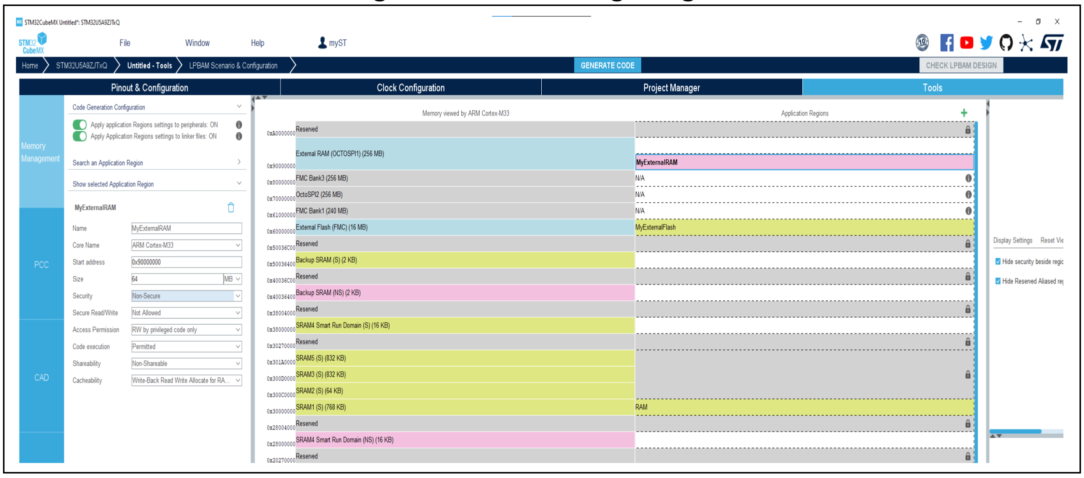


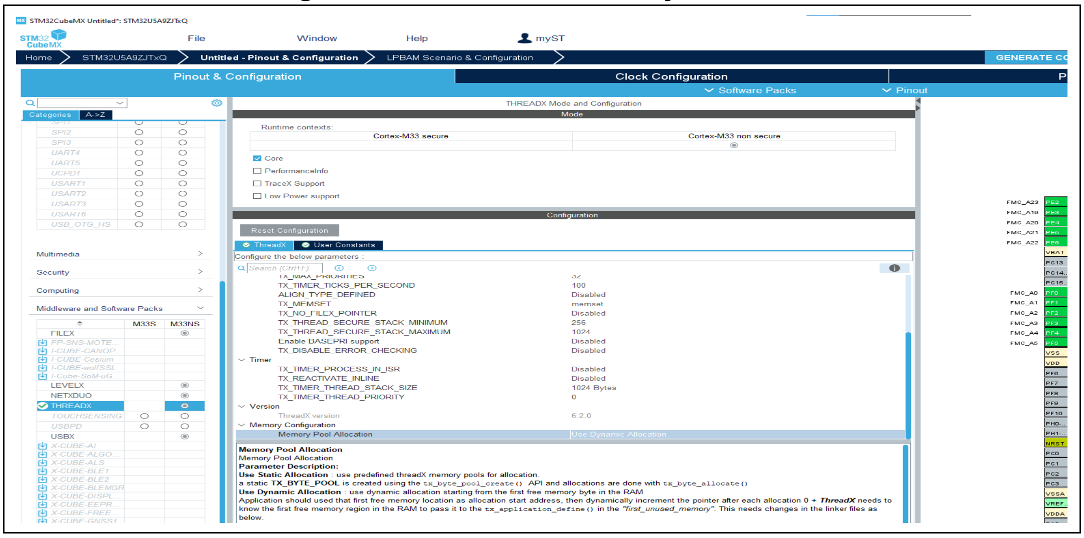
Figure 379. Allocating a region



Setting up a middleware memory location

The application needs ThreadX. Go back to the “Pinout & Configuration” tab. Choose ThreadX, then use the Use Dynamic Allocation under Memory Configuration.

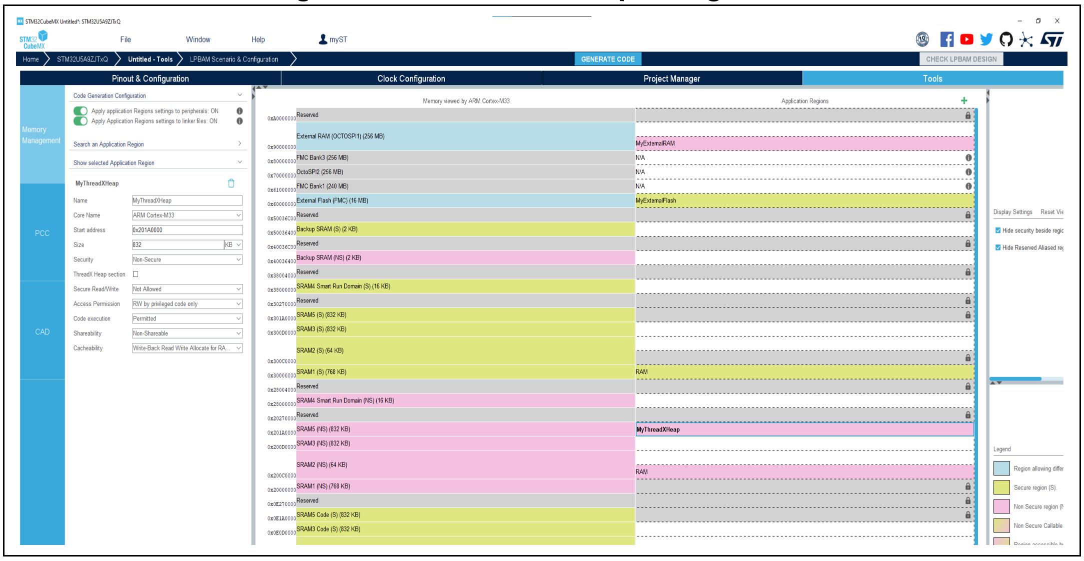
Figure 380. Middleware memory allocation



To finish the configuration, go back to MMT. We want ThreadX to use a dedicated application region for its heap memory allocation. To do so, simply click the RAM region, and reduce its size to 17 Kbytes using the left panel. We then add a new region to the newly freed space, and call it MyThreadXHeap.

As ThreadX has been selected, on the Pinout & Configuration you can see a tick box called ThreadX Heap section. When this box is selected, the tool ensures that ThreadX memory allocation happens only in that particular region.

Figure 381. Middleware heap configuration

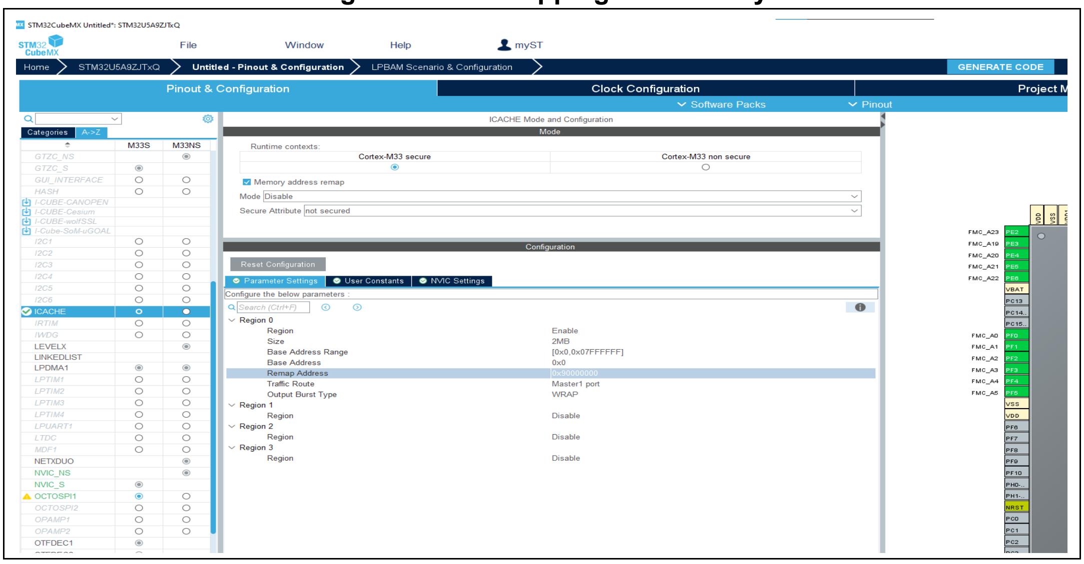


Remap

For performance reasons, part of the application must run on the internal memory (much faster than the external memory). To do so, remap the added external RAM to an available internal memory region:

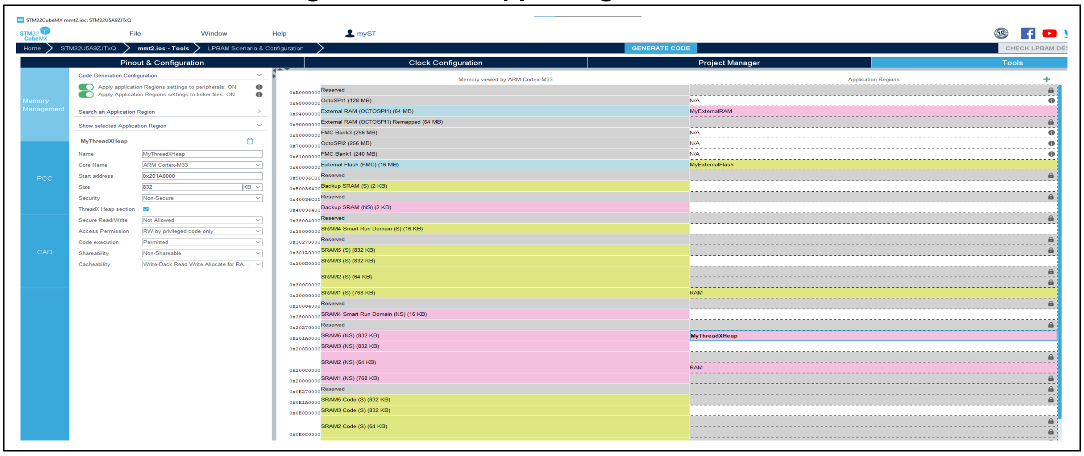
- Go to the Pinout & Configuration tab
- Enable ICACHE, select the Memory address remap tick box
- Select a region and set the memory size to 64 Mbytes
- Change the Remap address to 0x9000 0000

Figure 382. Remapping the memory



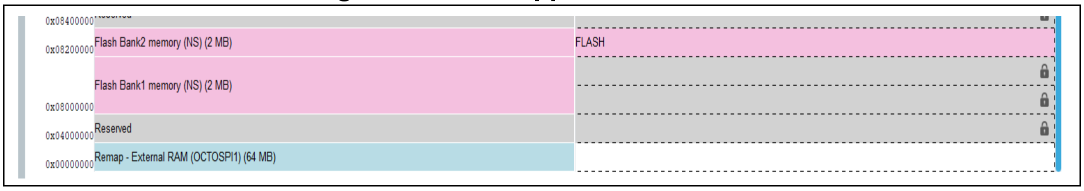
- Go back to the MMT tab. Region 0x9000 0000 is named with Remapped, with the amount of RAM previously selected.

Figure 383. Remapped region is renamed



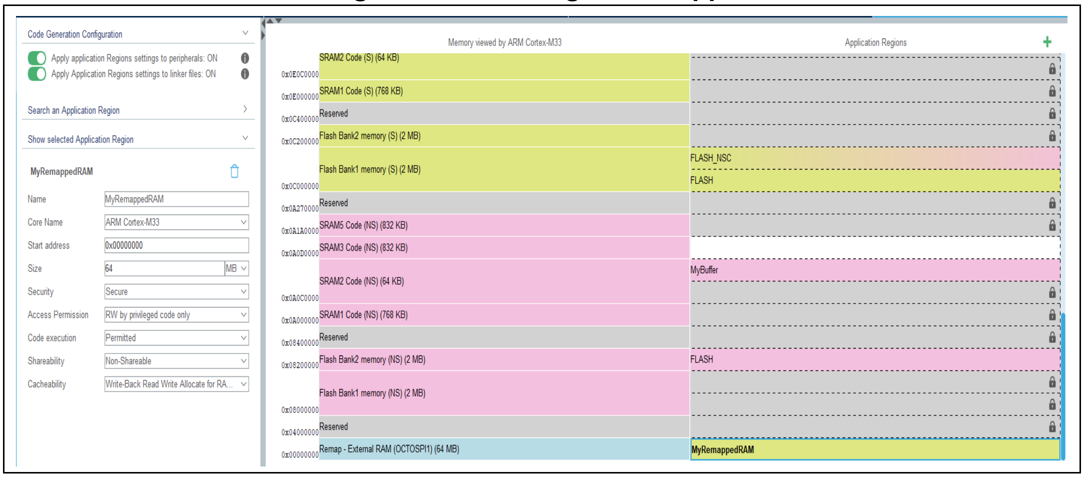
- There is also a Remap – External RAM(OCTOSPI1) added at address 0x0000 0000.

Figure 384. Remapped start address



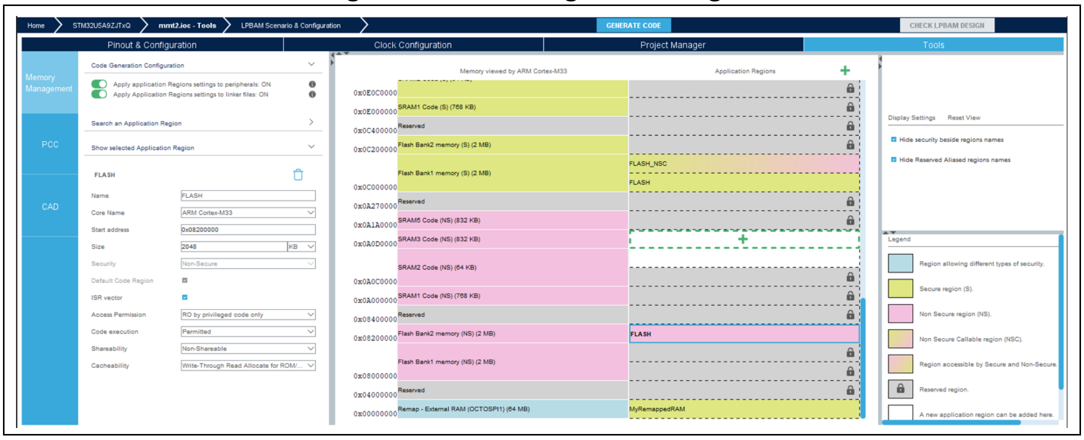
- Add a new region named “MyRemappedRAM” at that address.

Figure 385. New region remapped



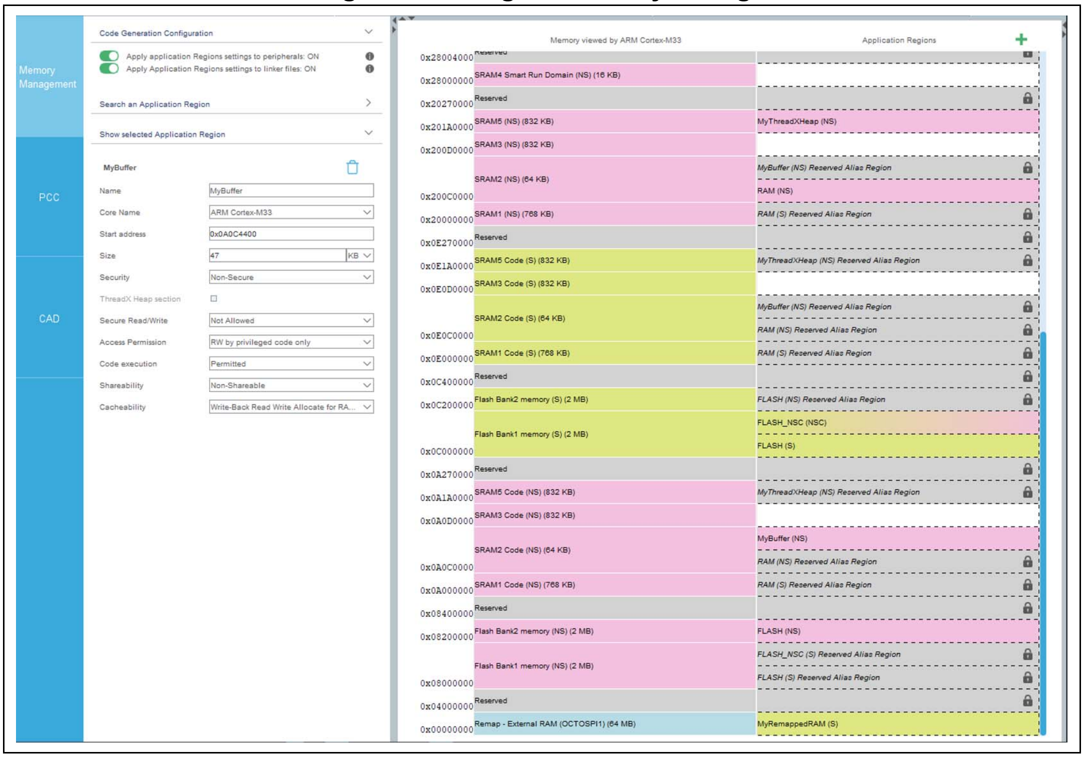
The default regions cannot be removed, but can be resized. As an example, the FLASH is where the application code is hosted. You cannot untick the Default Region.

Figure 386. Resizing default region



Changing the security of an application region mapped on aliased RAM or FLASH moves it in an aliased RAM or FLASH corresponding to the new security setting. Graphically, the region moves up and down, depending on the area it will go, as the same physical memory is seen by the core at different locations.

Figure 387. Region security change

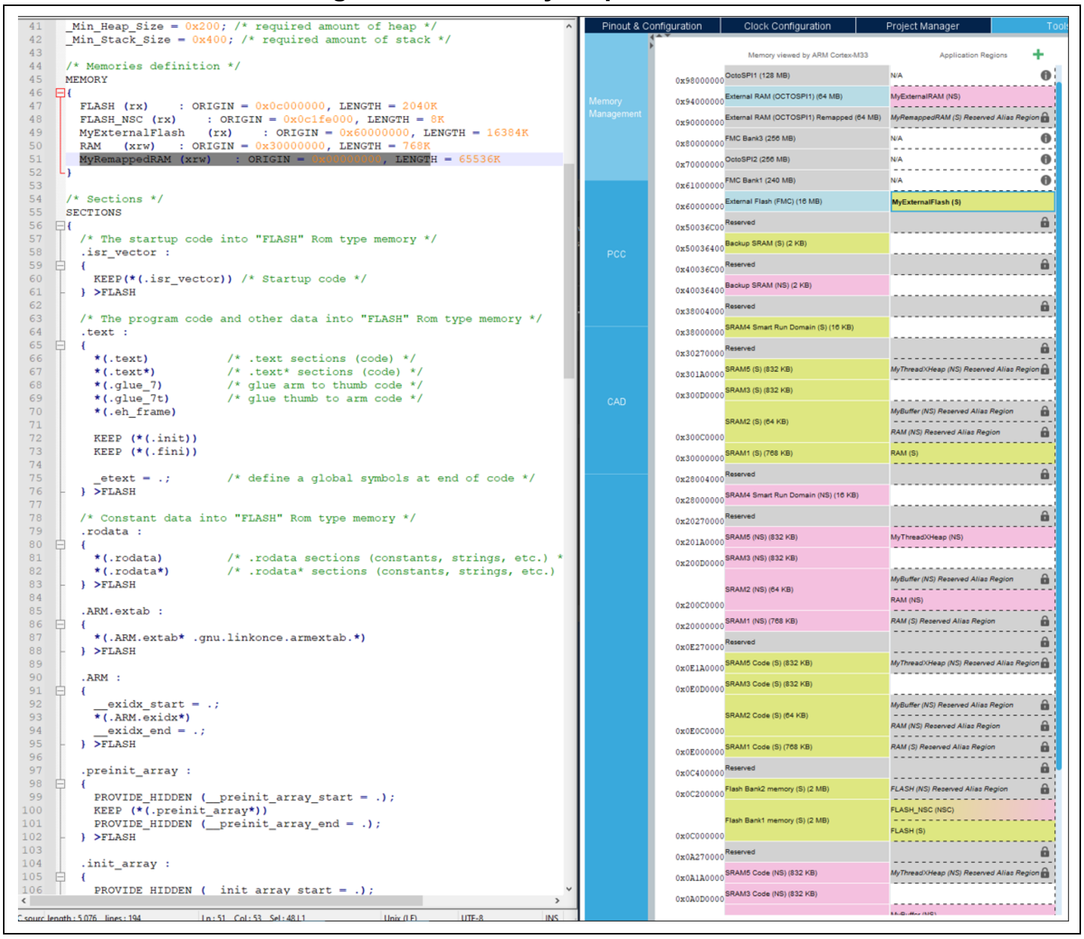


Code generation

- Go to the project manager, set a name to your project, Choose CubeIDE as a toolchain and press GENERATE CODE
- Navigate to the generated Secure Project and open the linker definition file. Under the Memories definition you will see the defined memories with their start address and

length. This file shows only the secure regions in green. Open the nonsecure linker file and check the same location for the memory regions allocated to the nonsecure area.

Figure 388. Memory map in linker file



5.5.3 STM32H7 single core and STM32U5 without TrustZone activated

Feature: MMT usage, pinout, and configuration user interface

When the first toggle button is ON, Cortex-M33 (MPU for STM32U5) and Cortex-M7 (MPU for STM32H7) are under MMT control (see, respectively, *Figure 389* and *Figure 390*): modes and parameters become read-only.

The middle panel (see, respectively, *Figure 391* and *Figure 392* for STM32U5 and STM32H7) represents the memory, split into two columns: the left one is the memory seen by the core(s), the right one the memory set-up for the application.

For the new project created under STM32CubeMX the tool creates the default application region to generate a valid project.

Figure 389. MMT usage (STM32U5)

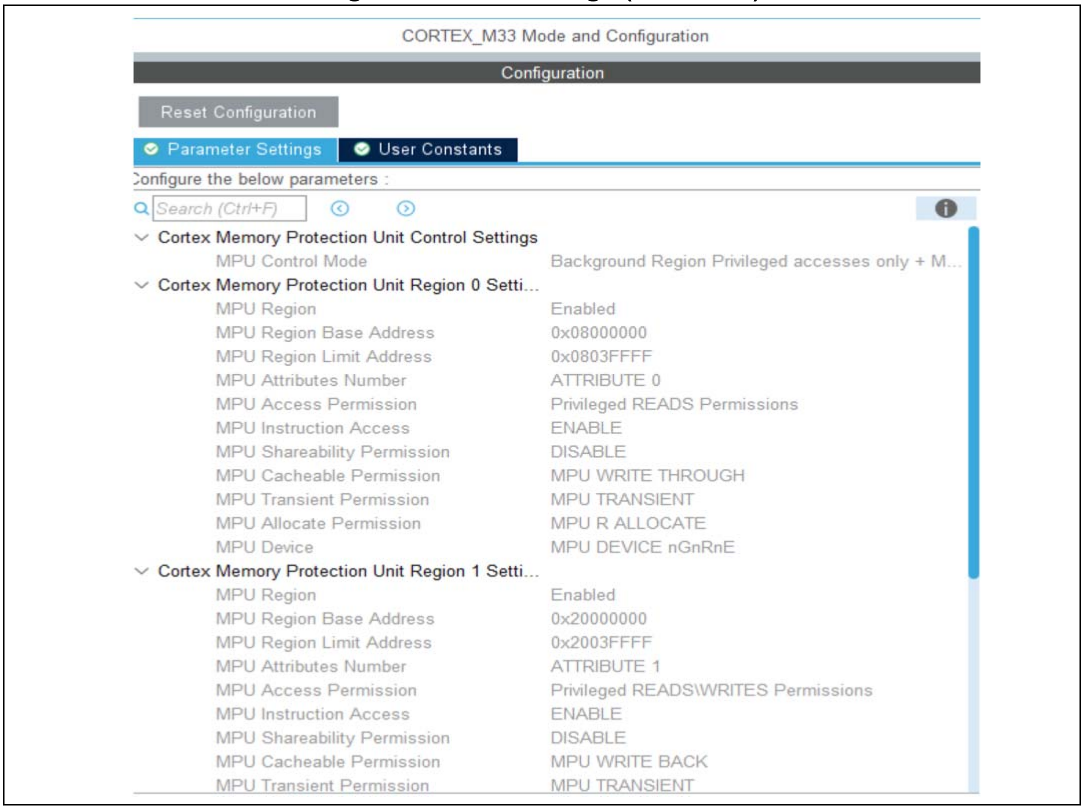


Figure 390. MMT usage (STM32H7 single core)

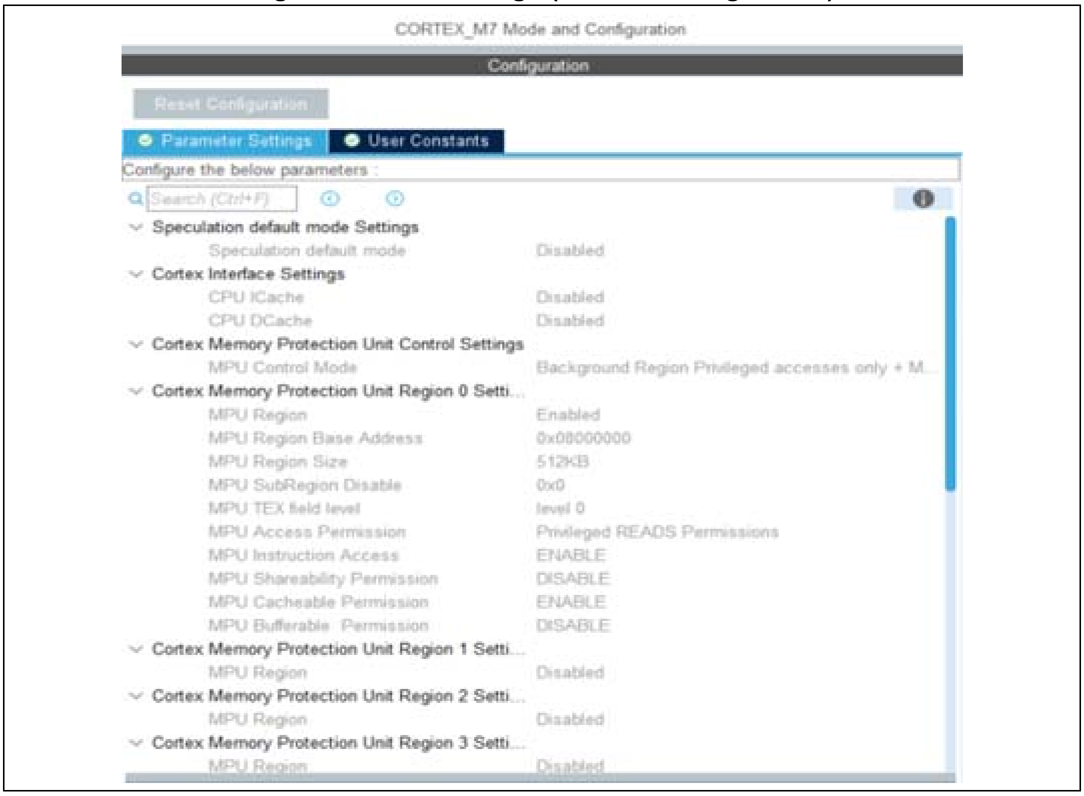


Figure 391. MMT view for U5 without TrustZone

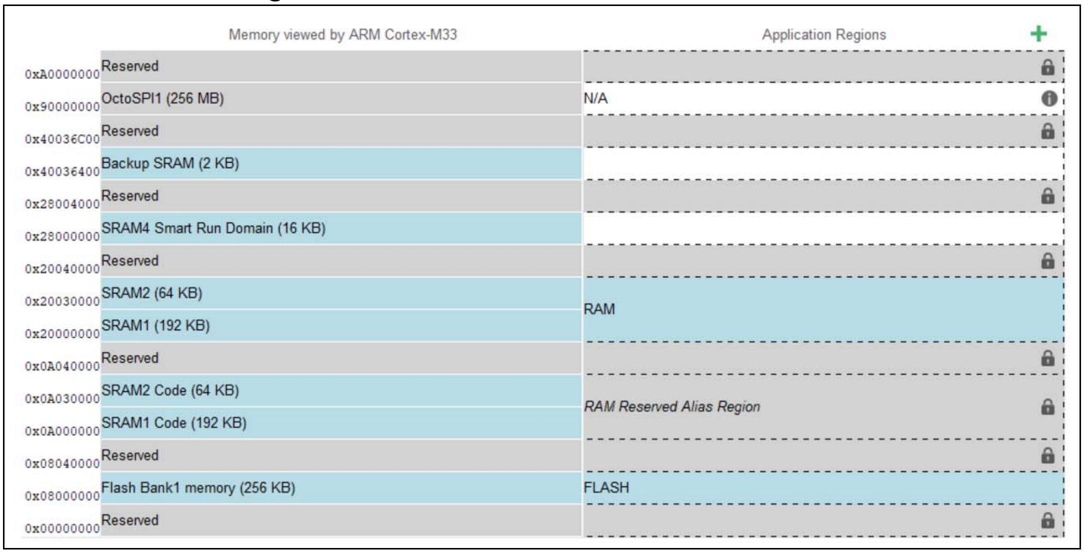
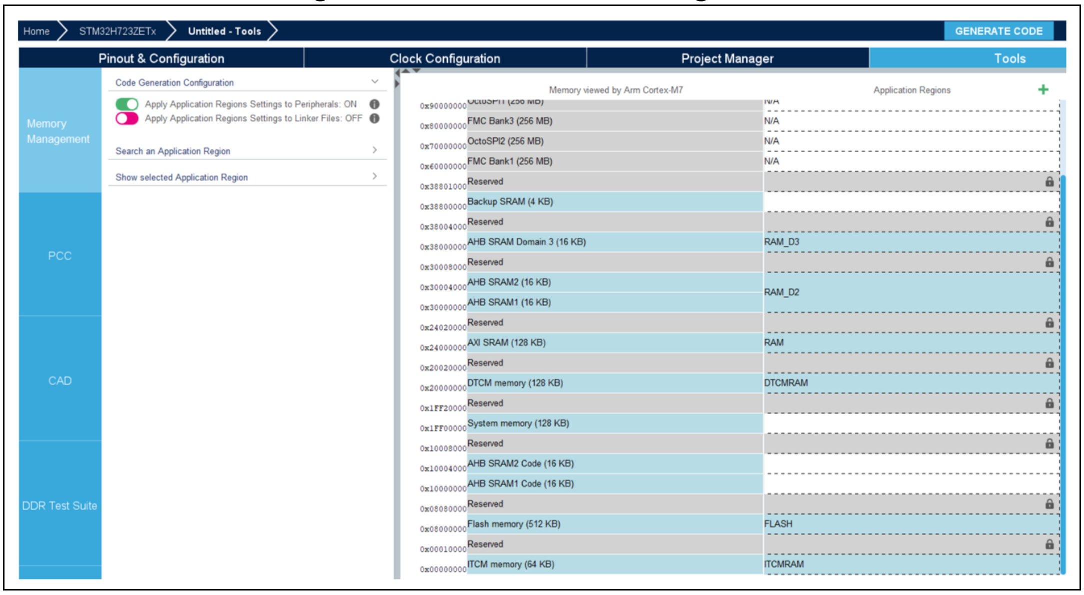


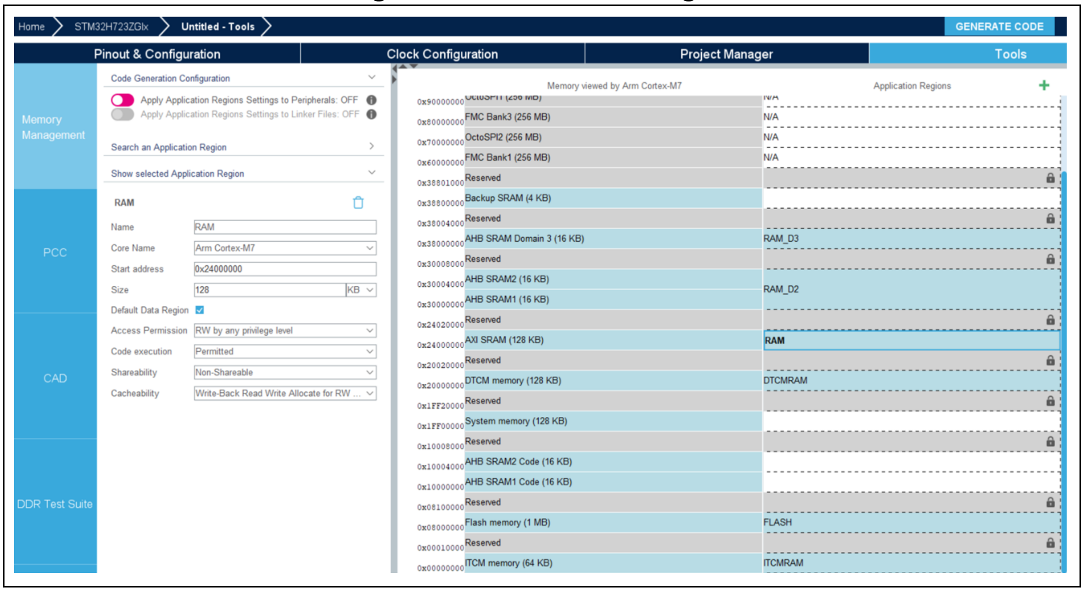
Figure 392. MMT view for H7 single core



The middle panel represents the memory, split into two columns: the left one is the memory seen by the core(s), the right one the memory set-up for the application.

For the new project created under STM32CubeMX the tool creates the default application region to generate a valid project. The default data region can be updated by the user to choose another region as RAM, but there must always be a default data region (*Figure 393*).

Figure 393. Default data region



FMC impact on MMT

When activating FMC and SDRAM Bank1, a tab mapping (see *Figure 394*) is displayed, with three options:

1. Default mapping (see *Figure 395*): MMT initializes as default position of SDRAM Bank1, SDRAM Bank2, and NOR PSRAM (default viewer of MMT)
2. NOR/PSRAM bank and SDRAM Bank1/2 are swapped: MMT swaps the position of SDRAM Bank1 and NOR PSRAM Bank1 (see *Figure 396* and *Figure 397*)
3. SDRAM Bank2 remapped on FMC Bank2 and still accessible at default mapping: MMT updates the position of SDRAM Bank1 to be remapped on position of FMC Bank2 (see *Figure 398* and *Figure 399*)

Figure 394. FMC activation

Figure 395. Default mapping

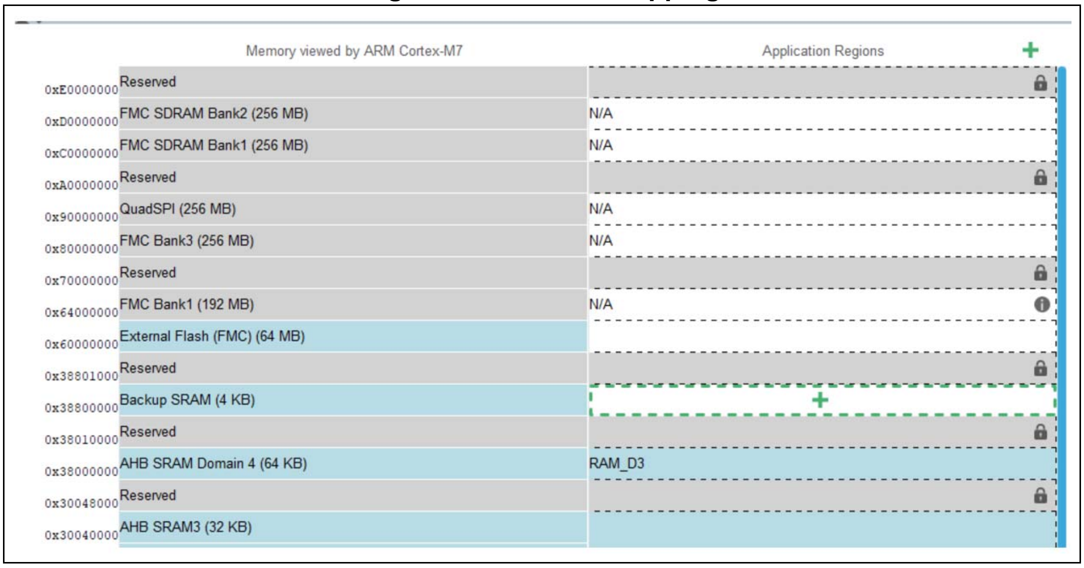


Figure 396. Before the swap

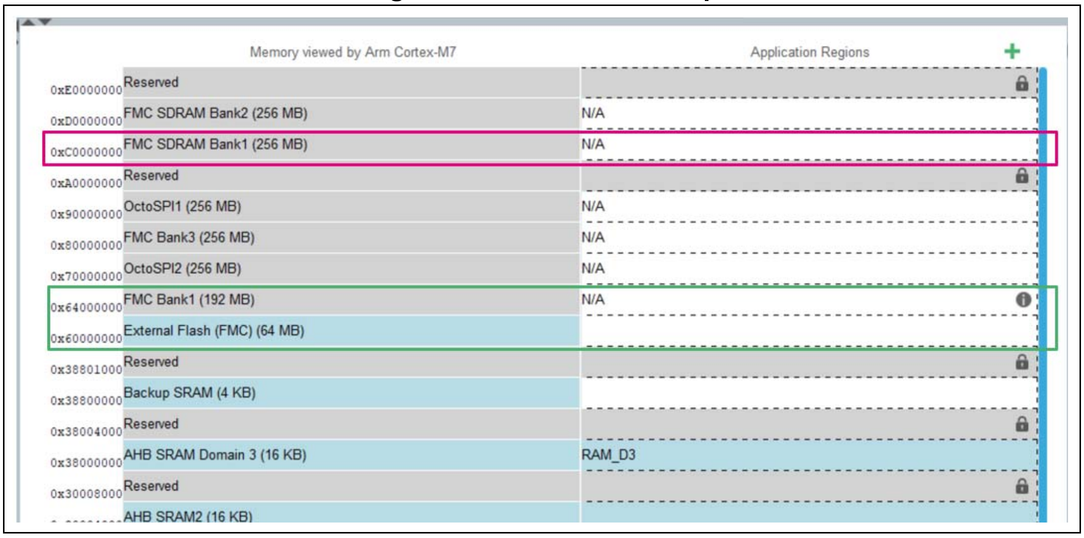


Figure 397. After the swap

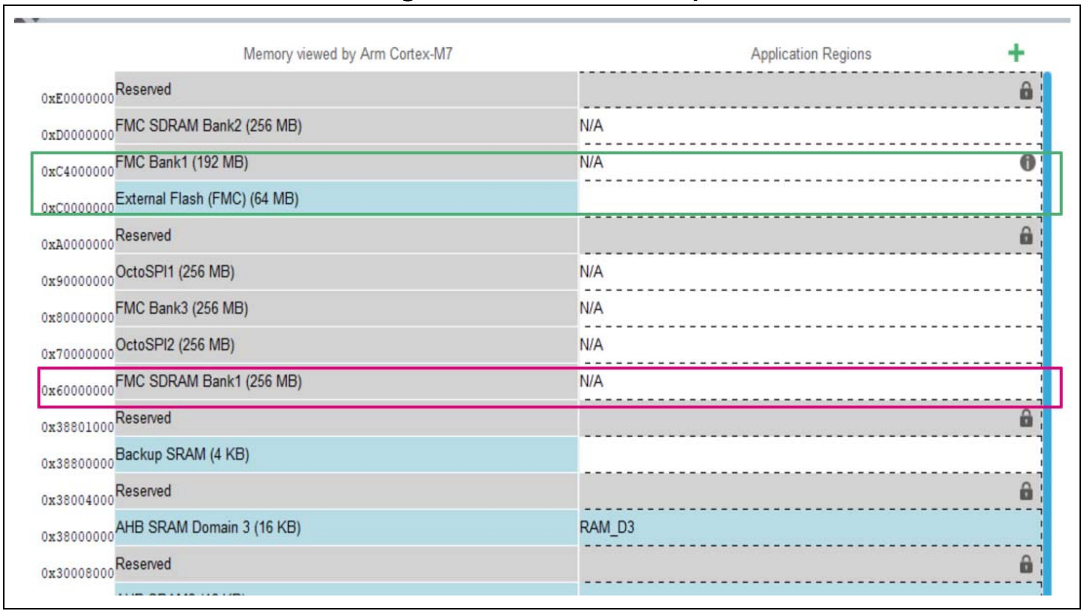


Figure 398. Before remapping

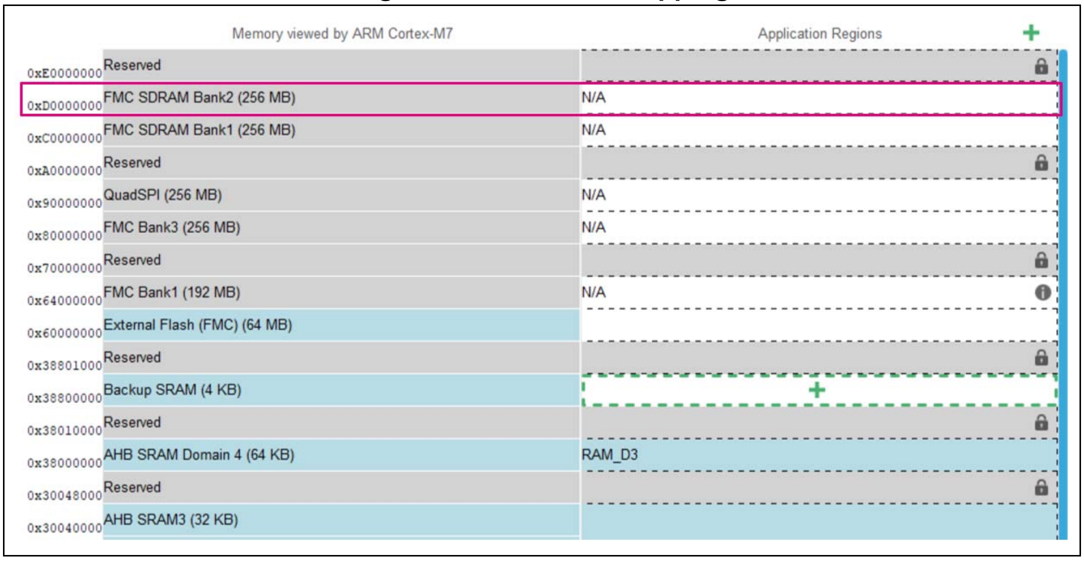
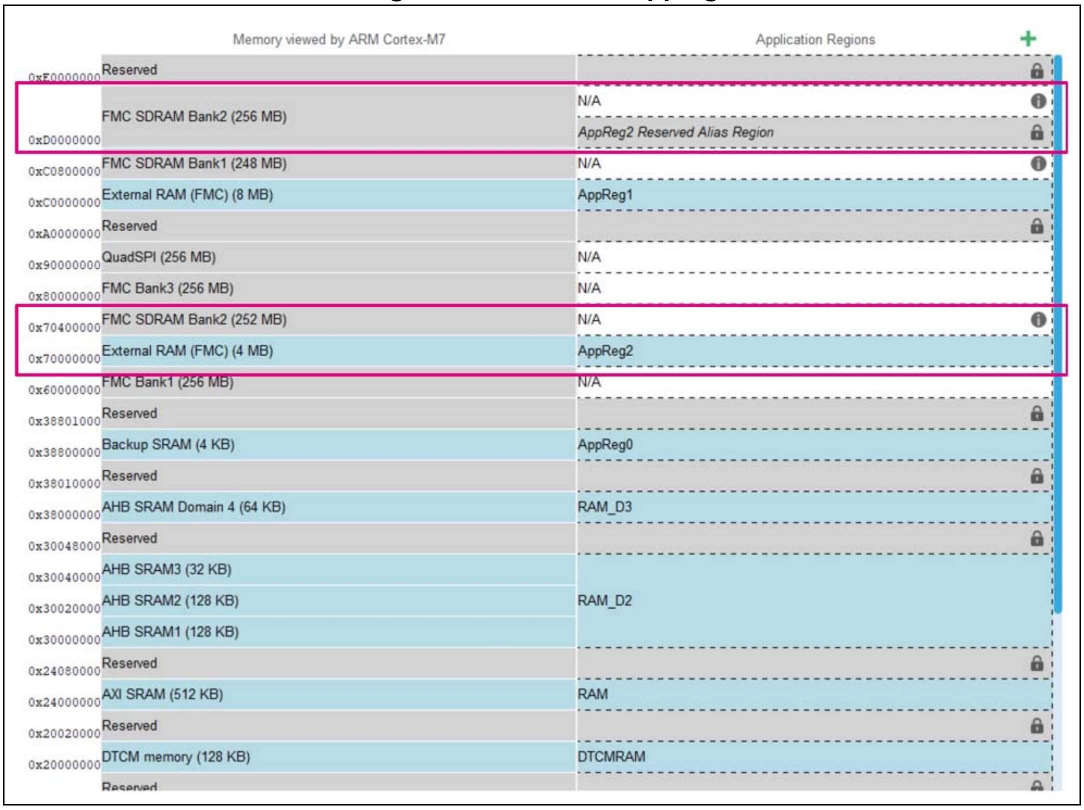


Figure 399. After remapping



ETH impact on MMT for STM32H7 single core

An example of MMT configuration of the ETH IP on the STM32H723VETx MCU

1. Activate the IP ETH:
   - MMT creates three application regions within the MMT view.
   - To change the start address and the size of each region, update the ETH parameters.
2. Press the radio button “Apply Application region Settings to Peripherals ON”, ETH will be partially under MMT control.
3. Press the Generate Code button to generate code for both applications.
   - Apply Application Regions settings to linker files:
4. When this button is on, the linker scripts are generated, considering the configuration.
5. After the code generation, navigate to the generated folder:
   - Open the linker definition file.
   - Under the Memories definition you can see the memories with their start address and length, according to the configuration made in STM32CubeMX.

Figure 400. ETH MMT regions

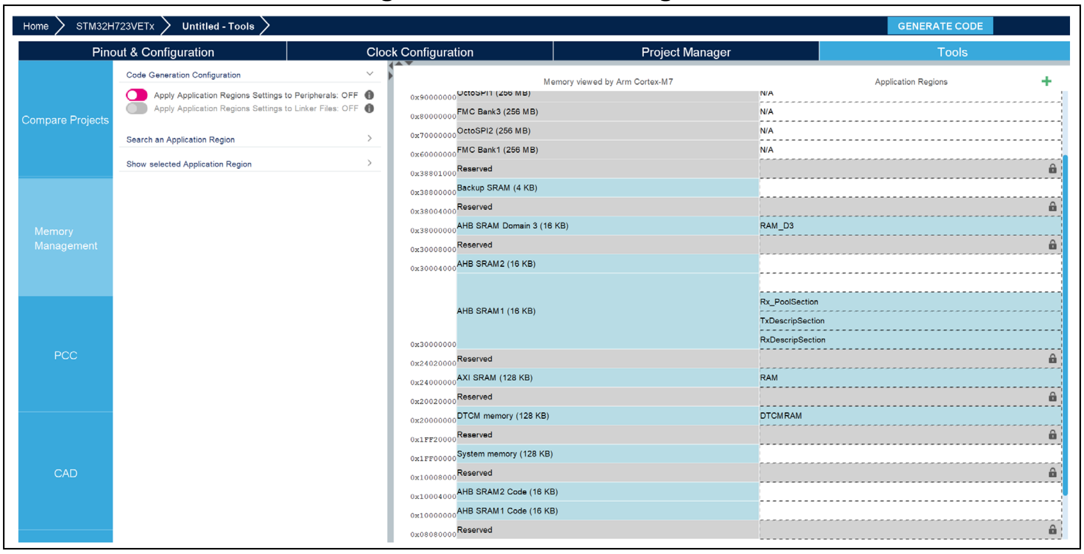


Figure 401. ETH configuration for STM32H723VETx MCU

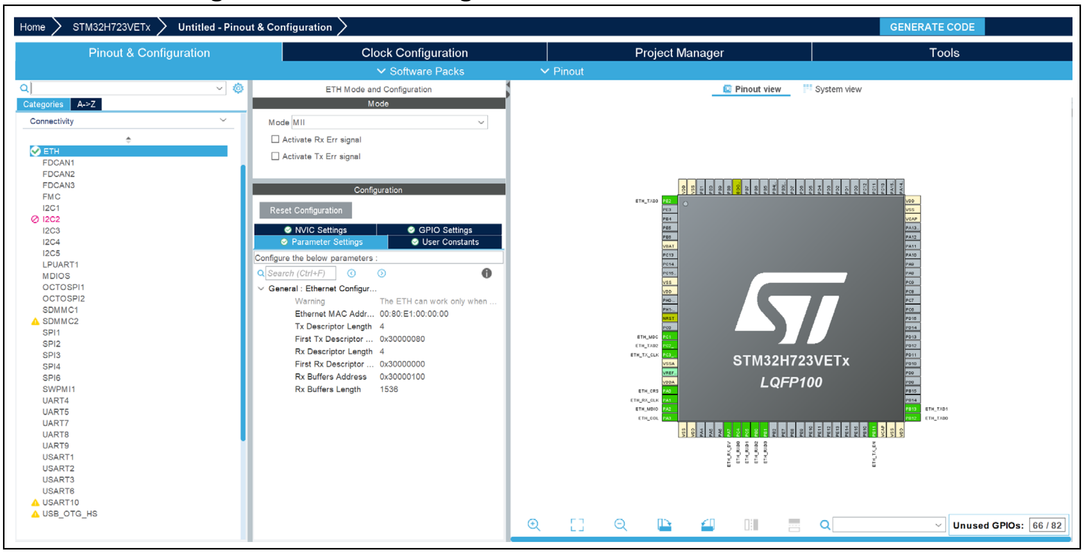


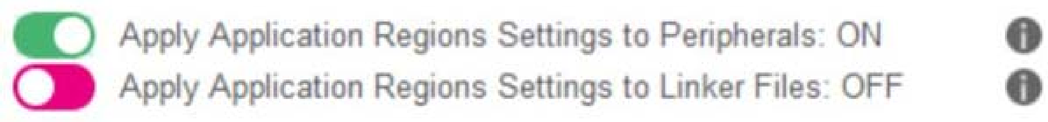
Figure 402. Defined memory regions under the linker file

```
/*-Sizes-*/
define symbol __ICFEDIT_intvec_start__ = 0x08000000;

define symbol __ICFEDIT_size_cstack__ = 0x400;
define symbol __ICFEDIT_size_heap__   = 0x200;


/*-Start of  symbols-Auto-generated By STM32CubeMX*/
define symbol __ICFEDIT_region_DTCMRAM_start__           = 0x20000000;
define symbol __ICFEDIT_region_DTCMRAM_end__             = 0x2001FFFF;
define symbol __ICFEDIT_region_RAM_D3_start__            = 0x38000000;
define symbol __ICFEDIT_region_RAM_D3_end__              = 0x38003FFF;
define symbol __ICFEDIT_region_ITCMRAM_start__           = 0x00000000;
define symbol __ICFEDIT_region_ITCMRAM_end__             = 0xFFFF;
define symbol __ICFEDIT_region_RxDescripSection_start__ = 0x30000000;
define symbol __ICFEDIT_region_RxDescripSection_end__   = 0x3000007F;
define symbol __ICFEDIT_region_TxDescripSection_start__ = 0x30000080;
define symbol __ICFEDIT_region_TxDescripSection_end__   = 0x300000FF;
define symbol __ICFEDIT_region_Rx_PoolSection_start__   = 0x30000100;
define symbol __ICFEDIT_region_Rx_PoolSection_end__     = 0x300006FF;

/*-End of MX Symbols*/

/*-Symbols-*/
define symbol __ICFEDIT_region_RAM_start__               = 0x24000000;
define symbol __ICFEDIT_region_RAM_end__                 = 0x2401FFFF;
define symbol __ICFEDIT_region_FLASH_start__             = 0x08000000;
define symbol __ICFEDIT_region_FLASH_end__               = 0x807FFFF;
/**** End of ICF editor section. ###ICF###*/

define memory mem with size = 4G;

/*-MEMORY Regions-*/
define region RAM_region             = mem:[from __ICFEDIT_region_RAM_start__             to __ICFEDIT_region_RAM_end__];
define region FLASH_region           = mem:[from __ICFEDIT_region_FLASH_start__           to __ICFEDIT_region_FLASH_end__];

/*-Start of  Regions- Auto-generated By STM32CubeMX*/
define region DTCMRAM_region         = mem:[from __ICFEDIT_region_DTCMRAM_start__         to __ICFEDIT_region_DTCMRAM_end__];
define region RAM_D3_region          = mem:[from __ICFEDIT_region_RAM_D3_start__          to __ICFEDIT_region_RAM_D3_end__];
define region ITCMRAM_region         = mem:[from __ICFEDIT_region_ITCMRAM_start__         to __ICFEDIT_region_ITCMRAM_end__];
define region RxDescripSection_region = mem:[from __ICFEDIT_region_RxDescripSection_start__ to __ICFEDIT_region_RxDescripSection_end__];
define region TxDescripSection_region = mem:[from __ICFEDIT_region_TxDescripSection_start__ to __ICFEDIT_region_TxDescripSection_end__];
define region Rx_PoolSection_region   = mem:[from __ICFEDIT_region_Rx_PoolSection_start__   to __ICFEDIT_region_Rx_PoolSection_end__];

/*-End of MX Regions-*/

/*-Blocks-*/
define block CSTACK with alignment = 8, size = __ICFEDIT_size_cstack__ {};
define block HEAP   with alignment = 8, size = __ICFEDIT_size_heap__  {};

/*-Initialization strategies-*/
initialize by copy { readwrite };
do not initialize  { section .noinit };

/*-Sections placements-*/

/*-Start of  Sections- Auto-generated By STM32CubeMX*/
place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };

place in DTCMRAM_region  { section DTCMRAM_section };
place in RAM_D3_region  { section RAM_D3_section };
place in ITCMRAM_region  { section ITCMRAM_section };
place in RxDescripSection_region  { section RxDescripSection_section };
place in TxDescripSection_region  { section TxDescripSection_section };
place in Rx_PoolSection_region  { section Rx_PoolSection_section };

/*-end of MX Sections-*/

/*-user Sections-*/
place in FLASH_region  { readonly };
place in RAM_region  { readwrite, block HEAP, block CSTACK };
```

5.5.4 STM32WBxx

Feature: MMT usage, pinout, and configuration user interface

When the first toggle button is ON, Cortex-M33 is under MMT control: its modes and parameters become read-only (see *Figure 403*).



The user must select the core and the STM32Cube firmware from a list. It is possible to choose any STM32Cube firmware version (see *Figure 404*).

The list proposed to user contains only the firmwares found in STM32Cube_FW_WB_Vx/Projects / STM32_Copro_Wireless_Binaries/STM32WBxx (all .bin files). Firmware Update Service (FUS) and SafeBoot firmware are not proposed, so they are not in the MMT list.

This example is based on an STM32WB5x MCU, so the list must contain only stm32wb5x_x binaries. The button “Refresh” is used to refresh the binaries list version in the repository of STM32Cube firmware (see *Figure 405*).

Figure 403. MMT usage

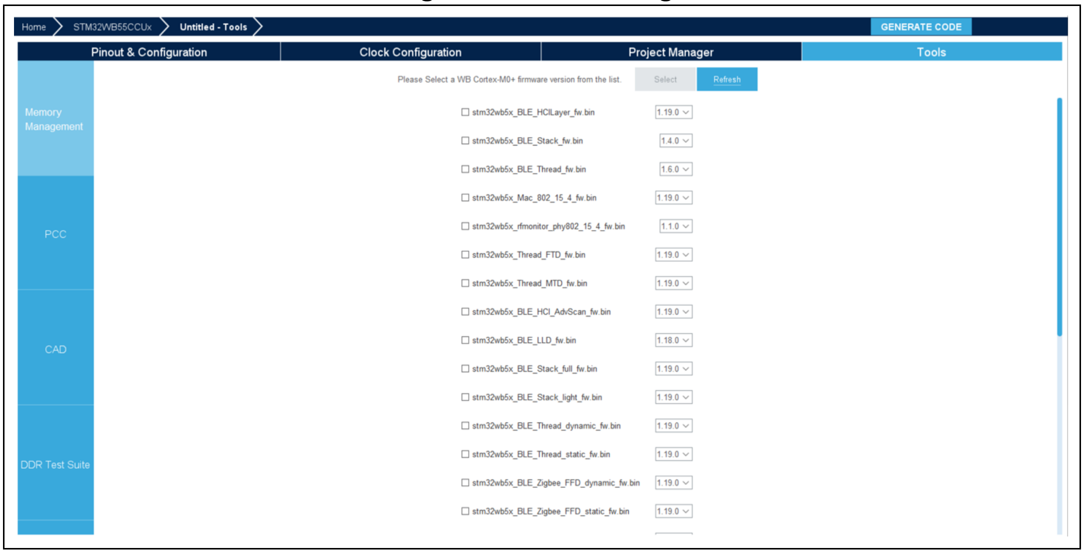


Figure 404. Firmware version

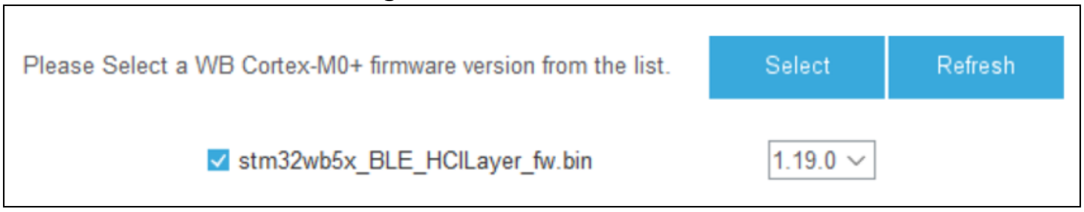
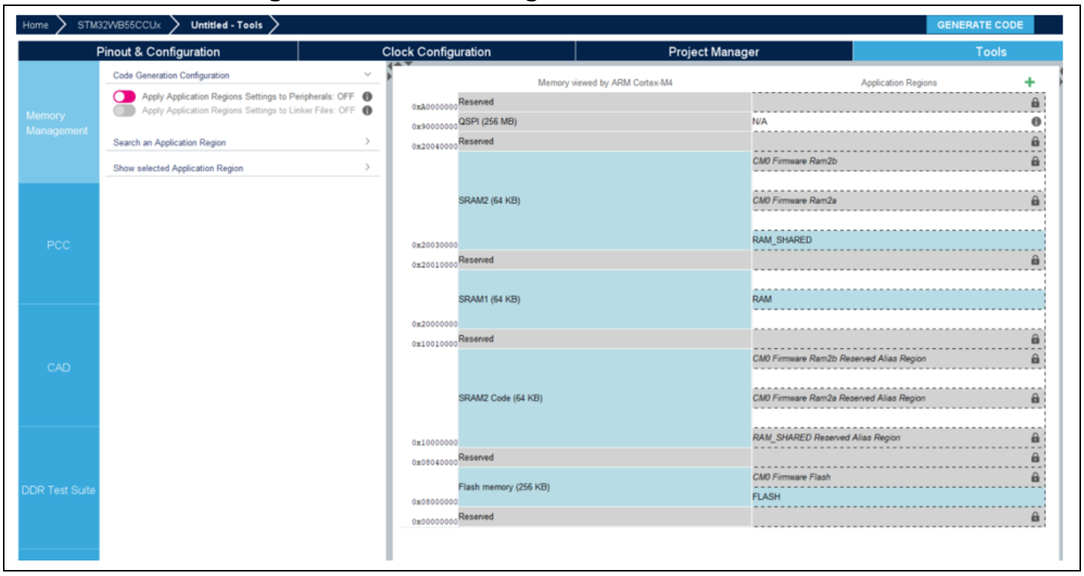


Figure 405. MMT configuration for STM32WB5x



After selecting the binary firmware, the MMT view is displayed and the reserved regions of Cortex M0+ are created.

The middle panel represents the memory, split into two columns: the left one is the memory seen by the core(s) Cortex-M4, the right one the memory set-up for the application.

For the new project created under STM32CubeMX the tool creates the default application region to generate a valid project.

5.5.5 STM32H7 Dual-core without Trust Zone activated

Feature: MMT usage, pinout, and user interface configuration

When the first toggle button is ON, Cortex-M7_BOOT (MPU) and Cortex-M7_APPLI (MPU) are under MMT control: their modes and parameters become read-only.

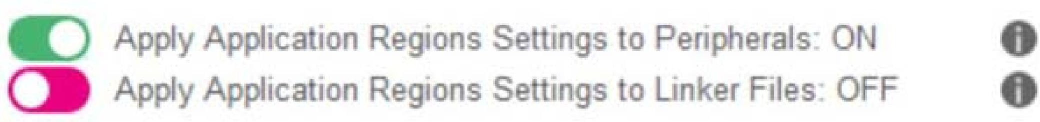


Figure 406. Cortex_M7 mode and configuration

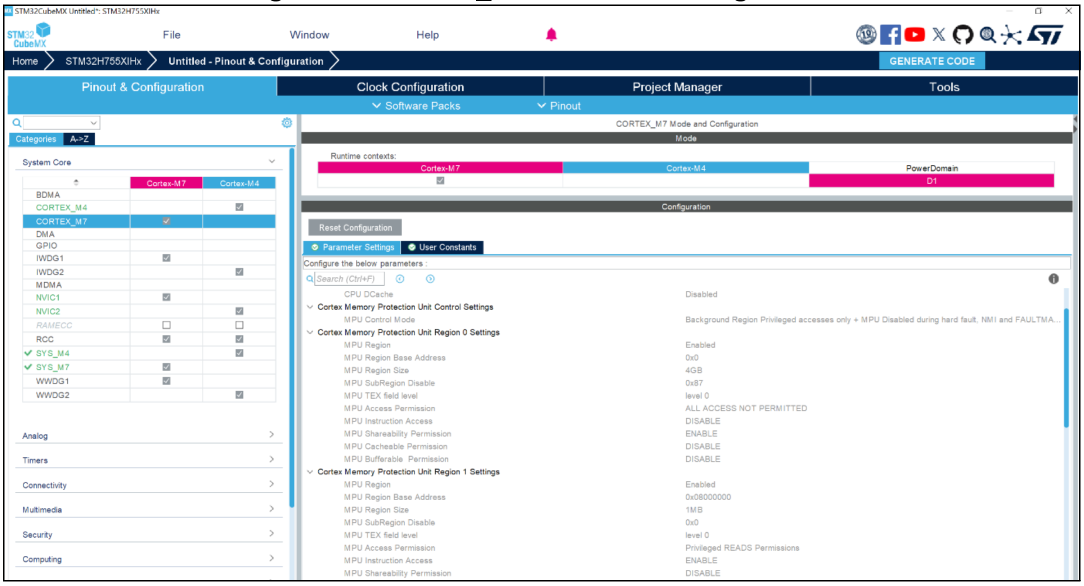
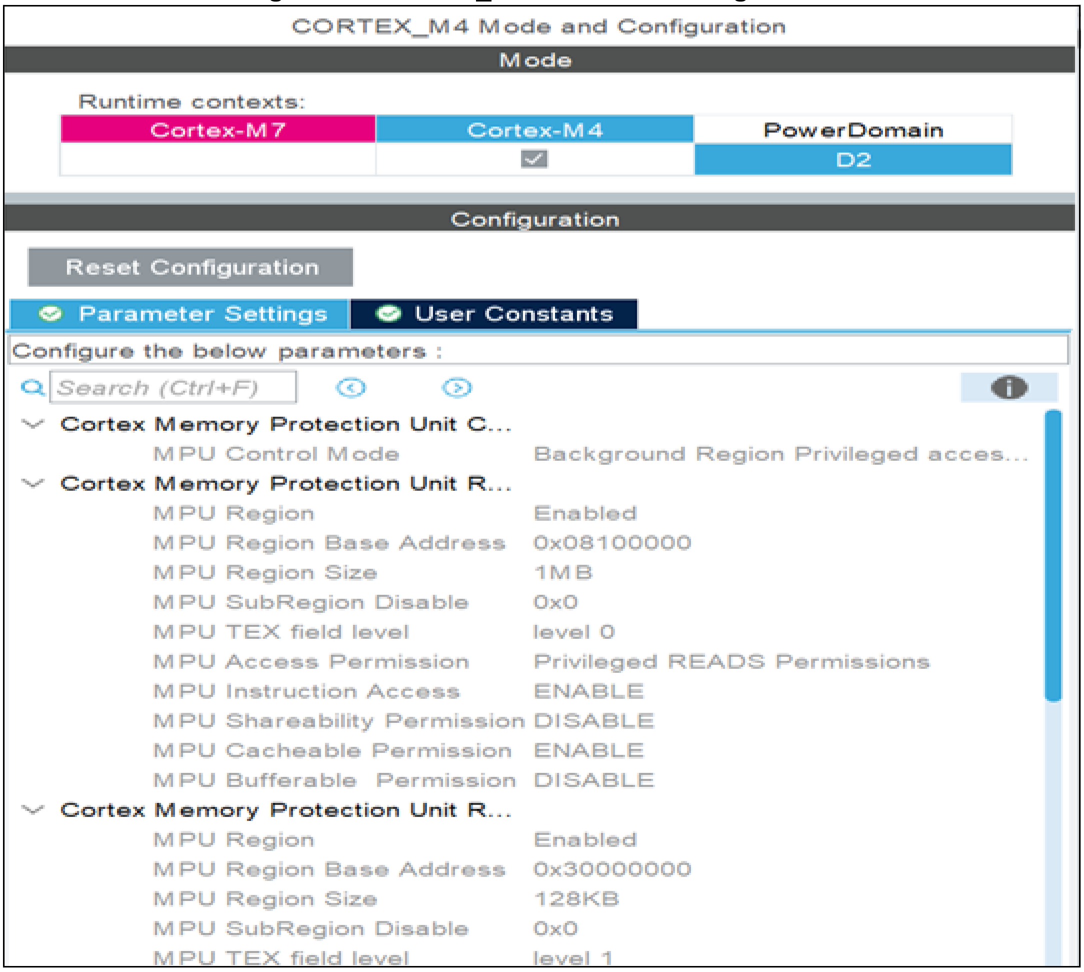
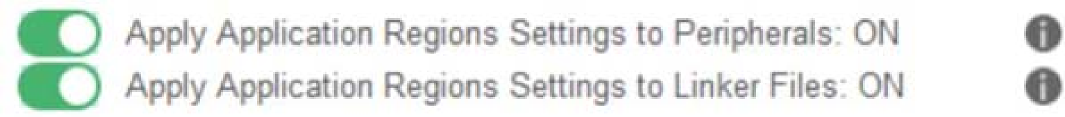


Figure 407. Cortex_M4 mode and configuration



Feature: MMT usage and linker script

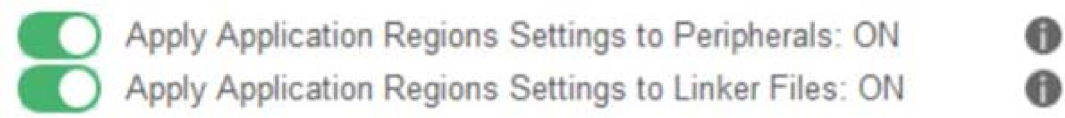
When the two radio buttons are activated, the memory management parameters are available, and the linker file content is generated according to the configuration of application regions.



There are two possible configurations of the application regions for the code generation:

- First configuration:

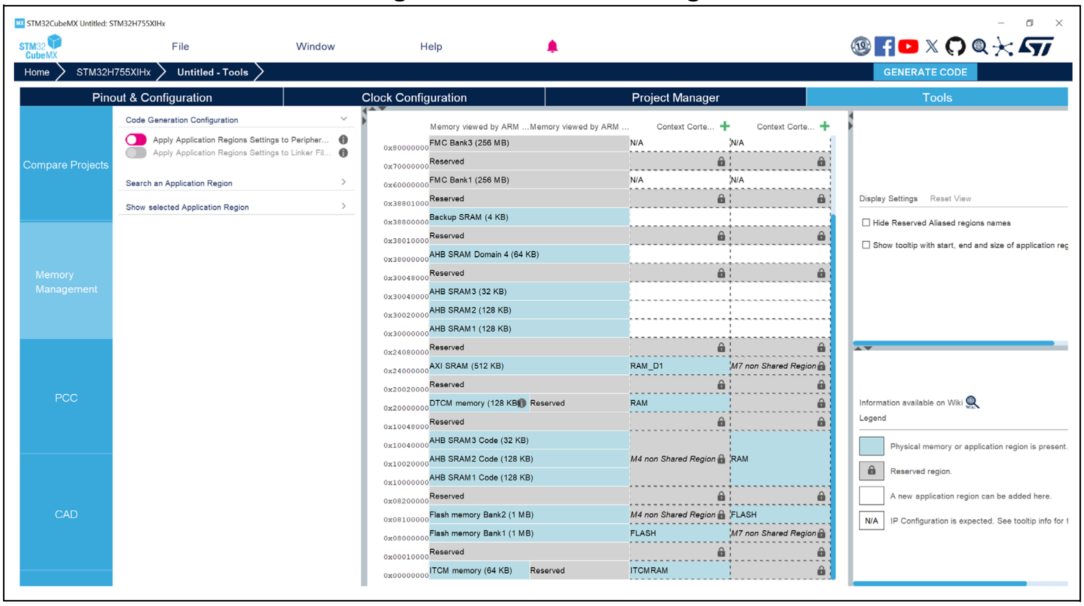
- Second configuration:



The Cortex-M7 and Cortex-M4 contexts are managed by the MMT. Each context has its own application region (AppReg0 and AppReg1, respectively).

User interface

Figure 408. Default settings



The middle panel represents the memory, split into three columns: the left one is the memory seen by the cores (CM7 and CM4), the middle one the memory set-up for the application in Context Cortex-M7, the right one the memory set-up for the application in the Context Cortex-M4.

For the new project created under STM32CubeMX, the tool creates the default application region to generate a valid project.

Region information

Clicking on a particular region in the Application Regions column shows the associated details on the left hand side.

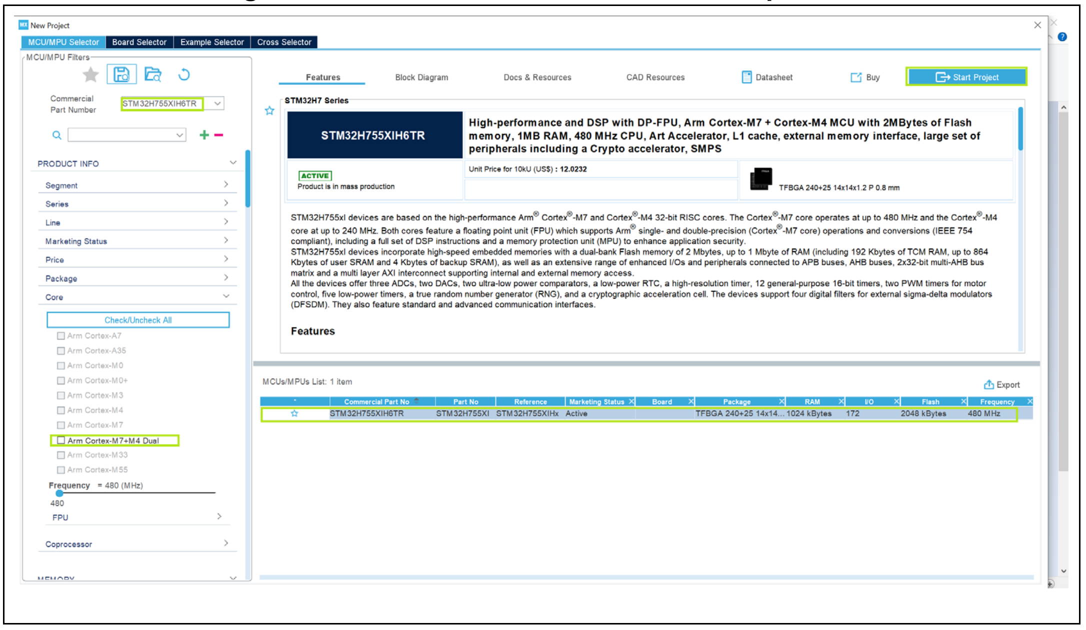
STM32CubeMX automatically adds a 4 Gbytes region for the system core, even if you are not planning to use the MMT.

An example of MMT configuration of the OPENAMP Middleware on the STM32H755XIH6TR MCU

Below are the steps for configuring the MMT with OPENAMP activated on the STM32H755XIH6TR MCU.

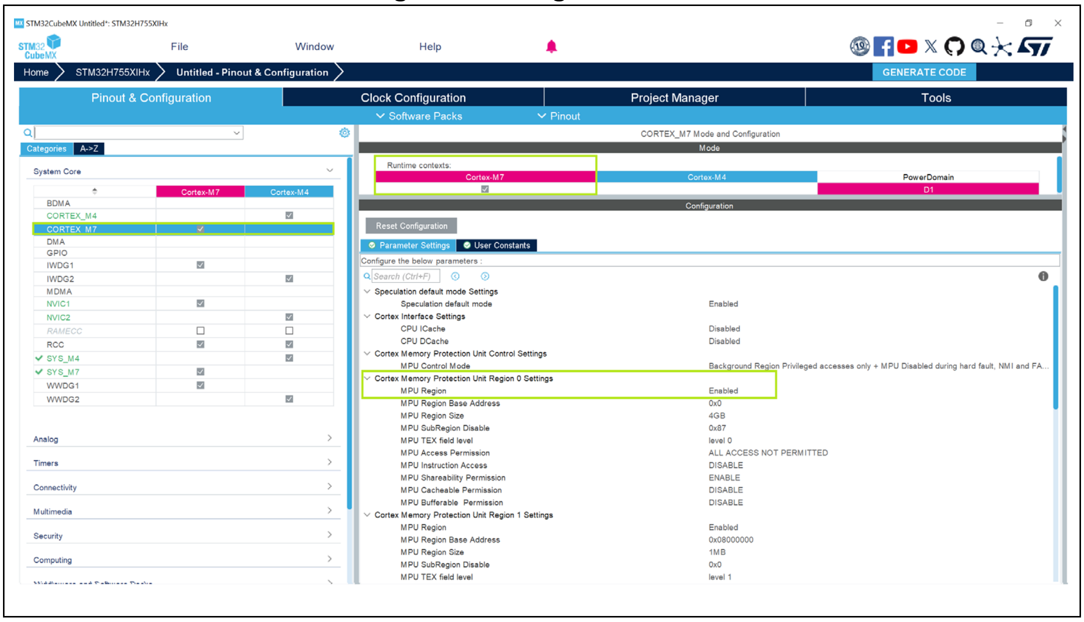
1. Choose a supported MCU.

Figure 409. Choose an STM32H7 dual-core product



2. Click on the Start Project button, then choose Yes on the “Memory Protection Unit for Cortex-M7” dialog box.

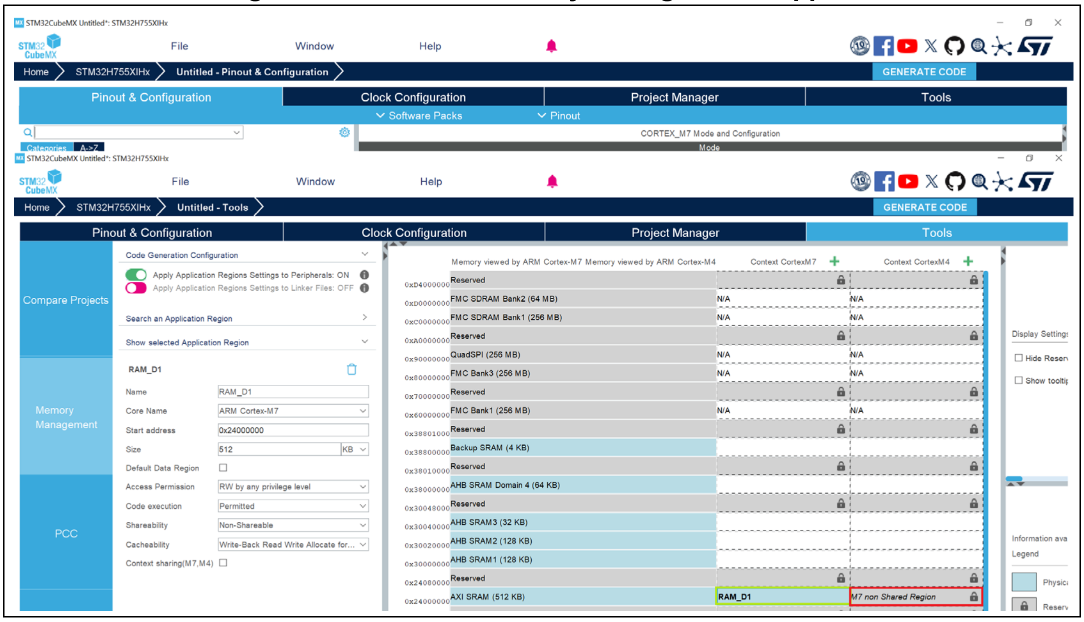
Figure 410. Region 0 added



Note: STM32CubeMX applies the default configuration, then adds a 4 Gbytes region called “Region 0” under the Cortex_M7 parameters. The new parameters can be checked using the Pinout and Configuration tab.

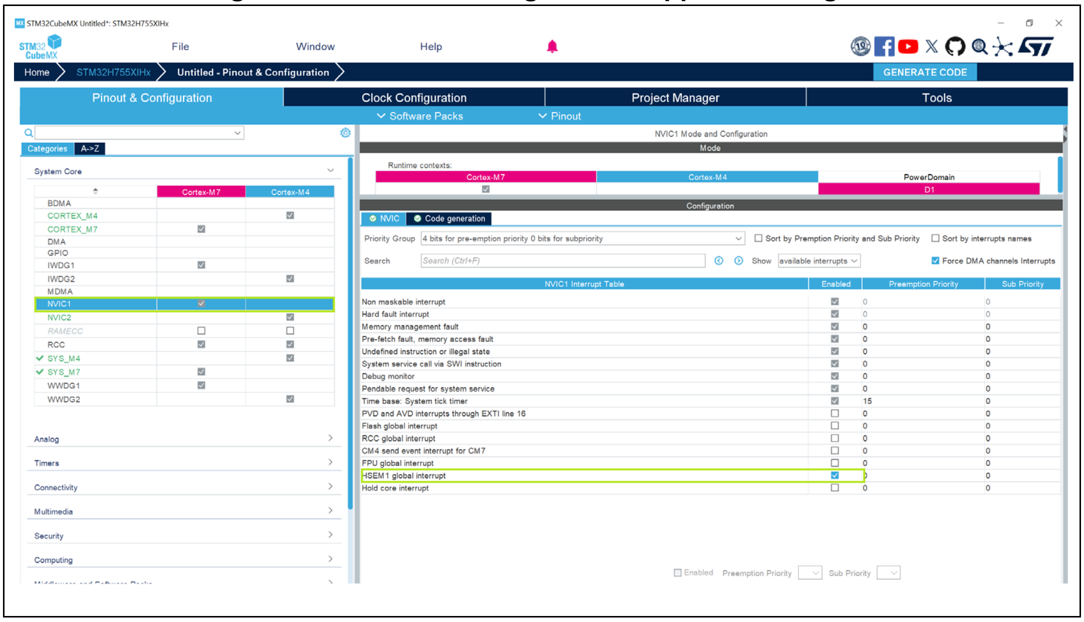
3. Select “Tools” in the toolbar
   – Choose Memory Management.
   – Activate the Memory Management Tool support by clicking the button “Apply Application Regions Settings to Peripherals”.

Figure 411. Activate Memory Management support



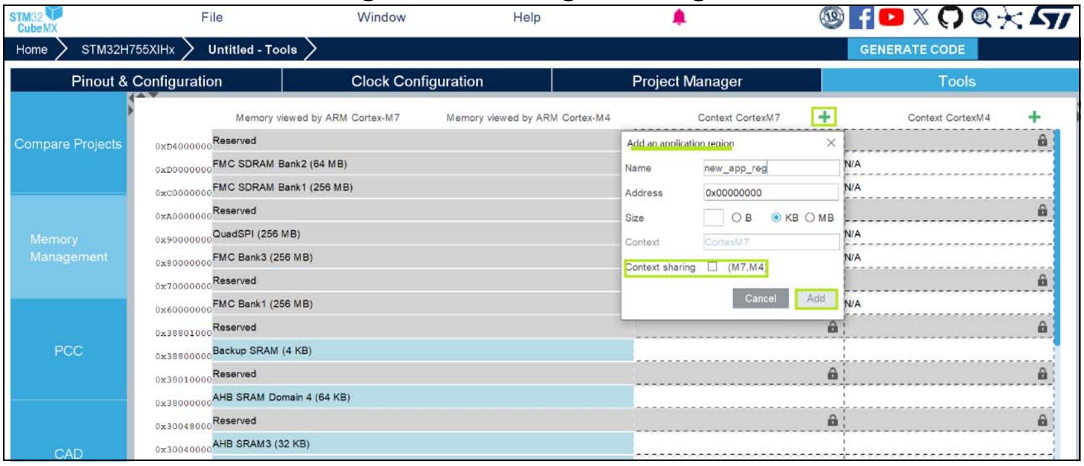
The default application regions are in exclusive mode (context sharing is unselected). A reserved region in the other context is created and mentioned as “Mx non-shared region”.

Figure 412. Default setting for new application region

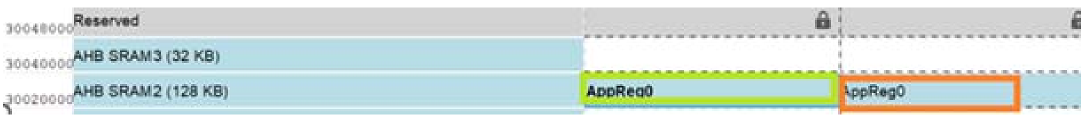


4. Add a new region by pressing the “+” button that appears in the white space when hovering with the mouse.

Figure 413. Adding a new region



5. Select “Context sharing (M7, M4)”, automatically another region is created with the same name, start address, and size.



6. Select the Project Manager tab.
7. Give a name to the project and press the Generate Code button.

8. OPENAMP activation
   – Configure the NVIC1 and 2 and select their related HSEM global interrupts.
   – Activate the Middleware OPENAMP_M4.
   – MMT creates two application regions for each core. The Master regions are defined by attribute mode.

Figure 414. Configure NVIC1 and NVIC2, and select their HSEM global interrupt

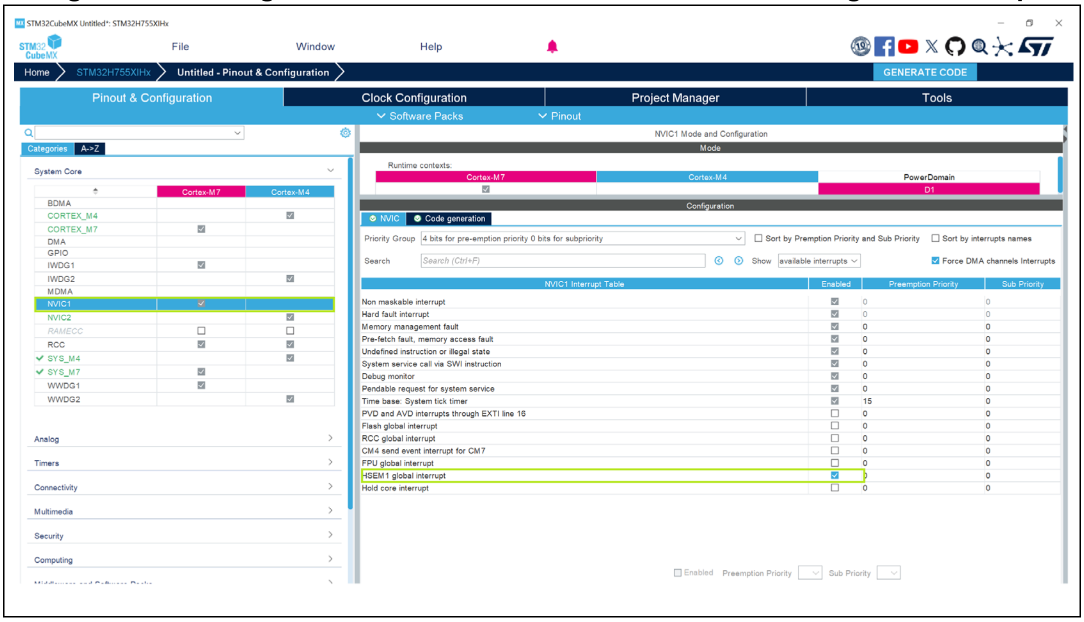


Figure 415. OPENAMP_M7 parameters settings

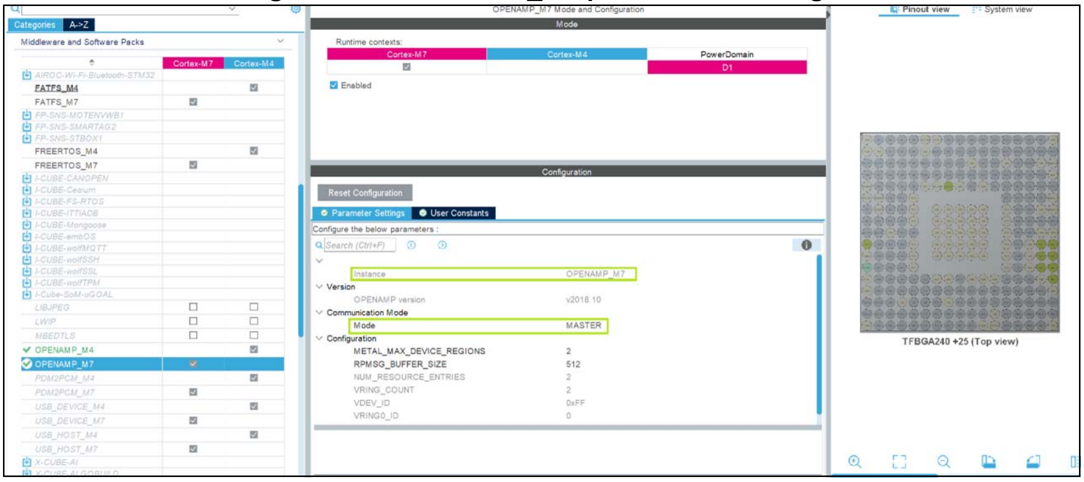


Figure 416. OPENAMP_M4 parameters settings

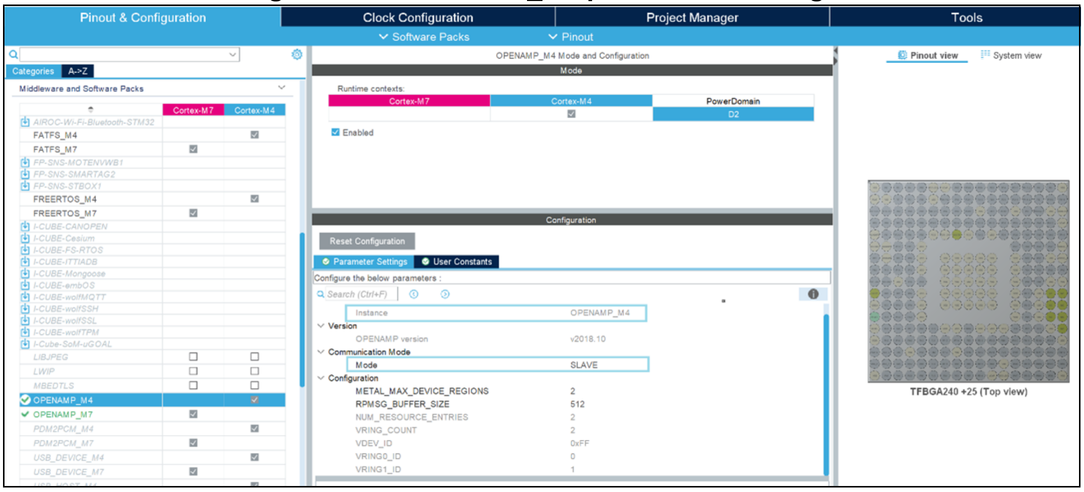
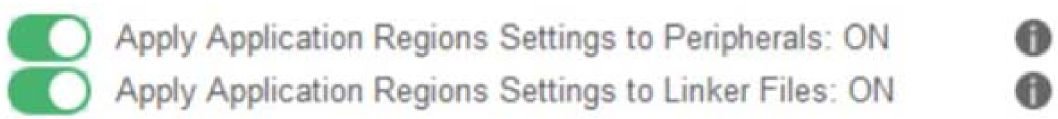


Figure 417. Reserved memory regions for OPENAMP using MMT

```
/******************************************************************************/
//*###ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\cortex_v1_0.xml" *//*-Specials-*/
/******************************************************************************/
/*-Sizes-*/
define symbol __ICFEDIT_intvec_start__ = 0x08100000;
define symbol __ICFEDIT_size_cstack__ = 0x400;
define symbol __ICFEDIT_size_heap__  = 0x200;
/*-Start of  symbols-Auto-generated By STM32CubeMX*/
define symbol __ICFEDIT_region_OPENAMP_RSC_TAB_start__ =0x38000000;
define symbol __ICFEDIT_region_OPENAMP_RSC_TAB_end__ =0x380003ff;
define symbol __ICFEDIT_region_OPENAMP_start__ =0x38000400;
define symbol __ICFEDIT_region_OPENAMP_end__ =0x3800ffff;
/*-End of MX Symbols*/
/*-Symbols-*/
define symbol __ICFEDIT_region_RAM_start__             = 0x10000000;
define symbol __ICFEDIT_region_RAM_end__               = 0x10047fff;
define symbol __ICFEDIT_region_FLASH_start__           = 0x08100000;
define symbol __ICFEDIT_region_FLASH_end__             = 0x81fffff;
/**** End of ICF editor section. ###ICF###*/
define memory mem with size = 4G;
/*-MEMORY Regions-*/
define region RAM_region              = mem:[from __ICFEDIT_region_RAM_start__          to __ICFEDIT_region_RAM_end__];
define region FLASH_region            = mem:[from __ICFEDIT_region_FLASH_start__        to __ICFEDIT_region_FLASH_end__];
/*-Start of  Regions- Auto-generated By STM32CubeMX*/
define region OPENAMP_RSC_TAB_region      = mem:[from __ICFEDIT_region_OPENAMP_RSC_TAB_start__      to __ICFEDIT_region_OPENAMP_RSC_TAB_end__];
define region OPENAMP_region      = mem:[from __ICFEDIT_region_OPENAMP_start__        to __ICFEDIT_region_OPENAMP_end__];
/*-End of MX Regions-*/
/*-Blocks-*/
define block CSTACK with alignment = 8, size = __ICFEDIT_size_cstack__ {};
define block HEAP   with alignment = 8, size = __ICFEDIT_size_heap__  {};
/*-Initialization strategies-*/
initialize by copy { readwrite };
do not initialize  { section .noinit };
/*-Sections placements-*/
/*-Start of  Sections- Auto-generated By STM32CubeMX*/
place in OPENAMP_RSC_TAB_region  { section OPENAMP_RSC_TAB_section };
place in OPENAMP_region  { section OPENAMP_section };
define symbol __OPENAMP_region_start__ = __ICFEDIT_region_OPENAMP_start__;
define symbol __OPENAMP_region_size__ = __ICFEDIT_region_OPENAMP_end__;
export symbol __OPENAMP_region_start__;
export symbol __OPENAMP_region_size__;
place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };

/*-end of MX Sections-*/
/*-user Sections-*/
place in FLASH_region  { readonly };
place in RAM_region  { readwrite, block HEAP, block CSTACK };
```

9. Press the Generate Code button to generate the code for both applications.

Apply Application Regions settings to linker files



When the second radio button is on, the linker scripts for the CM7 and CM4 projects are generated considering the configuration.

Figure 418. Linker files update (stm32h755xxx_flash_cm4.icf)

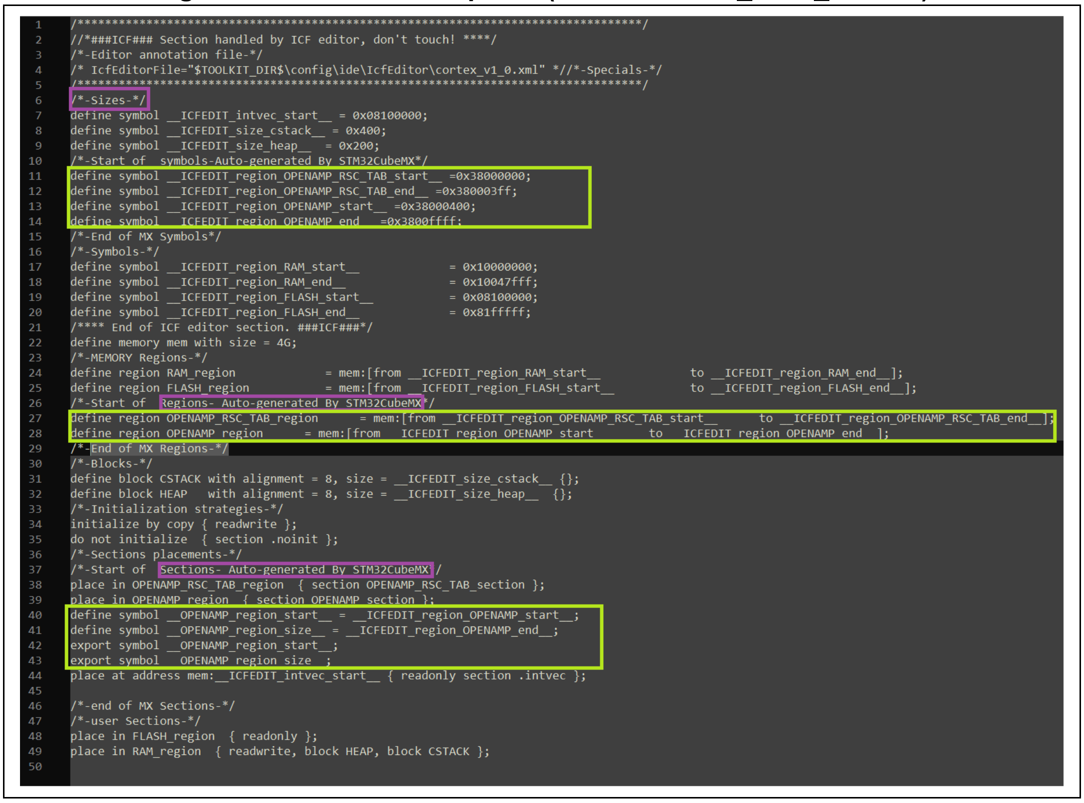
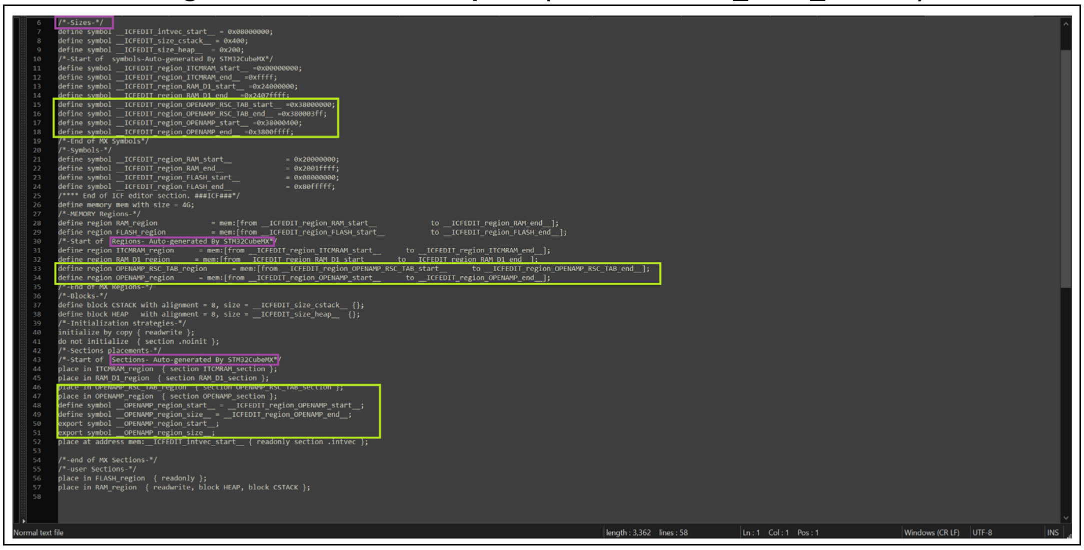


Figure 419. Linker files update(stm32h755xxx_flash_cm7.icf)



The middleware can be enabled or disabled:

- If disabled, it automatically chooses the configured memory along with the associated driver and sets the execution memory location in the linker file.
- If enabled, the two regions and corresponding 'export symbol' must be added in the generated linker file.

After the code generation, navigate to the generated folder to check the linker file updates.

Example of MMT configuration of the ETH on STM32H755XIH6TR MCU

1. Activate the IP ETH: MMT creates three application regions for each core. To change the start address and the size of each region, update the ETH parameters.

Figure 420. Configuration of ETH IP

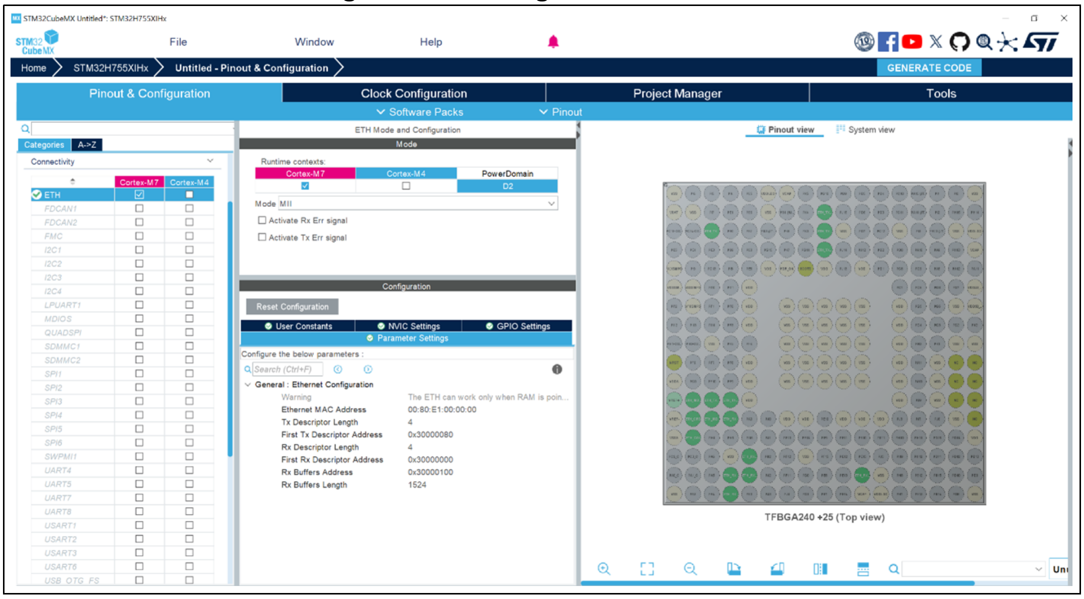
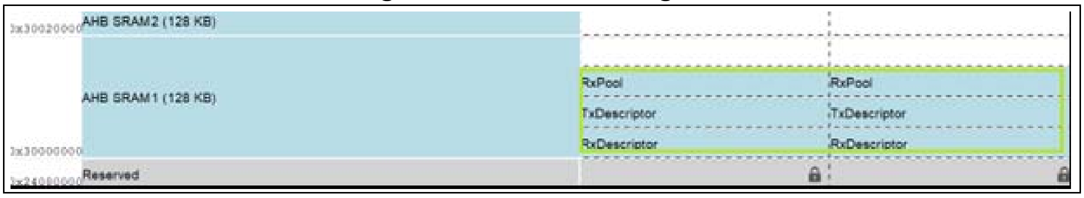
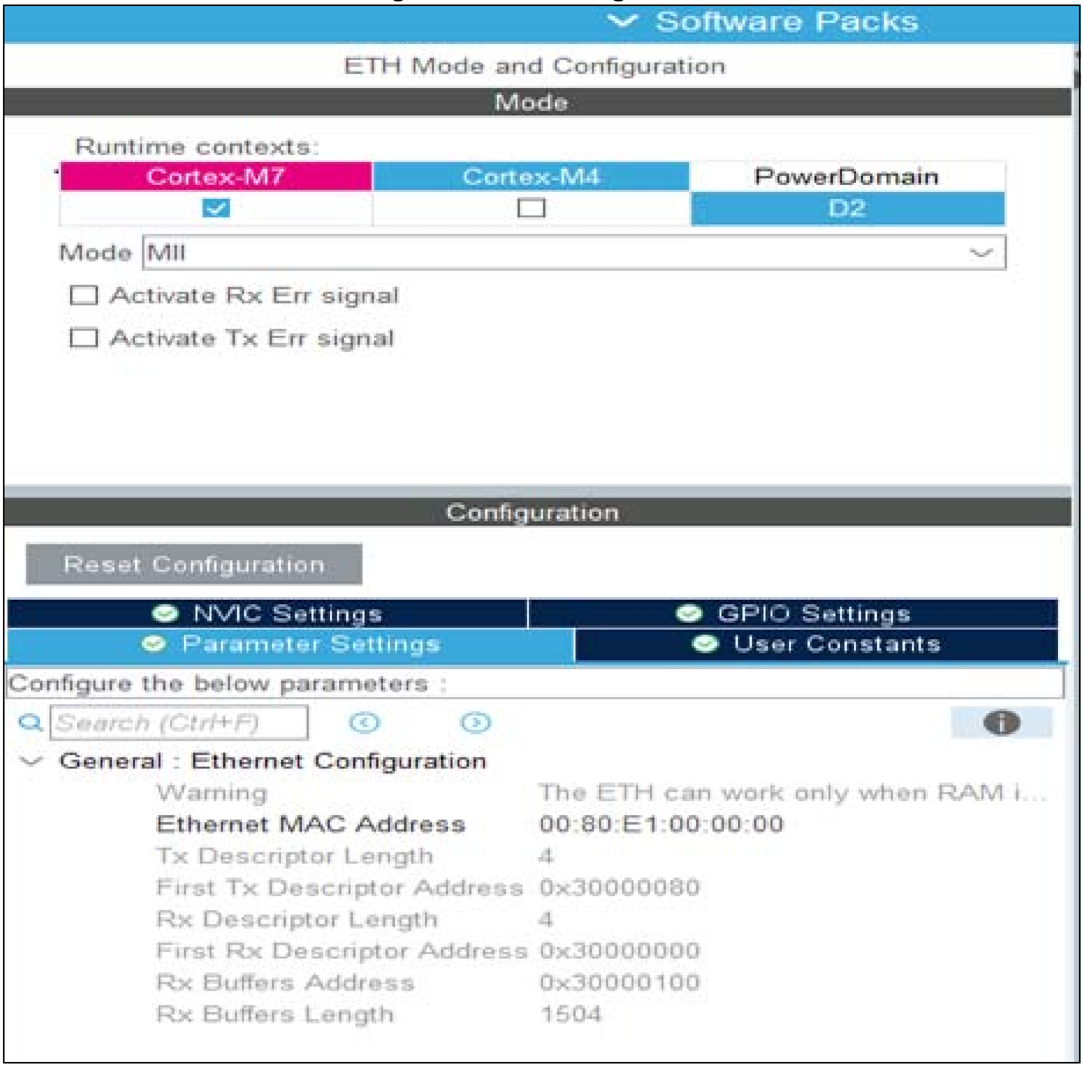


Figure 421. ETH MMT regions



2. Press the radio button “Apply Application region Settings to Peripherals ON”, ETH will be partially under MMT control.
3. Press the Generate Code button to generate code for both applications.

Figure 422. IP configuration



Apply Application Regions settings to linker files:

4. When this button is on, the linker scripts for the CM7 project and CM4 project are generated, considering the configuration.
5. After the code generation, navigate to the generated folder:
   – Under the CM7 Project, open the linker definition file.
   – Under the Memories definition you can see the defined memories with their start address and length, according to the configuration made in STM32CubeMX.

Figure 423. Defined memories under the linker file (Cortex-M7)

```
/*-Sizes-*/
define symbol __ICFEDIT_intvec_start__ = 0x08000000;

define symbol __ICFEDIT_size_cstack__ = 0x400;
define symbol __ICFEDIT_size_heap__  = 0x200;


/*-Start of  symbols-Auto-generated By STM32CubeMX*/
define symbol __ICFEDIT_region_ITCMRAM_start__           = 0x00000000;
define symbol __ICFEDIT_region_ITCMRAM_end__             = 0xFFFF;
define symbol __ICFEDIT_region_RAM_D1_start__            = 0x24000000;
define symbol __ICFEDIT_region_RAM_D1_end__              = 0x2407FFFF;
define symbol __ICFEDIT_region_RxDescripSection_start__ = 0x30000000;
define symbol __ICFEDIT_region_RxDescripSection_end__   = 0x3000007F;
define symbol __ICFEDIT_region_TxDescripSection_start__ = 0x30000080;
define symbol __ICFEDIT_region_TxDescripSection_end__   = 0x300000FF;
define symbol __ICFEDIT_region_Rx_PoolSection_start__   = 0x30000100;
define symbol __ICFEDIT_region_Rx_PoolSection_end__     = 0x300006FF;

/*-End of MX Symbols*/

/*-Symbols-*/
define symbol __ICFEDIT_region_RAM_start__               = 0x20000000;
define symbol __ICFEDIT_region_RAM_end__                 = 0x2001FFFF;
define symbol __ICFEDIT_region_FLASH_start__             = 0x08000000;
define symbol __ICFEDIT_region_FLASH_end__               = 0x80FFFFF;
/**** End of ICF editor section. ###ICF###*/

define memory mem with size = 4G;

/*-MEMORY Regions-*/
define region RAM_region              = mem:[from __ICFEDIT_region_RAM_start__              to __ICFEDIT_region_RAM_end__];
define region FLASH_region            = mem:[from __ICFEDIT_region_FLASH_start__            to __ICFEDIT_region_FLASH_end__];

/*-Start of  Regions- Auto-generated By STM32CubeMX*/
define region ITCMRAM_region          = mem:[from __ICFEDIT_region_ITCMRAM_start__          to __ICFEDIT_region_ITCMRAM_end__];
define region RAM_D1_region           = mem:[from __ICFEDIT_region_RAM_D1_start__           to __ICFEDIT_region_RAM_D1_end__];
define region RxDescripSection_region = mem:[from __ICFEDIT_region_RxDescripSection_start__ to __ICFEDIT_region_RxDescripSection_end__];
define region TxDescripSection_region = mem:[from __ICFEDIT_region_TxDescripSection_start__ to __ICFEDIT_region_TxDescripSection_end__];
define region Rx_PoolSection_region   = mem:[from __ICFEDIT_region_Rx_PoolSection_start__   to __ICFEDIT_region_Rx_PoolSection_end__];

/*-End of MX Regions-*/

/*-Blocks-*/
define block CSTACK with alignment = 8, size = __ICFEDIT_size_cstack__ {};
define block HEAP   with alignment = 8, size = __ICFEDIT_size_heap__  {};

/*-Initialization strategies-*/
initialize by copy { readwrite };
do not initialize  { section .noinit };

/*-Sections placements-*/

/*-Start of  Sections- Auto-generated By STM32CubeMX*/
place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };

place in ITCMRAM_region  { section ITCMRAM_section };
place in RAM_D1_region  { section RAM_D1_section };
place in RxDescripSection_region  { section RxDescripSection_section };
place in TxDescripSection_region  { section TxDescripSection_section };
place in Rx_PoolSection_region  { section Rx_PoolSection_section };

/*-end of MX Sections-*/

/*-user Sections-*/
place in FLASH_region  { readonly };
place in RAM_region  { readwrite, block HEAP, block CSTACK };
```

Figure 424. Defined memories under the linker file (Cortex-M4)

```
/*-Sizes-*/
define symbol __ICFEDIT_intvec_start__ = 0x08100000;

define symbol __ICFEDIT_size_cstack__ = 0x400;
define symbol __ICFEDIT_size_heap__  = 0x200;


/*-Start of  symbols-Auto-generated By STM32CubeMX*/
define symbol __ICFEDIT_region_RxDescripSection_start__ = 0x30000000;
define symbol __ICFEDIT_region_RxDescripSection_end__   = 0x3000007F;
define symbol __ICFEDIT_region_TxDescripSection_start__ = 0x30000080;
define symbol __ICFEDIT_region_TxDescripSection_end__   = 0x300000FF;
define symbol __ICFEDIT_region_Rx_PoolSection_start__   = 0x30000100;
define symbol __ICFEDIT_region_Rx_PoolSection_end__     = 0x300006FF;

/*-End of MX Symbols*/

/*-Symbols-*/
define symbol __ICFEDIT_region_RAM_start__              = 0x10000000;
define symbol __ICFEDIT_region_RAM_end__                = 0x10047FFF;
define symbol __ICFEDIT_region_FLASH_start__            = 0x08100000;
define symbol __ICFEDIT_region_FLASH_end__              = 0x81FFFFF;
/**** End of ICF editor section. ###ICF###*/

define memory mem with size = 4G;

/*-MEMORY Regions-*/
define region RAM_region            = mem:[from __ICFEDIT_region_RAM_start__            to __ICFEDIT_region_RAM_end__];
define region FLASH_region          = mem:[from __ICFEDIT_region_FLASH_start__          to __ICFEDIT_region_FLASH_end__];

/*-Start of  Regions- Auto-generated By STM32CubeMX*/
define region RxDescripSection_region = mem:[from __ICFEDIT_region_RxDescripSection_start__  to __ICFEDIT_region_RxDescripSection_end__];
define region TxDescripSection_region = mem:[from __ICFEDIT_region_TxDescripSection_start__  to __ICFEDIT_region_TxDescripSection_end__];
define region Rx_PoolSection_region   = mem:[from __ICFEDIT_region_Rx_PoolSection_start__    to __ICFEDIT_region_Rx_PoolSection_end__];

/*-End of MX Regions-*/

/*-Blocks-*/
define block CSTACK with alignment = 8, size = __ICFEDIT_size_cstack__ {};
define block HEAP   with alignment = 8, size = __ICFEDIT_size_heap__  {};

/*-Initialization strategies-*/
initialize by copy { readwrite };
do not initialize  { section .noinit };

/*-Sections placements-*/

/*-Start of  Sections- Auto-generated By STM32CubeMX*/
place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };

place in RxDescripSection_region  { section RxDescripSection_section };
place in TxDescripSection_region  { section TxDescripSection_section };
place in Rx_PoolSection_region  { section Rx_PoolSection_section };

/*-end of MX Sections-*/

/*-user Sections-*/
place in FLASH_region  { readonly };
place in RAM_region  { readwrite, block HEAP, block CSTACK };
```

5.5.6 STM32H7RS

Feature: MMT usage, pinout, and configuration user interface

When the first toggle button is ON, Cortex-M7_BOOT (MPU) and Cortex-M7_APPLI (MPU) are under MMT control: their modes and parameters become read-only.

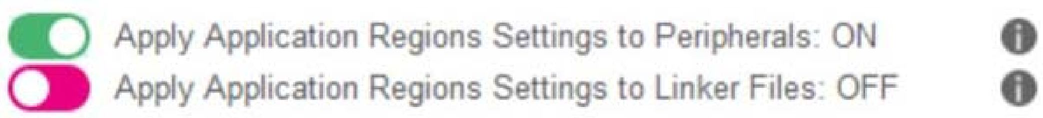
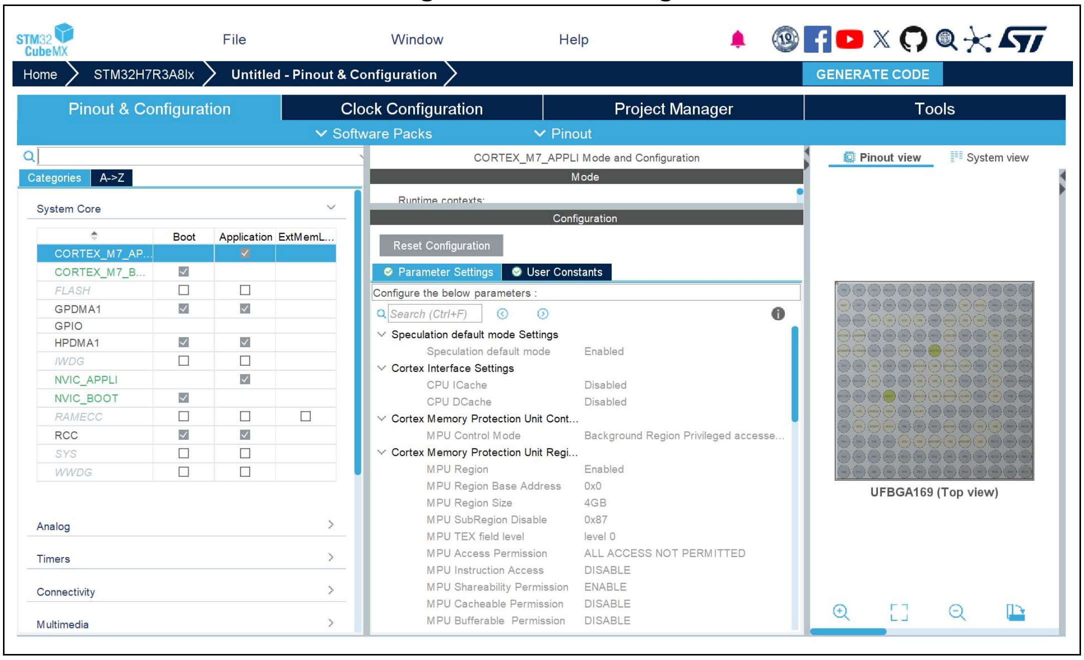
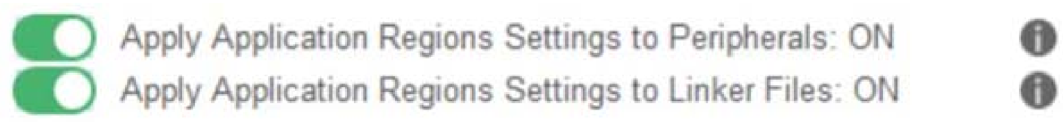


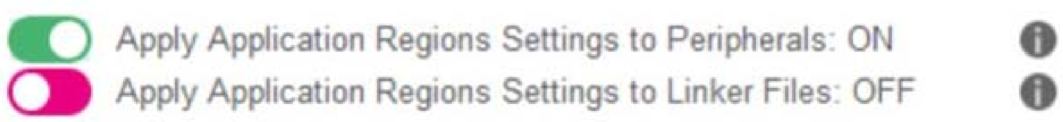
Figure 425. MMT usage

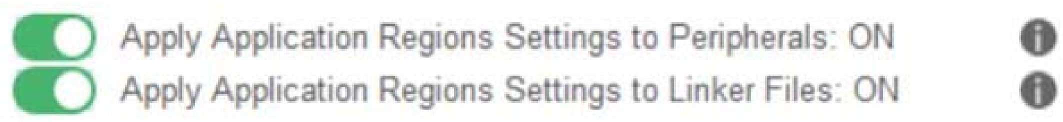


Feature: MMT usage and linker script



Linker files content is generated according to the configuration of application regions.

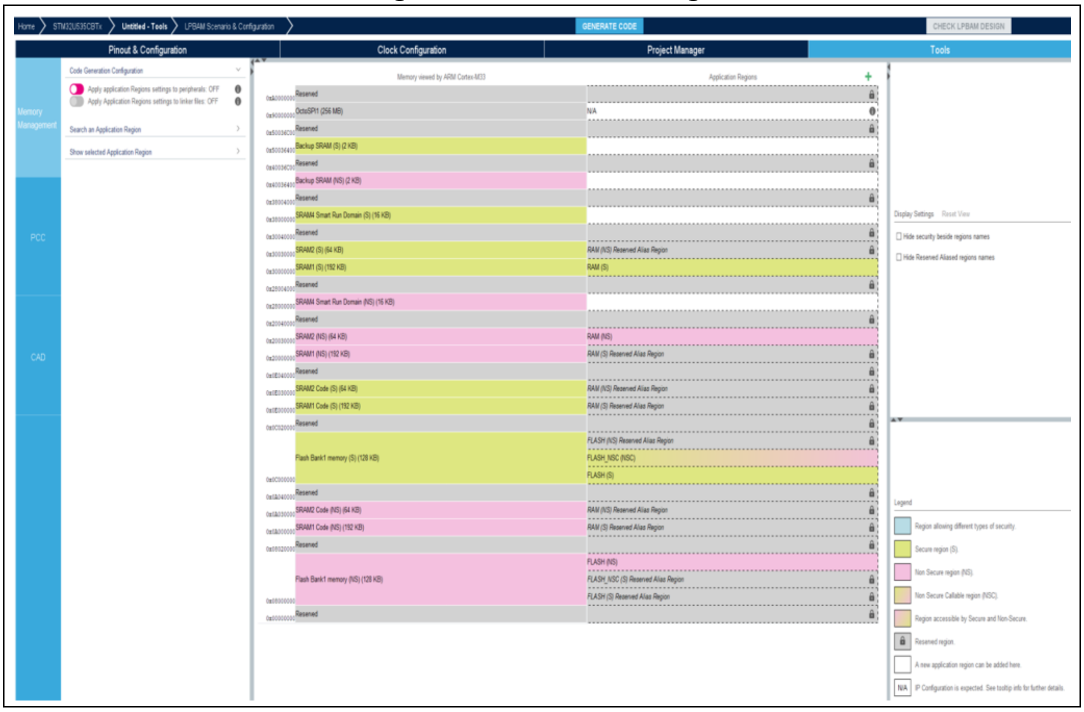




Only “Boot” and “Appli” contexts are managed by the MMT. Each context has its own application region (AppReg0 and AppReg1, respectively).

User interface

Figure 426. Default settings



The middle panel represents the memory, split into three columns: the left one is the memory seen by the core(s), the middle one the memory set-up for the application in Context Boot, the right one the memory set-up for the application in the Context Appli.

For the new project created under STM32CubeMX, the tool creates the default application region to generate a valid project.

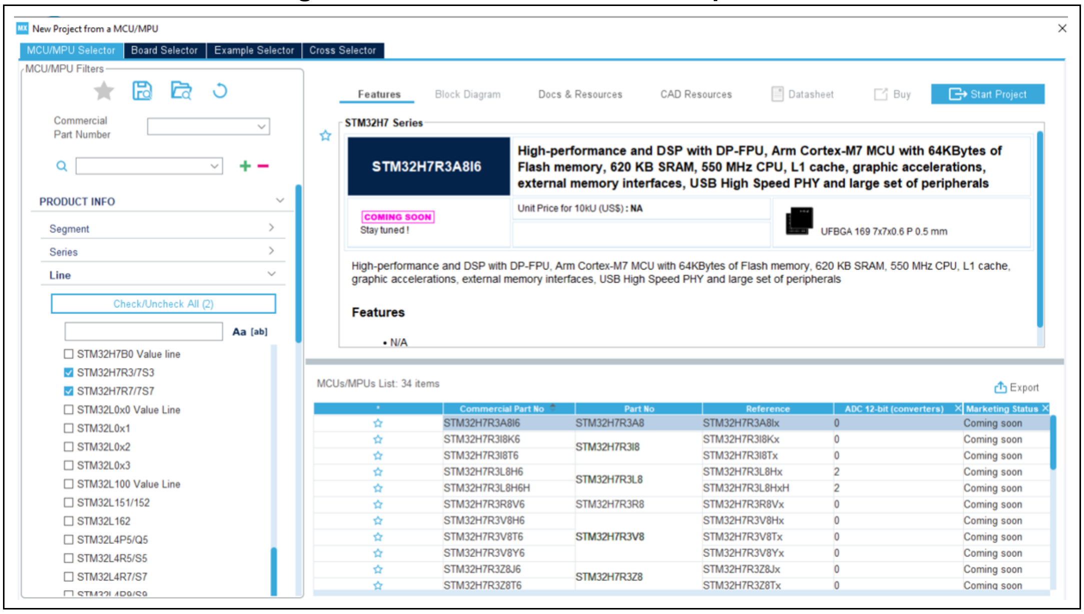
Region information

Clicking on a particular region in the Application Regions column shows the associated details on the left hand side.

STM32CubeMX automatically adds a 4-Gbyte region for the system core, even if you are not planning to use the MMT.

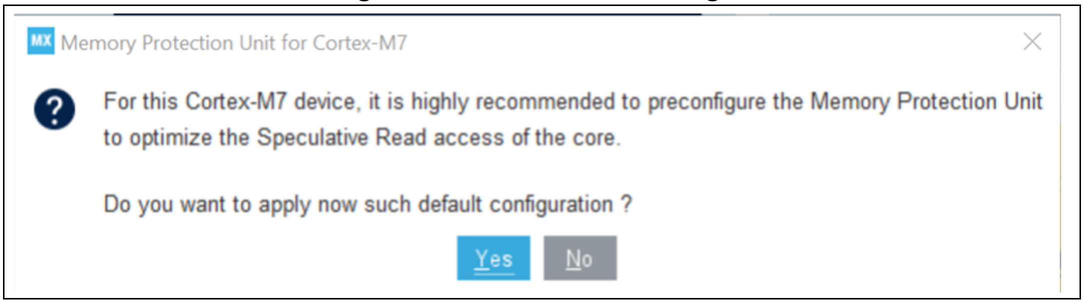
1. Choose a supported MCU (the following example is based on STM32H7R3A8I6).

Figure 427. Choose an STM32H7R product



2. Click on the Start Project button, then choose “Yes” on the “Memory Protection Unit for Cortex-M7” dialog box.

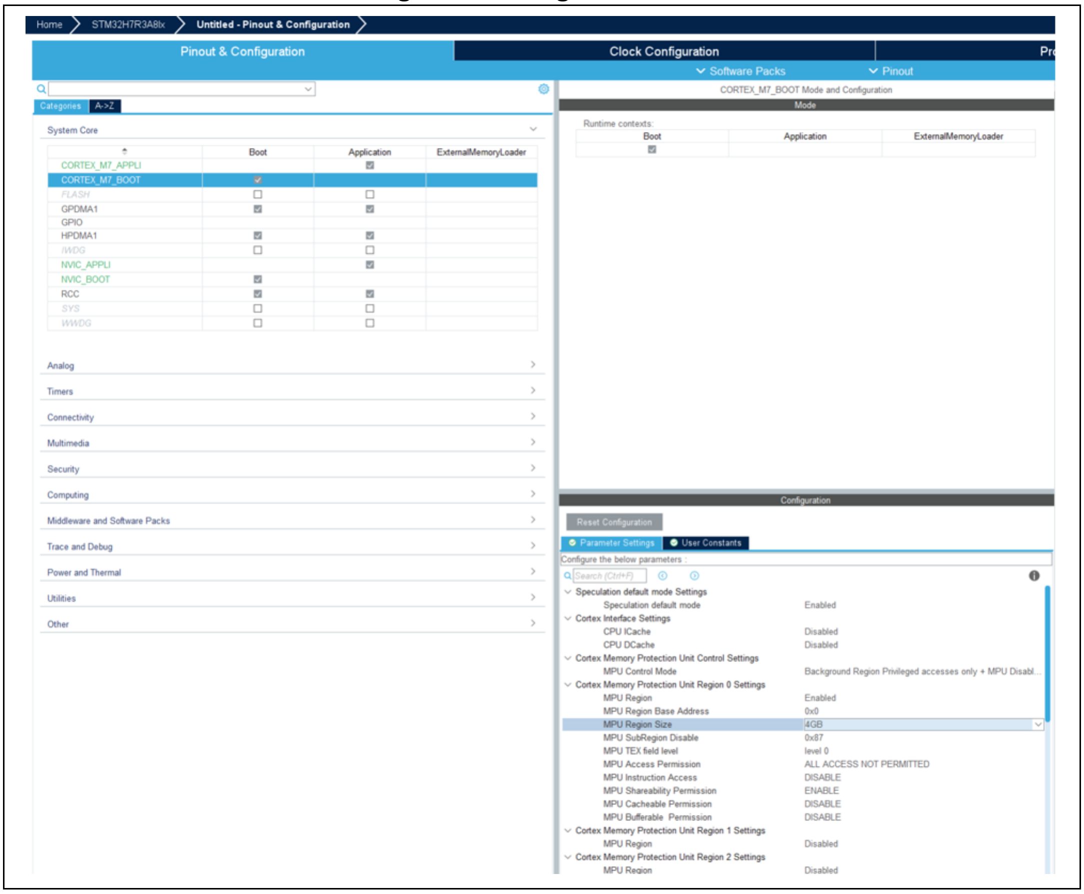
Figure 428. Initialization dialogue



STM32CubeMX applies the default configuration, then adds a 4-Gbyte region called “Region 0” under the CORTEX_M7_BOOT parameters, and a 4-Gbyte region called “Region 0” under the CORTEX_M7_APPLI parameters. The two regions start at the same address, adjust it to avoid overlap.

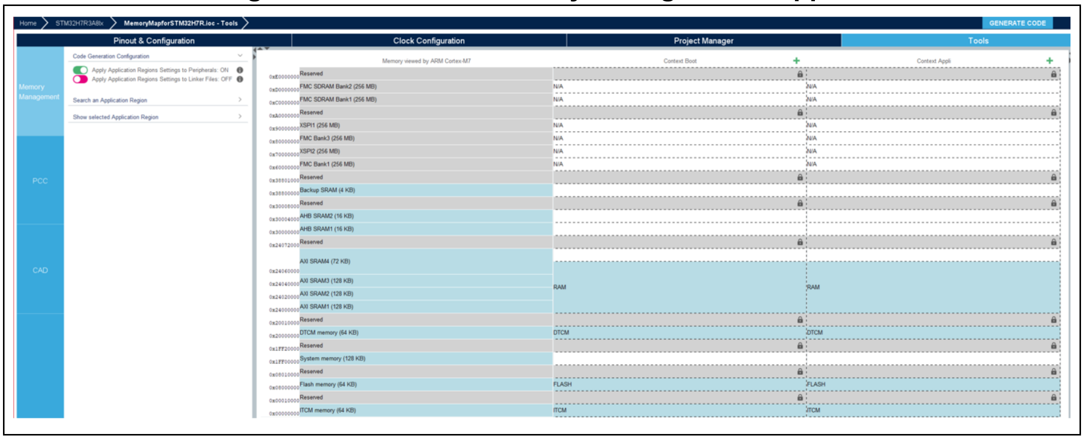
The new parameters can be checked using the Pinout and Configuration tab.

Figure 429. Region0 added



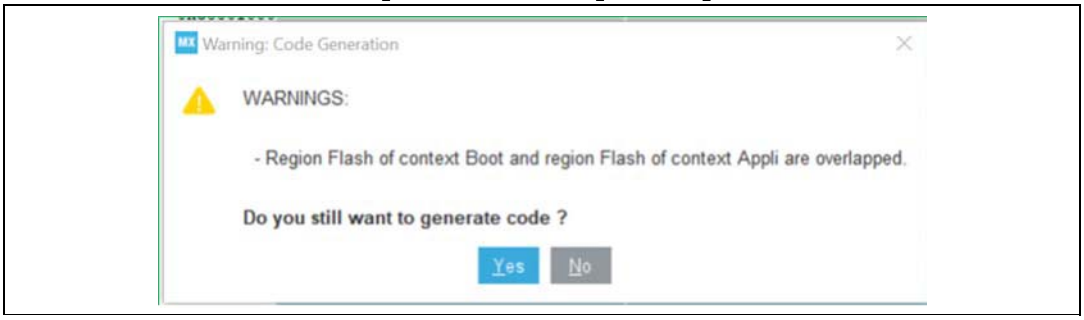
3. Select the Tools tab:
   a) Choose Memory Management
   b) Activate the Memory Management Tool support by clicking on "Apply Application Regions Settings to Peripherals"

Figure 430. Activate Memory Management support



4. Select the Project Manager tab
5. Give a name to the project and press the Generate Code button: a warning message is displayed.

Figure 431. Warning message

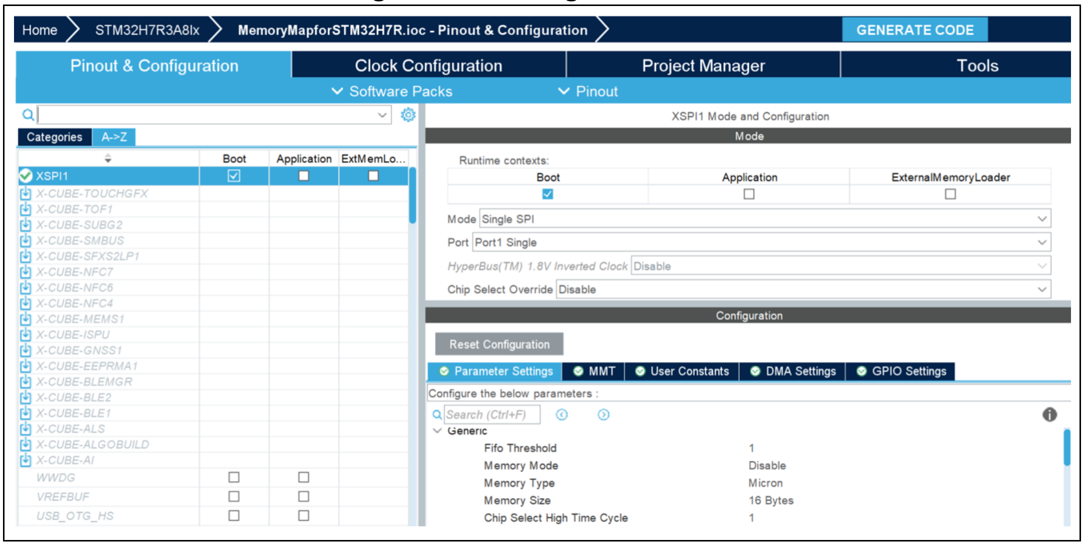


The flash region overlap issue can be solved in different ways, the preferred one goes through the following steps:

a) Select the Pinout and configuration tab

b) Enable XSPI1 for the boot context and choose the 'Single SPI' mode

Figure 432. Configure the XSPI

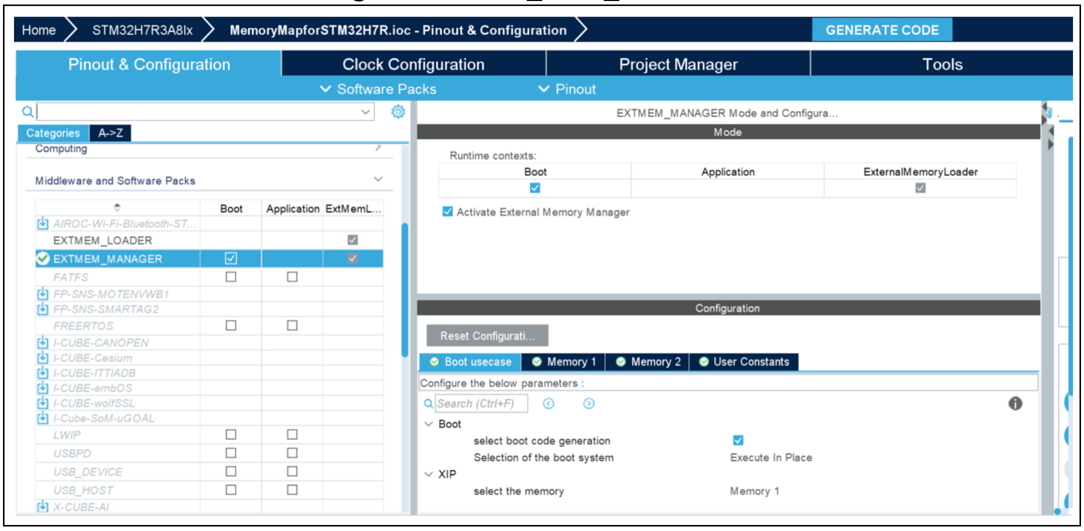


c) Activate the Middleware EXTMEM_MANAGER for the boot context:

> MMT solves the issue

> Press the Generate Code button to generate code for both applications. The overlap message does not appear any longer.

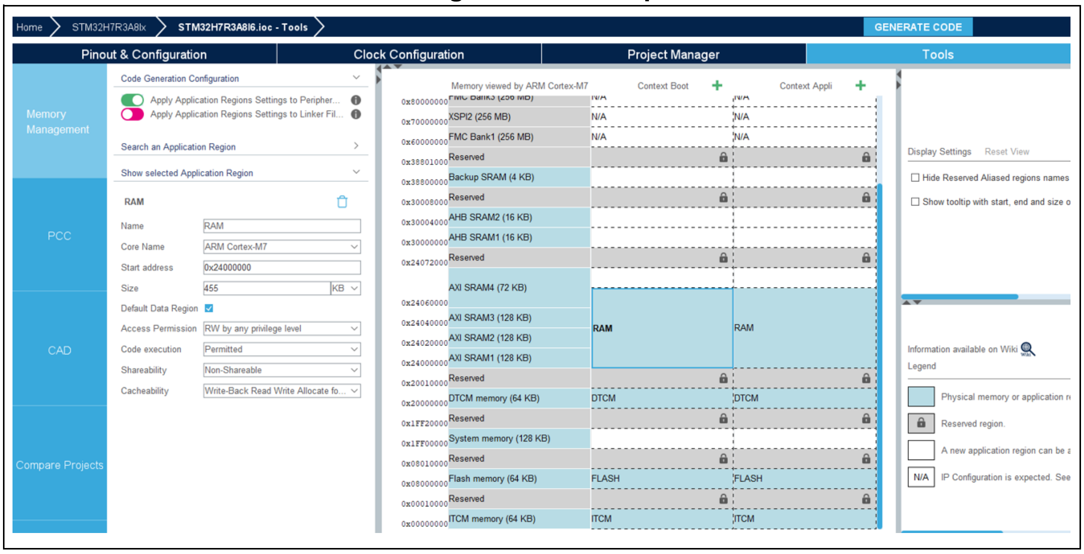
Figure 433. EXT_MEM_MANAGER



Code generation configuration

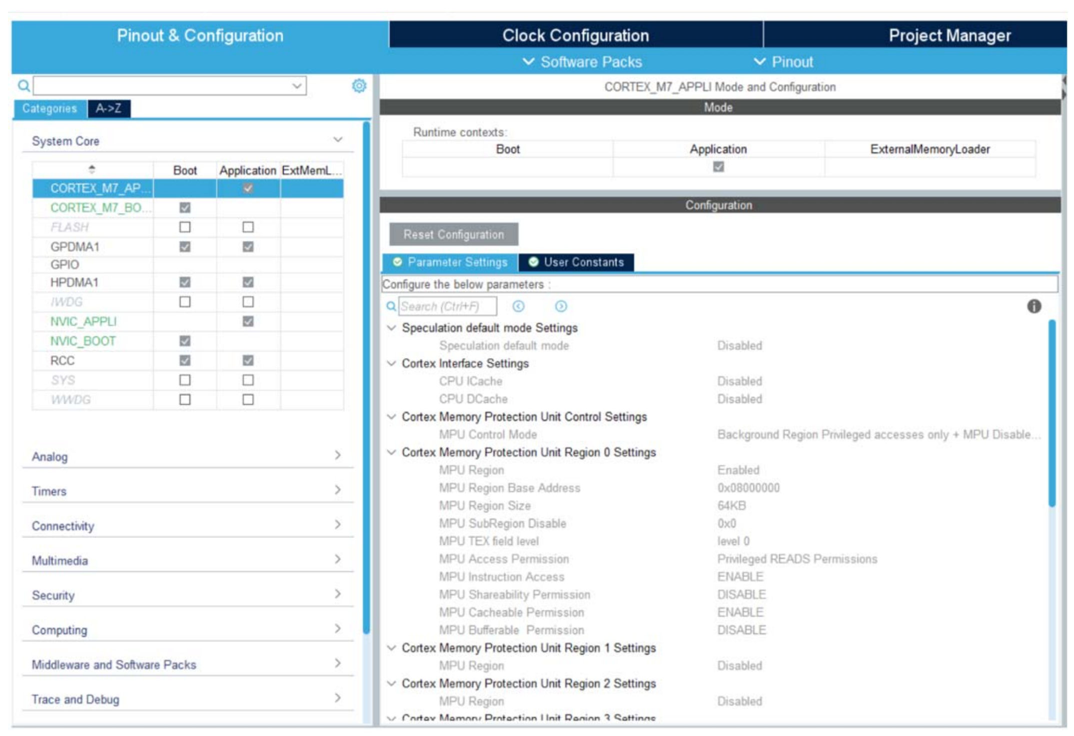
The application regions settings can be applied to peripherals on the left-hand side of the screen. The concerned peripherals are shown on the associated tooltip. This can impact their availability on the pinout screen configuration.

Figure 434. Tooltip



In this example, on the Pinout & Configuration panel, Cortex-M7_BOOT (MPU) and Cortex-M7_APPLI (MPU) are set and correspond to the region configuration on the Memory Management Tool. They are grayed out, as they cannot be modified.

Figure 435. IP configuration



Apply Application Regions settings to linker files

When this button is on, the linker scripts for the Boot project and Appli project are generated, taking into account the configuration.

Figure 436. Linker files update

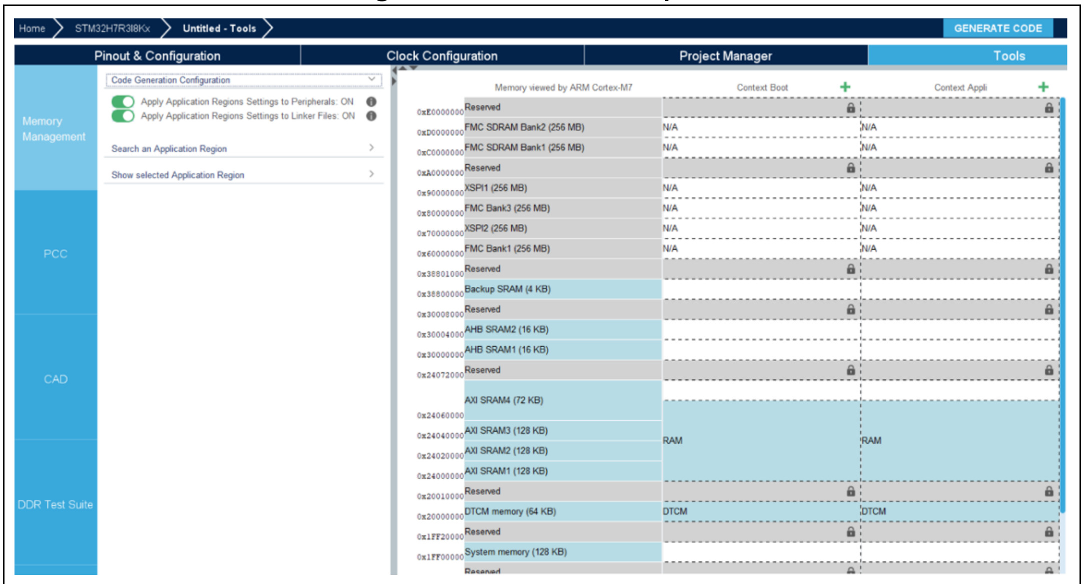
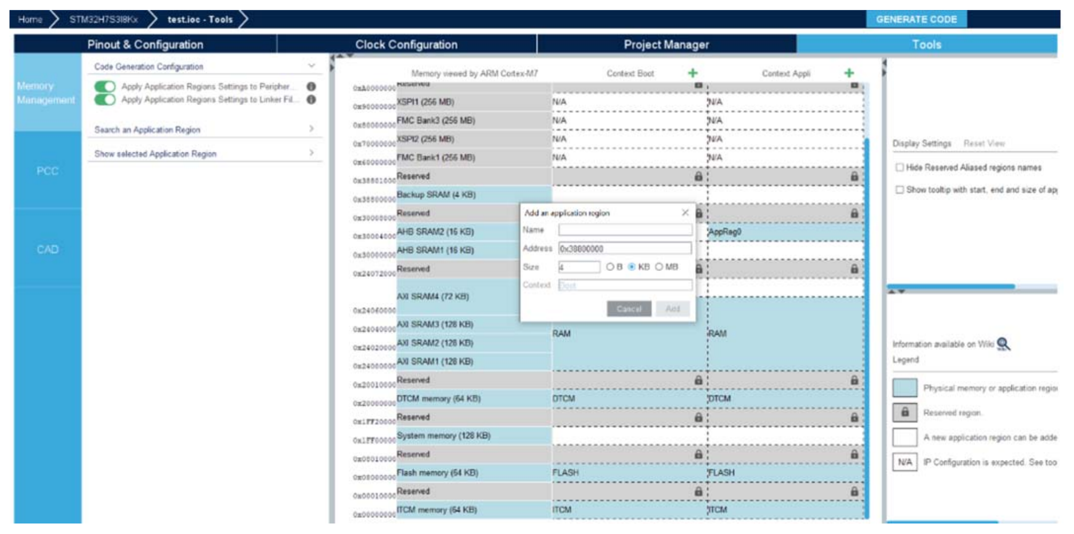


Figure 437. Memory assignment for context Boot H7RS



EXTMEM_MANAGER when using H7Rx/H7Sx

The middleware can be used with the “Select boot code generation” disabled or enabled.

If disabled, MMT automatically chooses the configured memory along with the associated driver, and sets the execution memory location in the linker file. This is the most straightforward way of configuring an external memory.

If enabled, by activating the “Select boot code generation” you can choose “Execute in Place” or “Load and Run”

- Execute in Place chooses and configures the memory zones
- Load and Run lets the user choose source, destination memory, and addresses to jump to. The configuration is translated into the linker file. The user must provide the source and destination addresses.

Figure 438. EXTMEM_MANAGER “Select boot code generation” disabled

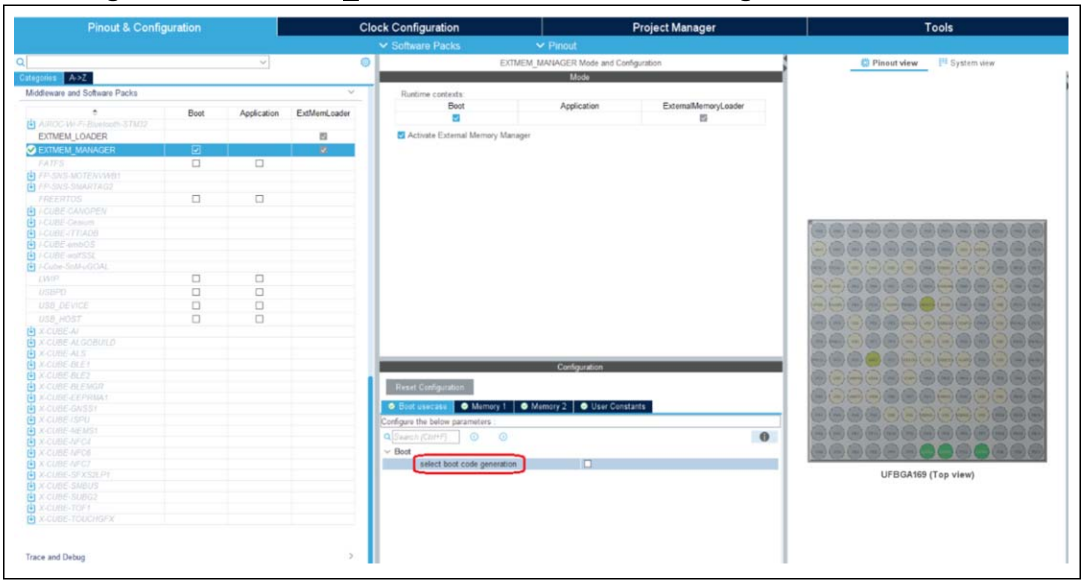


Figure 439. Execute In Place

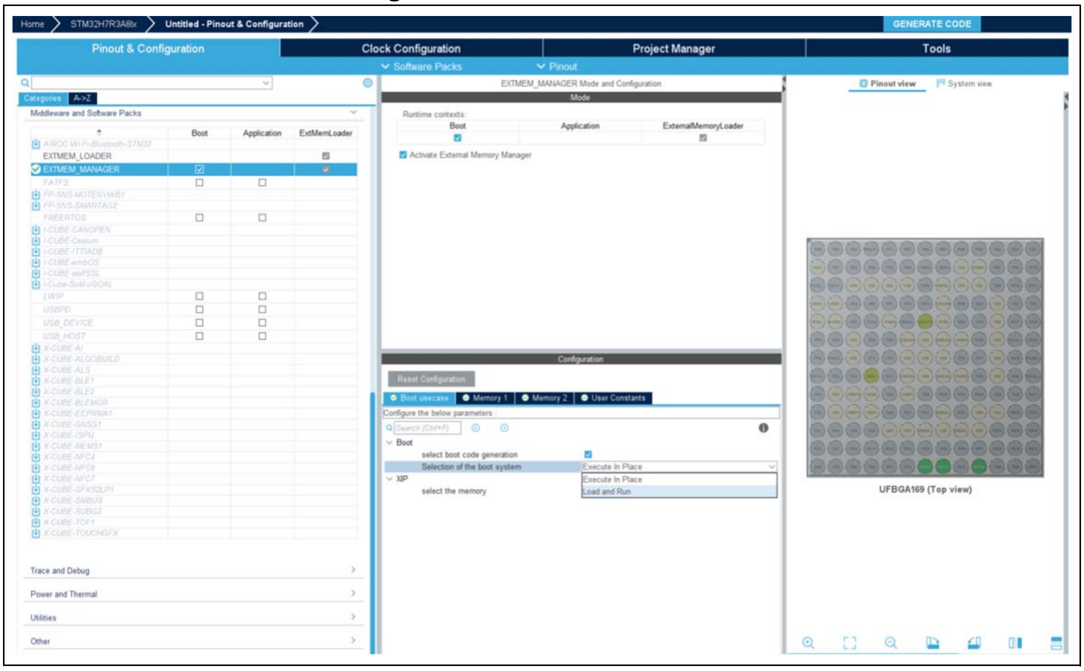


Figure 440. MMT Execute In Place

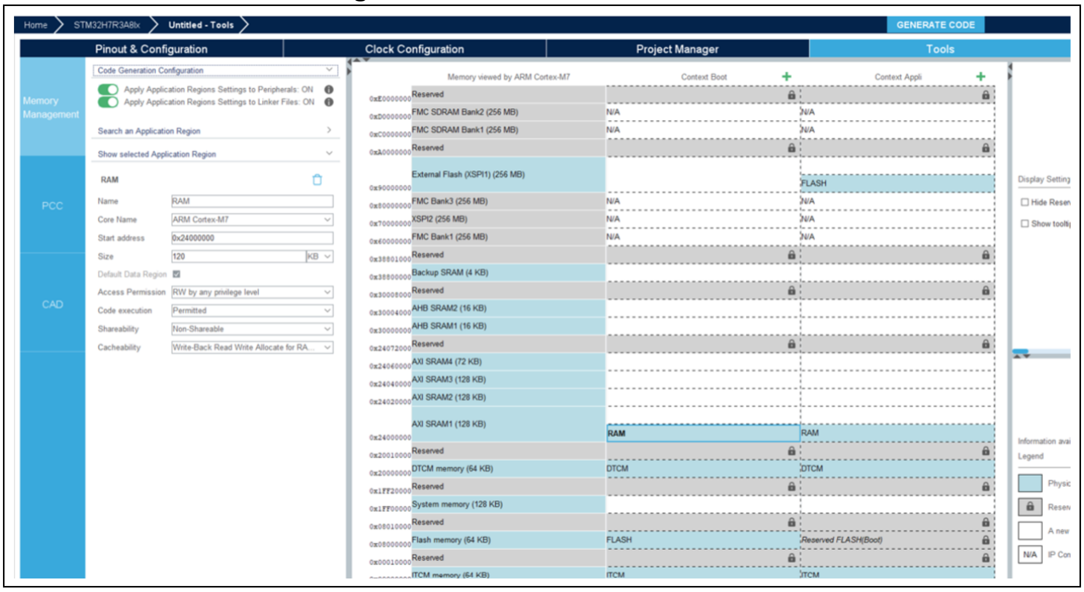


Figure 441. Load and Run

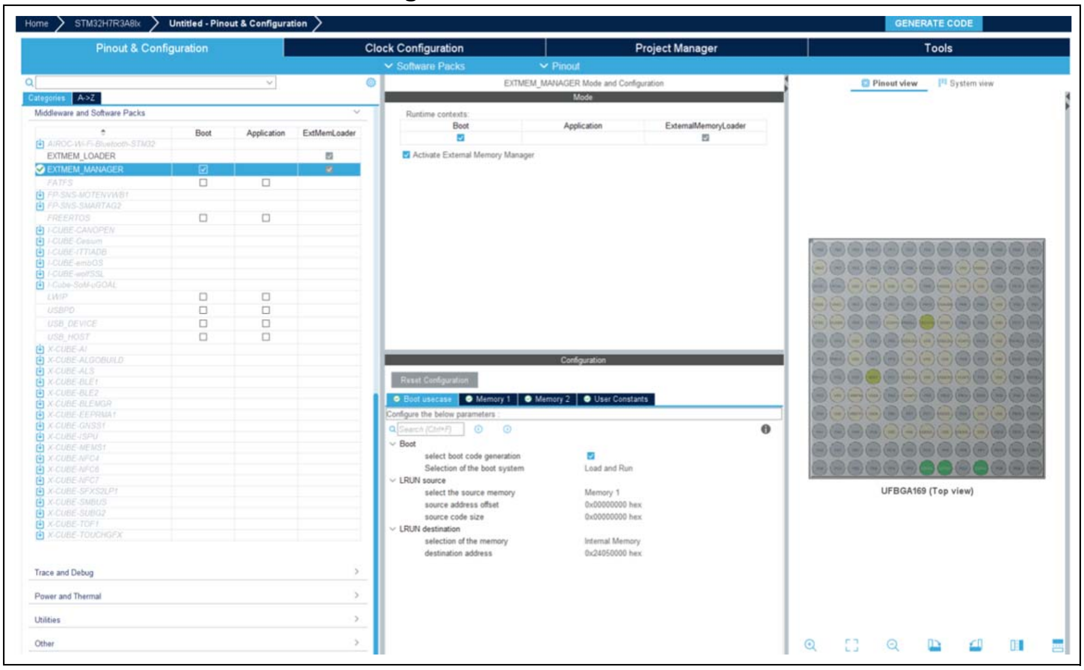
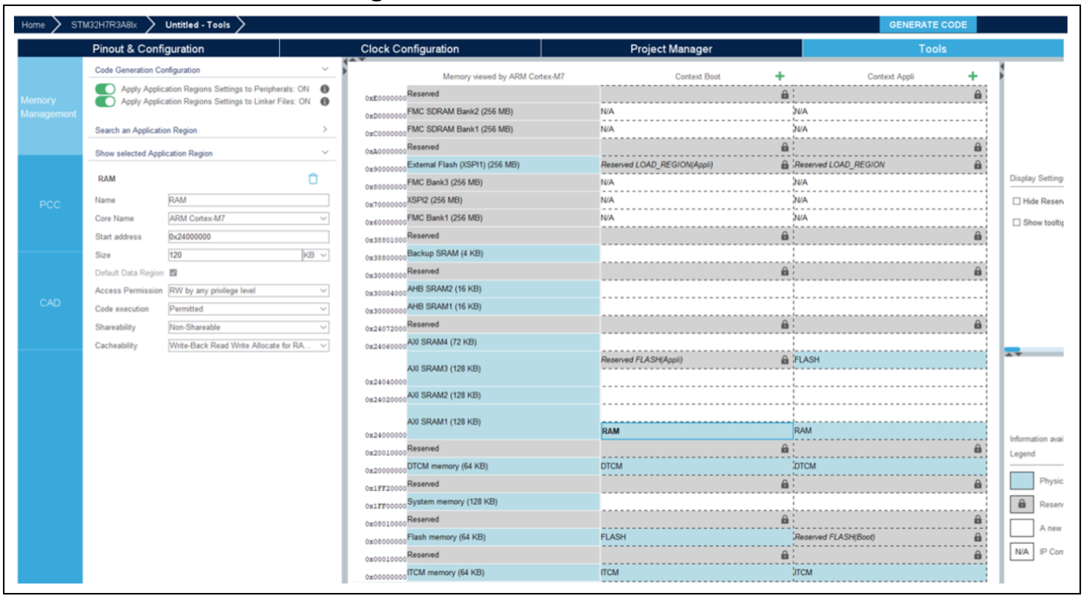


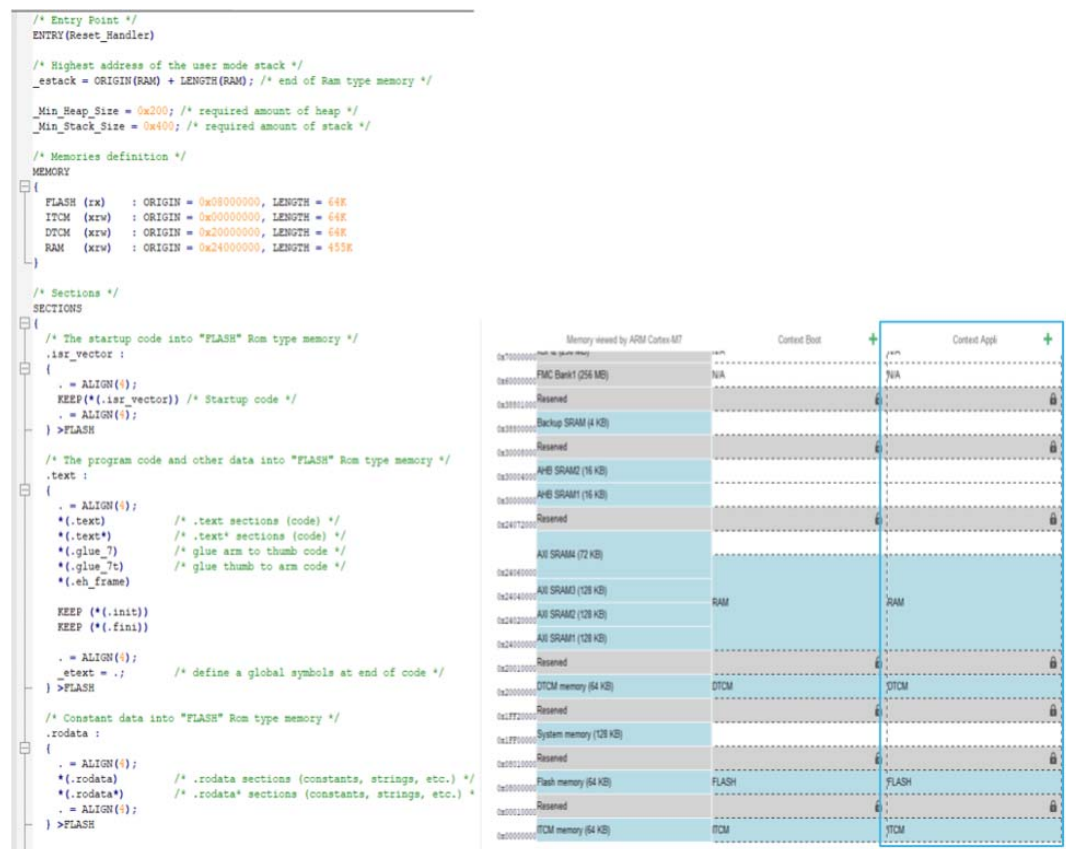
Figure 442. MMT Load and Run



After the code generation, navigate to the generated folder.

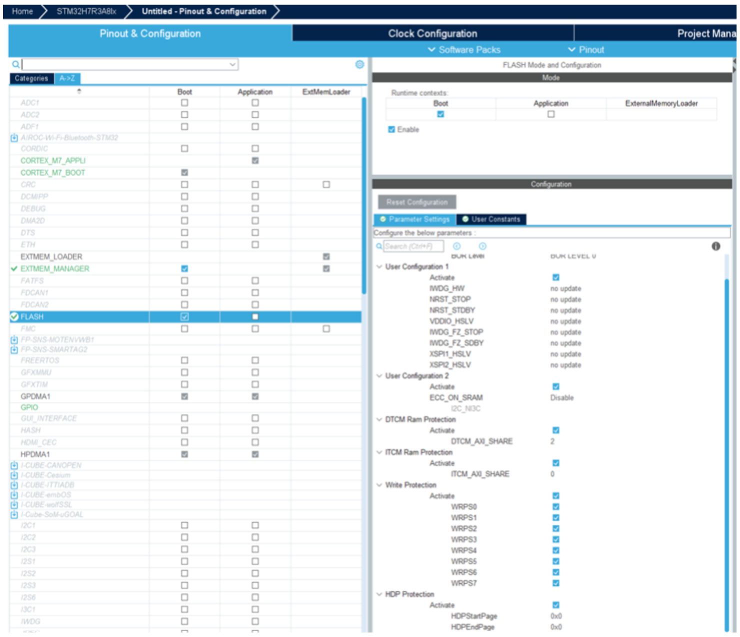
- Under the boot Project, open the linker definition file.
- Under the Memories definition you can see the defined memories with their start address and length, according to the configuration made in STM32CubeMX.

Figure 443. Linker files



Three option bytes can be used to configure the regions in the MMT. To see them, activate the IP FLASH on the Pinout and Configuration tab.

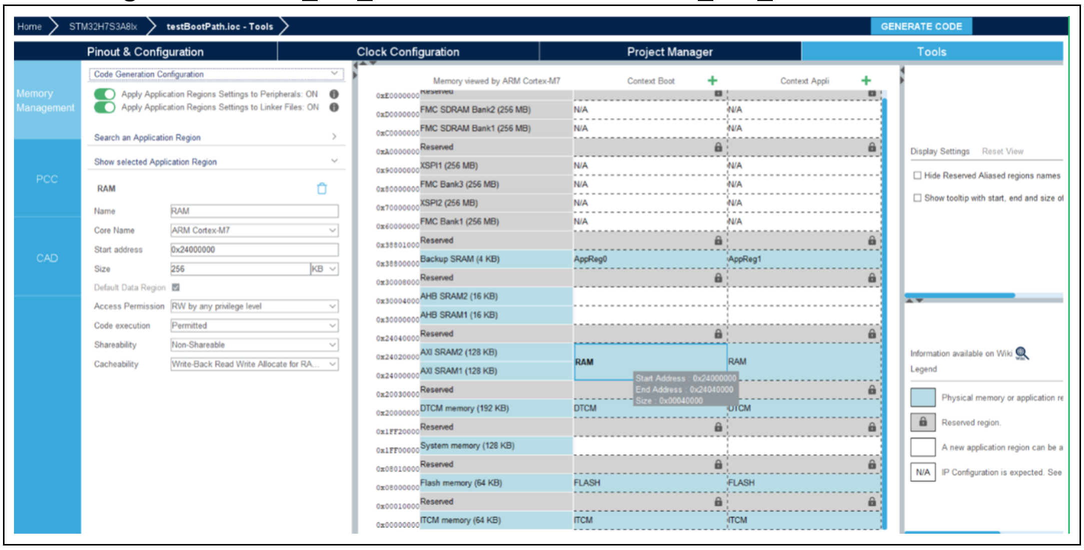
Figure 444. Flash option bytes



The option bytes interacting with the MMT are:

- ECC_ON_SRAM:
  - Linked to the AXI SRAM4 region on the MMT
  - When value is “disable” or “no update”, the AXI SRAM4 region size is set to 72 KB
  - When value is set to “enable” the AXI SRAM4 region is removed
- DTCM_AXI_SHARED:
  - Linked to the AXI SRAM3 region on the MMT
  - When set to 0 or 3, the AXI SRAM3 region size is set to 128 KB, and the size of region named DTCM is set to 64 KB
  - When set to 1, the AXI SRAM3 region size is set to 64 KB, and the size of region named DTCM is set to 128 KB
  - When set to 2, the AXI SRAM3 region is removed, and the size of region named DTCM is set to 192 KB
- ITCM_AXI_SHARED:
  - Linked to the AXI SRAM1 region on the MMT
  - When set to 0 or 3, the AXI SRAM1 region size is set to 128 KB
  - When set to 1, the AXI SRAM1 region size is set to 64 KB
  - When set to 2, the AXI SRAM1 region size is removed

Figure 445. ECC_ON_SRAM enabled and DTCM_AXI_SHARED set to 2



ETH impact on MMT when using H7RS/H7SX

An example of MMT configuration of the ETH IP on the STM32H7R3A8Ix MCU

1. Activate the IP ETH:
   - MMT creates three application regions for each context.
   - To change the start address and the size of each region, update the ETH parameters.
2. Press the radio button “Apply Application region Settings to Peripherals ON”, ETH will be partially under MMT control.
3. Press the Generate Code button to generate code for both applications.
   - Apply Application Regions settings to linker files:
4. When this button is on, the linker scripts are generated, considering the configuration.
5. After the code generation, navigate to the generated folder:
   - Open the linker definition file.
   - Under the Memories definition you can see the memories with their start address and length, according to the configuration made in STM32CubeMX.

Figure 446. ETH MMT regions for STM32H7R3A8Ix

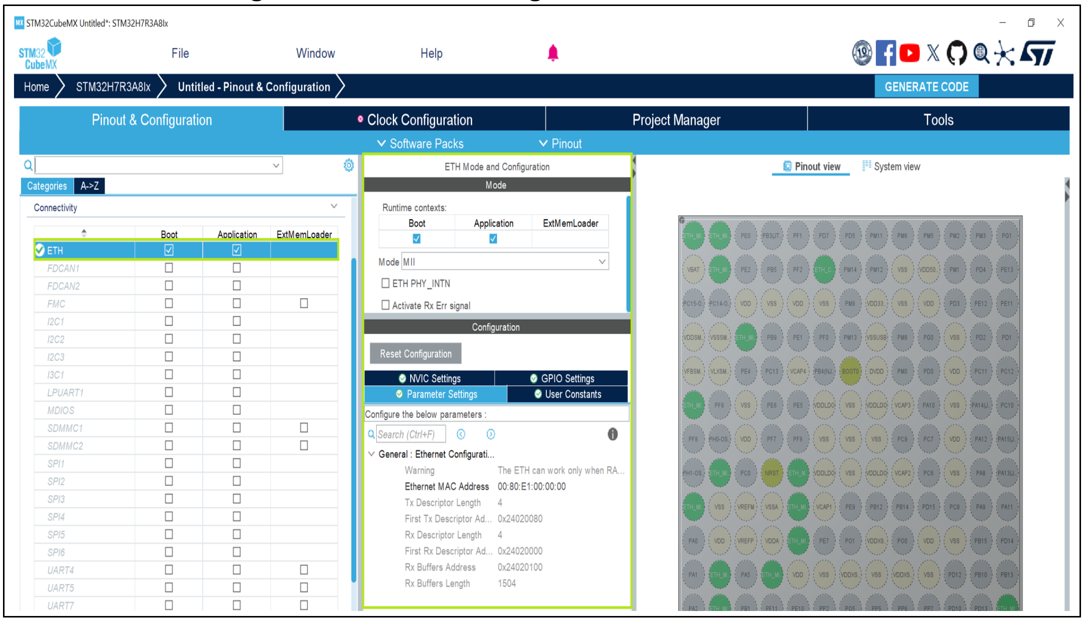


Figure 447. ETH configuration for STM32H7R3A8Ix

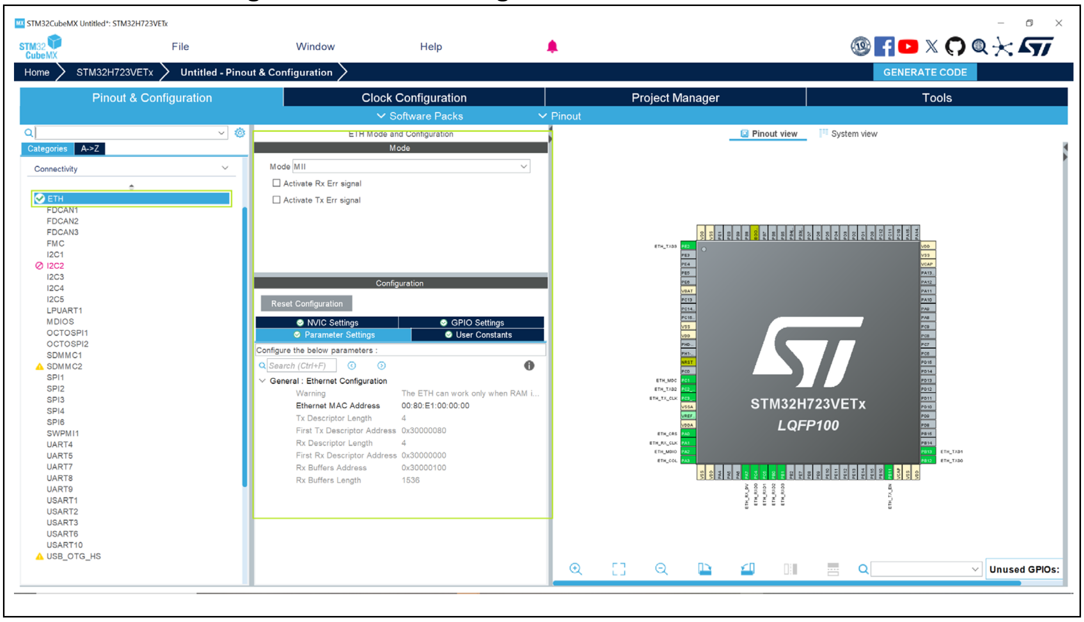


Figure 448. Application of the MMT configuration to the linker file

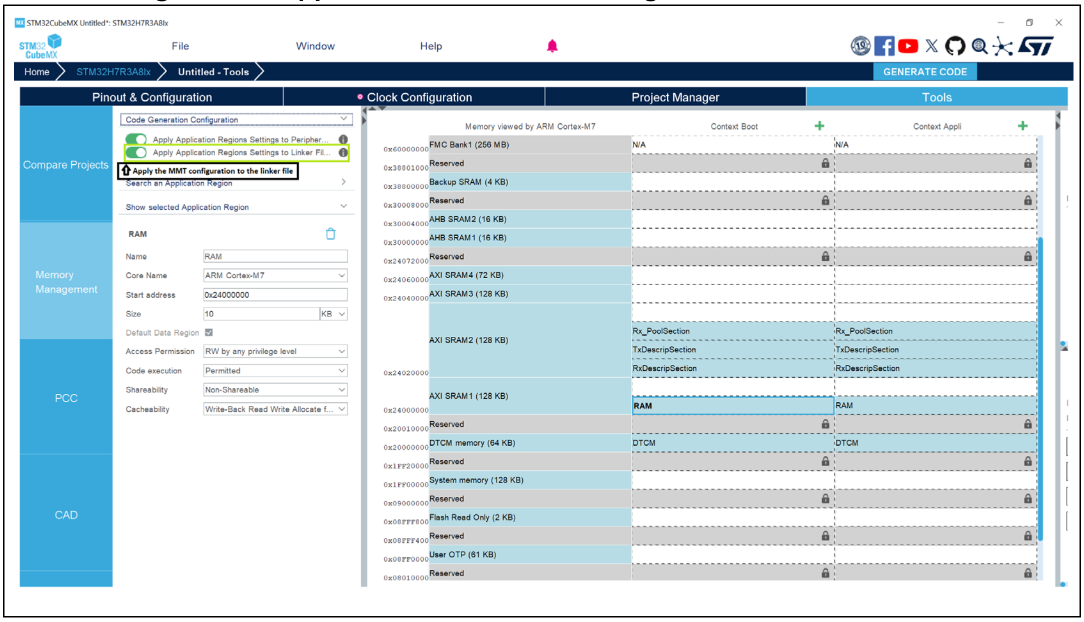


Figure 449. Defined memory regions under the linker file of the application context

```
/*******************************************************************************/
//*###ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\cortex_v1_0.xml" *//*-Specials-*/

/*******************************************************************************/

/*-Sizes-*/
define symbol __ICFEDIT_intvec_start__ = 0x08000000;

define symbol __ICFEDIT_size_cstack__ = 0x400;
define symbol __ICFEDIT_size_heap__  = 0x200;


/*-Start of  symbols-Auto-generated By STM32CubeMX*/
define symbol __ICFEDIT_region_ITCM_start__              = 0x00000000;
define symbol __ICFEDIT_region_ITCM_end__                = 0xFFFF;
define symbol __ICFEDIT_region_RxDescripSection_start__ = 0x24020000;
define symbol __ICFEDIT_region_RxDescripSection_end__   = 0x2402007F;
define symbol __ICFEDIT_region_TxDescripSection_start__ = 0x24020080;
define symbol __ICFEDIT_region_TxDescripSection_end__   = 0x240200FF;
define symbol __ICFEDIT_region_Rx_PoolSection_start__   = 0x24020100;
define symbol __ICFEDIT_region_Rx_PoolSection_end__     = 0x240206DF;

/*-End of MX Symbols*/

/*-Symbols-*/
define symbol __ICFEDIT_region_DTCM_start__              = 0x20000000;
define symbol __ICFEDIT_region_DTCM_end__                = 0x2000FFFF;
define symbol __ICFEDIT_region_RAM_start__               = 0x24000000;
define symbol __ICFEDIT_region_RAM_end__                 = 0x240027FF;
define symbol __ICFEDIT_region_FLASH_start__             = 0x08000000;
define symbol __ICFEDIT_region_FLASH_end__               = 0x800FFFF;
/**** End of ICF editor section. ###ICF###*/

define memory mem with size = 4G;

/*-MEMORY Regions-*/
define region DTCM_region             = mem:[from __ICFEDIT_region_DTCM_start__            to __ICFEDIT_region_DTCM_end__];
define region RAM_region              = mem:[from __ICFEDIT_region_RAM_start__             to __ICFEDIT_region_RAM_end__];
define region FLASH_region            = mem:[from __ICFEDIT_region_FLASH_start__           to __ICFEDIT_region_FLASH_end__];

/*-Start of  Regions- Auto-generated By STM32CubeMX*/
define region ITCM_region             = mem:[from __ICFEDIT_region_ITCM_start__            to __ICFEDIT_region_ITCM_end__];
define region RxDescripSection_region = mem:[from __ICFEDIT_region_RxDescripSection_start__ to __ICFEDIT_region_RxDescripSection_end__];
define region TxDescripSection_region = mem:[from __ICFEDIT_region_TxDescripSection_start__ to __ICFEDIT_region_TxDescripSection_end__];
define region Rx_PoolSection_region   = mem:[from __ICFEDIT_region_Rx_PoolSection_start__   to __ICFEDIT_region_Rx_PoolSection_end__];

/*-End of MX Regions-*/

/*-Blocks-*/
define block CSTACK with alignment = 8, size = __ICFEDIT_size_cstack__ {};
define block HEAP   with alignment = 8, size = __ICFEDIT_size_heap__  {};

/*-Initialization strategies-*/
initialize by copy { readwrite };
do not initialize  { section .noinit };

/*-Sections placements-*/

/*-Start of  Sections- Auto-generated By STM32CubeMX*/
place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };

place in ITCM_region  { section ITCM_section };
place in RxDescripSection_region  { section RxDescripSection_section };
place in TxDescripSection_region  { section TxDescripSection_section };
place in Rx_PoolSection_region  { section Rx_PoolSection_section };

/*-end of MX Sections-*/

/*-user Sections-*/
place in FLASH_region  { readonly };
place in RAM_region  { readwrite, block HEAP, block CSTACK };
```

5.5.7 STM32WB0

Feature: MMT usage, pinout, and configuration user interface

When the first toggle button is ON, Cortex-M0+ (MPU) is under MMT control: its modes and parameters become read-only (see *Figure 450*).

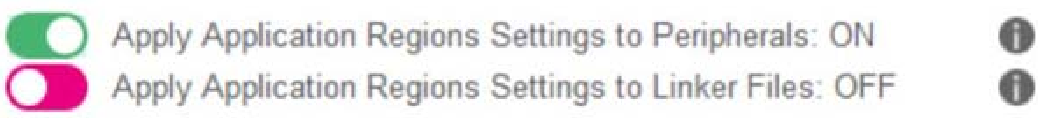
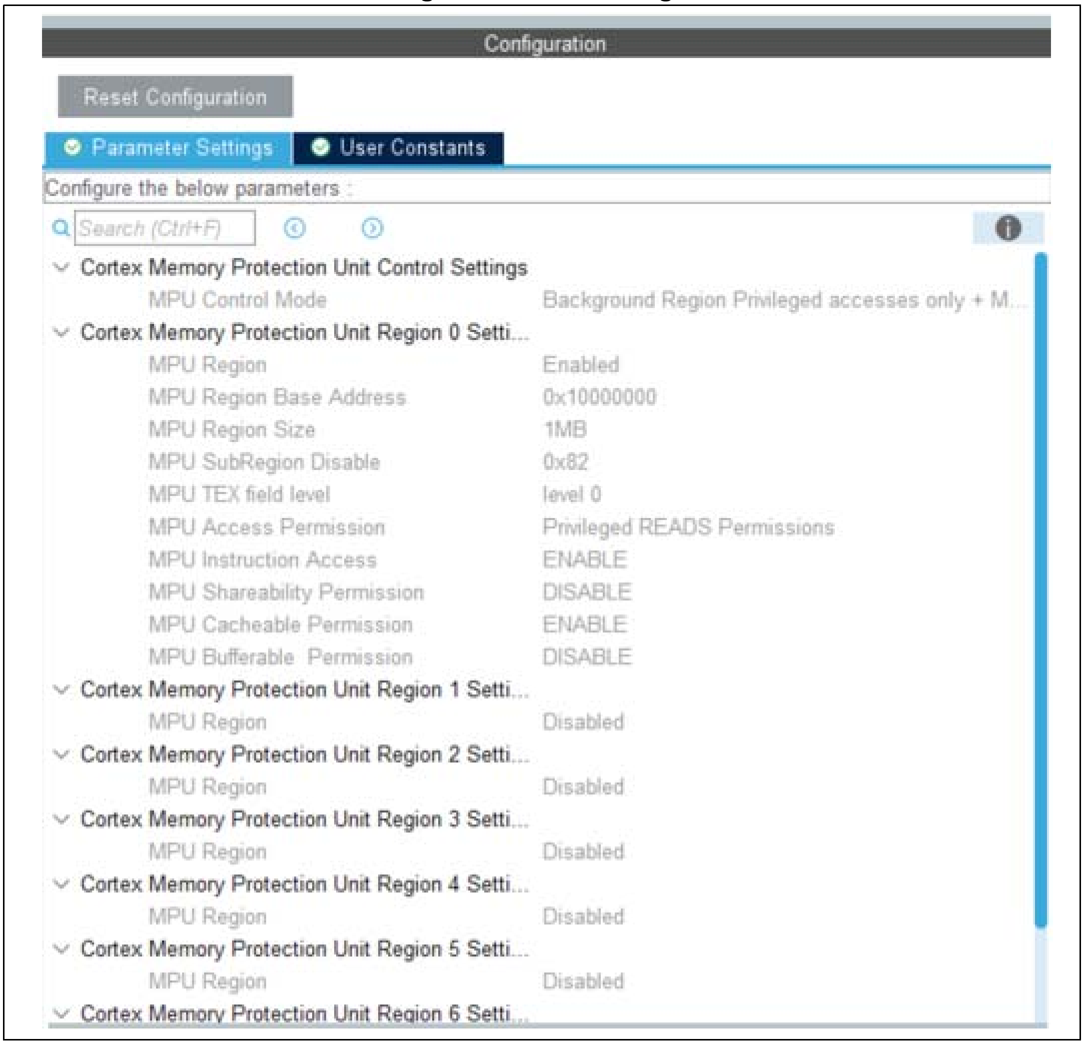


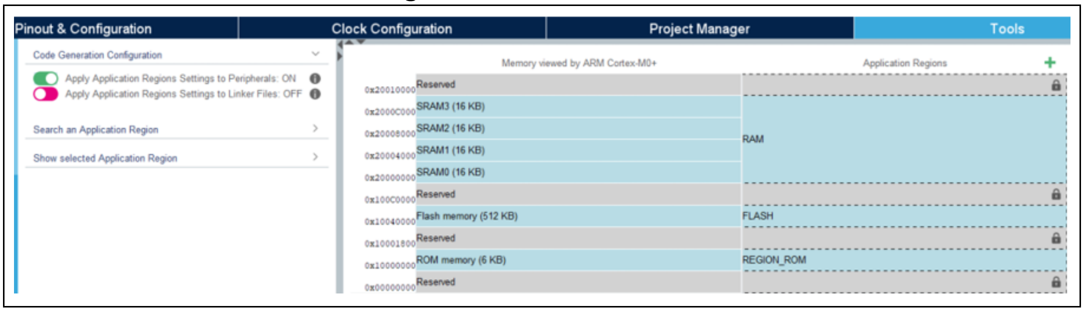
Figure 450. MMT usage



User interface

The middle panel represents the memory, split into two columns: the left one is the memory seen by the core Cortex-M0+, the right one the memory set-up for the application.

Figure 451. User interface



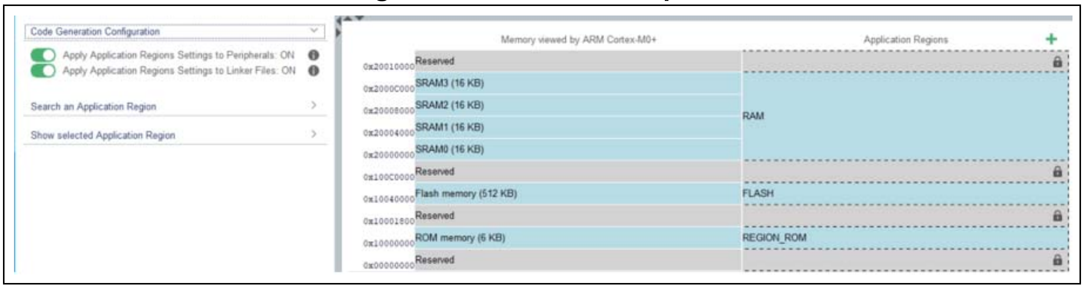
For a new project created under STM32CubeMX, the MMT creates the default application region to generate a valid project.

Apply Application Regions settings to linker files

When this button is on, the linker scripts for the project are generated, considering the configuration.

- The REGION_ROM is a default code region used in linker.
- The linker file copies the STM32Cube firmware linkers files and only MMT region is updated or added.
- OTA tag is not managed by MMT and usually exists in the linker file.

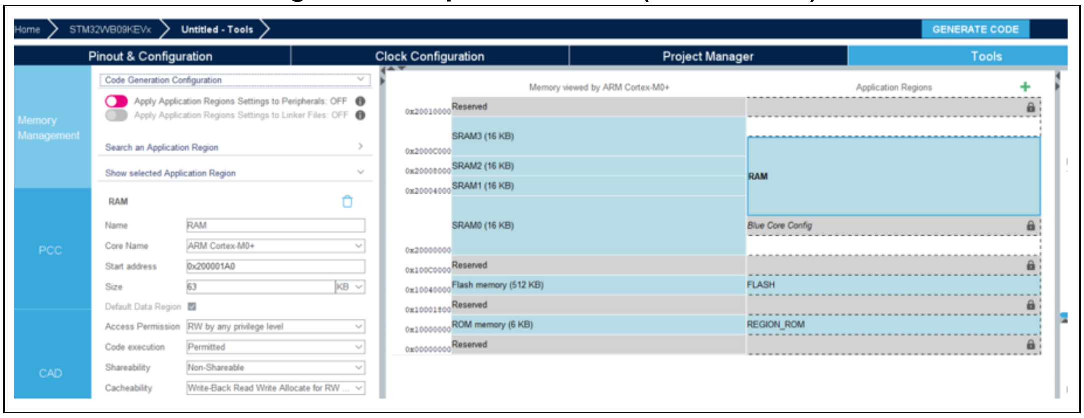
Figure 452. Linker files update



Impact on STM32WB09 RADIO

When this IP is activated, a reserved region “Blue Core Config” calculated by value of CFG_NUM_RADIO_TASKS, which varies from 1 to 128, is added.

Figure 453. Impact on RADIO (STM32WB09)



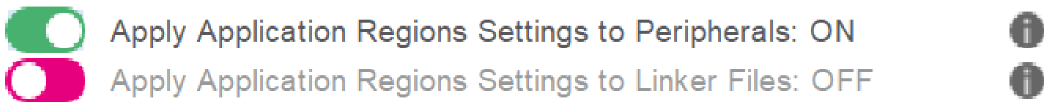
5.5.8 MMT for STM32N6 products

MMT usage, pinout, and configuration of the user interface

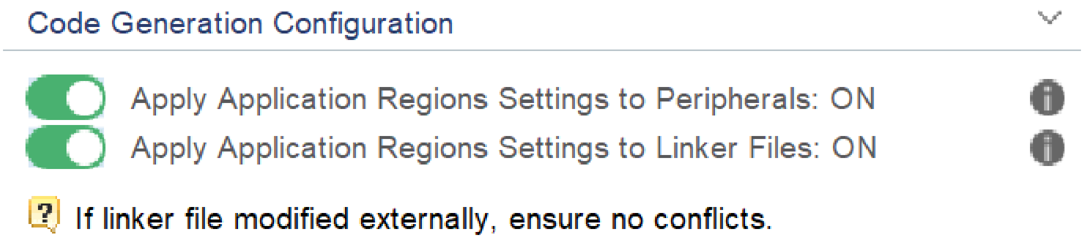
MMT interaction with peripherals starts from the moment the user enters their interfaces:

- Checks their settings
- Updates other peripherals involved in the memory map configuration

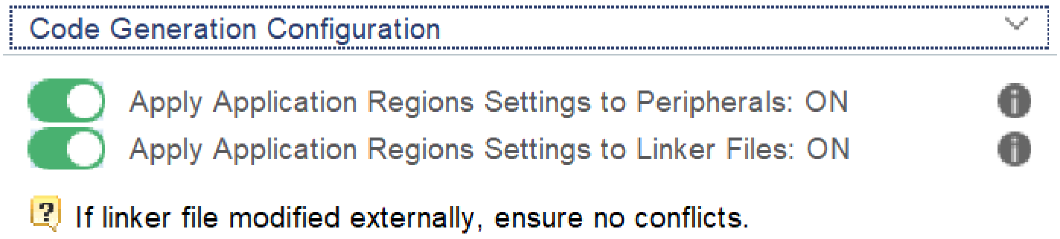
The peripherals are updated only when the first toggle button is “ON” in the panel Memory Management (under STM32CubeMx Tools).



MMT updates the scripts only when the second toggle button is ON. The linker file content is generated according to the configuration of the application regions.



The applicative regions are saved in the user project even if the first toggle button is OFF.

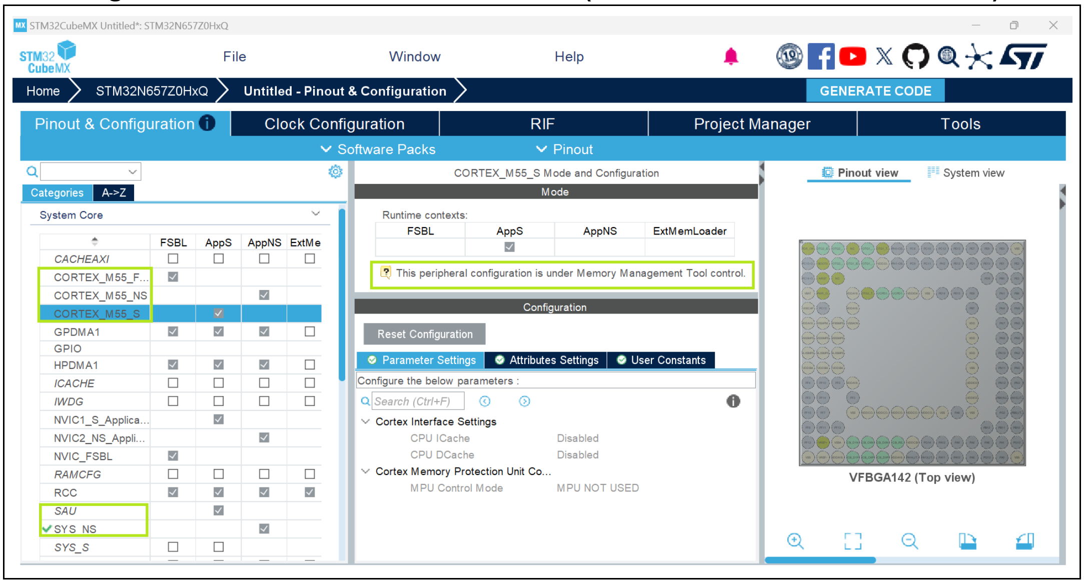


Feature under MMT control for secure and nonsecure domains

When the first toggle button is ON, the following features are controlled by MMT (their modes and parameters become read-only):

- CORTEXM55-FSBL (MPU)
- CORTEXM55_S (MPU)
- CORTEXM55_NS (MPU)
- SAU
- RIF (RISAF panel)

Figure 454. Feature under MMT control (secure and nonsecure domains)

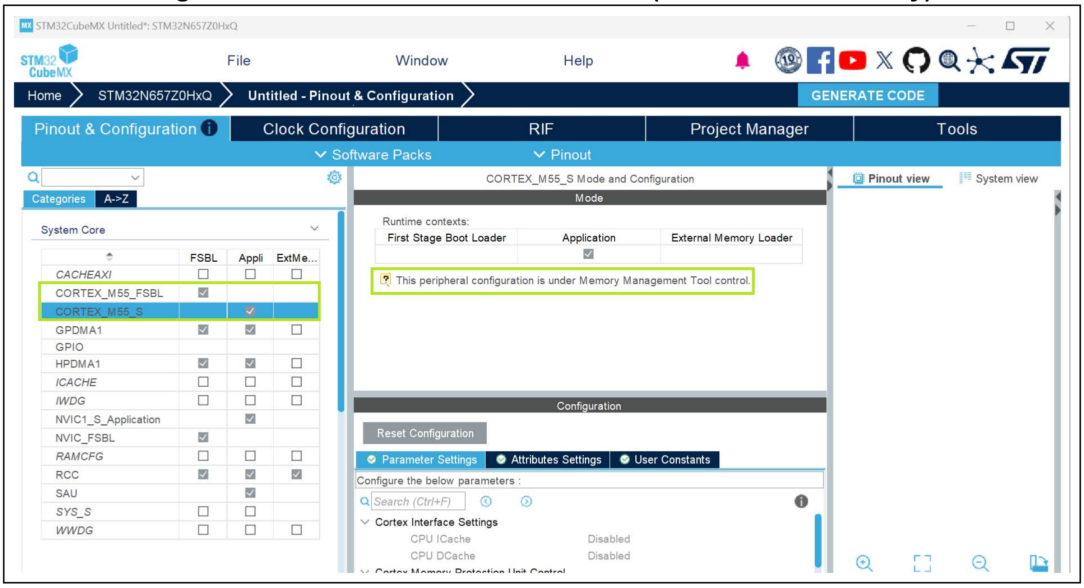


Feature under MMT control for secure domains

When the first toggle button is ON the following features are controlled by MMT (their modes and parameters become read-only):

- CORTEXM55-FSBL (MPU)
- CORTEXM55_S (MPU)
- RIF (RISAF panel)

Figure 455. Feature under MMT control (secure domains only)

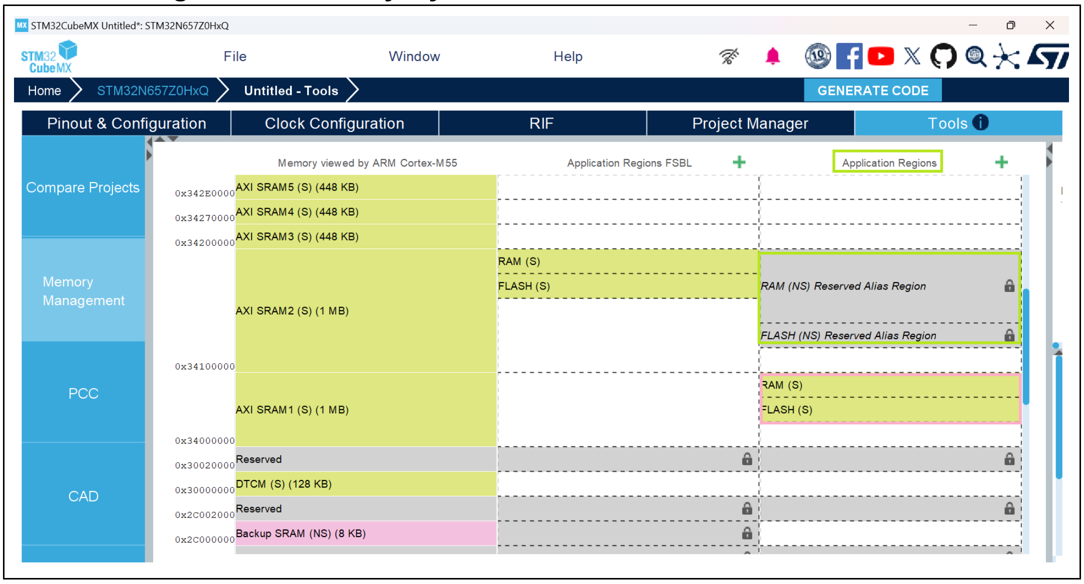


Application regions for secure and nonsecure domains

Only FSBL, CORTEX_M55_S, and CORTEX_M55_NS contexts are managed by the MMT. Each context has its own application region.

In the memory layout the CORTEX_M55_S and CORTEX_M55_NS use the same column for application regions.

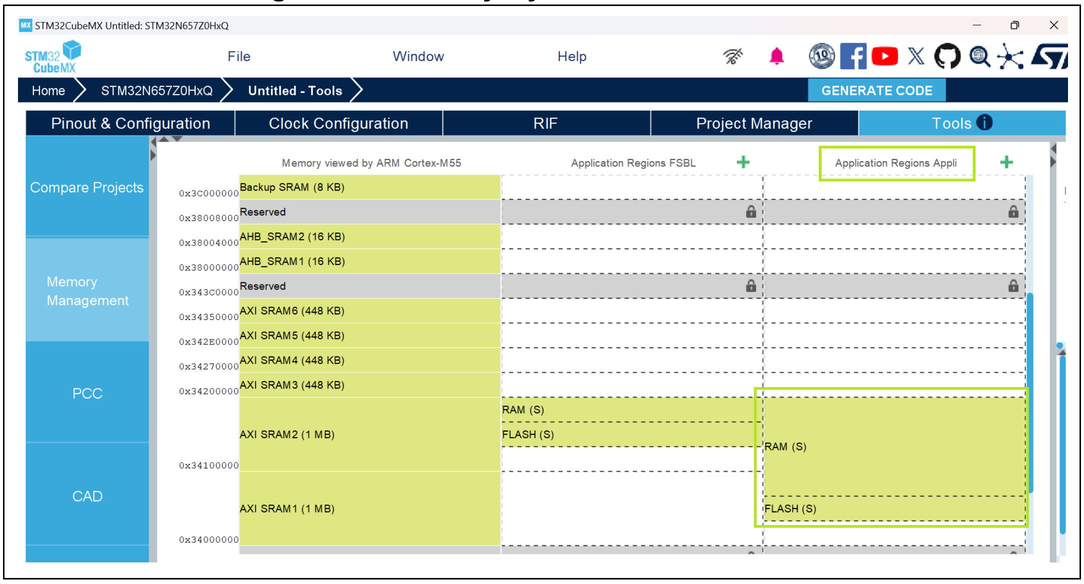
Figure 456. Memory layout for secure and nonsecure domains



Application regions for secure domains

Only FSBL and CORTEX_M55_S contexts are managed by the MMT. Each context has its own application region.

Figure 457. Memory layout for secure domains



Description of the memory layout in STM32CubeMX user interface

The memory is split in three columns:

1. The left one is the memory seen by the core(s)
2. The middle one is the memory set-up for the application in FSBL context

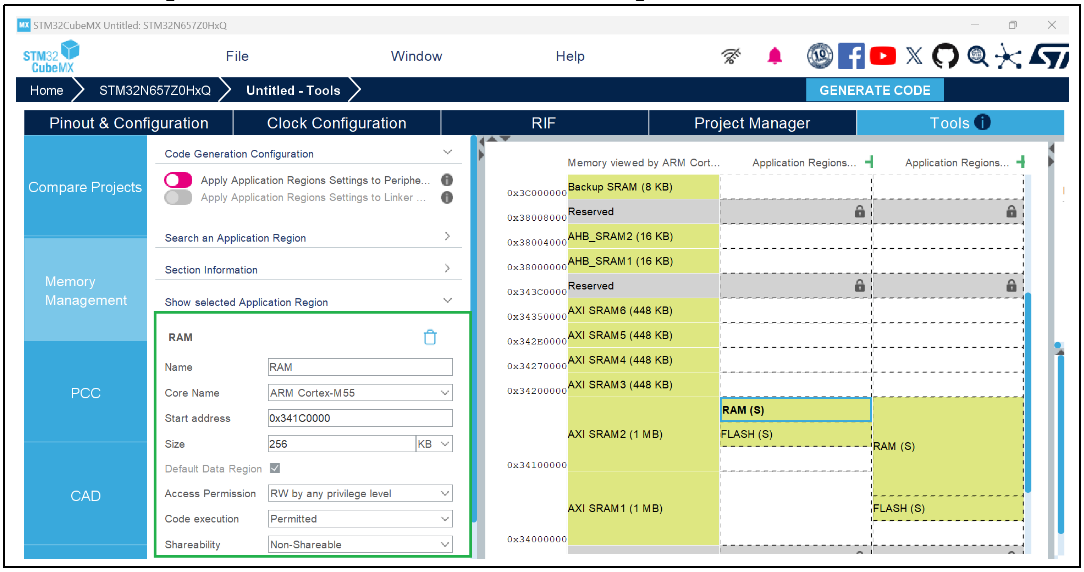
3. The right one is the memory set-up for the application in the Context Appli

For the new project under STM32CubeMX, the tool creates the default application region to generate a valid project.

Region information

Clicking on a particular region in the Application Regions column shows the associated details on the left-hand side.

Figure 458. Details about the clicked region in the FSBL context



STM32CubeMX automatically adds a 4-Gbyte region for the system core, even if you are not planning to use the MMT. Each region in the memory layout has its own characteristics, detailed at the bottom of the STM32CubeMX interface.

Figure 459. Regions designation for secure and nonsecure domains

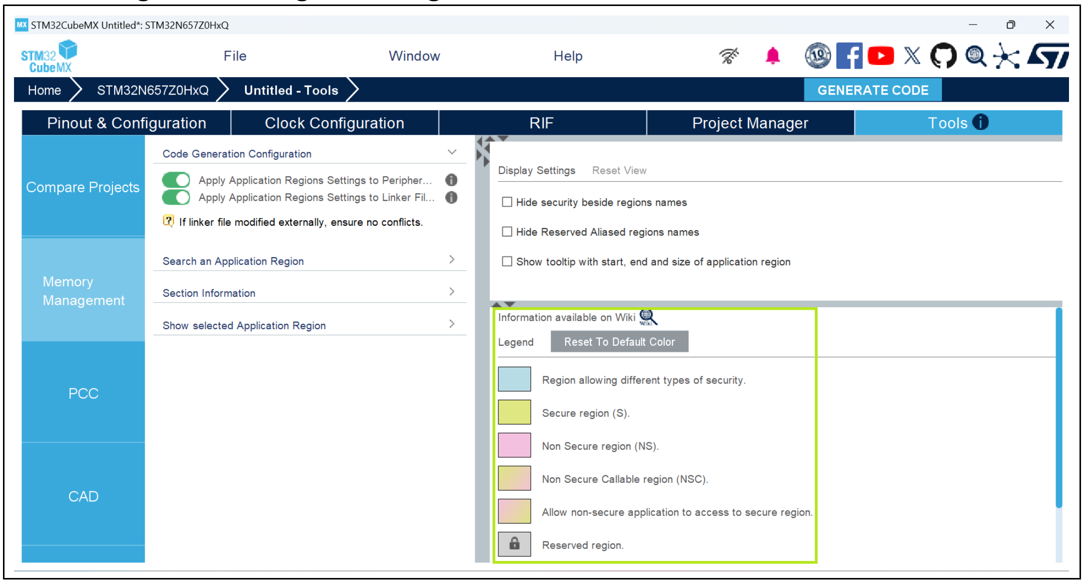
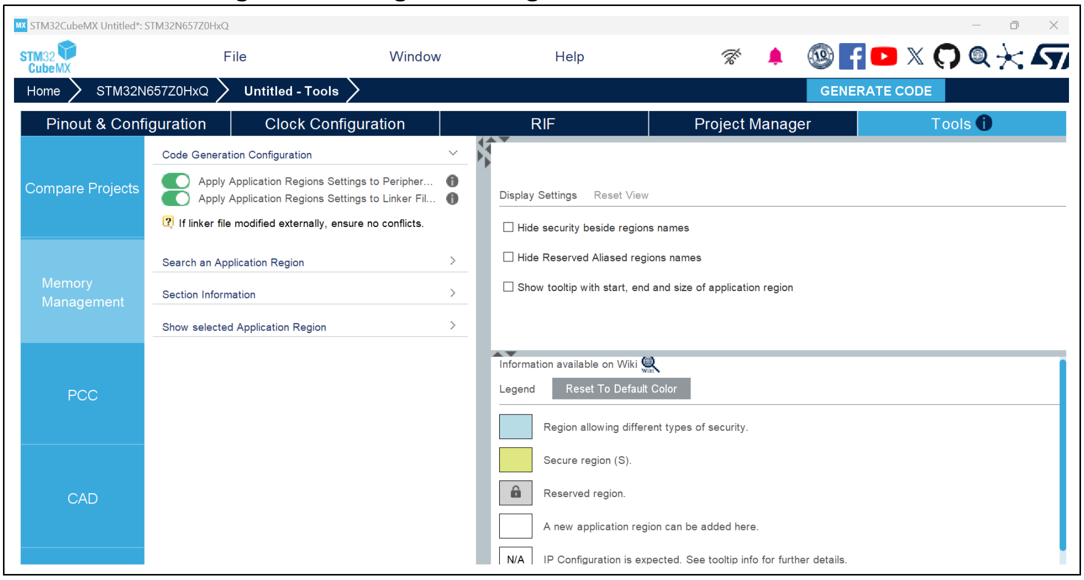


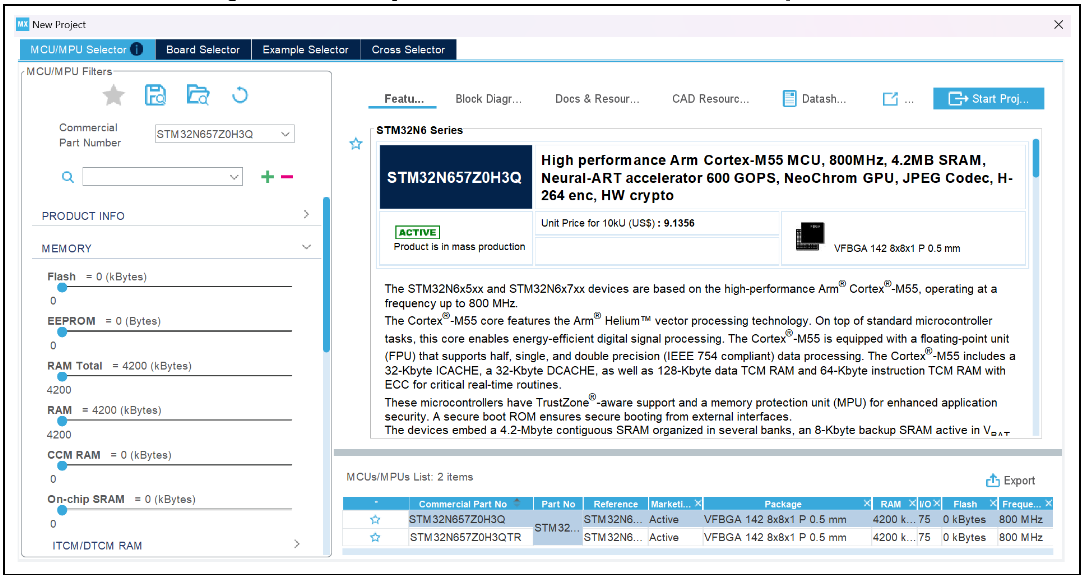
Figure 460. Regions designation for secure domains



Example of using MMT with STM32N6 products

1. Choose an MCU that supports the MMT feature (the following example is based on STM32N657Z0HxQ).

Figure 461. Project creation with an STM32N6 product



2. Choose the project structure, then initiate the project creation by clicking Start Project. Upon display of the *TrustZone feature available* dialog box, specify the domain configuration as *Secure and Non-Secure domains* before confirming with OK (*Figure 462*). Choose the second option if you intend to work exclusively within secure domains (*Figure 463*). STM32CubeMX applies the default configuration (RAM and FLASH regions are created in the MMT view).

Figure 462. Choosing the project structure (secure and nonsecure domains)

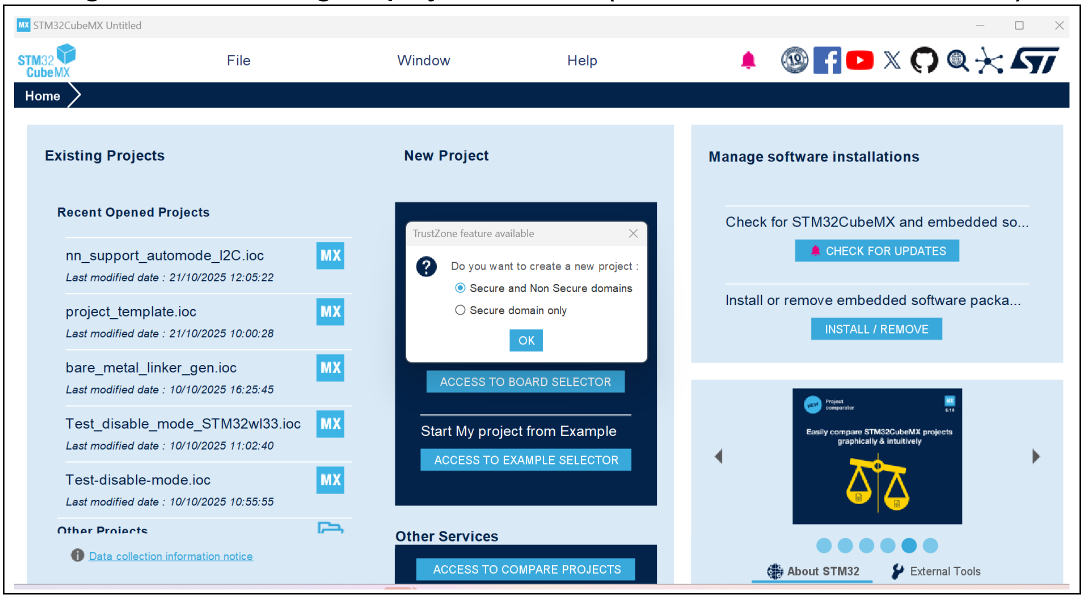
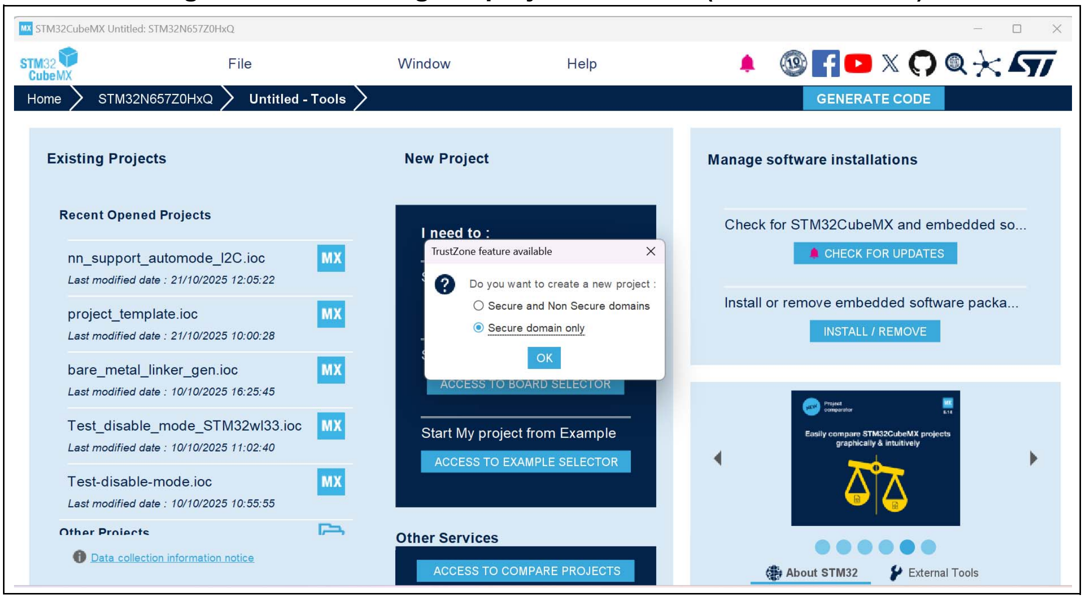


Figure 463. Choosing the project structure (secure domains)



3. Configure the MMT
   a) Click on the Tools tab
   b) Choose Memory Management panel
   c) Activate the Memory Management Tool support by clicking on “Apply Application Regions Settings to Peripherals”

Figure 464. MMT configuration (secure and nonsecure domains)

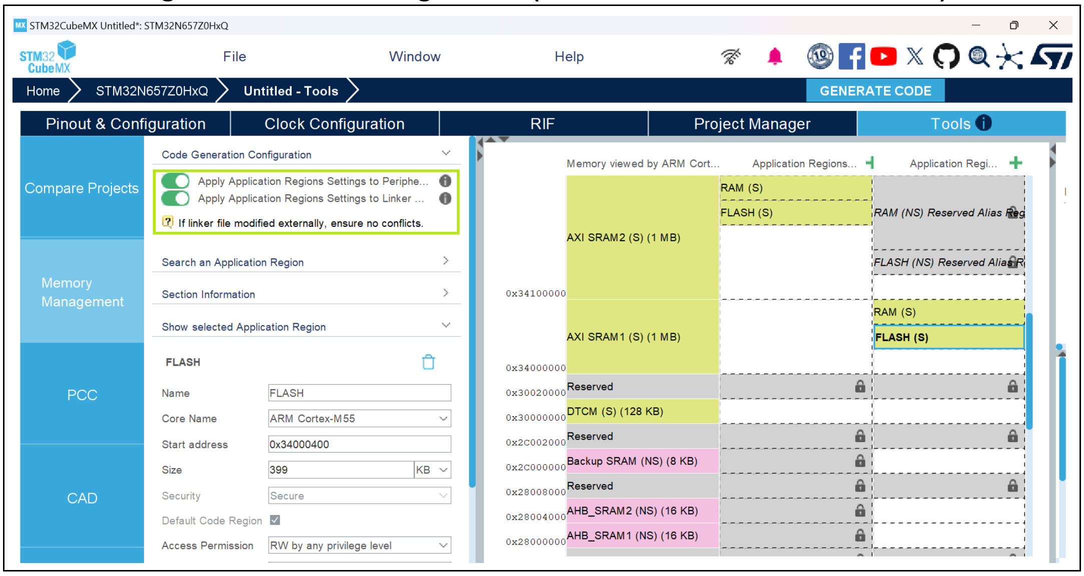
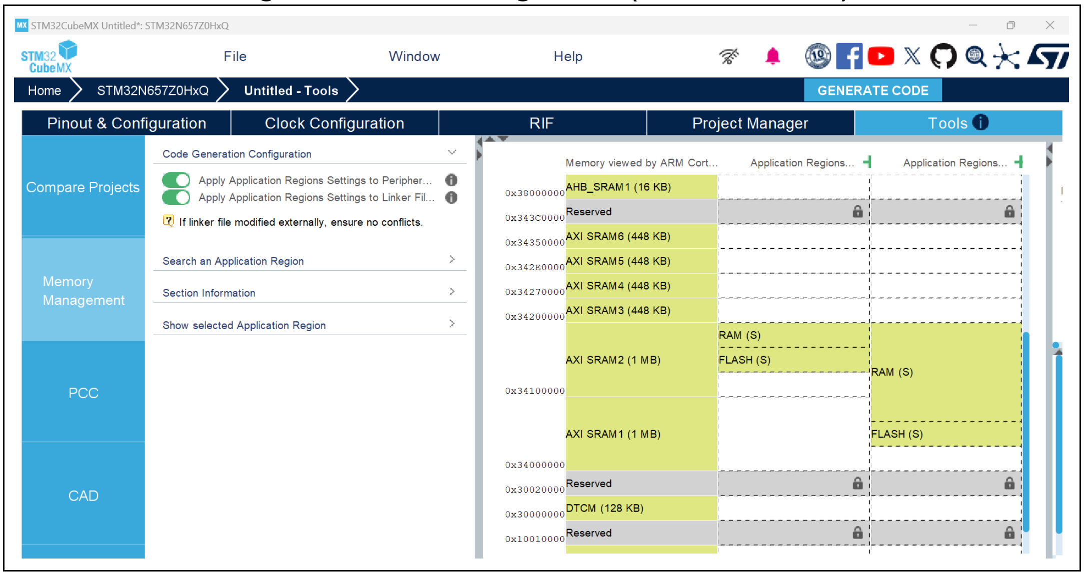


Figure 465. MMT configuration (secure domains)



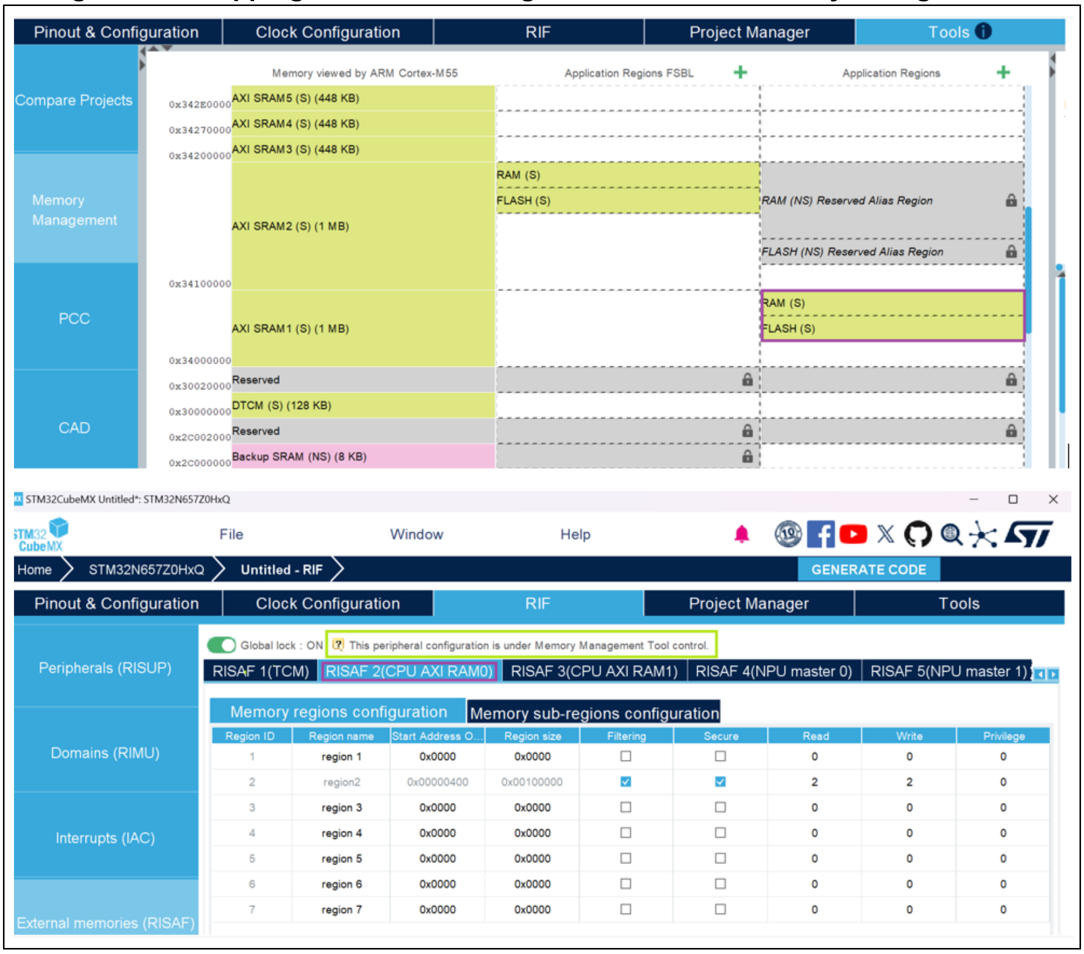
4. Select the Project Manager tab
5. Give a name to the project and press the Generate Code button

Changes in RIF resulting from MMT activation

By selecting “Apply Application region Settings to Peripherals ON,” the RISAF feature (located in RIF panel) is put partially under MMT.

The RAM and flash memory regions are configured with the same security, read/write, and privilege parameters. They are represented as a single RISAF region in the view corresponding to RISAF 2 (CPU AXI RAM0).

Figure 466. Mapping between MMT regions and RISAF memory configurations



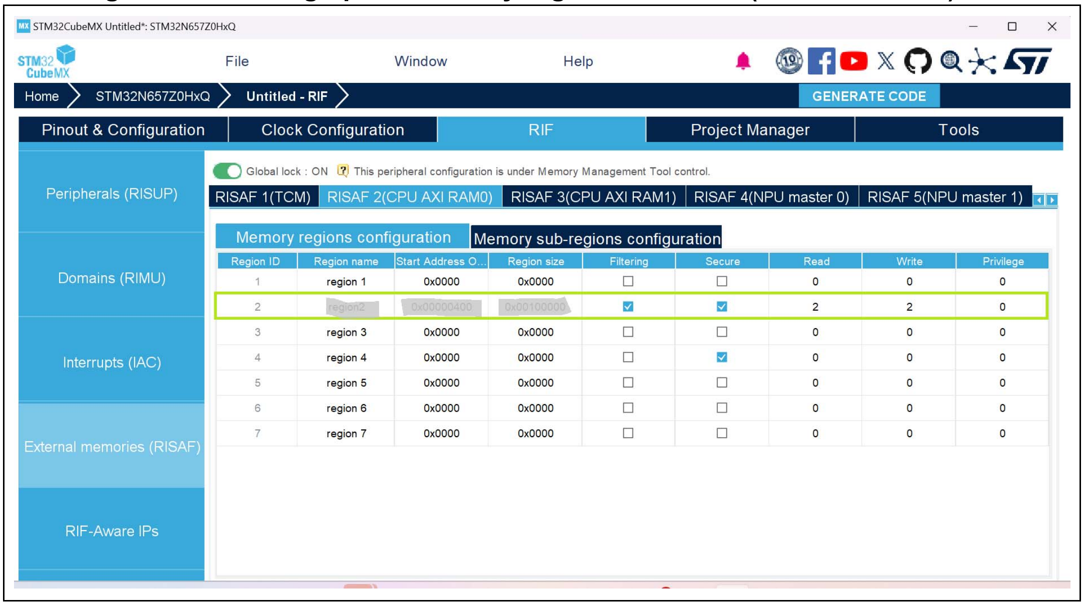
Configuration of memory regions in RISAF2 (CPU AXI RAM0) tab (region 2)

Under the RISAF2 (CPU AXI RAM0) tab, region 2 has two groups of settings:

- Fixed fields (grayed out): Region ID, Region Name, Start address Offset, Region size, and Filtering are read-only and cannot be modified.
- Controllable fields (editable): The security and access parameters (Secure, Read, Write, and Privilege) are editable and are updated by the (MMT).

The values that MMT can set for the Read, Write, or Privilege fields are 2 (00000010b) and 0 (00000000b).

Figure 467. Setting up the memory region of RISAF2 (CPU AXI RAM0) tab

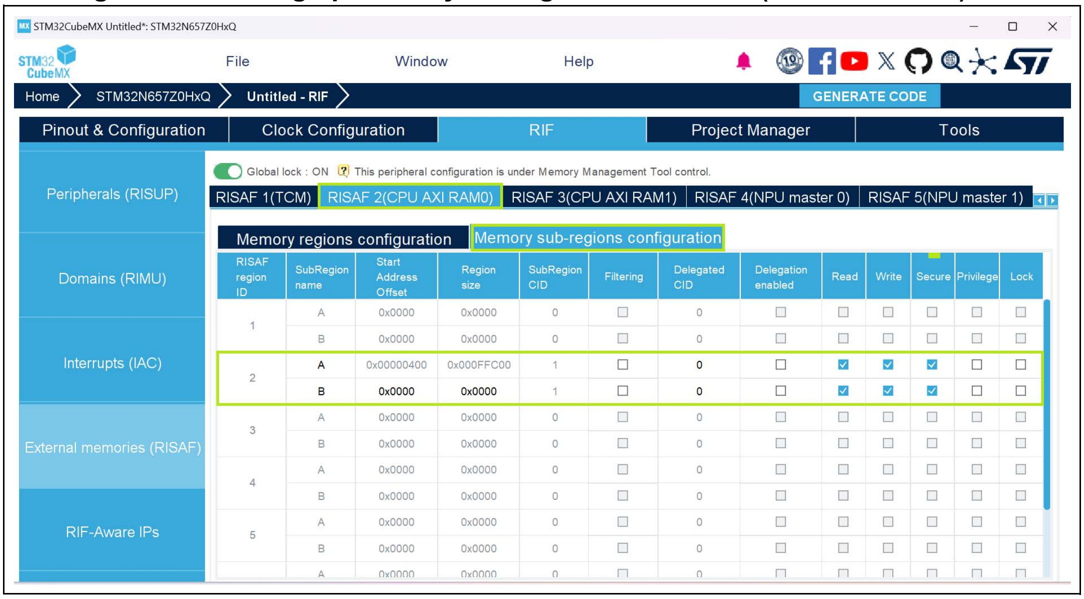


Setting up memory subregions for RISAF 2 (CPU AXI RAM0) region 2

The Sub-Region A has two groups of settings:

- Fixed fields (grayed out): RISAF Region ID, SubRegion name, Start address Offset, Region size, SubRegion CID (fixed to 1) are read-only and cannot be modified.
- Controllable fields (editable): the security and access parameters (Secure, Read, Write, and Privilege are editable and are updated by the MMT.

Figure 468. Setting up memory subregions of RISAF 2 (CPU AXI RAM0) tab



On RISAF1 (TCM) tab, when the user adds an application region in ITCM or DTCM ranges, MMT adds automatically memory regions in RISAF1 (TCM) with the same start address as in the MMT layout.

Figure 469. Setting up memory regions of RISAF1 (TCM) tab

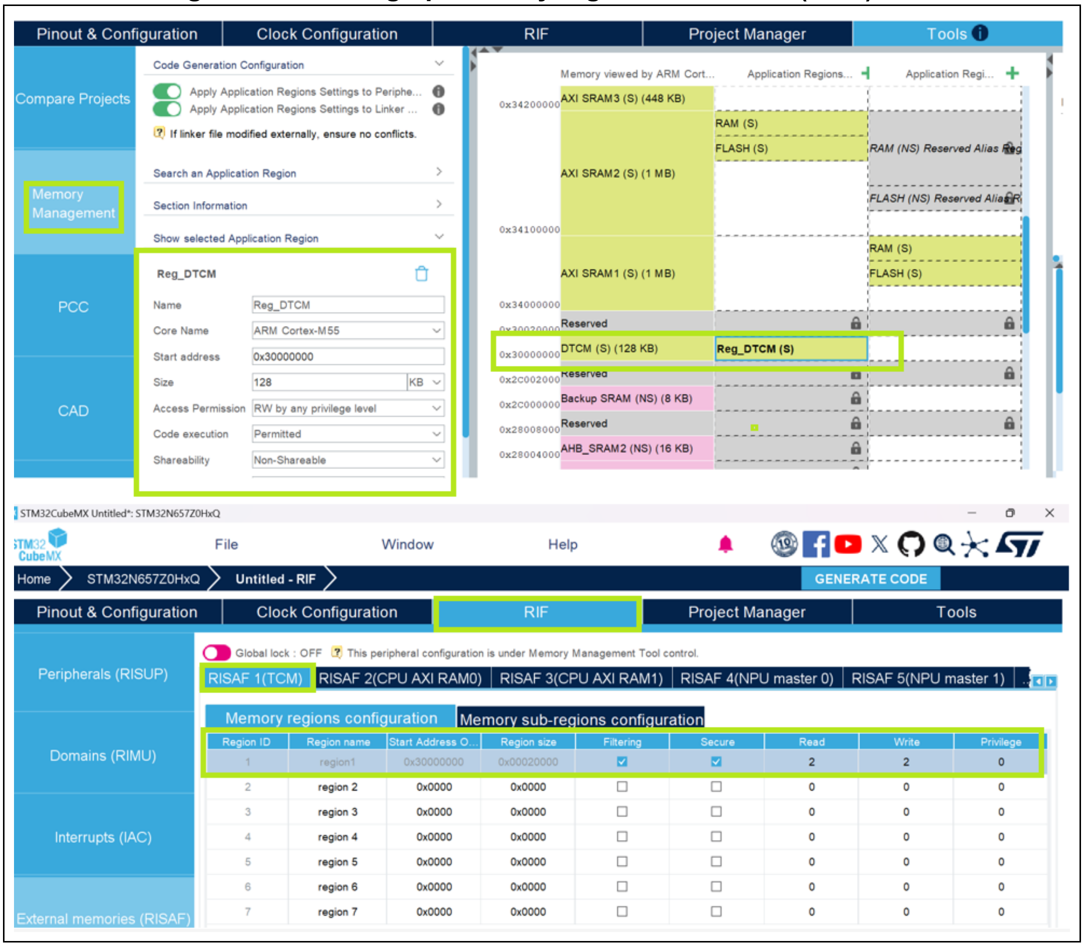
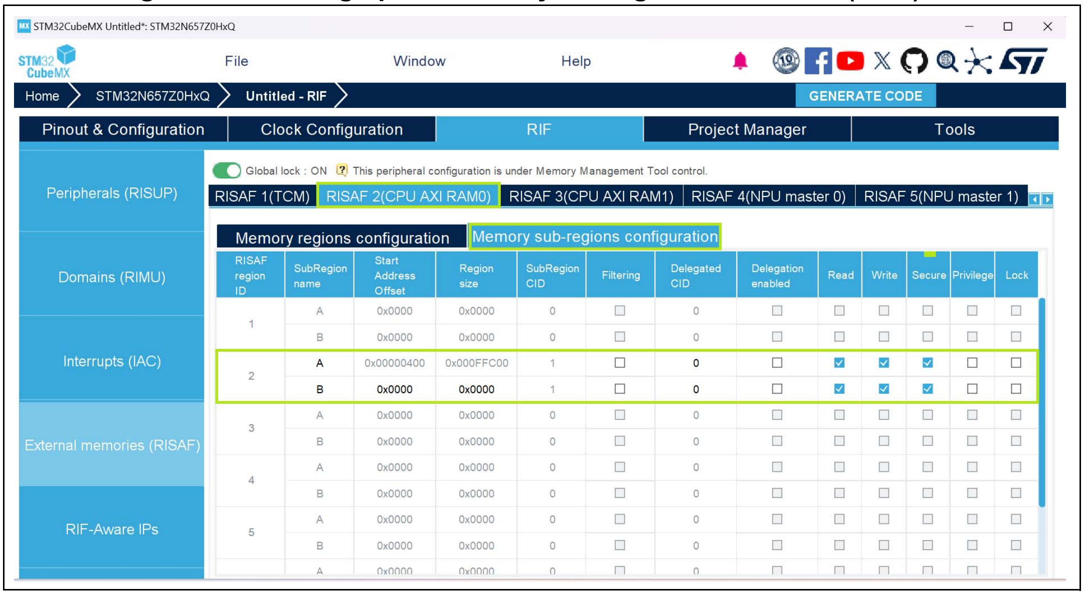


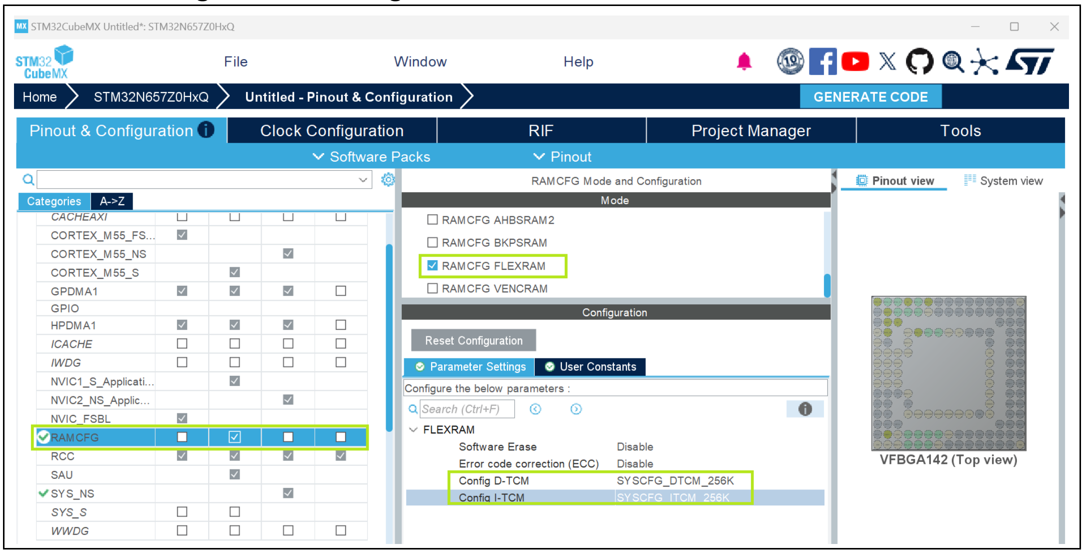
Figure 470. Setting up the memory subregions of RISAF1 (TCM) tab



On RISAF 7 (FLEXRAM) tab, when the user adds an application region in ITCM or DTCM ranges MMT adds automatically in RISAF 7 (FLEXRAM) memory regions with the same start address as in the MMT layout.

Use the configuration shown in *Figure 471* for the right mapping between MMT settings and RISAF 7 (FLEXRAM) memory regions.

Figure 471. Configurations needed for RAMCFG FLEXRAM



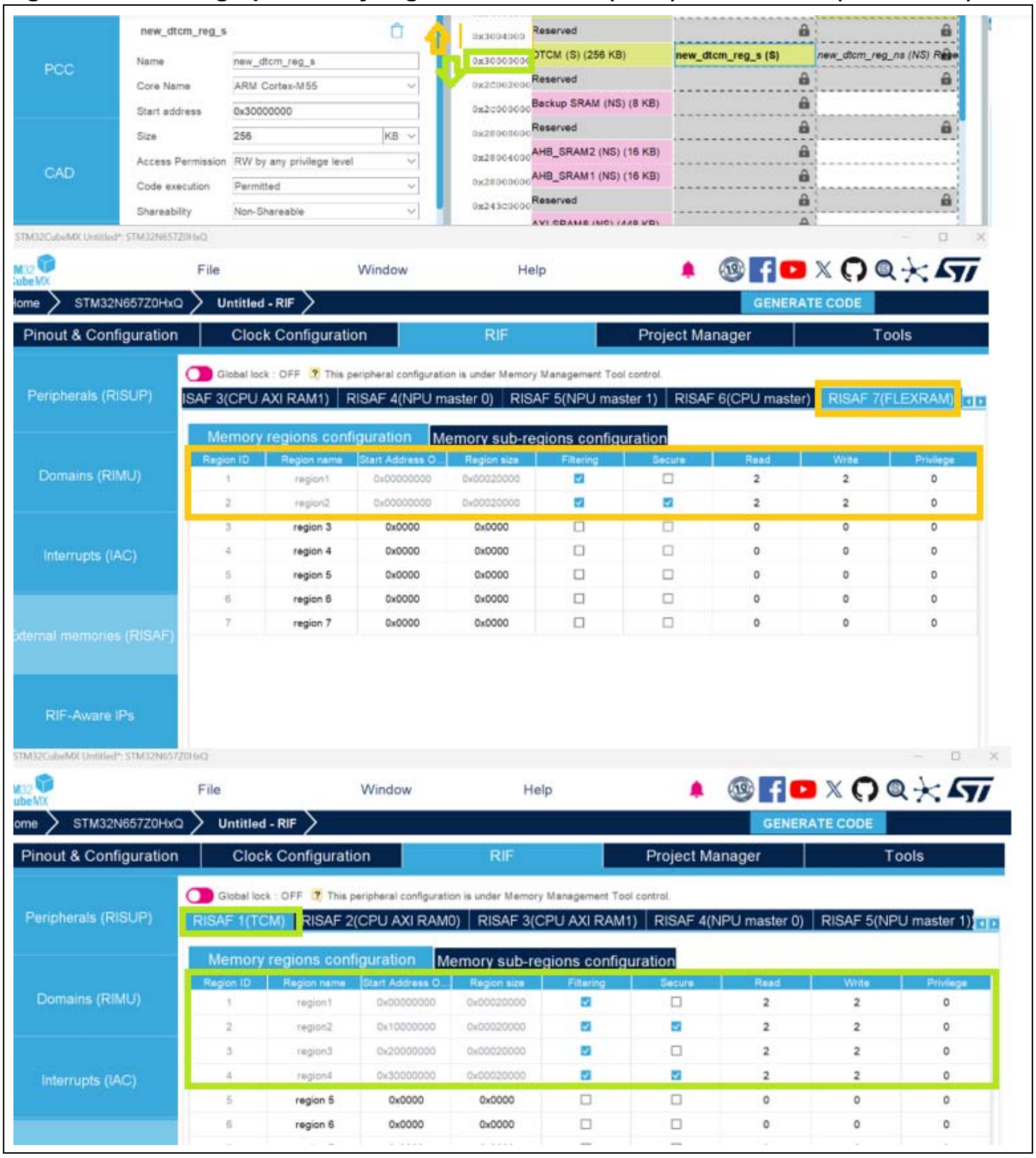
When FLEXRAM (RAMCFG) is active, not utilizing its default settings, and the user adds an application region in ITCM or DTCM ranges, the memory split is specifically defined. Example of configurations:

- D-TCM is set to SYSCFG_DTCM_256K
- I-TCM is set to SYSCFG_ITCM_256K

When adding a 256 KB application region that overlaps the ITCM and DTCM ranges:

- The configuration of RISAF1 (TCM) remains consistent for its allocated portion.
- Any memory belonging to that region above 0x340020000 is automatically redirected (or split) and configured under RISAF7 (FLEXRAM). This segment starts at 0x340020000 and is treated as the beginning 0x00000000 offset of the RISAF7 region.

Figure 472. Setting up memory regions of RISAF1 (TCM) and RISAF7 (FLEXRAM) tabs

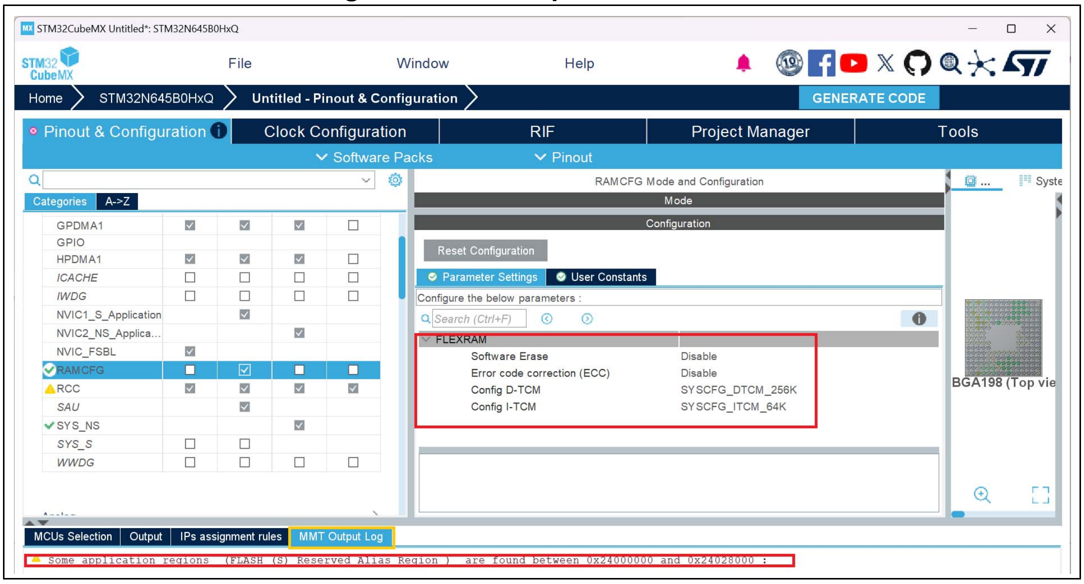


When the user applies a custom FLEXRAM configuration, such as:

- D-TCM to SYSCFG_DTCM_256K, I-TCM to SYSCFG_ITCM_64K, or
- D-TCM: SYSCFG_DTCM_256K, I-TCM: SYSCFG_ITCM_128K, or
- D-TCM: SYSCFG_DTCM_256K, I-TCM: SYSCFG_ITCM_256K

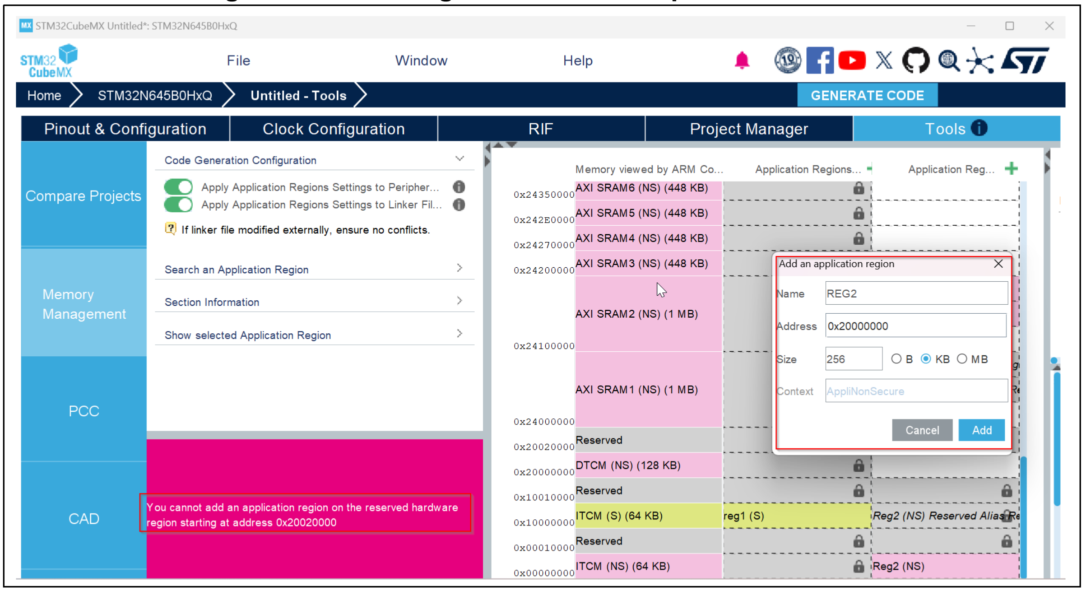
a warning appears, indicating that the flash memory region overlaps with the memory area reserved to FLEXRAM.

Figure 473. Overlap with FLEXRAM



To avoid the warning, locate the flash memory region to another range, and update the configuration of FLEXRAM.

Figure 474. Warning in case of overlap with FLEXRAM

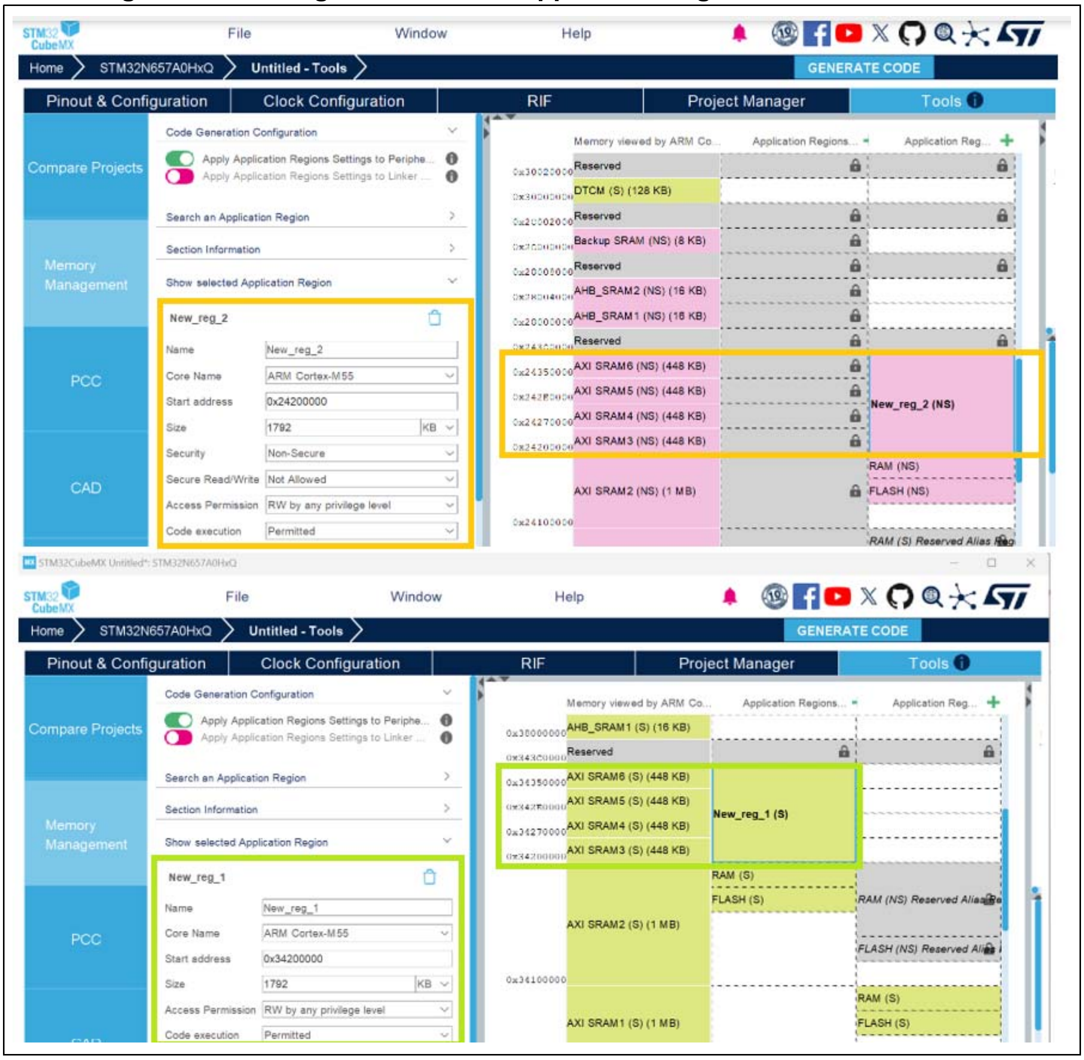


On RISAF4 (NPN master1), RISAF5 (NPN master2), and RISAF6 (CPU master) tabs RISAF4/5 are available.

Configuration: add two application regions:

- Region 1: start address 0x34200000, size 1792 KB in FSBL (S) context
- Region 2: start address 0x24200000, size 1792 KB in Appli NS context

Figure 475. Configurations of the application regions on the MMT view



The MMT adds:

- Two RISAF regions in RISAF4 (secure region, nonsecure region), see *Figure 476*
- Two RISAF regions in RISAF5 (secure region, nonsecure region), see *Figure 477*
- Two RISAF regions in RISAF6 (CPU master), see *Figure 478*

Figure 476. Setting up memory regions of RISAF4 (NPU master 0)

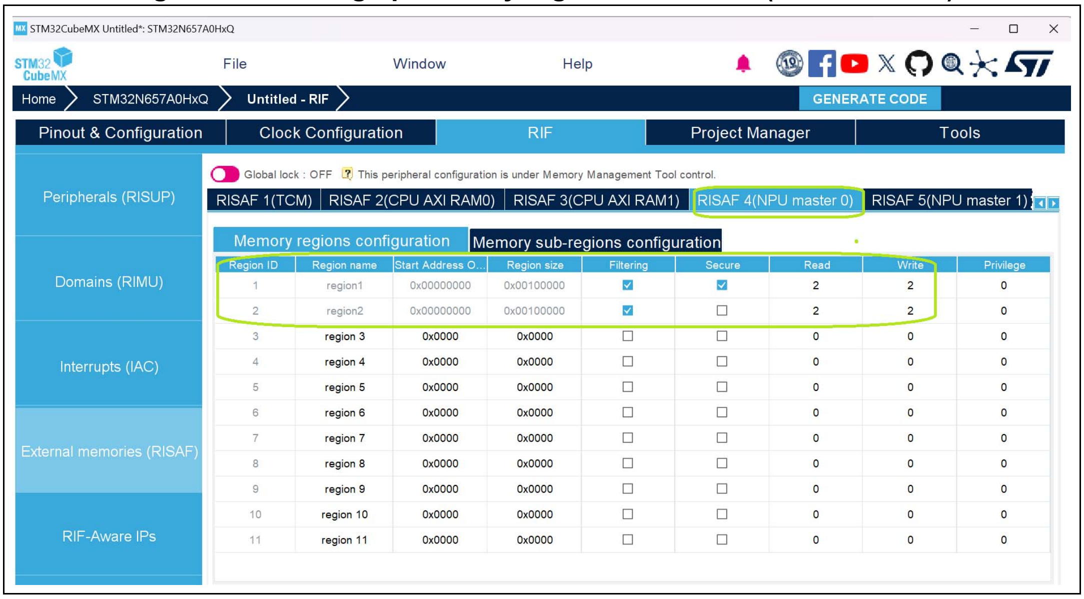


Figure 477. Setting up memory regions of RISAF5 (NPU master 1)

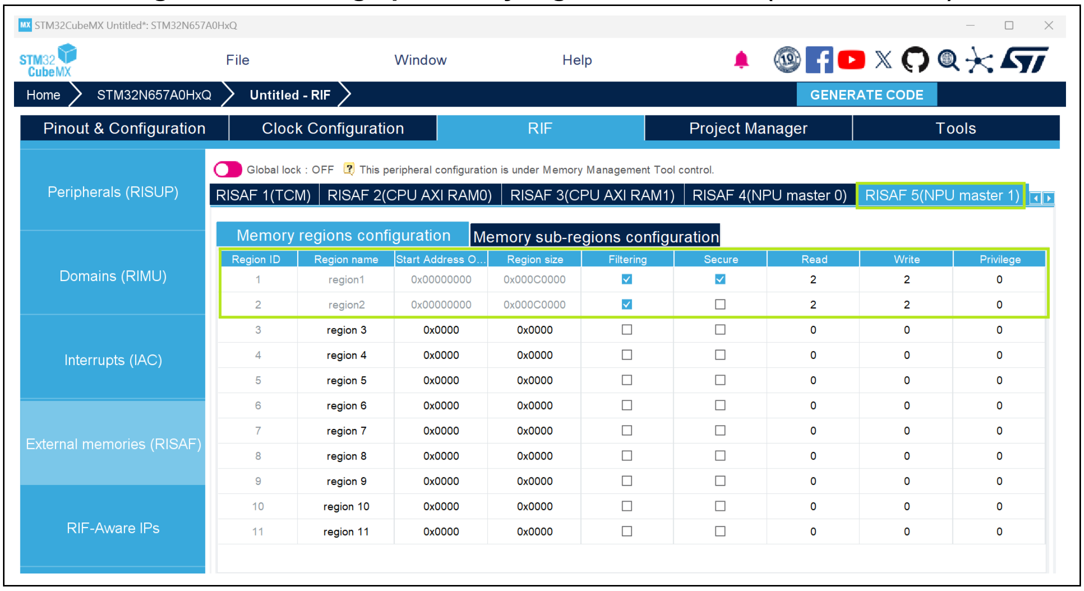
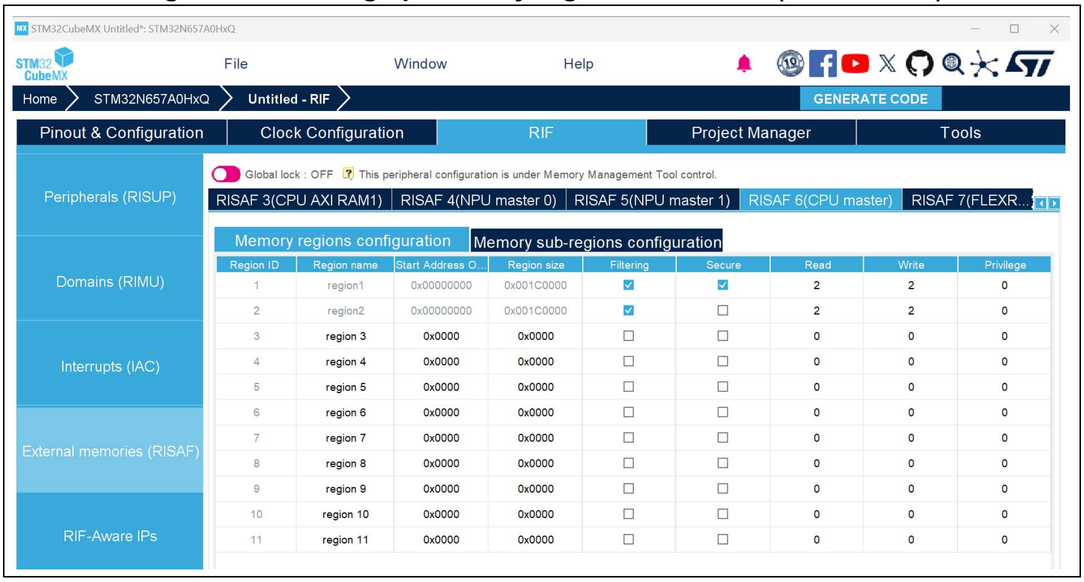


Figure 478. Setting up memory regions of RISAF6 (CPU master)



The two secure regions in RISAF4 and RISAF5 are consolidated into a larger secure region, identified and managed in RISAF6. The two nonsecure regions in RISAF4 and RISAF5 are consolidated into a larger nonsecure region, identified and managed in RISAF6.

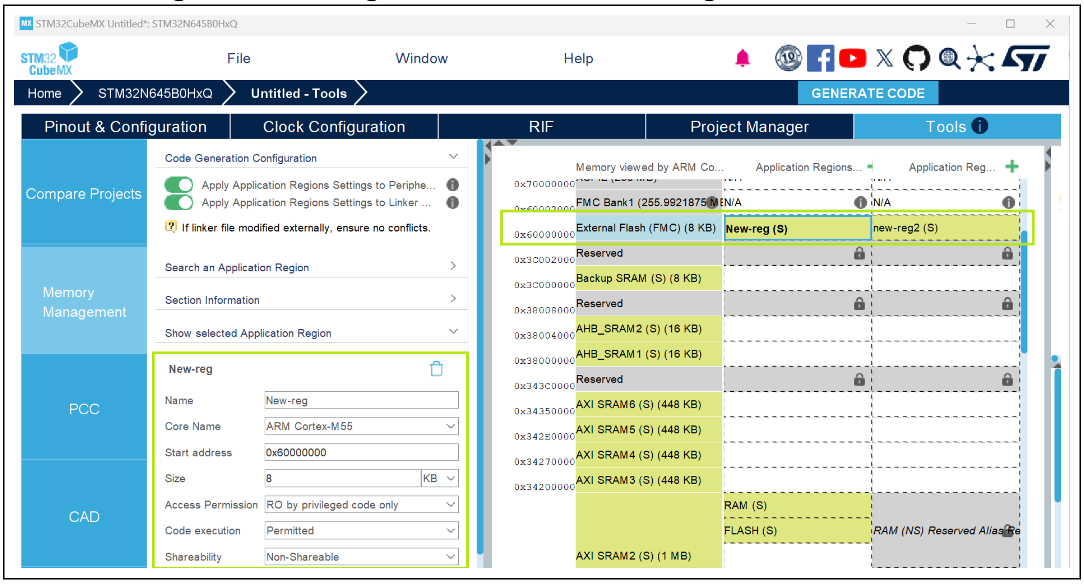
On RISAF14(FMC) tab the available FMC (Flexible Memory Controller) ranges include:

- FMC Bank 1
- FMC SDRAM Bank 1
- FMC SDRAM Bank 2

Configuration:

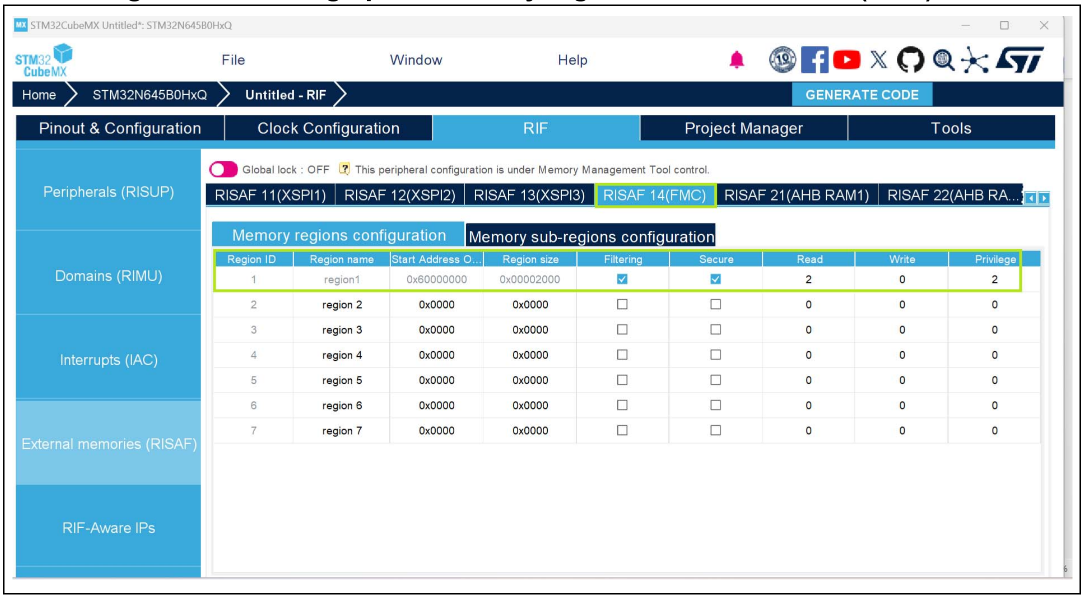
- Configure the FMC IP
- Add application regions in FMC ranges

Figure 479. Configurations of FMC added regions in MMT view



MMT adds in RISAF14 (FMC) regions with the same start address as in MMT view (no offset).

Figure 480. Setting up the memory regions for the RISAF14 (FMC) tab

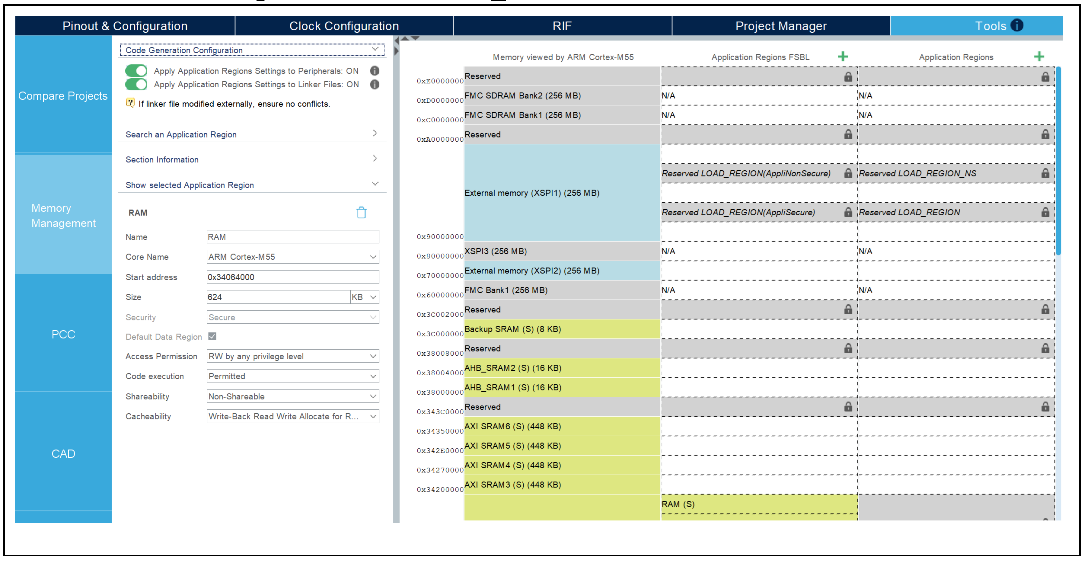


EXTMEM_MANAGER

Code generation configuration

- The application regions settings can be applied to peripherals on the left-hand side of the screen. The concerned peripherals are shown on the associated tooltip.
- The MMT configuration may impact their availability on the pinout screen configuration.
- The EXTMEM_MANAGER can be used with the “Select boot code generation” parameter enabled or disabled.
  – If disabled, MMT automatically chooses the configured memory along with the associated driver and sets the execution memory location in the linker file. This is the most straightforward way of configuring an external memory.
  – If enabled, by activating the “Select boot code generation” you can choose “Execute in Place” or “Load and Run”

Figure 481. EXTMEM_MANAGER in the MMT view



The activation of the EXMEM_MANAGER depends upon the XSPIx activation.

1. Go to the Pinout and Configuration tab
2. Enable XSPI1 for the FSBL context and choose “Single SPI” mode
3. Activate the middleware EXTMEM_MANAGER in the FSBL context
4. Activate XSPIM and XSPI instances (XSPI1/2/3)

Figure 482. Setting up the XSPI feature

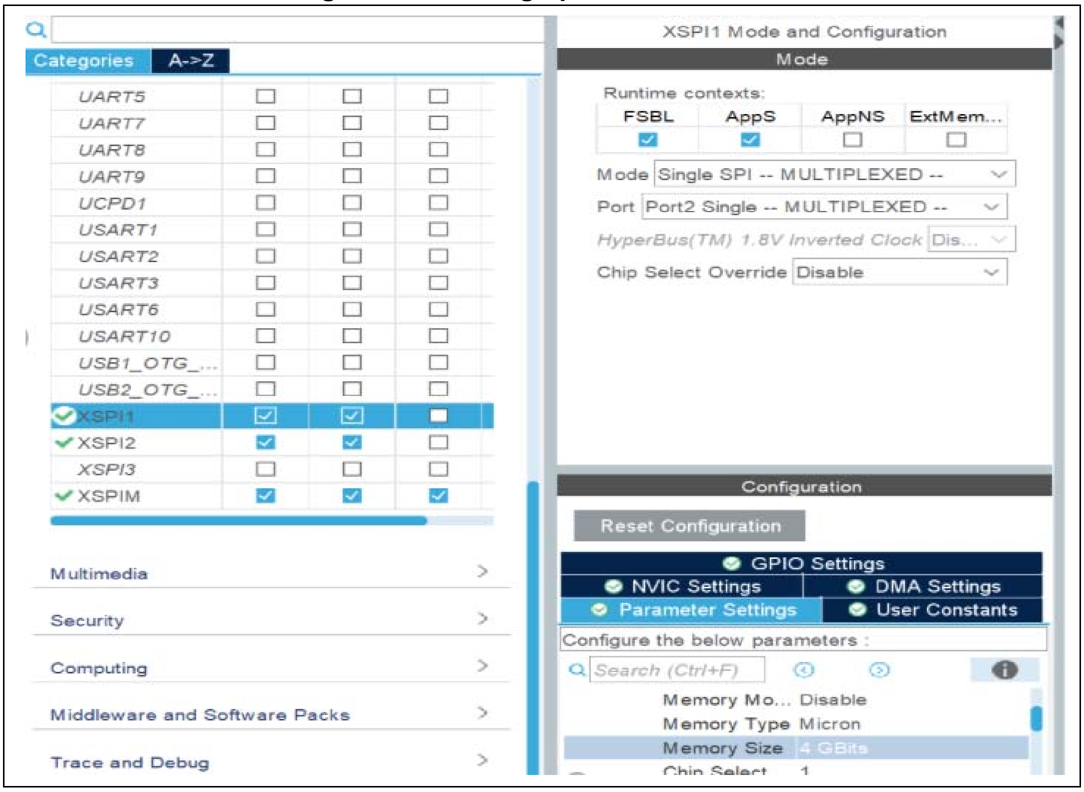
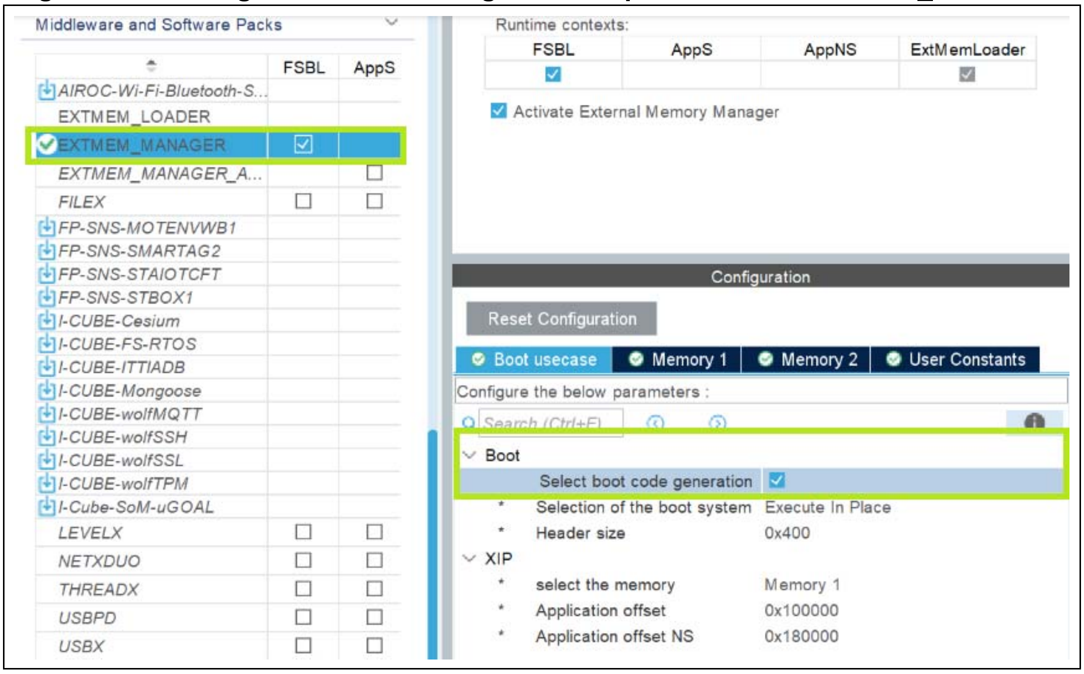


Figure 483. Setting Select boot code generation parameter for EXTMEM_MANAGER



For the selection of the boot system there are two possibilities:

1. Execute In Place (*Figure 484*) chooses and configures the memory zones.
2. Load and Run (*Figure 486*) lets the user choose source, destination memory, and addresses to jump to.The configuration is translated into the linker file. The user must provide the source and destination address.

Figure 484. Selection of the boot system - Execute In Place

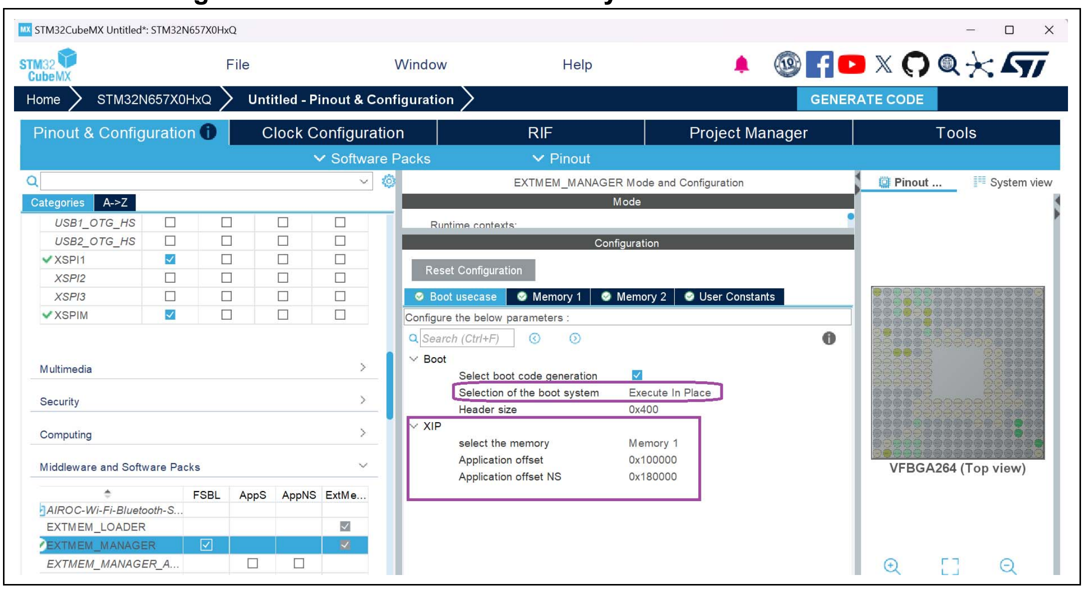


Figure 485. Configuration of the MMT view after enabling XIP

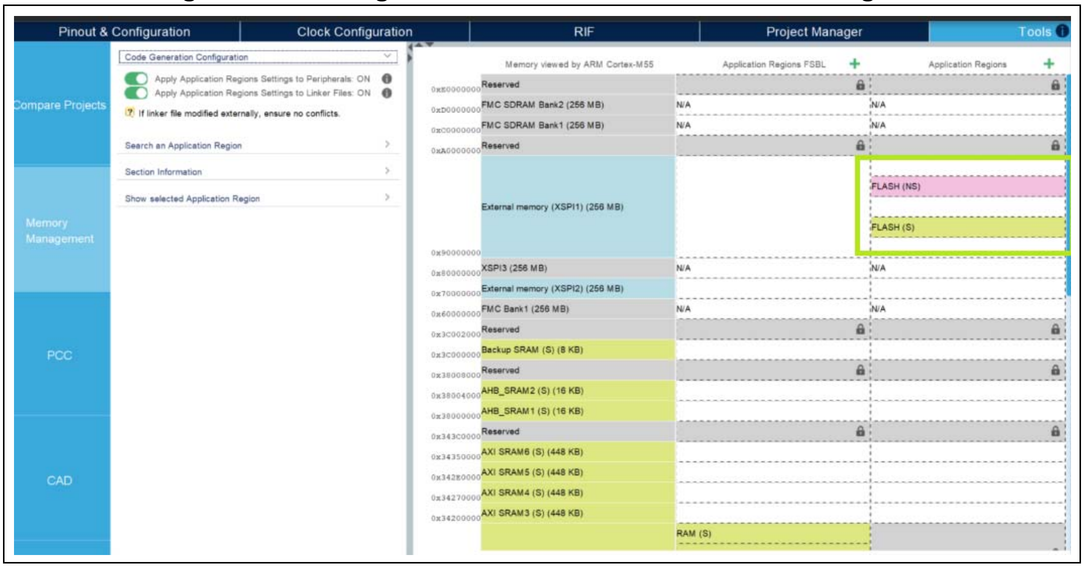


Figure 486. Selection of the boot system - Load and Run

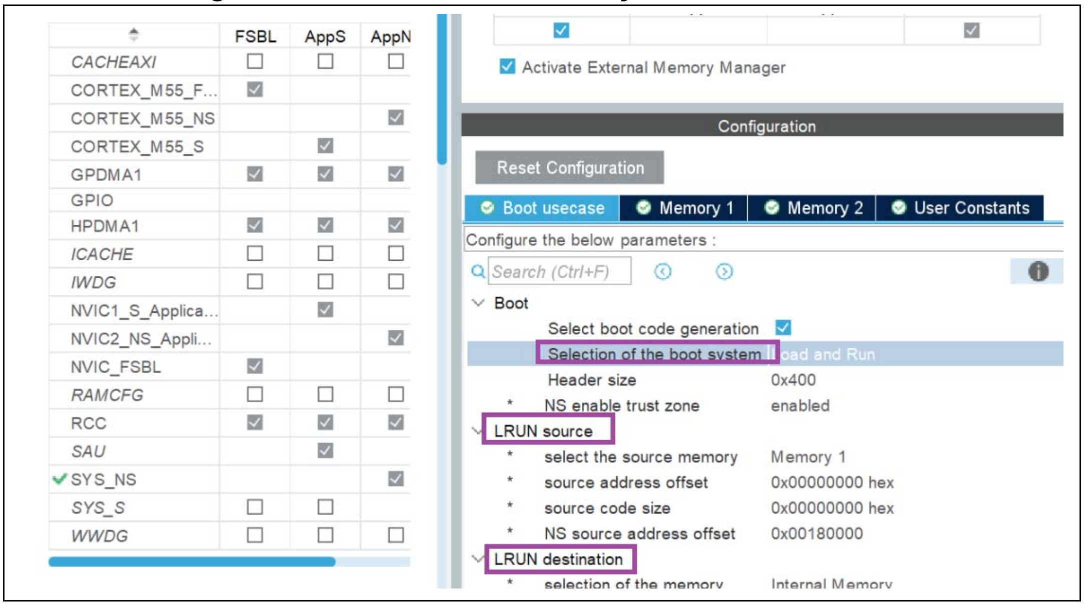
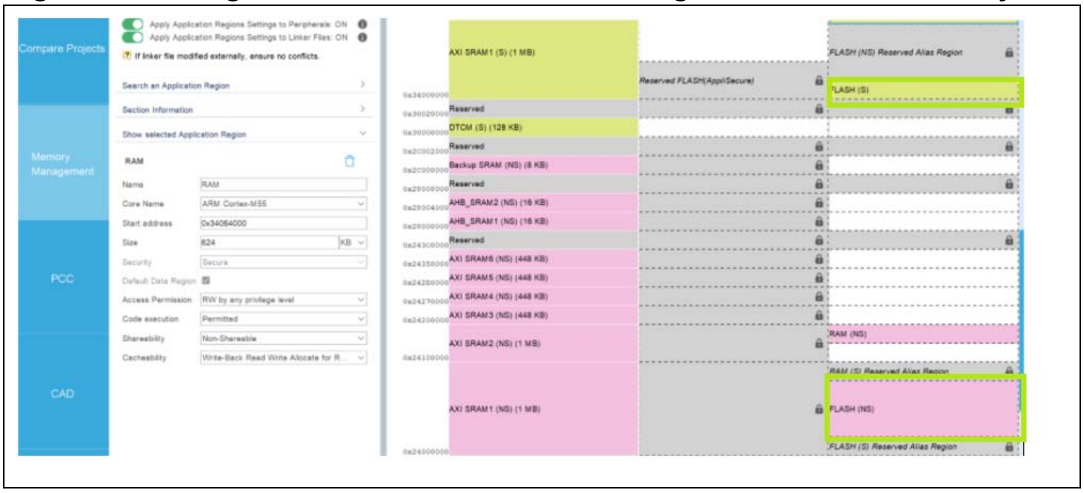


Figure 487. Configuration of MMT view after choosing Load and Run as boot system

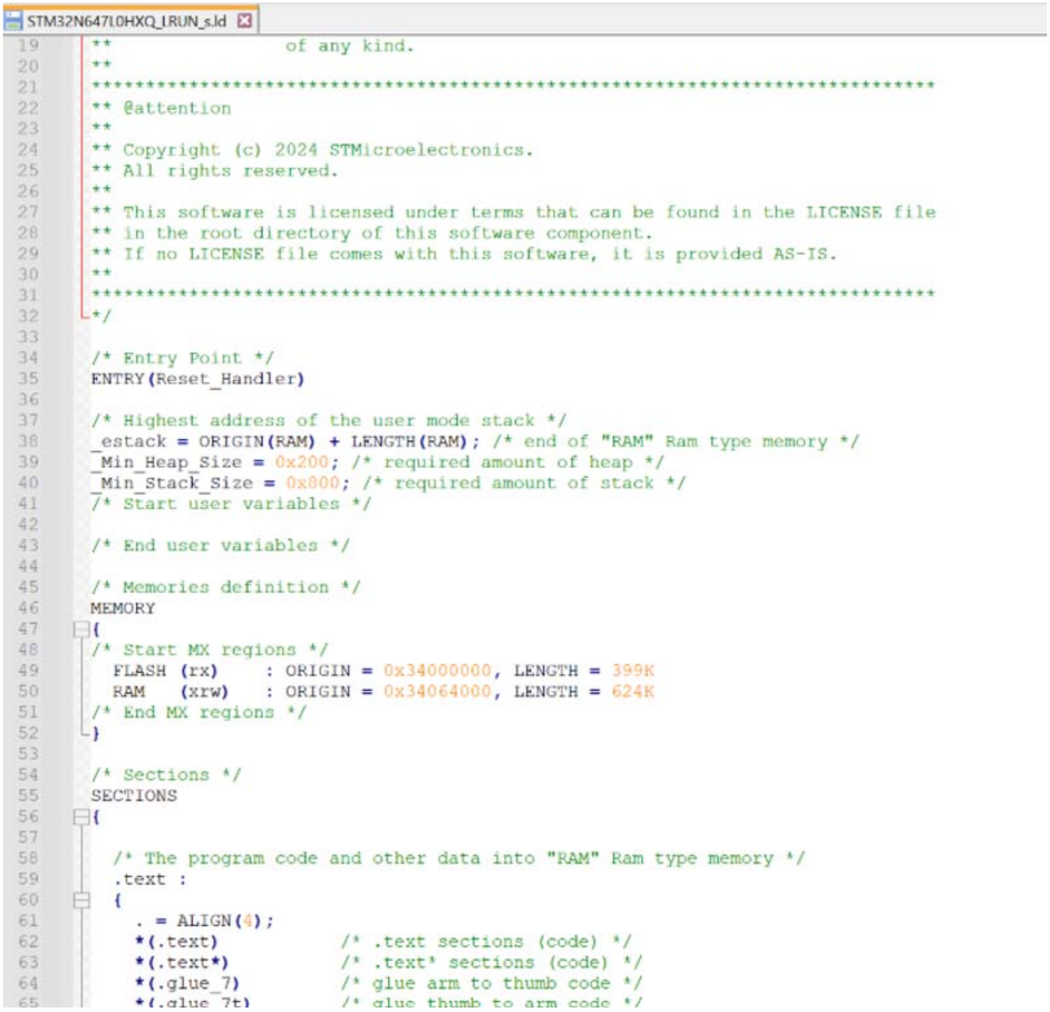


Once the code is generated, follow these steps:

1. Navigate to the generated project folder.
2. In the boot Project directory, open the linker definition file (usually an .ld or .icf file).
3. In the Memories definition section you can review the memory blocks (start addresses and lengths) that reflect your configuration settings.

Figure 488. Defined linker script for XIP

Figure 489. MMT linker file after choosing Load and Run as boot system



STM32N647L0HXQ_LRUN_s.ld

```
**                  of any kind.
**
******************************************************************************
** @attention
**
** Copyright (c) 2024 STMicroelectronics.
** All rights reserved.
**
** This software is licensed under terms that can be found in the LICENSE file
** in the root directory of this software component.
** If no LICENSE file comes with this software, it is provided AS-IS.
**
******************************************************************************
*/

/* Entry Point */
ENTRY(Reset_Handler)

/* Highest address of the user mode stack */
_estack = ORIGIN(RAM) + LENGTH(RAM); /* end of "RAM" Ram type memory */
_Min_Heap_Size = 0x200; /* required amount of heap */
_Min_Stack_Size = 0x800; /* required amount of stack */
/* Start user variables */

/* End user variables */

/* Memories definition */
MEMORY
{
/* Start MX regions */
  FLASH  (rx)     : ORIGIN = 0x34000000, LENGTH = 399K
  RAM    (xrw)    : ORIGIN = 0x34064000, LENGTH = 624K
/* End MX regions */
}

/* Sections */
SECTIONS
{

  /* The program code and other data into "RAM" Ram type memory */
  .text :
  {
    . = ALIGN(4);
    *(.text)           /* .text sections (code) */
    *(.text*)          /* .text* sections (code) */
    *(.glue_7)         /* glue arm to thumb code */
    *(.glue_7t)        /* glue thumb to arm code */
```

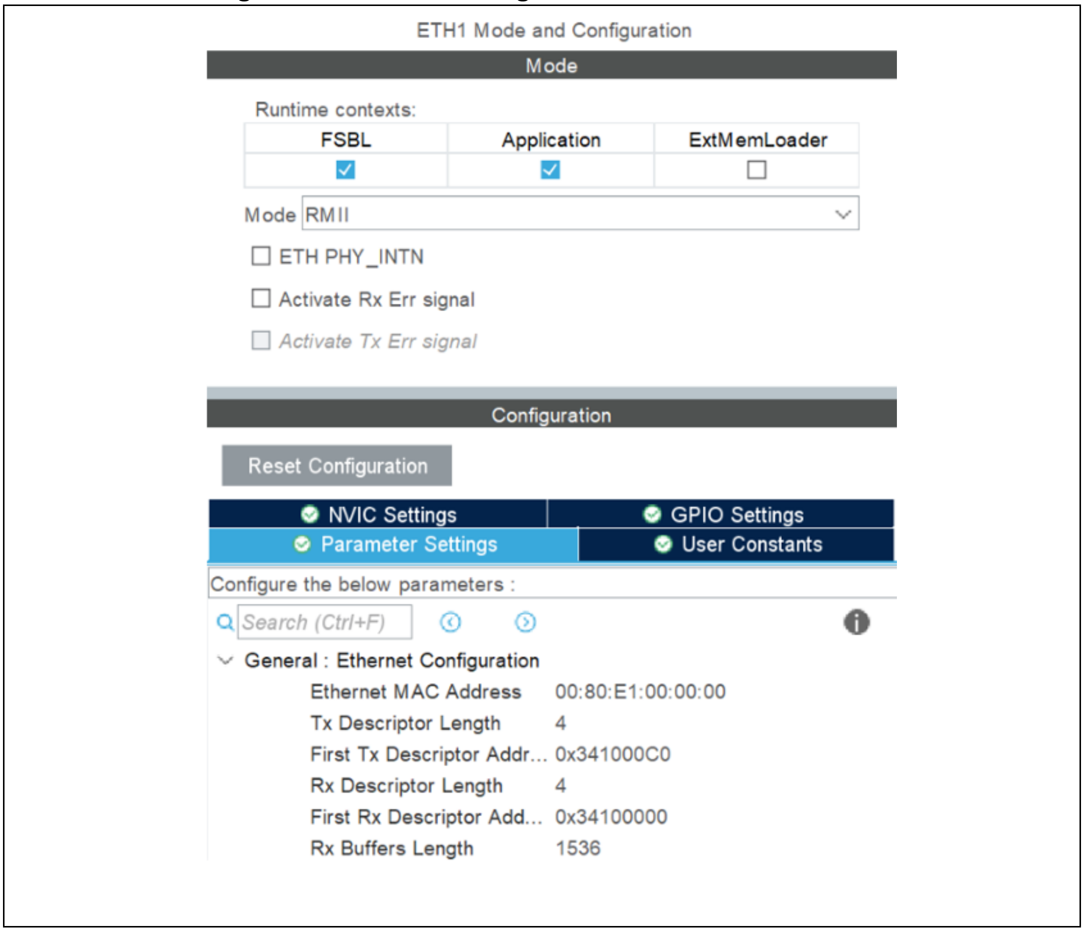
Ethernet

Example of MMT configuration for the ETH IP on the STM32N645B0HxQ MCU

Configure the ETH:

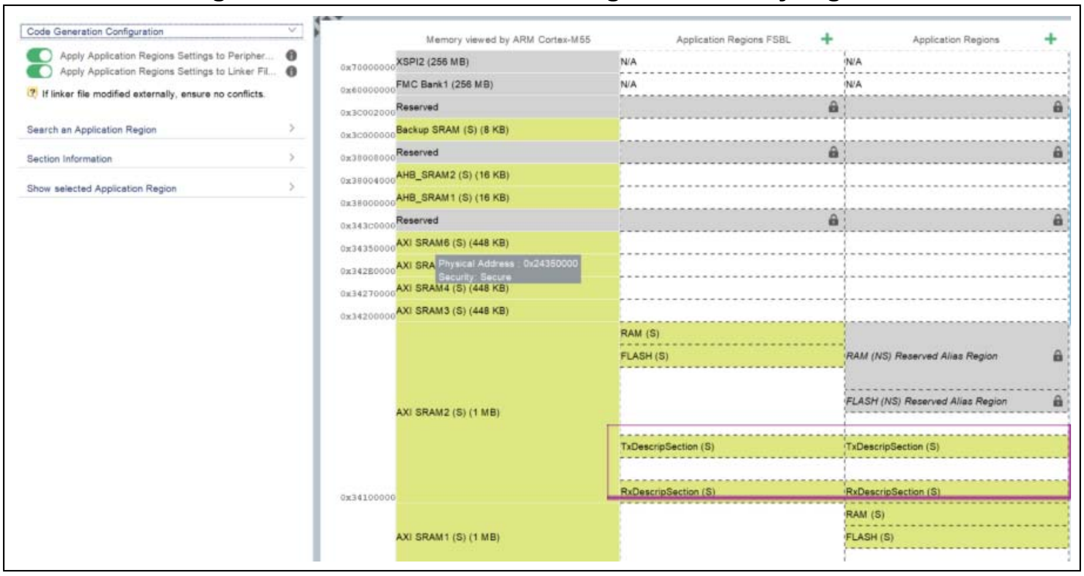
- For secure domains:
  - Activate the ETH: MMT creates three application regions for each context

Figure 490. ETH MMT regions for STM32N647L0HxQ



– Update the start address and the size of each region, update the ETH parameters.
– Press the radio button “Apply Application region Settings to Peripherals ON”, to make ETH partially under MMT control (*Figure 491*).
– Press the Generate Code button to generate code for both applications.
– Apply Application Regions settings to linker files:

Figure 491. MMT view after adding ETH memory regions



When this button is on, the linker scripts are generated, considering the configuration. After the code generation, navigate to the generated folder:

– Open the linker definition file.

Figure 492. ETH linker file (fully secure domain only)

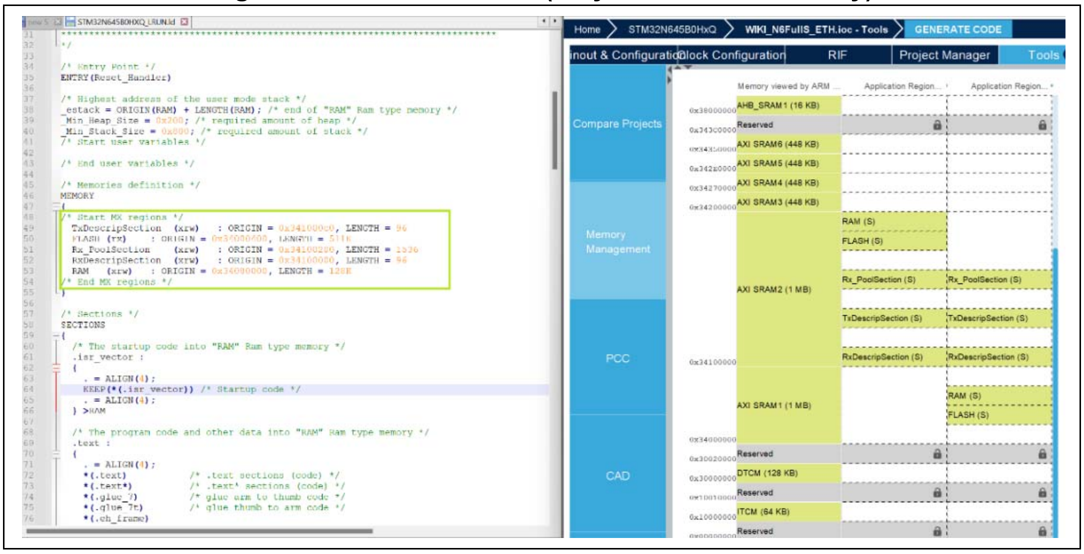
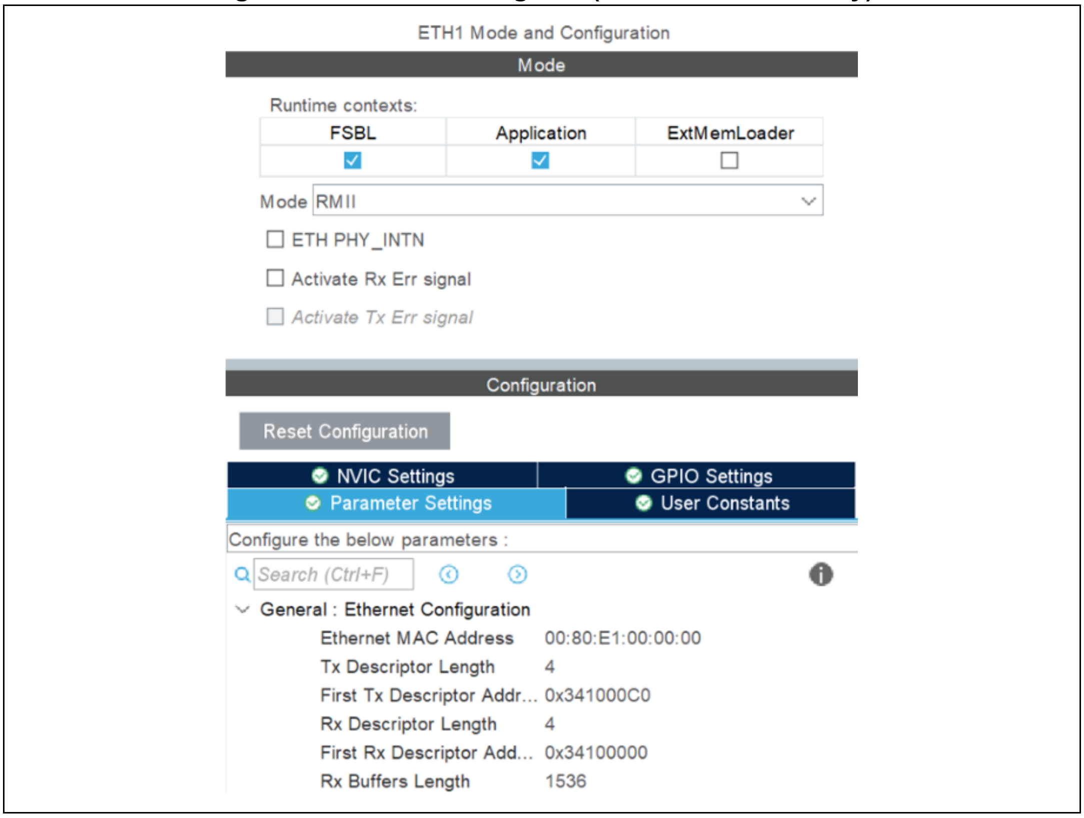
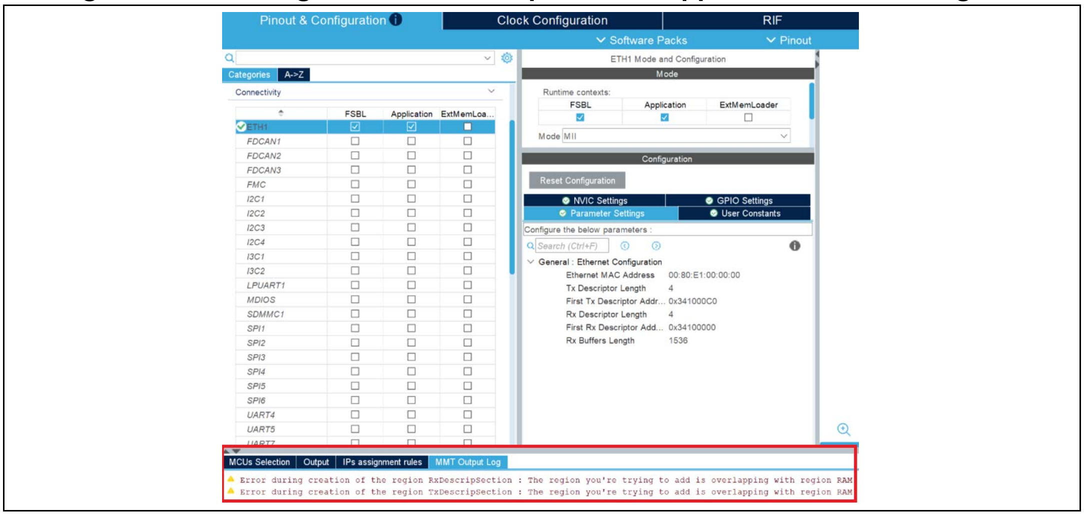


Figure 493. ETH MMT regions (secure domains only)



– Under the Memory definition you can see the memory parts with their start address and length, according to the configuration made in STM32CubeMX. If an application region overlaps an ETH region a warning is displayed in the MMT log.

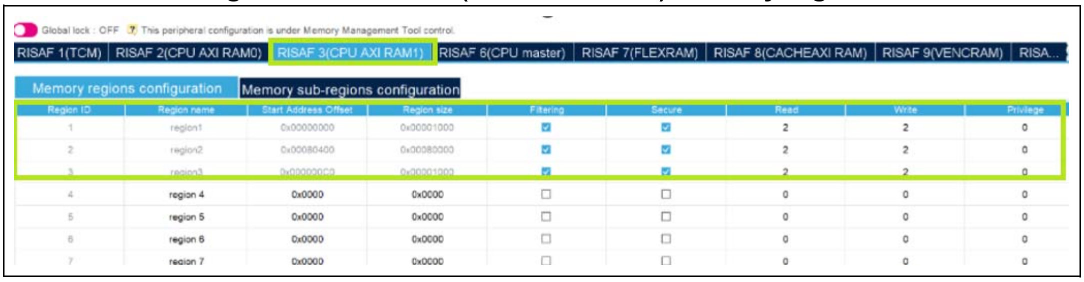
Figure 494. Warning in case of overlap between application and ETH regions



– The default parameters are used for the secure context, but if the selected context is nonsecure, other nonsecure parameters are displayed.

Impact of configuring the ETH memory regions with ETH on RISAF panels

Figure 495. RISAF 3 (CPU AXI RAM1) memory regions



For secure and nonsecure domains ETH application regions are created for the selected contexts.

Figure 496. ETH application regions for secure and nonsecure domains (part 1)

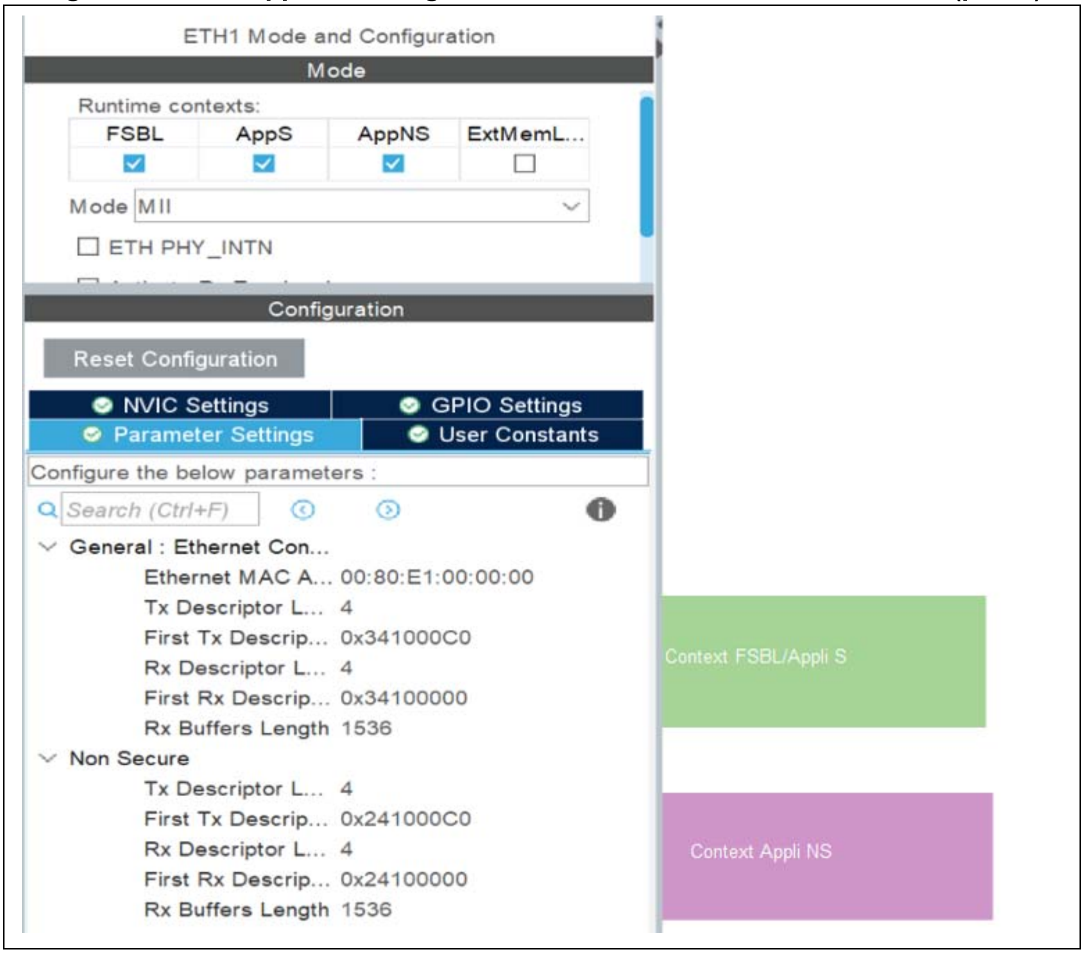
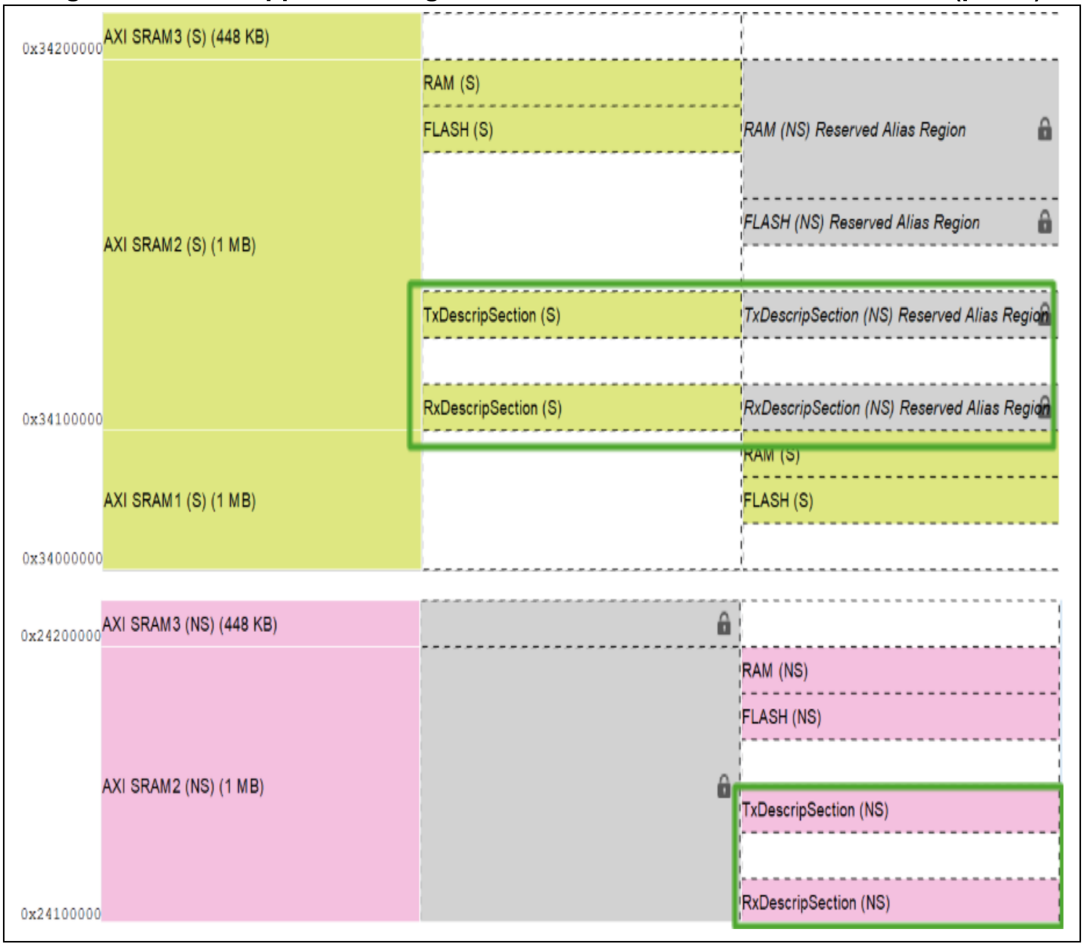


Figure 497. ETH application regions for secure and nonsecure domains (part 2)



5.5.9 Notification MMT/boot path (STM32H7RS and STM32H5)

After the activation of boot path and MMT, all regions of MMT are deleted and replaced by the regions of Boot path in Appli context.

In this example, we use the boot path OEM-iRoT for STM32H7RS and for STM32H5.

Figure 498. MMT/boot path (STM32H7RS)

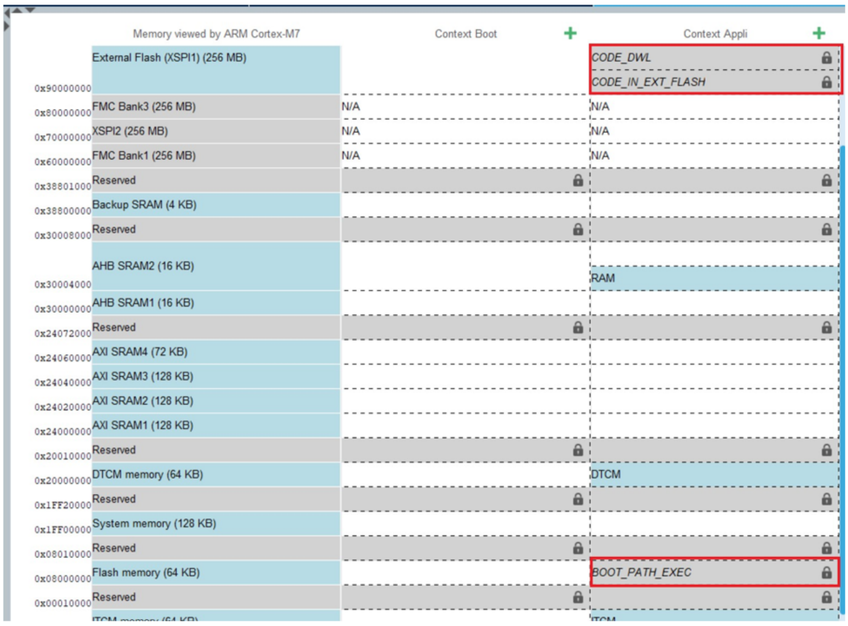
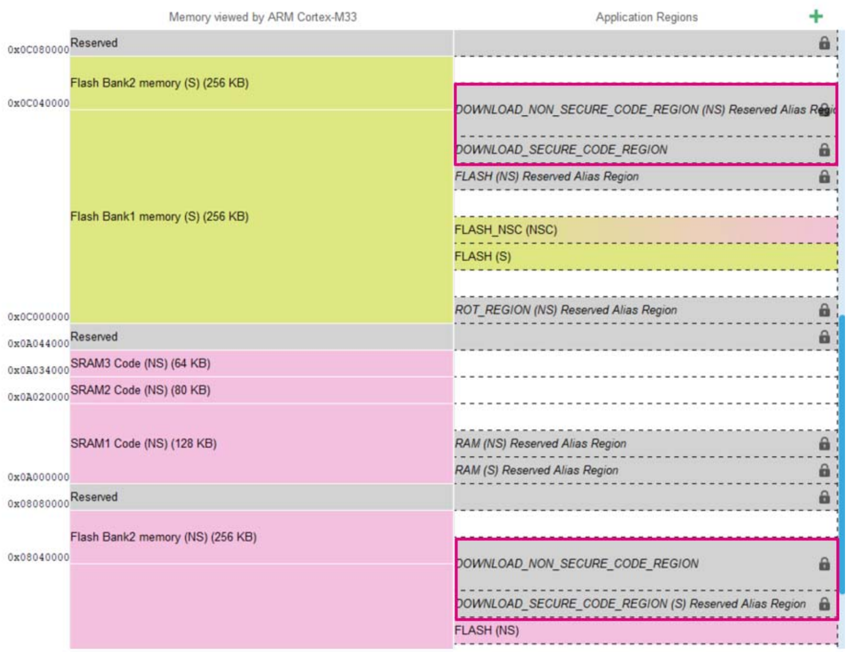


Figure 499. MMT/boot path (STM32H5)



The linker files are copied from STM32Cube firmware of boot path, and MMT integrates all added application regions (App_User).

- Open the linker files STM32H7S3I8KX_OEMiROT_Appli_app.ld or STM32H523CETX_FLASH.ld (respectively, left or right side of *Figure 500*)
- Look at the memory definition: check the App_User declaration in the Appli project in case of an OEM-iRoT boot path (see *Figure 501* and *Figure 502*).

Figure 500. Linker files location (STM32H7RS on the left, STM32H5 on the right)

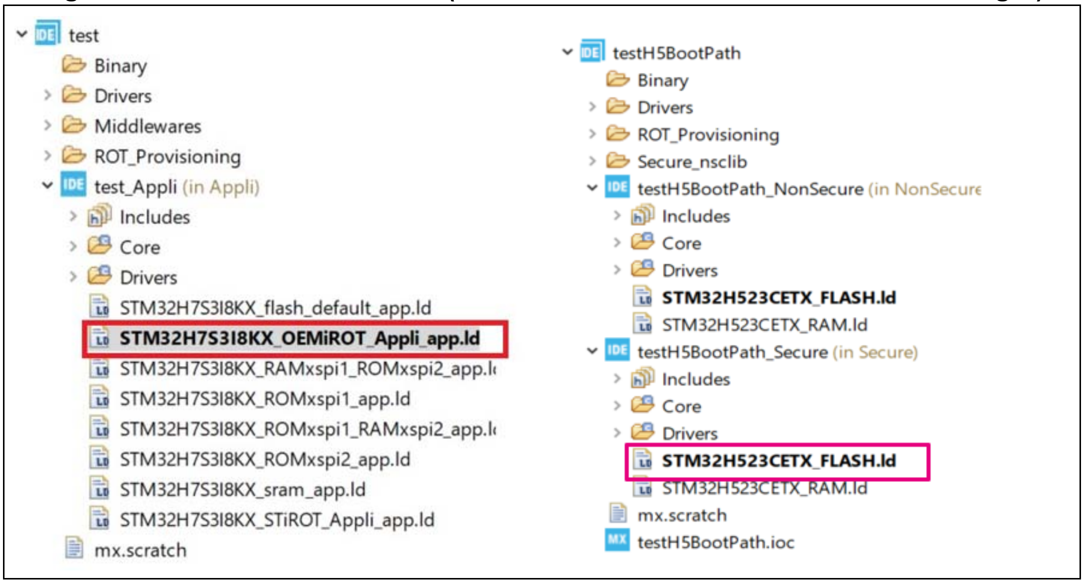


Figure 501. App_User declaration (STM32H7RS)

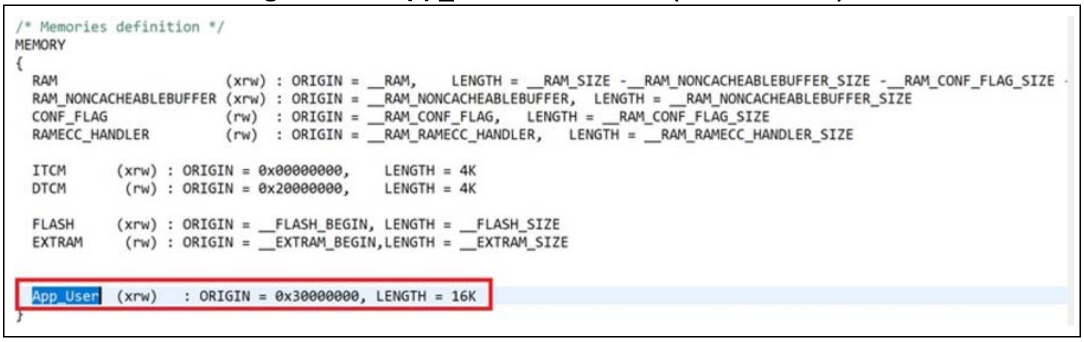
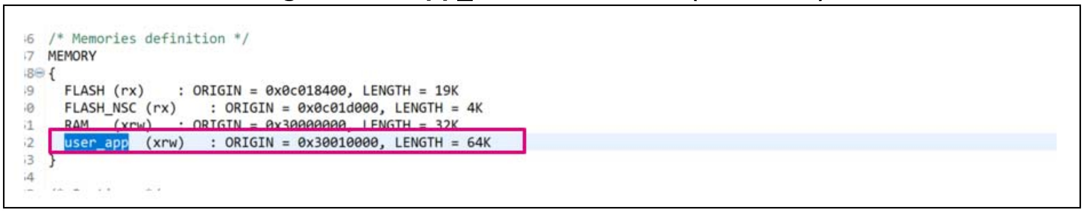


Figure 502. App_User declaration (STM32H5)



6 STM32CubeMX C Code generation overview

6.1 STM32Cube code generation using only HAL drivers (default mode)

During the C code generation process, STM32CubeMX performs the following actions:

1. If it is missing, it downloads the relevant STM32Cube MCU package from the user repository. STM32CubeMX repository folder is specified in the **Help > Updater settings** menu.
2. It copies from the firmware package, the relevant files in *Drivers/CMSIS* and *Drivers/STM32F4_HAL_Driver* folders and in the *Middleware* folder if a middleware was selected.
3. It generates the initialization C code (.c/.h files) corresponding to the user MCU configuration and stores it in the *Inc* and *Src* folders. By default, the following files are included:
   - **stm32f4xx_hal_conf.h** file: this file defines the enabled HAL modules and sets some parameters (e.g. External High Speed oscillator frequency) to predefined default values or according to user configuration (clock tree).
   - **stm32f4xx_hal_msp.c** (MSP = MCU Support package): this file defines all initialization functions to configure the peripheral instances according to the user configuration (pin allocation, enabling of clock, use of DMA and Interrupts).
   - **main.c** is in charge of:

     Resetting the MCU to a known state by calling the *HAL_init()* function that resets all peripherals, initializes the flash memory interface and the SysTick.

     Configuring and initializing the system clock.

     Configuring and initializing the GPIOs that are not used by peripherals.

     Defining and calling, for each configured peripheral, a peripheral initialization function that defines a handle structure that will be passed to the corresponding peripheral *HAL init* function which in turn will call the peripheral HAL MSP initialization function. Note that when LwIP (respectively USB) middleware is used, the initialization C code for the underlying Ethernet (respectively USB peripheral) is moved from main.c to LwIP (respectively USB) initialization C code itself.
   - **main.h** file:

     This file contains the define statements corresponding to the pin labels set from the **Pinout** tab, as well as the user project constants added from the **Configuration** tab (refer to *Figure 503* and *Figure 504* for examples):

```
#define MyTimeOut        10
#define LD4_Pin          GPIO_PIN_12
#define LD4_GPIO_Port    GPIOD
#define LD3_Pin          GPIO_PIN_13
#define LD3_GPIO_Port    GPIOD
#define LD5_Pin          GPIO_PIN_14
#define LD5_GPIO_Port    GPIOD
#define LD6_Pin          GPIO_PIN_15
#define LD6_GPIO_Port    GPIOD
```

Figure 503. Labels for pins generating define statements

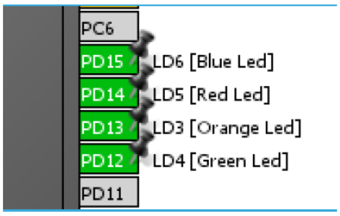
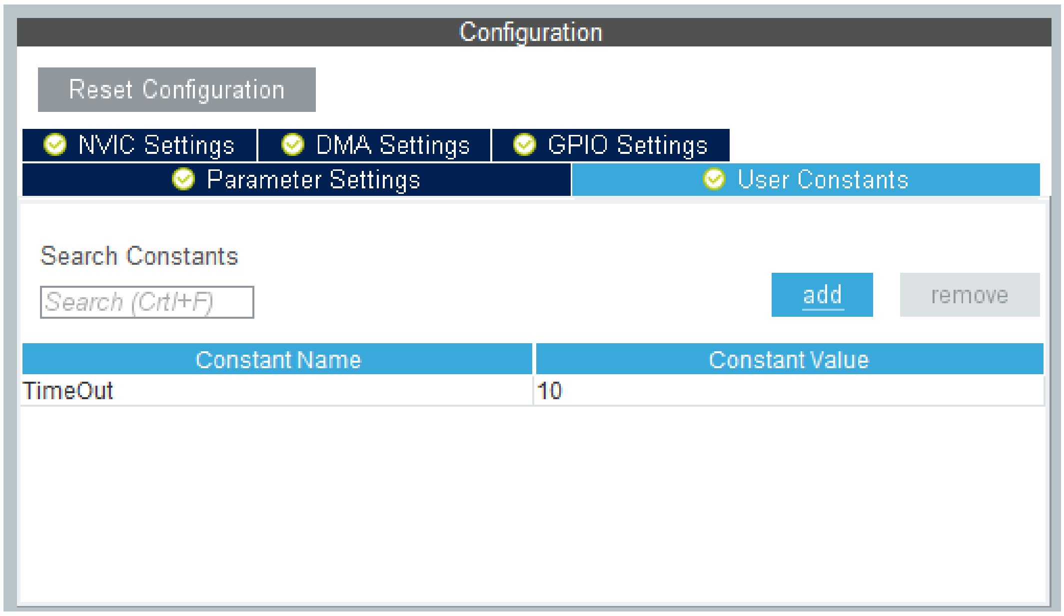


Figure 504. User constant generating define statements

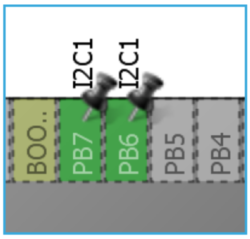


In case of duplicate labels, a unique suffix, consisting of the pin port letter and the pin index number, is added and used for the generation of the associated define statements.

In the example of a duplicate I2C1 labels shown in *Figure 505*, the code generation produces the following code, keeping the I2C1 label on the original port B pin 6 define statements and adding B7 suffix on pin 7 define statements:

```
#define I2C1_Pin           GPIO_PIN_6
#define I2C1_GPIO_Port     GPIOB
#define I2C1B7_Pin         GPIO_PIN_7
#define I2C1B7_GPIO_Port   GPIOB
```

Figure 505. Duplicate labels



In order for the generated project to compile, define statements shall follow strict naming conventions. They shall start with a letter or an underscore as well as the corresponding label. In addition, they shall not include any special character such as minus sign, parenthesis or brackets. Any special character within the label is replaced by an underscore in the define name.

If the label contains character strings between "[]" or "()", only the first string listed is used for the define name. As an example, the label "**LD6** [Blue Led]" corresponds the following define statements:

```
#define LD6_Pin    GPIO_PIN_15
#define LD6_GPIO_Port    GPIOD
```

The define statements are used to configure the GPIOs in the generated initialization code. In the following example, the initialization of the pins labeled *Audio_RST_Pin* and *LD4_Pin* is done using the corresponding define statements:

```
/*Configure GPIO pins : LD4_Pin Audio_RST_Pin */
GPIO_InitStruct.Pin = LD4_Pin | Audio_RST_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_LOW;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
```

4. Finally it generates a *Projects* folder that contains the toolchain specific files that match the user project settings. Double-clicking the IDE specific project file launches the IDE and loads the project ready to be edited, built and debugged.

6.2 STM32Cube code generation using Low Layer drivers

STM32CubeMX allows the user to generate peripheral initialization code based either on the peripheral HAL driver or on the peripheral Low Layer (LL) driver.

The choice is made through the Project Manager view (see *Section 4.11.3: Advanced Settings tab*).

The LL drivers are available only for the peripherals which require an optimized access and do not have a complex software configuration. The LL services allow performing atomic operations by changing the relevant peripheral registers content:

- Examples of supported peripherals: RCC, ADC, GPIO, I2C, SPI, TIM, USART,…
- Examples of peripherals not supported by LL drivers: USB, SDMMC, FSMC.

The LL drivers are available within the STM32CubeL4 package:

- They are located next to the HAL drivers (**stm32l4_hal_<peripheral_name>**) within the *Inc* and *Src* directory of the *STM32Cube_FW_L4_V1.6\Drivers\STM32L4xx_HAL_Driver* folder.
- They can be easily recognizable by their naming convention: **stm32l4_ll_<peripheral_name>**

For more details on HAL and LL drivers refer to the *STM32L4 HAL and Low-layer drivers* user manual (UM1884).

As the decision to use LL or HAL drivers is made on a peripheral basis, the user can mix both HAL and LL drivers within the same project.

The following tables show the main differences between the three possible STM32CubeMX project generation options: HAL-only, LL-only, and mix of HAL and LL code.

Table 24. LL versus HAL code generation: drivers included in STM32CubeMX projects

Project configuration and drivers to be included	HAL only	LL only	Mix of HAL and LL	Comments
CMSIS	Yes	Yes	Yes	-
STM32xxx_HAL_Driver	Only HAL driver files	Only LL driver files	Mix of HAL and LL driver files	Only the driver files required for a given configuration (selection of peripherals) are copied when the project settings option is set to “Copy only the necessary files”. Otherwise (“all used libraries” option) the complete set of driver files is copied.

Table 25. LL versus HAL code generation: STM32CubeMX generated header files

Generated header files	HAL only	LL only	Mix of HAL and LL	Comments
main.h	Yes	Yes	Yes	This file contains the include statements and the generated define statements for user constants (GPIO labels and user constants).
stm32xxx_hal_conf.h	Yes	No	Yes	This file enables the HAL modules necessary to the project.
stm32xxx_it.h	Yes	Yes	Yes	Header file for interrupt handlers
stm32xx_assert.h	No	Yes	Yes	This file contains the assert macros and the functions used for checking function parameters.

Table 26. LL versus HAL: STM32CubeMX generated source files

Generated source files	HAL only	LL only	Mix of HAL and LL	Comments
main.c	Yes	Yes	Yes	Contains the main functions and, optionally, STM32CubeMX generated functions.
stm32xxx_hal_msp.c	Yes	No	Yes	Contains the following functions: – HAL_MspInit – for peripherals using HAL drivers: HAL_<Peripheral>_MspInit, HAL_<Peripheral>_MspDeInit, These functions are available only for the peripherals that use HAL drivers.
stm32xxx_it.c	Yes	Yes	Yes	Source file for interrupt handlers

Table 27. LL versus HAL: STM32CubeMX generated functions and function calls

Generated source files	HAL only	LL only	Mix of HAL and LL	Comments
Hal_init()	Called in main.c	Not used	Called in main.c	This file performs the following functions: – Configuration of flash memory prefetch and instruction and data caches – Selection of the SysTick timer as timebase source – Setting of NVIC group priority – MCU low-level initialization.
Hal_msp_init()	Generated in stm32xxx_hal_msp.c and called by HAL_init()	Not used	Generated in stm32xxx_hal_msp.c And called by HAL_init()	This function performs the peripheral resources configuration(1).
MX_<Peripheral>_Init()	[1]: Peripheral configuration and call to HAL_<Peripheral>_Init()	[2]: Peripheral and peripheral resource configuration(1) using LL functions Call to LL_Peripheral_Init()	– When HAL driver is selected for the <Peripheral>, function generation and calls are done following *[1]: Peripheral configuration and call to HAL_<Peripheral>_Init()* – When LL driver selected for the <Peripheral>, function generation and calls are done following *[2]: Peripheral and peripheral resource configuration using LL functions*	This file takes care of the peripherals configuration. When the LL driver is selected for the <Peripheral>, it also performs the peripheral resources configuration(1).

Table 27. LL versus HAL: STM32CubeMX generated functions and function calls (continued)

Generated source files	HAL only	LL only	Mix of HAL and LL	Comments
HAL_<Peripheral>_MspInit()	[3]: Generated in stm32xxx_hal_msp.c when HAL driver selected for the <Peripheral>	Not used	Only HAL driver can be selected for the <Peripheral>: function generation and calls are done following *[3]: Generated in stm32xxx_hal_msp.c when HAL driver selected for the <Peripheral>*	Peripheral resources configuration(1)
HAL_<Peripheral>_MspDeInit()	[4]: Generated in stm32xxx_hal_msp.c when HAL driver selected for the <Peripheral>	Not used	Only HAL driver can be selected for the <Peripheral>: function generation and calls are done following *[4]: Generated in stm32xxx_hal_msp.c when HAL driver selected for the <Peripheral>*	This function can be used to free peripheral resources.

1. Peripheral resources include:
   - peripheral clock
   - pinout configuration (GPIOs)
   - peripheral DMA requests
   - peripheral Interrupt requests and priorities.

Figure 506. HAL-based peripheral initialization: usart.c code snippet

USART Peripheral initialization - HAL-based

Peripheral Configuration

```
void MX_USART1_UART_Init(void)
{
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_7B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  ...
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
}
```

Peripheral Resources Configuration

```
void HAL_UART_MspInit(UART_HandleTypeDef* uartHandle)
{
  GPIO_InitTypeDef GPIO_InitStruct;
  if(uartHandle->Instance==USART1)
  {
    /* Peripheral clock enable */
    __HAL_RCC_USART1_CLK_ENABLE();
    /* USART1 GPIO Configuration */
    GPIO_InitStruct.Pin = GPIO_PIN_10;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    ...
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  }
}
```

Peripheral Resources Release

```
void HAL_UART_MspDeInit(UART_HandleTypeDef* uartHandle)
{
  if(uartHandle->Instance==USART1)
  {
      /* Peripheral clock disable */
    __HAL_RCC_USART1_CLK_DISABLE();
    /* USART1 GPIO Configuration */
    HAL_GPIO_DeInit(GPIOA, GPIO_PIN_10);
    HAL_GPIO_DeInit(GPIOB, GPIO_PIN_6);
  }
}
```

Figure 507. LL-based peripheral initialization: usart.c code snippet

USART Peripheral Initialization using LL drivers

```
void MX_USART1_UART_Init(void)
{
  LL_USART_InitTypeDef USART_InitStruct;
  LL_GPIO_InitTypeDef GPIO_InitStruct;
  /* Peripheral clock enable */
  LL_APB2_GRP1_EnableClock(LL_APB2_GRP1_PERIPH_USART1);

    /**USART1 GPIO Configuration
    PA10     ------> USART1_RX
    PB6     ------> USART1_TX
    */
  GPIO_InitStruct.Pin = LL_GPIO_PIN_10;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
  GPIO_InitStruct.Alternate = LL_GPIO_AF_7;
  LL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  GPIO_InitStruct.Pin = LL_GPIO_PIN_6;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
  GPIO_InitStruct.Alternate = LL_GPIO_AF_7;
  LL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  USART_InitStruct.BaudRate = 115200;
  USART_InitStruct.DataWidth = LL_USART_DATAWIDTH_7B;
  USART_InitStruct.StopBits = LL_USART_STOPBITS_1;
  USART_InitStruct.Parity = LL_USART_PARITY_NONE;
  USART_InitStruct.TransferDirection = LL_USART_DIRECTION_TX_RX;
  USART_InitStruct.HardwareFlowControl = LL_USART_HWCONTROL_NONE;
  USART_InitStruct.OverSampling = LL_USART_OVERSAMPLING_16;

  LL_USART_Init(USART1, &USART_InitStruct);
  LL_USART_ConfigAsyncMode(USART1);
}
```

Peripheral Resources Configuration

Peripheral Configuration

Figure 508. HAL versus LL: main.c code snippet

main.c HAL-based

```
  */
/* Includes ------------------------------------------
#include "main.h"
#include "stm32l4xx_hal.h"
#include "usart.h"
#include "gpio.h"

void SystemClock_Config(void);
void Error_Handler(void);

int main(void)
{
  /* Reset of all peripherals,
  Initializes the Flash interface and the Systick.
  **/
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART1_UART_Init();
```

main.c LL-based

```
  */
/* Includes ------------------------------------------
#include "main.h"

#include "usart.h"
#include "gpio.h"

void SystemClock_Config(void);
void Error_Handler(void);

int main(void)
{
  /* Reset of all peripherals,
  Initializes the Flash interface and the Systick.
  **/
  LL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART1_UART_Init();
```

6.3 Custom code generation

STM32CubeMX supports custom code generation by means of a FreeMarker template engine (see http://www.freemarker.org).

6.3.1 STM32CubeMX data model for FreeMarker user templates

STM32CubeMX can generate a custom code based on a FreeMarker template file (.ftl extension) for any of the following MCU configuration information:

- List of MCU peripherals used by the user configuration
- List of parameters values for those peripherals
- List of resources used by these peripherals: GPIO, DMA requests and interrupts.

The user template file must be compatible with STM32CubeMX data model. This means that the template must start with the following lines:

```
[#ftl]
[#list configs as dt]
[#assign data = dt]
[#assign peripheralParams =dt.peripheralParams]
[#assign peripheralGPIOParams =dt.peripheralGPIOParams]
[#assign usedIPs =dt.usedIPs]
```

and end with

```
[/#list]
```

A sample template file is provided for guidance (see *Figure 509*).

STM32CubeMX will also generate user-specific code if any is available within the template.

As shown in the example below, when the sample template is used, the ftl commands are provided as comments next to the data they have generated:

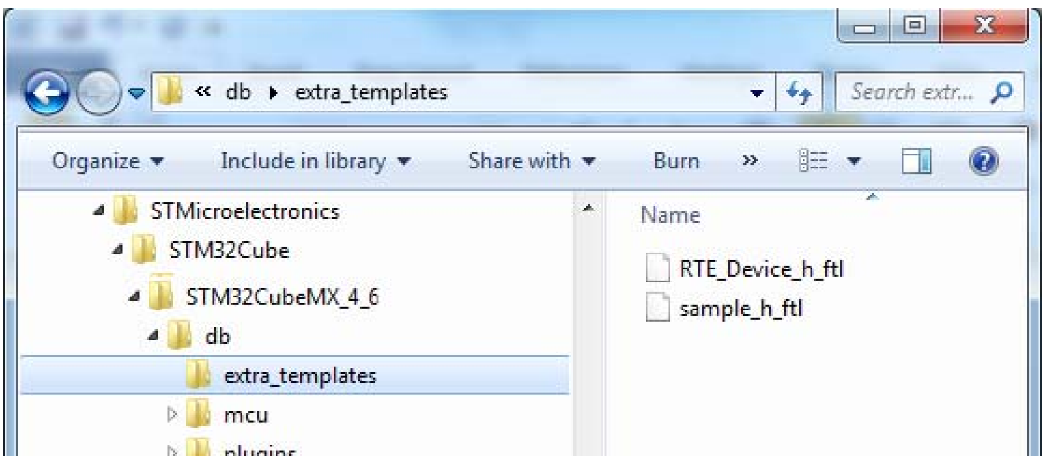
FreeMarker command in template:

```
${peripheralParams.get("RCC").get("LSI_VALUE")}
```

Resulting generated code:

```
LSI_VALUE :  32000       [peripheralParams.get("RCC").get("LSI_VALUE")]
```

Figure 509. Default content of the extra_templates folder



6.3.2 Saving and selecting user templates

The user can either place the FreeMarker template files under STM32CubeMX installation path within the db/extra_templates folder or in any other folder.

Then for a given project, the user will select the template files relevant for its project via the **Template Settings** window accessible from the Code Generator Tab in the **Project Manager** view menu (see *Section 4.11*)

6.3.3 Custom code generation

To generate custom code, the user must place the FreeMarker template file under STM32CubeMX installation path within the db/extra_templates folder (see *Figure 510*).

The template filename must follow the naming convention <user filename>_<file extension>.ftl in order to generate the corresponding custom file as <user filename>.<file extension>.

By default, the custom file is generated in the user project root folder, next to the .ioc file (see *Figure 511*).

To generate the custom code in a different folder, the user shall match the destination folder tree structure in the extra_template folder (see *Figure 512*).

Figure 510. extra_templates folder with user templates

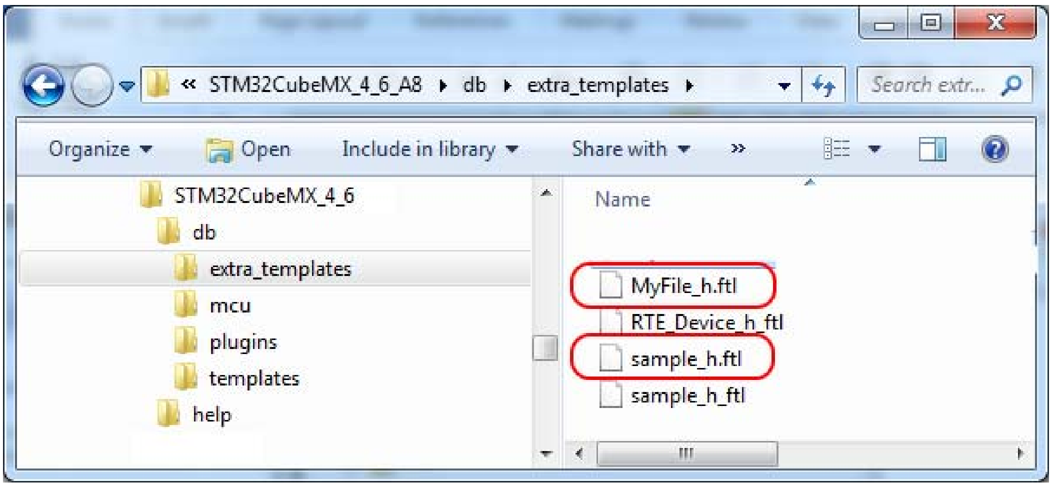


Figure 511. Project root folder with corresponding custom generated files

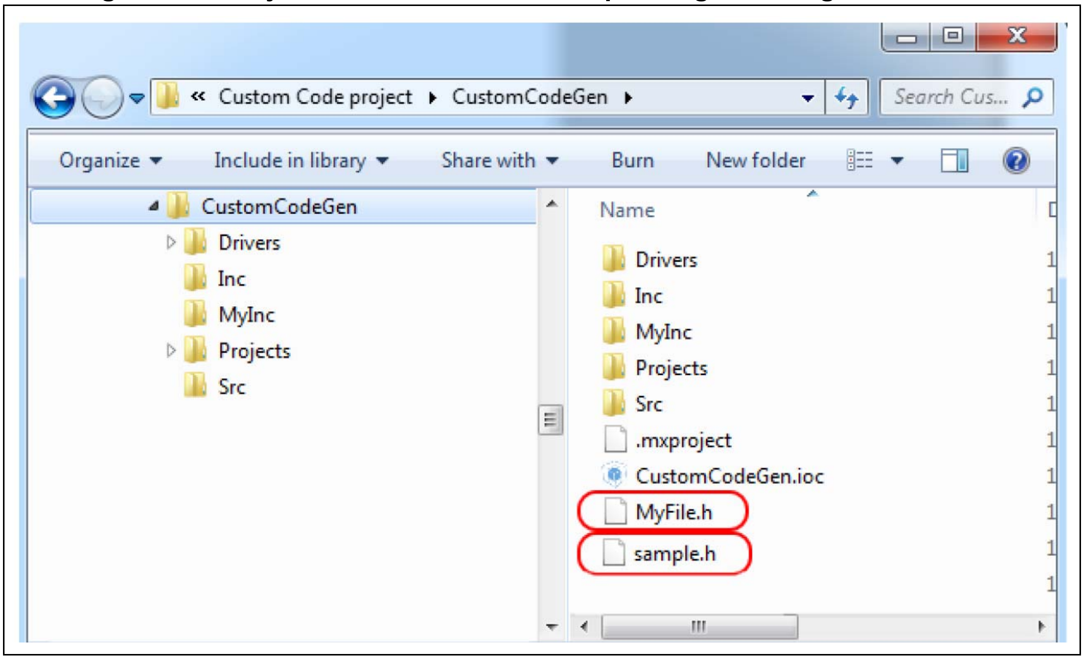


Figure 512. User custom folder for templates

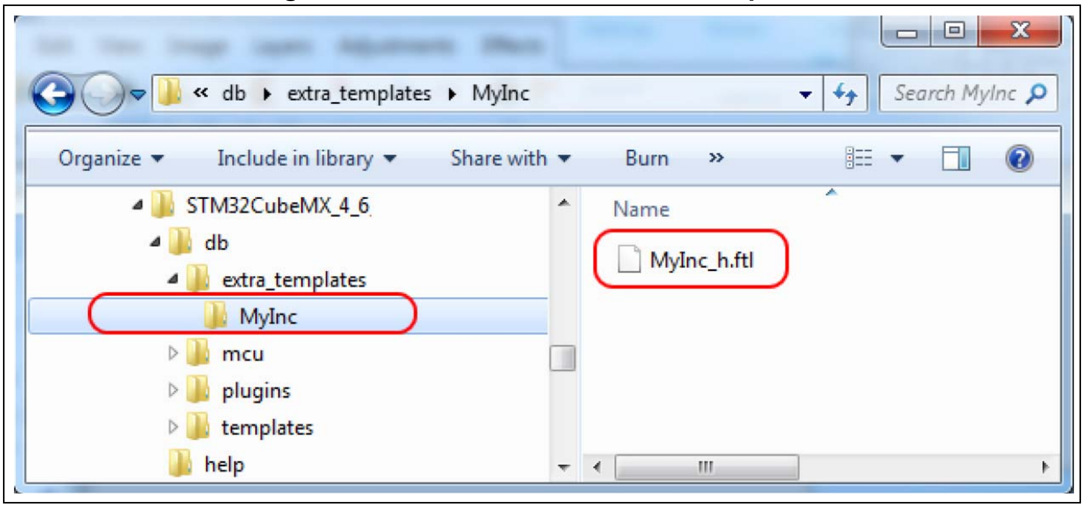
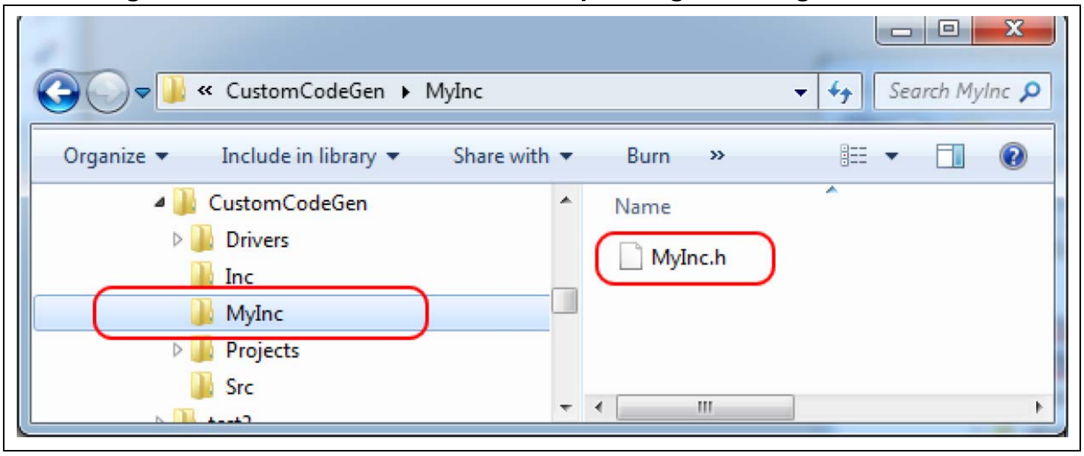


Figure 513. Custom folder with corresponding custom generated files



6.4 Additional settings for C project generation

STM32CubeMX allows specifying additional project settings through the .extSettings file. This file must be placed in the same project folder and at the same level as the .ioc file.

As an example, additional settings can be used when external tools call STM32CubeMX to generate the project and require specific project settings.

Possible entries and syntax

All entries are optional. They are organized under the followings three categories: ProjectFiles, Groups or Others.

- **[ProjectFiles]:** section where to specify additional include directories

  Syntax

  ```
  HeaderPath = <include directory 1 path>;< include directory 2 path >
  ```

  Example

  ```
  HeaderPath=../../IIR_Filter_int32/Inc ;
  ```

- **[Groups]**: section where to create new groups of files and/or add files to a group

  Syntax

  ```
  <Group name> = <file pathname1>;< file pathname2>
  ```

  Example

  ```
  Doc=$ PROJ_DIR$\..\readme.txt
  Lib=C:\libraries\mylib1.lib; C:\libraries\mylib2.lib;
  Drivers/BSP/MyRefBoard = C:\MyRefBoard\BSP\board_init.c;
  C:\MyRefBoard\BSP\board_init.h;
  ```

- **[Others]**: section where to enable HAL modules and/or specify preprocessor define statements

  – Enabling preprocessor define statements

    Preprocessor define statements can be specified using the following syntax after the [Others] line:

    Syntax

    ```
    Define = <define1_name>;<define2_name>
    ```

Example

```
Define= USE_STM32F429I_DISCO
```

– Enabling HAL modules in generated stm32f4xx_hal_conf.h

HAL modules can be enabled using the following syntax after the [Others] line:

Syntax

```
HALModule = <ModuleName1>; <ModuleName1>;
```

Example

```
HALModule=I2S;I2C
```

.extSettings file example and generated outcomes

For the purpose of the example, a new project is created by selecting the STM32F429I-DISCO board from STM32CubeMX board selector. The EWARM toolchain is selected in the Project tab of the **Project Manager** view. The project is saved as *MyF429IDiscoProject*. In the project folder, next to the generated .ioc file, a .extSettings text file is placed with the following contents:

[Groups]

```
Drivers/BSP/STM32F429IDISCO=C:\Users\frq09031\STM32Cube\Repository\STM32Cube_FW_F4_V1.14.0\Drivers\BSP\STM32F429I-Discovery\stm32f429i_discovery.c;
C:\Users\frq09031\STM32Cube\Repository\STM32Cube_FW_F4_V1.14.0\Drivers\BSP\STM32F429I-Discovery\stm32f429i_discovery.h
Lib=C:\Users\frq09031\STM32Cube\Repository\STM32Cube_FW_F4_V1.14.0\Middlewares\Third_Party\FreeRTOS\Source\portable\IAR\ARM_CM4F\portasm.s
Doc=$PROJ_DIR$\..\readme.txt
```

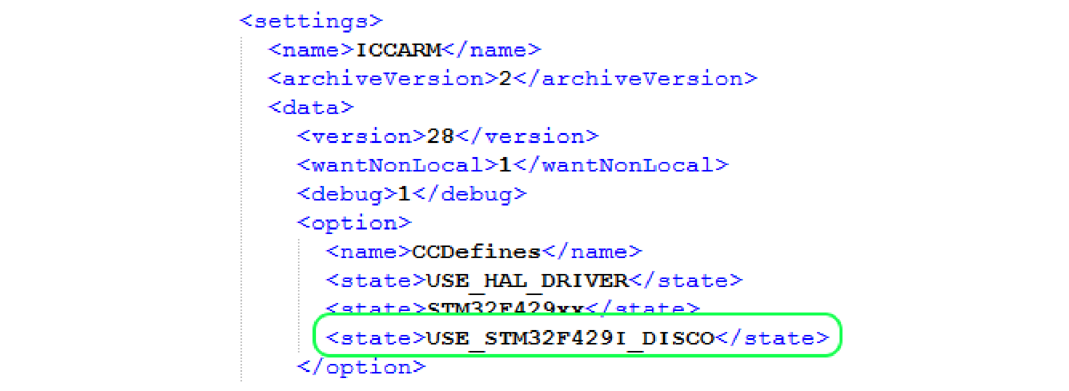
[Others]

```
Define = USE_ STM32F429I_DISCO
HALModule = UART;SPI
```

Upon project generation, the presence of this .extSettings file triggers the update of:

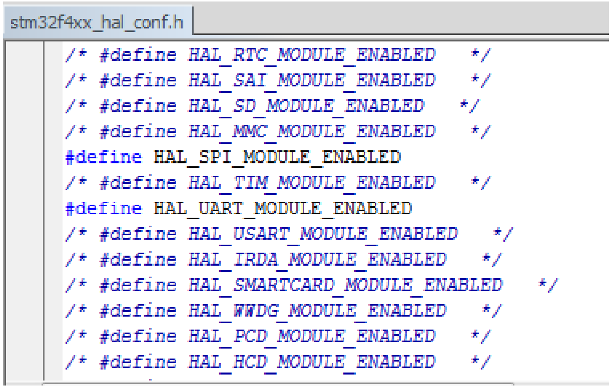
- the project MyF429IDiscoProject.ewp file in EWARM folder (see *Figure 514*)
- the stm32f4xx_hal_conf.h file in the project Inc folder (see *Figure 515*)
- the project view within EWARM user interface as shown in *Figure 516* and *Figure 517*.

Figure 514. Update of the project .ewp file (EWARM IDE) for preprocessor define statements



```
<settings>
  <name>ICCARM</name>
  <archiveVersion>2</archiveVersion>
  <data>
    <version>28</version>
    <wantNonLocal>1</wantNonLocal>
    <debug>1</debug>
    <option>
      <name>CCDefines</name>
      <state>USE_HAL_DRIVER</state>
      <state>STM32F429xx</state>
      <state>USE_STM32F429I_DISCO</state>
    </option>
```

Figure 515. Update of stm32f4xx_hal_conf.h file to enable selected modules



stm32f4xx_hal_conf.h

```
/* #define HAL_RTC_MODULE_ENABLED   */
/* #define HAL_SAI_MODULE_ENABLED   */
/* #define HAL_SD_MODULE_ENABLED   */
/* #define HAL_MMC_MODULE_ENABLED   */
#define HAL_SPI_MODULE_ENABLED
/* #define HAL_TIM_MODULE_ENABLED   */
#define HAL_UART_MODULE_ENABLED
/* #define HAL_USART_MODULE_ENABLED   */
/* #define HAL_IRDA_MODULE_ENABLED   */
/* #define HAL_SMARTCARD_MODULE_ENABLED   */
/* #define HAL_WWDG_MODULE_ENABLED   */
/* #define HAL_PCD_MODULE_ENABLED   */
/* #define HAL_HCD_MODULE_ENABLED   */
```

Figure 516. New groups and new files added to groups in EWARM IDE

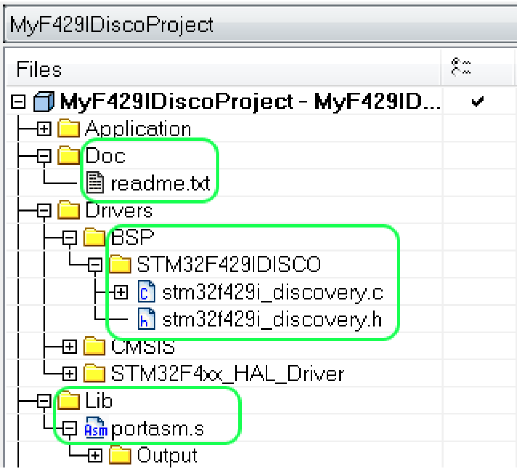
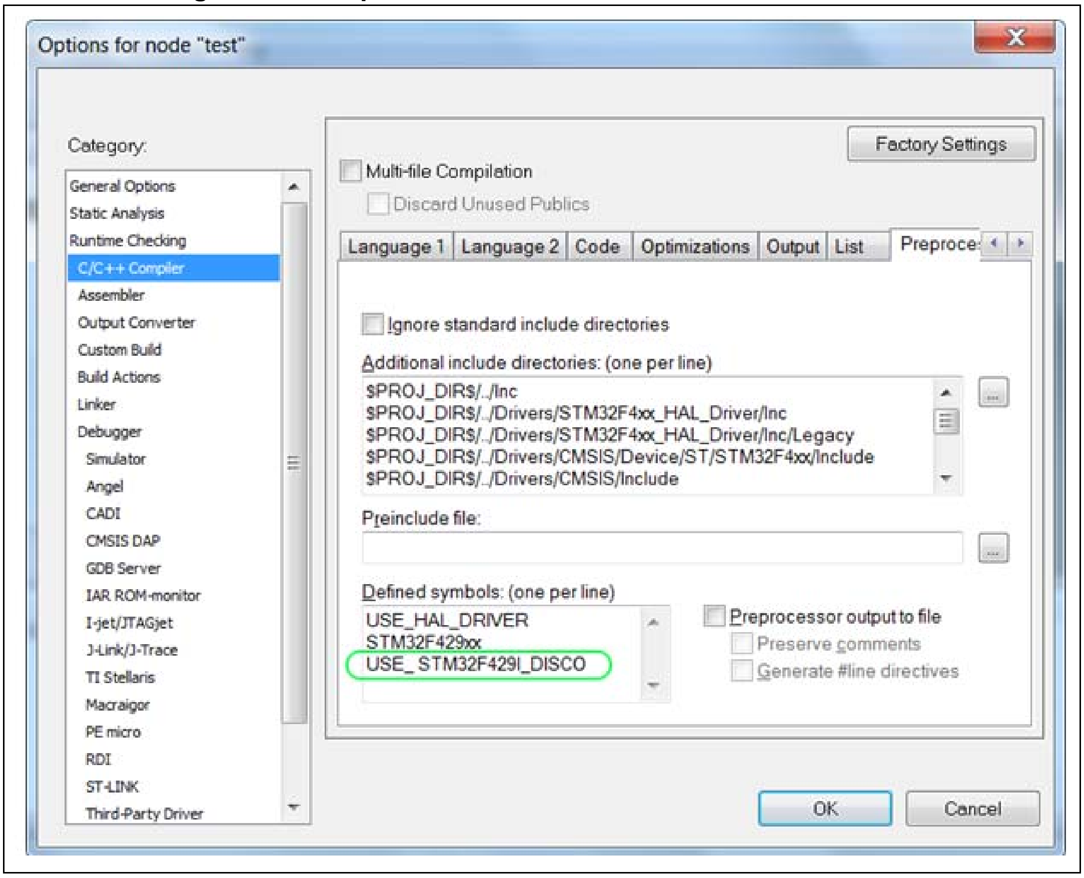


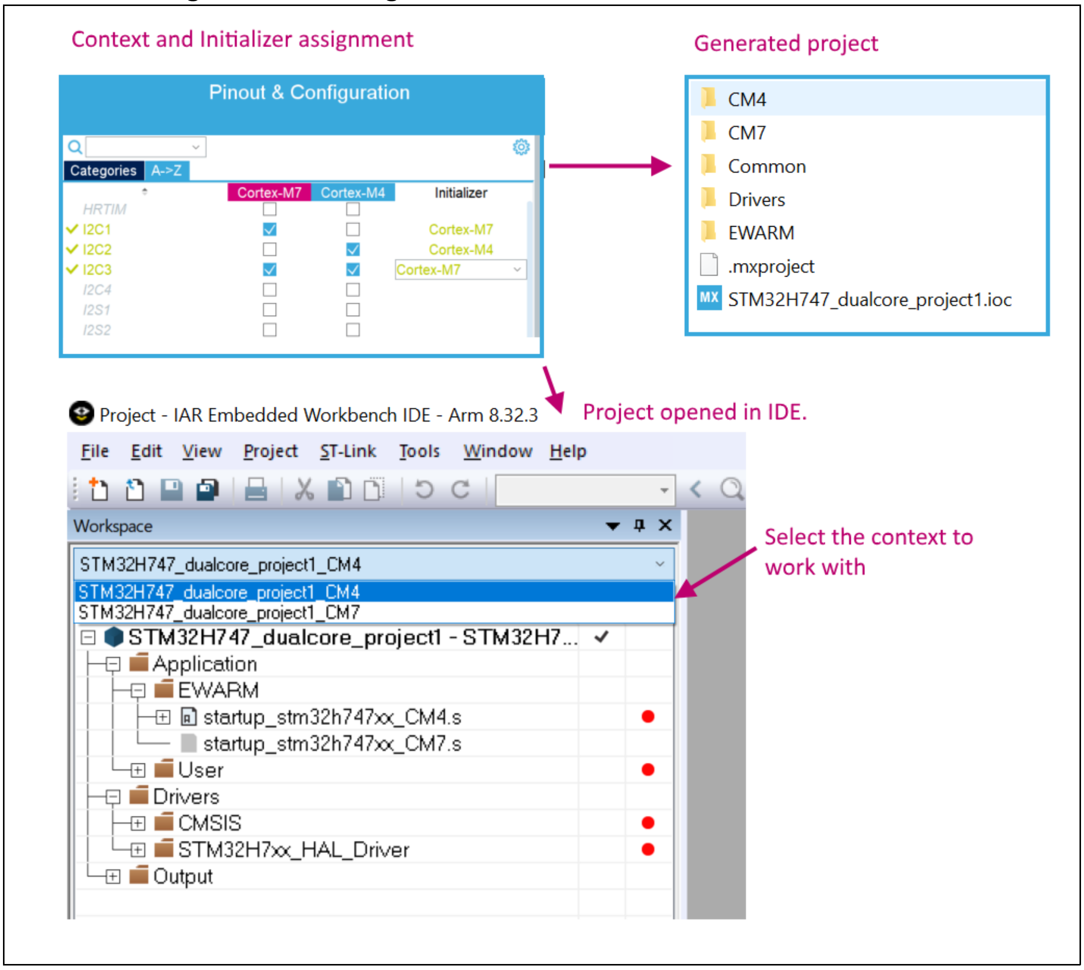
Figure 517. Preprocessor define statements in EWARM IDE



7 Code generation for dual-core MCUs (STM32H7 dual-core product lines only)

For working with Arm Cortex-M dual-core products, STM32CubeMX generates code for both cores automatically according to the context assignment and initializer choices made in the user interface (see *Section 4.8: Pinout & Configuration view for STM32H7 dual-core products* for details).

Figure 518. Code generation for STM32H7 dual-core devices



Generated initialization code

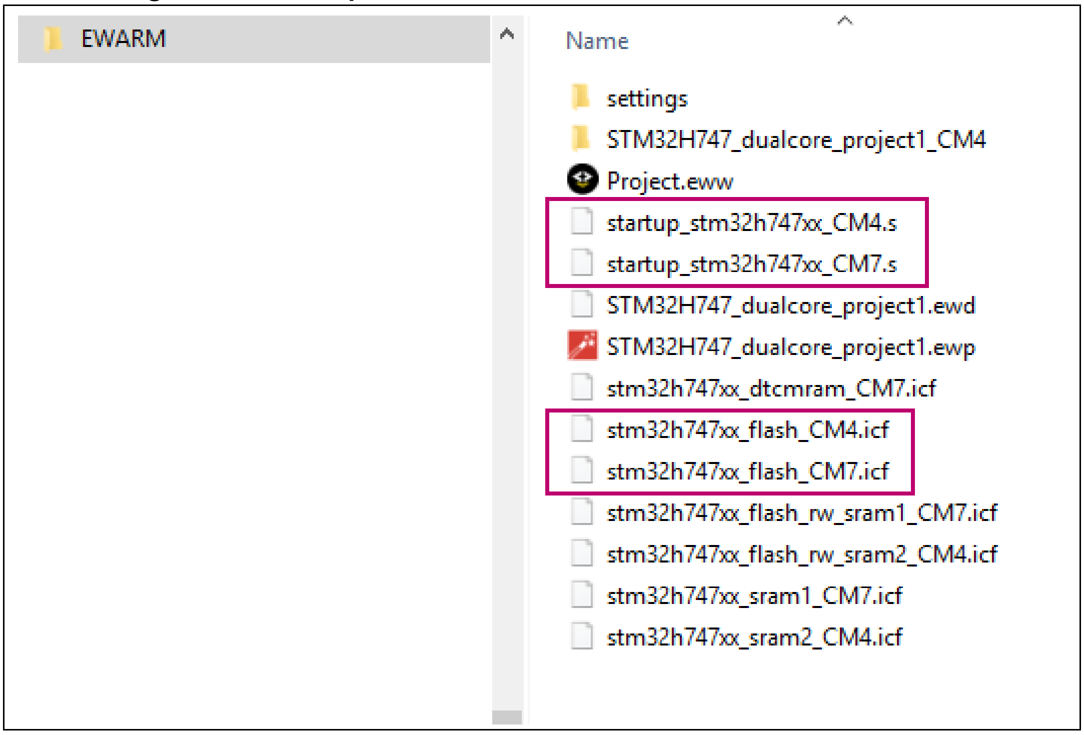
The code is generated in CM4, CM7 and Common folders. The Common folder holds the system_stm32h7xx.c, that contains the clock tree settings.

When a peripheral or middleware is assigned to both contexts, the function MX_<name>_init will be generated for both contexts but will be called only from the initializer side.

Generated startup and linker files

Each configuration (_M4 or _M7) of the project shall come with a startup file and a linker file, each suffixed with _M4 or _M7 respectively.

Figure 519. Startup and linker files for STM32H7 dual-core devices



Generated boot mode code

STM32CubeMX supports only one mode of boot for now, where both ARM Cortex-M cores boot at once.

The other boot modes will be introduced later as a project option in the project manager view:

- Arm Cortex-M7 core booting, Arm Cortex-M4 gated
- Arm Cortex-M4 core booting, Arm Cortex-M7 gated
- A first core booting executing from flash, loads the second core code to the SRAM then enables the second core to boot.

STM32CubeMX uses template files delivered with STM32CubeH7 MCU packages as reference.

8 Code generation with TrustZone enabled (STM32L5 series only)

In STM32CubeMX project manager view, all project generation options remain available.

However, the choice of toolchains is limited to the IDEs/compilers supporting the Cortex®-M33 core:

- EWARM v8.32 or higher
- MDK-ARM v5.27 or higher (ARM compiler 6)
- STM32CubeIDE (GCC v4.2 or higher)
- Makefile (GCC v4.2 or higher)

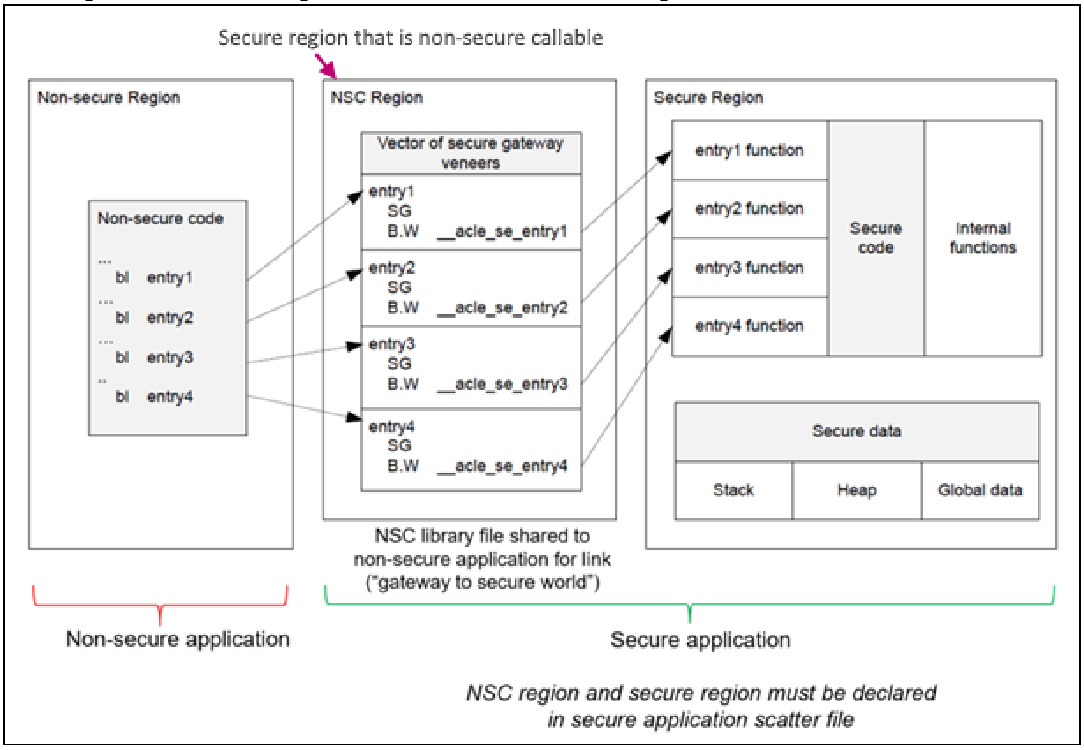
Upon product selection, STM32CubeMX requires to choose between enabling TrustZone or not.

- When TrustZone is enabled, STM32CubeMX generates two C projects: one secure and one nonsecure. After compilation, two images are available for download, one for each context.
- When TrustZone is disabled, STM32CubeMX generates a nonsecure C project, as for other products not supporting it.

Specificities

When TrustZone is enabled, the project generation must be adjusted to ensure that secure and nonsecure images can be built.

Figure 520. Building secure and nonsecure images with ARMv8-M TrustZone



When TrustZone is enabled for the project, STM32CubeMX generates three folders:

- NonSecure for nonsecure code
- Secure for secure code
- Secure_nsclib for nonsecure callable region

See *Figure 521* (use TZ_BasicStructure_project_inCubeIDE.png) and *Figure 522* (use STM32L5_STM32CubeMX_Project_settings_inCubeIDE.png).

Figure 521. Project explorer view for STM32L5 TrustZone enabled projects

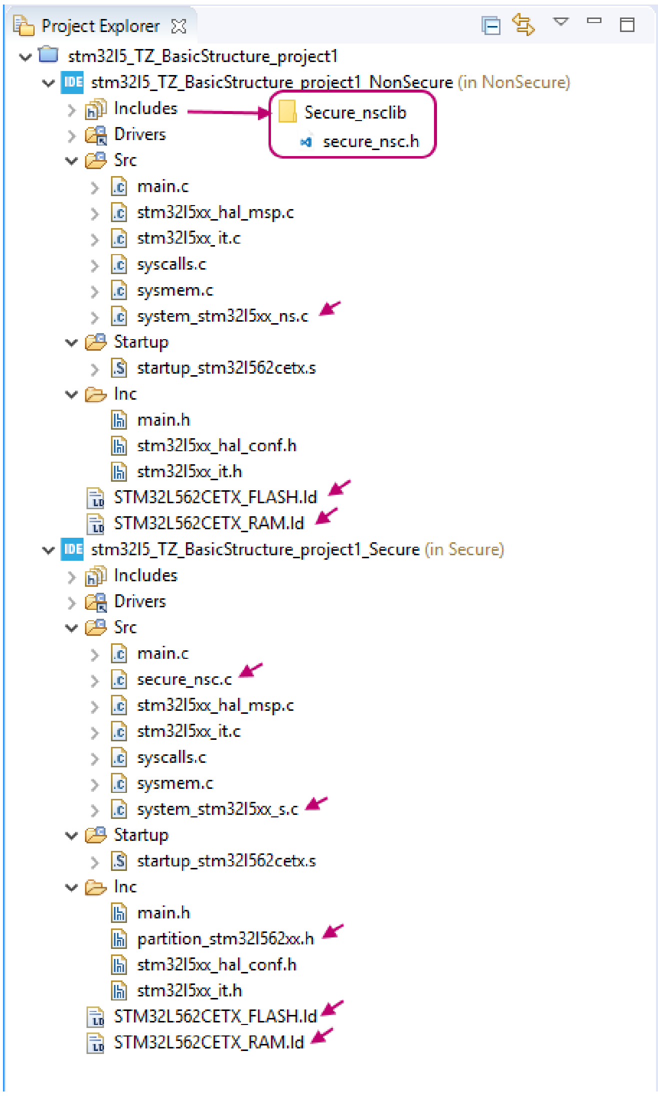
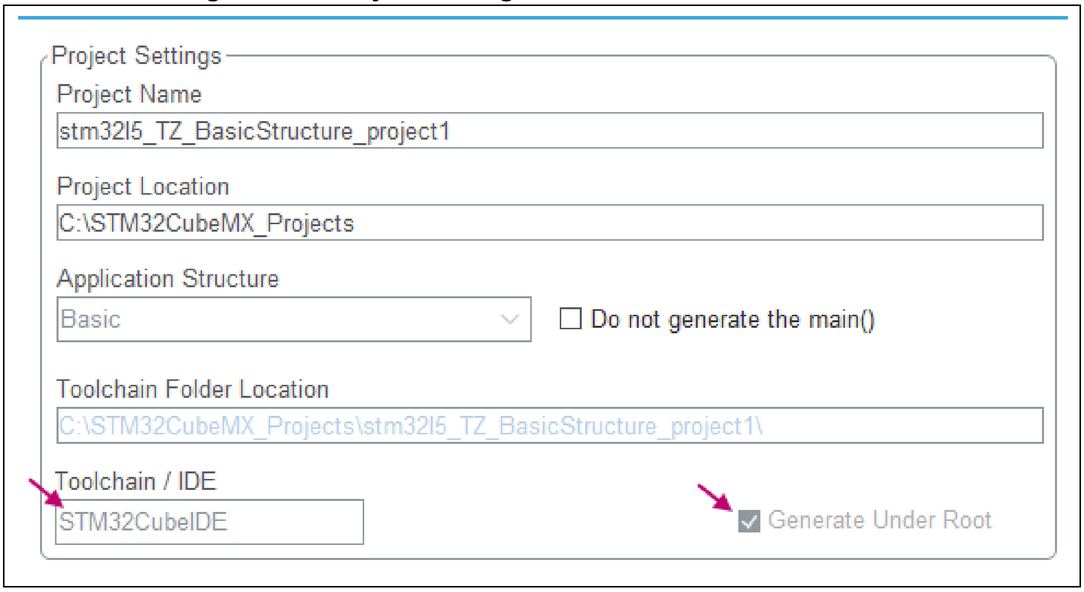


Figure 522. Project settings for STM32CubeIDE toolchain



STM32CubeMX also generates specific files, detailed in *Table 28*.

Table 28. Files generated when TrustZone is enabled

File	Folder	Details
The product core secure/nonsecure partitioning .h “template” file Example: partition_stm32l552xx.h	Secure	Initial setup for secure/nonsecure zones for ARMCM33 based on CMSIS CORE V5.3.1 partition_ARMCM33.h Template. It initializes Security attribution unit (SAU) CTRL register, setup behavior of Sleep and Exception Handling, Floating Point Unit and Interrupt Target.
secure_nsc.h file	Secure_nsclib	Must be filled by the user with the list of nonsecure callable APIs. Templates are available as reference in STM32L5Cube embedded software package in Templates\TrustZone®\Secure_nsclib folders.
System_stm32l5xx_s.c	Secure	CMSIS Cortex-M33 device peripheral access layer system source file to be used in secure application when the system implements security.

Table 28. Files generated when TrustZone is enabled (continued)

File	Folder	Details
System_stm32l5xx_ns.c	NonSecure	CMSIS Cortex-M33 device peripheral access layer system source file to be used in nonsecure application when the system implements security.
STM32L562CETX_FLASH STM32L562CETX_RAM or STM32L552CETX_FLASH STM32L552CETX_RAM	Secure, NonSecure	Linker files for the secure and nonsecure memory layouts. File extensions and naming conventions: – .icf (EWARM) – .sct (MDK-ARM), or – .ld (GCC compiler toolchains)

9 Device tree generation (STM32MPUs only)

The Device tree in Linux is used to provide a way to describe non-discoverable hardware. STMicroelectronics is widely using the device tree for all the platform configuration data, including DDR configuration.

Linux developers can manually edit device tree source files (dts), but as an alternative STM32CubeMX offers a partial device-tree generation service to reduce effort and to ease new comers. STM32CubeMX intends to generate partially device trees corresponding to board level configuration. Partial means that the entire (board level) device-trees are not generated, but only main sections that usually imply huge efforts and can cause compilation errors and dysfunction:

- folders structure and files to folders distribution
- dtsi and headers inclusions
- pinCtrl and clocks generation
- System-On-Chip device nodes positioning
- multi-core related configurations (Etzpc binding, resources manager binding, peripherals assignment)

9.1 Device tree overview

To run properly, any piece of software needs to get the hardware description of the platform on which it is executed, including the kind of CPU, the memory size and the pin configuration. OpenSTLinux firmware has put such non-discoverable hardware description in a separate binary, the device tree blob (dtb). The device tree blob is compiled from the device tree source files (dts) using the dtc compiler provided with the OpenSTLinux distribution.

The device tree structure consist of a board level file (.dts) that includes two device tree source include files (.dtsi): a soc level file and a –pinctrl file, that lists the pin muxing configurations.

The device tree structure is very close to C language multiple level structures with the “root” (/) being the highest level then “peripherals” being sub-nodes described further in the hierarchy (see figures *523*, *524* and *525*).

STM32CubeMX generation uses widely overloading mechanisms to complete or change some SOC devices definitions when user configurations require it.

Figure 523. STM32CubeMX generated DTS – Extract 1

```
System and Board information
    model = "STMicroelectronics custom STM32CubeMX board";
    compatible = "st,stm32mp157c-project2-mx", "st,stm32mp157";

      memory@c0000000 {
        ...
    };

    /* USER CODE BEGIN root */
    /* USER CODE END root */        User customization

    clocks {                        Full clock configuration
        clk_lsi: clk-lsi {
            #clock-cells = <0>;
            compatible = "fixed-clock";
            clock-frequency = <32000>;
            u-boot,dm-pre-reloc;
        };
        ...
    };

}; /*root*/

&pinctrl {      Pin control configuration, including GPIO configuration
    u-boot,dm-pre-reloc;
    tim1_pins_mx: tim1_mx-0 {
        pins {
            pinmux = <STM32_PINMUX('A', 8, AF1)>, /* TIM1_CH1 */
                     <STM32_PINMUX('A', 9, AF1)>; /* TIM1_CH2 */
            bias-disable;
            drive-push-pull;
            slew-rate = <0>;
        };
    };
};
```

Figure 524. STM32CubeMX generated DTS – Extract 2

```
&m4_rproc{              Multi-core management
    recovery;

    m4_system_resources{
        status = "okay";

        /* USER CODE BEGIN m4_system_resources */
        /* USER CODE END m4_system_resources */
    };

    status = "okay";

    /* USER CODE BEGIN m4_rproc */
    /* USER CODE END m4_rproc */
};

&m4_timers1{            Peripheral assignment to Cortex-M4 run time context
    pinctrl-names = "rproc_default", "rproc_sleep";
    pinctrl-0 = <&tim1_pins_mx>;
    pinctrl-1 = <&tim1_sleep_pins_mx>;
    status = "okay";

    /* USER CODE BEGIN m4_timers1 */
    /* USER CODE END m4_timers1 */
};
```

Figure 525. STM32CubeMX generated DTS – Extract 3

```
&timers2{
    status = "okay";

    /* USER CODE BEGIN timers2 */
    /* USER CODE END timers2 */

    pwm{
        pinctrl-names = "default", "sleep";
        pinctrl-0 = <&tim2_pwm_pins_mx>;
        pinctrl-1 = <&tim2_pwm_sleep_pins_mx>;
        status = "okay";

        /* USER CODE BEGIN timers2_pwm */
        /* USER CODE END timers2_pwm */
    };
};

/* USER CODE BEGIN dts_addons */
/* USER CODE END dts_addons */
```

Peripheral node structure with

PinCtrl configuration
Status configuration
User customization

For more details refer to "Device Tree for Dummies" from Thomas Petazzoni, available on https://elinux.org.

For more information about STM32MPUs device tree specificities, refer to ST Wiki https://wiki.st.com/stm32mpu.

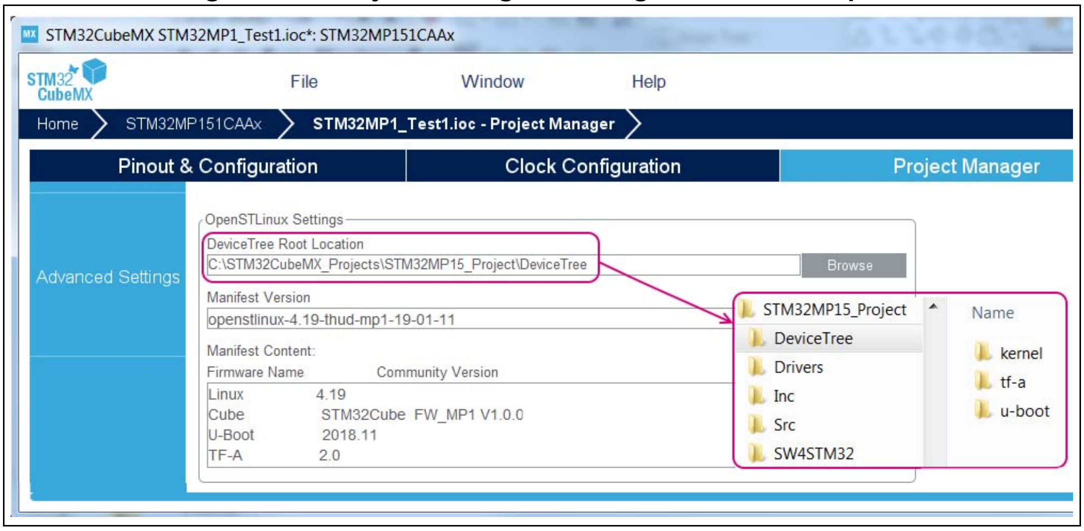
9.2 STM32CubeMX Device tree generation

For STM32MPUs, STM32CubeMX code generation feature has been extended to generate Device trees (DT) configuring the firmware.

DTS generation is accessible through the same **GENERATE CODE** button.

The DT generation path can be configured from the Project Manager view, in the Advanced Settings tab, under OpenSTLinux Settings (see *Figure 526*). For each Device tree STM32CubeMX generates Device tree source (DTS) files.

Figure 526. Project settings to configure Device tree path



The Device tree structure consists of:

- a complete clock-tree
- a complete pin control
- a complete multi-cores references definition
- a set of device nodes and sub-nodes
- user sections that can be filled to have complete and bootable Device trees (contents are not lost at next generation).

The generated DTS files reflect the user configuration, such as the assignment of peripherals to runtime contexts and boot loaders, or clock tree settings.

STM32CubeMX DT generation ensures the coherency between the different DTs. Additionally, it generates the DDR configuration file as part of the boot loader Device trees.

These files, along with the files they include, are compiled to create the device tree blob for the targeted firmware.

The STM32CubeMX Device tree structure depends upon the targeted firmware and, in a few cases, upon the OpenSTLinux manifest version and/or the MPU family. The structures are detailed in https://wiki.st.com/stm32mpu/wiki/Category:Platform_configuration.

The device tree nodes generated by STM32CubeMX can be completed by filling the user sections following the device tree bindings of the different firmware.

Note: To continue the process and learn how to use the generated files, see the dedicated Wiki pages for MPUs.

10 Support of additional software components using CMSIS-Pack standard

The CMSIS-Pack standard describes a delivery mechanism for software components, device parameters, and evaluation board support.

The XML-based package description (pdsc) file describes the content of a software pack (file collection). It includes source code, header files, software libraries, documentation and source code templates. A software pack consists of the complete file collection along with the pdsc file, shipped in ZIP-format. After installing a software pack, all the included software components are available to the development tools.

A software component is a collection of source modules, header and configuration files as well as libraries. Packs containing software components can also include example projects and user code templates.

Refer to http://www.keil.com website for more details.

STM32CubeMX supports third-party and other STMicroelectronics embedded software solutions, delivered as software packs. STM32CubeMX enables to:

1. Install software packs and check for updates (see *Section 3.4.5*).
2. Select software components for the current project (see *Section 4.15*). Once this is done, the selected components appear in the tree view (see *Figure 527*).
3. Enable the software component from the tree view (see *Figure 528*). Use contextual help to get more details on the selection.
4. Configure software components (see *Figure 528*). This function is possible only for components coming with files in STM32CubeMX proprietary format.
5. Generate the C project for selected toolchains (see *Figure 529*).
   a) Software components files are automatically copied to the project.
   b) Software component configuration and initialization code are automatically generated. This function is possible only for components coming with files in STM32CubeMX proprietary format.

Figure 527. Selecting a CMSIS-Pack software component

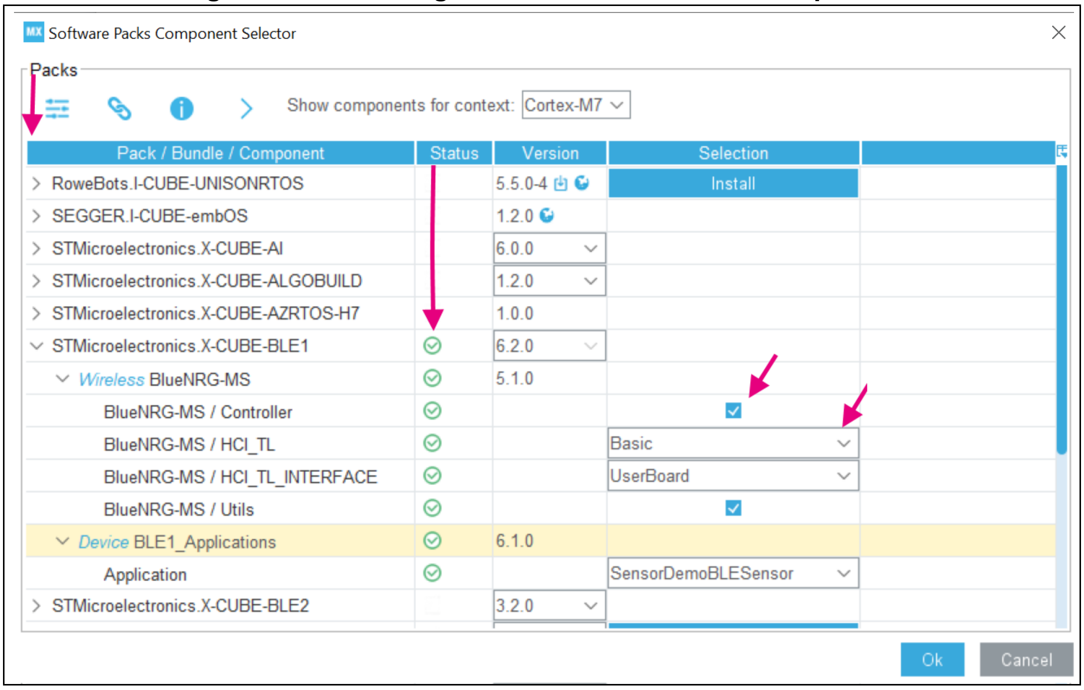


Figure 528. Enabling and configuring a CMSIS-Pack software component

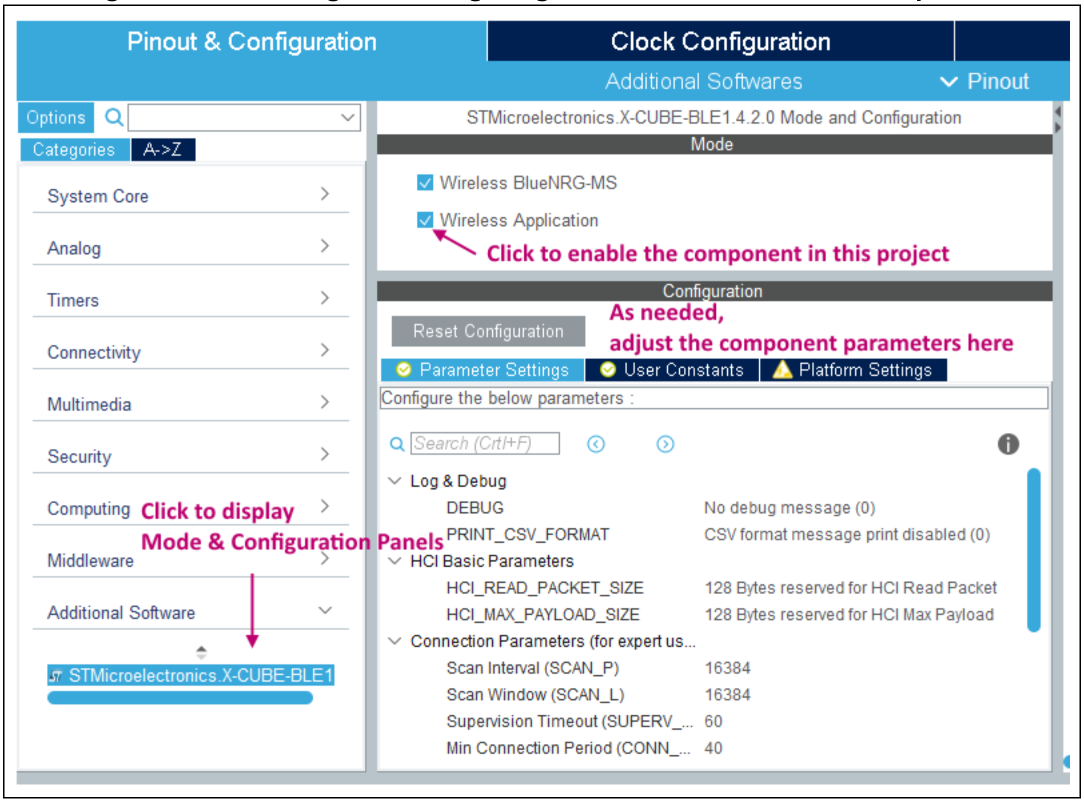
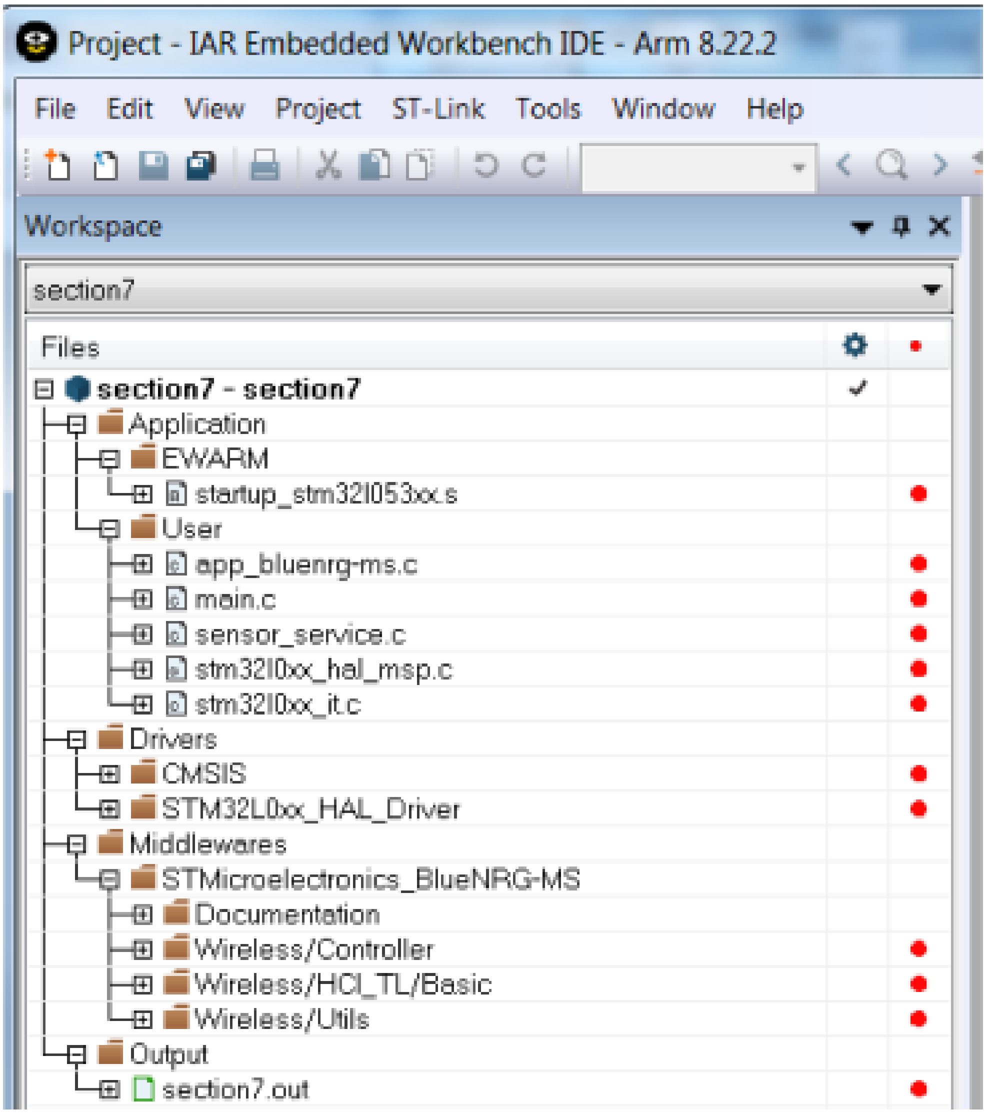


Figure 529. Project generated with CMSIS-Pack software component



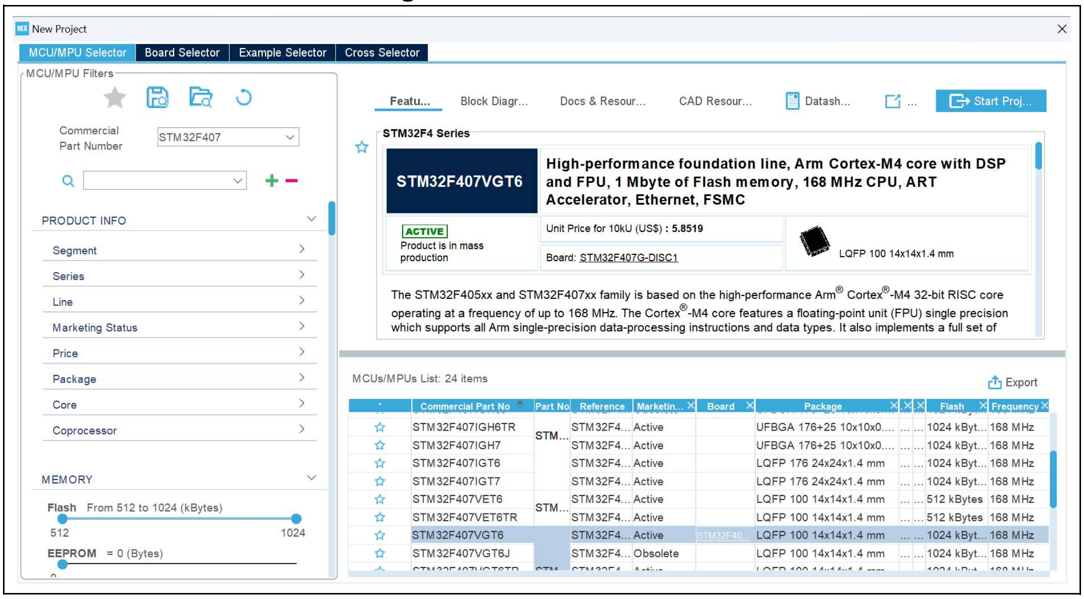
11 Tutorial 1: From pinout to project C code generation using an MCU of the STM32F4 series

This section describes the configuration and C code generation process. It takes as an example a simple LED toggling application running on the STM32F4DISCOVERY board.

11.1 Creating a new STM32CubeMX project

1. Select **File > New project** from the main menu bar or **New project** from the Home page.
2. Select the **MCU/MPU Selector** tab and filter down the STM32 portfolio by selecting STM32F4 as 'Series', STM32F407 as 'Lines', and LQFP100 as 'Package' (see *Figure 530*).
3. Select the STM32F407VGTx from the MCU list and click **OK**.

Figure 530. MCU selection



STM32CubeMX views are then populated with the selected MCU database (*Figure 531*). Optionally, remove the MCUs Selection bottom window by deselecting **Window > Outputs** submenu (see *Figure 532*).

Figure 531. Pinout view with MCUs selection

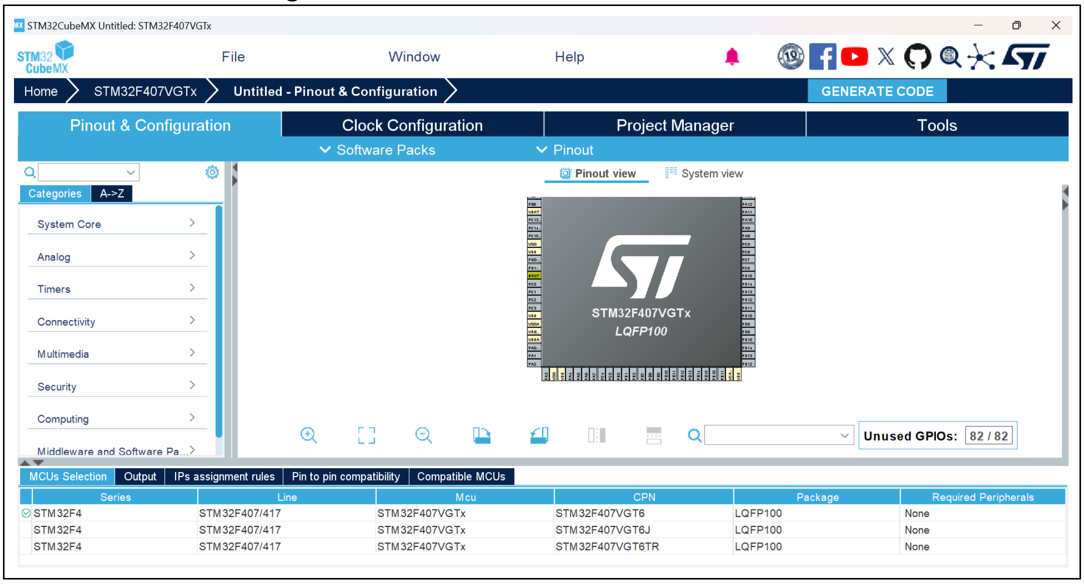


Figure 532. Pinout view without MCUs selection window

11.2 Configuring the MCU pinout

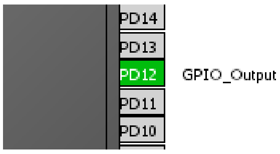
For a detailed description of menus, advanced actions and conflict resolutions, refer to *Section 4* and *Appendix A*.

1. By default, STM32CubeMX shows the **Pinout** view.
2. By default, ☐ Keep Current Signals Placement is unchecked, allowing STM32CubeMX to move the peripheral functions around and to find the optimal pin allocation (the one that accommodates the maximum number of peripheral modes).

   Since the MCU pin configurations must match the STM32F4DISCOVERY board, enable ☐ Keep Current Signals Placement for STM32CubeMX to maintain the peripheral function allocation (mapping) to a given pin. This setting is saved as a user preference, to be restored when reopening the tool or when loading another project.

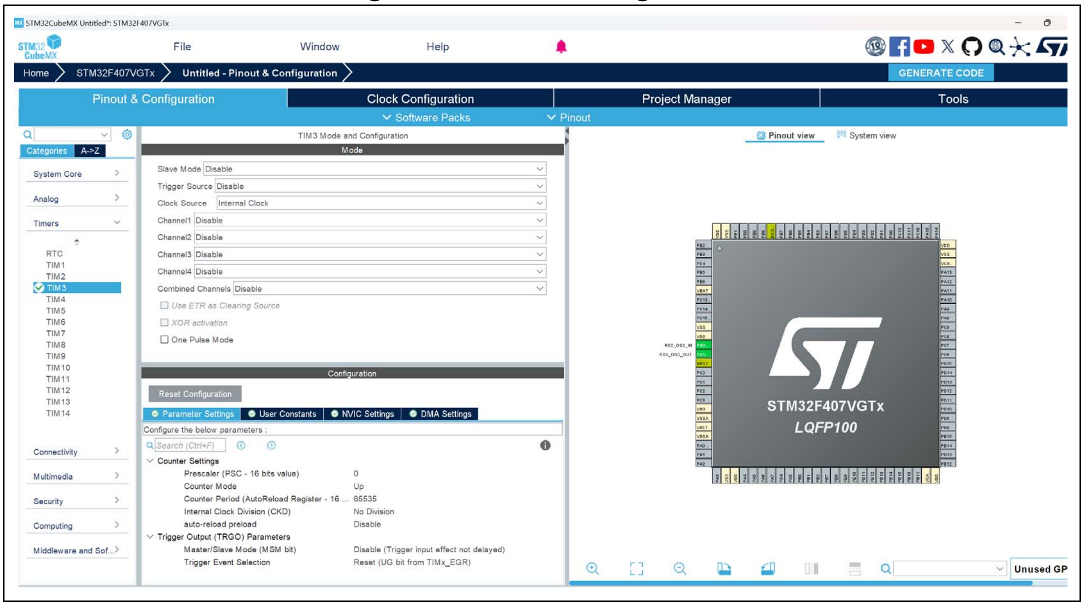
3. Select the required peripherals and peripheral modes:
   a) Configure the GPIO to output the signal on the STM32F4DISCOVERY green LED by right-clicking PD12 from the **Pinout** view, then select GPIO_output:

Figure 533. GPIO pin configuration



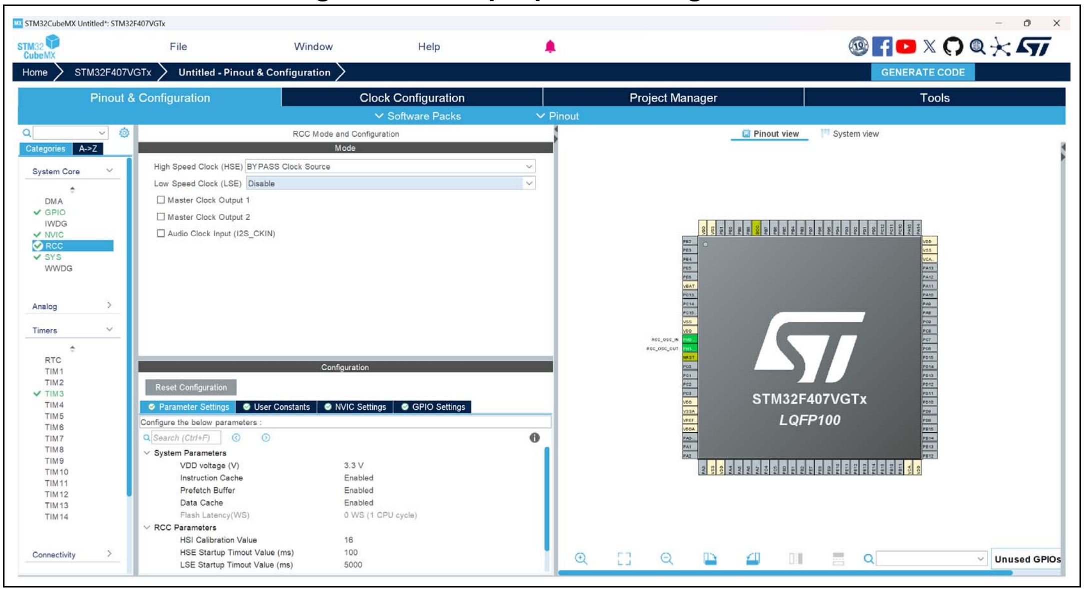
   b) Enable a timer to be used as timebase for toggling the LED. This is done by selecting Internal Clock as TIM3 clock source from the peripheral tree (see *Figure 534*).

Figure 534. Timer configuration



c) You can also configure the RCC to use an external oscillator as clock source (see *Figure 535*).

Figure 535. Simple pinout configuration



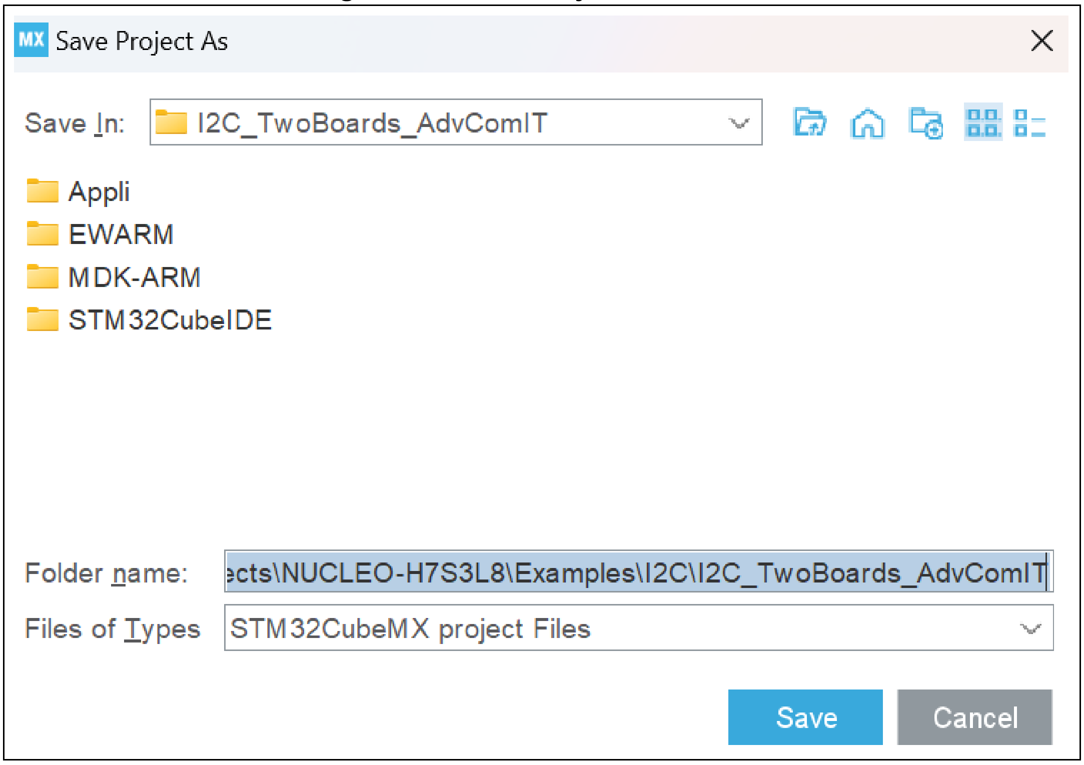
This completes the pinout configuration for this example.

*Note: Starting with STM32CubeMX 4.2, the user can skip the pinout configuration by directly loading ST Discovery board configuration from the **Board selector** tab.*

11.3 Saving the project

1. Click  to save the project.
   When saving for the first time, select a destination folder and filename for the project. The .ioc extension is added automatically to indicate that this is an STM32CubeMX configuration file.

Figure 536. Save Project As window



2. Click  to save the project under a different name or location.

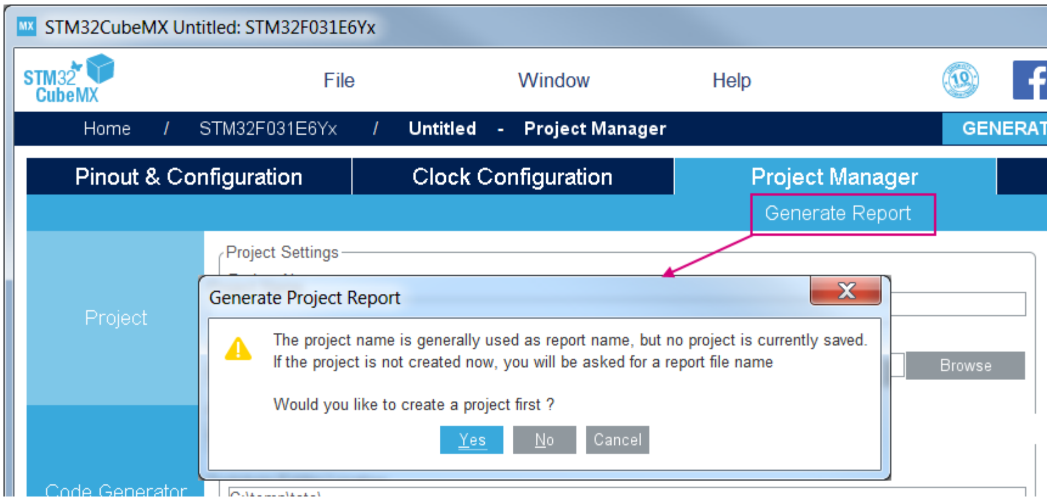
11.4 Generating the report

Reports can be generated at any time during the configuration:

1. Click to generate .pdf and .txt reports.

   If a project file has not been created yet, a warning prompts the user to save the project first and requests a project name and a destination folder (see *Figure 537*). An .ioc file is then generated for the project along with a .pdf and .txt reports with the same name.

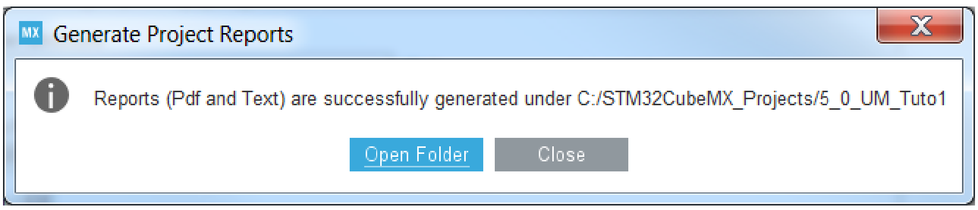
Figure 537. Generate Project Report - New project creation



Answering **No** will require to provide a name and location for the report only.

As shown in *Figure 538*, a confirmation message is displayed when the operation is successful.

Figure 538. Generate Project Report - Project successfully created



2. Open the .pdf report using Adobe Reader or the .txt report using your favorite text editor. The reports summarize all the settings and MCU configuration performed for the project.

11.5 Configuring the MCU clock tree

The following sequence describes how to configure the clocks required by the application based on an STM32F4 MCU.

STM32CubeMX automatically generates the system, CPU and AHB/APB bus frequencies from the clock sources and prescalers selected by the user. Wrong settings are detected and highlighted in fuchsia through a dynamic validation of minimum and maximum conditions. Useful tooltips provide a detailed description of the actions to undertake when the settings are unavailable or wrong. User frequency selection can influence some peripheral parameters (e.g. UART baud rate limitation).

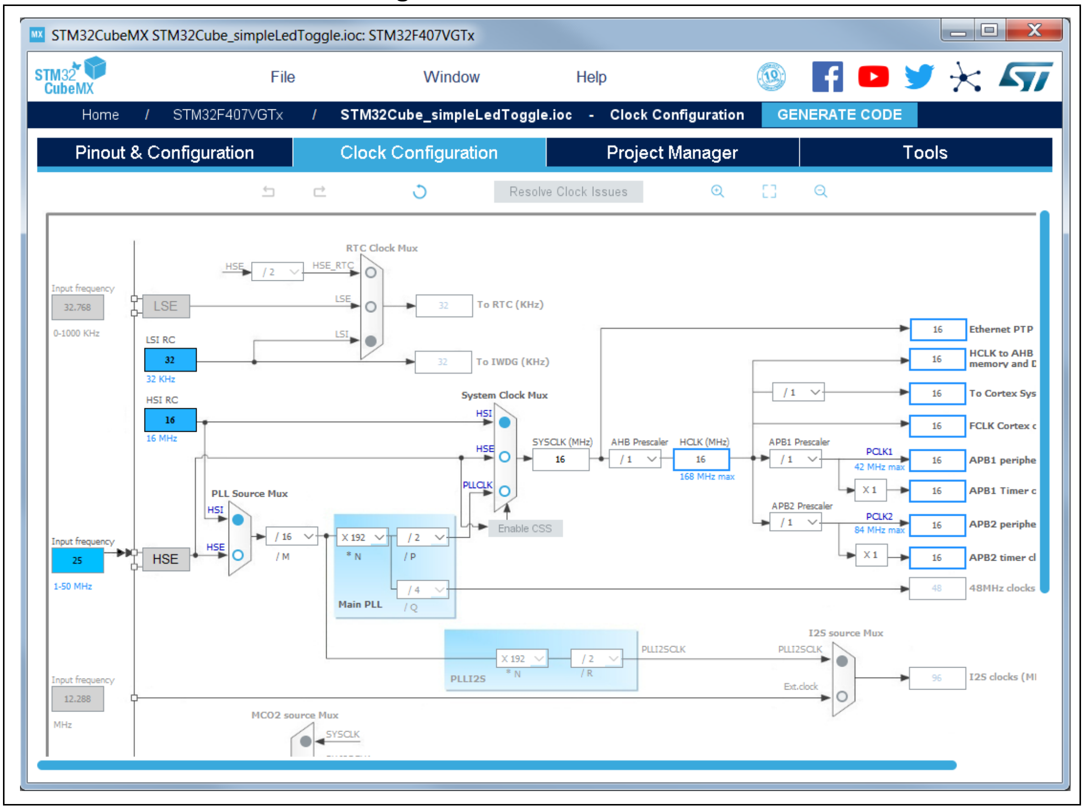
STM32CubeMX uses the clock settings defined in the Clock tree view to generate the initialization C code for each peripheral clock. Clock settings are performed in the generated C code as part of RCC initialization within the project main.c and in stm32f4xx_hal_conf.h (HSE, HSI and external clock values expressed in Hertz).

Follow the sequence below to configure the MCU clock tree:

1. Click the **Clock Configuration** tab to display the clock tree (see *Figure 539*).

   The internal (HSI, LSI), system (SYSCLK) and peripheral clock frequency fields cannot be edited. The system and peripheral clocks can be adjusted by selecting a clock source, and optionally by using the PLL, prescalers and multipliers.

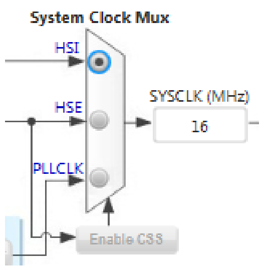
Figure 539. Clock tree view



2. Select the clock source (HSE, HSI or PLLCLK) that will drive the system clock.

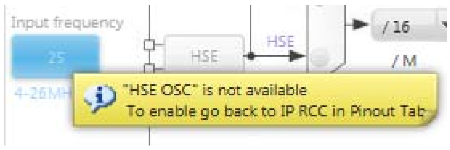
   In the example taken for the tutorial, select HSI to use the internal 16 MHz clock (see *Figure 540*).

Figure 540. HSI clock enabled



To use an external clock source (HSE or LSE), the RCC peripheral must be configured in the **Pinout** view, as pins will be used to connect the external clock crystals (see *Figure 541*).

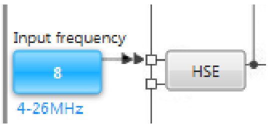
Figure 541. HSE clock source disabled



Other clock configuration options for the STM32F4DISCOVERY board:

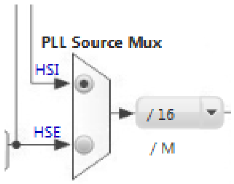
– Select the external HSE source and enter 8 in the HSE input frequency box since an 8 MHz crystal is connected on the discovery board:

Figure 542. HSE clock source enabled



– Select the external PLL clock source and the HSI or HSE as the PLL input clock source.

Figure 543. External PLL clock source enabled



3. Keep the core and peripheral clocks to 16 MHz using HSI, no PLL and no prescaling.

Note: Optionally, further adjust the system and peripheral clocks using PLL, prescalers and multipliers:

Other clock sources independent from the system clock can be configured as follows:

- *USB OTG FS, RNG and SDIO clocks are driven by an independent PLL output.*
- *I2S peripherals come with their own internal clock (PLLI2S), alternatively derived by an independent external clock source.*
- *USB OTG HS and Ethernet clocks are derived from an external source.*

4. Optionally, configure the prescaler for the Microcontroller Clock Output (MCO) pins that allow to output two clocks to the external circuit.
5. Click to save the project.
6. Go to the **Configuration** tab to proceed with the project configuration.

11.6 Configuring the MCU initialization parameters

Caution: The C code generated by STM32CubeMX covers the initialization of the MCU peripherals and middlewares using the STM32Cube firmware libraries.

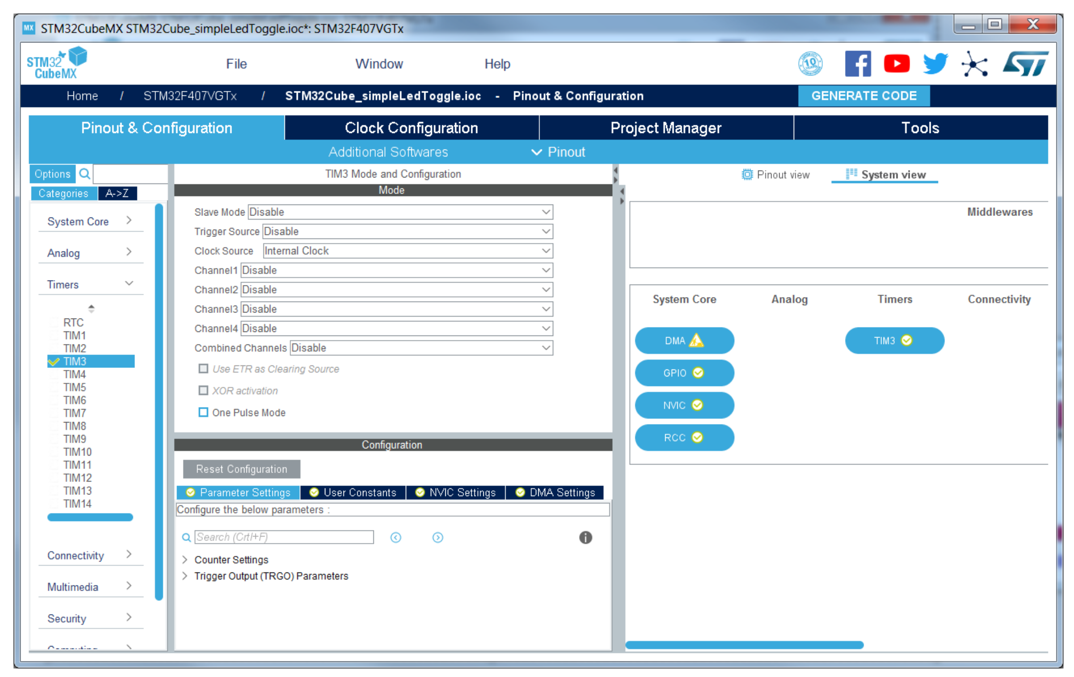
11.6.1 Initial conditions

From the **Pinout & Configuration** tab, select and configure (one by one) every component (peripheral, middleware, additional software) required by the application using the **Mode** and **Configuration** panels (see *Figure 544*).

Tooltips and warning messages are displayed when peripherals are not properly configured (see *Section 4* for details).

Note: The ***RCC*** *peripheral initialization uses the parameter configuration done in this view as well as the configuration done in the* ***Clock tree*** *view (clock source, frequencies, prescaler values).*

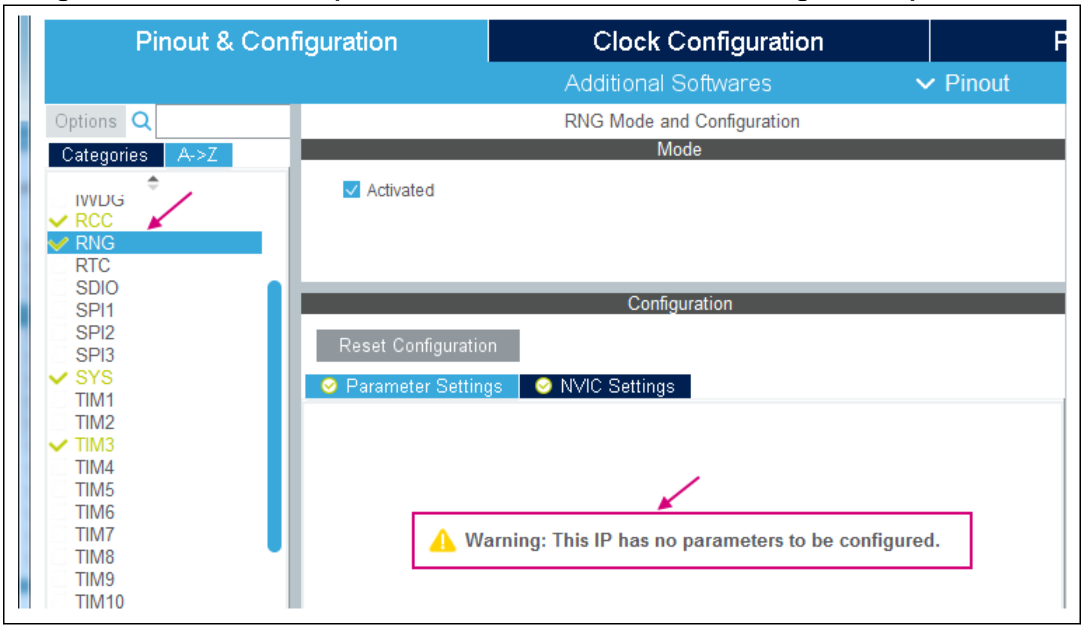
Figure 544. Pinout & Configuration view



11.6.2 Configuring the peripherals

Each peripheral instance corresponds to a dedicated button in the main panel. Some peripheral modes have no configurable parameters, as illustrated below.

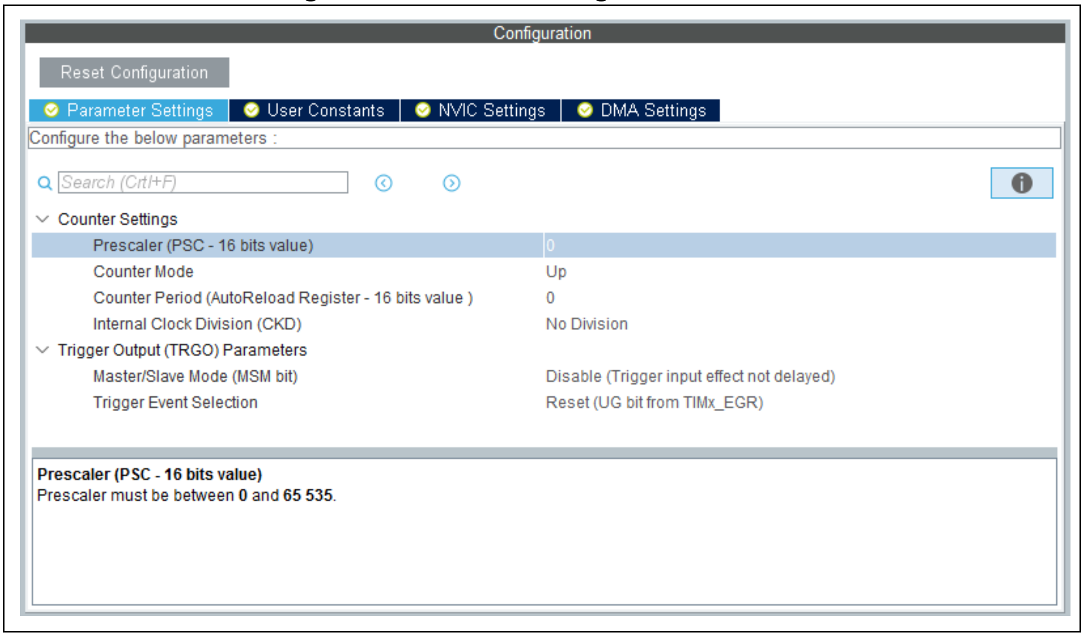
Figure 545. Case of Peripheral and Middleware without configuration parameters



Follow the steps below to proceed with peripheral configuration:

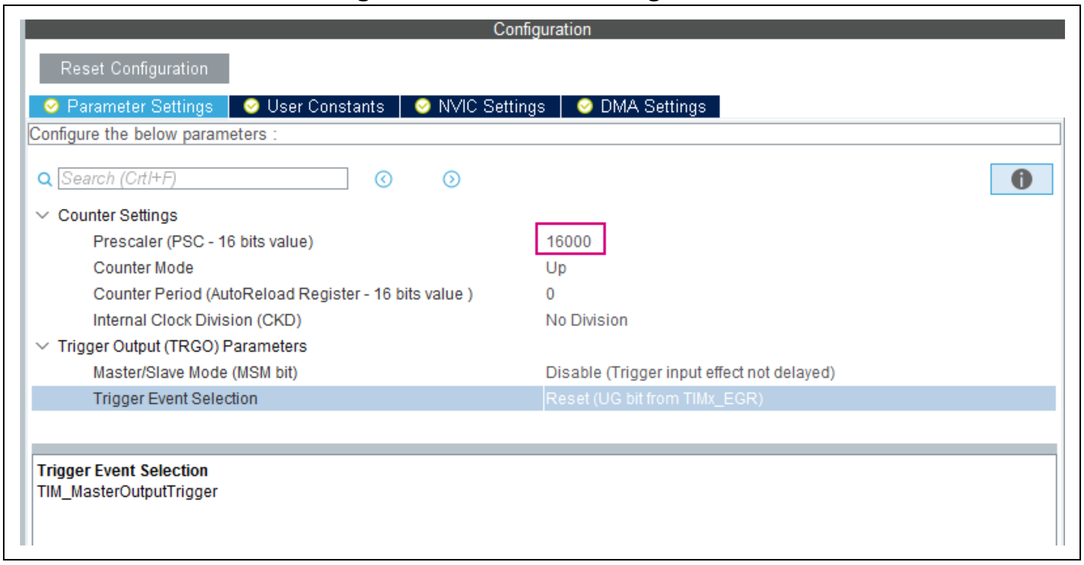
1. Click the peripheral button to open the corresponding configuration window.
   In our example
   a) click **TIM3** to open the timer configuration window.

Figure 546. Timer 3 configuration window



   b) with a 16 MHz APB clock (Clock tree view), set the prescaler to 16000 and the counter period to 1000 to make the LED blink every millisecond.

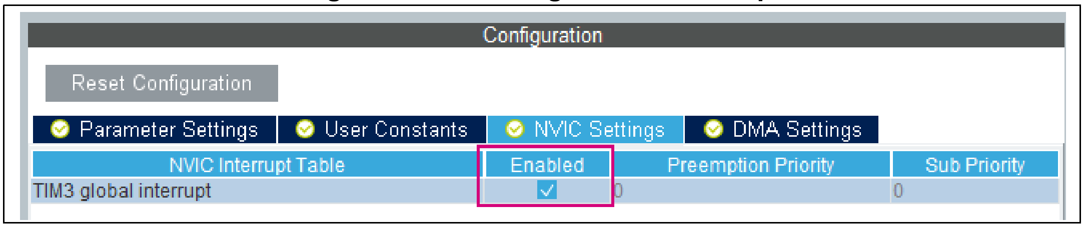
Figure 547. Timer 3 configuration



2. Optionally, and when available, select:
   – The **NVIC Settings** tab to display the NVIC configuration and enable interruptions for this peripheral.
   – The **DMA Settings** tab to display the DMA configuration and to configure DMA transfers for this peripheral.

     In the tutorial example, the DMA is not used and the GPIO settings remain unchanged. The interrupt is enabled, as shown in *Figure 548*.
   – The **GPIO Settings** tab to display the GPIO configuration and to configure the GPIOs for this peripheral.
   – Insert an item:
   – The **User Constants** tab to specify constants to be used in the project.

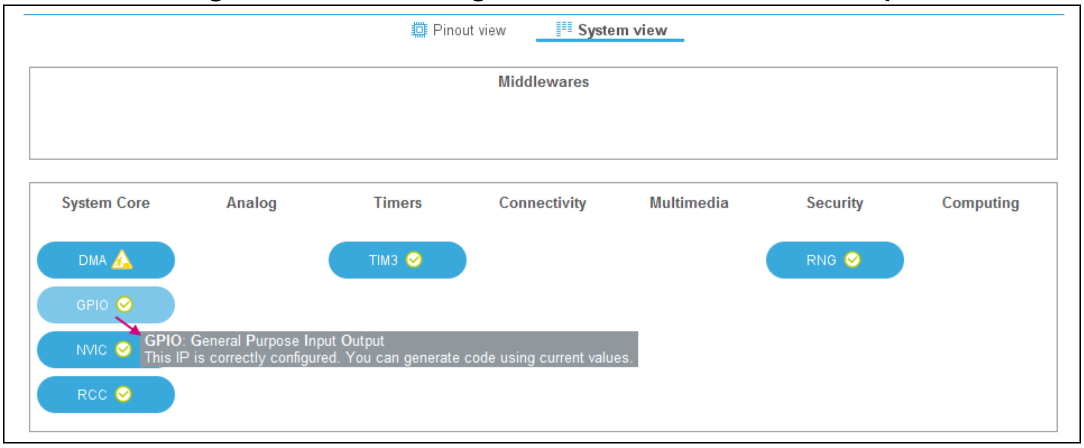
Figure 548. Enabling Timer 3 interrupt



11.6.3 Configuring the GPIOs

The user can adjust all pin configurations from this window. A small icon along with a tooltip indicates the configuration status.

Figure 549. GPIO configuration color scheme and tooltip

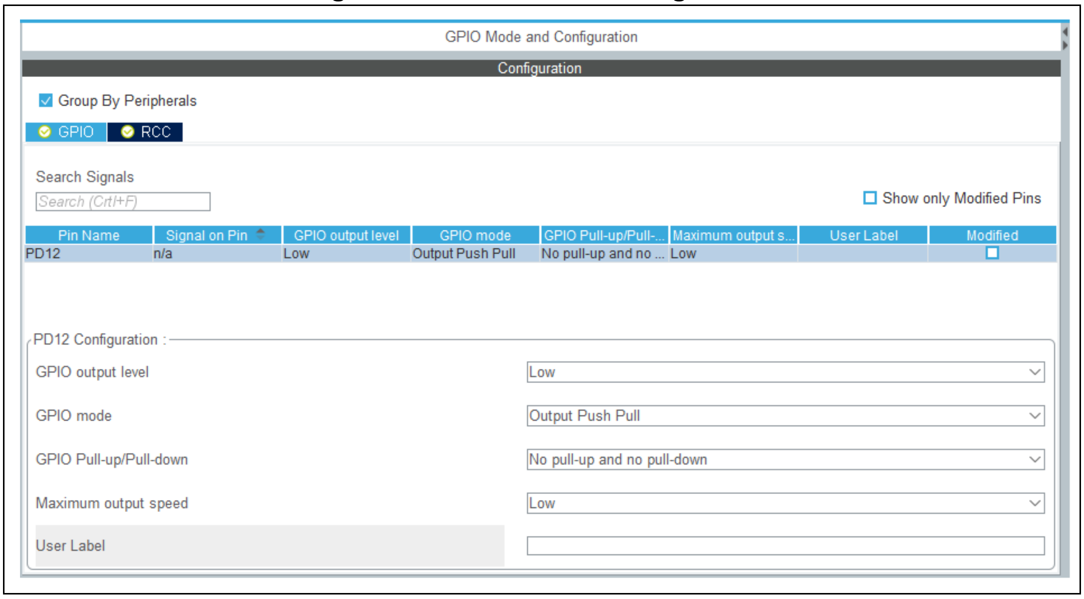


Follow the sequence below to configure the GPIOs:

1. Click the **GPIO button** in the Configuration view to open the **Pin Configuration** window.
2. The first tab shows pins that have been assigned a GPIO mode, but not for a dedicated peripheral and middleware. Select Pin Name to open the configuration for that pin.

   In the tutorial example, select PD12 and configure it in output push-pull mode to drive the STM32F4DISCOVERY LED (see *Figure 550*).

Figure 550. GPIO mode configuration



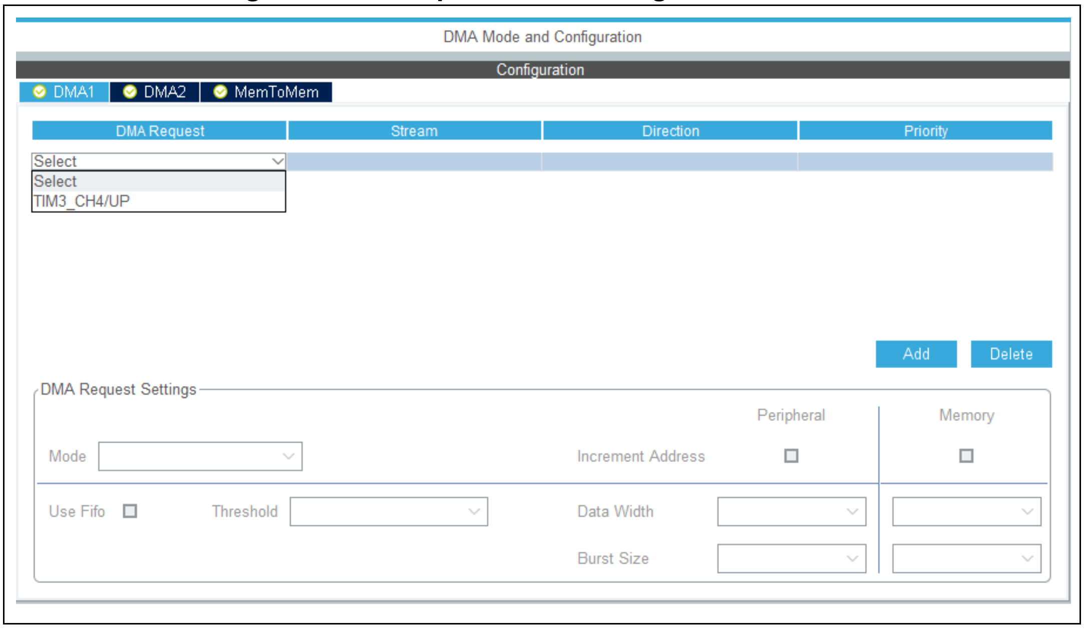
11.6.4 Configuring the DMAs

This is not required for this example. It is recommended to use DMA transfers to offload the CPU. The DMA Configuration window provides a fast and easy way to configure the DMAs (see *Figure 551*):

1. add a new DMA request and select among a list of possible configurations.
2. select among the available streams.
3. select the Direction: Memory to Peripheral or Peripheral to Memory.
4. select a Priority.
5. enable the FIFO.

Note: Configuring the DMA for a given peripheral and middleware can also be performed using the Peripheral and Middleware configuration window.

Figure 551. DMA parameters configuration window

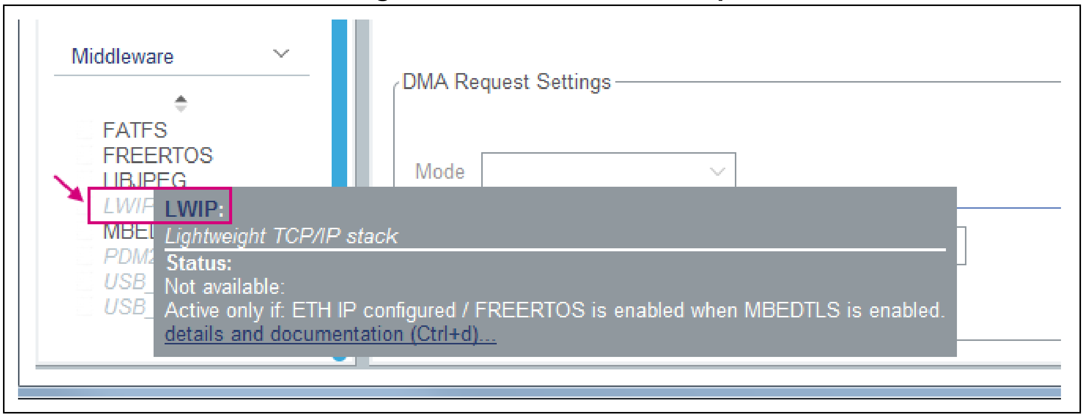


11.6.5 Configuring the middleware

This is not required for the example taken for the tutorial.

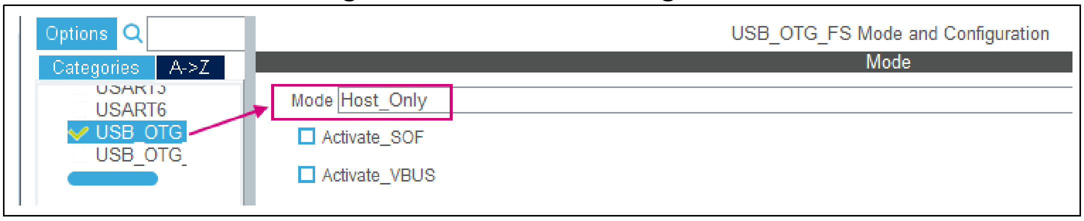
If a peripheral is required for a middleware mode, the peripheral must be configured in the **Pinout** view for the middleware mode to become available. A tooltip can guide the user as shown below.

Figure 552. Middleware tooltip



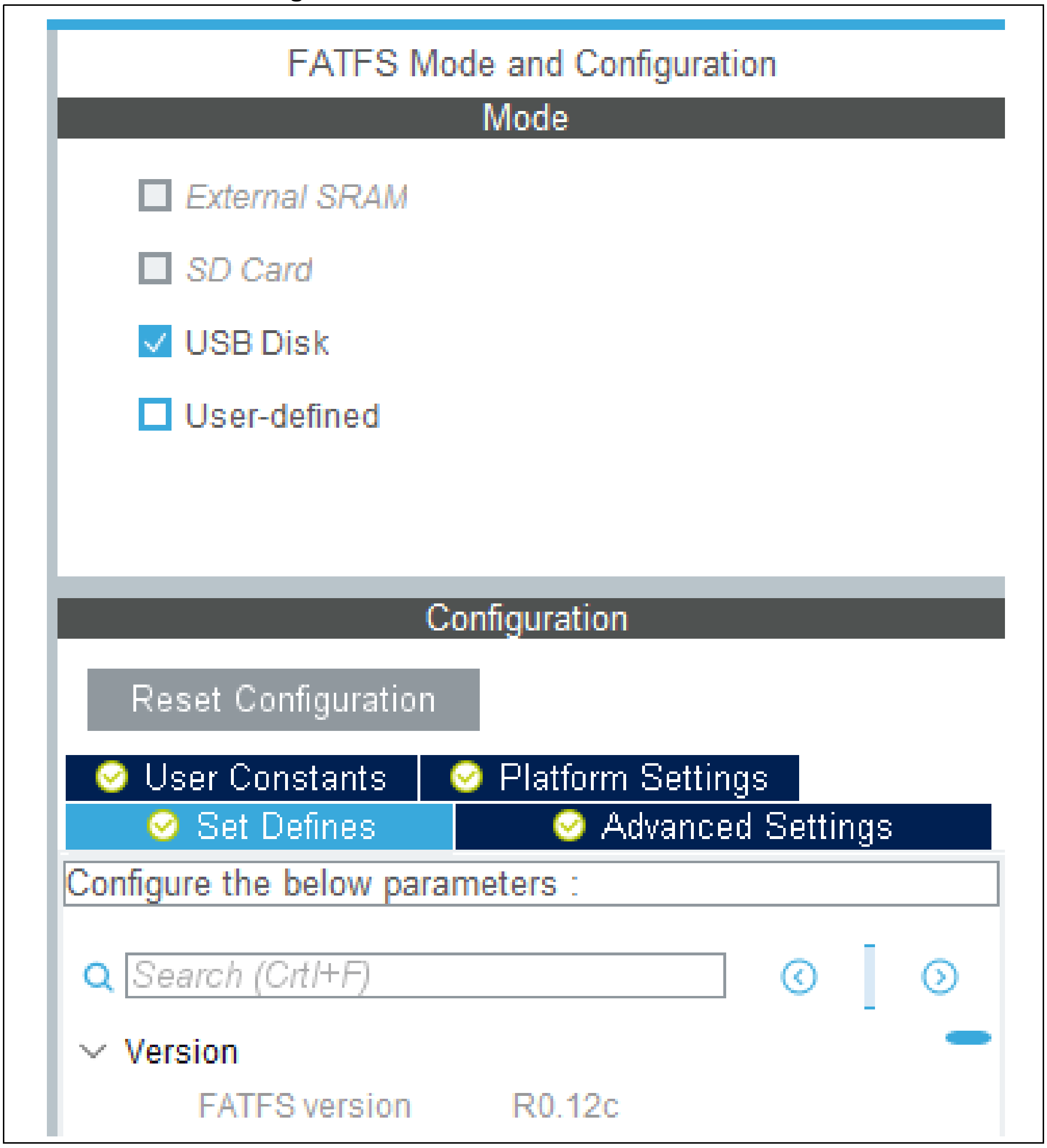
1. Configure the USB peripheral from the **Pinout** view.

Figure 553. USB Host configuration



2. Select MSC_FS class from USB Host middleware.
3. Select the checkbox to enable FatFs USB mode in the tree panel.

Figure 554. FatFs over USB mode enabled

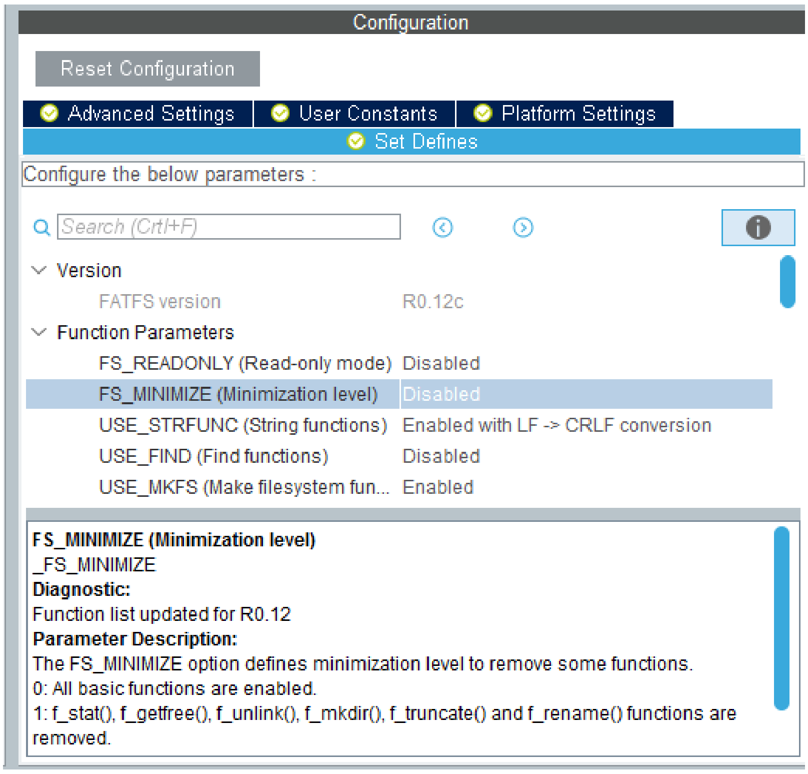


4. Select the **Configuration** view. FatFs and USB buttons are then displayed.

Figure 555. System view with FatFs and USB enabled

5. FatFs and USB using default settings are already marked as configured . Click **FatFs** and **USB** buttons to display default configuration settings. You can also change them by following the guidelines provided at the bottom of the window.

Figure 556. FatFs define statements



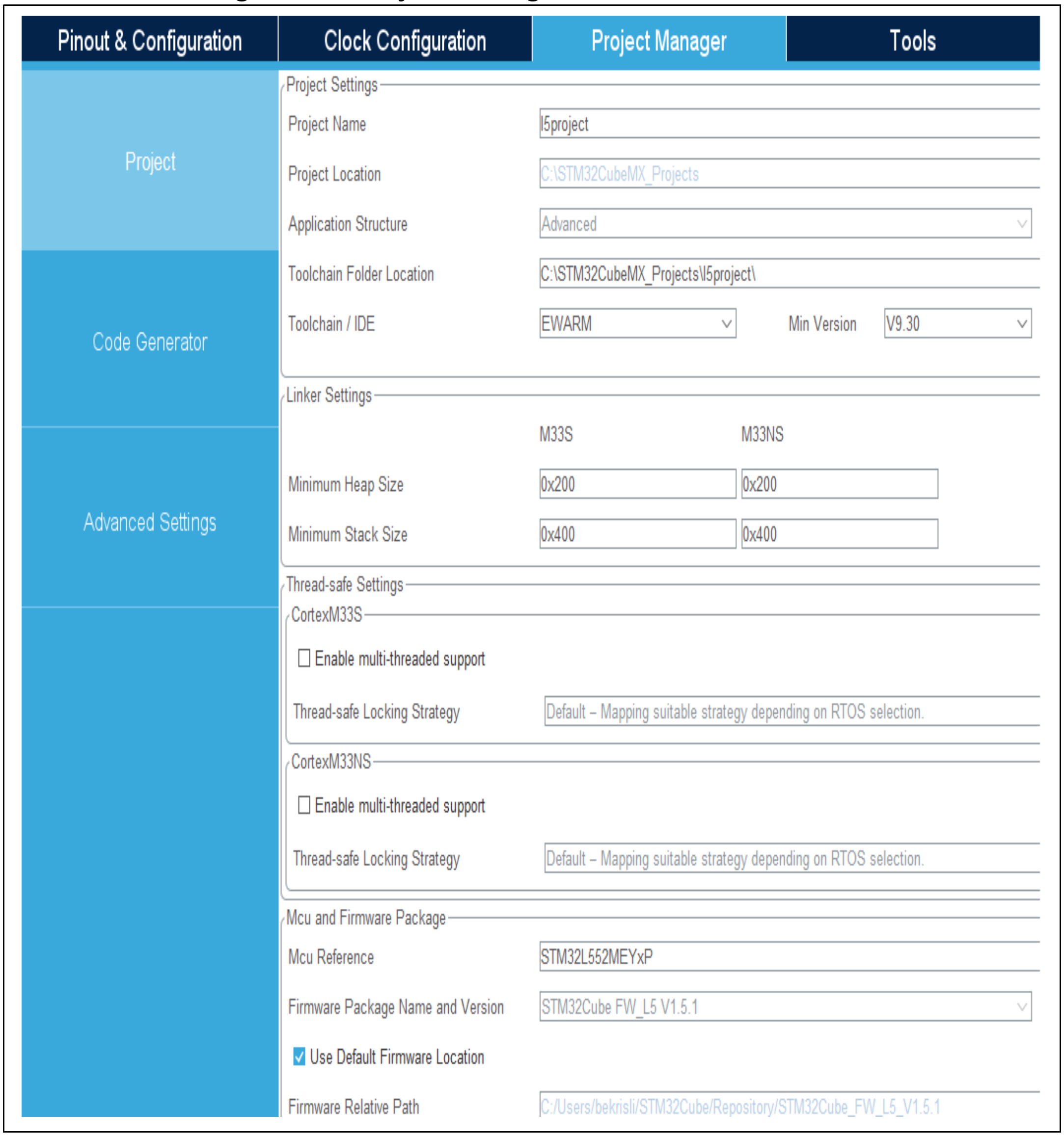
11.7 Generating a complete C project

11.7.1 Setting project options

Default project settings can be adjusted prior to C code generation as shown in *Figure 557*.

1. Select the **Project Manager** view to update project settings and generation options.
2. Select the **Project Tab** and choose a Project **name**, **location,** a **toolchain** and a **toolchain version** to generate the project (see *Figure 557*).

Figure 557. Project Settings and toolchain selection

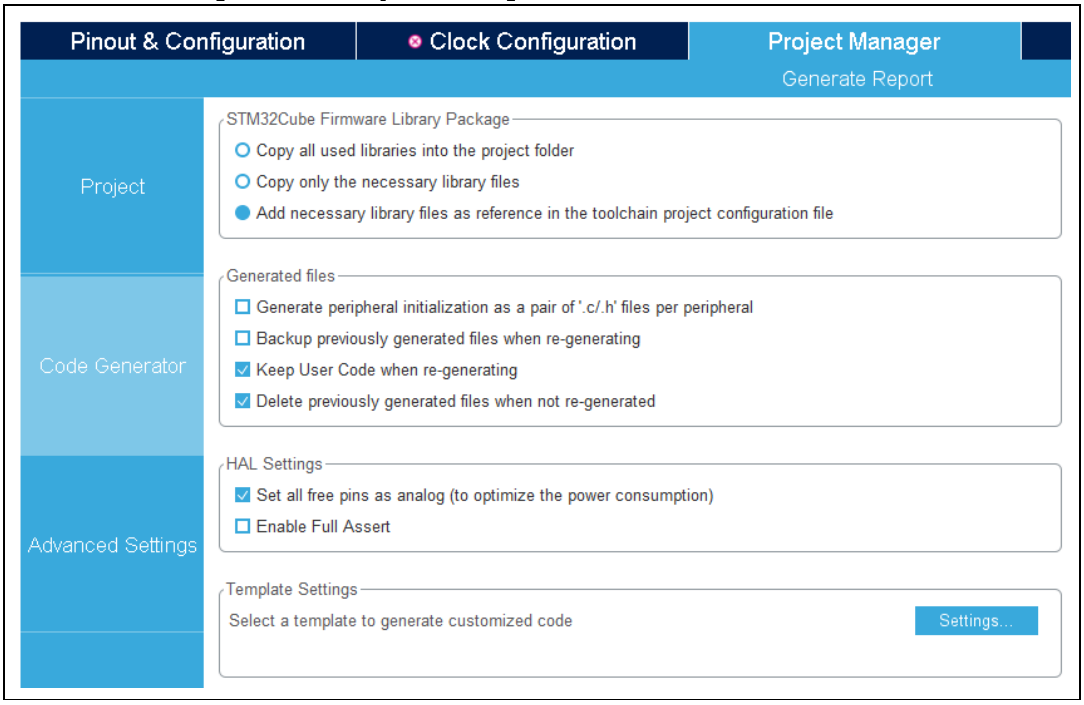


3. Select the **Code Generator** tab to choose various C code generation options:
   – The library files copied to *Projects* folder.
   – C code regeneration (e.g. what is kept or backed up during C code regeneration).
   – HAL specific action (for example, set all free pins as analog I/Os to reduce power consumption).

   In the tutorial example, select the settings as displayed in *Figure 558*, and click **OK**.

Note: A dialog window appears when the firmware package is missing. Go to next section for explanation on how to download the firmware package.

Figure 558. Project Manager menu - Code Generator tab

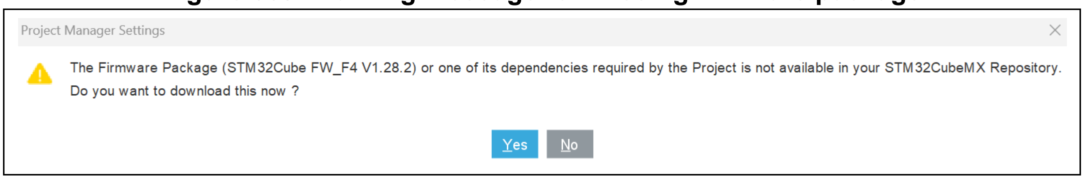


11.7.2 Downloading firmware package and generating the C code

1. Click GENERATE CODE to generate the C code.

   During the generation, STM32CubeMX copies files from the relevant STM32Cube MCU package into the project folder so that the project can be compiled. When generating a project for the first time, the firmware package is not available on the user PC and a warning is displayed.

Figure 559. Warning message for missing firmware package

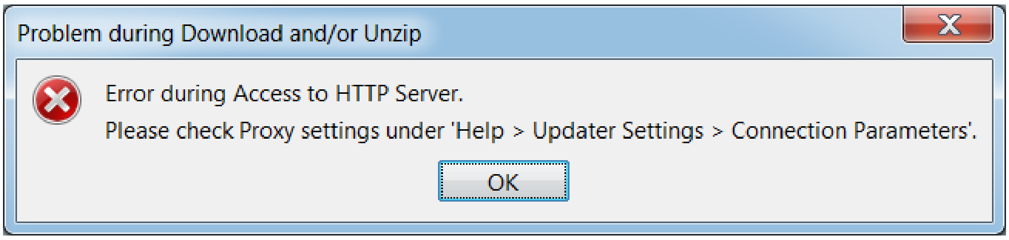


2. STM32CubeMX offers to download the relevant firmware package or to go on. Click **Download** to obtain a complete project, ready to use in the selected IDE.

   By clicking **Continue**, only *Inc* and *Src* folders are created, holding STM32CubeMX generated initialization files. The necessary firmware and middleware libraries must be copied manually to obtain a complete project.

   If the download fails, an error message is displayed.

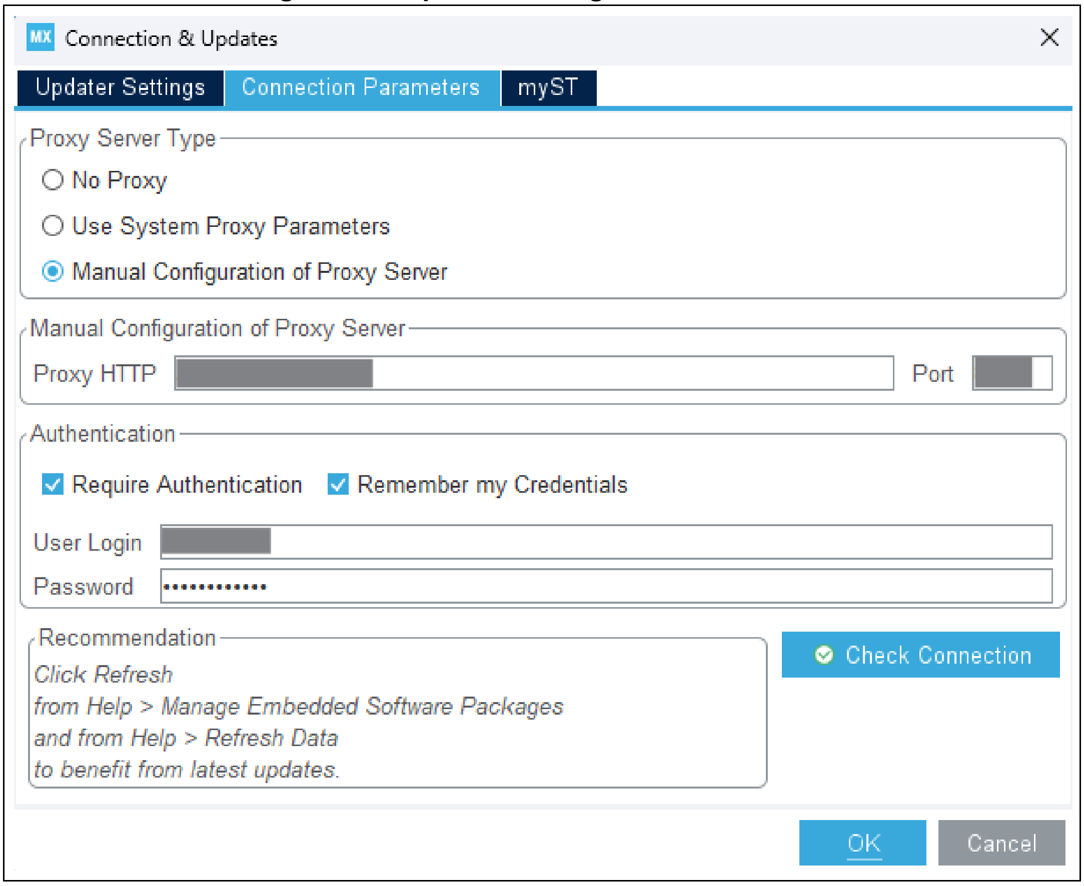
Figure 560. Error during download



To solve this issue, execute the next two steps. Skip them otherwise.

3. Select **Help > Connection & Updates**, a new window appears.
4. Go to the Connection Parameters tab and adjust the parameters to match your network configuration.
5. Click **Check connection.** The check mark turns green once the connection is established.

Figure 561. Updated settings with connection



6. Once the connection is functional, click GENERATE CODE to generate the C code. The progress is displayed (see next figures).

Figure 562. Downloading the firmware package

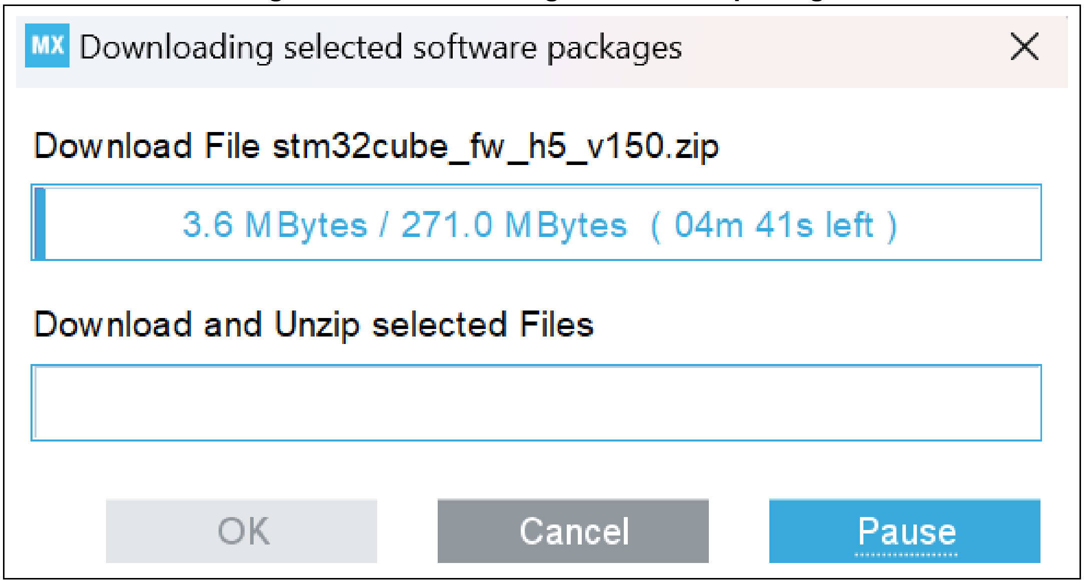
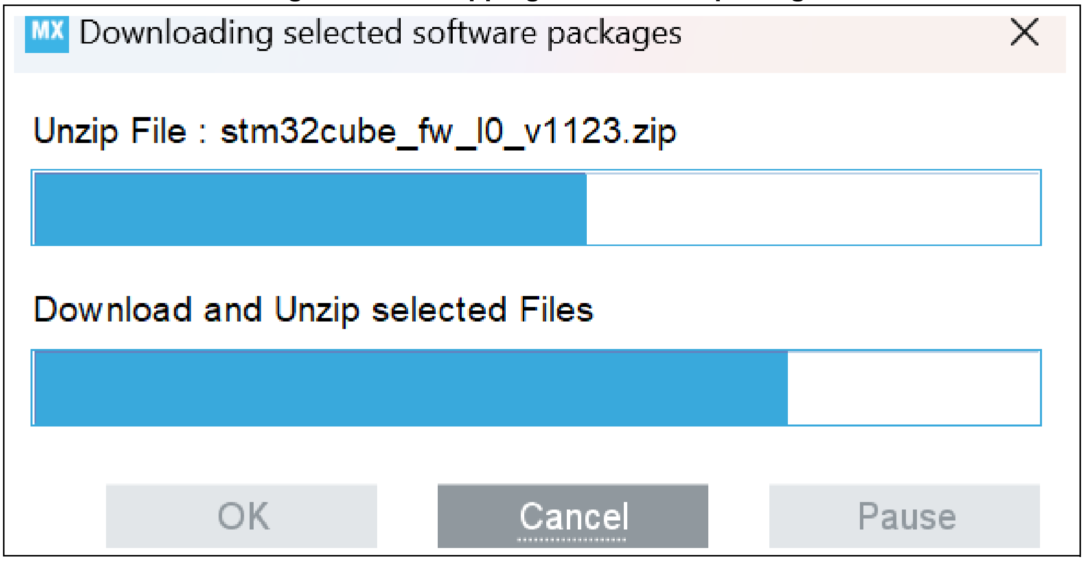
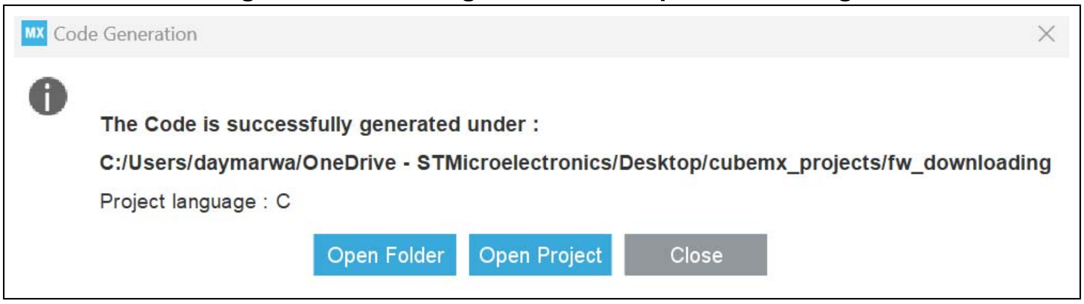


Figure 563. Unzipping the firmware package



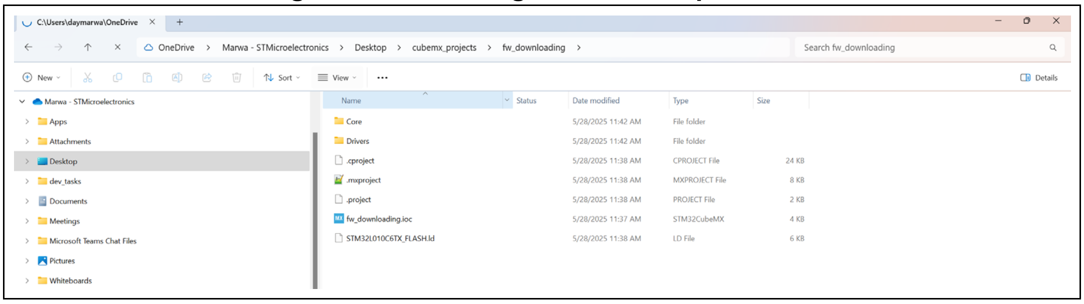
7. Finally, a confirmation message is displayed to indicate that the C code generation has been successful.

Figure 564. C code generation completion message



8. Click **Open Folder** to display the generated project contents or click **Open Project** to open the project directly in your IDE. Then proceed with *Section 11.8*.

Figure 565. C code generation output folder



The generated project contains:

- The STM32CubeMX .ioc project file located in the root folder. It contains the project user configuration and settings generated through STM32CubeMX user interface.
- The *Drivers* and *Middlewares* folders hold copies of the firmware package files relevant for the user configuration.
- The *Projects* folder contains IDE specific folders with all the files required for the project development and debug within the IDE.
- The *Inc* and *Src* folders contain STM32CubeMX generated files for middleware, peripheral and GPIO initialization, including the main.c file. The STM32CubeMX generated files contain user-dedicated sections allowing to insert user-defined C code.

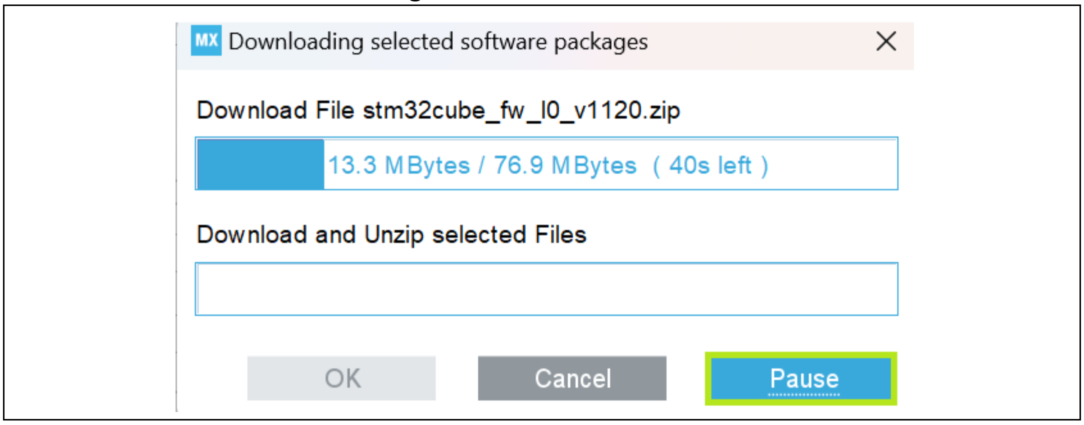
Caution: C code written within the user sections is preserved at next C code generation, while C code written outside these sections is overwritten.

User C code is lost if user sections are moved or if user sections delimiters are renamed.

STM32CubeMX includes a feature that allows users to pause and resume the download actions. This capability is particularly useful in scenarios where network connectivity is unstable or frequently interrupted. This also offers the ability to verify network issue, and connection speed.

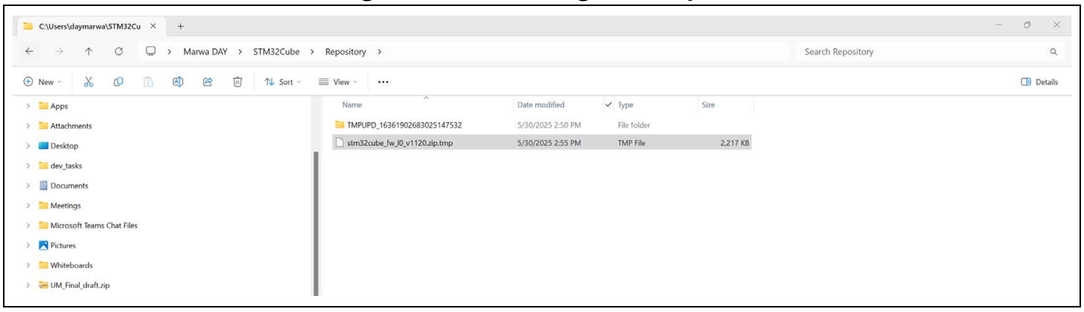
The user does not need to restart large downloads from scratch after a brief Internet hiccup. To stop the download, click the Pause button.

Figure 566. Pause button



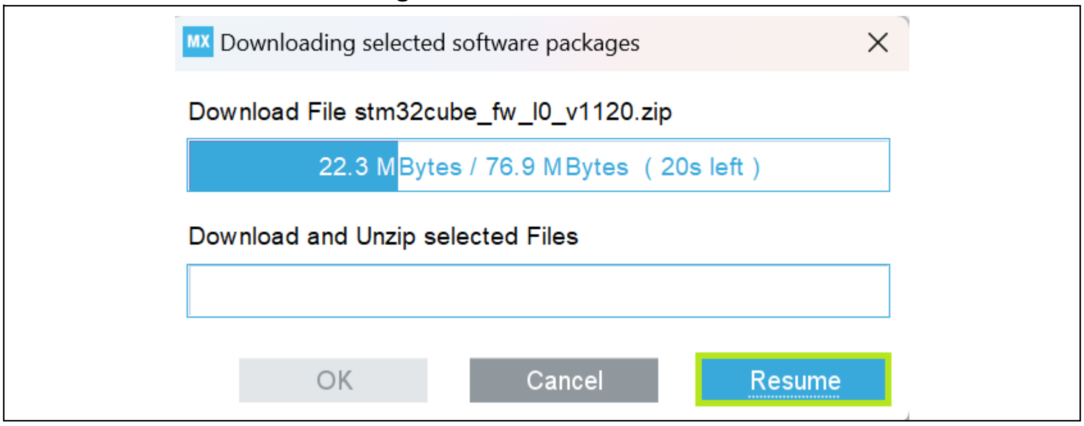
Once the Pause button has been clicked, a temporary file is created under the path ~/STM32Cube/Repository, to keep the download context

Figure 567. Creating the .tmp file



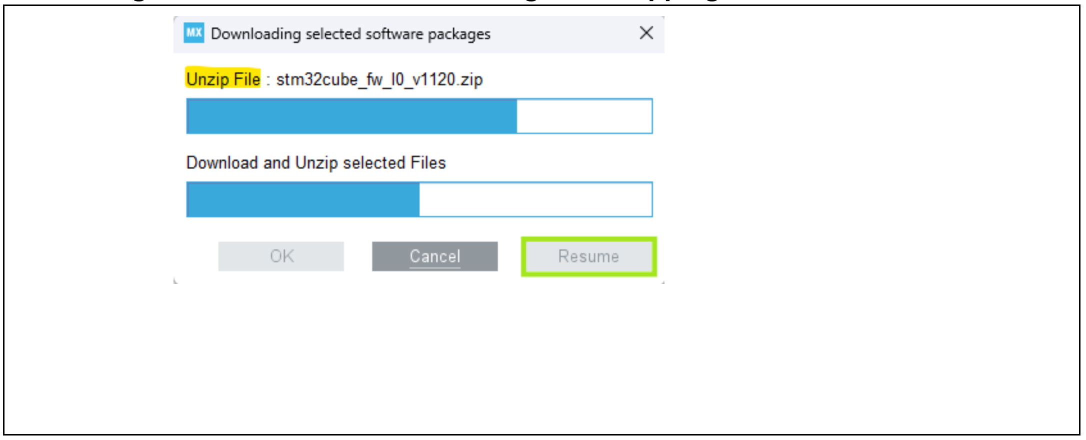
To restart the download, click the Resume button.

Figure 568. Resume button



The button is grayed if the download has been completed, and the unzip of the downloaded folder is started.

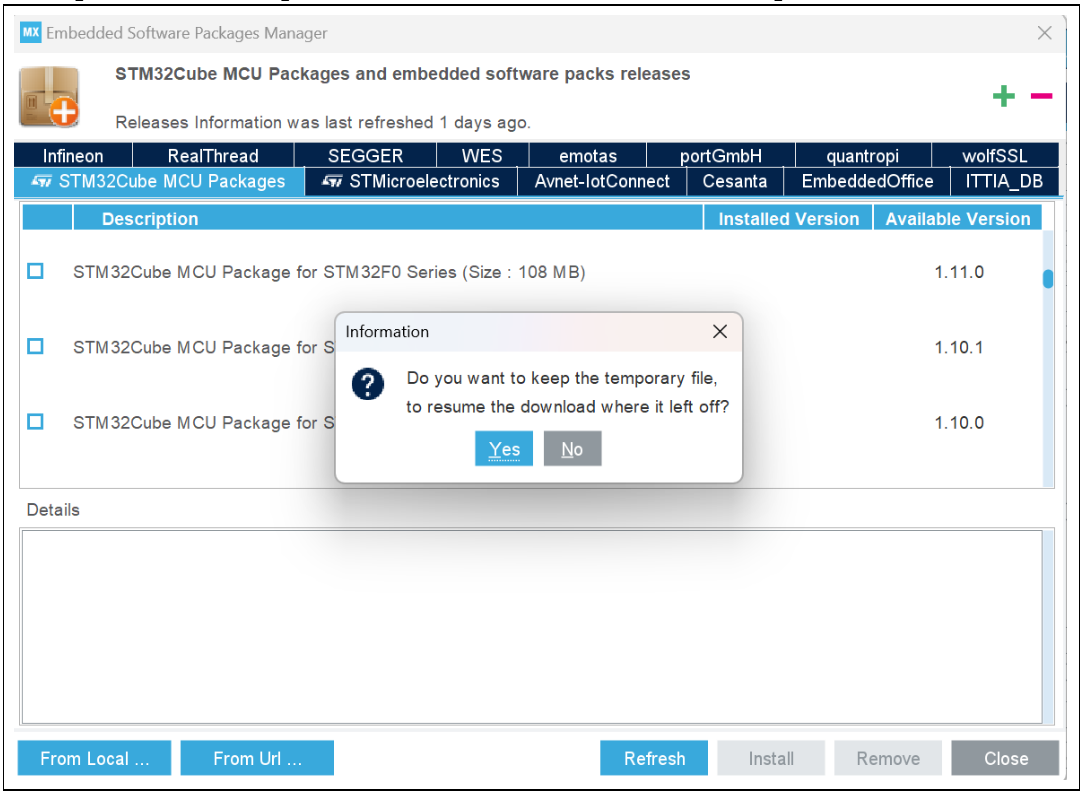
Figure 569. Resume button during the unzipping of downloaded files



When closing the download dialog box an additional window appears, prompting the user to decide whether to keep the temporary file:

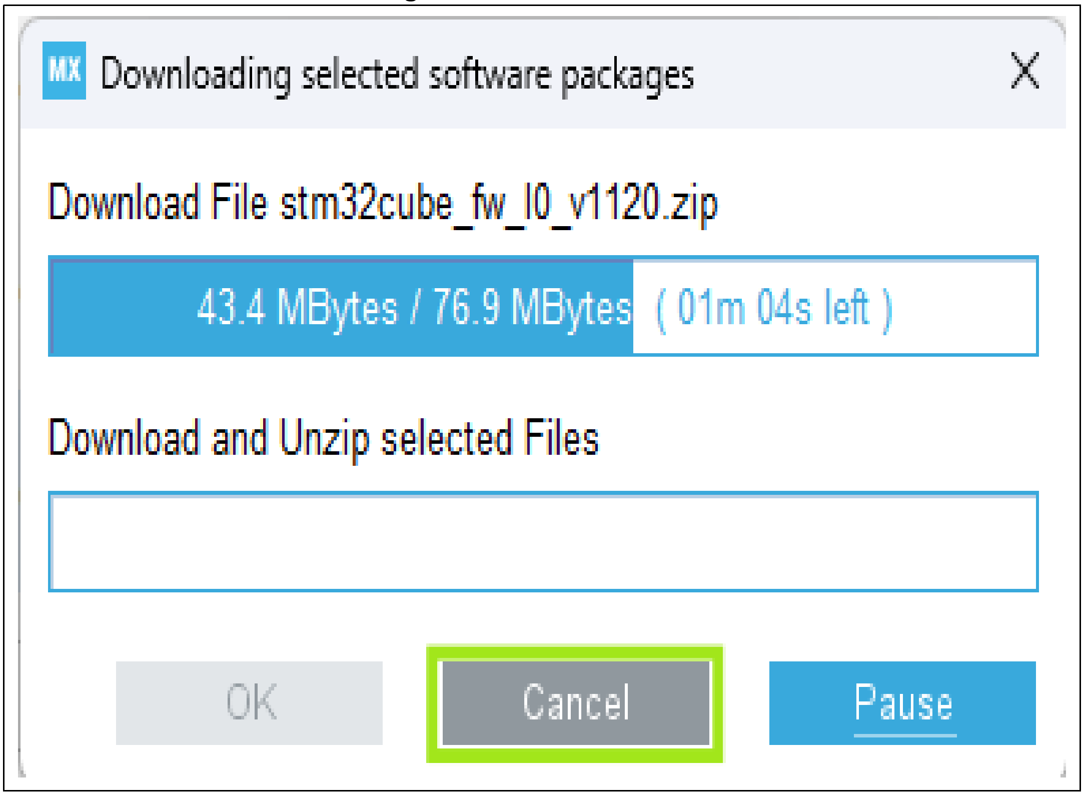
- If the user clicks Yes, the system retains the temporary file to facilitate the download.
- If the user clicks No, the system deletes the temporary file.

Figure 570. Closing the download window without clicking the Resume button



When the Cancel button is clicked, all procedures are canceled. The system stops the download and deletes the temporary file.

Figure 571. Cancel button



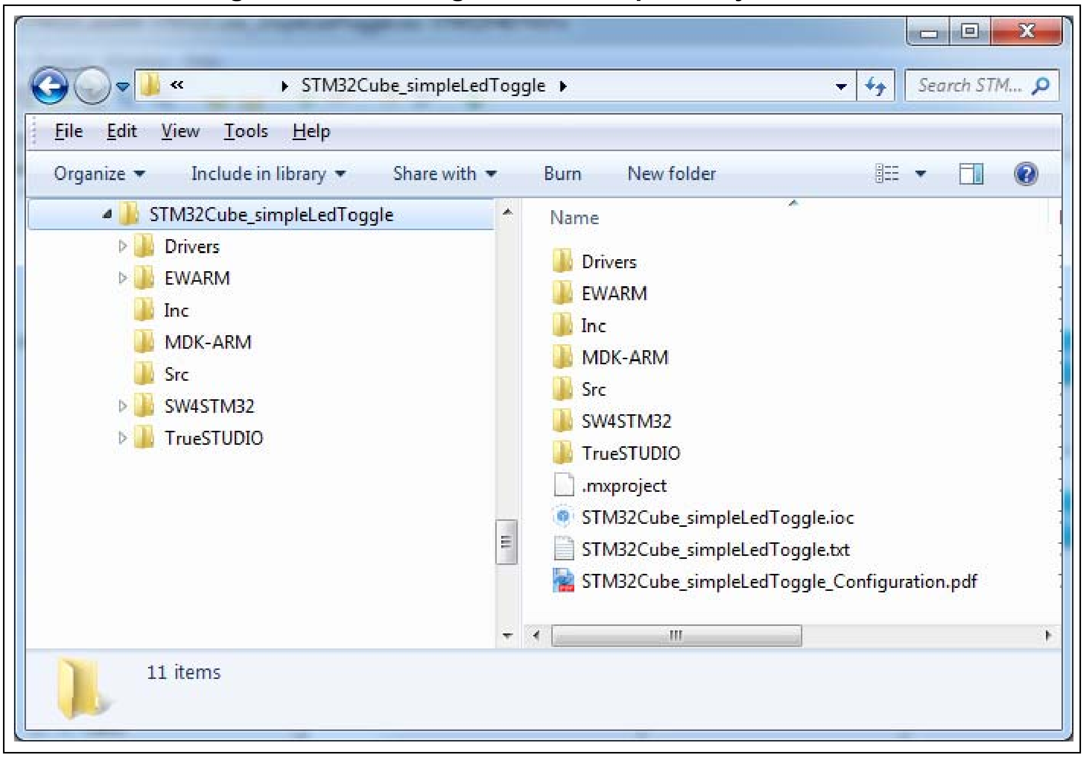
11.8 Building and updating the C code project

This example explains how to use the generated initialization C code and complete the project, within IAR™ EWARM toolchain, to have the LED blink according to the TIM3 frequency.

A folder is available for the toolchains selected for C code generation: the project can be generated for more than one toolchain by choosing a different toolchain from the **Project Manager** menu and clicking Generate code once again.

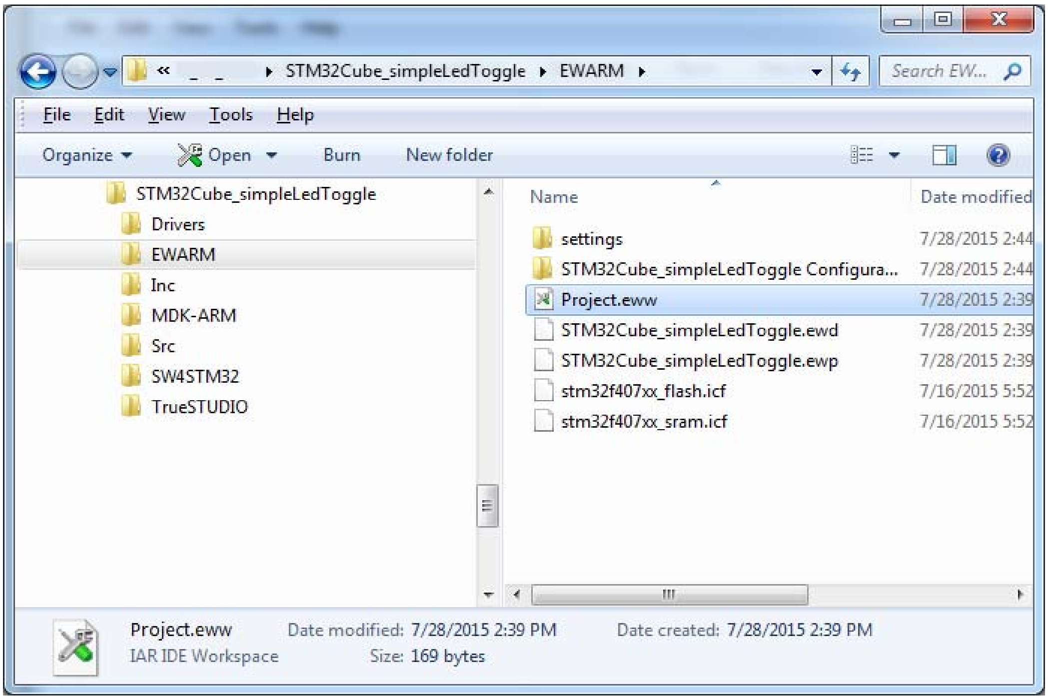
1. Open the project directly in the IDE toolchain by clicking **Open Project** from the dialog window or by double-clicking the relevant IDE file available in the toolchain folder under STM32CubeMX generated project directory (see *Figure 564*).

Figure 572. C code generation output: Projects folder



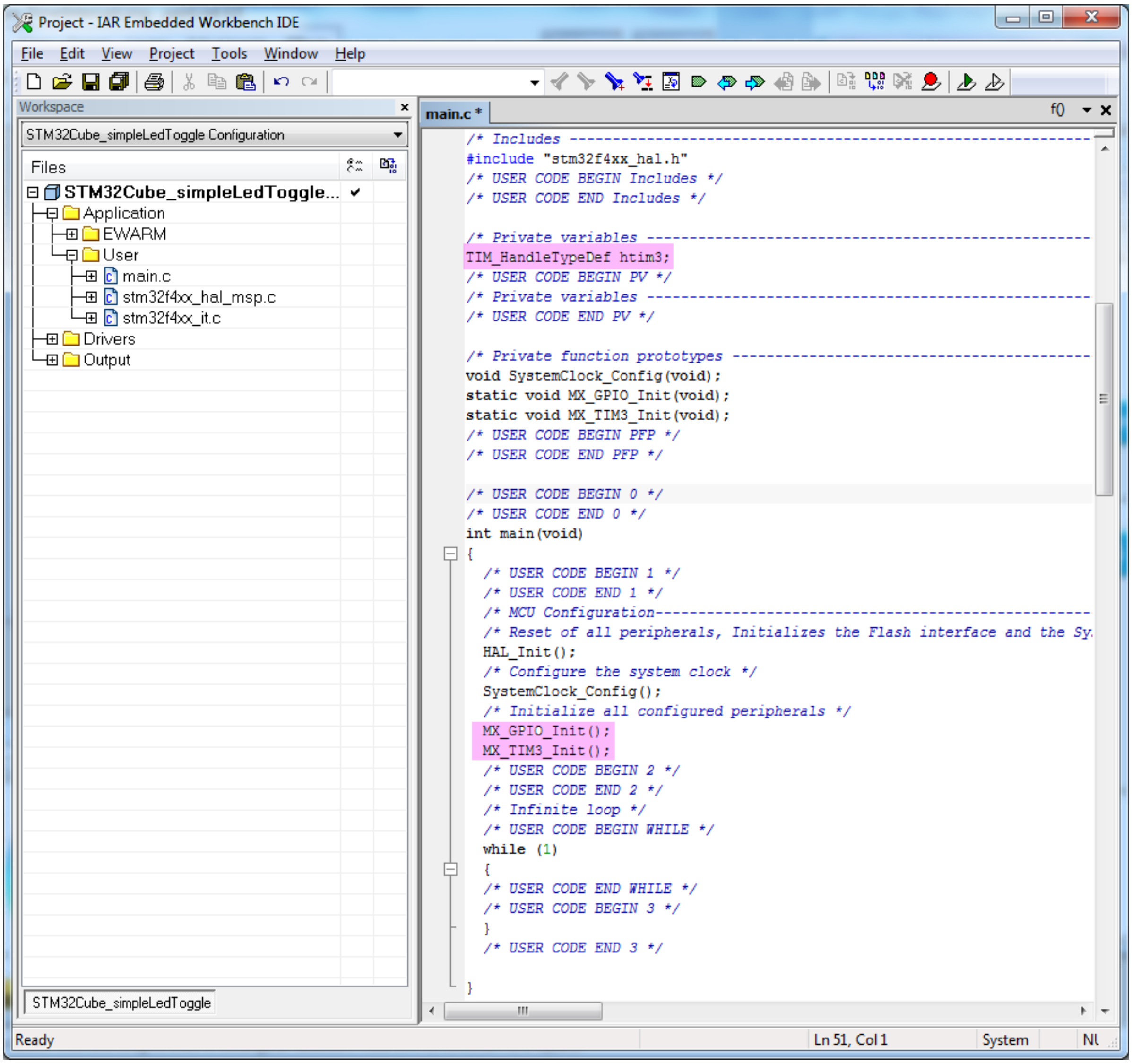
2. As an example, select .eww file to load the project in the IAR™ EWARM IDE.

Figure 573. C code generation for EWARM



3. Select the main.c file to open in editor.

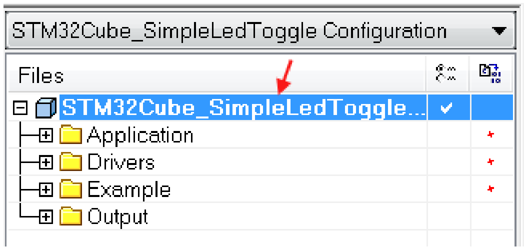
Figure 574. STM32CubeMX generated project open in IAR™ IDE



The htim3 structure handler, system clock, GPIO and TIM3 initialization functions are defined. The initialization functions are called in the main.c. For now the user C code sections are empty.

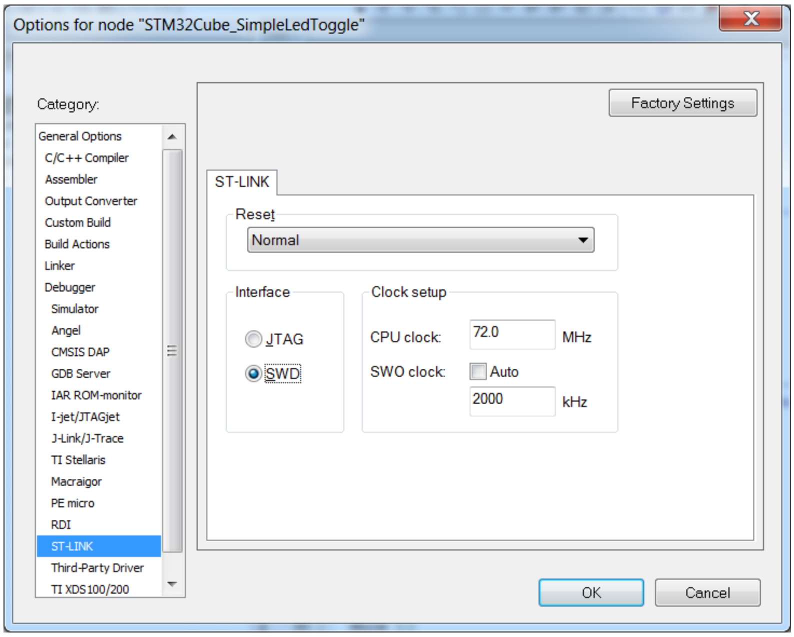
4. In the IAR™ IDE, right-click the project name and select **Options**.

Figure 575. IAR™ options



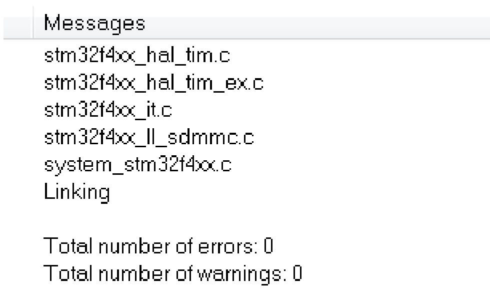
5. Click the ST-LINK category and make sure SWD is selected to communicate with the STM32F4DISCOVERY board. Click **OK**.

Figure 576. SWD connection



6. Select **Project > Rebuild all**. Check if the project building has succeeded.

Figure 577. Project building log



7. Add user C code in the dedicated user sections **only**.

Note: The main while(1) loop is placed in a user section.

For example:

a) Edit the main.c file.

b) To start timer 3, update User Section 2 with the following C code:

Figure 578. User Section 2

```
HAL_Init();
/* Configure the system clock */
SystemClock_Config();
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_TIM3_Init();

/* USER CODE BEGIN 2 */
HAL_TIM_Base_Start_IT(&htim3);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
```

c) Then, add the following C code in User Section 4:

Figure 579. User Section 4

```
/* USER CODE BEGIN 4 */

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if ( htim->Instance == htim3.Instance )
  {
    HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12);
  }
}
/* USER CODE END 4 */
```

This C code implements the weak callback function defined in the HAL timer driver (stm32f4xx_hal_tim.h) to toggle the GPIO pin driving the green LED when the timer counter period has elapsed.

8. Rebuild and program your board using . Make sure the SWD ST-LINK option is checked as a Project options otherwise board programming will fail.
9. Launch the program using . The green LED on the STM32F4DISCOVERY board will blink every second.
10. To change the MCU configuration, go back to STM32CubeMX user interface, implement the changes and regenerate the C code. The project will be updated, preserving the C code in the user sections if Keep User Code when re-generating option in Project Manager's Code Generator tab is enabled.

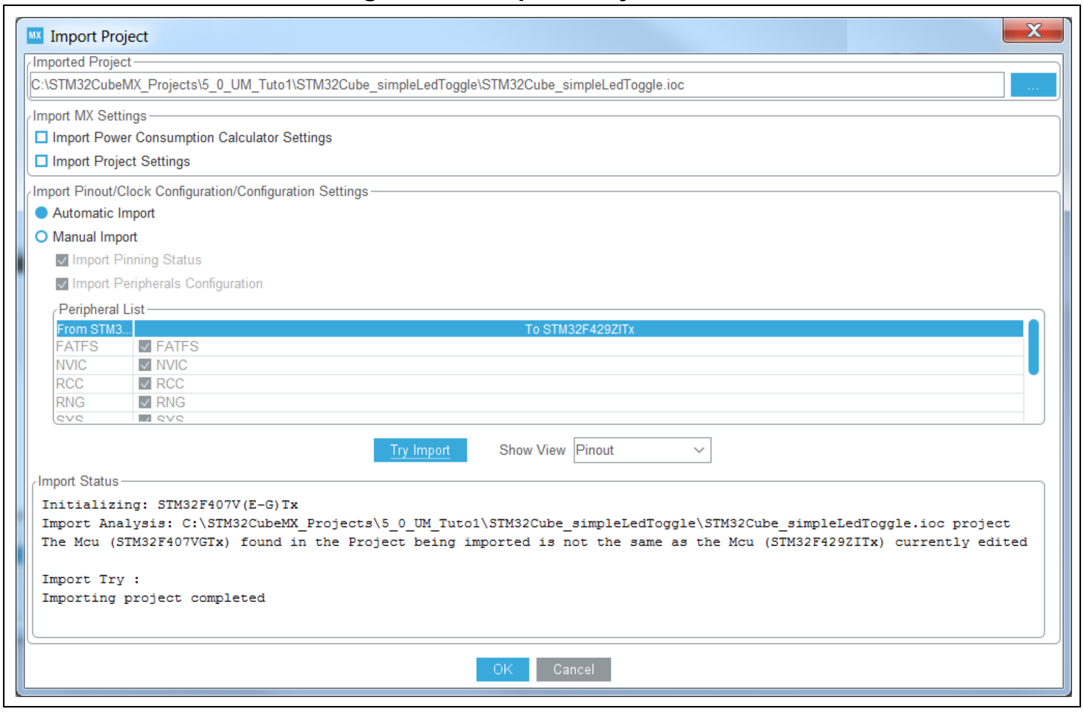
11.9 Switching to another MCU

STM32CubeMX allows loading a project configuration on an MCU of the same series.

Proceed as follows:

1. Select **File > New Project**.
2. Select an MCU belonging to the same series. As an example, you can select the STM32F429ZITx that is the core MCU of the 32F429IDISCOVERY board.
3. Select **File > Import project**. In the **Import project** window, browse to the .ioc file to load. A message warns you that the currently selected MCU (STM32F429ZITx) differs from the one specified in the .ioc file (STM32F407VGTx). Several import options are proposed (see *Figure 580*).
4. Click the **Try Impor**t button and check the import status to verify if the import has been successful.
5. Click **OK** to really import the project. An output tab is then displayed to report the import results.
6. The green LED on 32F429IDISCOVERY board is connected to PG13: CTRL+ right click **PD12** and drag and drop it on PG13.
7. From **Project Manager** project tab configure the new project name and folder location. Click **Generate icon** to save the project and generate the code.
8. Select **Open the project** from the dialog window, update the user sections with the user code, making sure to update the GPIO settings for PG13. Build the project and flash the board. Launch the program and check that LED blinks once per second.

Figure 580. Import Project menu



12 Tutorial 2 - Example of FatFs on an SD card using STM32429I-EVAL evaluation board

The tutorial consists in creating and writing to a file on the STM32429I-EVAL1 SD card using the FatFs file system middleware.

To generate a project and run tutorial 2, follow the sequence below:

1. Launch STM32CubeMX.
2. Select **File > New Project**. The Project window opens.
3. Click the **Board Selector** Tab to display the list of ST boards.
4. Select **EvalBoard** as type of Board and **STM32F4** as Series to filter down the list.
5. Answer Yes to Initialize all peripherals with their default mode so that the code is generated only for the peripherals used by the application.
6. Select the STM32429I-EVAL board and click **OK**. Answer No in the dialog box asking to initialize all peripherals to their default modes (see *Figure 581*). The **Pinout** view is loaded, matching the MCU pinout configuration on the evaluation board (see *Figure 582*).

Figure 581. Board peripheral initialization dialog box

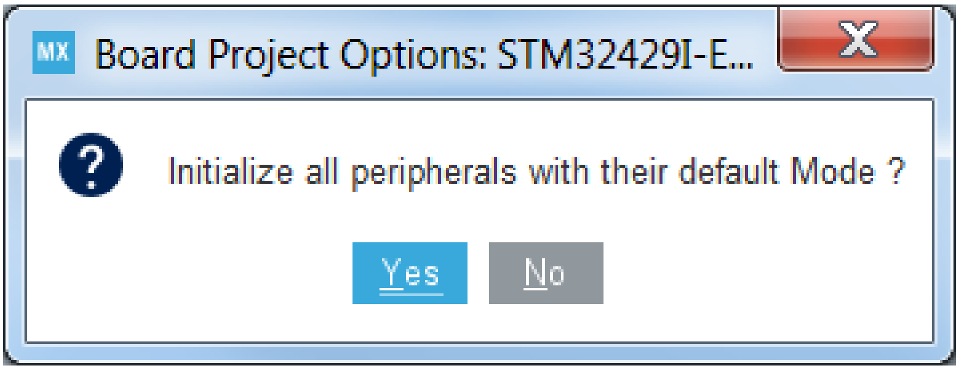
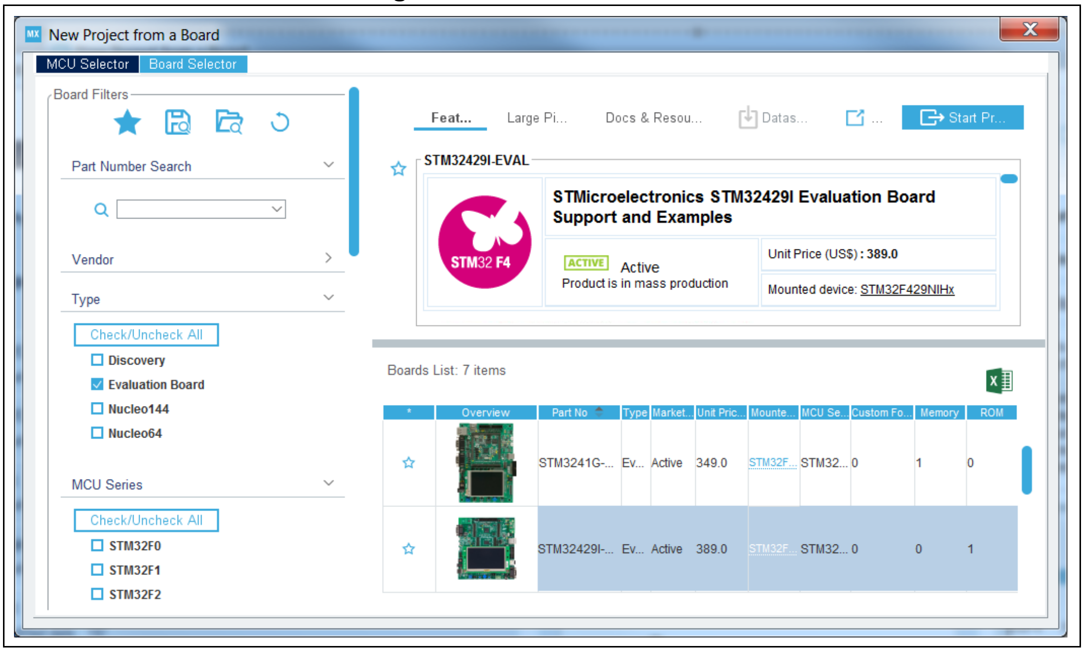
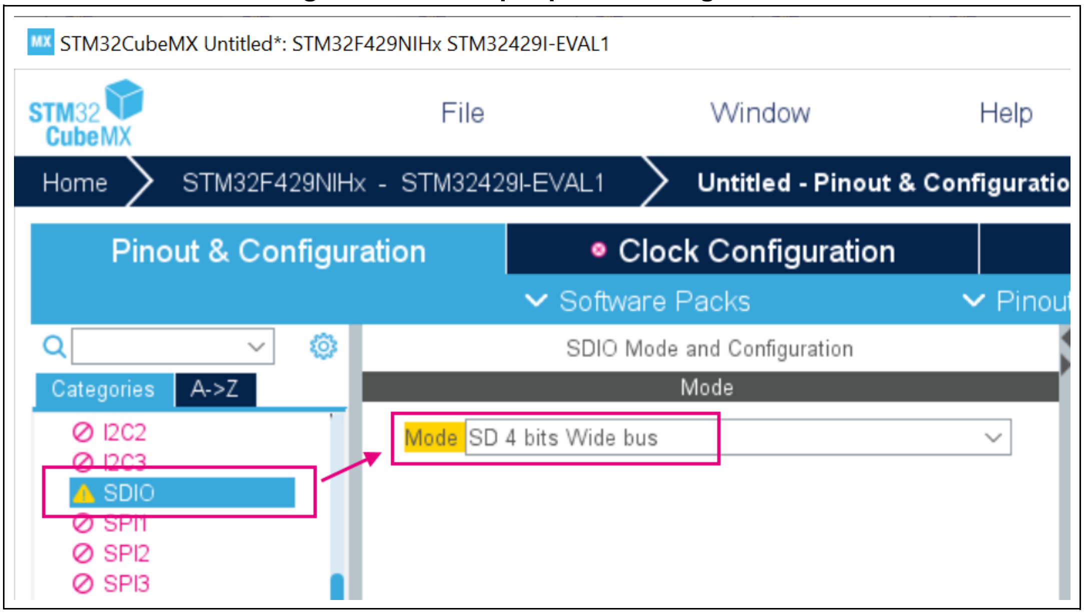


Figure 582. Board selection



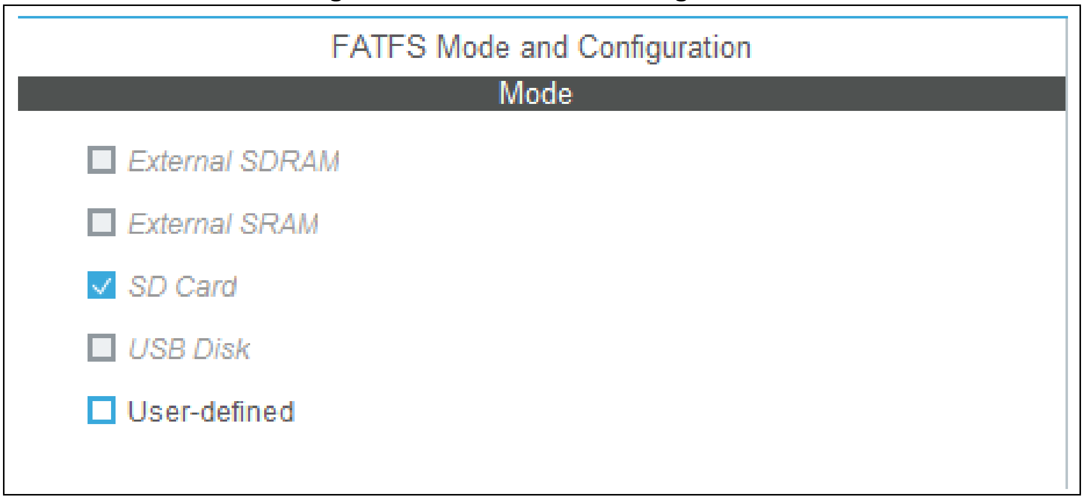
7. From the Peripheral tree on the left, expand the SDIO peripheral and select "SD 4 bits wide bus" (see *Figure 583*). In the configuration panel, from the DMA settings tab, add SDIO_RX and SDIO_TX DMA requests.
8. Finally, go pack to the peripheral tree panel, select NVIC and enable the SDIO global interrupt from the configuration panel.

Figure 583. SDIO peripheral configuration



9. Under the Middlewares category, check SD card as FatFs mode (see *Figure 584*).

Figure 584. FatFs mode configuration



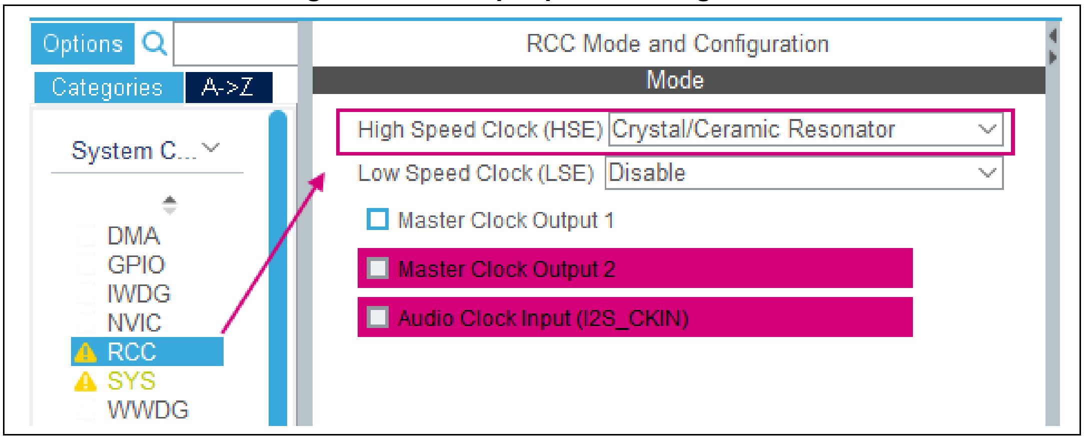
From the Pinout view on the right, enable, as GPIO input, a pin to be used for the SDIO detection.

In the configuration panel below the mode panel, go to the platform settings tab and configure the SD_detection using the pin previously enabled.

Finally, go to FatFs "Advanced settings tab" and enable "Use DMA template".

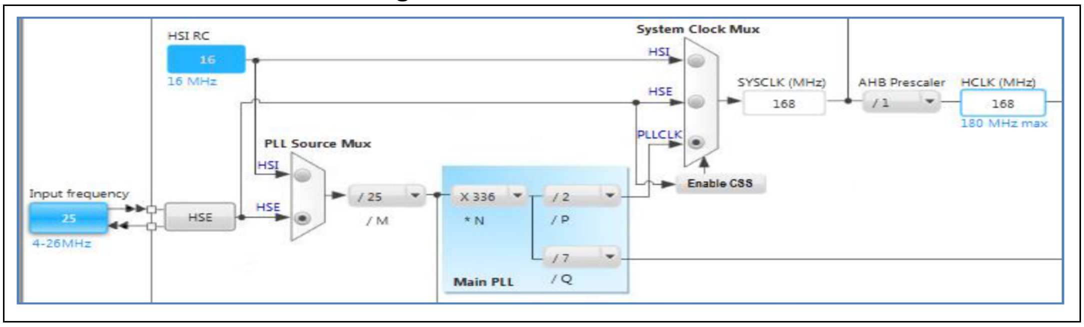
10. Configure the clocks as follows:
   a) Select the RCC peripheral from the **Pinout** view (see *Figure 585*).

Figure 585. RCC peripheral configuration



b) Configure the clock tree from the clock tab (see *Figure 586*).

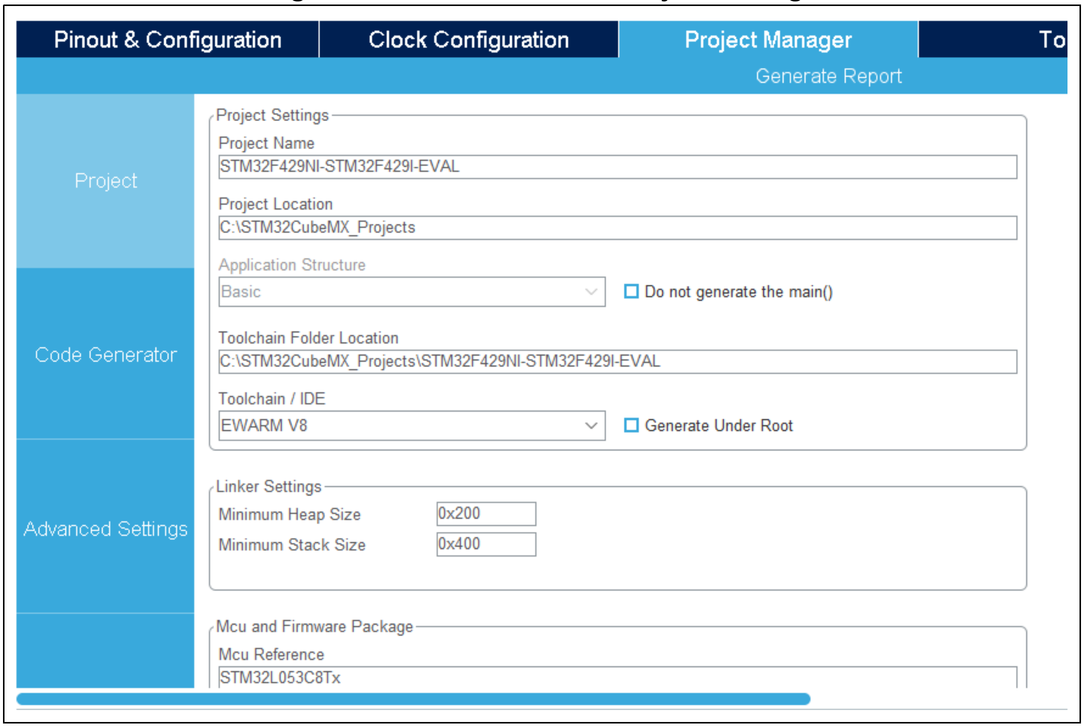
Figure 586. Clock tree view



11. In the **Project** tab, specify the project name and destination folder. Then, select the EWARM IDE toolchain.

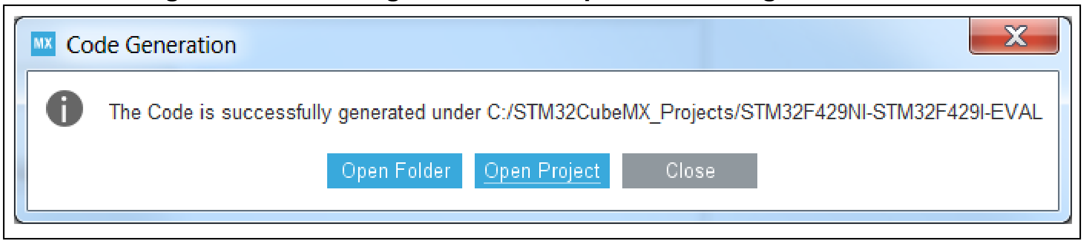
    Note that project heap and stack size can be adjusted to the minimum required for the FATFS application.

Figure 587. FATFS tutorial - Project settings



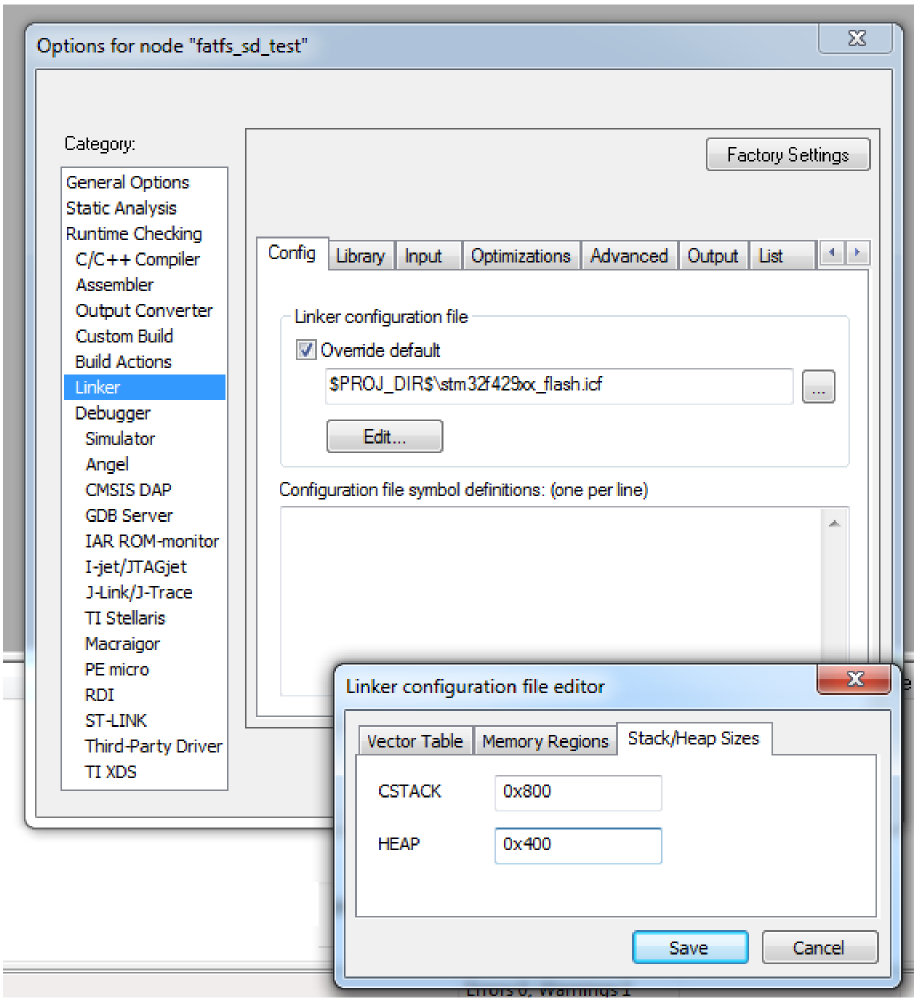
12. Click **Ok**. Then, on the toolbar menu, click GENERATE CODE to generate the project.
13. Upon code generation completion, click **Open Project** in the **Code Generation** dialog window (see *Figure 588*). This opens the project directly in the IDE.

Figure 588. C code generation completion message



14. In the IDE, check that heap and stack sizes are sufficient: right click the project name and select **Options**, then select **Linker**. Check **Override default** to use the icf file from STM32CubeMX generated project folder. if not already done through STM32CubeMX User interface (under Linker Settings from Project Manager's project tab), adjust the heap and stack sizes (see *Figure 589*).

Figure 589. IDE workspace



Note: When using the MDK-Arm toolchain, go to the Application/MDK-ARM folder and double-click the startup_xx.s file to edit and adjust the heap and stack sizes there.

15. Go to the Application/User folder. Double-click the main.c file and edit it.
16. The tutorial consists in creating and writing to a file on the evaluation board SD card using the FatFs file system middleware:
    a) At startup all LEDs are OFF.
    b) The red LED is turned ON to indicate that an error occurred (e.g. FatFs initialization, file read/write access errors).
    c) The orange LED is turned ON to indicate that the FatFs link has been successfully mounted on the SD driver.
    d) The blue LED is turned ON to indicate that the file has been successfully written to the SD card.
    e) The green LED is turned ON to indicate that the file has been successfully read from file the SD card.
17. For use case implementation, update main.c with the following code:
    a) Insert main.c private variables in a dedicated user code section:

```
/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/
FATFS SDFatFs;  /* File system object for SD card logical drive */
FIL MyFile;     /* File object */
const char wtext[] = "Hello World!";
static uint8_t buffer[_MAX_SS]; /* a work buffer for the f_mkfs() */
/* USER CODE END PV */
```

    b) Insert main functional local variables:

```
int main(void)
{

  /* USER CODE BEGIN 1 */
  FRESULT res;                    /* FatFs function common result code */
  uint32_t byteswritten, bytesread; /* File write/read counts */
  char rtext[256];                  /* File read buffer */
  /* USER CODE END 1 */

  /* MCU Configuration----------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
HAL_Init();
```

    c) Insert user code in the main function, after initialization calls and before the while loop, to perform actual read/write from/to the SD card:

```
int main(void)
{
      ….
      MX_FATFS_Init();

        /* USER CODE BEGIN 2 */
      /*##-0- Turn all LEDs off(red, green, orange and blue) */
         HAL_GPIO_WritePin(GPIOG, (GPIO_PIN_10 | GPIO_PIN_6 | GPIO_PIN_7 |
      GPIO_PIN_12), GPIO_PIN_SET);
      /*##-1- FatFS: Link the SD disk I/O driver ##########*/
```

```
    if(retSD == 0){
      /* success: set the orange LED on */
     HAL_GPIO_WritePin(GPIOG, GPIO_PIN_7, GPIO_PIN_RESET);
/*##-2- Register the file system object to the FatFs module ###*/
    if(f_mount(&SDFatFs, (TCHAR const*)SDPath, 0) != FR_OK){
     /* FatFs Initialization Error : set the red LED on */
       HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
       while(1);
     }  else   {
/*##-3- Create a FAT file system (format) on the logical drive#*/
 /* WARNING: Formatting the uSD card will delete all content on the
device */
if(f_mkfs((TCHAR const*)SDPath, FM_ANY, 0, buffer, sizeof(buffer))
!= FR_OK){
    /* FatFs Format Error : set the red LED on */
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
    while(1);
  } else {
/*##-4- Create & Open a new text file object with write access#*/
  if(f_open(&MyFile, "Hello.txt", FA_CREATE_ALWAYS | FA_WRITE) !=
FR_OK){
  /* 'Hello.txt' file Open for write Error : set the red LED on */
  HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
  while(1);
      } else {
 /*##-5- Write data to the text file ####################*/
   res = f_write(&MyFile, wtext, sizeof(wtext), (void
*)&byteswritten);
  if((byteswritten == 0) || (res != FR_OK)){
    /* 'Hello.txt' file Write or EOF Error : set the red LED on */
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
    while(1);
    } else {
 /*##-6- Successful open/write : set the blue LED on */
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_12, GPIO_PIN_RESET);
    f_close(&MyFile);
 /*##-7- Open the text file object with read access #*/
    if(f_open(&MyFile, "Hello.txt", FA_READ) != FR_OK){
   /* 'Hello.txt' file Open for read Error : set the red LED on */
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
    while(1);
    } else {
 /*##-8- Read data from the text file #########*/
    res = f_read(&MyFile, rtext, sizeof(wtext), &bytesread);
    if((byteswritten == 0)|| (res != FR_OK)){
  /* 'Hello.txt' file Read or EOF Error : set the red LED on */
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
    while(1);
    } else {
  /* Successful read : set the green LED On */
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_6, GPIO_PIN_RESET);
```

```
      /*##-9- Close the open text file ################*/
         f_close(&MyFile);
       }}}}}}}
      /*##-10- Unlink the micro SD disk I/O driver #########*/
         FATFS_UnLinkDriver(SDPath);

       /* USER CODE END 2 */

       /* Infinite loop */
       /* USER CODE BEGIN WHILE */
while (1)
```

13 Tutorial 3 - Using the Power Consumption Calculator to optimize the embedded application consumption and more

13.1 Tutorial overview

This tutorial focuses on STM32CubeMX Power Consumption Calculator (Power Consumption Calculator) feature and its benefits to evaluate the impacts of power-saving techniques on a given application sequence.

The key considerations to reduce a given application power consumption are:

- Reducing the operating voltage
- Reducing the time spent in energy consuming modes

  It is up to the developer to select a configuration that gives the best compromise between low-power consumption and performance.
- Maximizing the time spent in non-active and low-power modes
- Using the optimal clock configuration

  The core should always operate at relatively good speed, since reducing the operating frequency can increase energy consumption if the microcontroller has to remain for a long time in an active operating mode to perform a given operation.
- Enabling only the peripherals relevant for the current application state and clock-gating the others
- When relevant, using the peripherals with low-power features (e.g. waking up the microcontroller with the I2C)
- Minimizing the number of state transitions
- Optimizing memory accesses during code execution
  - Prefer code execution from RAM to flash memory
  - When relevant, consider aligning CPU frequency with flash memory operating frequency for zero wait states.

The following tutorial shows how the STM32CubeMX Power Consumption Calculator feature can help to tune an application to minimize its power consumption and extend the battery life.

Note: The Power Consumption Calculator does not account for I/O dynamic current consumption and external board components that can also affect current consumption. For this purpose, an “additional consumption” field is provided for the user to specify such consumption value.

13.2 Application example description

The application is designed using the NUCLEO-L476RG board, based on an STM32L476RGTx device, and supplied by a 2.4 V battery.

The main purpose of this application is to perform ADC measurements and transfer the conversion results over UART. It uses:

- Multiple low-power modes: Low-power run, Low-power sleep, Sleep, Stop and Standby
- Multiple peripherals: USART, DMA, Timer, COMP, DAC and RTC
  - The RTC is used to run a calendar and to wake up the CPU from Standby when a specified time has elapsed.
  - The DMA transfers ADC measurements from ADC to memory
  - The USART is used in conjunction with the DMA to send/receive data via the virtual COM port and to wake up the CPU from Stop mode.

The process to optimize such complex application is to start describing first a functional only sequence then to introduce, on a step by step basis, the low-power features provided by the STM32L476RG microcontroller.

13.3 Using the Power Consumption Calculator

13.3.1 Creating a power sequence

Follow the steps below to create the sequence (see *Figure 590*):

1. Launch STM32CubeMX.
2. Click **new project** and select the Nucleo-L476RG board from the **Board** tab.
3. Click the **Power Consumption Calculator** tab to select the Power Consumption Calculator view. A first sequence is then created as a reference.
4. Adapt it to minimize the overall current consumption. To do this:
   a) Select 2.4 V V_{DD} power supply. This value can be adjusted on a step by step basis (see *Figure 591*).
   b) Select the Li-MnO2 (CR2032) battery. This step is optional. The battery type can be changed later on (see *Figure 591*).

Figure 590. Power Consumption Calculation example

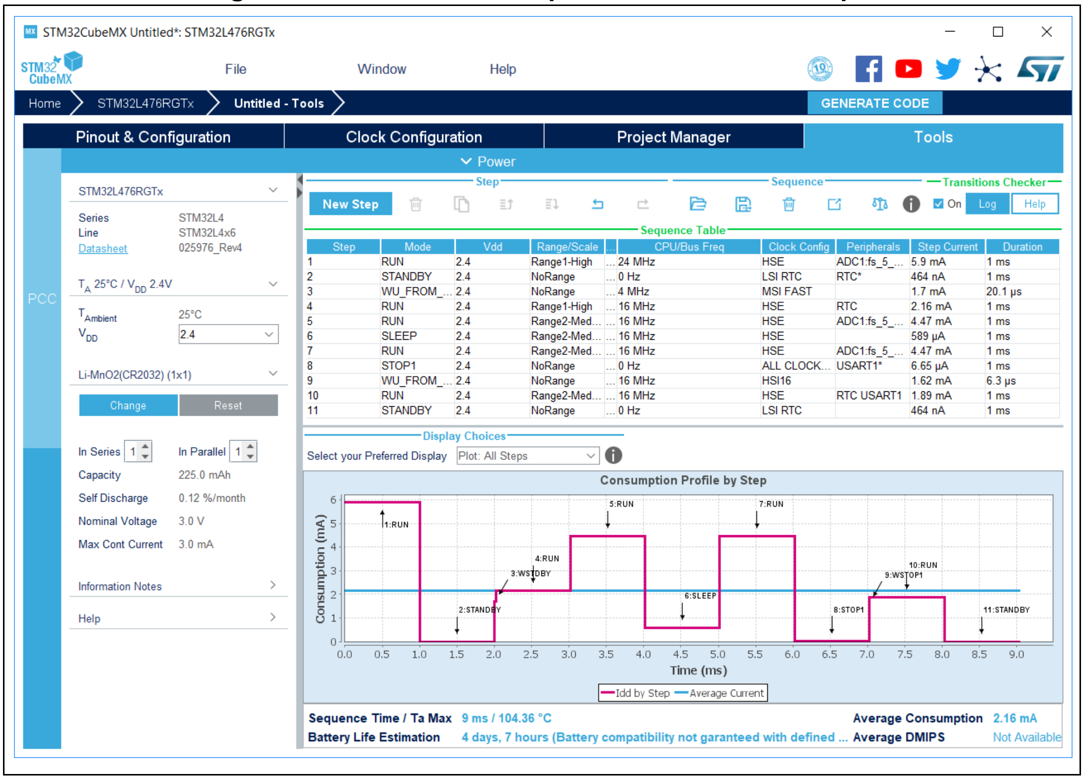
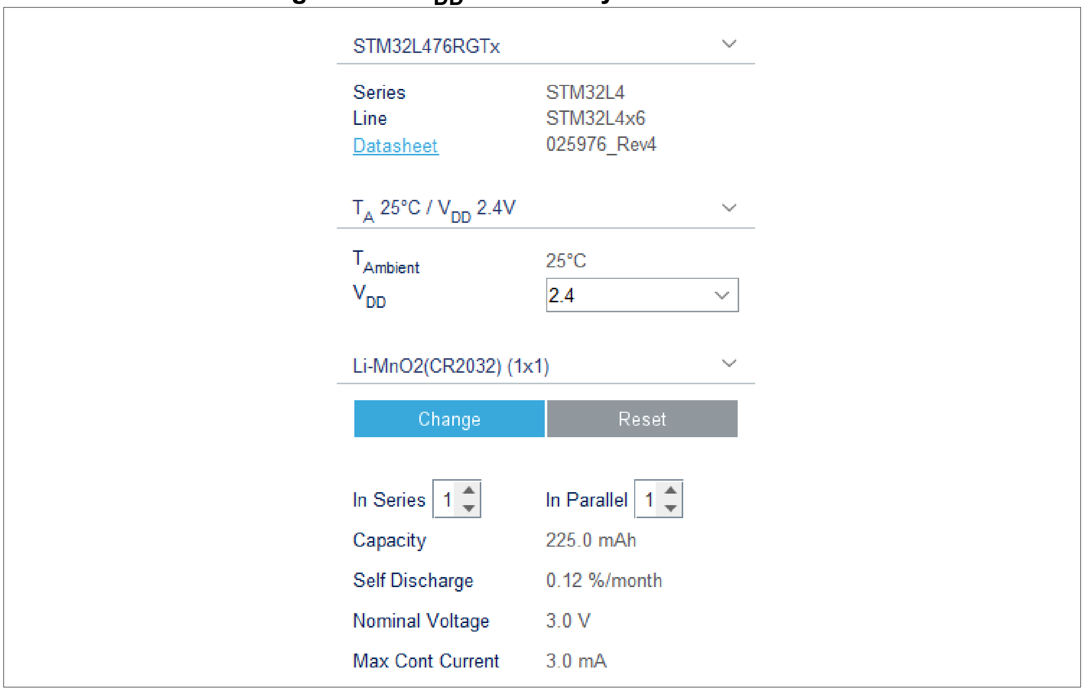


Figure 591. V_{DD} and battery selection menu



5. Enable the **Transition checker** to ensure the sequence is valid (see *Figure 591*). This option allows verifying that the sequence respects the allowed transitions implemented within the STM32L476RG.
6. Click the **Add** button to add steps that match the sequence described in *Figure 591*.
   - By default the steps last 1 ms each, except for the wake-up transitions preset using the transition times specified in the product datasheet (see *Figure 592*).
   - Some peripherals for which consumption is unavailable or negligible are highlighted with * (see *Figure 592*).

Figure 592. Sequence table

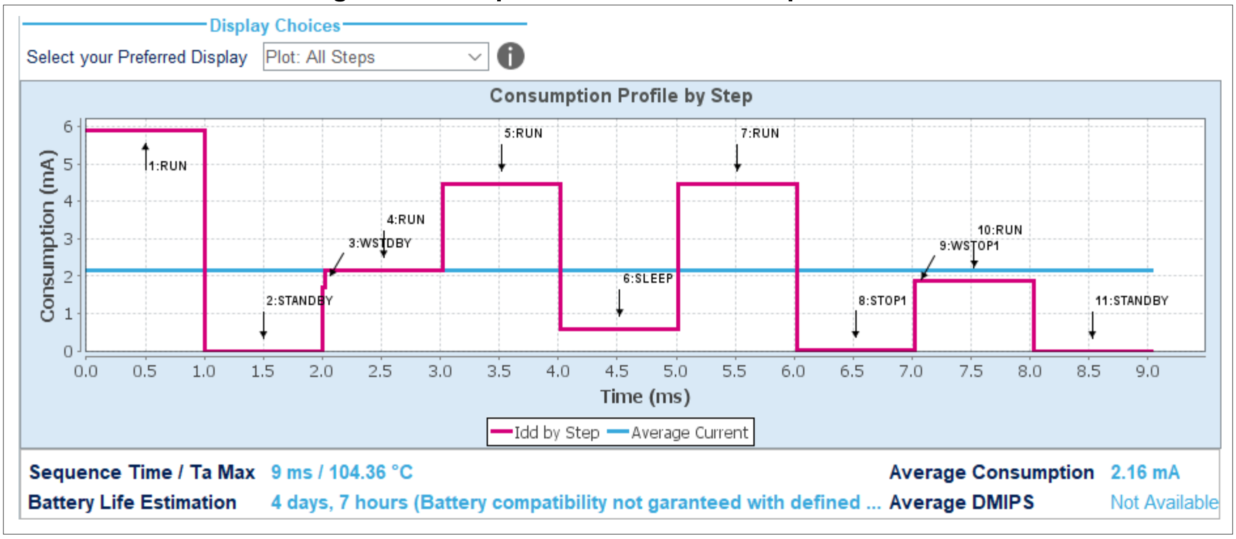
Sequence Table

Step	Mode	Vdd	Range/Scale	...	CPU/Bus Freq	Clock Config	Peripherals	Step Current	Duration
1	RUN	2.4	Range1-High	...	24 MHz	HSE	ADC1:fs_5_...	5.9 mA	1 ms
2	STANDBY	2.4	NoRange	...	0 Hz	LSI RTC	RTC*	464 nA	1 ms
3	WU_FROM_...	2.4	NoRange	...	4 MHz	MSI FAST		1.7 mA	20.1 µs
4	RUN	2.4	Range1-High	...	16 MHz	HSE	RTC	2.16 mA	1 ms
5	RUN	2.4	Range2-Med...	...	16 MHz	HSE	ADC1:fs_5_...	4.47 mA	1 ms
6	SLEEP	2.4	Range2-Med...	...	16 MHz	HSE		589 µA	1 ms
7	RUN	2.4	Range2-Med...	...	16 MHz	HSE	ADC1:fs_5_...	4.47 mA	1 ms
8	STOP1	2.4	NoRange	...	0 Hz	ALL CLOCK...	USART1*	6.65 µA	1 ms
9	WU_FROM_...	2.4	NoRange	...	16 MHz	HSI16		1.62 mA	6.3 µs
10	RUN	2.4	Range2-Med...	...	16 MHz	HSE	RTC USART1	1.89 mA	1 ms
11	STANDBY	2.4	NoRange	...	0 Hz	LSI RTC		464 nA	1 ms

7. Click the **Save** button to save the sequence as SequenceOne.

The application consumption profile is generated. It shows that the overall sequence consumes an average of 2.01 mA for 9 ms, and that the battery lifetime is only four days (see *Figure 593*).

Figure 593. sequence results before optimization



13.3.2 Optimizing application power consumption

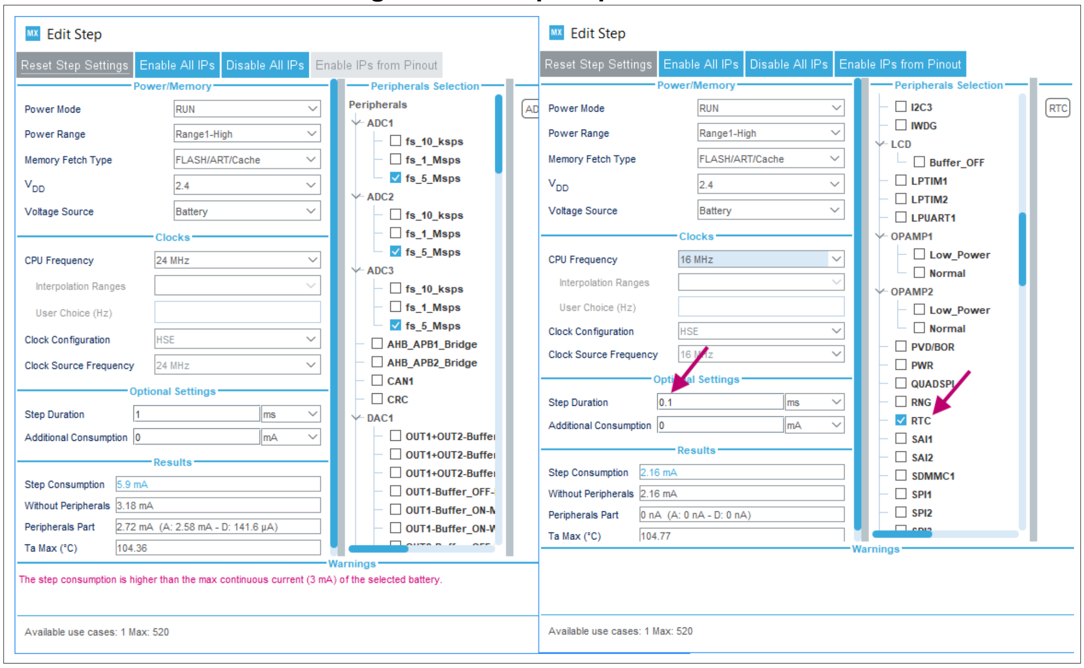
Let us now take actions to optimize the overall consumption and the battery lifetime. These actions are performed on steps 1, 4, 5, 6, 7, 8, and 10.

The next figures show on the left the original step, and on the right the step updated with optimization actions.

Step 1 (Run)

- Findings
  All peripherals are enabled although the application requires only the RTC.
- Actions
  – Lower the operating frequency
  – Enable only the RTC peripheral
  – To reduce the average current consumption, reduce the time spent in this mode
- Result
  The current is reduced from 9.05 to 2.16 mA (see *Figure 594*).

Figure 594. Step 1 optimization



Step 4 (Run, RTC)

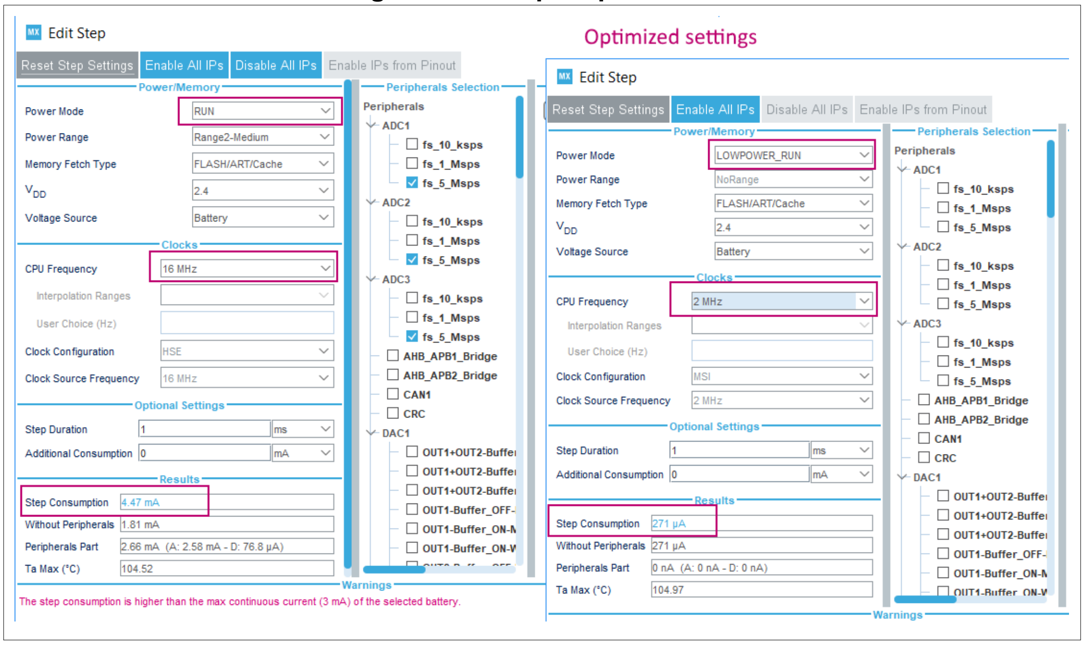
- Action
  Reduce the time spent in this mode to 0.1 ms

Step 5 (Run, ADC, DMA, RTC)

- Actions
  - Change to Low-power run mode
  - Lower the operating frequency
- Results

  The current consumption is reduced from 6.17 mA to 271 µA (see *Figure 595*).

Figure 595. Step 5 optimization

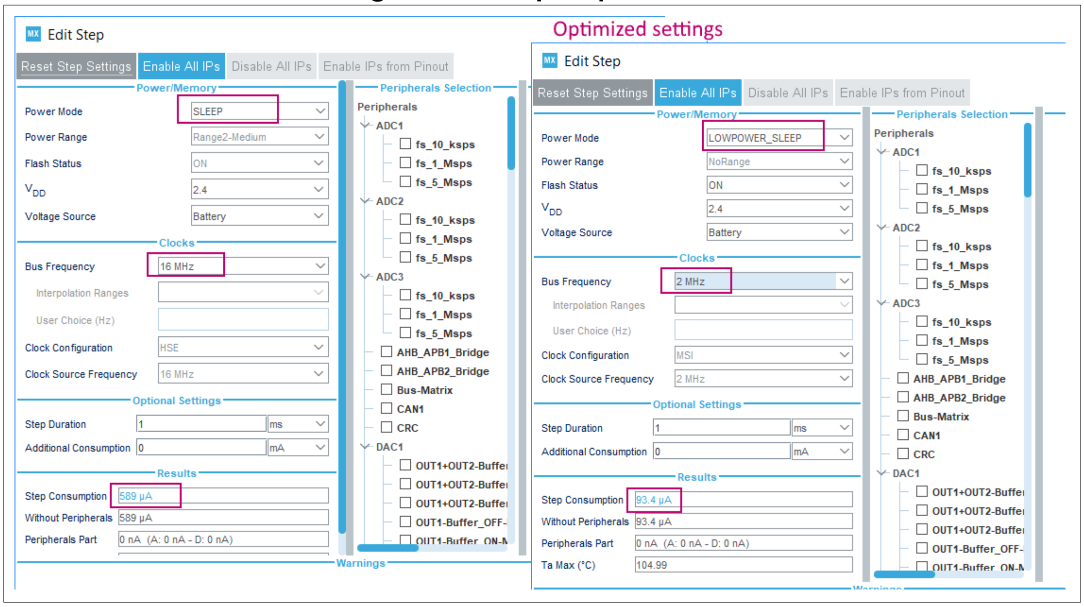


Step 6 (Sleep, DMA, ADC, RTC)

- Actions
  - Switch to Lower-power sleep mode (BAM mode)
  - Reduce the operating frequency to 2 MHz
- Results

  The current consumption is reduced from 703 µA to 93 µA (see *Figure 596*).

Figure 596. Step 6 optimization

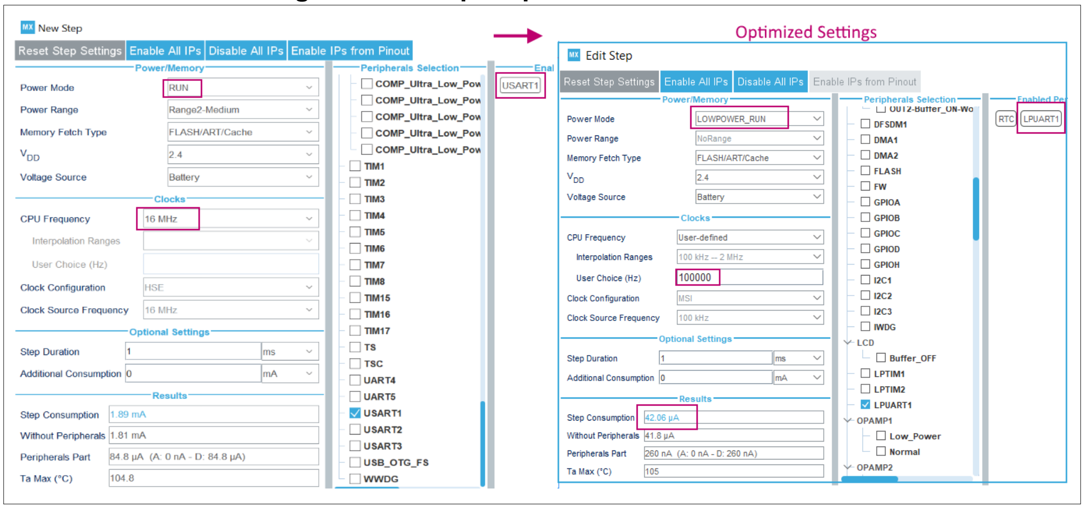


Step 7 (Run, DMA, RTC, USART)

- Actions
  - Switch to Low-power run mode
  - Use the power efficient LPUART peripheral
  - Reduce the operating frequency to 1 MHz using the interpolation feature
- Results

  The current consumption is reduced from 1.92 mA to 42 µA (see *Figure 597*).

Figure 597. Step 7 optimization

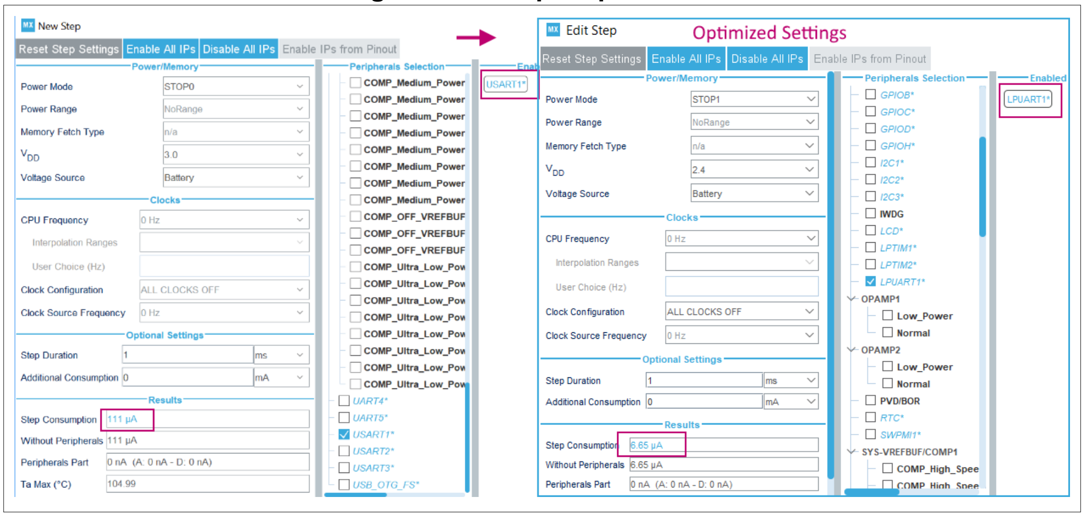


Step 8 (Stop 0, USART)

- Actions
  - Switch to Stop1 low-power mode
  - Use the power-efficient LPUART peripheral
- Results

  The current consumption is reduced (see *Figure 598*).

Figure 598. Step 8 optimization



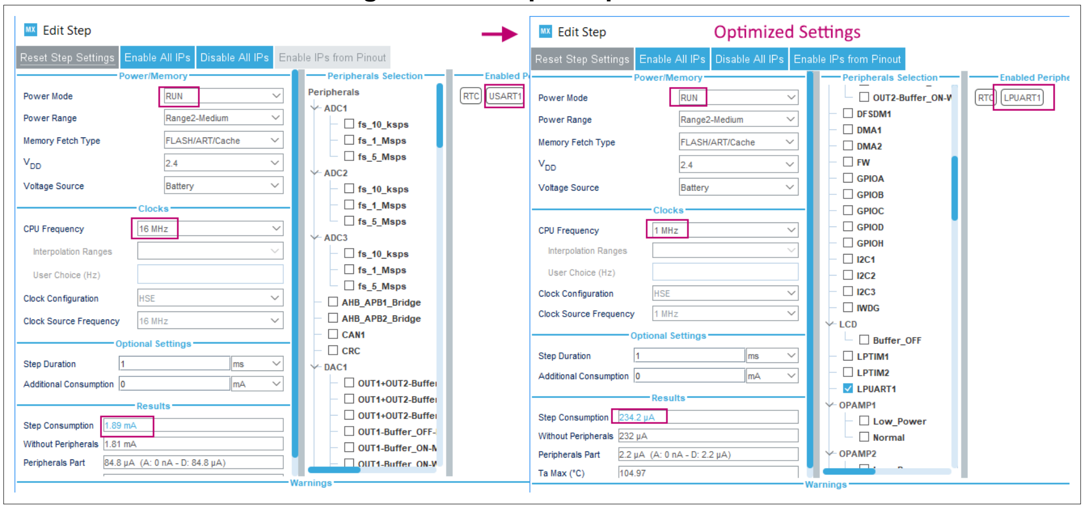
Step 10 (RTC, USART)

- Actions
  - Use the power-efficient LPUART peripheral
  - Reduce the operating frequency to 1 MHz
- Results

  The current consumption is reduced from 1.89 mA to 234 µA (see *Figure 599*).

  The example given in *Figure 600* shows an average current consumption reduction of 155 µA.

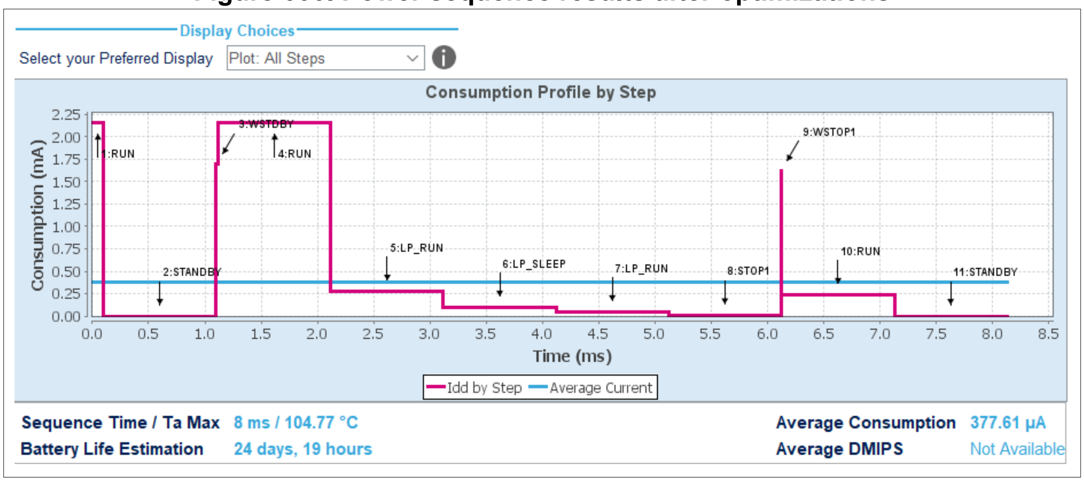
Figure 599. Step 10 optimization



See *Figure 600* for the overall results: 7 ms duration, about two months battery life, and an average current consumption of 165.25 µA.

Use the **compare** button to compare the current results to the original ones saved as SequenceOne.pcs.

Figure 600. Power sequence results after optimizations



14 Tutorial 4 - Example of UART communications with an STM32L053xx Nucleo board

This tutorial aims at demonstrating how to use STM32CubeMX to create a UART serial communication application for a NUCLEO-L053R8 board.

A Windows PC is required for the example. The ST-Link USB connector is used both for serial data communications, and firmware downloading and debugging on the MCU. A Type-A to mini-B USB cable must be connected between the board and the computer. The USART2 peripheral uses PA2 and PA3 pins, which are wired to the ST-Link connector. In addition, USART2 is selected to communicate with the PC via the ST-Link Virtual COM Port. A serial communication client, such as Tera Term, needs to be installed on the PC to display the messages received from the board over the virtual communication Port.

14.1 Tutorial overview

Tutorial 4 will take you through the following steps:

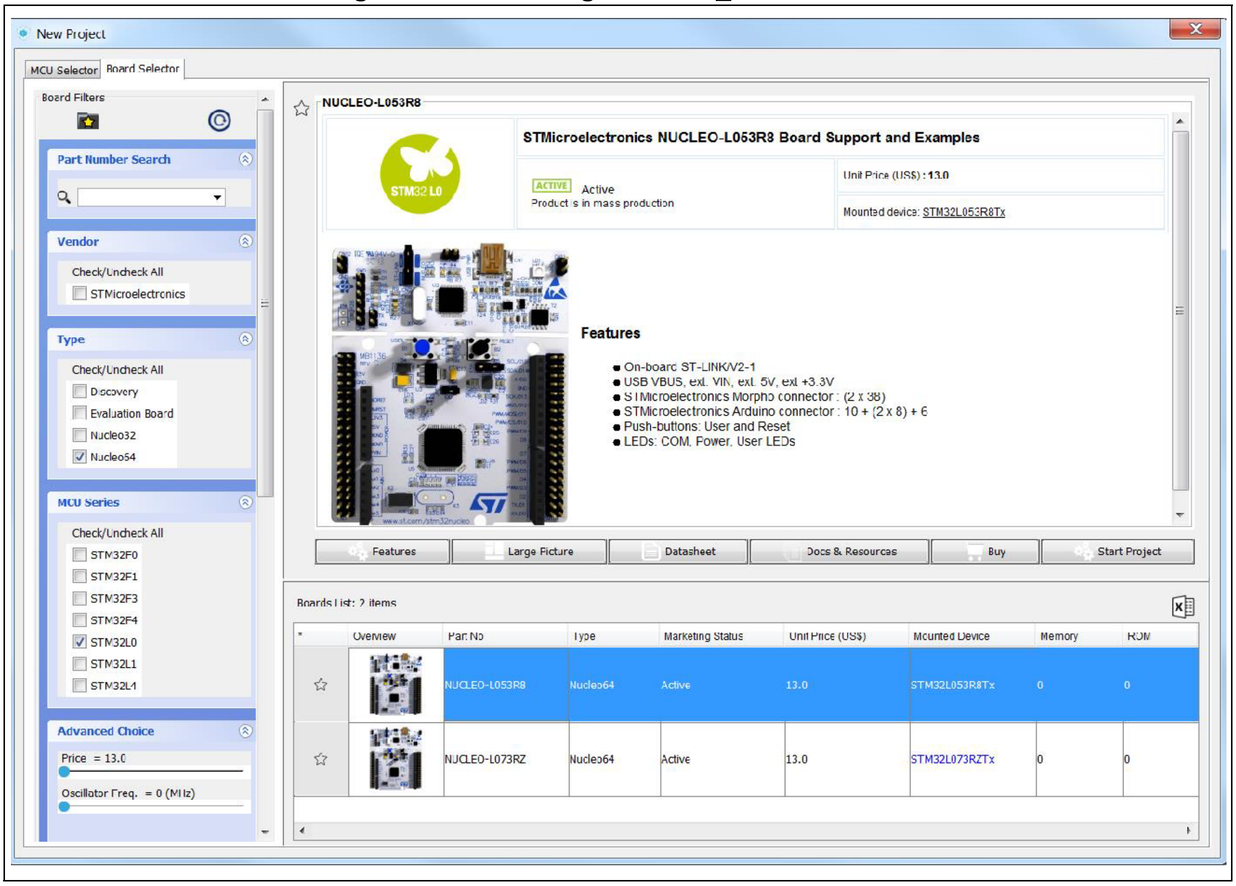
1. Selection of the NUCLEO-L053R8 board from the **New Project** menu.
2. Selection of the required features (debug, USART, timer) from the **Pinout** view: peripheral operating modes as well as assignment of relevant signals on pins.
3. Configuration of the MCU clock tree from the **Clock Configuration** view.
4. Configuration of the peripheral parameters from the **Configuration** view
5. Configuration of the project settings in the **Project Manager** menu and generation of the project (initialization code only).
6. Project update with the user application code corresponding to the UART communication example.
7. Compilation, and execution of the project on the board.
8. Configuration of Tera Term software as serial communication client on the PC.
9. The results are displayed on the PC.

14.2 Creating a new STM32CubeMX project and selecting the Nucleo board

To do this, follow the sequence below:

1. Select **File > New project** from the main menu bar. This opens the **New Project** window.
2. Go to the **Board selector** tab and filter on STM32L0 series.
3. Select NUCLEO-L053R8 and click **OK** to load the board within the STM32CubeMX user interface (see *Figure 601*).

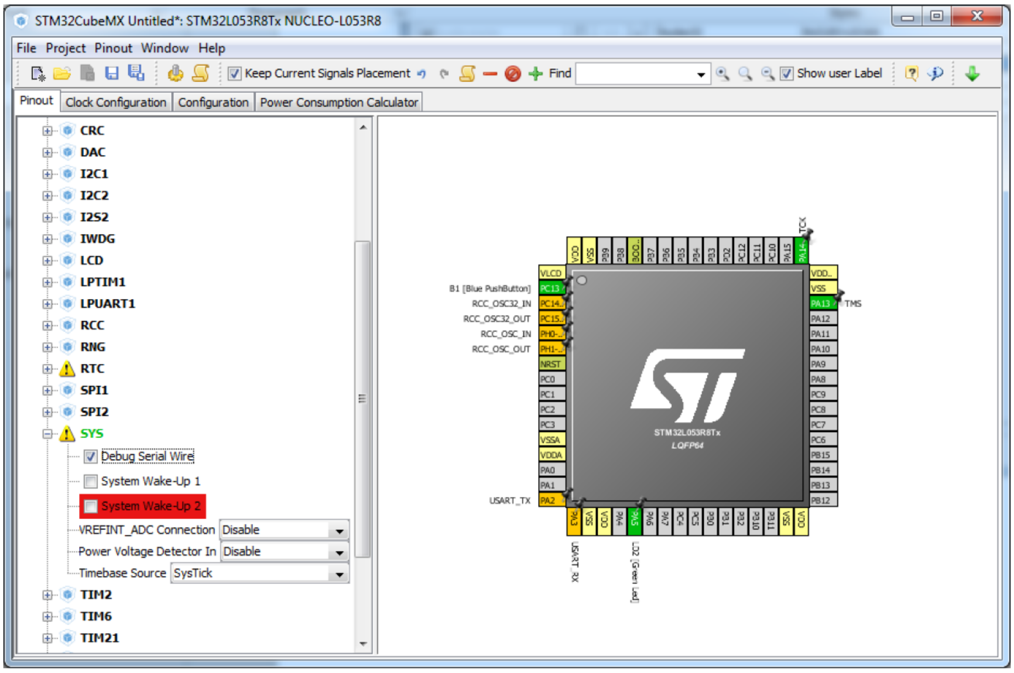
Figure 601. Selecting NUCLEO_L053R8 board



14.3 Selecting the features from the Pinout view

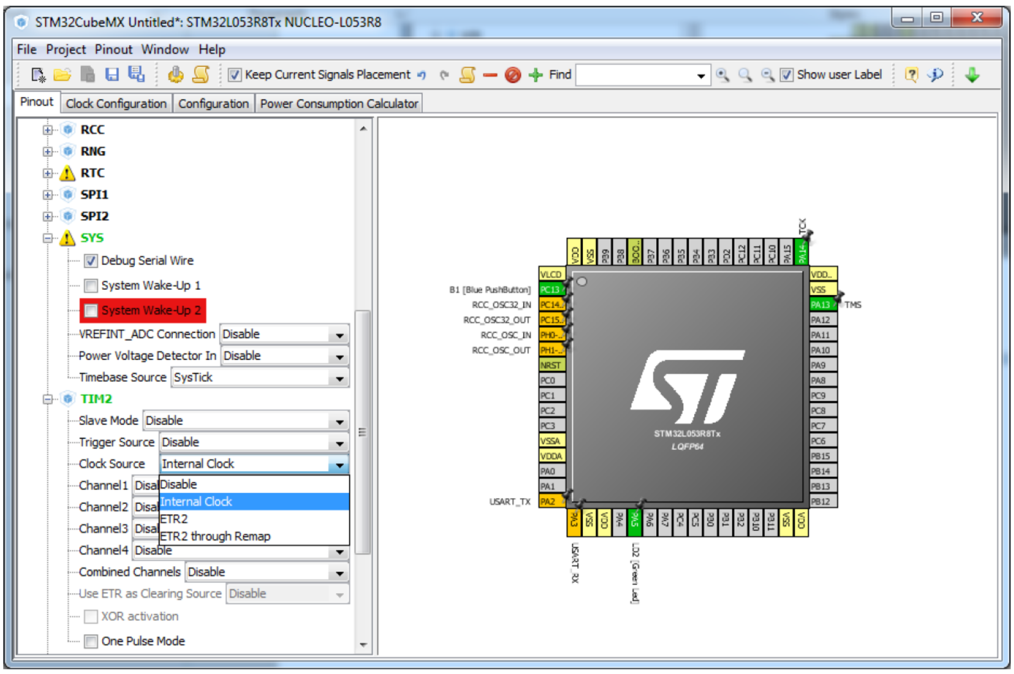
1. Select Debug Serial Wire under SYS (see *Figure 602*).

Figure 602. Selecting debug pins



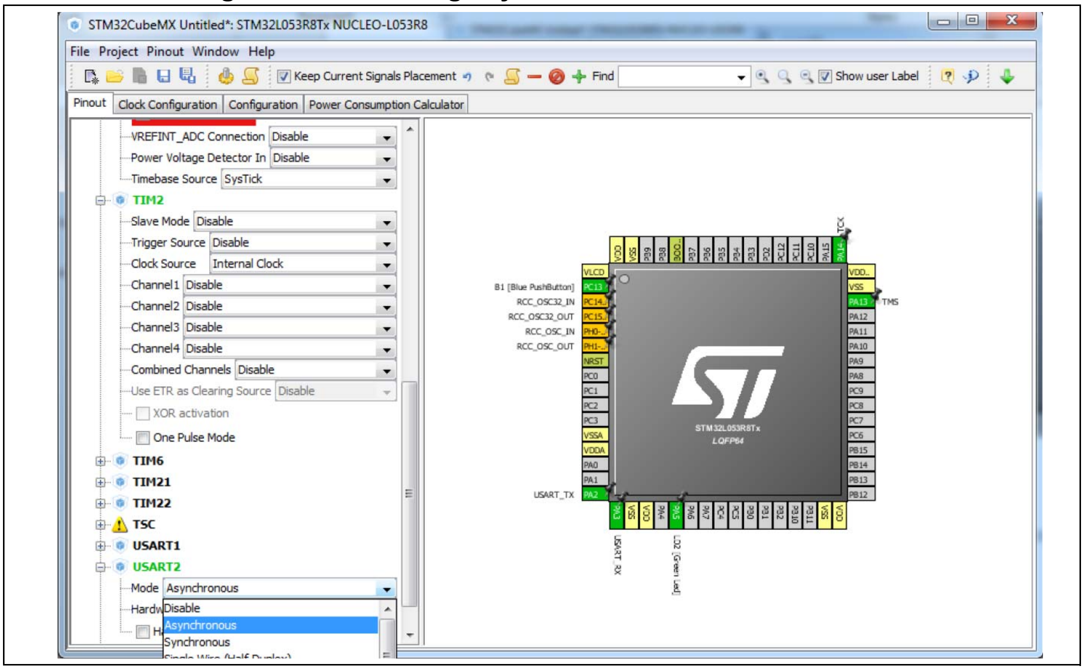
2. Select Internal Clock as clock source under TIM2 peripheral (see *Figure 603*).

Figure 603. Selecting TIM2 clock source



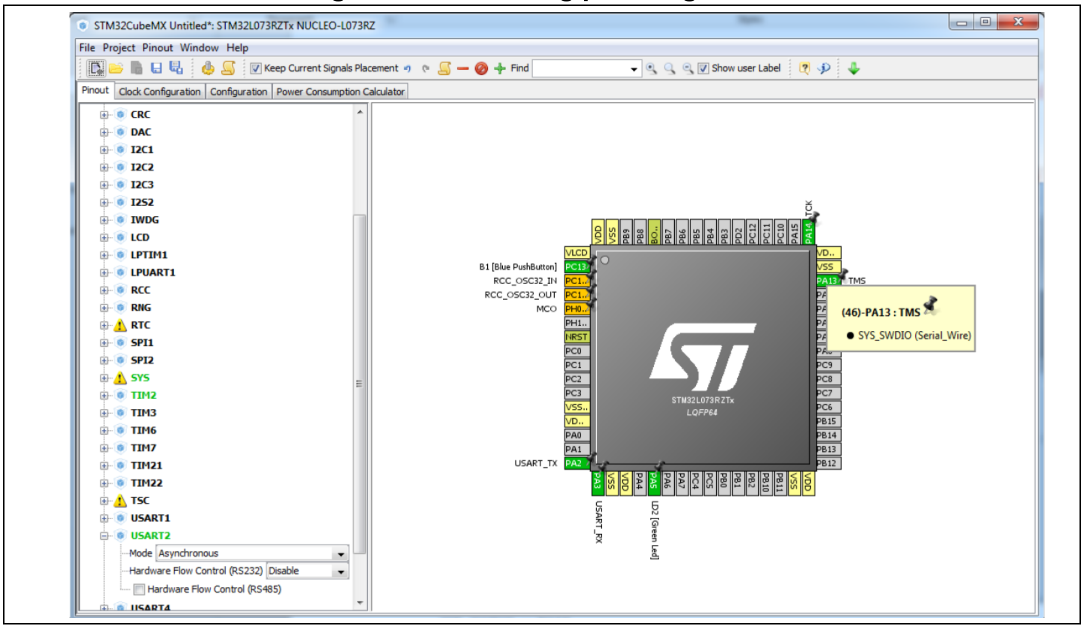
3. Select the Asynchronous mode for the USART2 peripheral (see *Figure 604*).

Figure 604. Selecting asynchronous mode for USART2



4. Check that the signals are properly assigned on pins (see *Figure 605*):
   – SYS_SWDIO on PA13
   – TCK on PA14
   – USART_TX on PA2
   – USART_RX on PA3

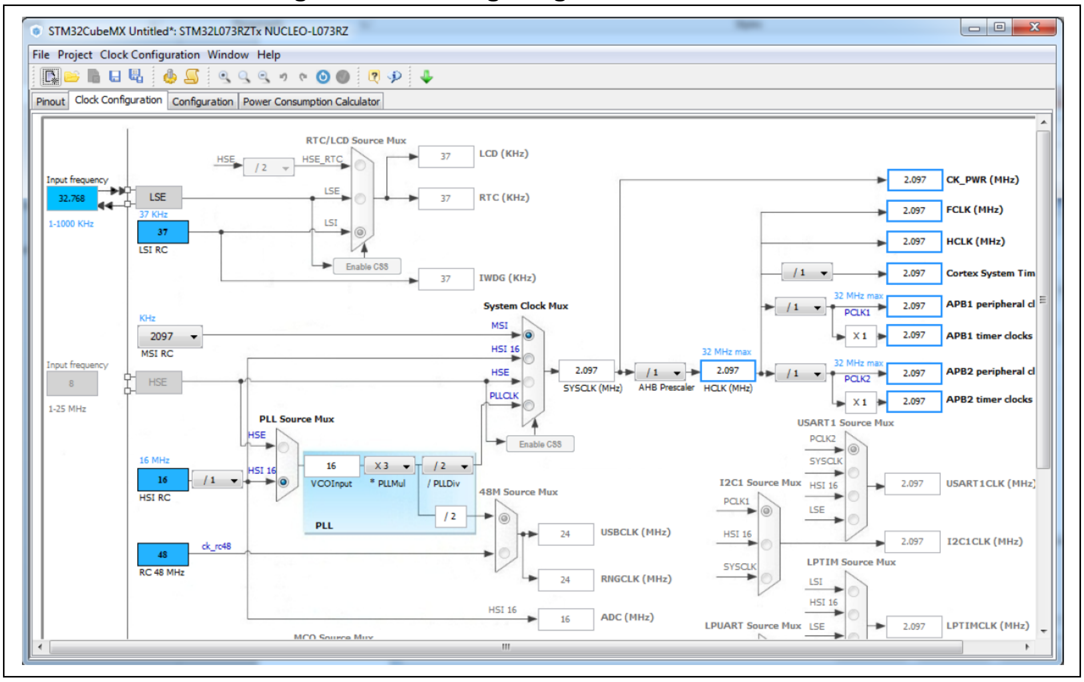
Figure 605. Checking pin assignment



14.4 Configuring the MCU clock tree from the Clock Configuration view

1. Go to the **Clock Configuration** tab and leave the configuration untouched, in order to use the MSI as input clock and an HCLK of 2.097 MHz (see *Figure 606*).

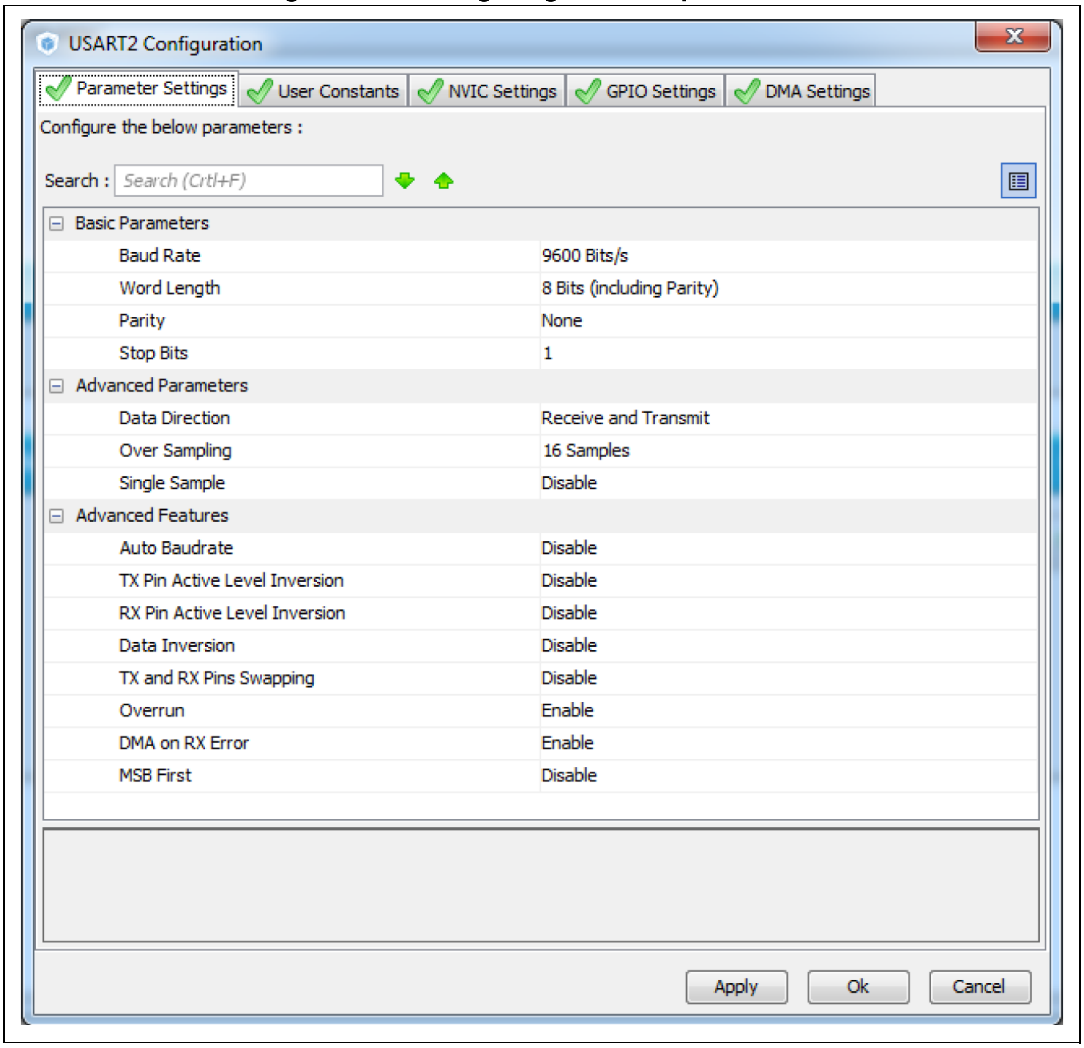
Figure 606. Configuring the MCU clock tree



14.5 Configuring the peripheral parameters from the Configuration view

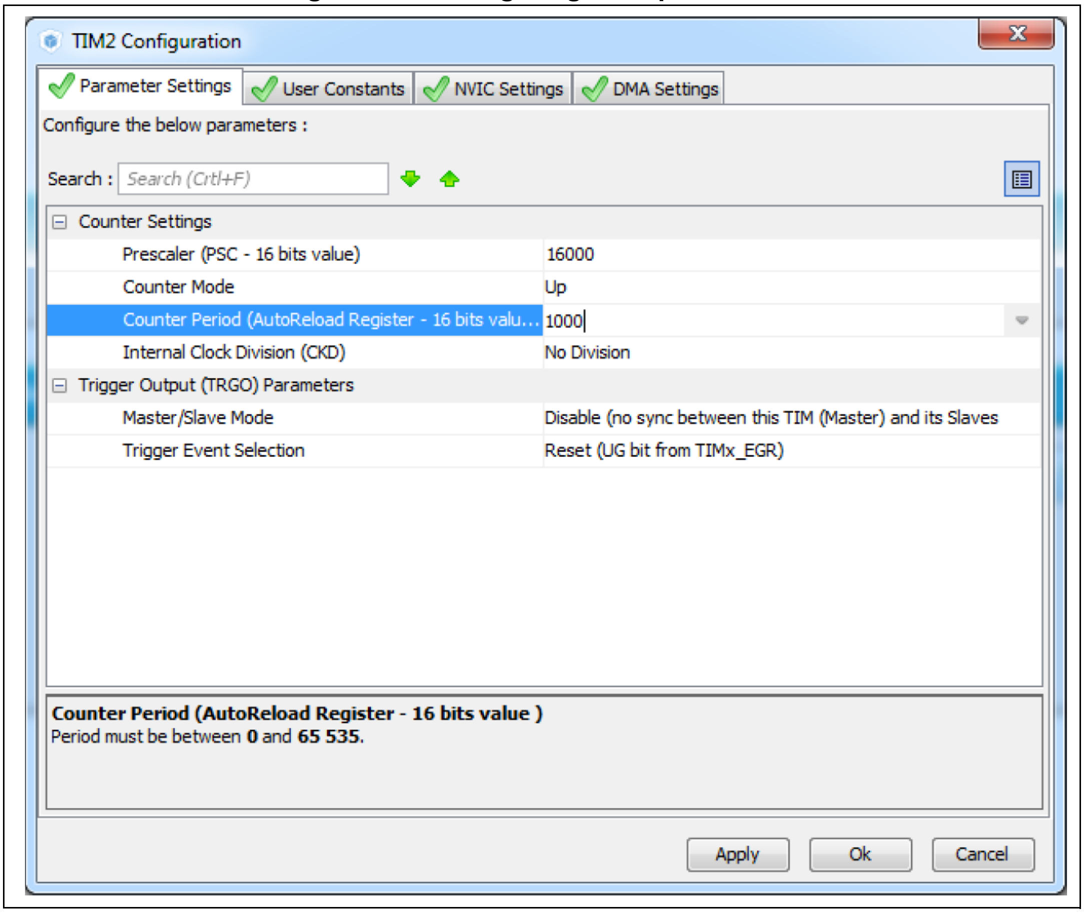
1. From the **Configuration** tab, click **USART2** to open the peripheral **Parameter Settings** window and set the baud rate to 9600. Make sure the Data direction is set to “Receive and Transmit” (see *Figure 607*).
2. Click **OK** to apply the changes and close the window.

Figure 607. Configuring USART2 parameters



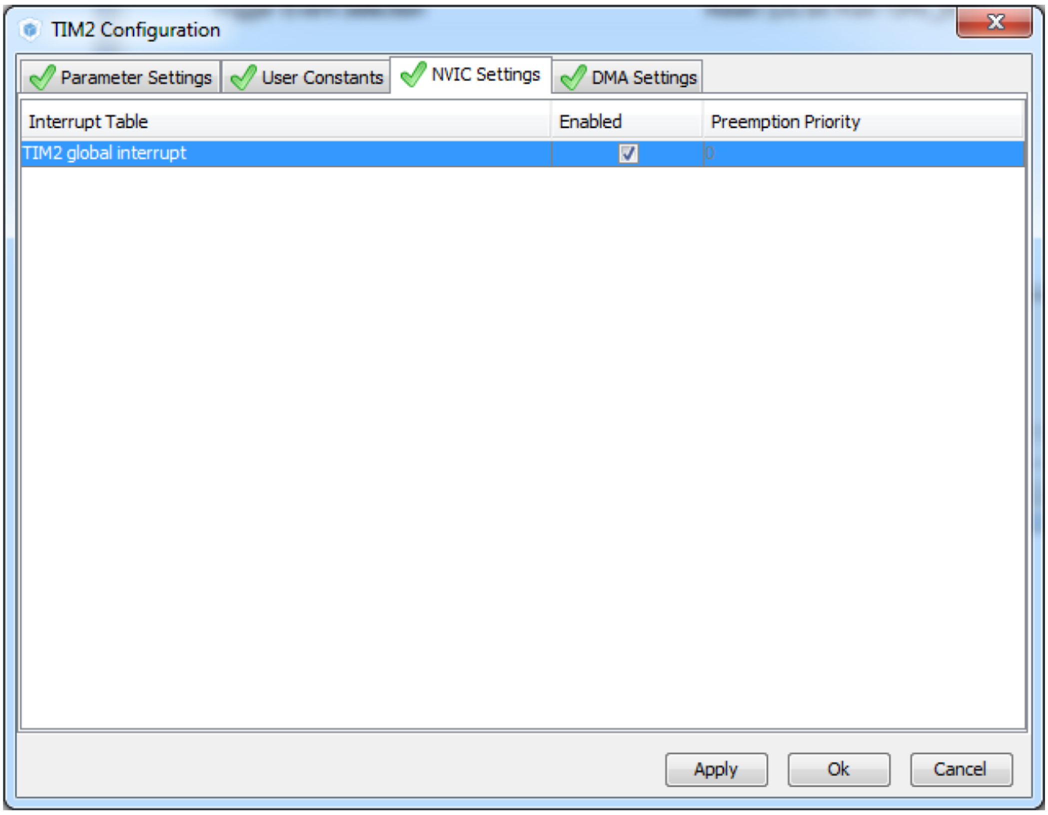
3. Click **TIM2** and change the prescaler to 16000, the Word Length to 8 bits and the Counter Period to 1000 (see *Figure 608*).

Figure 608. Configuring TIM2 parameters



4. Enable TIM2 global interrupt from the **NVIC Settings** tab (see *Figure 609*).

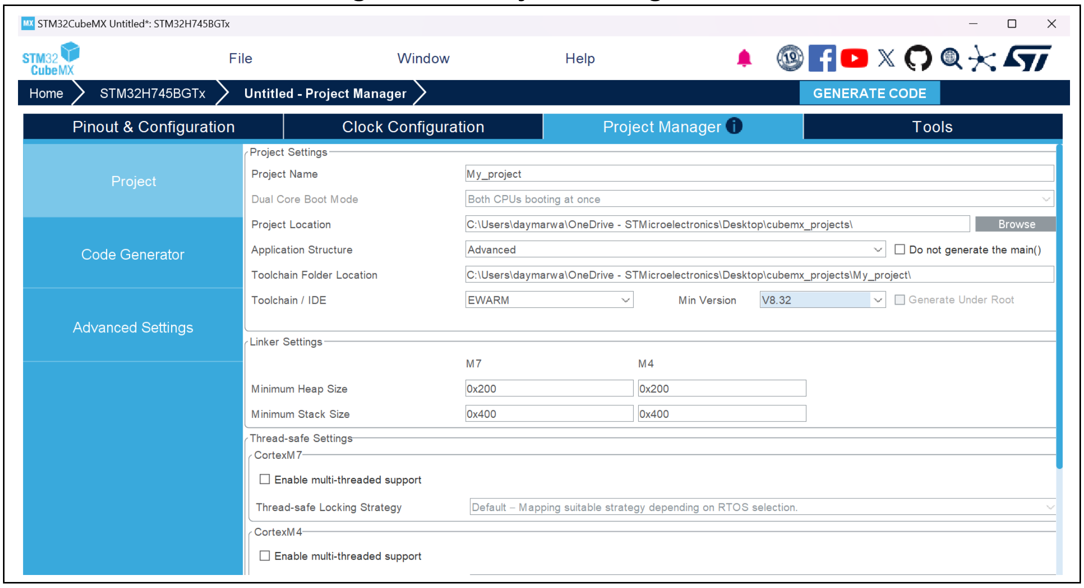
Figure 609. Enabling TIM2 interrupt



14.6 Configuring the project settings and generating the project

1. In the **Project Settings** menu, specify project name and destination folder, and select the EWARM IDE toolchain (see *Figure 610*).

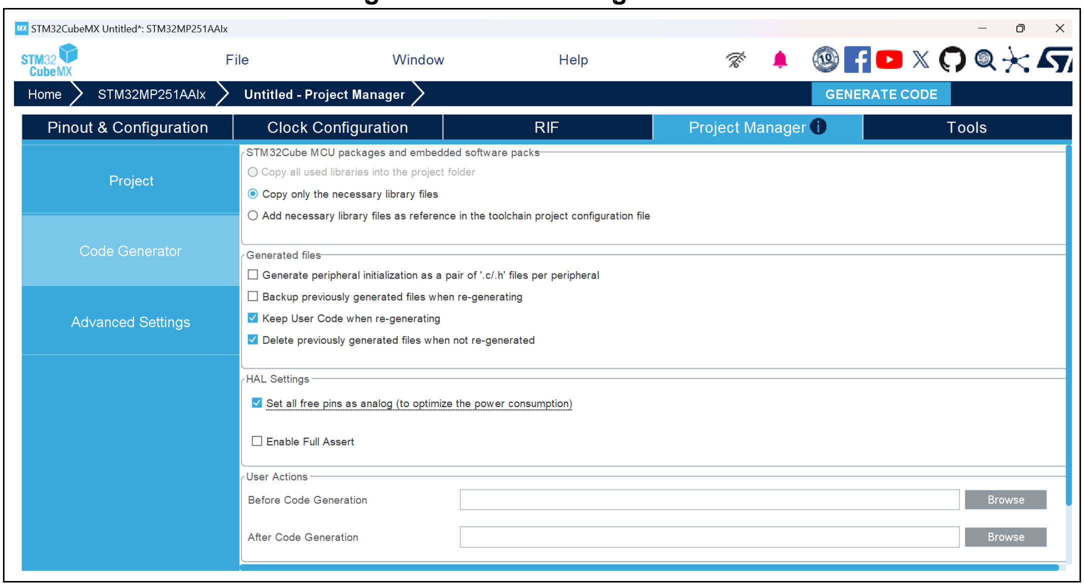
Figure 610. Project Settings menu



If the firmware package version is not already available on the user PC, a progress window opens to show the download progress.

2. In the **Code Generator** tab, configure the code to generate as shown in *Figure 611*, and click **OK** to generate it.

Figure 611. Generating the code



14.7 Updating the project with the user application code

Add the user code as follows:

```
/* USER CODE BEGIN 0 */
#include "stdio.h"
#include "string.h"
/* Buffer used for transmission and number of transmissions */
char aTxBuffer[1024];
int nbtime=1;
/* USER CODE END 0 */
```

Within the main function, start the timer event generation function as follows:

```
/* USER CODE BEGIN 2 */
  /* Start Timer event generation */
    HAL_TIM_Base_Start_IT(&htim2);
  /* USER CODE END 2 */

/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
sprintf(aTxBuffer,"STM32CubeMX rocks %d times \t", ++nbtime);
HAL_UART_Transmit(&huart2,(uint8_t *) aTxBuffer, strlen(aTxBuffer), 5000);
}
/* USER CODE END 4 */
```

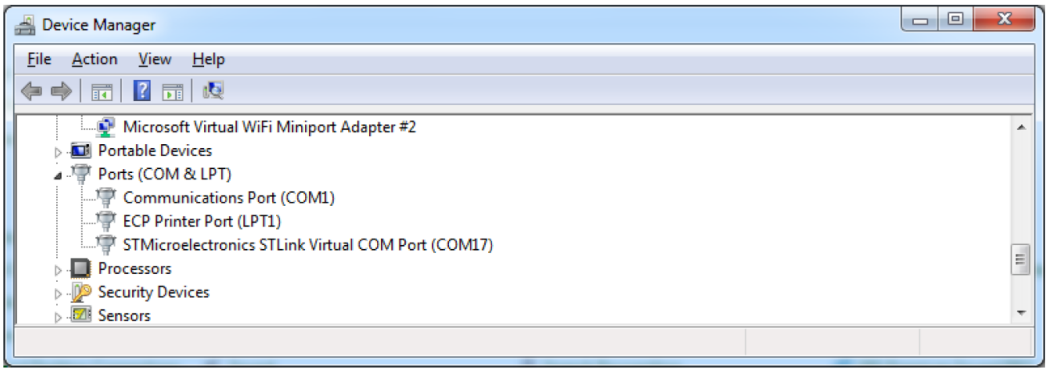
14.8 Compiling and running the project

1. Compile the project within your favorite IDE.
2. Download it to the board.
3. Run the program.

14.9 Configuring Tera Term software as serial communication client on the PC

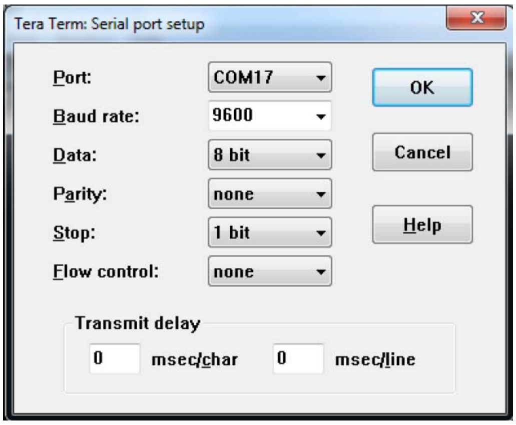
1. On the computer, check the virtual communication port used by ST Microelectronics from the Device Manager window (see *Figure 612*).

Figure 612. Checking the communication port



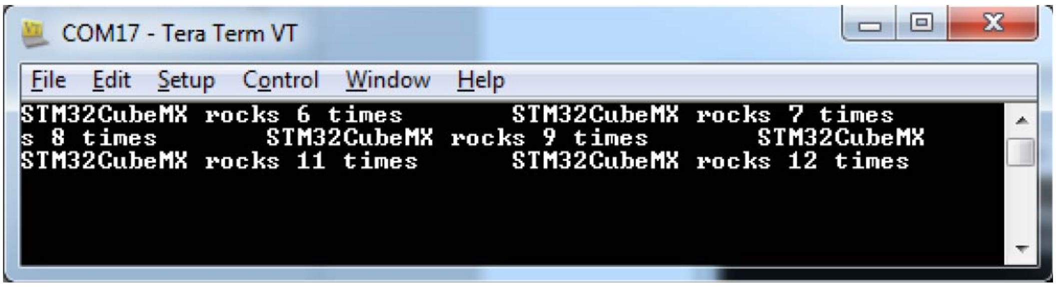
2. To configure Tera Term to listen to the relevant virtual communication port, adjust the parameters to match the USART2 parameter configuration on the MCU (see *Figure 613*).

Figure 613. Setting Tera Term port parameters



3. The Tera Term window displays a message coming from the board at a period of a few seconds (see *Figure 614*).

Figure 614. Setting Tera Term port parameters



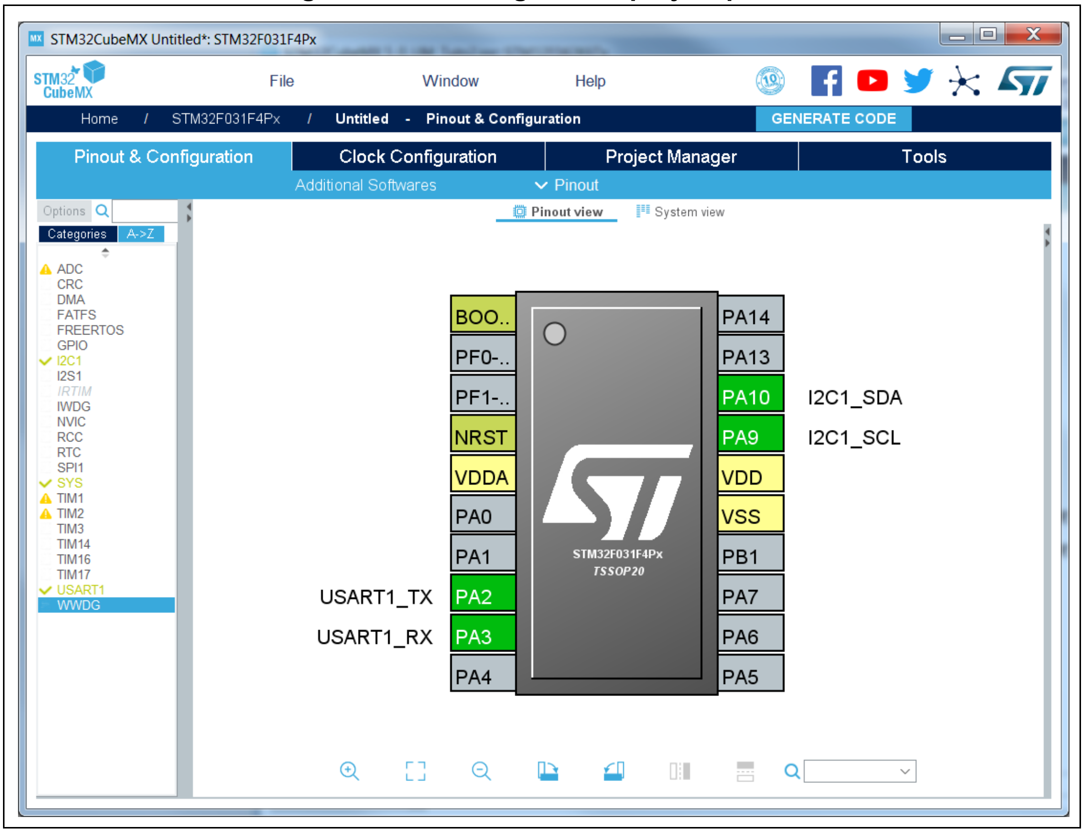
15 Tutorial 5: Exporting current project configuration to a compatible MCU

When **List pinout compatible MCUs** is selected from the **Pinout** menu, STM32CubeMX retrieves the list of the MCUs which are compatible with the current project configuration, and offers to export the current configuration to the newly selected compatible MCU.

This tutorial shows how to display the list of compatible MCUs and export your current project configuration to a compatible MCU:

1. Load an existing project, or create and save a new project:

Figure 615. Existing or new project pinout



2. Go to the **Pinout** menu and select **List Pinout Compatible MCUs**. The **Pinout compatible** window pops up (see *Figure 616* and *Figure 617*).

   If needed, modify the search criteria and the filter options and restart the search process by clicking the **Search** button.

   The color shading and the *Comments* column indicate the level of matching:
   - Exact match: the MCU is fully compatible with the current project (see *Figure 617* for an example).
   - Partial match with hardware compatibility: the hardware compatibility can be ensured but some pin names could not be preserved. Hover the mouse over the desired MCU to display an explanatory tooltip (see *Figure 616* for an example).

– Partial match without hardware compatibility: not all signals can be assigned to the exact same pin location and a remapping will be required. Hover the mouse over the desired MCU to display an explanatory tooltip (see *Figure 617* for an example).

Figure 616. List of pinout compatible MCUs - Partial match with hardware compatibility

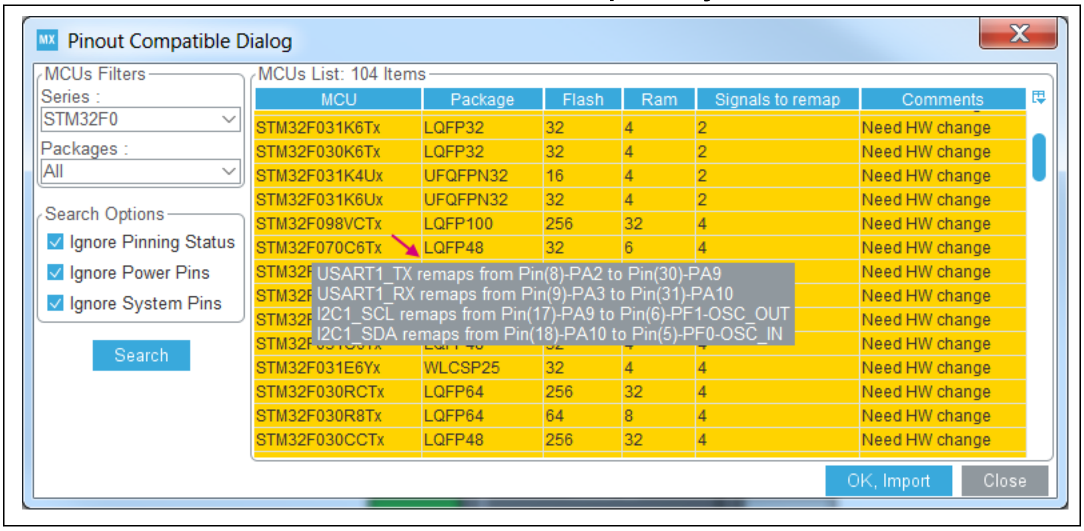
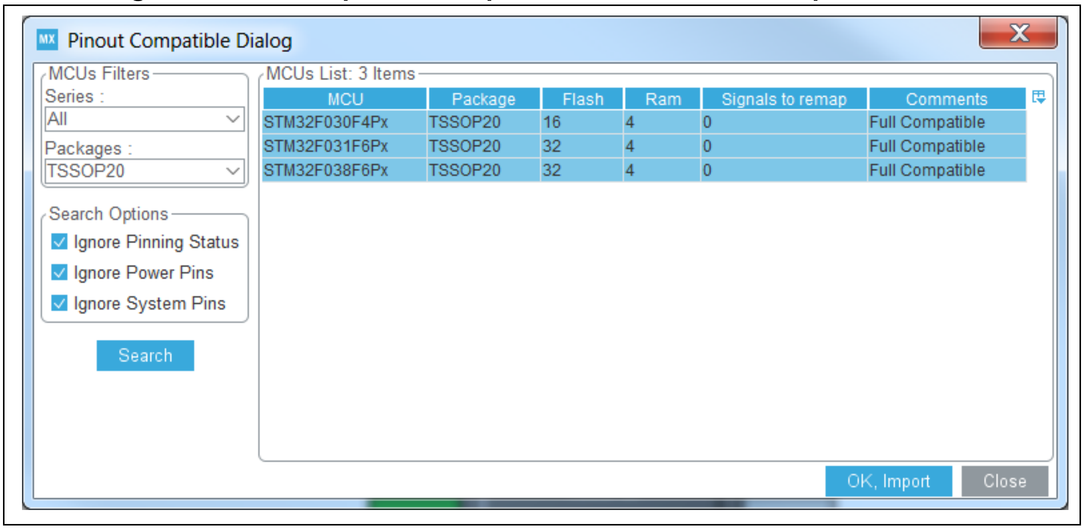
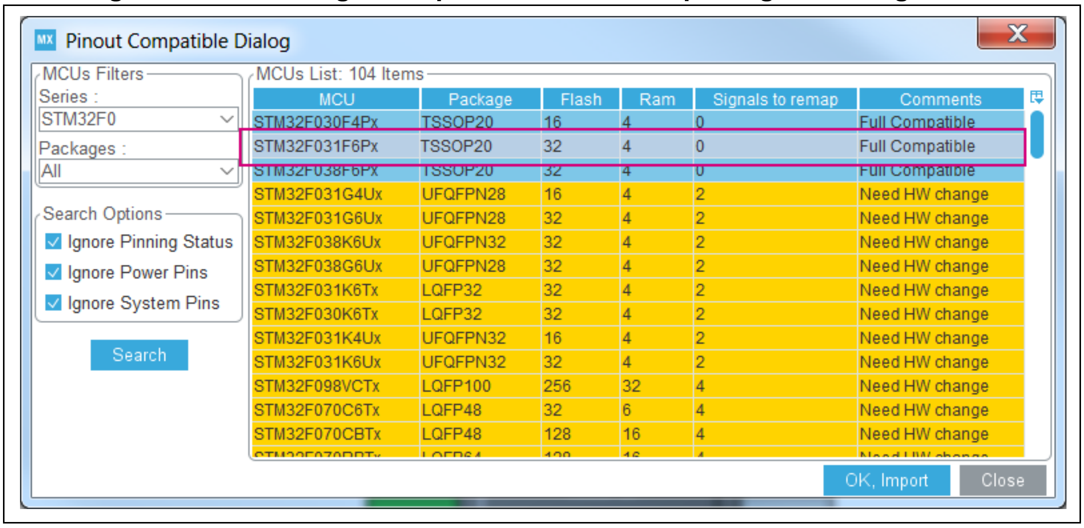


Figure 617. List of pinout compatible MCUs - Exact and partial match



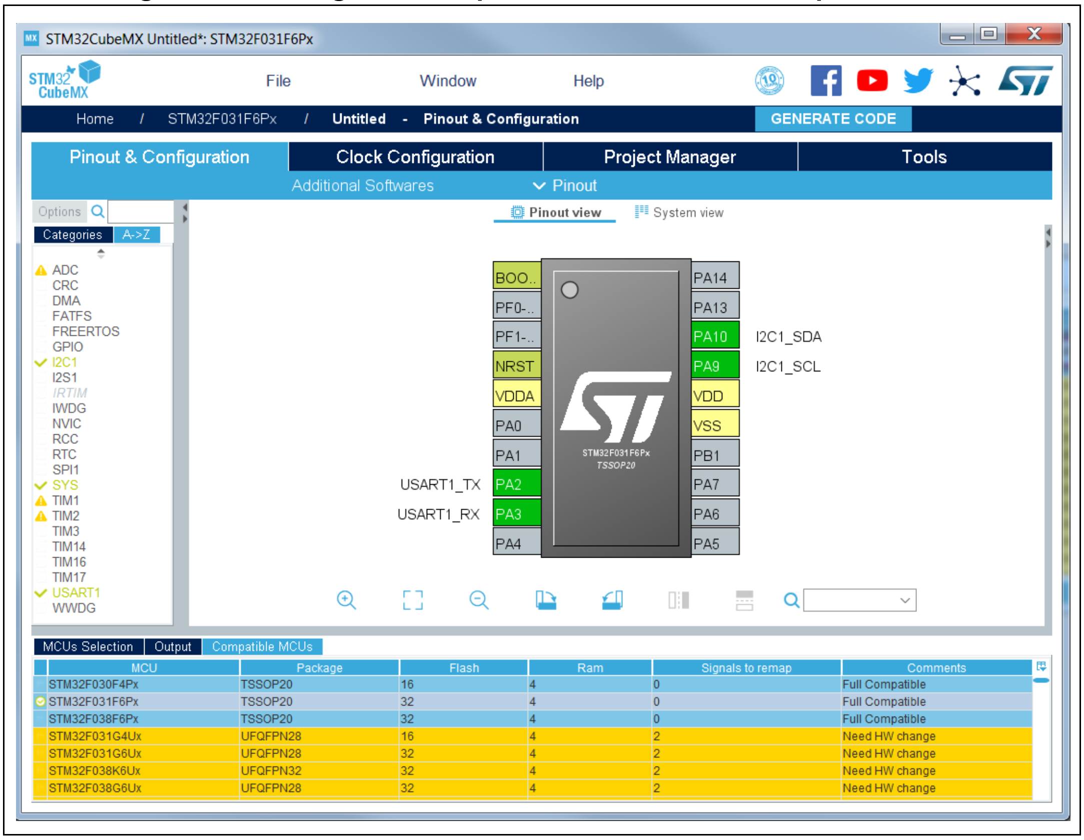
3. Select an MCU to import the current configuration to, and click **OK, Import**:

Figure 618. Selecting a compatible MCU and importing the configuration



The configuration is now available for the selected MCU:

Figure 619. Configuration imported to the selected compatible MCU



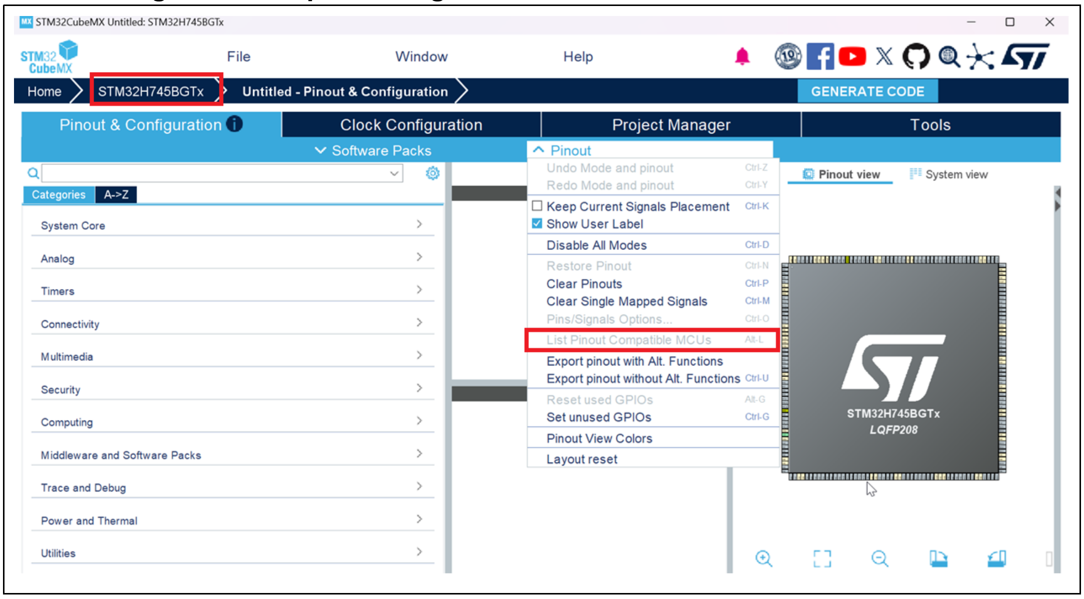
4. To see the list of compatible MCUs, select **Outputs** under the **Window** menu.

   To load the current configuration to another compatible MCU, double-click the list of compatible MCUs.

5. To remove some constraints on the search criteria, several solutions are possible:
   - Select the **Ignore Pinning Status** checkbox to ignore pin status (locked pins).
   - Select the **Ignore Power Pins** checkbox to not take into account the power pins.
   - Select the **Ignore System Pins** to not take into account the system pins. Hover the mouse over the checkbox to display a tooltip that lists the system pins available on the current MCU.

Note: The option "List of Compatible MCU" is available only for single-core products, and grayed out for dual-core microcontrollers, as exemplified in Figure 620.

Figure 620. Import configuration not available for dual core MCUs



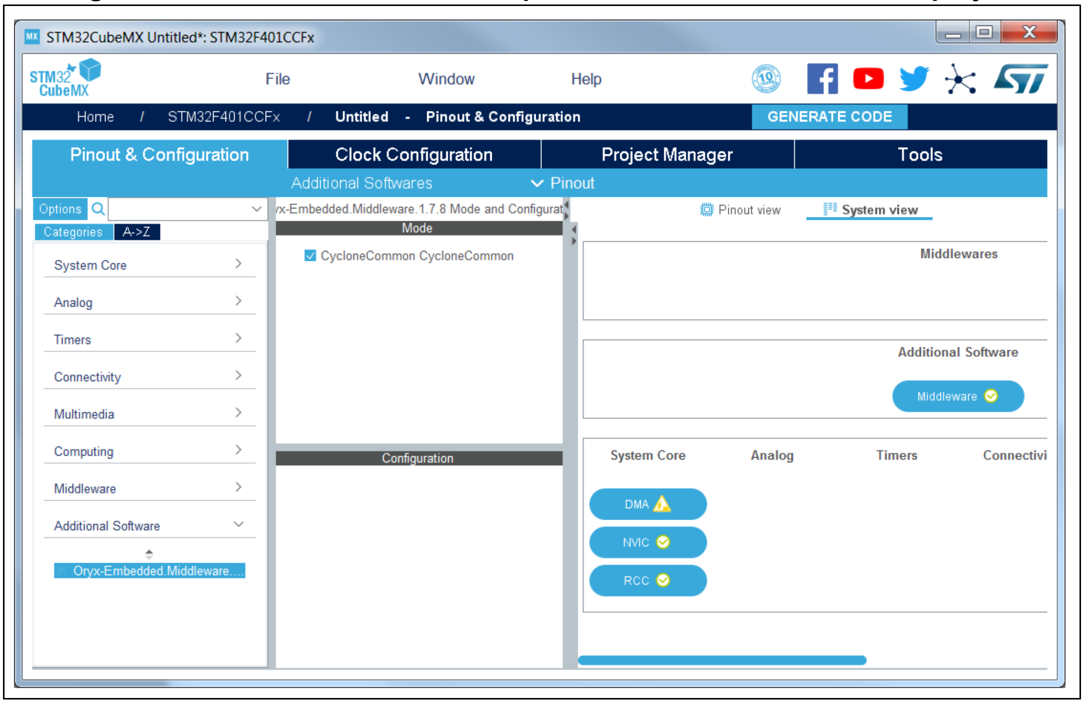
16 Tutorial 6 – Adding embedded software packs to user projects

In this tutorial, the Oryx-Embedded.Middleware.1.7.8. pack is taken as an example to demonstrate how to a to add pack software components to STM32CubeMX projects. The use of this package shall not be understood as an STMicroelectronics recommendation.

To add embedded software packs to your project, proceed as follows:

1. Install Oryx-Embedded.Middleware.1.7.8.pack using the .pdsc file available from http://www.oryx-embedded.com (see *Section 3.4.5: Installing embedded software packs*).
2. Select **New project**.
3. Select STM32F01CCFx from the **MCU selector**.
4. Select **Additional Software** from the **Pinout & Configuration** view to open the additional software component window and choose the following software components: Compiler Support, RTOS Port/None and Date Time Helper Routines from the CycloneCommon bundle (see *Section 4.15: Software Packs component selection window*).
5. Click **OK** to display the selected components on the tree view and click the checkbox to enable the software components for the current project (see *Figure 621*).

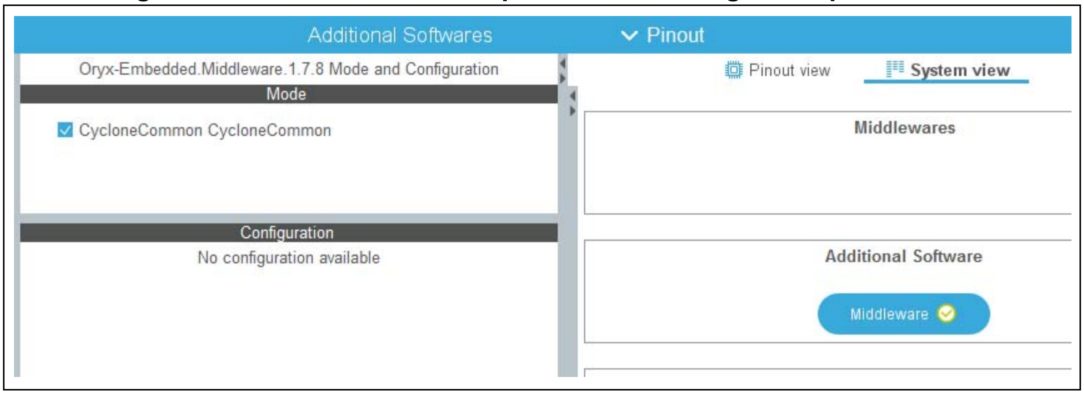
Figure 621. Additional software components enabled for the current project



The pack name highlighted in green indicates that all conditions for the selected software components resolve to true. If at least one condition is not resolved, the pack name is highlighted in orange.

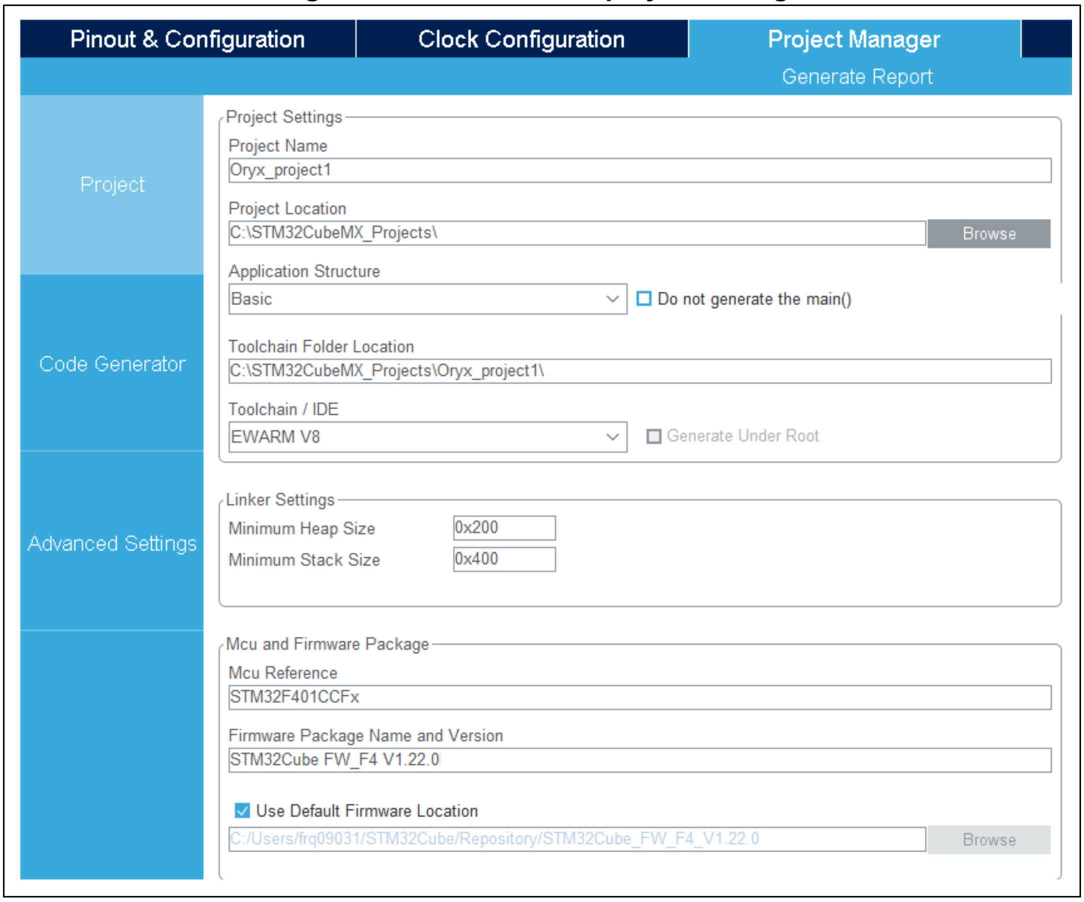
6. Check that no parameters can be configured in the **Configuration** tab (see *Figure 622*).

Figure 622. Pack software components: no configurable parameters



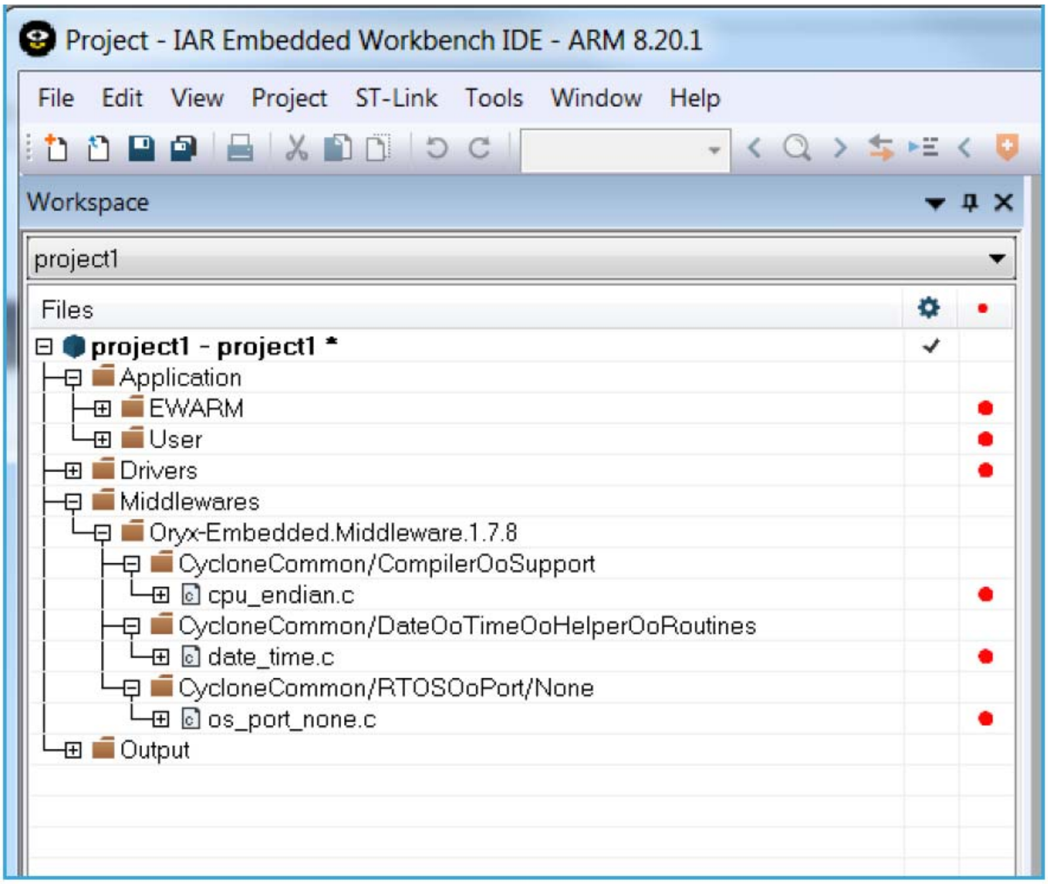
7. Select the **Project manager** project tab to specify project parameters (see *Figure 623*), and choose IAR™ EWARM as IDE.

Figure 623. Pack tutorial: project settings



8. Generate your project by clicking GENERATE CODE . Accept to download the STM32CubeF4 MCU package if it is not present in STM32Cube repository.
9. Click **Open project**. The Oryx software components are displayed in the generated project (see *Figure 624*).

Figure 624. Generated project with third party pack components



17 Tutorial 7 – Using the X-Cube-BLE1 software pack

This tutorial demonstrates how to achieve a functional project using the X-Cube-BLE1 software pack.

Below the prerequisites to run this tutorial:

- Hardware: NUCLEO-L053R8, X-NUCLEO-IDB05A1 and mini-USB cable (see *Figure 625*)
- Tools: STM32CubeMX, IDE (Atollic® or any other toolchain supported by STM32CubeMX)
- Embedded software package: STM32CubeL0 (version 1.10.0 or higher), X-Cube-BLE1 1.1.0 (see *Figure 626*).
- Mobile application (see *Figure 627*): STMicroelectronics BlueNRG application for iOS® or Android™

Figure 625. Hardware prerequisites

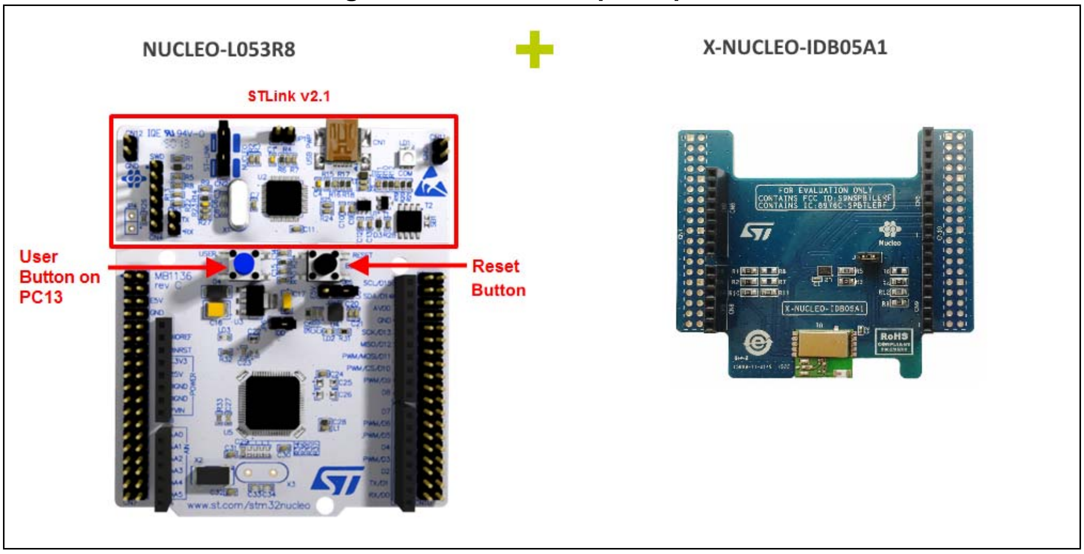


Figure 626. Embedded software packages

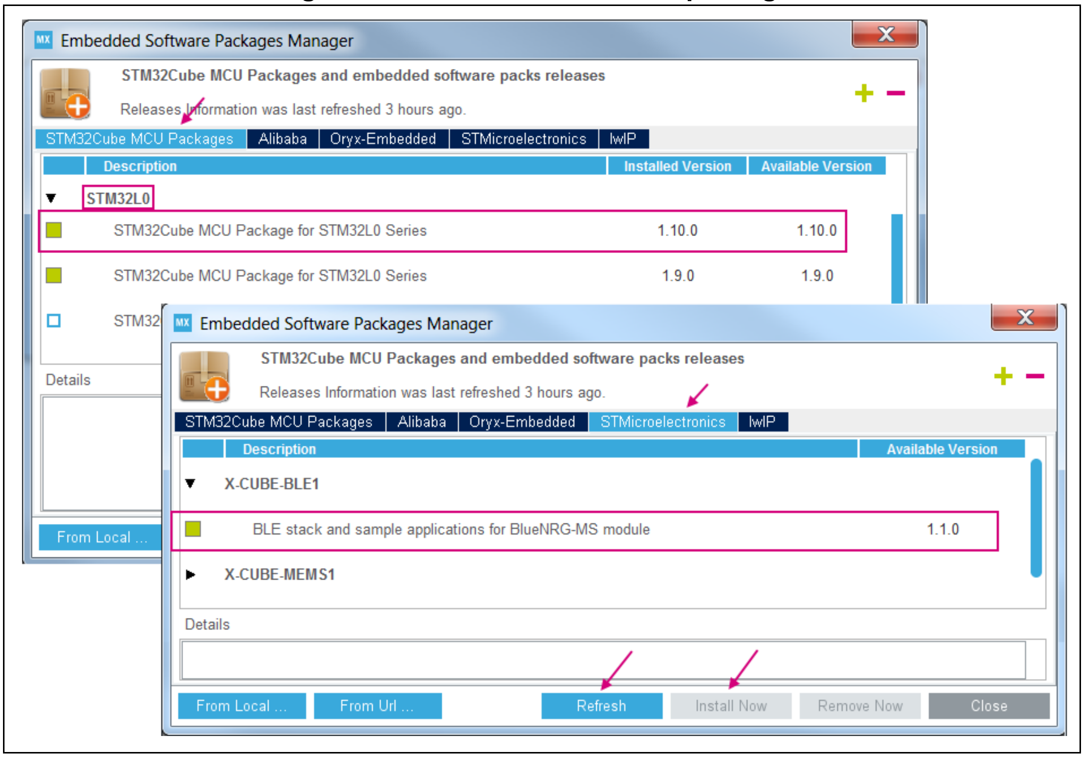
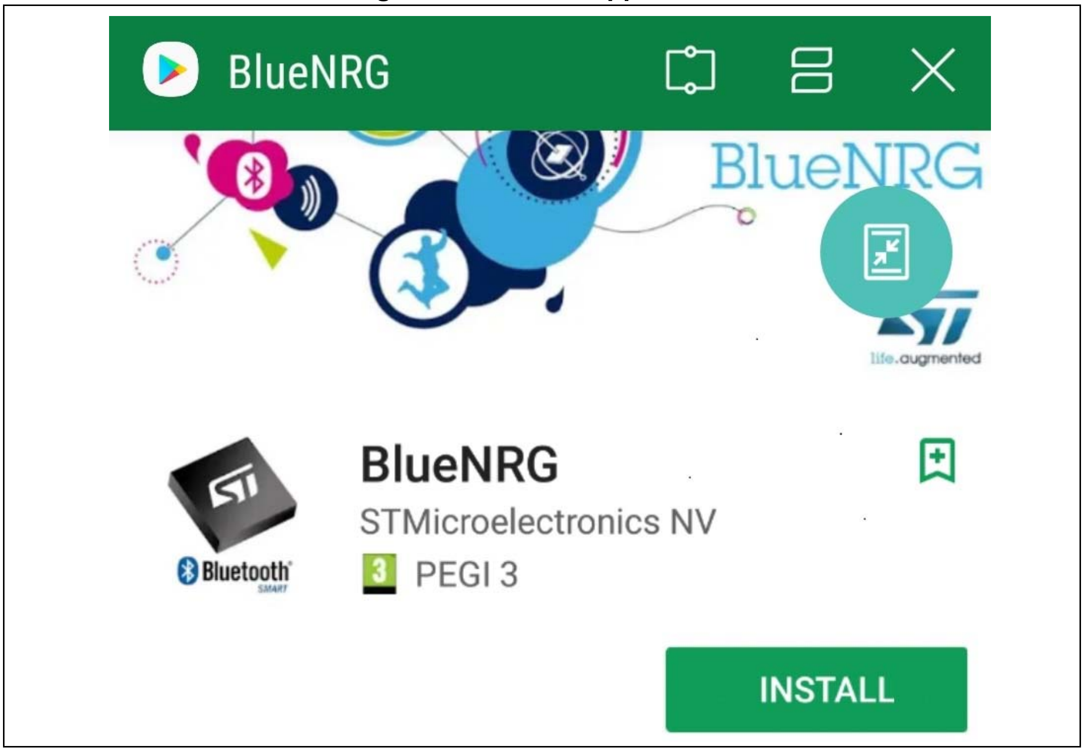


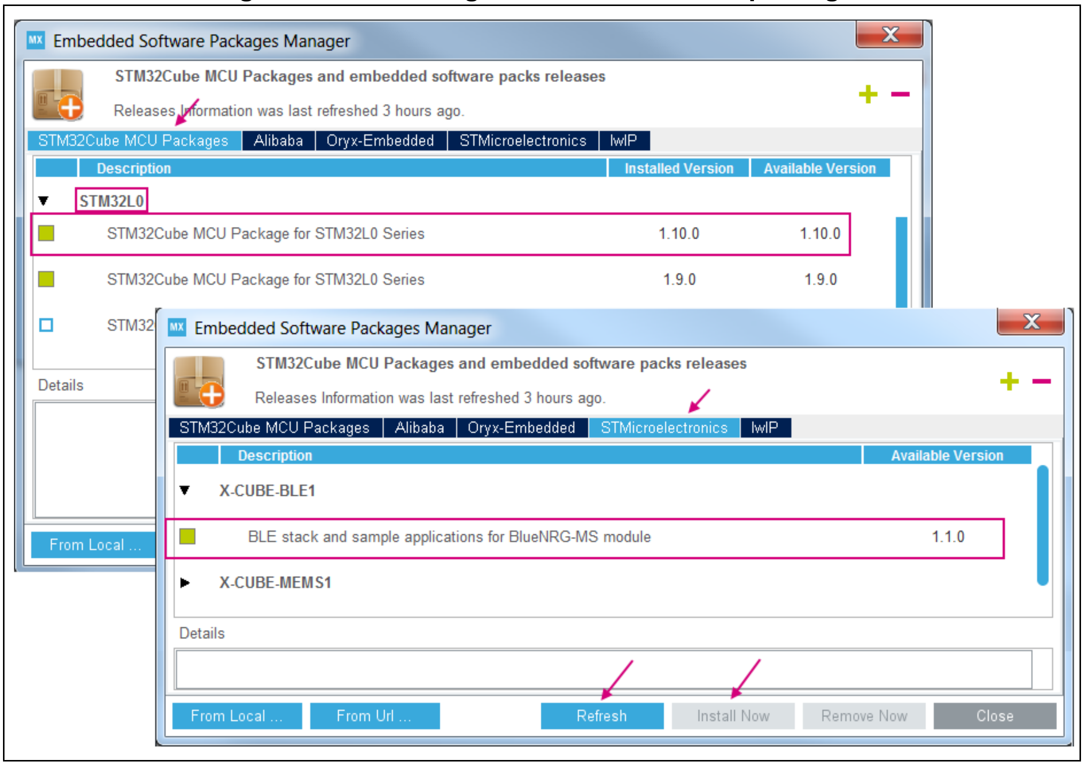
Figure 627. Mobile application



Proceed as follows to install and run the tutorial:

1. Check STM32CubeMX Internet connection:
   a) Select the **Help > Updater Settings** menu to open the updater window.
   b) Verify in the **Connection** tab that the Internet connection is configured and up.
2. Install the required embedded software packages (see *Figure 628*):
   a) Select the **Help > Manage Embedded software packages** menu to open the **embedded software package manager** window.
   b) Click the **Refresh** button to refresh the list with the latest available package versions.
   c) Select the **STM32Cube MCU Package** tab and check that the STM32CubeL0 firmware package version 1.10.0 or higher is installed (the checkbox must be green). Otherwise select the checkbox and click **Install now**.
   d) Select the **STMicrolectronics** tab and check that the X-Cube-BLE1 software pack version 1.0.0 is installed (checkbox must be green). Otherwise, select the checkbox and click **Install now**.

Figure 628. Installing Embedded software packages



3. Start a new project:
   a) Select **New Project** to open the new project window.
   b) Select the **Board selector** tab.
   c) Select Nucleo64 as board type and STM32L0 as MCU Series.
   d) Select the NUCLEO-L053R8 from the resulting board list (see *Figure 629*).
   e) Answer **No** when prompted to initialize all peripherals in their default mode (see *Figure 630*).

Figure 629. Starting a new project - selecting the NUCLEO-L053R8 board

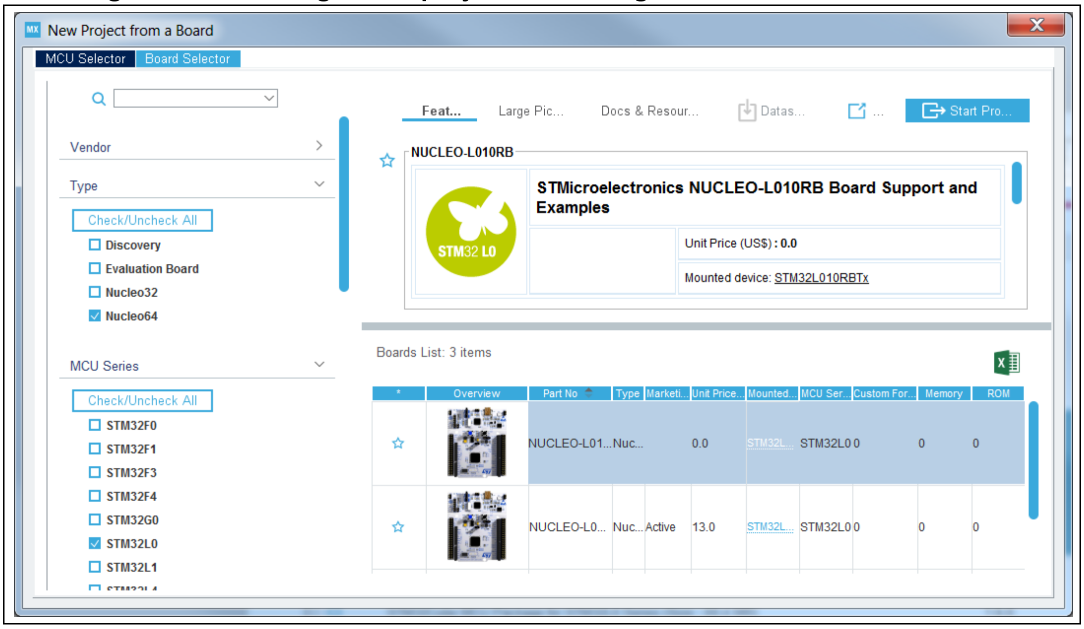
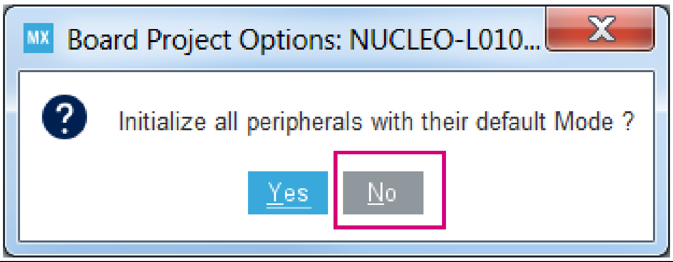


Figure 630. Starting a new project - initializing all peripherals



4. Add X-Cube-BLE1 components to the project:
   a) Click **Additional Software** from **Pinout & Configuration** view to open the **Additional Software component Selection** window.
   b) Select the relevant components (see *Figure 631*)

      The Application group comes with a list of applications: the C files implement the application loop, that is the *Process()* function. From the Application group, select the **SensorDemo** application.

      Select the **Controller** and **Utils** components

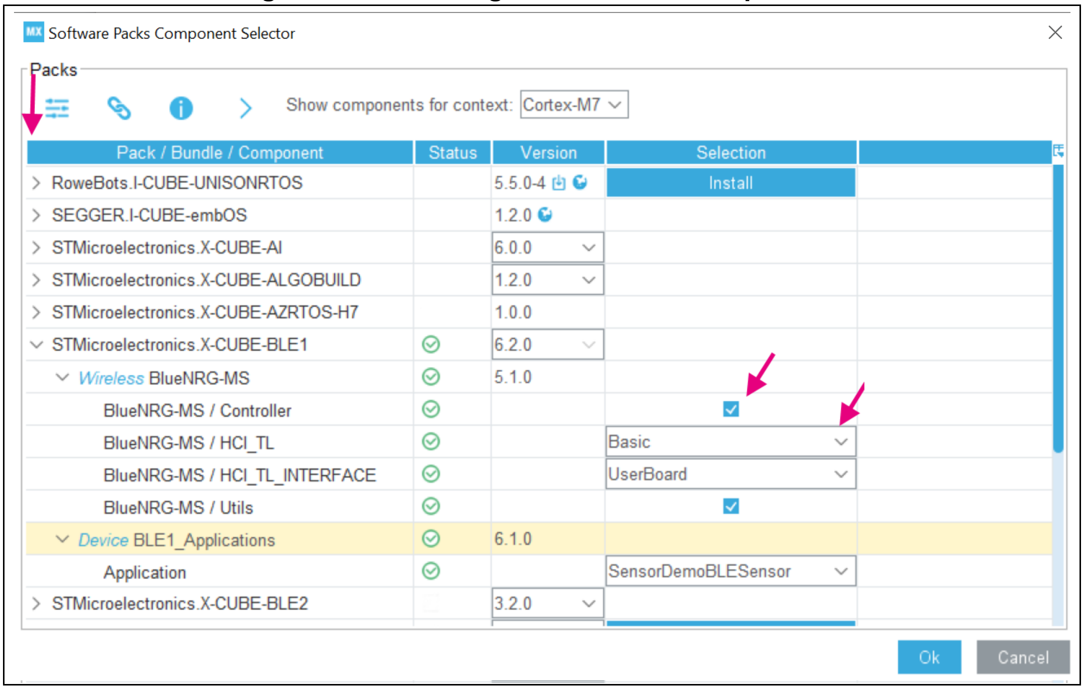
      Select the **Basic** variant for the **HCI_TL** component**.** The Basic variant provides the STMicroelectronics implementation of the HCI_TL API while the template option requires users to implement their own code.

      Select the **UserBoard** variant as **HCI_TL_INTERFACE** component. Using the UserBoard option generates the <boardname>_bus.c file, that is nucleo_l053r8_bus.c for this tutorial, while the template option generates the custom_bus.c file and requires users to provide their own implementation.

      Refer to the X-Cube-BLE1 pack documentation for more details on software components.

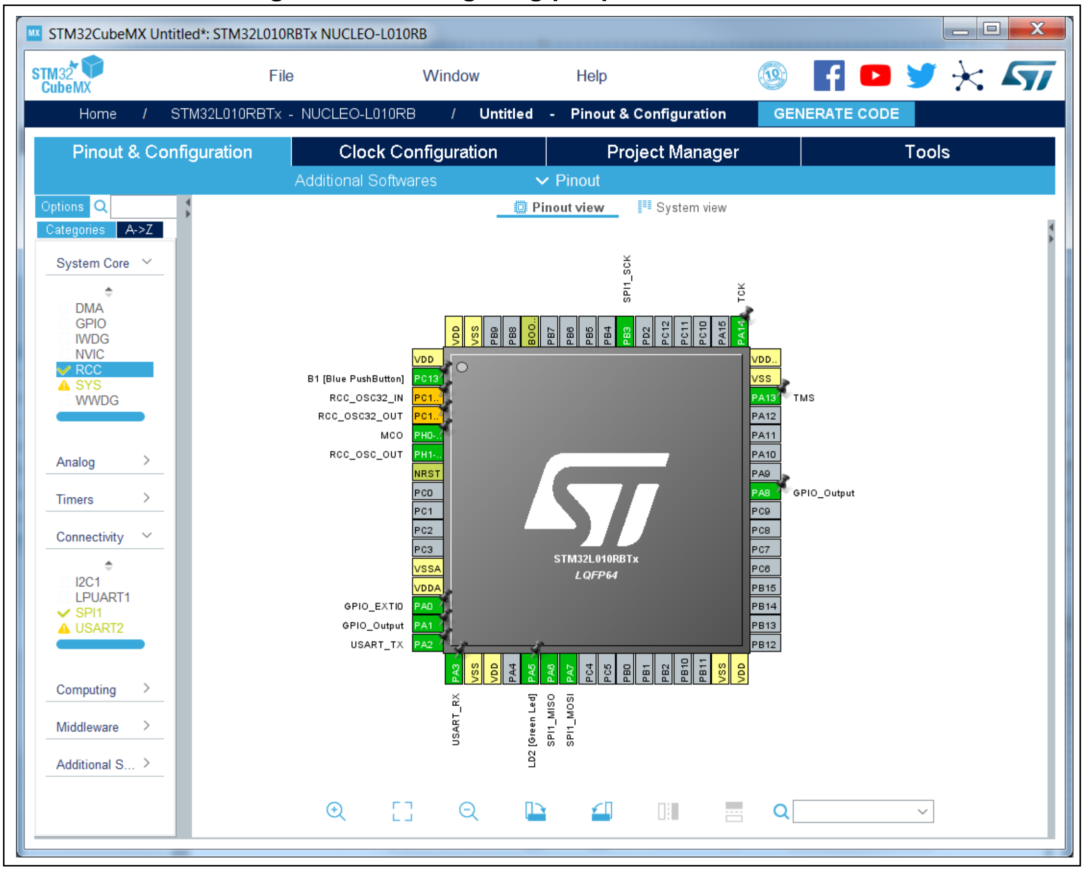
c) Click **OK** to apply the selection to the project and close the window. The left panel **Additional Software** section is updated accordingly.

Figure 631. Selecting X-Cube-BLE1 components



5. Enable peripherals and GPIOs from the **Pinout** tab (see *Figure 632*):
   a) Configure **USART2** in **Asynchronous** mode.
   b) Configure **SPI1** in **Full-duplex master** mode.
   c) Left-click the following pins and configure them for the required GPIO settings:
      PA0: GPIO_EXTI0
      PA1: GPIO_Output
      PA8: GPIO_Output
   d) Enable **Debug Serial Wire** under **SYS** peripheral.

Figure 632. Configuring peripherals and GPIOs



6. Configure the peripherals from the **Configuration** tab:
   a) Click the **NVIC** button under the **System** section to open the **NVIC configuration** window. Enable EXTI line 0 and line 1 interrupts and click **OK** (see *Figure 633*).
   b) Click the **SPI** button under the **Connectivity** section to open the **SPI configuration** window. Check that the data size is set to 8 bits and the prescaler value to 16 so that HCLK divided by the prescaler value is less or equal to 8 MHz.
   c) Click **USART2** under the **Connectivity** section to open the **Configuration** window and check the following parameter settings:

      Under **Parameter Settings**:

      Baud rate: 115200 bits/s

      Word length: 8 bits (including parity)

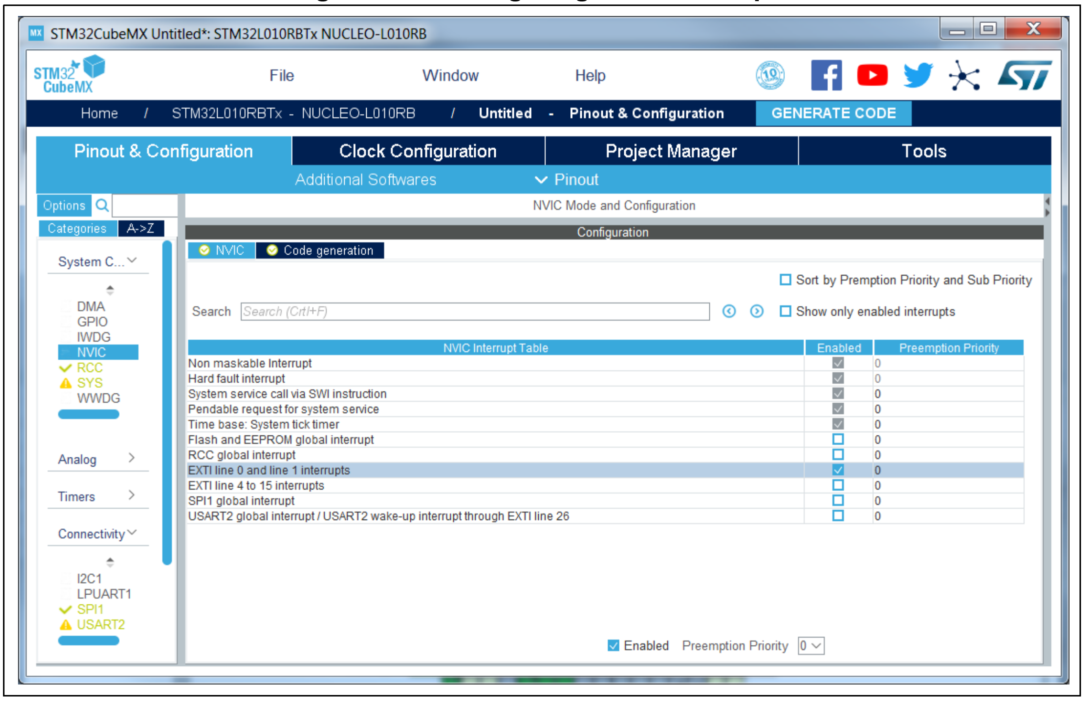
      Parity: none

      Stop bits: 1

      Under **GPIO Settings**:

      User labels: USART_TX and USART_RX

Figure 633. Configuring NVIC interrupts



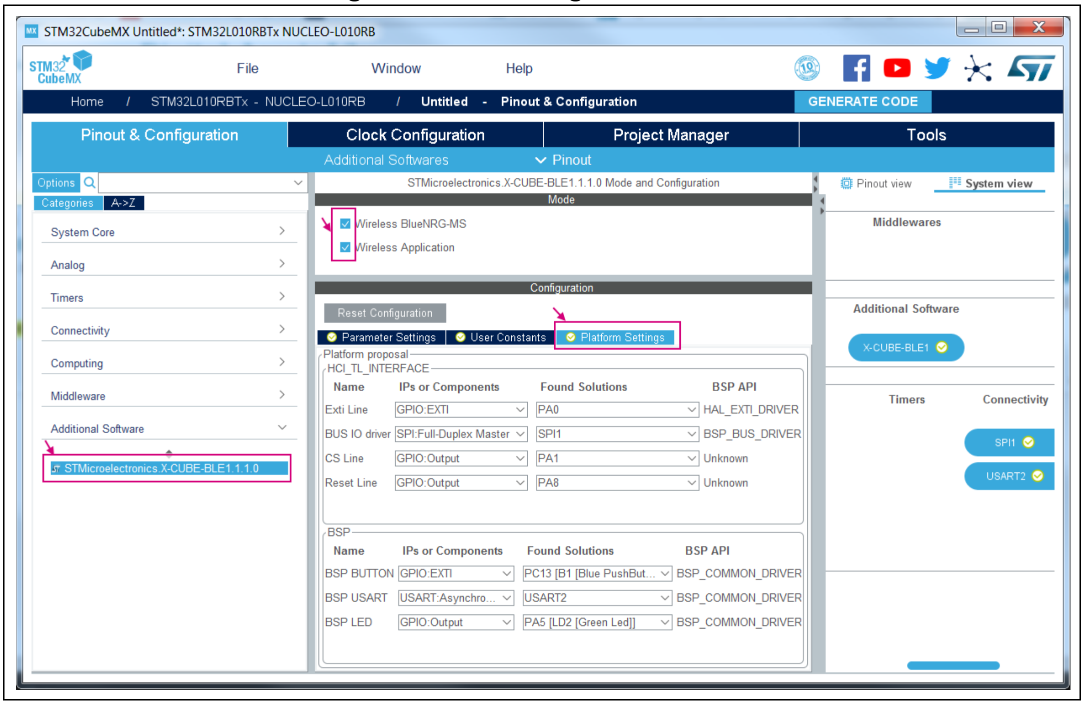
7. Enable and configure X-Cube-BLE1 pack components from the **Pinout & Configuration** view:
   a) Click the pack items from the left panel to show the mode and configuration tabs.
   b) Click the check boxes from the Mode panel to enable X-Cube-BLE1, the configuration panel appears showing the parameters to configure. An orange triangle indicates that some parameters are not configured. It turns into a green check mark once all parameters are correctly configured (see *Figure 634*).
   c) Leave the Parameter Settings Tab unchanged.
   d) Go the Platform settings tab, configure the connection with the hardware resources as indicated in *Figure 634* and *Table 29*.

Table 29. Connection with hardware resources

Name	IPs or components	Found solutions
BUS IO driver	SPI in Full-duplex master mode	SPI1
EXTI Line	GPIO:EXTI	PA0
CS Line	GPIO:output	PA1
Reset Line	GPIO:output	PA8
BSP LED	GPIO:output	PA5
BSP Button	GPIO:EXTI	PC13
BSP USART	USART in Asynchronous mode	USART2

Check that the icon turns to ✓. Click **OK** to close the **Configuration** window**.**

Figure 634. Enabling X-Cube-BLE1



8. Generate the SensorDemo project:
   a) Click GENERATE CODE to generate the code. The **Project Settings** window opens if the project has not yet been saved.
   b) Click GENERATE CODE to generate the code once the project settings have been properly configured (see *Figure 635*). When the generation is complete, a dialog window requests to open the project folder (Open Folder) or to open the project in IDE toolchain (Open Project). Select **Open Project** (see *Figure 636*).

Figure 635. Configuring the SensorDemo project

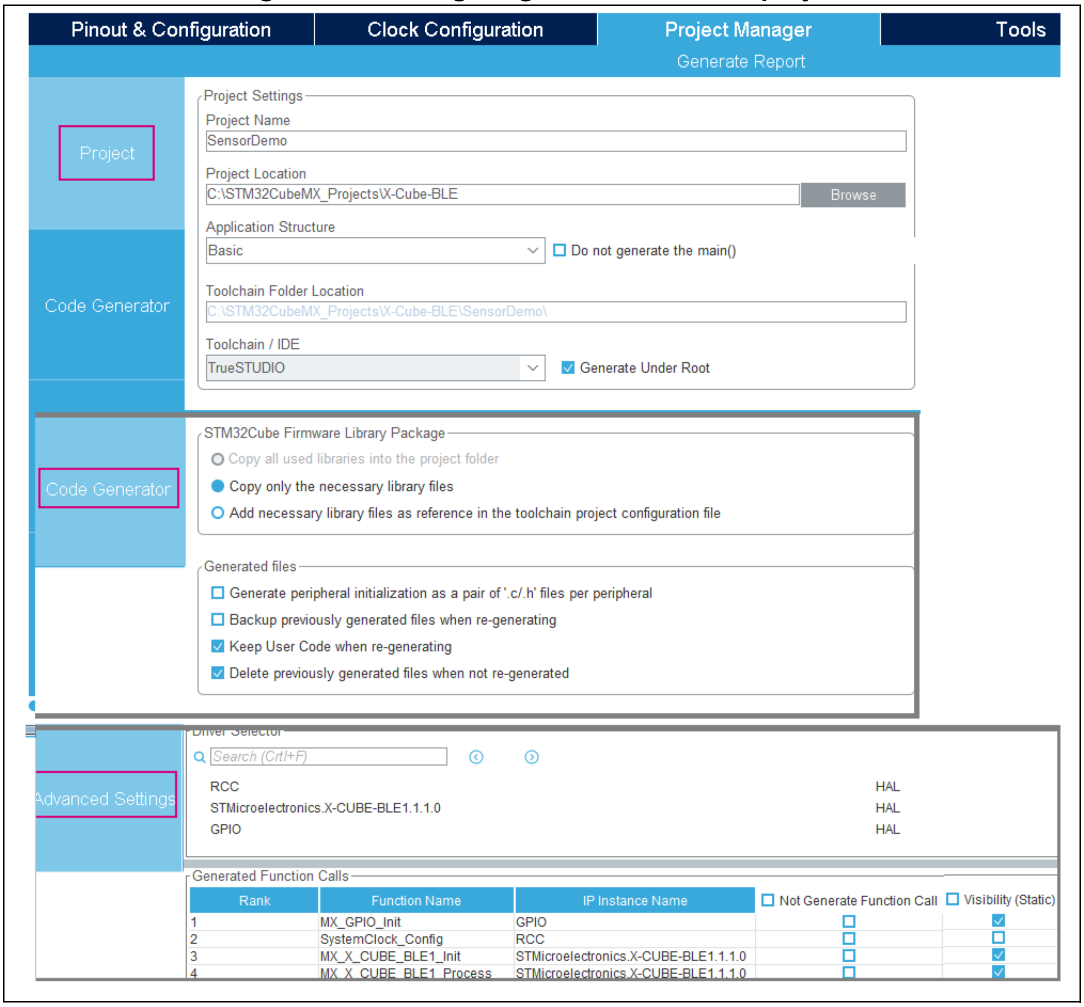
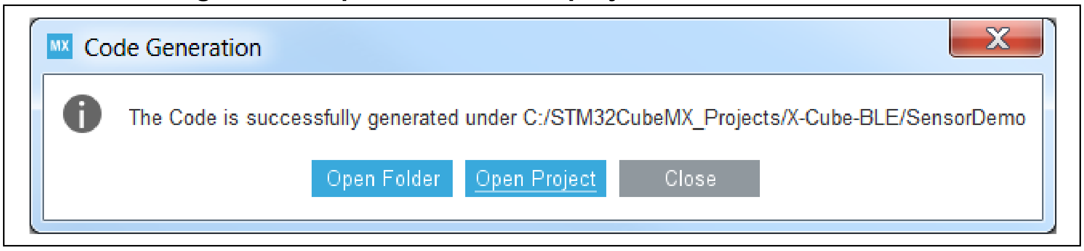


Figure 636. Open SensorDemo project in the IDE toolchain



18 Creating LPBAM projects

18.1 LPBAM overview

Disclaimer: to learn about the LPBAM mode and its usage, it is recommended to read the LPBAM application note available on *www.st.com,* and the LPBAM utility getting started guide located under the Utilities folder of the STM32Cube firmware package.

18.1.1 LPBAM operating mode

LPBAM stands for low power background autonomous mode. It is an operating mode that allows peripherals to be functional and autonomous independently from power modes and without any software running. It is performed thanks to a hardware subsystem embedded in STM32 products. Thanks to DMA transfers in Linked-list mode, the LPBAM subsystem can chain different actions to build a useful functionality (peripheral configurations and transfers). Optionally, it can generate asynchronous events and interrupts. It operates without any CPU intervention. Consequently, the two major benefits from using the LPBAM subsystem mechanisms are an optimized power consumption, and an offloaded CPU.

18.1.2 LPBAM firmware

The LPBAM firmware has been designed to help users create LPBAM applications: the LPBAM utility is a set of modular drivers located under the Utilities folder of the STM32Cube firmware package. Each module comes as a pair of C file that provides the APIs needed to build an application scenario. Each module manages the configurability and the data transfers for a given peripheral. The LPBAM utility is designed to be compatible with any STM32 devices supporting LPBAM subsystem mechanisms through a configuration module: it requires a configuration file stm32_lpbam_conf.h aligned with the application needs. The LPBAM utility has a single application entry point, the stm32_lpbam.h, that must be included in the project.

18.1.3 Supported series

The LPBAM firmware supports STM32U575/585, STM32U595/5A5 and STM32U599/5A9 products, for projects with or without TrustZone activated.

STM32CubeMX 6.5.0 introduces LPBAM for projects without TrustZone activated on the STM32U575/585 product line: users can create LPBAM applications for their project using STM32CubeMX LPBAM Scenario & Configuration view and generate the corresponding code. The generated C project embeds the LPBAM firmware.

STM32CubeMX 6.6.0 adds LPBAM support for projects with TrustZone activated.

18.1.4 LPBAM design

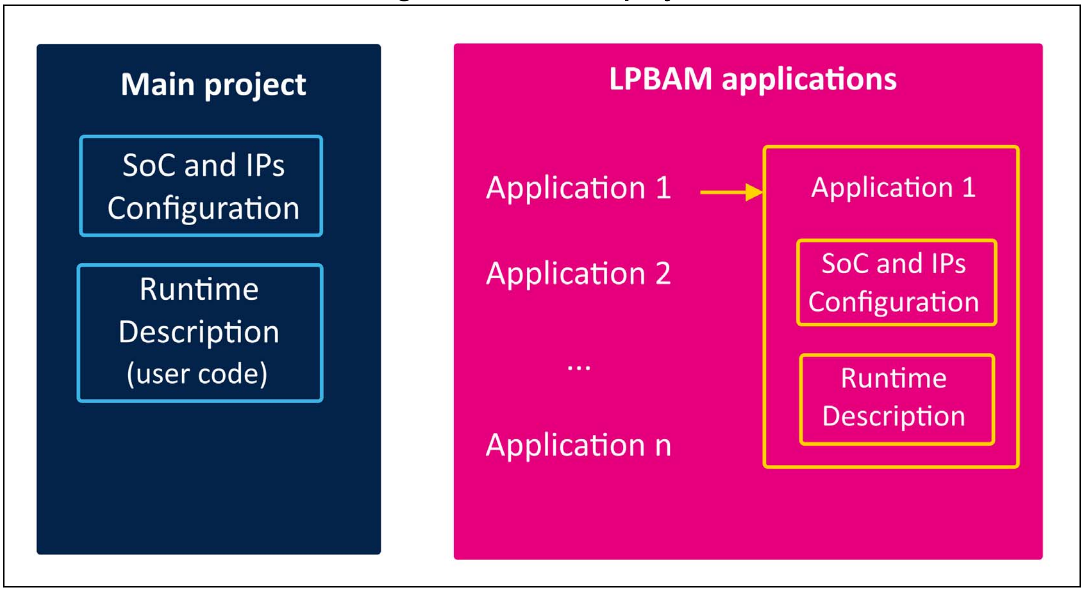
It is recommended to use LPBAM to save power and offload the CPU.

- The LPBAM mechanism supports the following set of peripherals on the Smart Run Domain: ADC4, COMP1/2, DAC1, I2C3, LPDMA1, LPGPIO, LPTIM1/2/3, LPUART1, OPAMP1/2, SPI3, VREFBUF.
- According to the LPDMA implementation in the Smart run domain, the LPBAM has access only to SRAM4.
- The LPBAM mechanism implementation can run autonomously until Stop2 mode.
- To reach the lowest power consumption, the system power usage, the system clock and the autonomous peripheral kernel clock can be configured:

18.1.5 LPBAM project support in STM32CubeMX

An LPBAM project is composed of a main project, and of one or more LPBAM applications.

Figure 637. LPBAM project



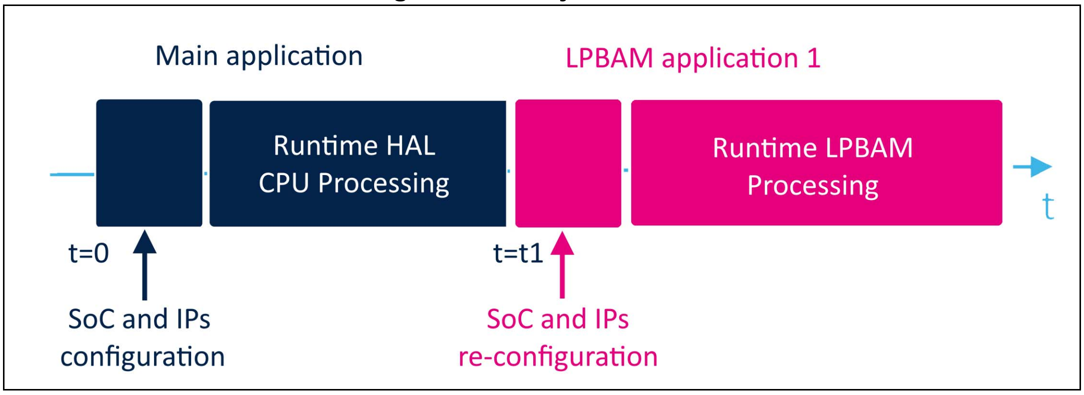
The “Main project” contains the “SoC and IPs configuration” at initialization time and a runtime description of the main application. STM32CubeMX allows to describe the “SOC and IPs Configuration” part.

Each LPBAM application contains a “SoC and IPs configuration” and a runtime description. STM32CubeMX allows to describe both.

STM32CubeMX generated code for “SoC and IPs configurations” uses the STM32Cube HAL and/or LL APIs, for both the main project and the LPBAM application. The code generated for the LPBAM application runtime uses the LPBAM firmware API.

Figure 638 is an example of what can be executed at runtime for a simple LPBAM project composed of the main application and of one LPBAM application.

Figure 638. Project timeline

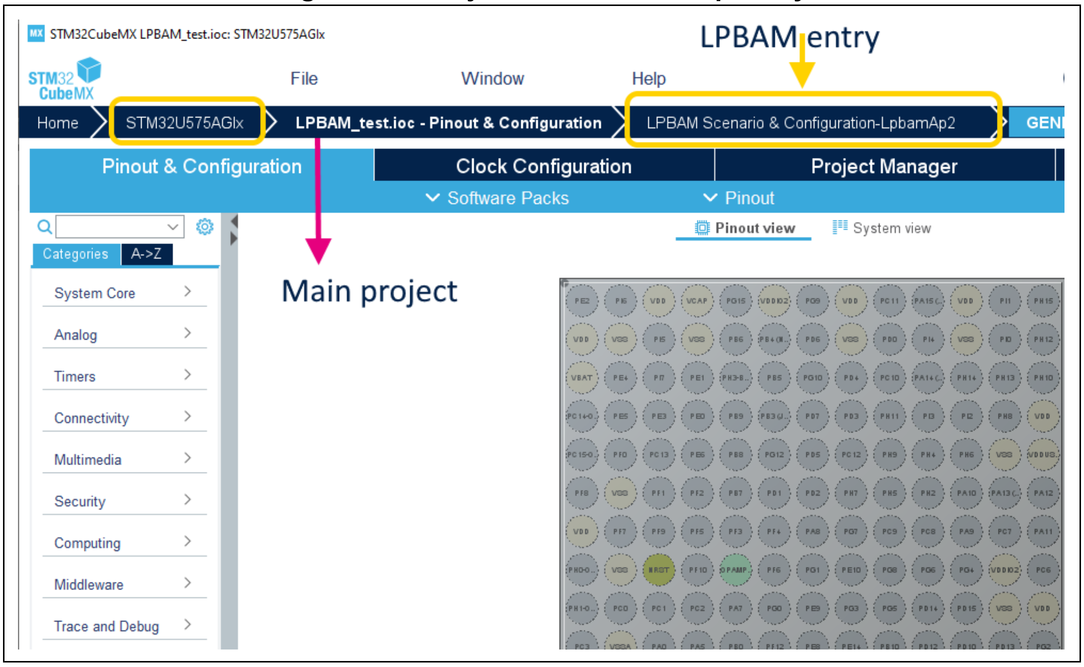


18.2 Creating an LPBAM project

18.2.1 LPBAM feature availability

When a project with LPBAM feature capability is opened, a dedicated entry is shown in the user interface (see *Figure 639*). The feature is optional and when it is not used, it has no impact on the generated project.

Figure 639. Project with LPBAM capability



18.2.2 Describing an LPBAM project

Describing an LPBAM project in STM32CubeMX consists in describing the main project using STM32CubeMX main project page, and one or more LPBAM applications using the dedicated LPBAM Scenario & Configuration page.

Starting with STM32CubeMX 6.5:

- Create a project by selecting an MCU or board part number from the STM32U575/585 product line.
- Do not activate TrustZone for the project.
- Click “LPBAM Scenario & Configuration” ribbon to view LPBAM dedicated page.

The LPBAM context is highlighted with a pink border. You can switch back and forth between the main project configuration and the LPBAM Scenario & Configuration by clicking the corresponding ribbon.

Figure 640. LPBAM Scenario & Configuration view

18.2.3 Managing LPBAM applications in a project

When entering the LPBAM Scenario & Configuration view, you must first add an LPBAM application.

Adding, removing, renaming, and switching between LPBAM applications is done from the left panel under the LPBAM manager section.

To add the first LPBAM application, click “Add Application”:

- If the default name is kept, the application “LpbamApp1” is created.
- The first Queue “Queue1” of LpbamApp1 is created.
- The configuration views (LPBAM scenario, pinout & ip, clock) necessary to describe Lpbam App1 are available.

To add more queues, click “Add Queue”

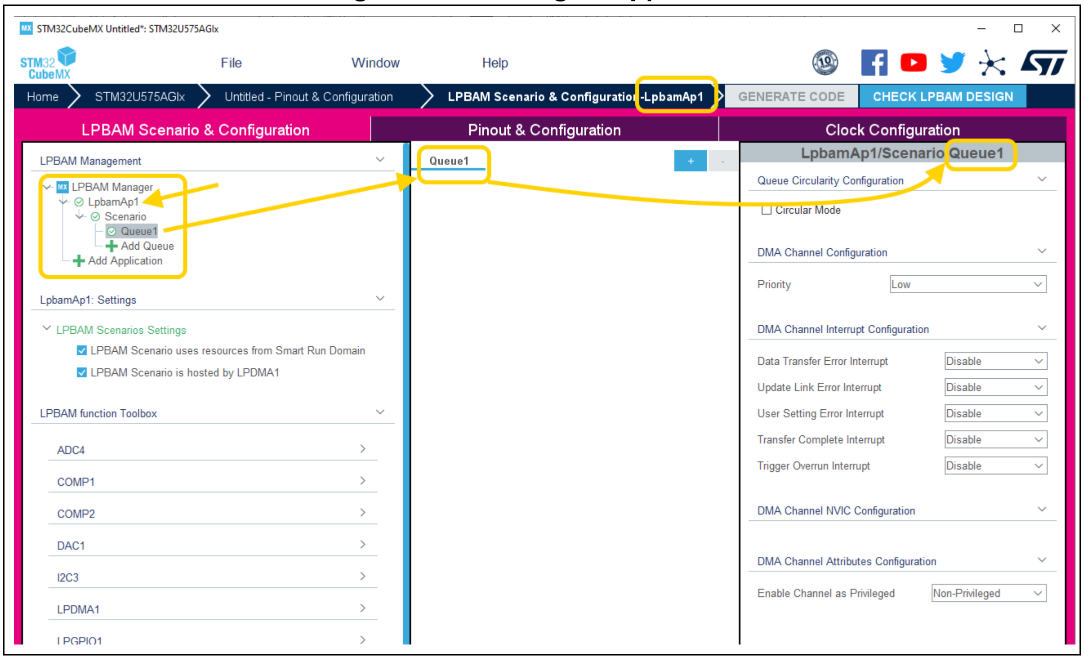
To delete an application (or a queue), right-click the application (or the queue) name and select “Delete”.

To rename an application (or a queue), right-click the application (or the queue) name and select “Rename”. Note that the application name is used in the generated project.

To switch between LPBAM applications, click the application name, this loads the LPBAM panel for the selected application.

To switch between queues in an LPBAM application, click the queue name: the middle and right panels are refreshed to display the selected queue and its configuration.

Figure 641. Adding an application



18.3 Describing an LPBAM application

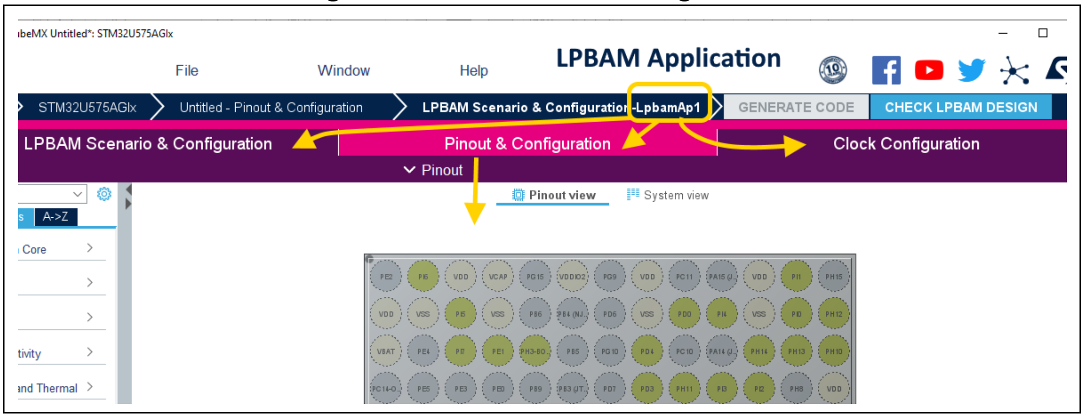
18.3.1 Overview (SoC & IPs configuration, runtime scenario)

Describing an LPBAM application consists in configuring the SoC and IPs, as it is done for a standard STM32CubeMX project, as well as describing the runtime part of the application.

SoC and IPs configuration

To configure IP and SOC in the context of an LPBAM application, use the Pinout & Configuration and Clock configuration provided with the LPBAM application.

Figure 642. SoC and IPs configuration

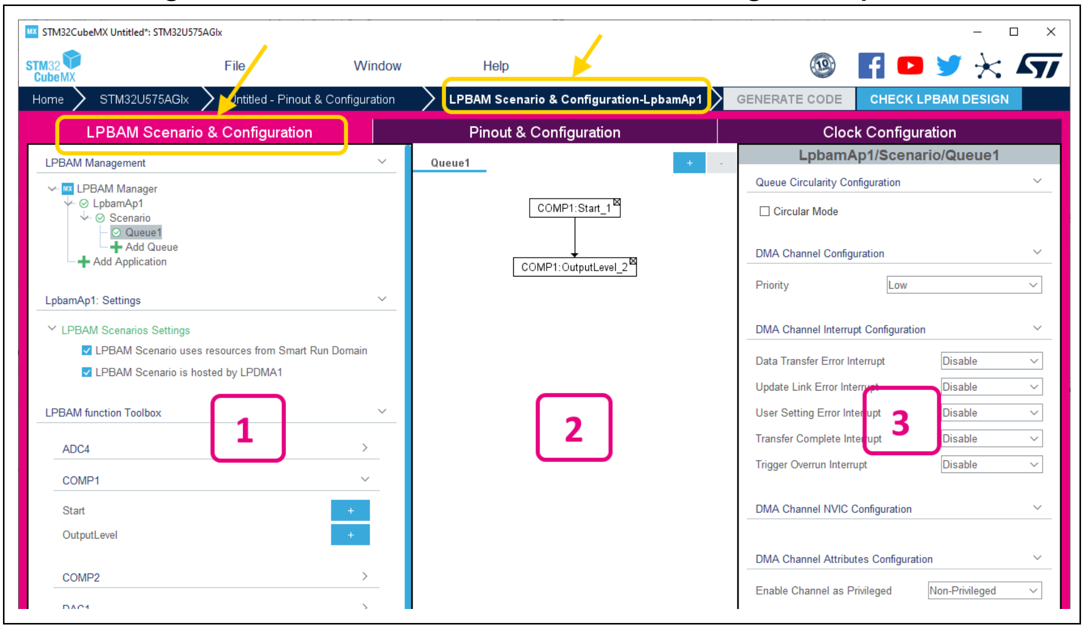


Runtime description (scenario)

With standard STM32CubeMX projects, the user must add the code to manage the runtime behavior of the main application based on STM32Cube HAL or LL driver APIs, such as HAL_COMP_Start, HAL_TIM_Start, HAL_TIM_Stop.

For LPBAM applications, STM32CubeMX provides the LPBAM Scenario & Configuration panel to create the runtime description (scenario). As shown in *Figure 643*, this panel is divided in three parts.

Figure 643. LPBAM scenario: creation and configuration panels



Note: LPBAM applications use the LPBAM firmware APIs and consist of chained DMA transfers.

In the context of an LPBAM application, the first panel is used for:

- Managing queues for the application.
- Browsing and adding nodes to the queue currently selected in STM32CubeMX user interface.
- Application specific settings. These settings cannot be changed nor disabled when using LPBAM on STM32U5 series.

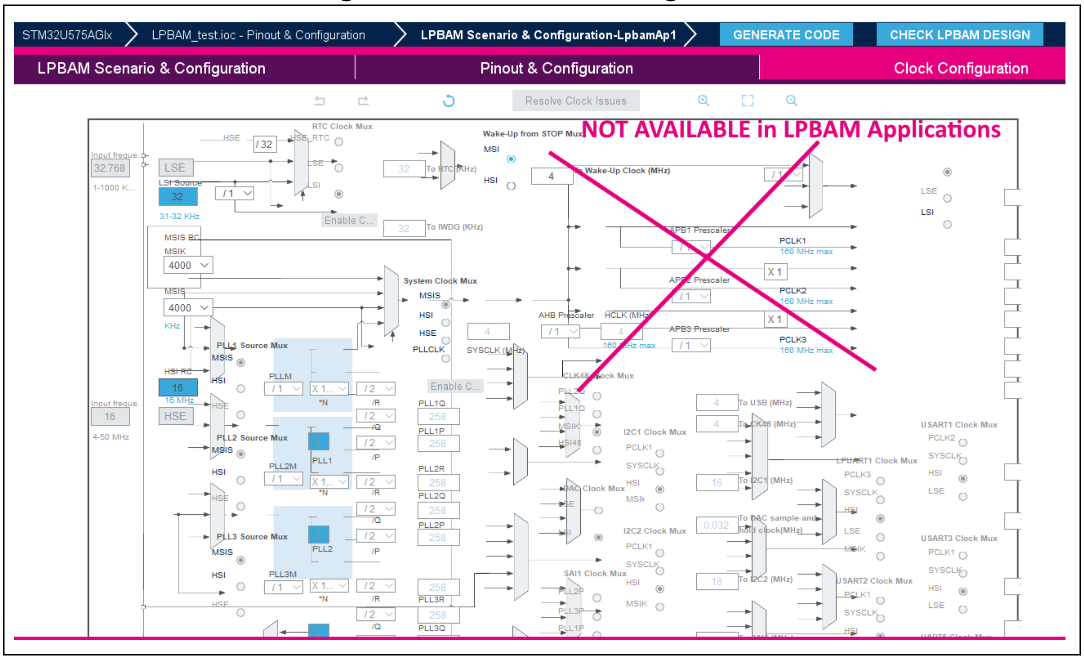
The second panel displays the diagram of the queue currently selected for one selected queue of the LPBAM application.

The third panel lets the user to configure either the queue (if the queue name is clicked), or a node (if the node is selected on the diagram).

18.3.2 SoC& IPs: configuring the clock

The LPBAM subsystem is functional down to STOP2 mode and supports only IPs on the Smart run domain. Consequently, in the LPBAM context, only a subset of the clock tree can be configured. Refer to *Section 4.10* for details on how to configure a clock tree in STM32CubeMX.

Figure 644. Clock tree configuration



18.3.3 SoC & IPs: configuring the IPs

Only IPs of the Smart run domain are available in the LPBAM context.

In the LPBAM context, most IPs show the same configuration possibilities as the main project. However, for some IPs, some additional configuration is needed. For example, when an IP internal interrupt can be used in the LPBAM context, a dedicated configuration Tab is shown.

Figure 645. Available IPs

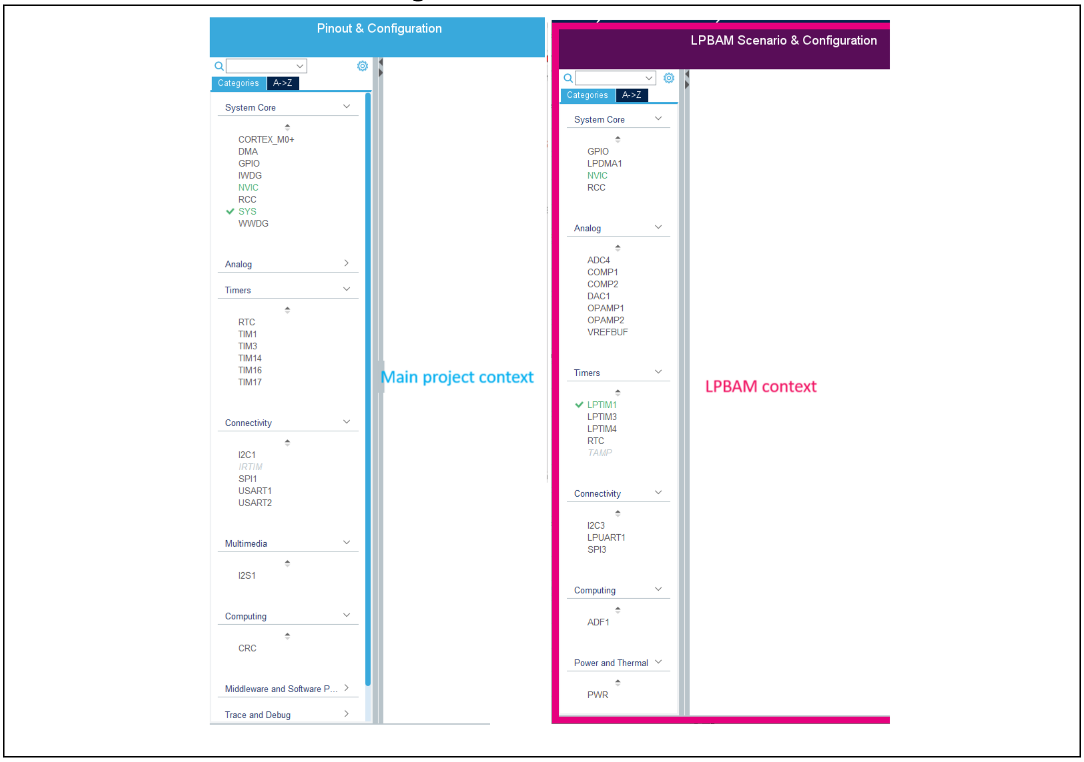
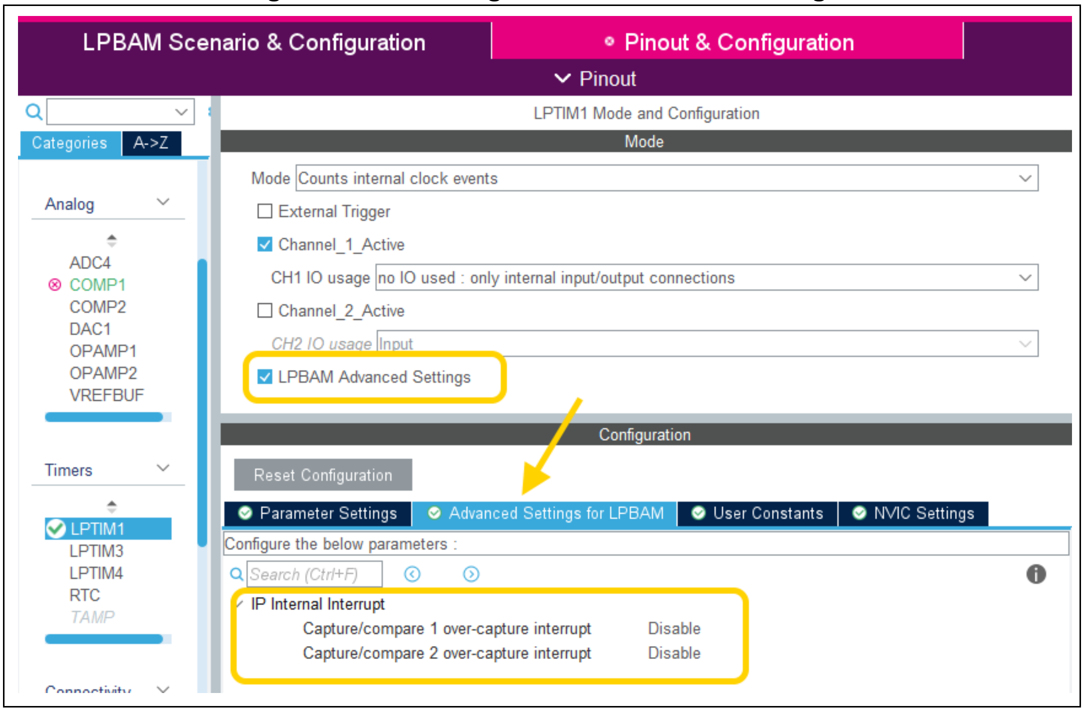


Figure 646. IP configuration: advanced settings



All IPs used at runtime by the LPBAM must be configured in the Pinout & Configuration view. Their configuration must be coherent with the LPBAM scenario.

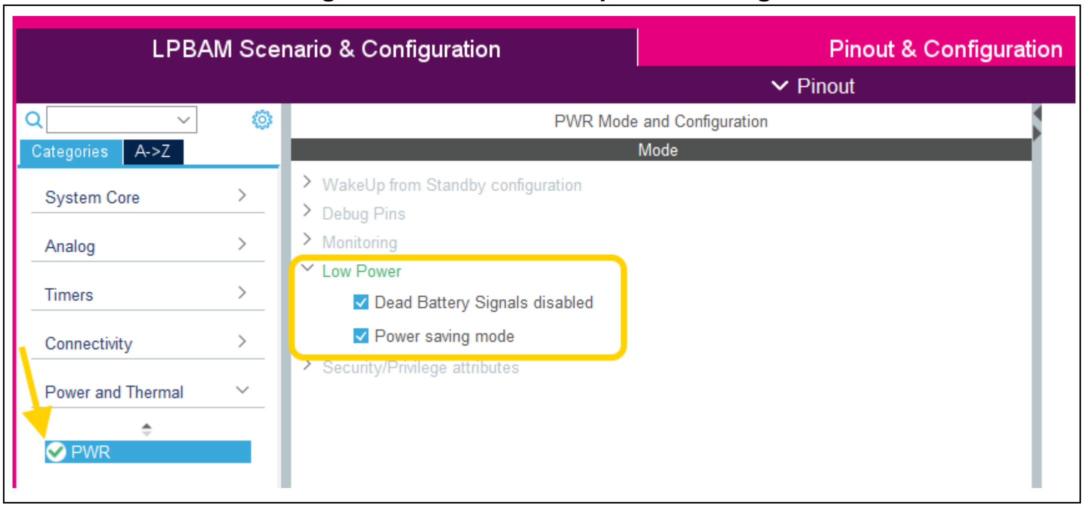
Clicking “Check LPBAM Design” on the upper right corner of the user interface returns, for each IP used but not configured in an LPBAM application, a warning in the output window.

Warning: “Check LPBAM Design” checks only that the IPs are configured in the “Pinout & Configuration”, it does not check whether the HAL configuration is coherent with the LPBAM APIs used in the scenario.

18.3.4 SoC & IPs: configuring low power settings

Starting with STM32CubeMX6.5, users can configure low power settings for their project. These settings (to be found under the PWR IP) are very important to minimize the power consumption of an LPBAM application.

Figure 647. LPBAM low power settings



18.3.5 LPBAM scenario: managing queues

An LPBAM scenario consists of one or more queues, each with one or more nodes. The center panel describes the scenario of the LPBAM application: click the queue name to display its diagram in the center panel and its configuration in the right panel. The name of the selected queue is underlined in blue.

To add more queues, click the “+” button in that panel, or click “Add queues” from the LPBAM management section in the left panel:

- The maximum number of queues is four on STM32U5 series, limited by the number of LPDMA1 channels.
- Adding an LPBAM application to the project automatically creates one empty queue for that application.

Warning: For LPBAM applications with multiple queues, STM32CubeMX does not manage the runtime synchronization between queues. It is the user’s

responsibility when assembling its final application to “start” the different queues at runtime.

The “LPBAM Management” section allows to remove and rename queues:

- To delete a queue, right-click the queue name and select “Delete”.
- To rename a queue, right-click the queue name and select “Rename”.
- To switch between queues in an LPBAM application, click the queue name: the middle and right panels are refreshed to display the selected queue and its configuration.

18.3.6 Queue description: managing nodes

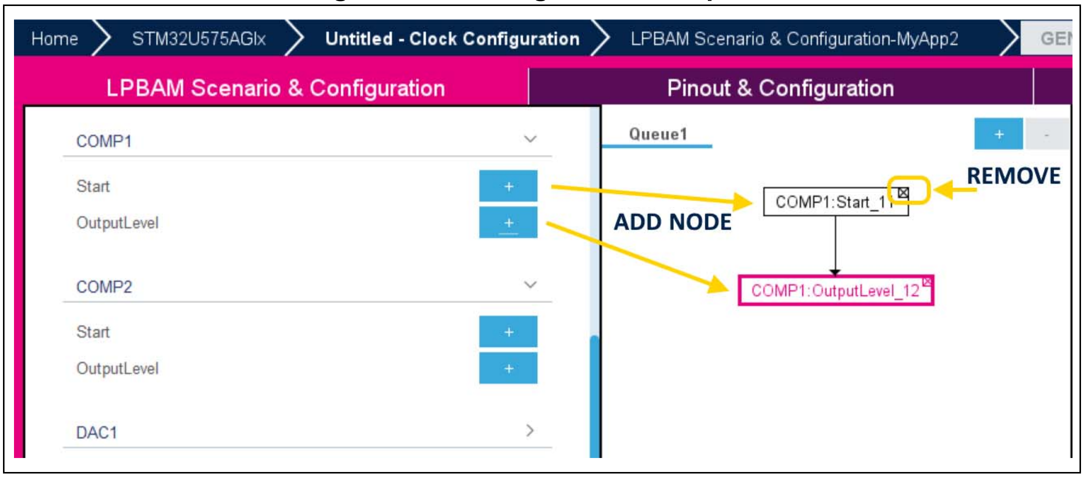
A queue description consists of a sequence of functional nodes on a timeline: the sequence is displayed as a diagram in the central panel and the queue configuration in the right panel.

To add nodes to a queue:

- Click the name of the queue to be updated.
- Use the “LPBAM function Toolbox”, in the left panel to browse the list of IPs and functions (LPBAM firmware APIs) that can be used to create nodes.
- Click the IP name to expand and see the list of available functions.
- Click the “+” sign next to the function name to add the function as a node in the queue: the queue diagram in the center panel is updated accordingly.
- Example: on Queue1 of LpbamAp1, COMP1 is started, then data transfer on COMP1 Output is performed (see *Figure 648*).

To remove nodes from the diagram, click the cross on the node right-end-upper corner.

Figure 648. Adding nodes to a queue

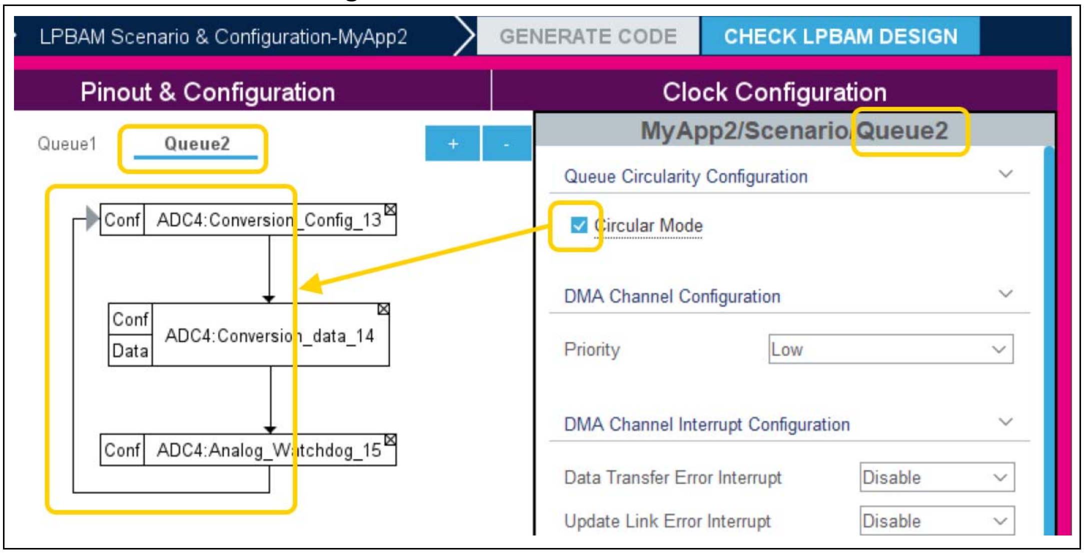


18.3.7 Queue description: configuring the queue in circular mode

STM32CubeMX offers the possibility to design circular queues:

- Select the queue to be configured by clicking the queue name in the center panel: the queue configuration is displayed in the right panel.
- Click the Circular mode checkbox to configure the queue in circular mode: by default, the queue loops back to the first node (see *Figure 649*).
- To loop back to a different node, click the end of the arrow and drag it to the node of choice.
- To remove the loop, uncheck Circular mode.

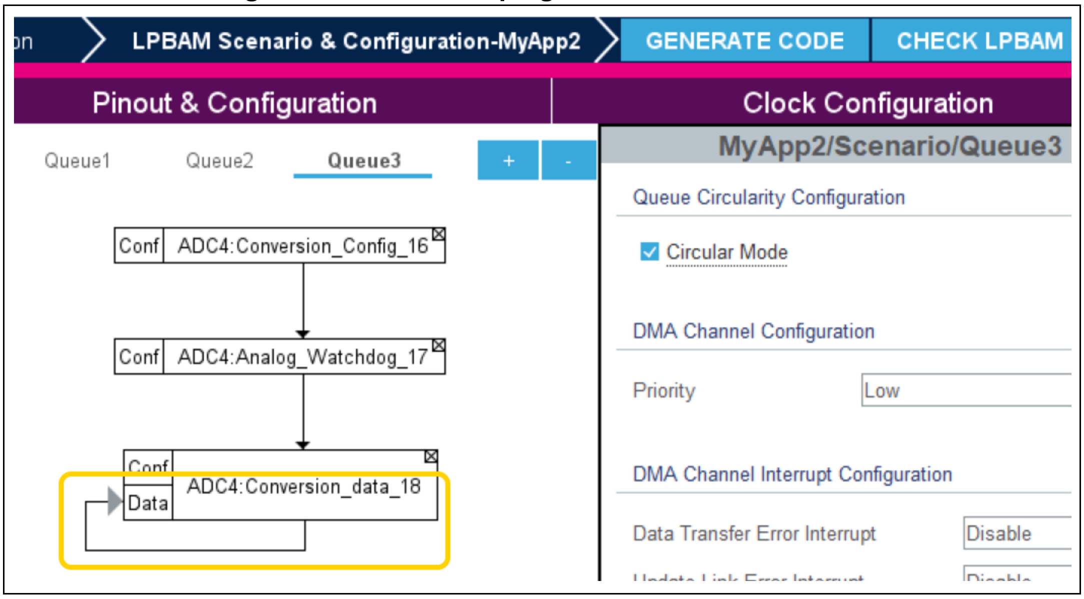
Figure 649. Queue in circular mode



Some functions first configure the IP, then manage the data transfer. In case of circular mode, the loop can be plugged on the configuration ("Conf") or on the data part ("Data") of the function.

An example is provided in *Figure 650*: when the queue is executed, the two first nodes and the configuration of the third node are executed once. whereas the data transfer is repeated as part of the loop.

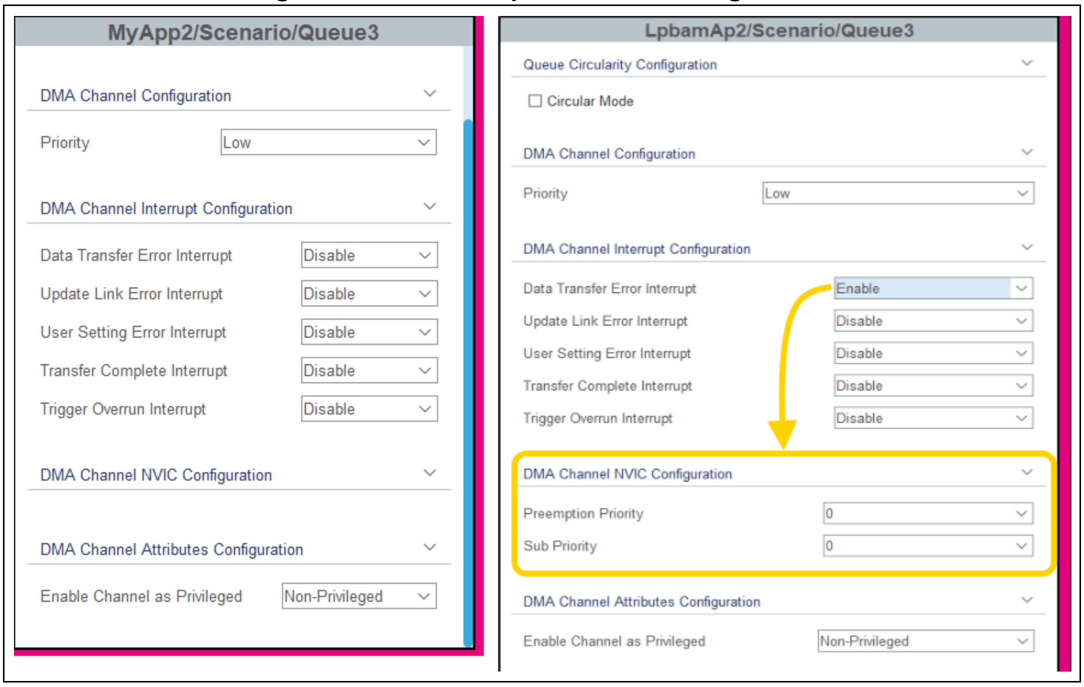
Figure 650. Queue looping back on IP data transfer



18.3.8 Queue description: configuring the DMA channel hosting the queue

The execution of an LPBAM queue consists of LPDMA chained transfers. The DMA hosting the queue execution must be configured as needed by the application (see *Figure 651*).

Figure 651. LPBAM queue: DMA configuration



Basic configuration

Select the queue to be configured by clicking the queue name on the center panel, the configuration of the DMA channel hosting that queue is shown in the right panel.

Note that some settings usually available for configuring a DMA channel are not provided in the user interface, as they are directly managed either by STM32CubeMX or by the LPBAM driver.

DMA channel NVIC configuration

NVIC settings are available only if one DMA channel interrupt is enabled (see right panel in *Figure 651*). The preemption priority and sub priority ranges in the LPBAM context depend on the NVIC priority group set for the whole project (the main project with the LPBAM applications).

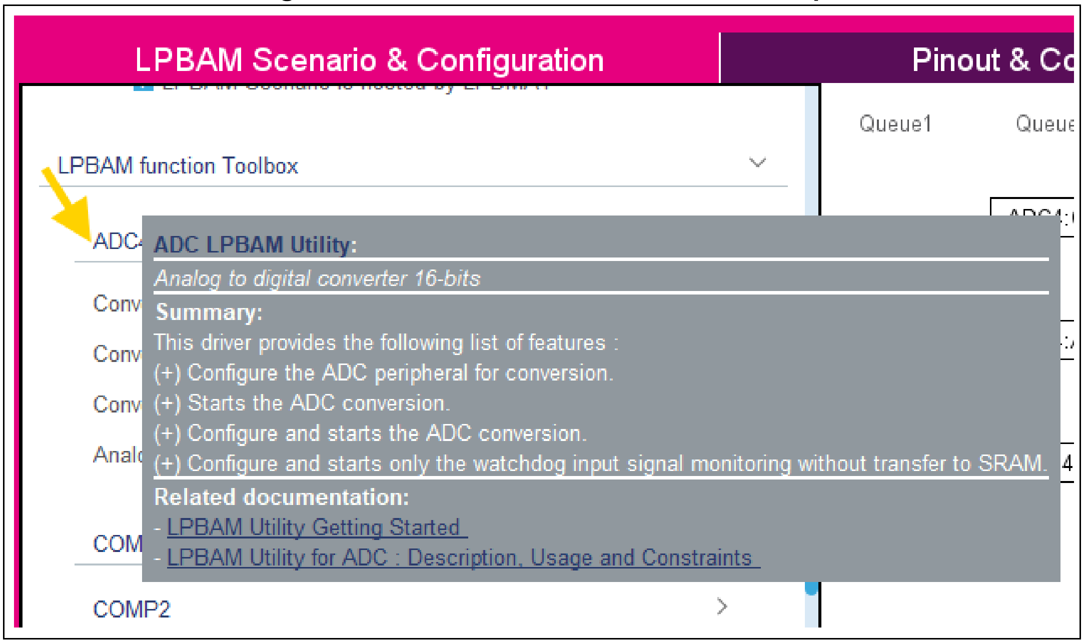
Warning: Always check preemption and sub-priorities in the LPBAM context after changing the NVIC priority group from the main project Pinout & Configuration view.

18.3.9 Node description: accessing contextual help and documentation

STM32CubeMX provides contextual help and link to reference documentation on LPBAM functions to guide the user during the function selection process:

- From the “LPBAM function Toolbox” in the left panel, hover the mouse on an IP name to show the contextual help with links to reference documentation (see *Figure 652*).
- It is recommended to read carefully the LPBAM global documentation and the IP “Description, Usage and Constraint” to learn how to assemble nodes in a queue, several queues, what can be done and what cannot be done. Some restrictions apply and are due to the LPBAM mechanism. They are not coming from the IP itself or from HAL constraints.

Figure 652. LPBAM functions contextual help

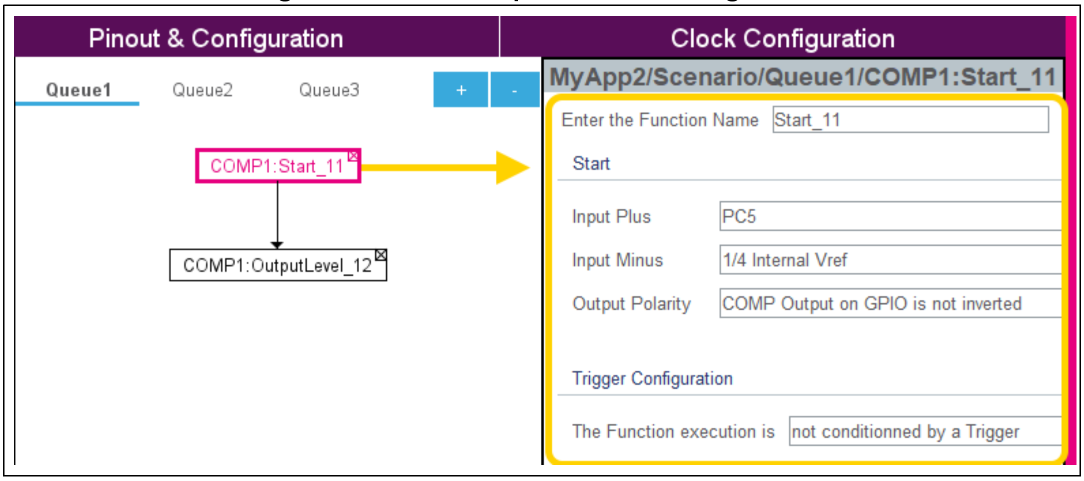


18.3.10 Node description: configuring node parameters

Once a function is chosen from the "LPBAM Function Toolbox" and added to a queue, it can be configured. In the center panel, click on a node to select it: the function is highlighted in pink, and its configuration is shown in the right panel (see *Figure 653*).

The example shows the "Start" parameters of the LPBAM COMP1_Start function. The HAL driver uses the same parameter names to configure a COMP IP. As mentioned before, the LPBAM firmware is not a HAL driver. However, the IP being unique, the LPBAM driver has been designed so that the IP parameters use, whenever possible, the same naming as found in the HAL driver.

Figure 653. LPBAM queue node configuration



Warning: LPBAM IP functions access IP hardware resources, to be properly configured in the "Pinout & Configuration" view.

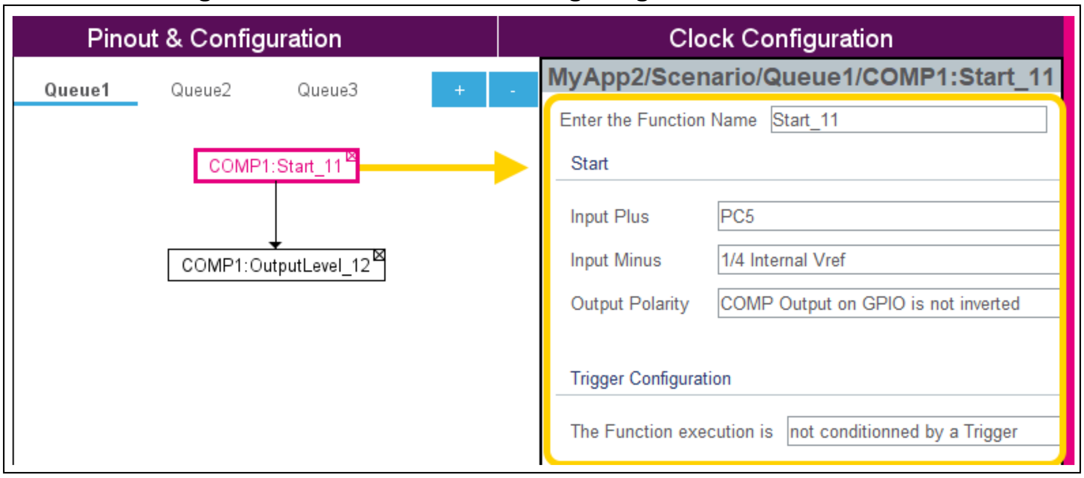
When a parameter is set to a hardware resource such as a GPIO, the resource must be configured in the Pinout & Configuration view.

In the example shown in *Figure 653*, the COMP "Input Plus" is set to PC5. If PC5 is not configured in the "Pinout & configuration" view, the generated LPBAM application can gets a "null signal" on Input Plus, and will be not functional.

To fix this issue:

- Go to the Pinout&Configuration view
- Search PC5 using the search field
- Right-click the PC5 pin and select COMP_Inp (see *Figure 654*)

Figure 654. LPBAM node: configuring hardware resources



Another example can be made using a timer to generate a PWM signal. The HAL driver requires a timer channel to be configured as output. Same applies when using the LPBAM firmware.

Note: All constraints concerning the initial configuration of the IP are mentioned in the LPBAM firmware documentation. Use STM32CubeMX "LPBAM Design check" mechanism (see dedicated section) to detect missing configurations.

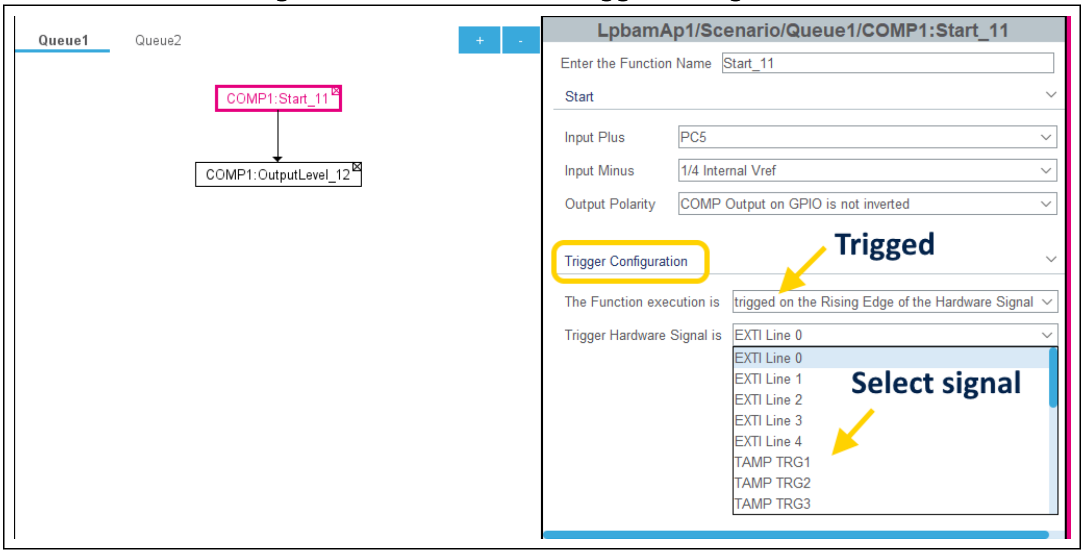
18.3.11 Node description: configuring a trigger

For all IPs and functions, with the LPBAM firmware it is possible to use a hardware signal to trigger a node. STM32CubeMX allows to configure such trigger from the node configuration panel. By default, the node execution is not triggered. When trigger is enabled, all possible trigger signals are listed.

Warning: It is the user responsibility to properly configure the triggers. STM32CubeMX does not check for configuration errors.

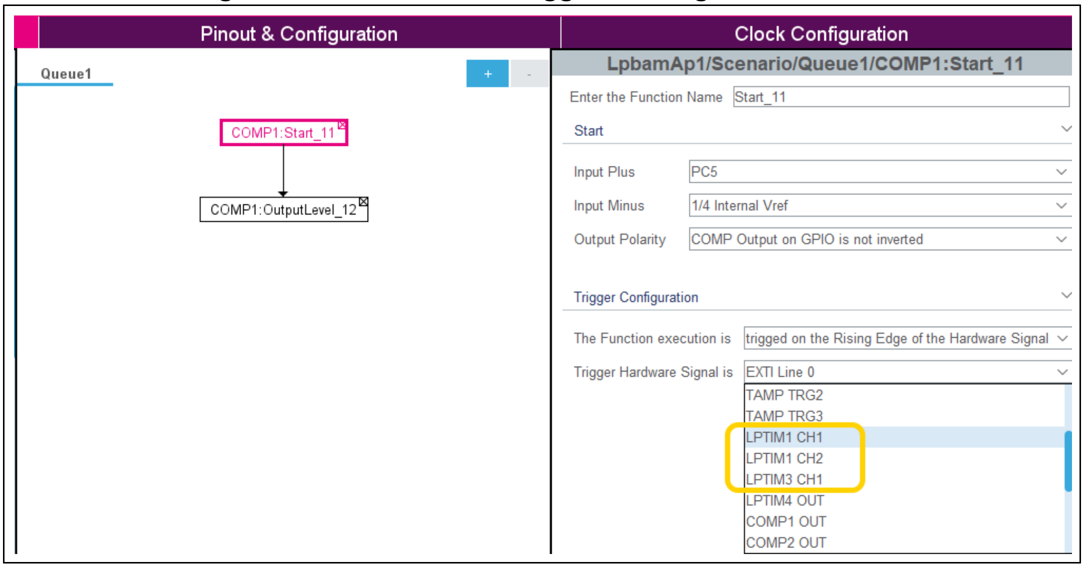
Taking the COMP function "Start" as an example (see *Figure 655*), choose the function execution to be triggered on the rising edge of hardware signal, for the example, then, select the hardware signal among the list of hardware signals proposed.

Figure 655. LPBAM node trigger configuration



If a node is a function managing LPTIM1_CH1, it is possible to select LPTIM1_CH1 as the trigger (see *Figure 656*).

Figure 656. LPBAM node triggered using timer channel

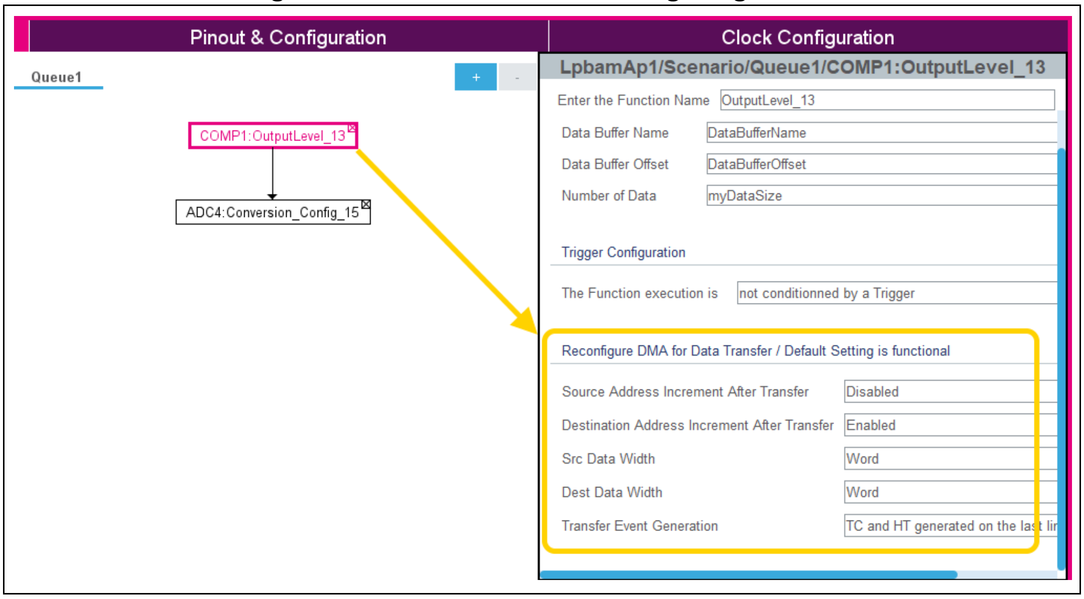


18.3.12 Node description: reconfiguring a DMA for data transfer

Nodes set to a function managing data transfers (all functions with associated data transfer and with a name not ending with _Config), come with a specific configuration section: “Reconfigure DMA for Data Transfer” (see *Figure 657*).

Each DMA data transfer is based on a specific configuration, including, among others, data size, buffer address, address increment. The DMA default settings are functional.

Figure 657. LPBAM node: reconfiguring a DMA

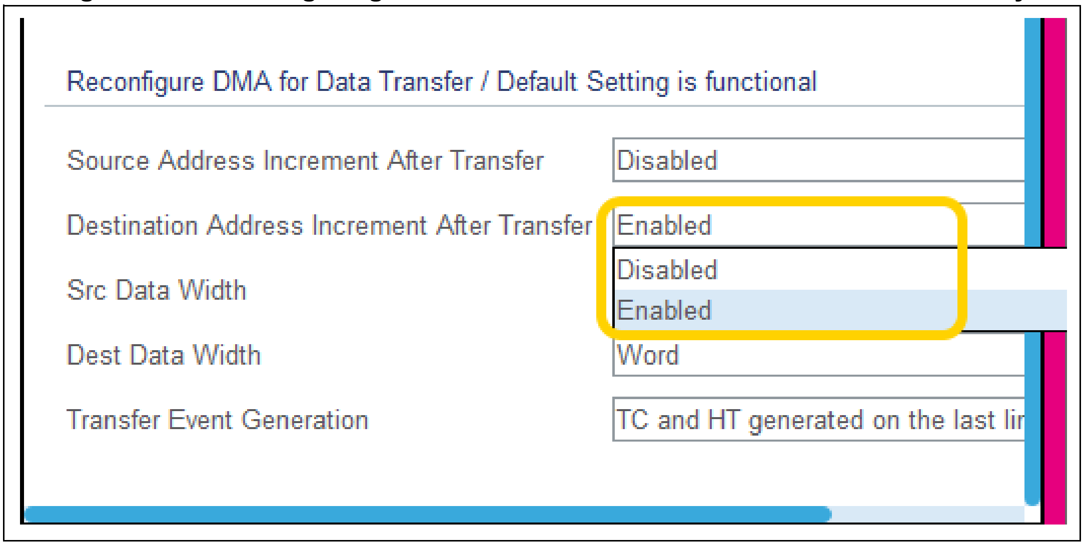


DMA settings can be changed, but they depend upon the IP and the function.

For example, for “COMP Output Level”:

- Data transferred are output data and are transferred from the register IP to the memory. The “Source Address” referring to the IP data register is not incremented: STM32CubeMX user interface shows that the “Source address increment after transfer” parameter cannot be enabled.
- Data transferred to memory can be saved at the same memory address, or in a Table: in this case, the “Destination Address increment after transfer” can be disabled or left enabled (see *Figure 657*).

Figure 658. Reconfiguring DMA for data transfer when destination is memory



18.4 Checking the LPBAM design

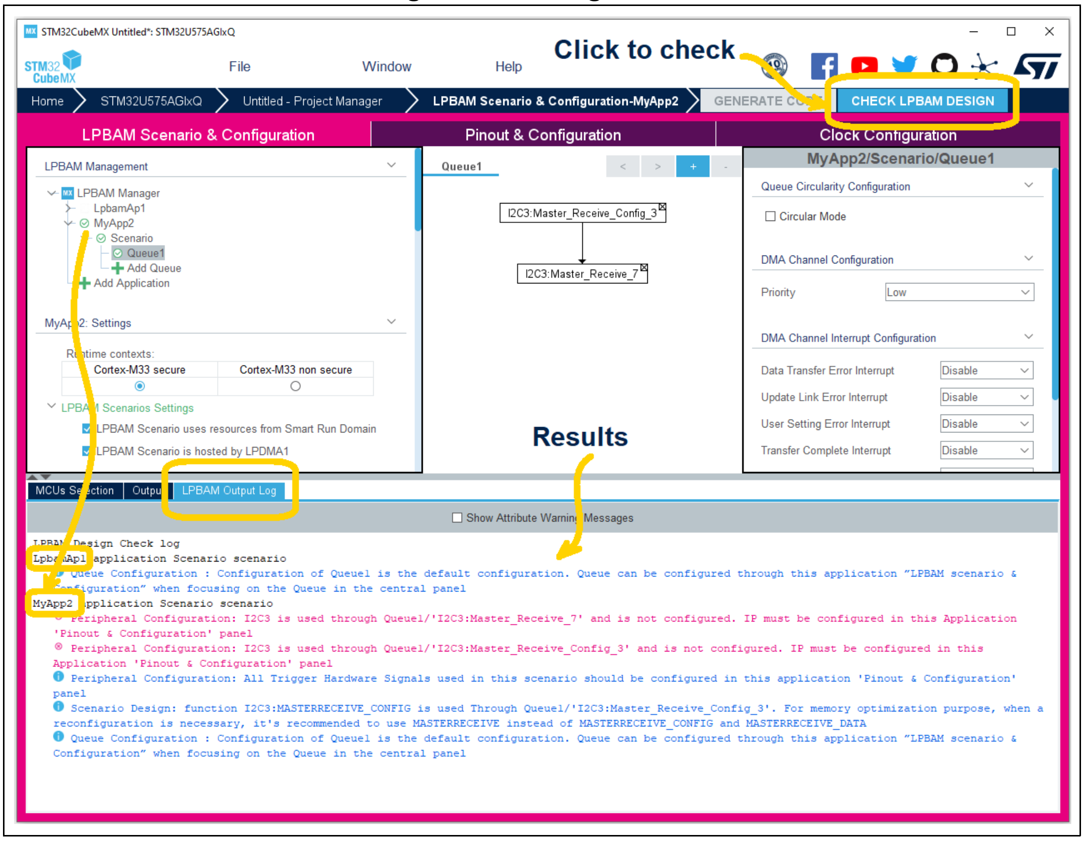
STM32CubeMX offers users with the possibility to check their LPBAM design for coherency and completeness, by detecting:

- Incoherences between the IP LPBAM function selected for a node and the corresponding IP configuration.
- Wrong queue designs (the sequence of nodes is invalid).

Click CHECK LPBAM DESIGN to check all LPBAM applications currently available in the project. Results appear in the LPBAM output log window (see *Figure 659*).

Note: Messages raised on the LPBAM design do not prevent users to generate the C code for their project. Supported type of messages are ERROR (in red), Warning (in orange), and Information (in blue).

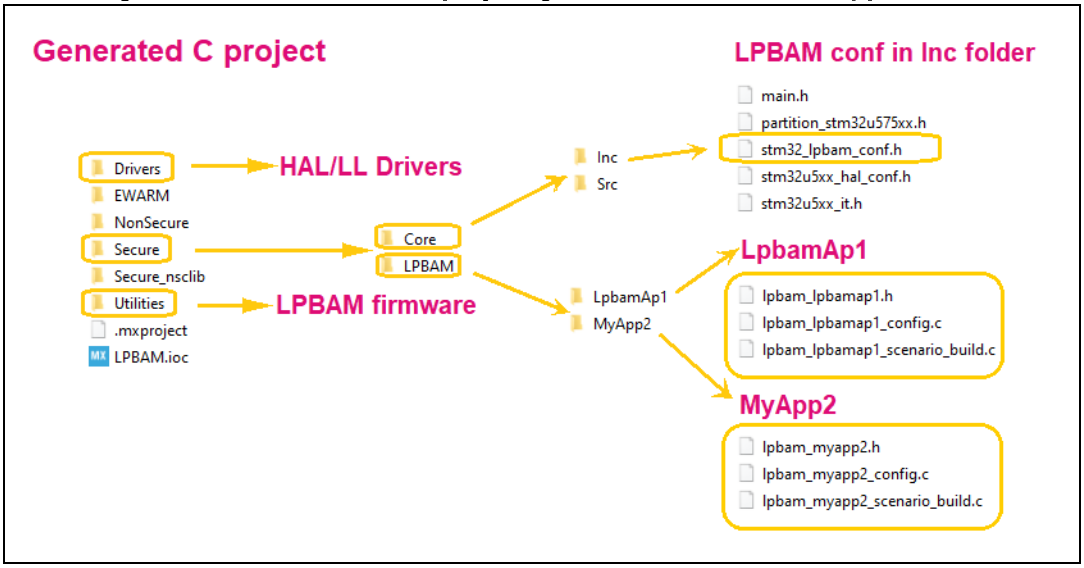
Figure 659. Design check



18.5 Generating a project with LPBAM applications

Click Generate Code from the main project view. As exemplified in *Figure 659*, the resulting project shows, in addition to the main project files and folders, the stm32_lpbam_conf.h file, a dedicated folder for the configuration code, and the utilities folder with the LPBAM utility firmware.

Figure 660. STM32CubeMX project generated with LPBAM applications



STM32CubeMX generates:

- In the Core/Inc folder, the stm32_lpbam_conf.h file that defines all the LPBAM modules enabled for the LPBAM applications, to be used by the LPBAM utility firmware.
- In the LPBAM folder, the code for the LPBAM applications and their scenarios. The lpbam_<application name>.h file provides the prototypes of the functions to call in the main project to initialize the application, build and initialize the scenario, link it with the DMA, start it, stop it, unlink it, and de-initialize it.

As an example, for the LpbamAp1 application, STM32CubeMX generates the following functions:

```
/* LpbamAp1 application initialization */
void MX_LpbamAp1_Init(void);

/* LpbamAp1 application - scenario initialization */
void MX_LpbamAp1_Scenario_Init(void);

/* LpbamAp1 application - scenario build */
void MX_LpbamAp1_Scenario_Build(void);

/* LpbamAp1 application - scenario link */
void MX_LpbamAp1_Scenario_Link(DMA_HandleTypeDef *hdma);

/* LpbamAp1 application - scenario start */
void MX_LpbamAp1_Scenario_Start(DMA_HandleTypeDef *hdma);
```

```
/* LpbamAp1 application - scenario stop */
void MX_LpbamAp1_Scenario_Stop(DMA_HandleTypeDef *hdma);

/* LpbamAp1 application - scenario unlink */
void MX_LpbamAp1_Scenario_UnLink(DMA_HandleTypeDef *hdma);

/* LpbamAp1 application - scenario de-initialization */
void MX_LpbamAp1_Scenario_DeInit(void);
```

18.6 LPBAM application for TrustZone activated projects

Starting with STM32CubeMX 6.6.0, users can create LPBAM applications for projects with TrustZone activated.

1. Access to MCU selector and select an STM32U575/585 device
2. Click Create a new project
3. Choose the option “with TrustZone activated”

STM32CubeMX standard project view

STM32CubeMX standard project view proposes security settings for peripherals (*Figure 661*) and the clock tree (*Figure 662*).

STM32CubeMX LPBAM view

In STM32CubeMX LPBAM Application configuration context, the peripherals and the clock tree do not come with dedicated security settings (see *Figure 663* and *Figure 664*). The choice of context, secure or nonsecure, is done at LPBAM application level (*Figure 665*).

Security settings coherency check

1. Click CHECK LPBAM DESIGN
2. Enable Show Attribute Warning Messages to see details about LPBAM security related configuration issues (see *Figure 666*)

Figure 661. STM32CubeMX project - Peripheral secure context assignment

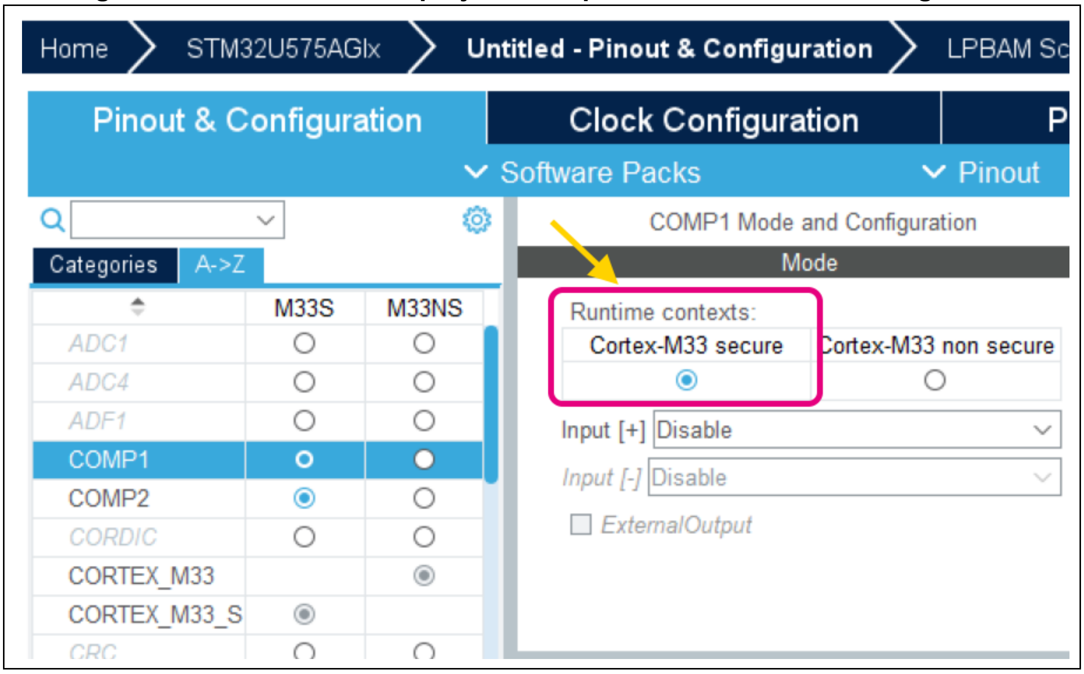


Figure 662. STM32CubeMX project - Clock source secure context assignment

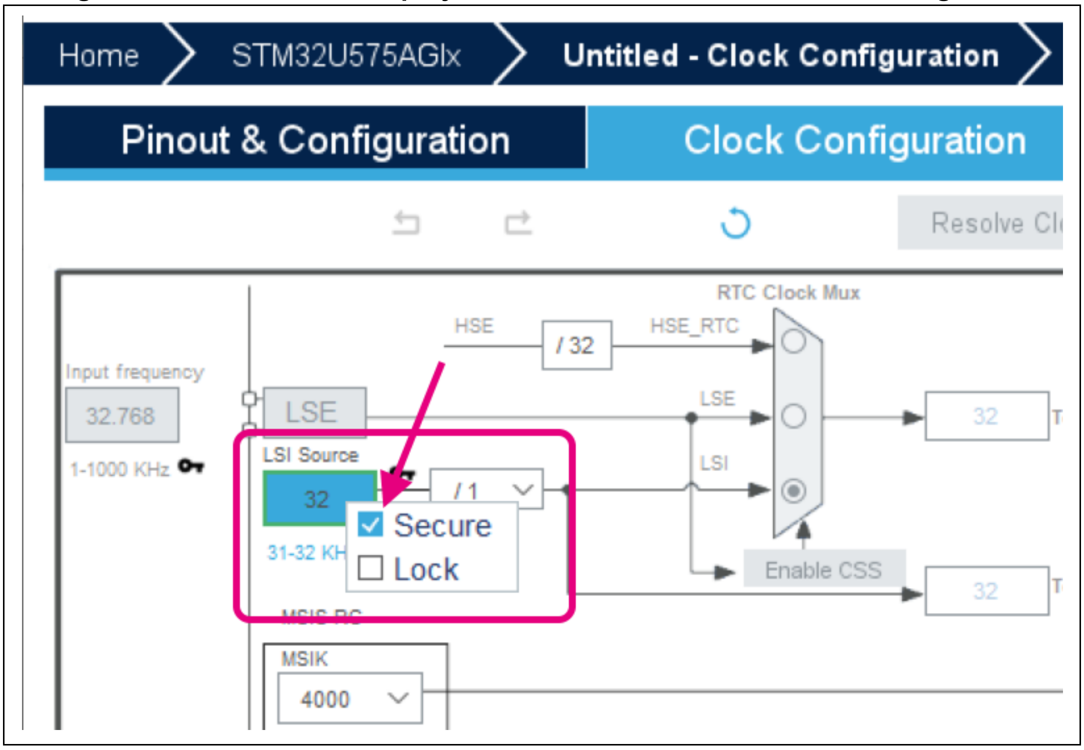


Figure 663. LPBAM project - Peripheral no context assignment

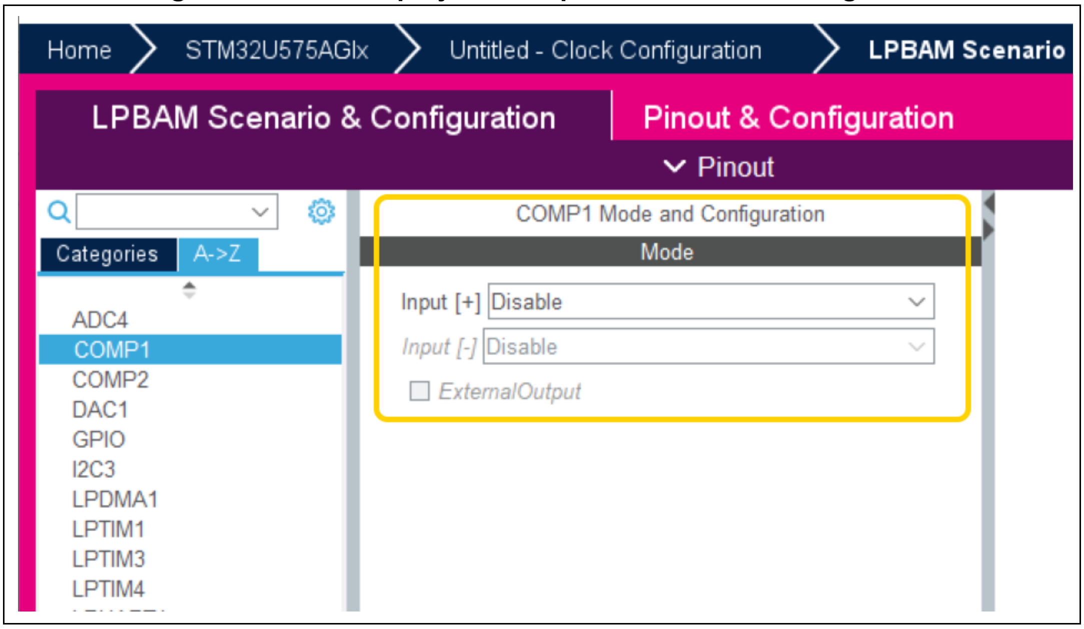


Figure 664. LPBAM application - Clock source no context assignment

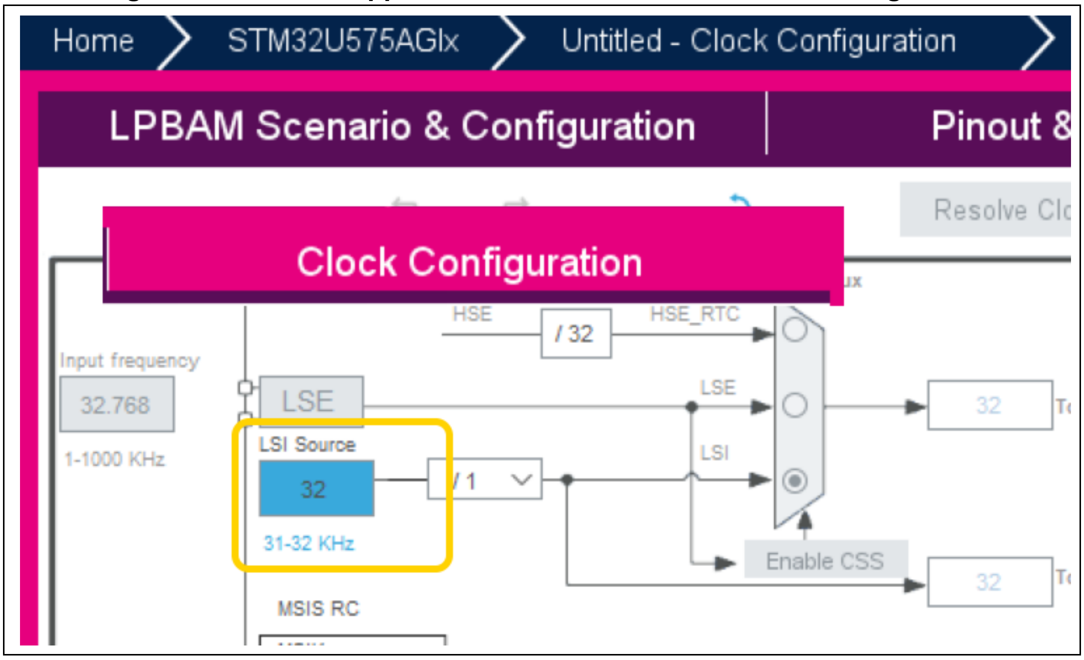


Figure 665. LPBAM application - Secure context assignment

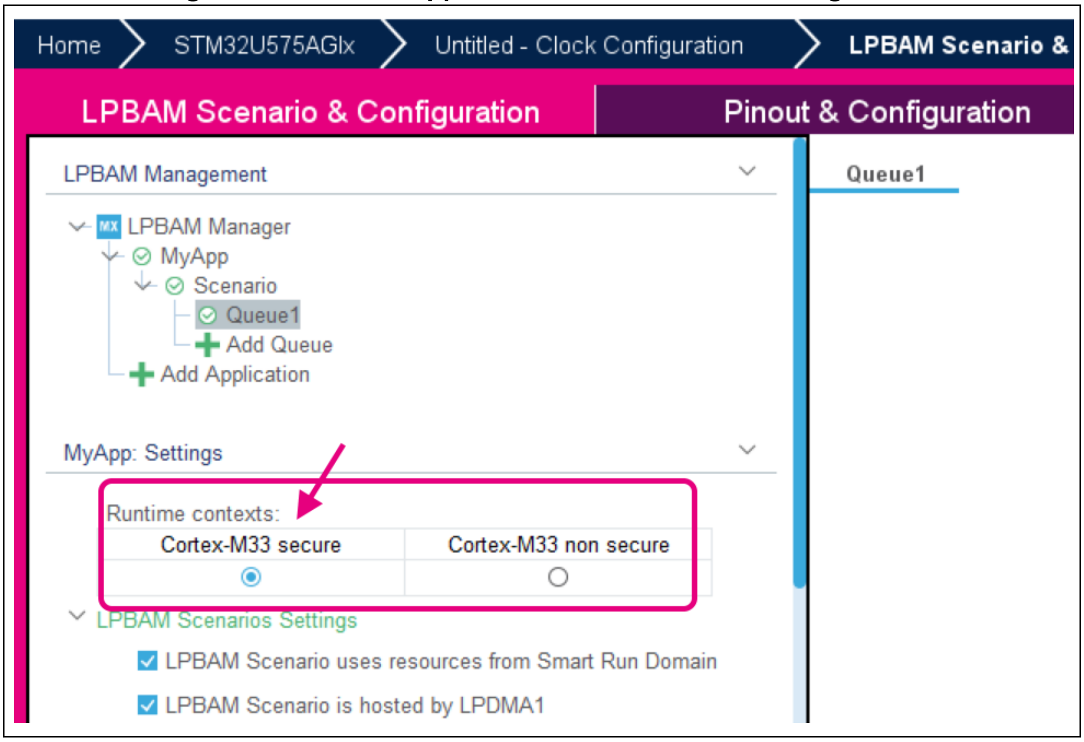
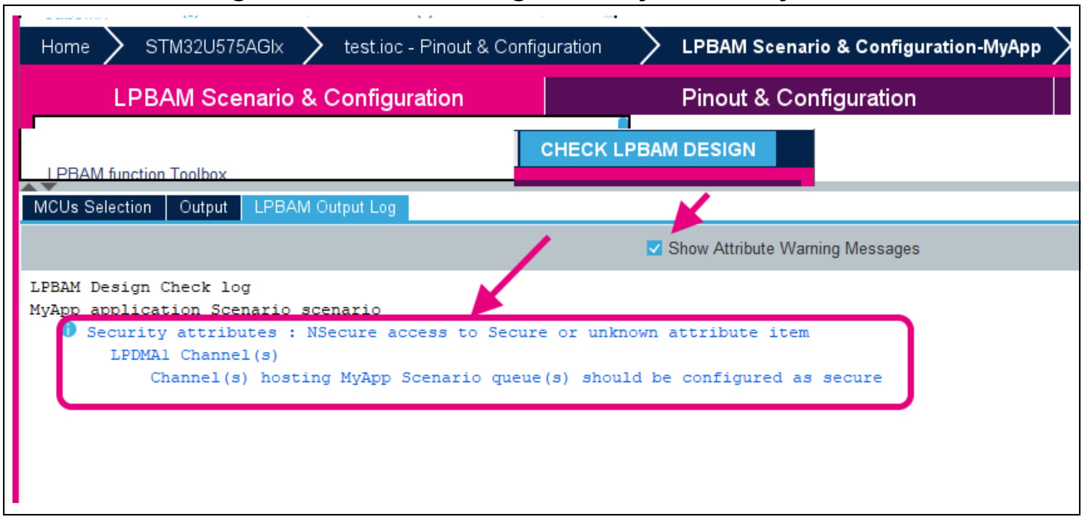


Figure 666. LPBAM design security coherency check



Appendix A STM32CubeMX pin assignment rules

The following pin assignment rules are implemented in STM32CubeMX:

- Rule 1: Block consistency
- Rule 2: Block inter-dependency
- Rule 3: One block = one peripheral mode
- Rule 4: Block remapping (only for STM32F10x)
- Rule 5: Function remapping
- Rule 6: Block shifting (only for STM32F10x)
- Rule 7: Setting or clearing a peripheral mode
- Rule 8: Mapping a function individually (if Keep Current Placement is unchecked)
- Rule 9: GPIO signals mapping

A.1 Block consistency

When setting a pin signal (provided there is no ambiguity about the corresponding peripheral mode), all the pins/signals required for this mode are mapped and pins are shown in green (otherwise the configured pin is shown in orange).

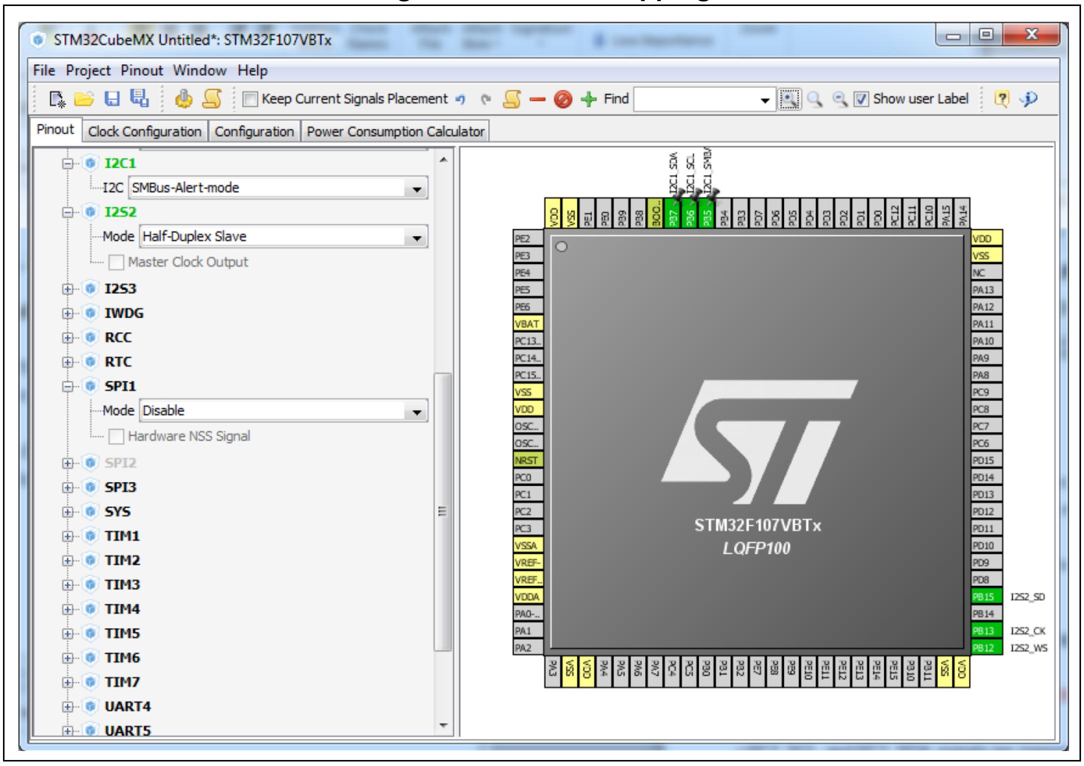
When clearing a pin signal, all the pins/signals required for this mode are unmapped simultaneously and the pins turn back to gray.

Example of block mapping with an STM32F107x MCU

If the user assigns I2C1_SMBA function to PB5, then STM32CubeMX configures pins and modes as follows:

- I2C1_SCL and I2C1_SDA signals are mapped to the PB6 and PB7 pins, respectively (see *Figure 667*).
- I2C1 peripheral mode is set to SMBus-Alert mode.

Figure 667. Block mapping

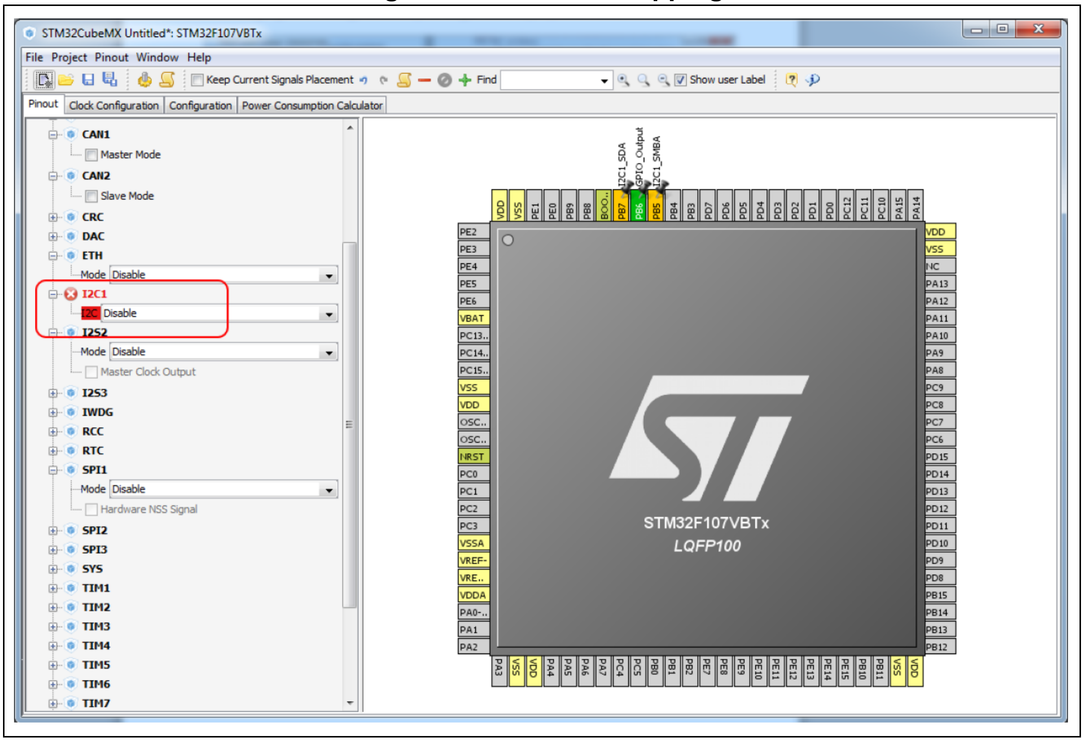


Example of block remapping with an STM32F107x MCU

If the user assigns GPIO_Output to PB6, STM32CubeMX automatically disables I2C1 SMBus-Alert peripheral mode from the peripheral tree view and updates the other I2C1 pins (PB5 and PB7) as follows:

- If they are unpinned, the pin configuration is reset (pin grayed out).
- If they are pinned, the peripheral signal assigned to the pins is kept and the pins are highlighted in orange since they no longer match a peripheral mode (see *Figure 668*).

Figure 668. Block remapping



For STM32CubeMX to find an alternative solution for the I2C peripheral mode, the user will need to unpin I2C1 pins and select the I2C1 mode from the peripheral tree view (see *Figure 669* and *Figure 670*).

Figure 669. Block remapping - Example 1

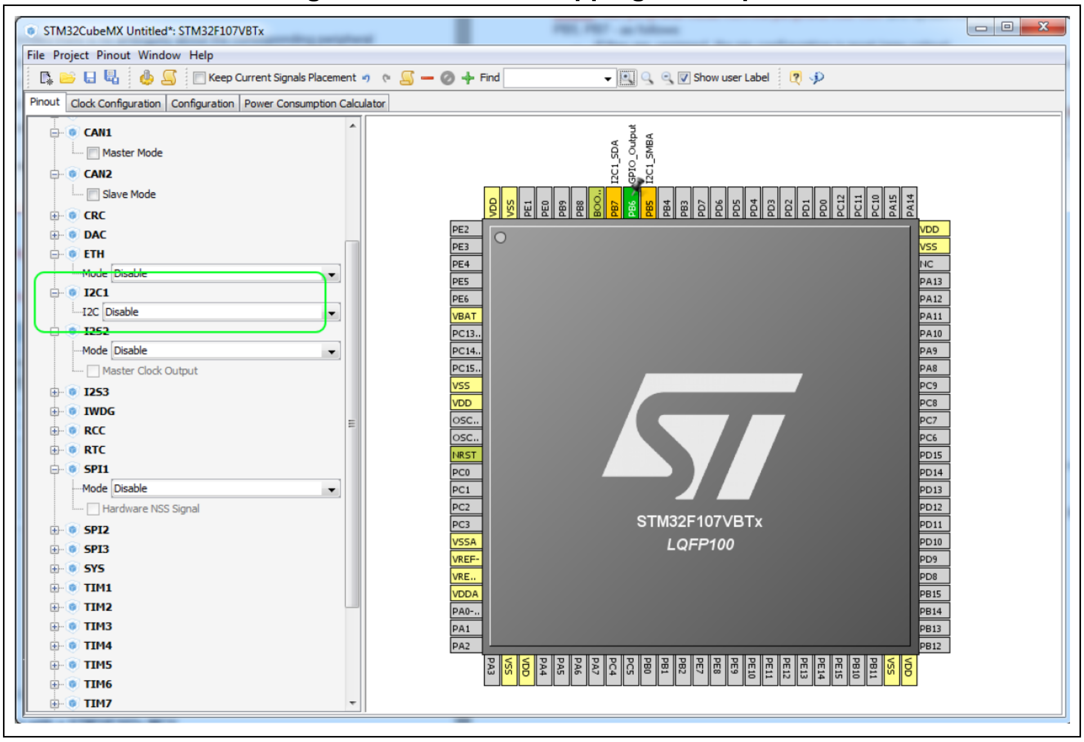
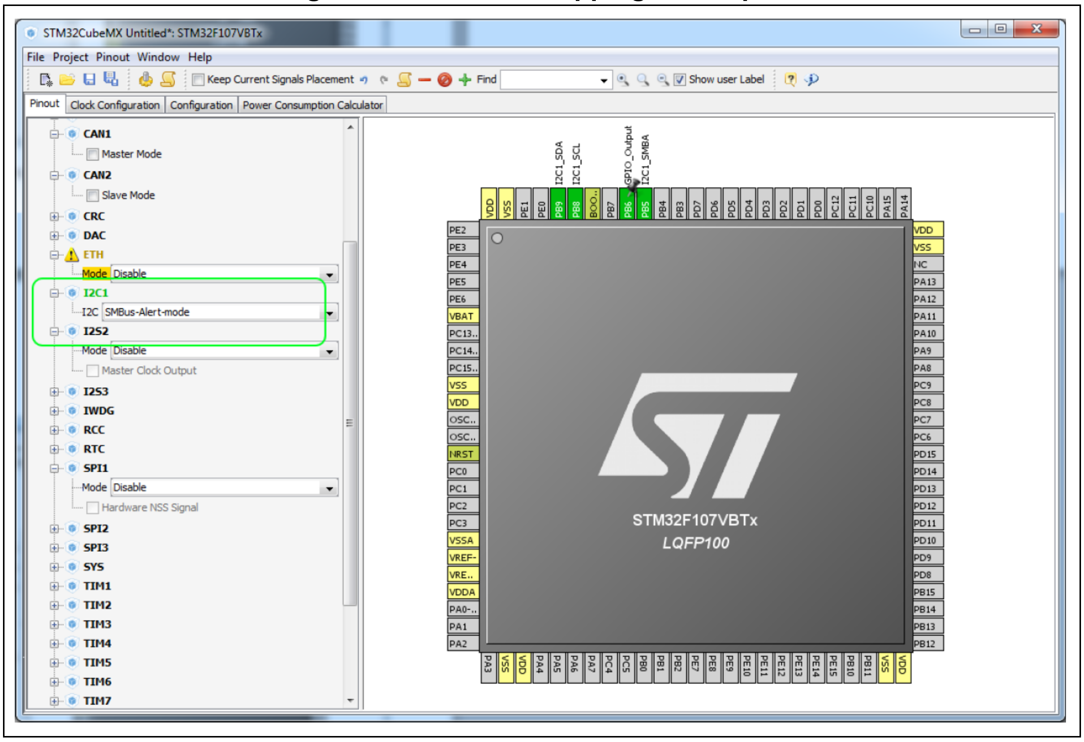


Figure 670. Block remapping - Example 2



A.2 Block inter-dependency

On the **Pinout** view, the same signal can appear as an alternate function for multiple pins. However it can be mapped only once.

As a consequence, for STM32F1 MCUs, two blocks of pins cannot be selected simultaneously for the same peripheral mode: when a block/signal from a block is selected, the alternate blocks are cleared.

Example of block remapping of SPI in full-duplex master mode with an STM32F107x MCU

If SPI1 full-duplex master mode is selected from the tree view, by default the corresponding SPI signals are assigned to PB3, PB4 and PB5 pins (see *Figure 671*).

If the user assigns to PA6 the SPI1_MISO function currently assigned to PB4, STM32CubeMX clears the PB4 pin from the SPI1_MISO function, as well as all the other pins configured for this block, and moves the corresponding SPI1 functions to the relevant pins in the same block as the PB4 pin (see *Figure 672*).

(by pressing CTRL and clicking PB4 to show PA6 alternate function in blue, then drag and drop the signal to pin PA6)

Figure 671. Block inter-dependency - SPI signals assigned to PB3/4/5

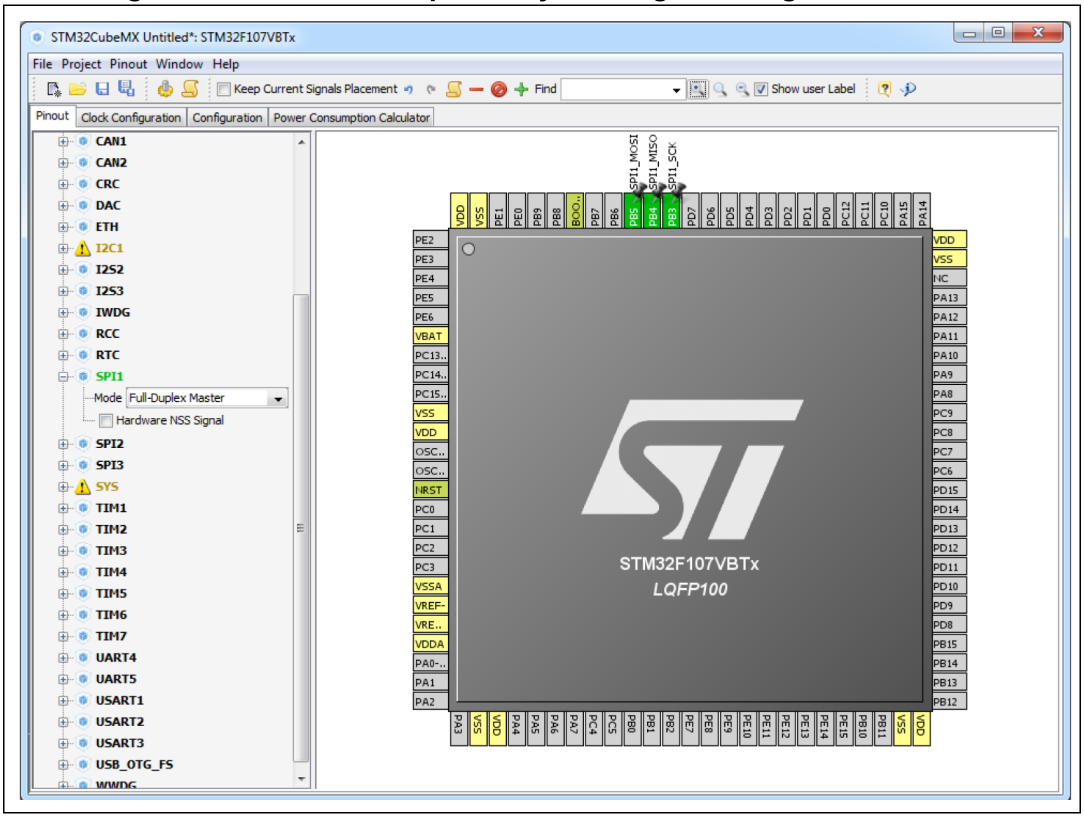
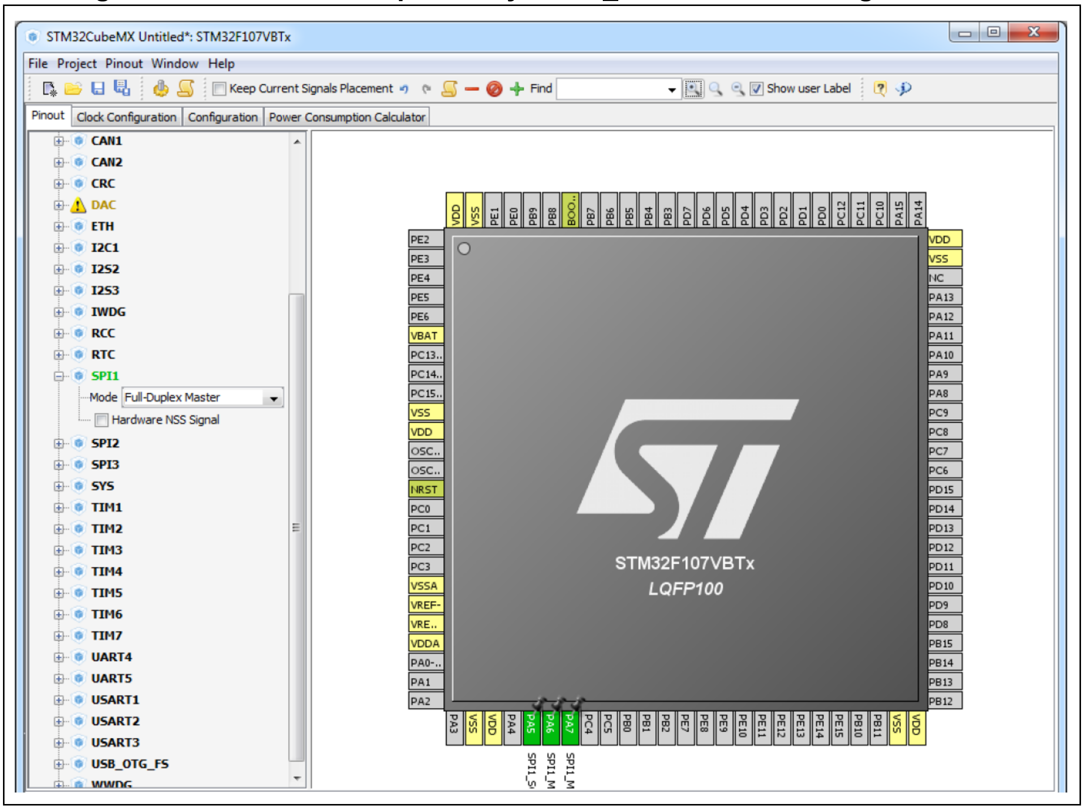


Figure 672. Block inter-dependency - SPI1_MISO function assigned to PA6



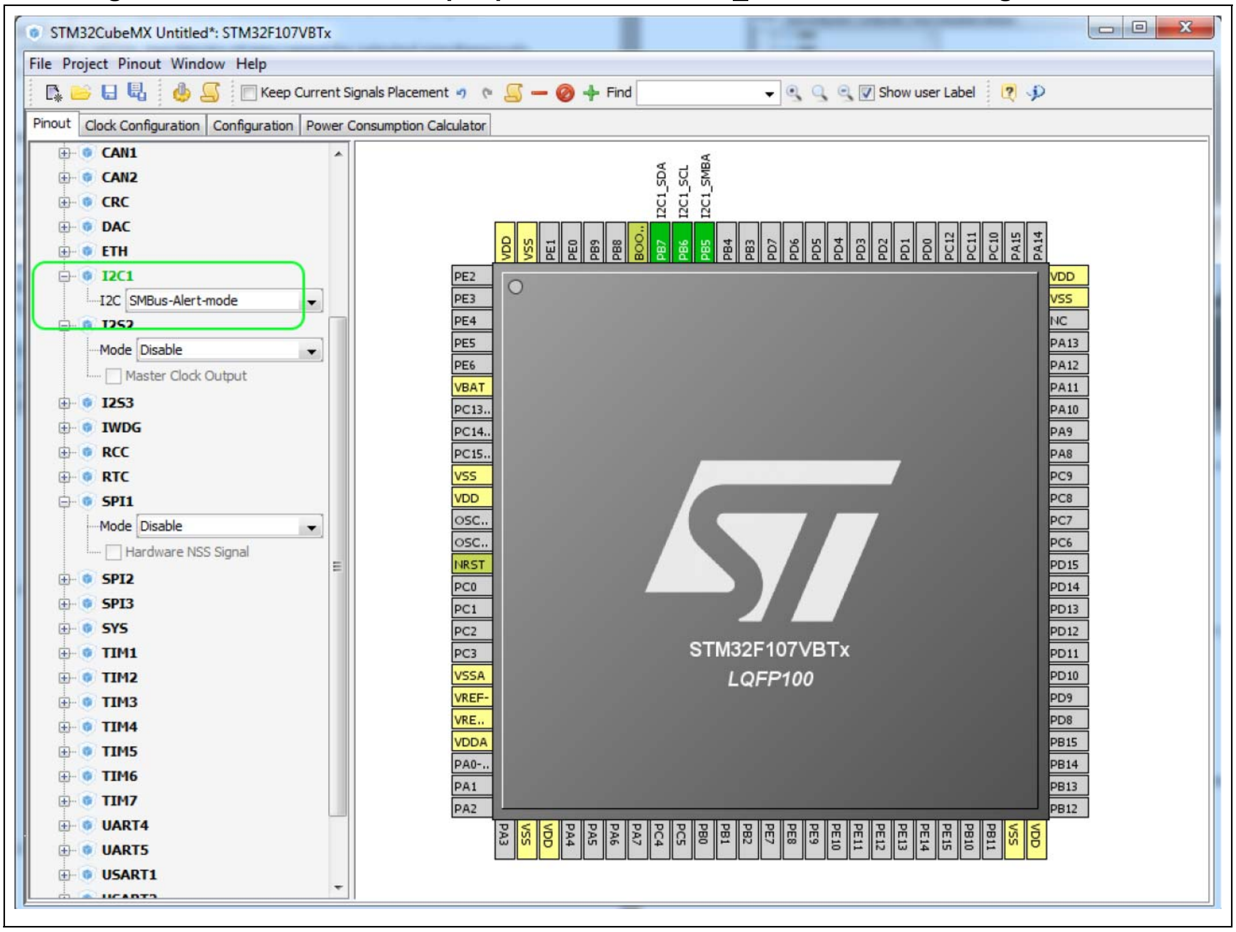
A.3 One block = one peripheral mode

When a block of pins is fully configured in the **Pinout** view (shown in green), the related peripheral mode is automatically set in the Peripherals tree.

Example of STM32F107x MCU

Assigning the I2C1_SMBA function to PB5 automatically configures I2C1 peripheral in SMBus-Alert mode (see Peripheral tree in *Figure 673*).

Figure 673. One block = one peripheral mode - I2C1_SMBA function assigned to PB5



A.4 Block remapping (STM32F10x only)

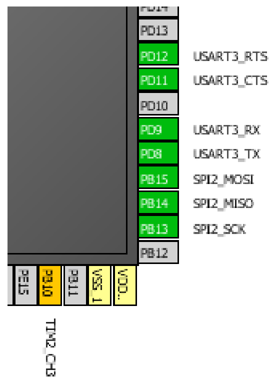
To configure a peripheral mode, STM32CubeMX selects a block of pins and assigns each mode signal to a pin in this block. In doing so, it looks for the first free block to which the mode can be mapped.

When setting a peripheral mode, if at least one pin in the default block is already used, STM32CubeMX tries to find an alternate block. If none can be found, it either selects the functions in a different sequence, or unchecks Keep Current Signals Placement, and remaps all the blocks to find a solution.

Example

STM32CubeMX remaps USART3 hardware-flow-control mode to the (PD8-PD9-PD11-PD12) block, because PB14 of USART3 default block is already allocated to the SPI2_MISO function (see *Figure 674*).

Figure 674. Block remapping - Example 2



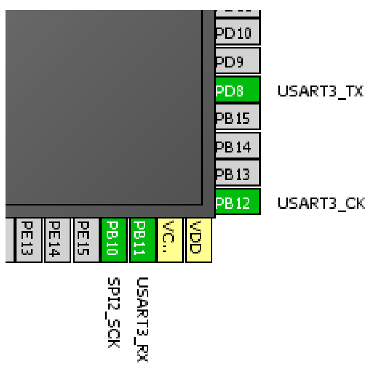
A.5 Function remapping

To configure a peripheral mode, STM32CubeMX assigns each signal of the mode to a pin. In doing so, it will look for the first free pin the signal can be mapped to.

Example using STM32F415x

When configuring USART3 for the Synchronous mode, STM32CubeMX discovered that the default PB10 pin for USART3_TX signal was already used by SPI. It thus remapped it to PD8 (see *Figure 675*).

Figure 675. Function remapping example



A.6 Block shifting (only for STM32F10x and when “Keep Current Signals placement” is unchecked)

If a block cannot be mapped and there are no free alternate solutions, STM32CubeMX tries to free the pins by remapping all the peripheral modes impacted by the shared pin.

Example

With the Keep current signal placement enabled, if USART3 synchronous mode is set first, the Asynchronous default block (PB10-PB11) is mapped and Ethernet becomes unavailable (shown in red) (see *Figure 676*).

Unchecking ☐ Keep Current Signals Placement allows STM32CubeMX shifting blocks around and freeing a block for the Ethernet MII mode. (see *Figure 677*).

Figure 676. Block shifting not applied

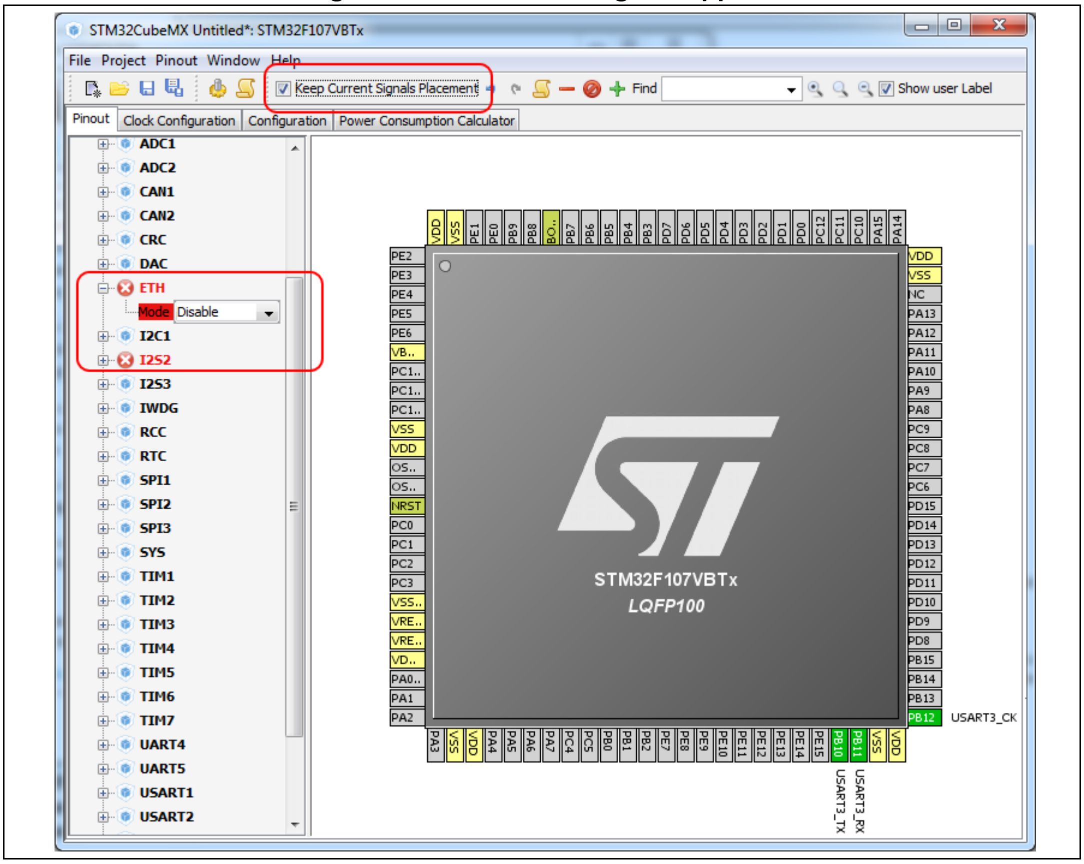
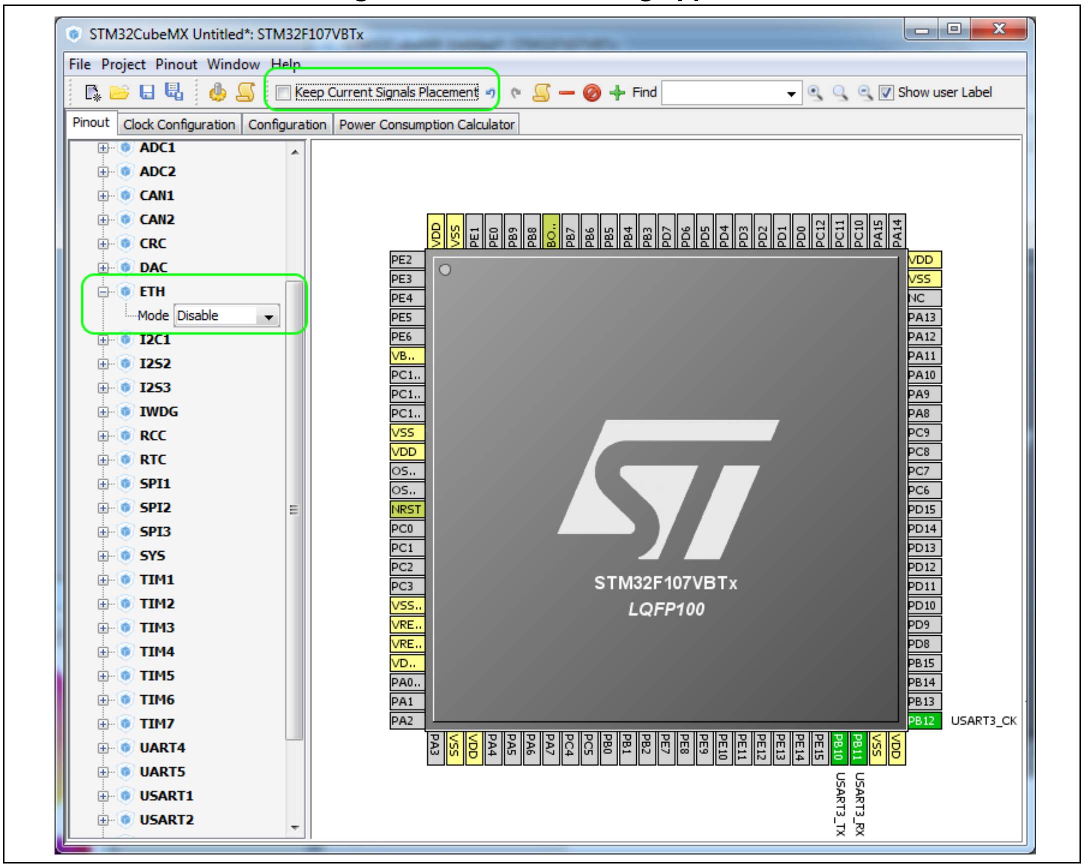


Figure 677. Block shifting applied



A.7 Setting and clearing a peripheral mode

The Peripherals panel and the **Pinout** view are linked: when a peripheral mode is set or cleared, the corresponding pin functions are set or cleared.

A.8 Mapping a function individually

When STM32CubeMX needs a pin that has already been assigned manually to a function (no peripheral mode set), it can move this function to another pin, only if Keep Current Signals Placement is unchecked and the function is not pinned (no pin icon).

A.9 GPIO signals mapping

I/O signals (GPIO_Input, GPIO_Output, GPIO_Analog) can be assigned to pins either manually through the **Pinout** view or automatically through the **Pinout** menu. Such pins can no longer be assigned automatically to another signal: STM32CubeMX signal automatic placement does not take into account this pin anymore since it does not shift I/O signals to other pins.

The pin can still be manually assigned to another signal or to a reset state.

Appendix B STM32CubeMX C code generation design choices and limitations

B.1 STM32CubeMX generated C code and user sections

The C code generated by STM32CubeMX provides user sections as illustrated below. They allow user C code to be inserted and preserved at next C code generation.

User sections shall neither be moved nor renamed. Only the user sections defined by STM32CubeMX are preserved. User created sections will be ignored and lost at next C code generation.

```
 /* USER CODE BEGIN 0 */
(..)
/* USER CODE END 0 */
```

Note: STM32CubeMX may generate C code in some user sections. It will be up to the user to clean the parts that may become obsolete in this section. For example, the while(1) loop in the main function is placed inside a user section as illustrated below:

```
/* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
  }
/* USER CODE END 3 */
```

B.2 STM32CubeMX design choices for peripheral initialization

STM32CubeMX generates peripheral *_Init* functions that can be easily identified thanks to the MX_ prefix:

```
static void MX_GPIO_Init(void);
static void MX_<Peripheral Instance Name>_Init(void);
static void MX_I2S2_Init(void);
```

An *MX_<peripheral instance name>_Init* function exists for each peripheral instance selected by the user (e.g, *MX_I2S2_Init*). It performs the initialization of the relevant handle structure (e.g, &hi2s2 for I2S second instance) that is required for HAL driver initialization (e.g., *HAL_I2S_Init*) and the actual call to this function:

```
void MX_I2S2_Init(void)
{
 hi2s2.Instance = SPI2;
  hi2s2.Init.Mode = I2S_MODE_MASTER_TX;
  hi2s2.Init.Standard = I2S_STANDARD_PHILLIPS;
  hi2s2.Init.DataFormat = I2S_DATAFORMAT_16B;
  hi2s2.Init.MCLKOutput = I2S_MCLKOUTPUT_DISABLE;
```

```
  hi2s2.Init.AudioFreq = I2S_AUDIOFREQ_192K;
  hi2s2.Init.CPOL = I2S_CPOL_LOW;
  hi2s2.Init.ClockSource = I2S_CLOCK_PLL;
  hi2s2.Init.FullDuplexMode = I2S_FULLDUPLEXMODE_ENABLE;
  HAL_I2S_Init(&hi2s2);
}
```

By default, the peripheral initialization is done in *main.c*. If the peripheral is used by a middleware mode, the peripheral initialization can be done in the middleware corresponding .c file.

Customized *HAL_<Peripheral Name>_MspInit()* functions are created in the stm32f4xx_hal_msp.c file to configure the low-level hardware (GPIO, CLOCK) for the selected peripherals.

B.3 STM32CubeMX design choices and limitations for middleware initialization

B.3.1 Overview

STM32CubeMX does not support C user code insertion in Middleware stack native files although stacks such as LwIP might require it in some use cases.

STM32CubeMX generates middleware *Init* functions that can be easily identified thanks to the MX_ prefix:

```
MX_LWIP_Init(); // defined in lwip.h file
MX_USB_HOST_Init(); // defined in usb_host.h file
MX_FATFS_Init(); // defined in fatfs.h file
```

Note however the following exceptions:

- No *Init* function is generated for FreeRTOS unless the user chooses, from the Project Settings window, to generate *Init* functions as pairs of .c/.h files. Instead, a *StartDefaultTask* function is defined in the *main.c* file and CMSIS-RTOS native function (*osKernelStart*) is called in the main function.
- If FreeRTOS is enabled, the *Init* functions for the other middlewares in use are called from the *StartDefaultTask* function in the main.c file.

  Example:

```
void StartDefaultTask(void const * argument)
{
   /* init code for FATFS */
   MX_FATFS_Init();
   /* init code for LWIP */
   MX_LWIP_Init();
  /* init code for USB_HOST */
  MX_USB_HOST_Init();
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
```

```
    osDelay(1);
  }
  /* USER CODE END 5 */
}
```

B.3.2 USB host

USB peripheral initialization is performed within the middleware initialization C code in the *usbh_conf.c* file, while USB stack initialization is done within the *usb_host.c* file.

When using the USB Host middleware, the user is responsible for implementing the *USBH_UserProcess* callback function in the generated *usb_host.c* file.

From STM32CubeMX user interface, the user can select to register one class or all classes if the application requires switching dynamically between classes.

B.3.3 USB device

USB peripheral initialization is performed within the middleware initialization C code in the *usbd_conf.c* file, while USB stack initialization is done within the *usb_device.*c file.

USB VID, PID and String standard descriptors are configured via STM32CubeMX user interface and available in the *usbd_desc.c* generated file. Other standard descriptors (configuration, interface) are hard-coded in the same file preventing support of USB composite devices.

When using the USB Device middleware, the user is responsible for implementing the functions in the *usbd_<classname>_if.c* class interface file for all device classes (such as usbd_storage_if.c).

USB MTP and CCID classes are not supported.

B.3.4 FatFs

FatFs is a generic FAT/exFAT file system solution well suited for small embedded systems.

FatFs configuration is available in *ffconf.h* generated file.

The initialization of the SDIO peripheral for the FatFs SD card mode and of the FMC peripheral for the FatFs External SDRAM and External SRAM modes are kept in the *main.c* file.

Some files need to be modified by the user to match user board specificities (BSP in STM32Cube embedded software package can be used as example):

- *bsp_driver_sd.c/.h* generated files when using FatFs SD card mode
- *bsp_driver_sram.c/.h* generated files when using FatFs External SRAM mode
- *bsp_driver_sdram.c/.h* generated files when using FatFs External SDRAM mode.

Multi-drive FatFs is supported, which means that multiple logical drives can be used by the application (External SDRAM, External SRAM, SD card, USB disk, User defined). However support of multiple instances of a given logical drive is not available (e.g. FatFs using two instances of USB hosts or several RAM disks).

NOR and NAND flash memory are not supported. In this case, the user shall select the FatFs user-defined mode and update the *user_diskio.c* driver file generated to implement the interface between the middleware and the selected peripheral.

B.3.5 FreeRTOS

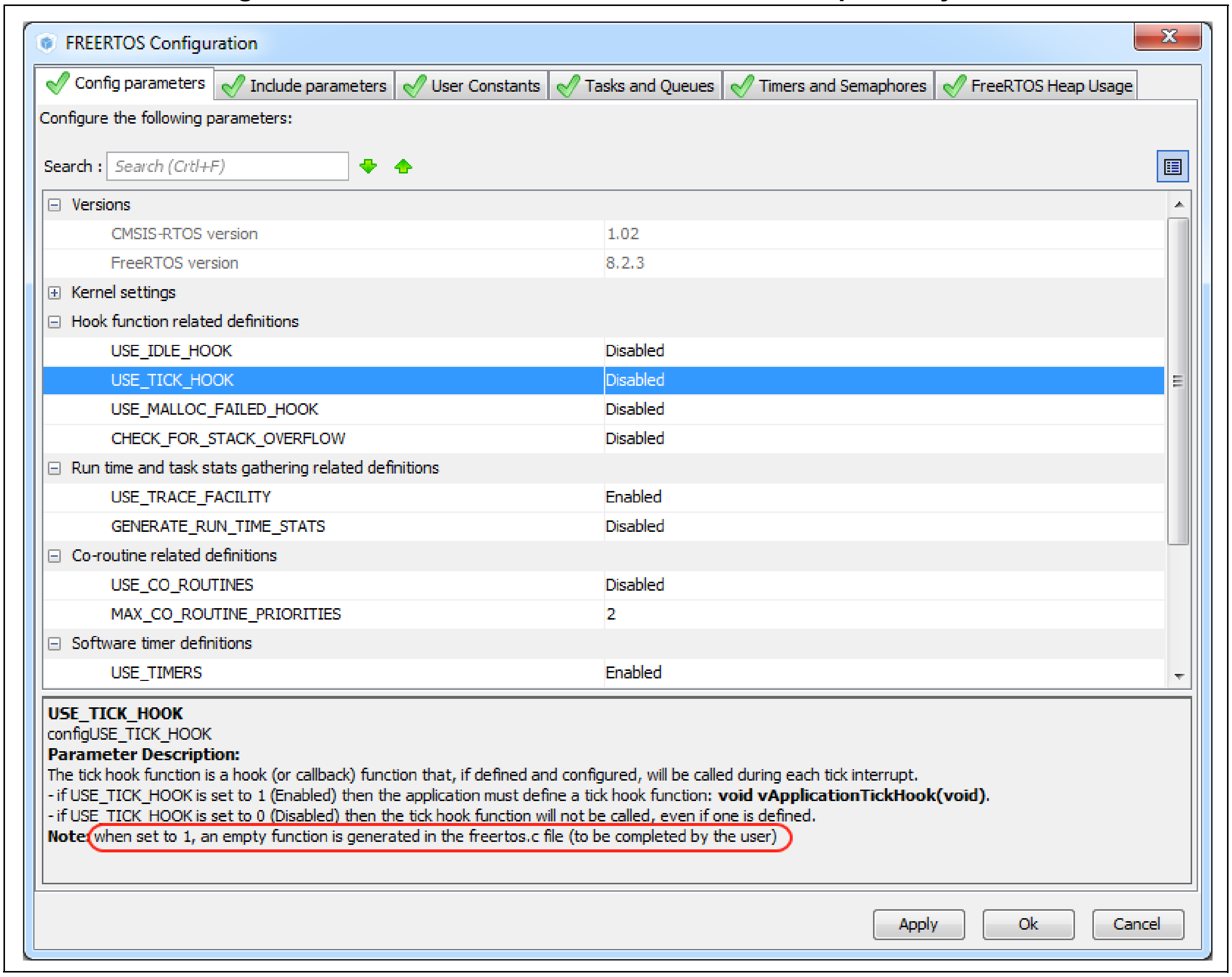
FreeRTOS is a free real-time embedded operating system well suited for microcontrollers.

FreeRTOS configuration is available in *FreeRTOSConfig.h* generated file.

When FreeRTOS is enabled, all other selected middleware modes (e.g., LwIP, FatFs, USB) will be initialized within the same FreeRTOS thread in the main.c file.

When GENERATE_RUN_TIME_STATS, CHECK_FOR_STACK_OVERFLOW, USE_IDLE_HOOK, USE_TICK_HOOK and USE_MALLOC_FAILED_HOOK parameters are activated, STM32CubeMX generates *freertos.c* file with empty functions that the user shall implement. This is highlighted by the tooltip (see *Figure 678*).

Figure 678. FreeRTOS HOOK functions to be completed by user



B.3.6 LwIP

LwIP is a small independent implementation of the TCP/IP protocol suite: its reduced RAM usage makes it suitable for use in embedded systems with tens of Kbytes of free RAM.

LwIP initialization function is defined in *lwip.c*, while LwIP configuration is available in *lwipopts.h* generated file.

STM32CubeMX supports LwIP over Ethernet only. The Ethernet peripheral initialization is done within the middleware initialization C code.

STM32CubeMX does not support user C code insertion in stack native files. However, some LwIP use cases require modifying stack native files (e.g., *cc.h, mib2.c*): user modifications shall be backed up since they will be lost at next STM32CubeMX generation.

Starting with LwIP release 1.5, STM32CubeMX LwIP supports IPv6 (see *Figure 680*).

DHCP must be disabled, to configure a static IP address.

Figure 679. LwIP 1.4.1 configuration

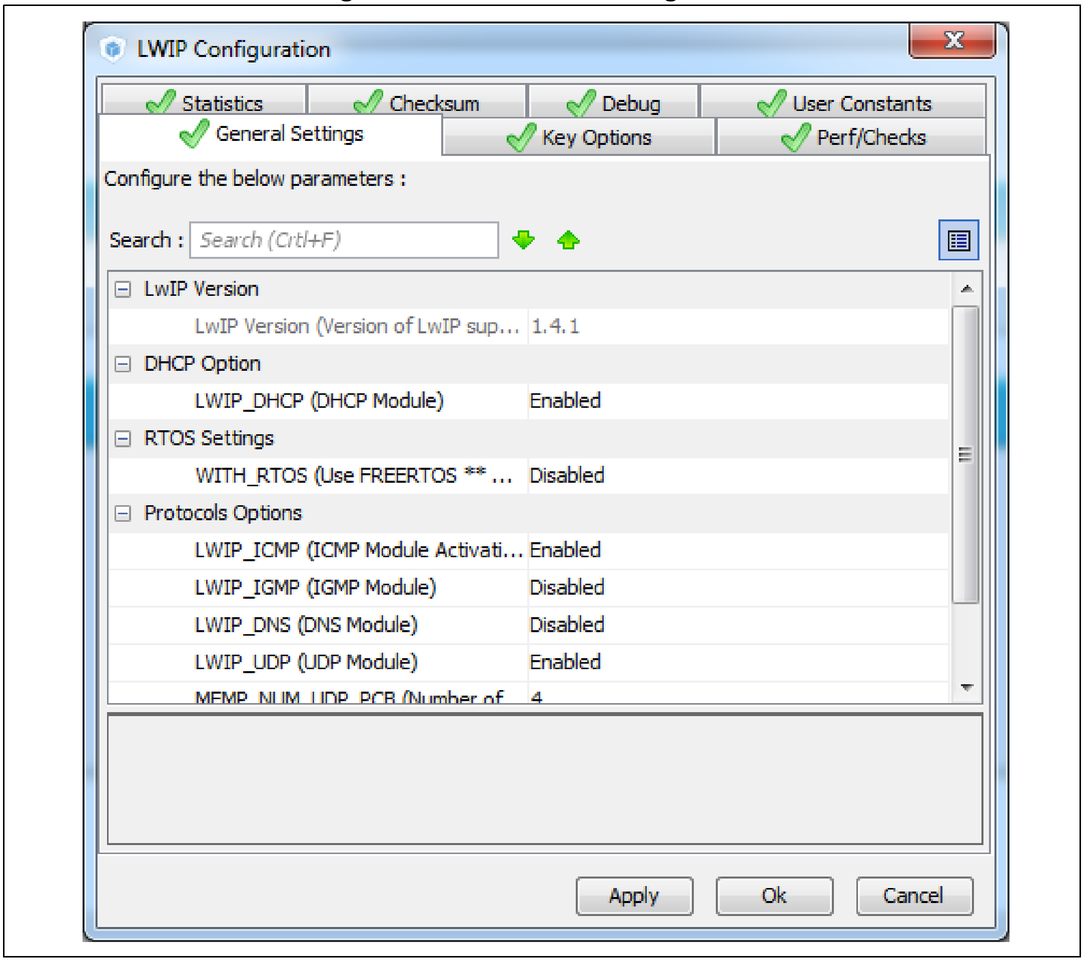
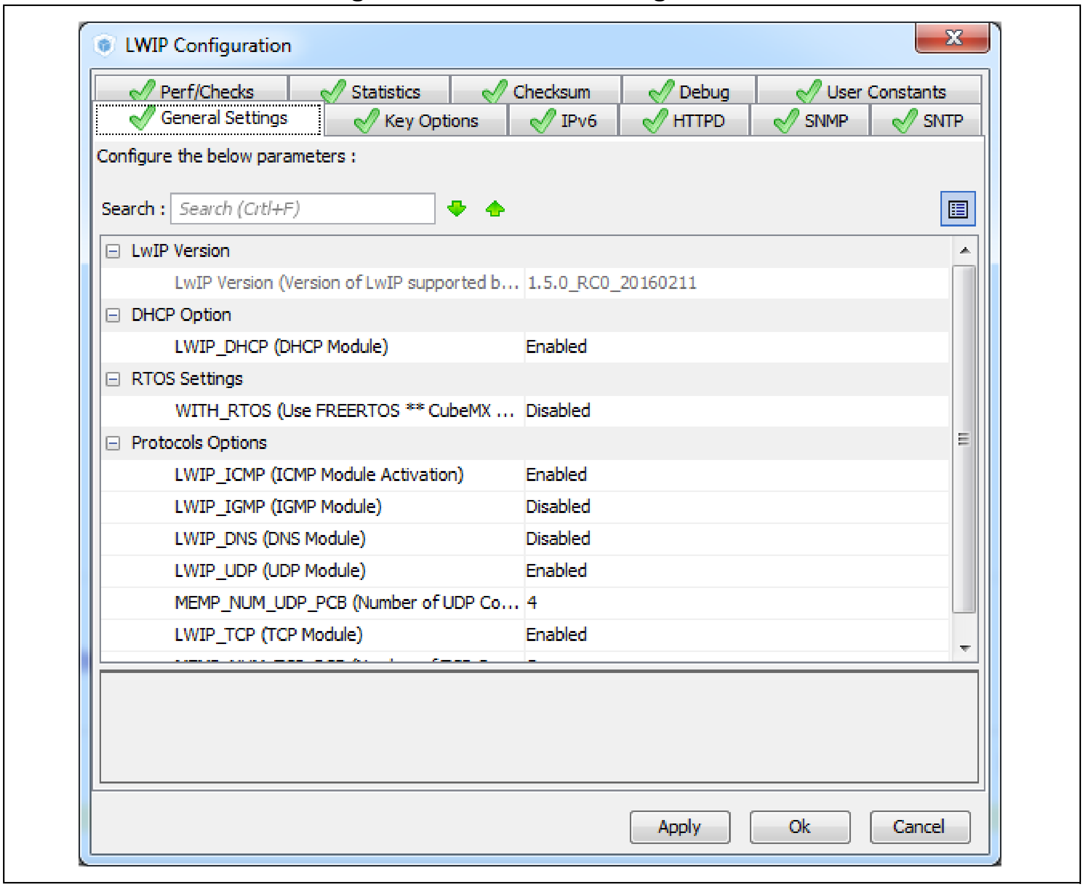


Figure 680. LwIP 1.5 configuration



STM32CubeMX generated C code reports compilation errors when specific parameters are enabled (disabled by default). The user must fix the issues with a stack patch (downloaded from Internet) or user C code. The following parameters generate an error:

- MEM_USE_POOLS: user C code to be added either in *lwipopts.h* or in *cc.h* (stack file).
- PPP_SUPPORT, PPPOE_SUPPORT: user C code required
- MEMP_SEPARATE_POOLS with MEMP_OVERFLOW_CHECK > 0: a stack patch required
- MEM_LIBC_MALLOC & RTOS enabled: stack patch required
- LWIP_EVENT_API: stack patch required

In STM32CubeMX, the user must enable FreeRTOS in order to use LwIP with the netconn and sockets APIs. These APIs require the use of threads and consequently of an operating system. Without FreeRTOS, only the LwIP event-driven raw API can be used.

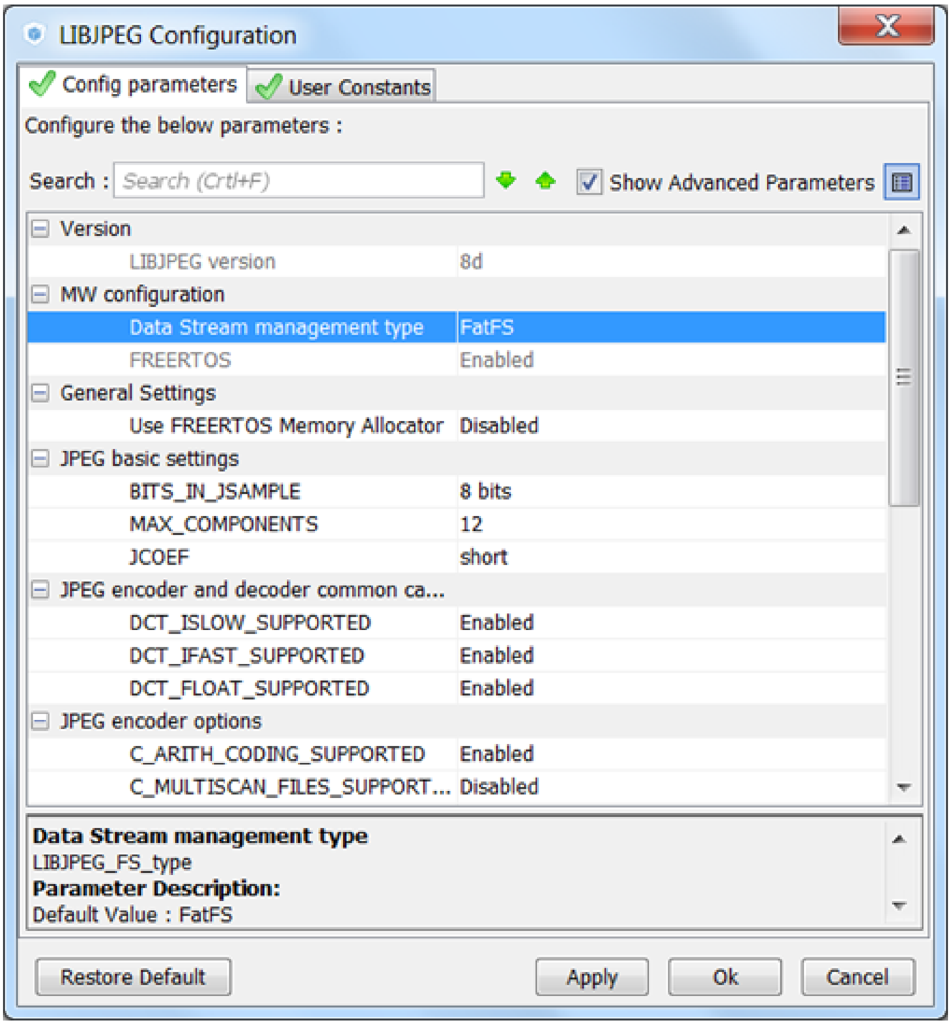
B.3.7 Libjpeg

Libjpeg is a widely used C-library that allows reading and writing JPEG files. It is delivered within STM32CubeF7, STM32CubeH7, STM32CubeF2 and STM32CubeF4 embedded software packages.

STM32CubeMX generates the following files, whose content can be configured by the user through STM32CubeMX user interface:

- **libjpeg.c/.h**
  The *MX_LIBJPEG_Init()* initialization function is generated within the libjpeg.c file. It is empty. It is up to the user to enter in the user sections the code and the calls to the libjpeg functions required for the application.
- **jdata_conf.c**
  This file is generated only when FatFs is selected as data stream management type.
- **jdata_conf.h**
  The content of this file is adjusted according to the datastream management type selected.
- **jconfig.h**
  This file is generated by STM32CubeMX. but cannot be configured.
- **jmorecfg.h**
  Some but not all the define statements contained in this file can be modified through the STM32CubeMX libjpeg configuration menu.

Figure 681. Libjpeg configuration window



B.3.8 Mbed TLS

Mbed TLS is a C-library that allows including cryptographic capabilities to embedded products. It handles Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols, that are used for establishing a secure, encrypted and authenticated link between two parties over an insecure network. Mbed TLS comes with an intuitive API and minimal coding footprint. Visit https://tls.mbed.org/ for more details.

Mbed TLS is delivered within STM32CubeF2, STM32CubeF4, STM32CubeF7 and STM32CubeH7 embedded software packages.

Mbed TLS can work without LwIP stack (see *Figure 682*).

If LwIP stack is used, FreeRTOS must be enabled as well (see *Figure 683*).

STM32CubeMX generates the following files, whose contents can be modified by the user through STM32CubeMX user interface (see *Figure 684*) and/or using user sections in the code itself:

- *mbedtls_config.h*
- *mbedtls.h*
- *net_sockets.c* (generated only if LwIP is enabled)
- *mbedtls.c*

Figure 682. Mbed TLS without LwIP

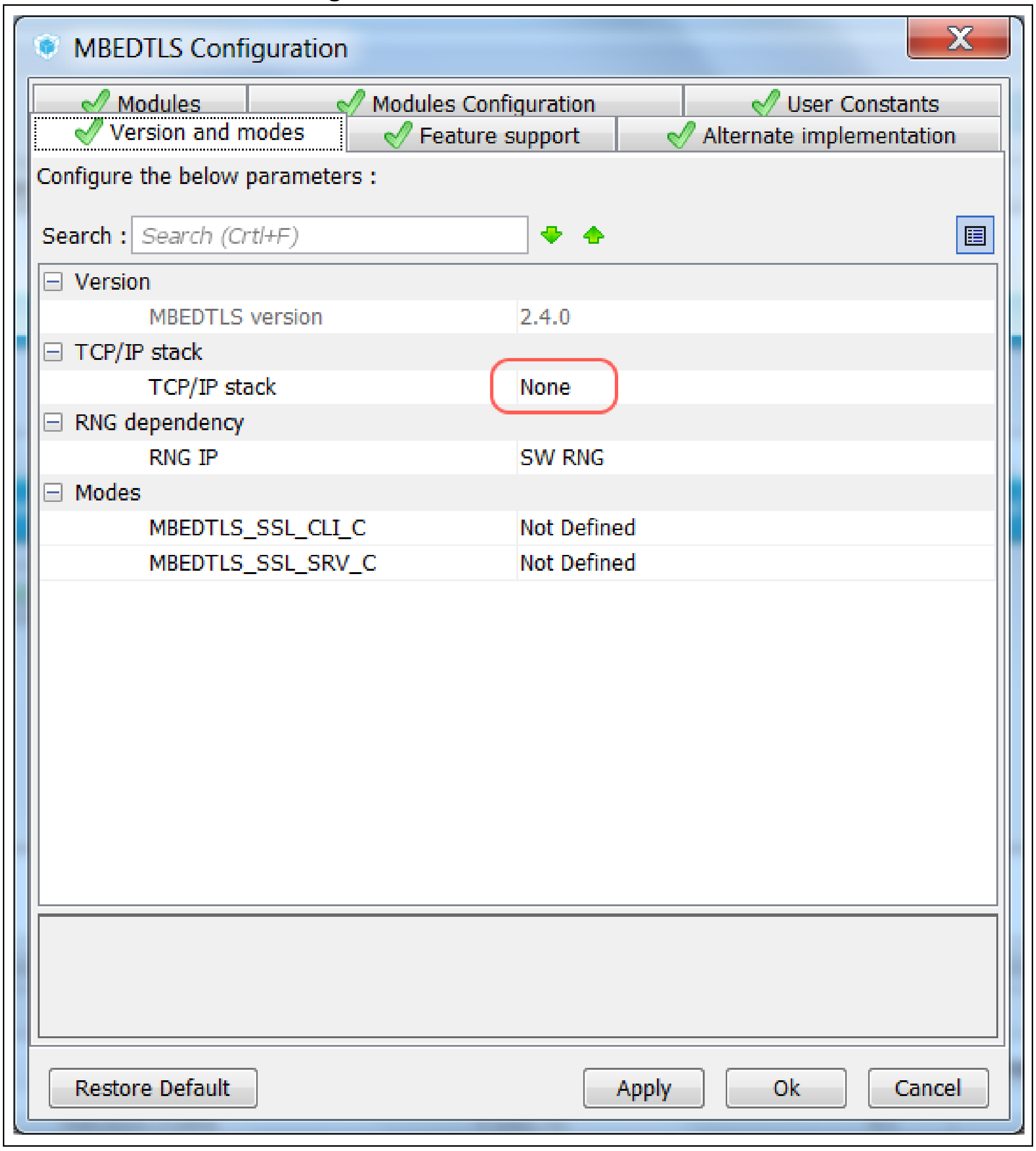


Figure 683. Mbed TLS with LwIP and FreeRTOS

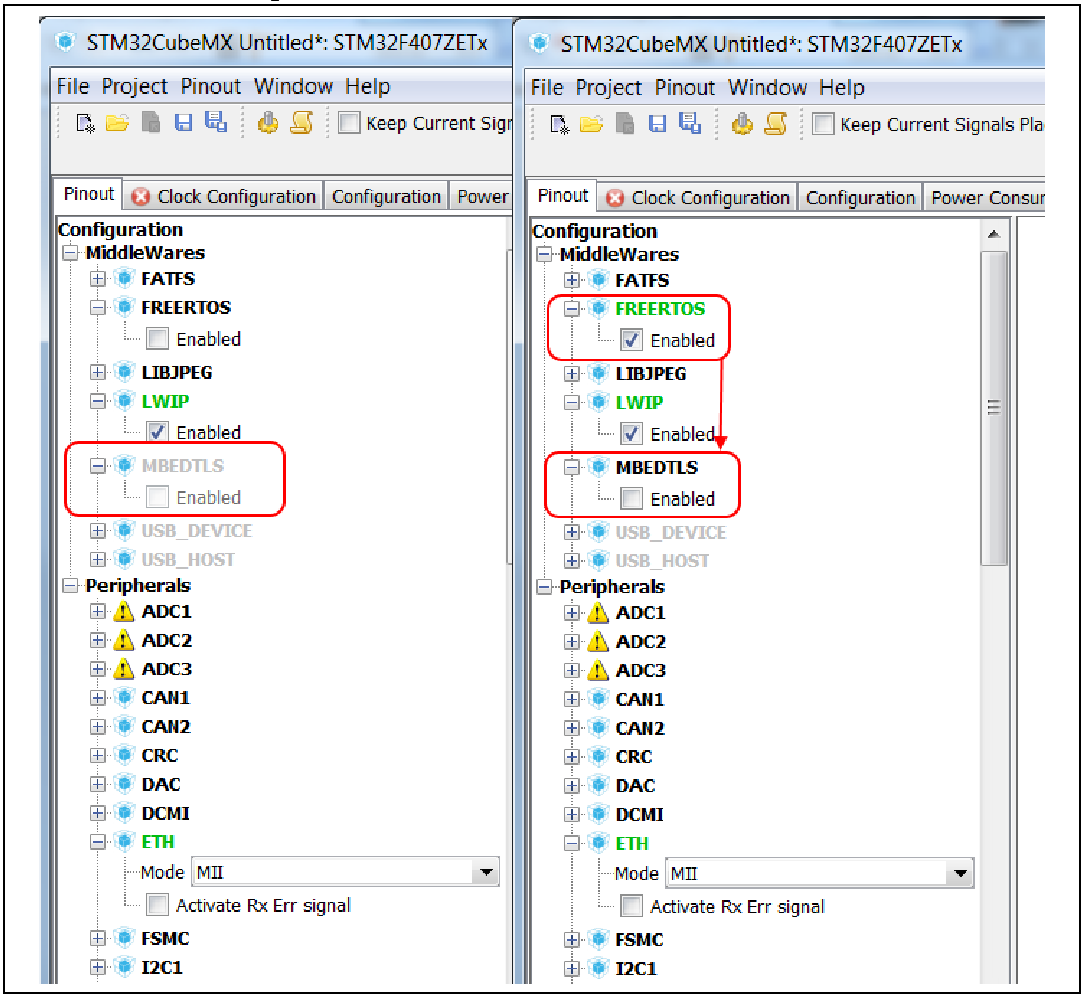
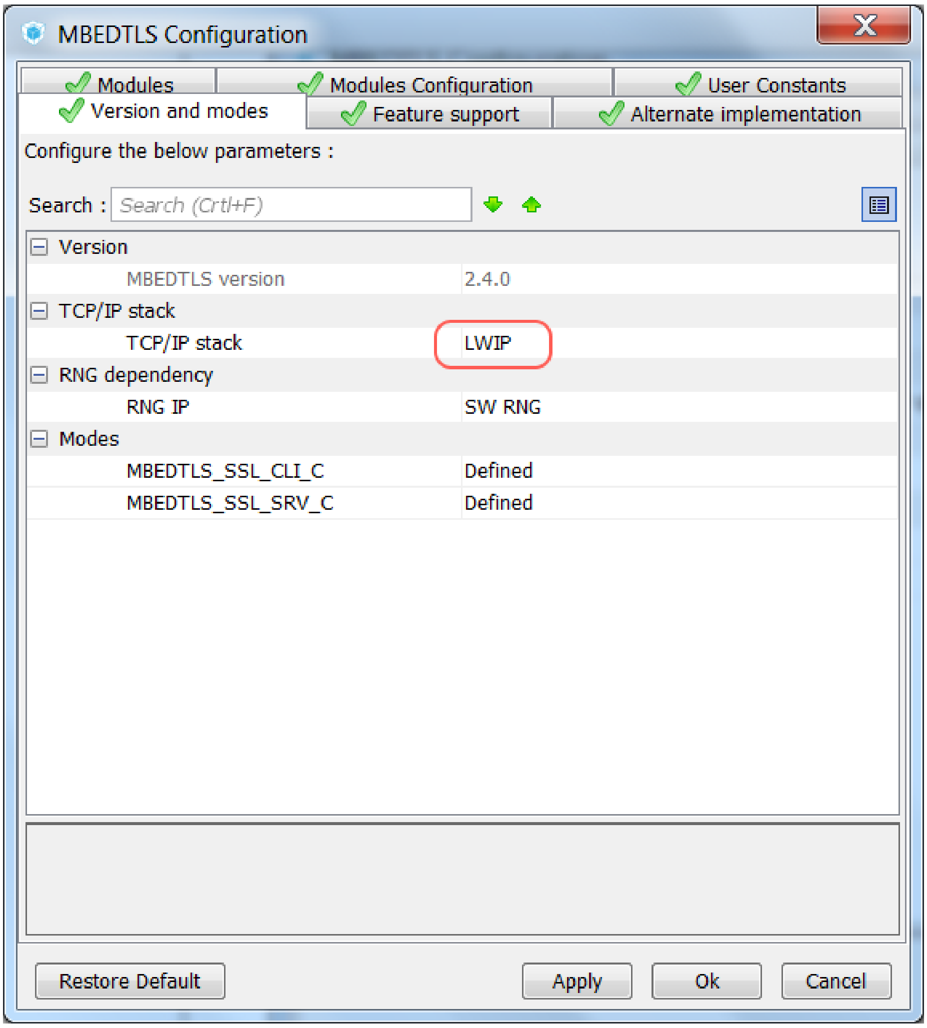


Figure 684. Mbed TLS configuration window



B.3.9 TouchSensing

The STM32 TouchSensing library is a C-library that allows the creation of higher-end human interfaces by replacing conventional electromechanical switches by capacitive sensors with STM32 microcontrollers.

It requires the touch-sensing peripheral to be configured on the microcontroller.

STM32CubeMX generates the following files, whose contents can be modified by the user through STM32CubeMX user interface (see *Figure 685*, *Figure 686*, and *Figure 687*) and/or using user sections in the code itself:

- *touchsensing.c/.h*
- *tsl_user.c/.h*
- *tsl_conf.h*

Figure 685. Enabling the TouchSensing peripheral

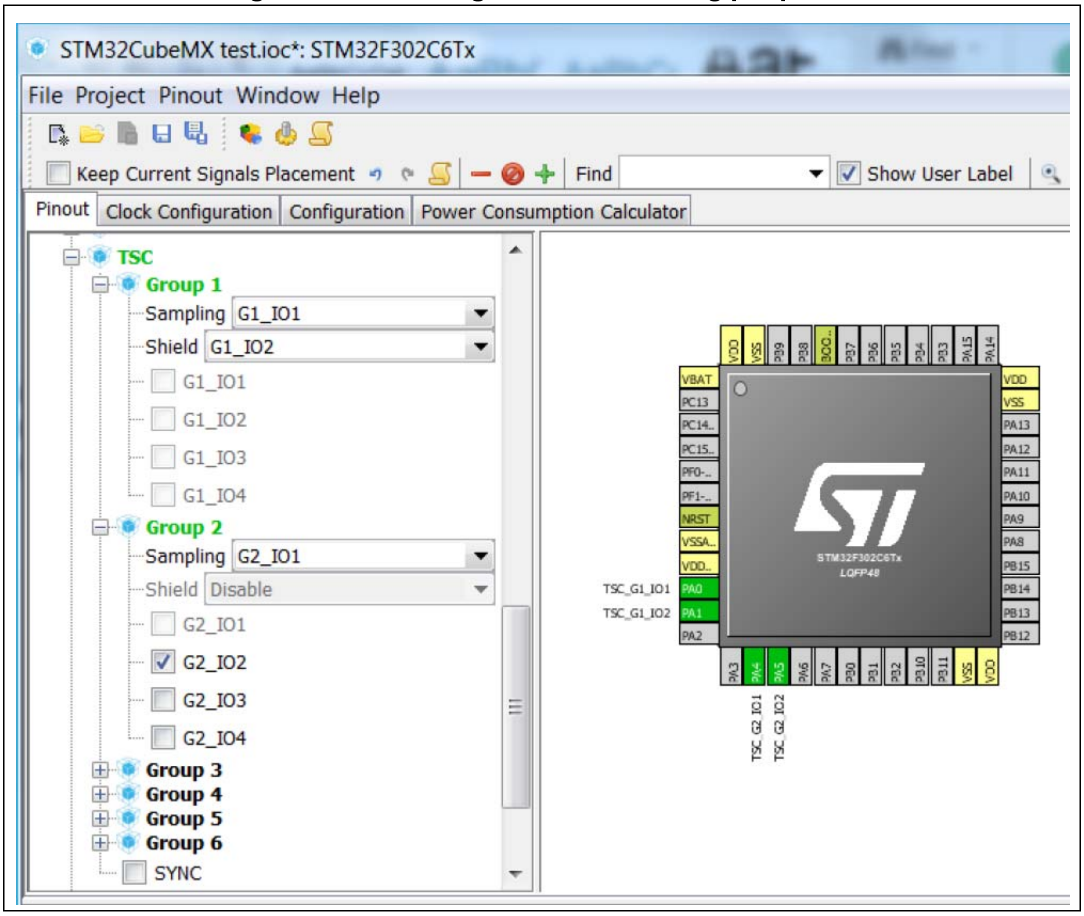


Figure 686. Touch-sensing sensor selection panel

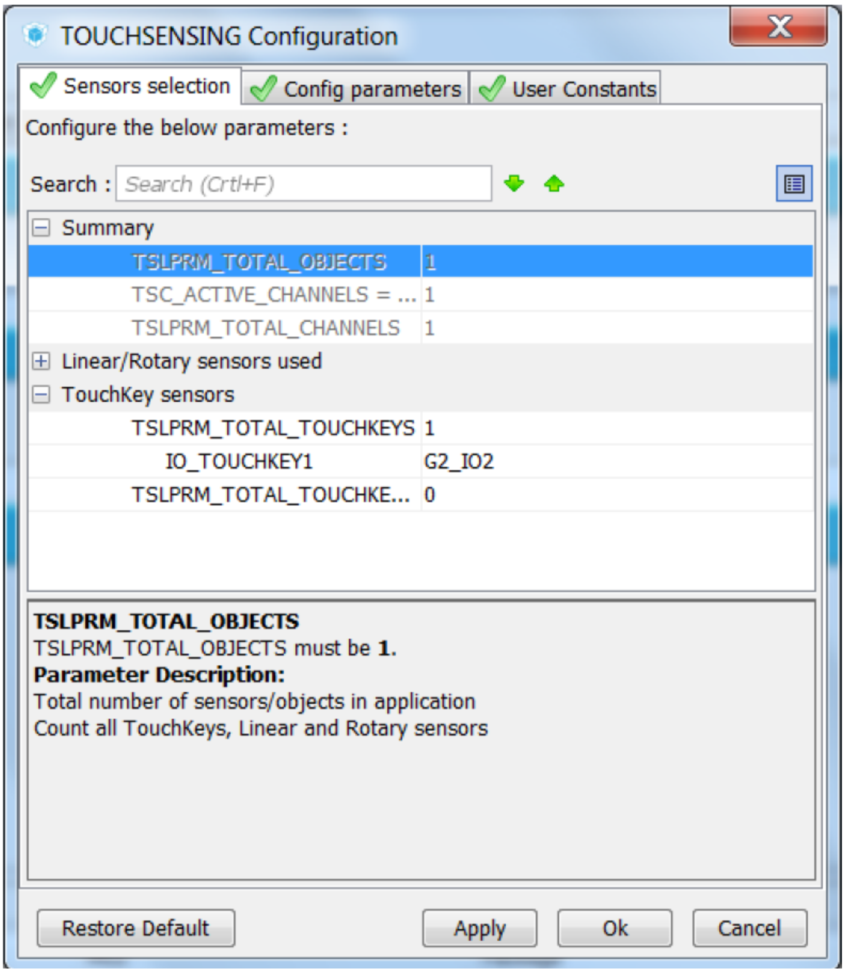
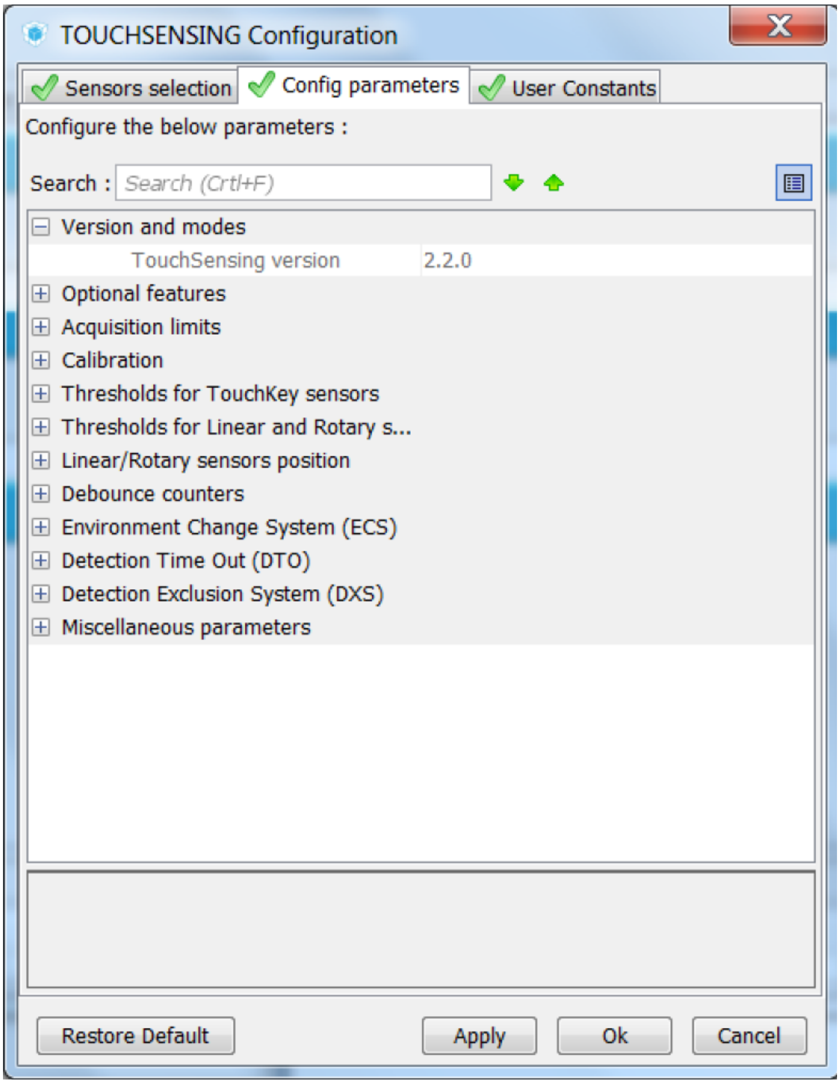


Figure 687. TouchSensing configuration panel



B.3.10 PDM2PCM

The PDM2PCM library is a C-library that allows converting a pulse density modulated (PDM) data output into a 16-bit pulse-code modulation (PCM) format. It requires the CRC peripheral to be enabled.

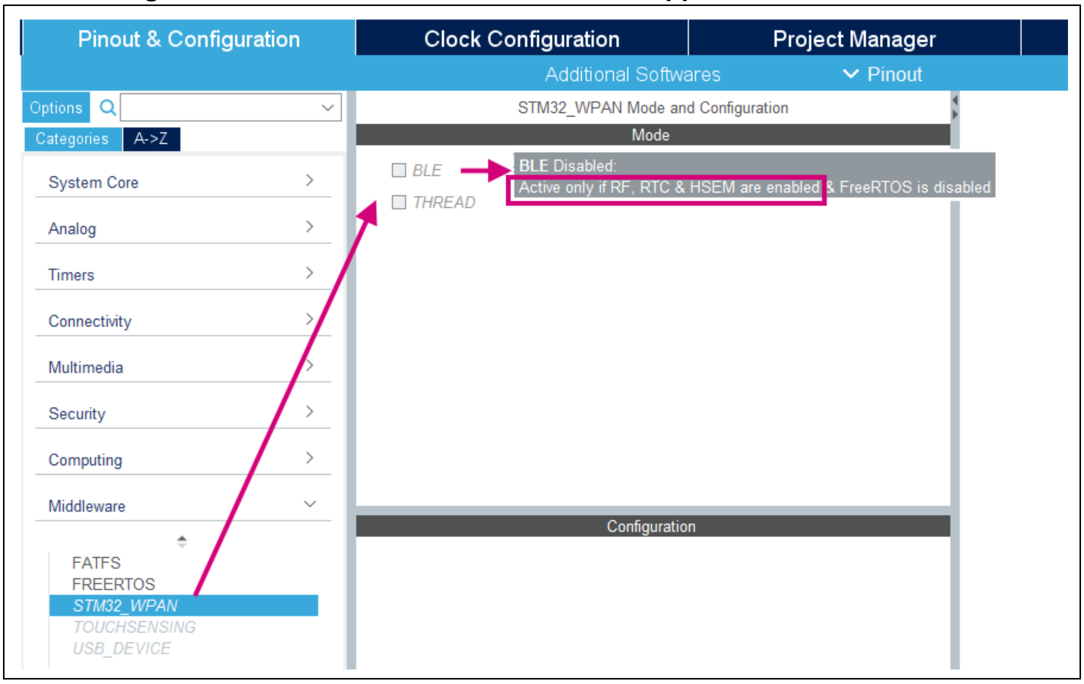
STM32CubeMX generates the following files, whose content can be modified by the user through STM32CubeMX user interface and/or using user sections in the code itself:

- *pdm2pcm.h/.c*

B.3.11 STM32WPAN BLE/Thread (STM32WB series only)

STM32WPAN BLE and Thread middleware are now supported in STM32CubeMX.

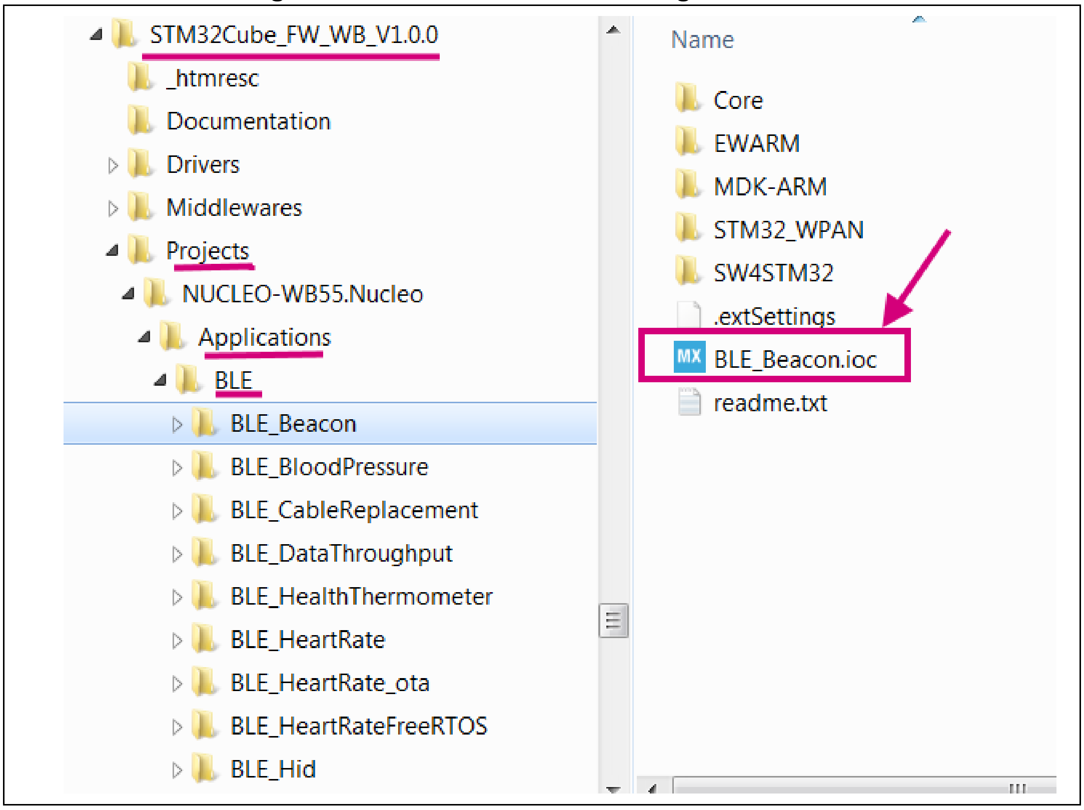
Figure 688. BLE and Thread middleware support in STM32CubeMX



They are exclusive in a given project and configuration with FreeRTOS is not yet supported.

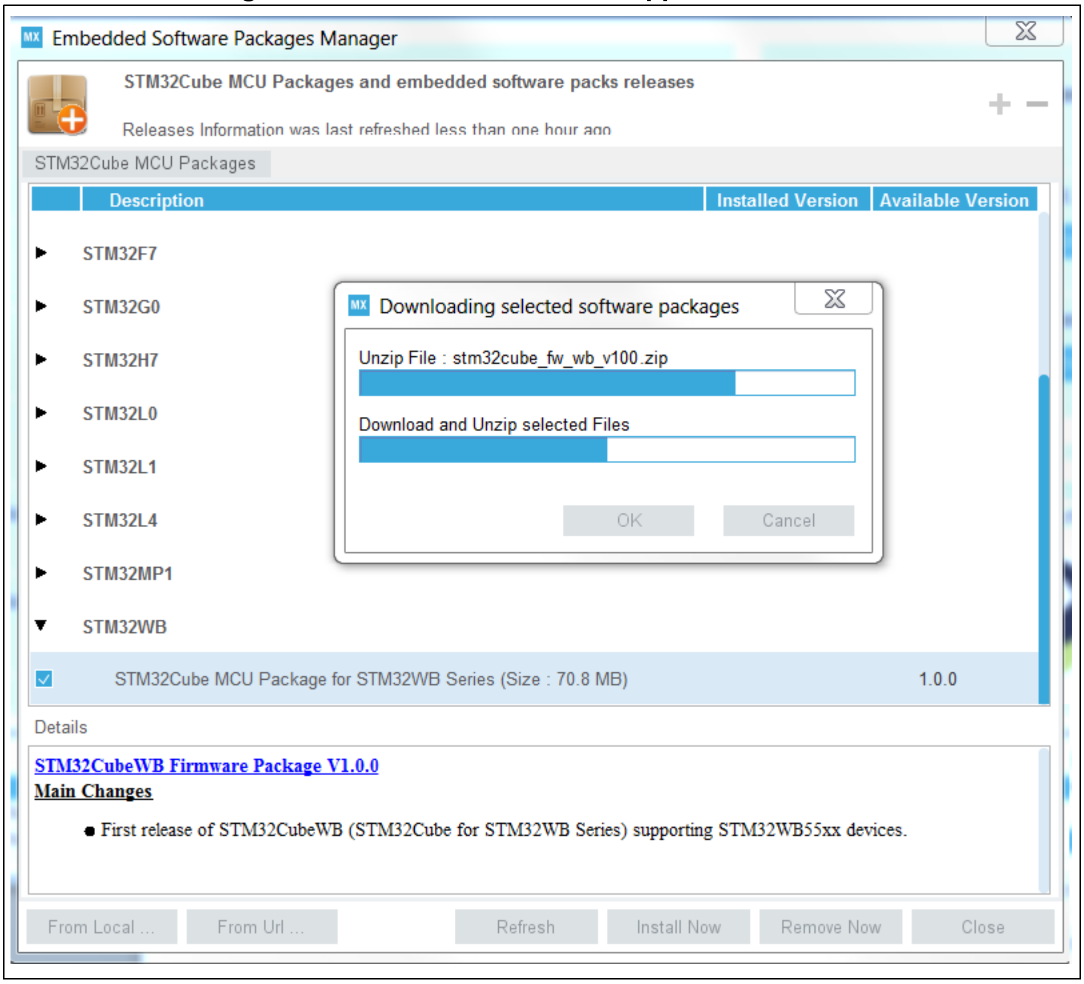
Application projects generated with STM32CubeMX can be found in the project folder of the STM32CubeWB MCU package.

Figure 689. STM32CubeWB Package download



This package can be installed through STM32CubeMX following the standard procedure described in *Section 3.4.3: Installing STM32 MCU packages*.

Figure 690. STM32CubeWB BLE applications folder



BLE configuration

To enable BLE some peripherals (RTC, HSEM, RF) must be activated first.

Then, an application type must be selected, it can be one among Transparent mode, Server profile, Router profile or Client profile.

Finally, the mode and other parameters relevant to this application type must be configured.

Note: *The BLE Transparent mode and all Thread applications require either the USART or the LPUART peripheral to be configured as well.*

Figure 691. BLE Server profile selection

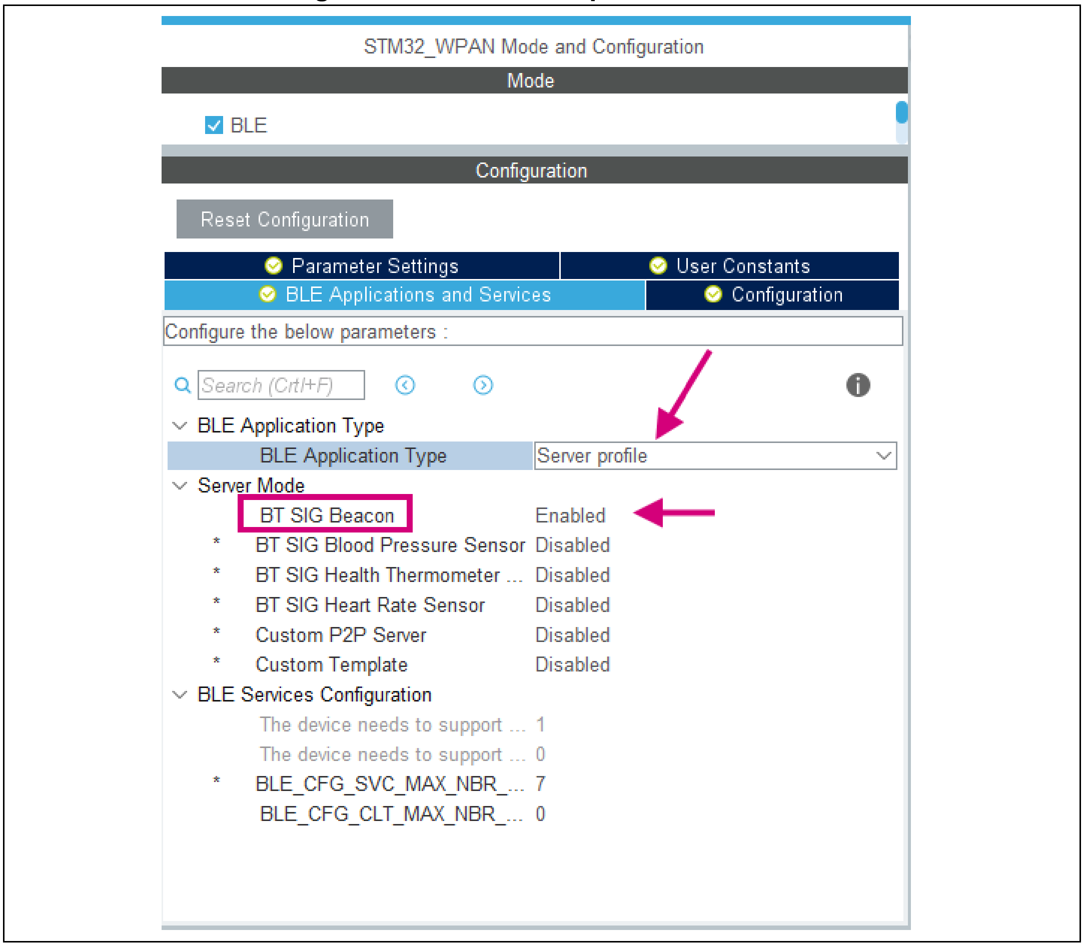
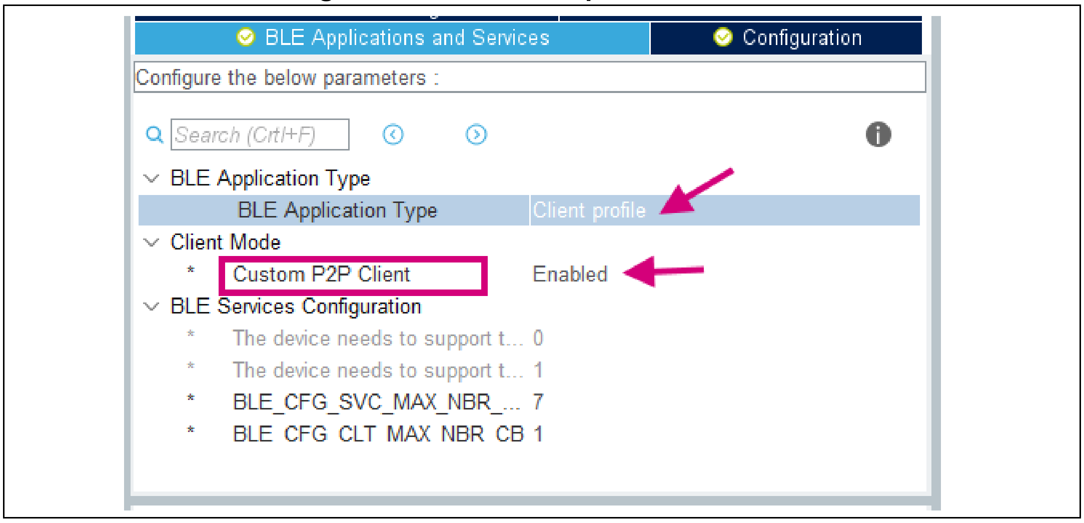


Figure 692. BLE Client profile selection

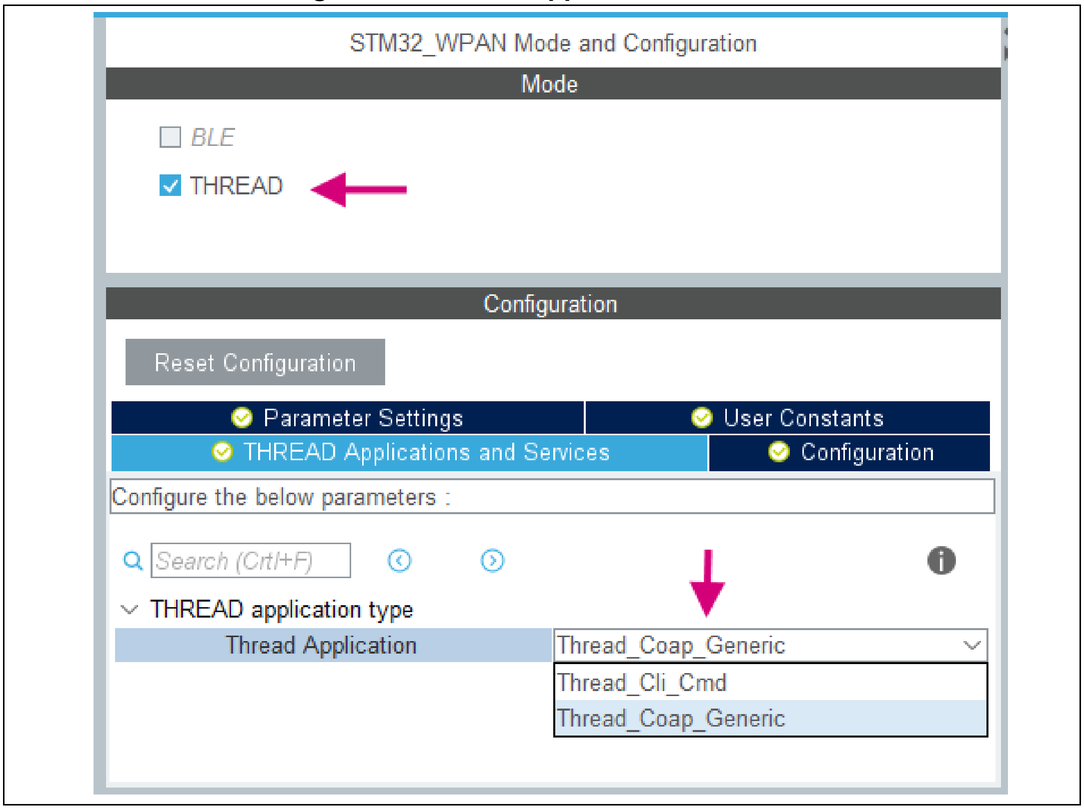


Thread configuration

To enable Thread some peripherals (RTC, HSEM, RF) must be activated first.

Then, an application type must be selected and the relevant parameters configured.

Figure 693. Thread application selection



B.3.12 CMSIS packs selection limitation

The restriction about applications comes from a simple generated code consideration: an application is meant to be the root of the execution (excluding the main function).

This means that the generated function defines the execution of the selected application. In that sense, it is meant to be the last call of the main method, and must not give hand back to the main function.Two applications cannot be called, as this means generating calls in the main function, and then the second call is never reached.

If you need to call both applications:

- An RTOS must run them in threads, or
- You manually add the right code to execute them (in that context, they are not applications, as they are not at the root of the execution), or
- Change the meaning of the application components.

B.3.13 OpenAmp and RESMGR_UTILITY (STM32MPUs and STM32H7 dual-core products)

New software and hardware have been introduced on dual-core products to enable multi-core cooperation.

- For STM32MPUs only: the inter-processor communication controller (IPCC) used to exchange data between two processor instances relies on the fact that shared memory buffers are allocated in the MCU SRAM and that each processor owns specific register bank and interrupts.
- For STM32MPUs only: the OpenAMP middleware for intercommunication between Cortex-A and Cortex-M cores implements the RPMsg messaging protocol (see *Figure 694*).
- The resource manager library (RESMGR_UTILITY) for system resource management: multi-processor devices give the possibility to run independent firmware on several cores (see *Figure 695*). This implies a core could use some peripherals without knowledge of the usage of these same peripherals: the role of the resource management library is to control the assignment of a peripheral to a dedicated core and to provide a method to configure the system resources used to operate that peripheral (see *Figure 696*).

Figure 694. Enabling OpenAmp for STM32MPUs

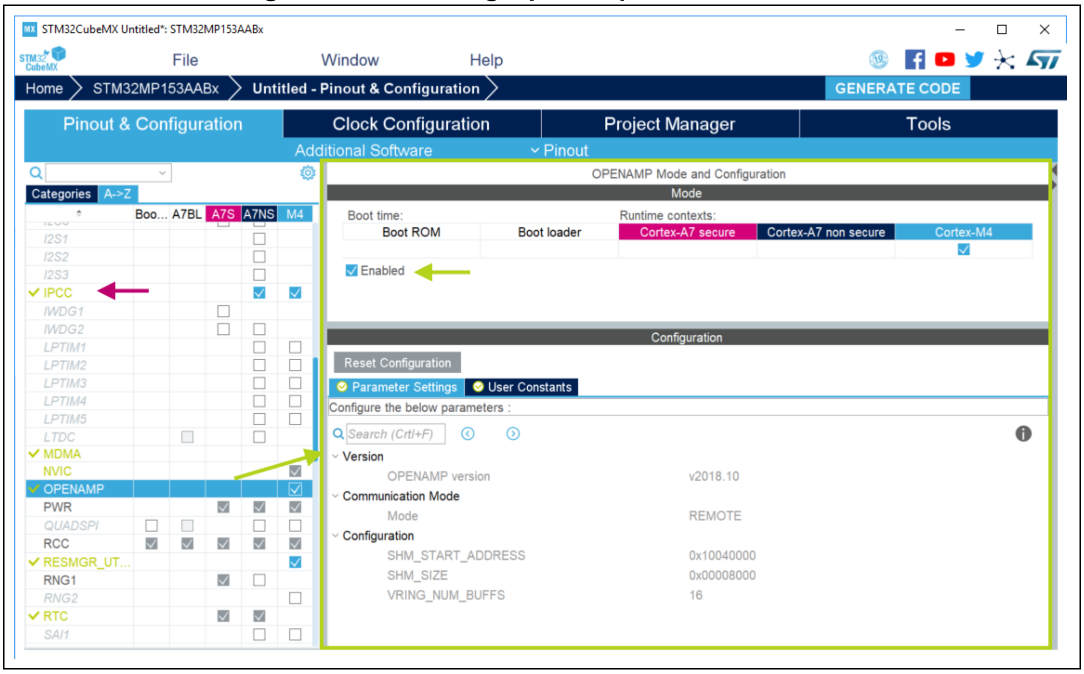


Figure 695. Enabling the Resource Manager for STM32MPUs

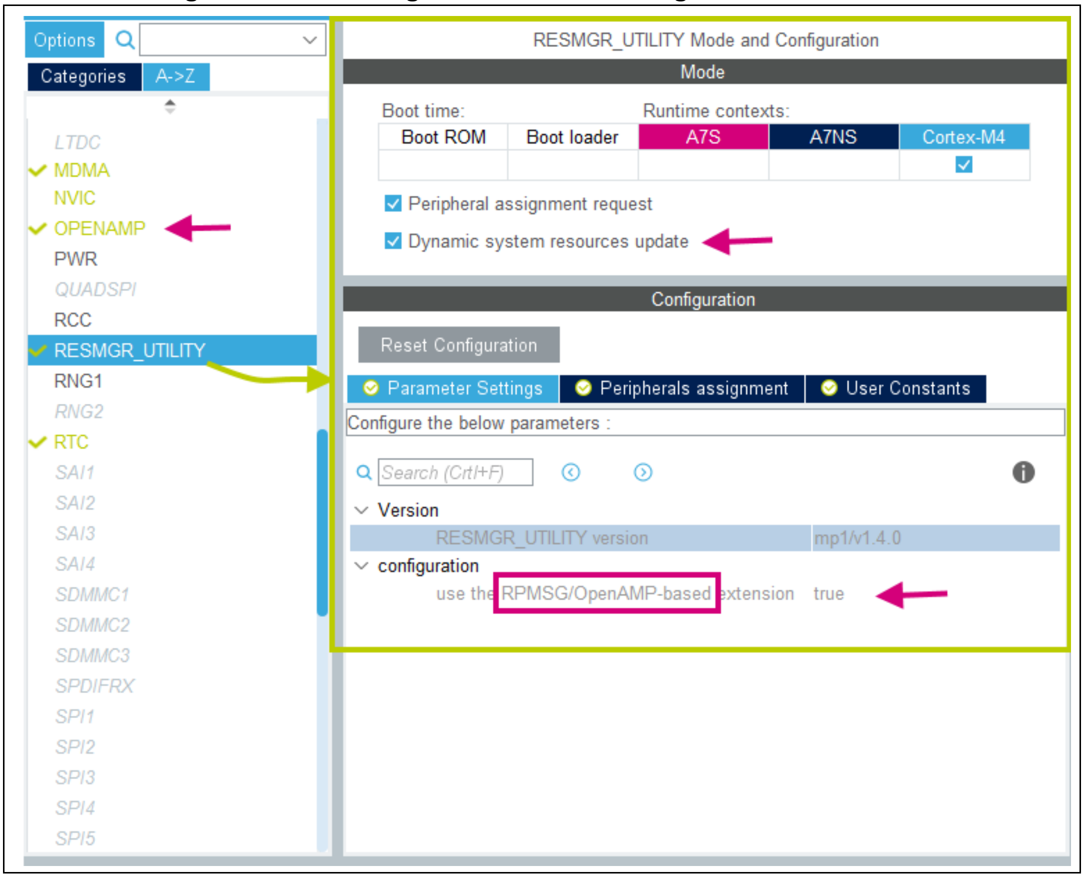
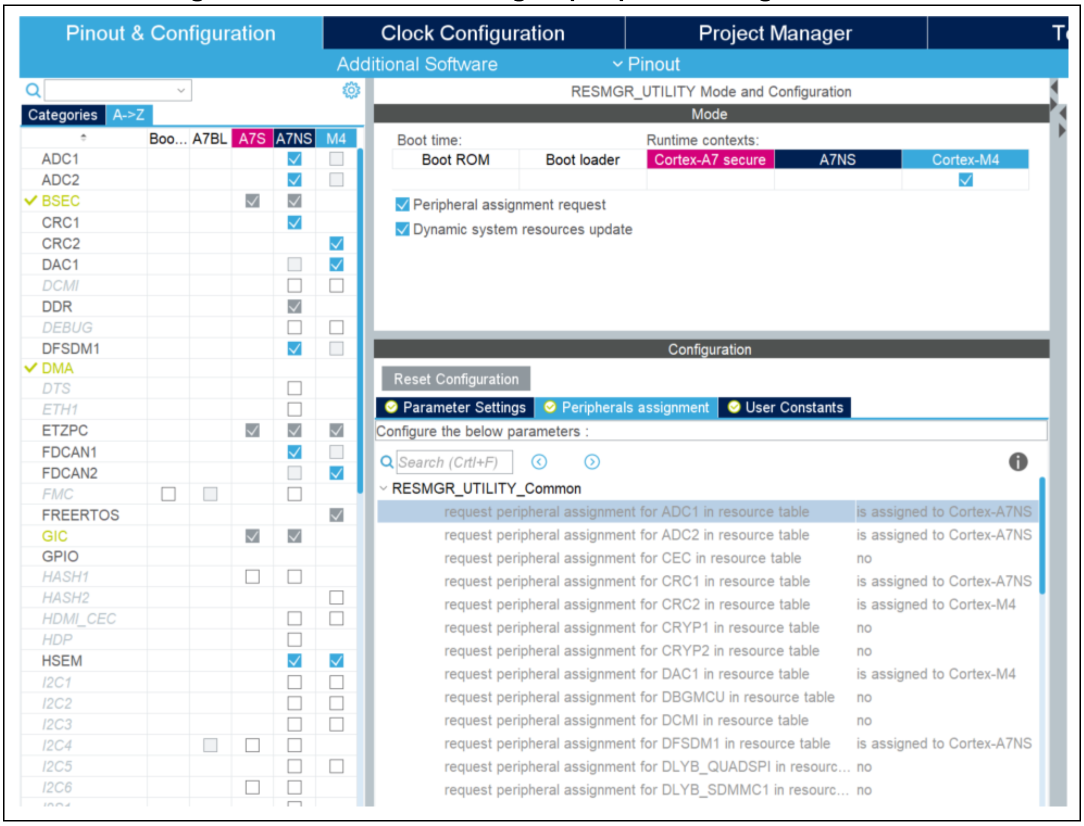


Figure 696. Resource Manager: peripheral assignment view



For more details visit STM32MPUs dedicated wiki site at https://wiki.st.com/stm32mpu.

Appendix C STM32 microcontrollers power consumption parameters

This section provides an overview on how to use STM32CubeMX Power Consumption Calculator.

Microcontroller power consumption depends on chip size, supply voltage, clock frequency and operating mode. Embedded applications can optimize STM32 MCU power consumption by reducing the clock frequency when fast processing is not required and choosing the optimal operating mode and voltage range to run from. A description of STM32 power modes and voltage range is provided below.

C.1 Power modes

STM32 MCUs support different power modes (refer to STM32 MCU datasheets for full details).

C.1.1 STM32L1 series

STM32L1 microcontrollers feature up to 6 power modes, including 5 low-power modes:

- **Run** mode

  This mode offers the highest performance using HSE/HSI clock sources. The CPU runs up to 32 MHz and the voltage regulator is enabled.
- **Sleep** mode

  This mode uses HSE or HSI as system clock sources. The voltage regulator is enabled and the CPU is stopped. All peripherals continue to operate and can wake up the CPU when an interrupt/event occurs.
- **Low- power run** mode

  This mode uses the multispeed internal (MSI) RC oscillator set to the minimum clock frequency (131 kHz) and the internal regulator in low-power mode. The clock frequency and the number of enabled peripherals are limited.
- **Low-power sleep** mode

  This mode is achieved by entering Sleep mode. The internal voltage regulator is in low-power mode. The clock frequency and the number of enabled peripherals are limited. A typical example would be a timer running at 32 kHz.

  When the wake-up is triggered by an event or an interrupt, the system returns to the Run mode with the regulator ON.
- **Stop** mode

  This mode achieves the lowest power consumption while retaining RAM and register contents. Clocks are stopped. The real-time clock (RTC) an be backed up by using LSE/LSI at 32 kHz/37 kHz. The number of enabled peripherals is limited. The voltage regulator is in low-power mode.

  The device can be woken up from Stop mode by any of the EXTI lines.
- **Standby** mode

  This mode achieves the lowest power consumption. The internal voltage regulator is switched off so that the entire V_{CORE} domain is powered off. Clocks are stopped and the real-time clock (RTC) can be preserved up by using LSE/LSI at 32 kHz/37 kHz.

RAM and register contents are lost except for the registers in the Standby circuitry. The number of enabled peripherals is even more limited than in Stop mode.

The device exits Standby mode upon reset, rising edge on one of the three WKUP pins, or if an RTC event occurs (if the RTC is ON).

Note: When exiting Stop or Standby modes to enter the Run mode, STM32L1 MCUs go through a state where the MSI oscillator is used as clock source. This transition can have a significant impact on the global power consumption. For this reason, the Power Consumption Calculator introduces two transition steps: ***WU_FROM_STOP*** *and* ***WU_FROM_STANDBY****. During these steps, the clock is automatically configured to MSI.*

C.1.2 STM32F4 series

STM32F4 microcontrollers feature a total of 5 power modes, including 4 low-power modes:

- **Run** mode

  This is the default mode at power-on or after a system reset. It offers the highest performance using HSE/HSI clock sources. The CPU can run at the maximum frequency depending on the selected power scale.

- **Sleep** mode

  Only the CPU is stopped. All peripherals continue to operate and can wake up the CPU when an interrupt/even occurs. The clock source is the clock that was set before entering Sleep mode.

- **Stop** mode

  This mode achieves a very low power consumption using the RC oscillator as clock source. All clocks in the 1.2 V domain are stopped as well as CPU and peripherals. PLL, HSI RC and HSE crystal oscillators are disabled. The content of registers and internal SRAM are kept.

  The voltage regulator can be put either in normal Main regulator mode (MR) or in Low-power regulator mode (LPR). Selecting the regulator in low-power regulator mode increases the wake-up time.

  The flash memory can be put either in Stop mode to achieve a fast wake-up time. or in Deep power-down to obtain a lower consumption with a slow wake-up time.

  The Stop mode features two sub-modes:

  - **Stop in Normal mode (default mode)**

    In this mode, the 1.2 V domain is preserved in nominal leakage mode and the minimum V12 voltage is 1.08 V.

  - **Stop in Under-drive mode**

    In this mode, the 1.2 V domain is preserved in reduced leakage mode and V12 voltage is less than 1.08 V. The regulator (in Main or Low-power mode) is in under-drive or low-voltage mode. The flash memory must be in Deep-power-down mode. The wake-up time is about 100 µs higher than in normal mode.

- **Standby** mode

  This mode achieves very low power consumption with the RC oscillator as a clock source. The internal voltage regulator is switched off so that the entire 1.2 V domain is powered off: CPU and peripherals are stopped. The PLL, the HSI RC and the HSE crystal oscillators are disabled. SRAM and register contents are lost except for registers in the backup domain and the 4-byte backup SRAM when selected. Only RTC and LSE oscillator blocks are powered. The device exits Standby mode when an external reset (NRST pin), an IWDG reset, a rising edge on the WKUP pin, or an RTC alarm/ wake-up/tamper/time stamp event occurs.

- **V_{BAT}** operation

  It allows to significantly reduced power consumption compared to the Standby mode. This mode is available when the V_{BAT} pin powering the Backup domain is connected to an optional standby voltage supplied by a battery or by another source. The V_{BAT} domain is preserved (RTC registers, RTC backup register and backup SRAM) and RTC and LSE oscillator blocks powered. The main difference compared to the Standby mode is external interrupts and RTC alarm/events do not exit the device from V_{BAT} operation. Increasing V_{DD} to reach the minimum threshold does.

C.1.3 STM32L0 series

STM32L0 microcontrollers feature up to 8 power modes, including 7 low-power modes to achieve the best compromise between low-power consumption, short startup time and available wake-up sources:

- **Run mode**

  This mode offers the highest performance using HSE/HSI clock sources. The CPU can run up to 32 MHz and the voltage regulator is enabled.
- **Sleep mode**

  This mode uses HSE or HSI as system clock sources. The voltage regulator is enabled and only the CPU is stopped. All peripherals continue to operate and can wake up the CPU when an interrupt/event occurs.
- **Low-power run mode**

  This mode uses the internal regulator in low-power mode and the multispeed internal (MSI) RC oscillator set to the minimum clock frequency (131 kHz). In Low-power run mode, the clock frequency and the number of enabled peripherals are both limited.
- **Low-power sleep mode**

  This mode is achieved by entering Sleep mode with the internal voltage regulator in low-power mode. Both the clock frequency and the number of enabled peripherals are limited. Event or interrupt can revert the system to Run mode with regulator on.
- **Stop mode with RTC**

  The Stop mode achieves the lowest power consumption with, while retaining the RAM, register contents and real time clock. The voltage regulator is in low-power mode. LSE or LSI is still running. All clocks in the V_{CORE} domain are stopped, the PLL, MSI RC, HSE crystal and HSI RC oscillators are disabled.

  Some peripherals featuring wake-up capability can enable the HSI RC during Stop mode to detect their wake-up condition. The device can be woken up from Stop mode by any of the EXTI line, in 3.5 µs, and the processor can serve the interrupt or resume the code.
- **Stop mode without RTC**

  This mode is identical to “Stop mode with RTC “, except for the RTC clock which is stopped here.
- **Standby mode with RTC**

  The Standby mode achieves the lowest power consumption with the real time clock running. The internal voltage regulator is switched off so that the entire V_{CORE} domain is powered off. The PLL, MSI RC, HSE crystal and HSI RC oscillators are also switched off. The LSE or LSI is still running.

After entering Standby mode, the RAM and register contents are lost except for registers in the Standby circuitry (wake-up logic, IWDG, RTC, LSI, LSE crystal 32 kHz oscillator, RCC_CSR register).

The device exits Standby mode in 60 µs when an external reset (NRST pin), an IWDG reset, a rising edge on one of the three WKUP pins, RTC alarm (Alarm A or Alarm B), RTC tamper event, RTC timestamp event or RTC wake-up event occurs.

- **Standby mode without RTC**

  This mode is identical to Standby mode with RTC, except that the RTC, LSE and LSI clocks are stopped.

  The device exits Standby mode in 60 µs when an external reset (NRST pin) or a rising edge on one of the three WKUP pin occurs.

Note: The RTC, the IWDG, and the corresponding clock sources are not stopped automatically by entering Stop or Standby mode. The LCD is not stopped automatically by entering Stop mode.

C.2 Power consumption ranges

STM32 MCUs power consumption can be further optimized thanks to the dynamic voltage scaling feature: the main internal regulator output voltage V12 that supplies the logic (CPU, digital peripherals, SRAM and flash memory) can be adjusted by software by selecting a power range (STM32L1 and STM32L0) or power scale (STM32 F4).

Power consumption range definitions are provided below (refer to STM32 MCU datasheets for full details).

C.2.1 STM32L1 series features three V_{CORE} ranges

- High performance **Range 1** (V_{DD} range limited to 2.0-3.6 V), with the CPU running at up to 32 MHz

  The voltage regulator outputs a 1.8 V voltage (typical) as long as the V_{DD} input voltage is above 2.0 V. Flash program and erase operations can be performed.
- Medium performance **Range 2** (full V_{DD} range), with a maximum CPU frequency of 16 MHz

  At 1.5 V, the flash memory is still functional but with medium read access time. Program and erase operations are still possible.
- Low performance **Range 3** (full V_{DD} range), with a maximum CPU frequency limited to 4 MHz (generated only with the multispeed internal RC oscillator clock source)

  At 1.2 V, the flash memory is still functional but with slow read access time. Program and erase operations are no longer available.

C.2.2 STM32F4 series features several V_{CORE} scales

The scale can be modified only when the PLL is OFF and when HSI or HSE is selected as system clock source.

- **Scale 1** (V12 voltage range limited to 1.26 - 1.40 V), default mode at reset.
  HCLK frequency range = 144 MHz to 168 MHz (180 MHz with over-drive).
  This is the default mode at reset.
- **Scale 2** (V12 voltage range limited to 1.20 - 1.32 V).
  HCLK frequency range is up to 144 MHz (168 MHz with over-drive).
- **Scale 3** (V12 voltage range limited to 1.08 - 1.20 V), default mode when exiting Stop mode.
  HCLK frequency ≤120 MHz.

The voltage scaling is adjusted to f_{HCLK} frequency as follows:

- **STM32F429x/39x MCUs**:
  - **Scale 1:** up to 168 MHz (up to 180 MHz with over-drive)
  - **Scale 2:** from 120 to 144 MHz (up to 168 MHz with over-drive)
  - **Scale 3:** up to 120 MHz.
- **STM32F401x MCUs**:
  No Scale 1
  - **Scale 2:** from 60 to 84 MHz
  - **Scale 3:** up to 60 MHz.
- **STM32F40x/41x MCUs**:
  - **Scale 1:** up to 168 MHz
  - **Scale 2:** up to 144 MHz

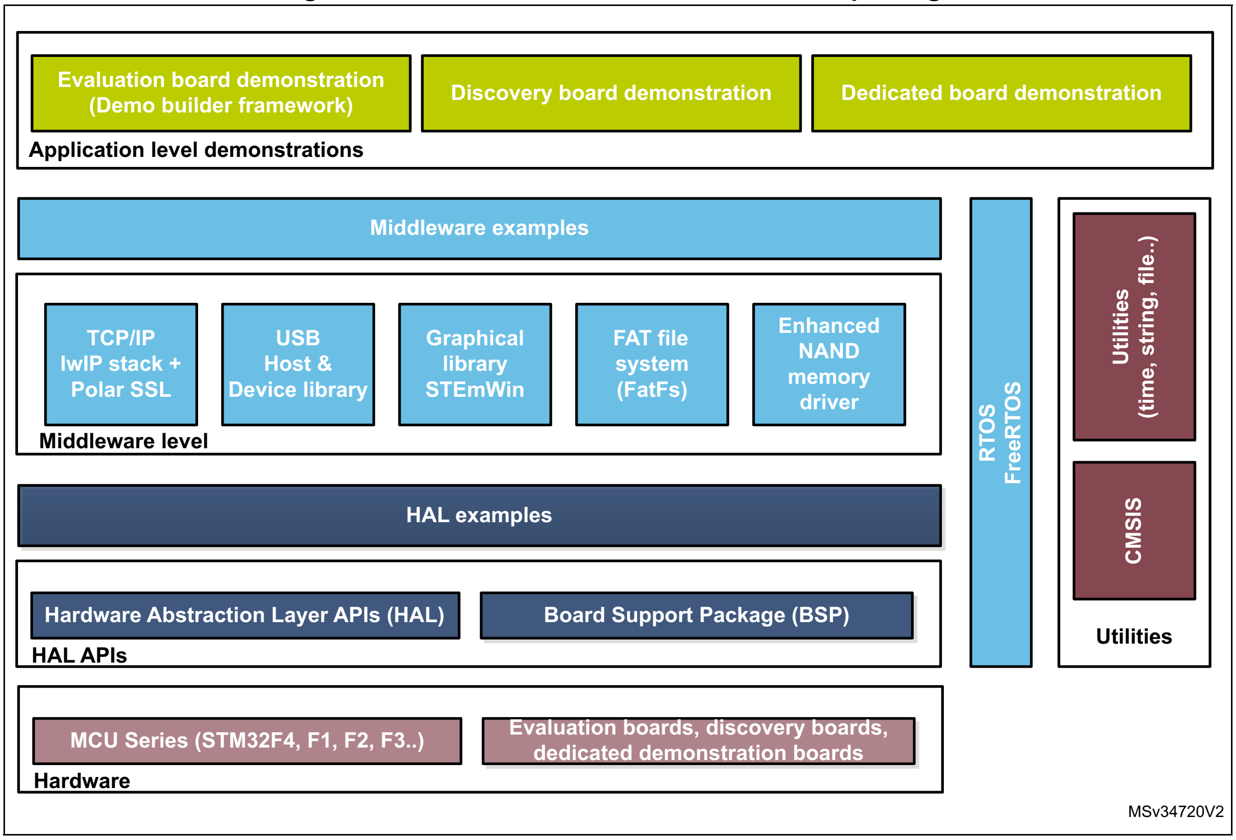
C.2.3 STM32L0 series features three V_{CORE} ranges

- Range 1 (V_{DD} range limited to 1.71 to 3.6 V), with CPU running at a frequency up to 32 MHz
- Range 2 (full V_{DD} range), with a maximum CPU frequency of 16 MHz
- Range 3 (full V_{DD} range), with a maximum CPU frequency limited to 4.2 MHz.

Appendix D STM32Cube embedded software packages

Along with STM32CubeMX C code generator, embedded software packages are part of STM32Cube initiative (refer to *DB2164 databrief*): these packages include a low-level hardware abstraction layer (HAL) that covers the microcontroller hardware, together with an extensive set of examples running on STMicroelectronics boards (see *Figure 697*). This set of components is highly portable across the STM32 series. The packages are fully compatible with STM32CubeMX generated C code.

Figure 697. STM32Cube Embedded Software package



Note: STM32CubeF0, STM32CubeF1, STM32CubeF2, STM32CubeF3, STM32CubeF4, STM32CubeL0 and STM32CubeL1 embedded software packages are available on st.com. They are based on STM32Cube release v1.1 (other series will be introduced progressively) and include the embedded software libraries used by STM32CubeMX for initialization C code generation.

The user should use STM32CubeMX to generate the initialization C code and the examples provided in the package to get started with STM32 application development.

Revision history

Table 30. Document revision history

Date	Revision	STM32CubeMX release number	Changes
17-Feb-2014	1	4.1	Initial release.
04-Apr-2014	2	4.2	Added support of STM32CubeF2 and STM32F2 Series in cover page, *Section 2.2: Key features*, *Section 5.14.1: Peripherals and Middleware Configuration window*, and *Appendix D: STM32Cube embedded software packages*. Updated *Section 11.1: Creating a new STM32CubeMX project*, *Section 11.2: Configuring the MCU pinout*, *Section 11.6: Configuring the MCU initialization parameters*. Section “Generating GPIO initialization C code move to *Section 8: Tutorial 3- Generating GPIO initialization C code (STM32F1 Series only)* and content updated. Added *Section 18.6: Why do I get the error “Java 8 update 45” when installing “Java 8 update 45” or a more recent version of the JRE?*.
24-Apr-2014	3	4.3	Added support of STM32CubeL0 and STM32L0 Series in cover page, *Section 2.2: Key features*, *Section 2.3: Rules and limitations* and *Section 5.14.1: Peripherals and Middleware Configuration window* Added board selection in *Table 13: File menu functions*, *Section 5.7.3: Pinout menu* and *Section 4.2: New Project window*. Updated *Table 15: Pinout menu*. Updated *Figure 321: Power Consumption Calculator default view* and added battery selection in *Section 5.3.1: Building a power consumption sequence*. Updated note in *Section 5.3: Power Consumption Calculator view* Updated *Section 11.1: Creating a new STM32CubeMX project*. Added *Section 19.7: Why does the RTC multiplexer remain inactive on the Clock tree view?*, *Section 19.8: How can I select LSE and HSE as clock source and change the frequency?*, and *Section 19.9: Why STM32CubeMX does not allow me to configure PC13, PC14, PC15, and PI8 as outputs when one of them is already configured as an output?*.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
19-Jun-2014	4	4.4	Added support of STM32CubeF0, STM32CubeF3, STM32F0 and STM32F3 Series in cover page, *Section 2.2: Key features*, *Section 2.3: Rules and limitations*, Added board selection capability and pin locking capability in *Section 2.2: Key features*, *Table 2: Home page shortcuts*, *Section 4.2: New Project window*, *Section 5.7: Toolbar and menus*, *Section 4.13: Set unused/reset used GPIOs windows*, *Section 4.11: Project Manager view*, and *Section 5.15: Pinout view*. Added *Section 5.15.1: Pinning and labeling signals on pins*. Updated *Section 5.16: Configuration view* and *Section 4.10: Clock Configuration view* and *Section 5.3: Power Consumption Calculator view*. Updated *Figure 50: STM32CubeMX Main window upon MCU selection*, *Figure 187: Project Settings window*, *Figure 302: About window*, *Figure 140: STM32CubeMX Pinout view*, *Figure 120: Chip view*, *Figure 321: Power Consumption Calculator default view*, *Figure 322: Battery selection*, *Figure 87: Building a power consumption sequence*, *Figure 324: Power consumption sequence: New Step default view*, *Figure 331: Power Consumption Calculator view after sequence building*, *Figure 332: Sequence table management functions*, *Figure 88: PCC Edit Step window*, *Figure 83: Power consumption sequence: new step configured (STM32F4 example)*, *Figure 329: ADC selected in Pinout view*, *Figure 330: Power Consumption Calculator configuration window: ADC enabled using import pinout*, *Figure 334: Description of the Results area*, *Figure 100: Peripheral power consumption tooltip*, *Figure 590: Power Consumption Calculation example*, *Figure 155: Sequence table* and *Figure 156: Power Consumption Calculation results*. Updated *Figure 142: STM32CubeMX Configuration view* and *Figure 39: STM32CubeMX Configuration view - STM32F1 Series* titles. Added STM32L1 in *Section 5.3: Power Consumption Calculator view*. Removed Figure *Add a new step using the PCC panel* from *Section 8.1.1: Adding a step*. Removed Figure *Add a new step to the sequence* from *Section 5.3.2: Configuring a step in the power sequence*. Updated *Section 8.2: Reviewing results*. Updated appendix *B.3.4: FatFs* and *Appendix C: STM32 microcontrollers power consumption parameters*. Added Appendix *C.1.3: STM32L0 series* and *C.2.3: STM32L0 series features three VCORE ranges*.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
19-Sep-2014	5	4.5	Added support of STM32CubeL1 Series in cover page, *Section 2.2: Key features*, *Section 2.3: Rules and limitations*, Updated *Section 3.2.3: Uninstalling STM32CubeMX standalone version*. Added off-line updates in *Section 3.4: Getting updates using STM32CubeMX*, modified *Figure 21: Embedded Software Packages Manager window*, and *Section 3.4.3: Installing STM32 MCU packages*. Updated *Section 4: STM32CubeMX user interface* introduction, *Table 2: Home page shortcuts* and *Section 4.2: New Project window*. Added *Figure 42: New Project window - Board selector*. Updated *Figure 196: Project Settings code generator*. Modified step 3 in *Section 4.11: Project Manager view*. Updated *Figure 39: STM32CubeMX Configuration view - STM32F1 Series*. Added STM32L1 in *Section 5.14.1: Peripherals and Middleware Configuration window*. Updated *Figure 83: GPIO configuration window - GPIO selection*; *Section 4.5.12: GPIO configuration window* and *Figure 88: DMA MemToMem configuration*. Updated introduction of *Section 4.10: Clock Configuration view*. Updated *Section 4.10.1: Clock tree configuration functions* and *Section 4.10.3: Recommendations*, *Section 5.3: Power Consumption Calculator view*, *Figure 324: Power consumption sequence: New Step default view*, *Figure 331: Power Consumption Calculator view after sequence building*, *Figure 83: Power consumption sequence: new step configured (STM32F4 example)*, and *Figure 330: Power Consumption Calculator configuration window: ADC enabled using import pinout*. Added *Figure 333: Power Consumption: Peripherals consumption chart* and updated *Figure 100: Peripheral power consumption tooltip*. Updated *Section 5.3.4: Power sequence step parameters glossary*. Updated *Section 6: STM32CubeMX C Code generation overview*. Updated *Section 11.1: Creating a new STM32CubeMX project* and *Section 11.2: Configuring the MCU pinout*. Added *Section 12: Tutorial 2 - Example of FatFs on an SD card using STM32429I-EVAL evaluation board* and updated *Section 8: Tutorial 3- Generating GPIO initialization C code (STM32F1 Series only)*. Updated *Section 5.3.2: Configuring a step in the power sequence*.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
19-Jan-2015	6	4.6	Complete project generation, power consumption calculation and clock tree configuration now available on all STM32 Series. Updated *Section 2.2: Key features* and *Section 2.3: Rules and limitations*. Updated Eclipse IDEs in *Section 3.1.3: Software requirements*. Updated *Figure 18: Updater Settings window*, *Figure 21: Embedded Software Packages Manager window* and *Figure 42: New Project window - Board selector*, Updated *Section 4.11: Project Manager view* and *Section 4.14: Update Manager windows*. Updated *Figure 302: About window*. Removed Figure *STM32CubeMX Configuration view - STM32F1 Series.* Updated *Table 17: STM32CubeMX Chip view - Icons and color scheme*. Updated *Section 5.14.1: Peripherals and Middleware Configuration window*. Updated *Figure 86: Adding a new DMA request* and *Figure 88: DMA MemToMem configuration*. Updated *Section 4.10.1: Clock tree configuration functions*. Updated *Figure 322: Battery selection*, *Figure 87: Building a power consumption sequence*, *Figure 88: PCC Edit Step window*. Added *Section 6.3: Custom code generation*. Updated *Figure 539: Clock tree view* and *Figure 544: Pinout & Configuration view*. Updated peripheral configuration sequence and *Figure 546: Timer 3 configuration window* in *Section 11.6.2: Configuring the peripherals*. Removed Tutorial 3: Generating GPIO initialization C code (STM32F1 Series only). Updated *Figure 550: GPIO mode configuration*. Updated *Figure 590: Power Consumption Calculation example* and *Figure 155: Sequence table*. Updated Appendix *A.1: Block consistency*, *A.2: Block inter-dependency* and *A.3: One block = one peripheral mode*. Appendix *A.4: Block remapping (STM32F10x only)*: updated *Section : Example*. Appendix *A.6: Block shifting (only for STM32F10x and when "Keep Current Signals placement" is unchecked)*: updated *Section : Example* Updated Appendix *A.8: Mapping a function individually*. Updated Appendix *B.3.1: Overview*. Updated Appendix *C.1.3: STM32L0 series*.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
19-Mar-2015	7	4.7	*Section 2.2: Key features*: removed *Pinout initialization C code generation for STM32F1 Series* from; updated *Complete project generation*. Updated *Figure 21: Embedded Software Packages Manager window*, *Figure 42: New Project window - Board selector*. Updated IDE list in *Section 4.11: Project Manager view* and modified *Figure 187: Project Settings window*. Updated *Section 4.10.1: Clock tree configuration functions*. Updated *Figure 183: STM32F469NIHx clock tree configuration view*. *Section 5.3: Power Consumption Calculator view*: added transition checker option. Updated *Figure 321: Power Consumption Calculator default view*, *Figure 322: Battery selection* and *Figure 87: Building a power consumption sequence*. Added *Figure 325: Enabling the transition checker option on an already configured sequence - All transitions valid*, *Figure 326: Enabling the transition checker option on an already configured sequence - At least one transition invalid* and *Figure 327: Transition checker option - Show log*. Updated *Figure 331: Power Consumption Calculator view after sequence building*. Updated *Section : Managing sequence steps*, *Section : Managing the whole sequence (load, save and compare)*. Updated *Figure 88: PCC Edit Step window* and *Figure 334: Description of the Results area*. Updated *Figure 590: Power Consumption Calculation example*, *Figure 155: Sequence table*, *Figure 156: Power Consumption Calculation results* and *Figure 158: Power consumption results - IP consumption chart*. Updated Appendix *B.3.1: Overview* and *B.3.5: FreeRTOS*.
28-May-2015	8	4.8	Added *Section 3.2.2: Installing STM32CubeMX from command line* and *Section 3.3.2: Running STM32CubeMX in command-line mode*.
09-Jul-2015	9	4.9	Added STLM32F7 and STM32L4 microcontroller Series. Added *Import project* feature. Added Import function in *Table 13: File menu functions*. Added *Section 4.12: Import Project window*. Updated *Figure 324: Power consumption sequence: New Step default view*, *Figure 88: PCC Edit Step window*, *Figure 83: Power consumption sequence: new step configured (STM32F4 example)*, *Figure 330: Power Consumption Calculator configuration window: ADC enabled using import pinout* and *Figure 87: Peripheral power consumption tooltip*. Updated command line to run STM32CubeMX in *Section 3.3.2: Running STM32CubeMX in command-line mode*. Updated note in *Section 5.16: Configuration view*. Added new clock tree configuration functions in *Section 4.10.1*. Updated *Figure 552: Middleware tooltip*. Modified code example in Appendix *B.1: STM32CubeMX generated C code and user sections*. Updated Appendix *B.3.1: Overview*. Updated generated .h files in Appendix *B.3.4: FatFs*.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
27-Aug-2015	10	4.10	Replace UM1742 by UM1940 in *Section : Introduction*. Updated command line to run STM32CubeMX in command-line mode in *Section 3.3.2: Running STM32CubeMX in command-line mode*. Modified *Table 1: Command line summary*. Updated board selection in *Section 4.2: New Project window*. Updated *Section 5.16: Configuration view* overview. Updated *Section 5.14.1: Peripherals and Middleware Configuration window*, *Section 4.5.12: GPIO configuration window* and *Section 4.5.13: DMA configuration window*. Added *Section 4.5.11: User Constants configuration window*. Updated *Section 4.10: Clock Configuration view* and added reserve path. Updated *Section 11.1: Creating a new STM32CubeMX project*, *Section 11.5: Configuring the MCU clock tree*, *Section 11.6: Configuring the MCU initialization parameters*, *Section 11.7.2: Downloading firmware package and generating the C code*, *Section 11.8: Building and updating the C code project*. Added *Section 11.9: Switching to another MCU*. Updated *Section 12: Tutorial 2 - Example of FatFs on an SD card using STM32429I-EVAL evaluation board* and replaced STM32F429I-EVAL by STM32429I-EVAL.
16-Oct-2015	11	4.11	Updated *Figure 21: Embedded Software Packages Manager window* and *Section 3.4.7: Checking for updates*. Character string constant supported in *Section 4.5.11: User Constants configuration window*. Updated *Section 4.10: Clock Configuration view*. Updated *Section 5.3: Power Consumption Calculator view*. Modified *Figure 590: Power Consumption Calculation example*. Updated *Section 13: Tutorial 3 - Using the Power Consumption Calculator to optimize the embedded application consumption and more*. Added Eclipse Mars in *Section 3.1.3: Software requirements*
03-Dec-2015	12	4.12	Code generation options now supported by the Project Settings menu. Updated *Section 3.1.3: Software requirements*. Added Project Settings in *Section 4.12: Import Project window*. Updated *Figure 201: Automatic project import*; modified *Manual project import* step and updated *Figure 202: Manual project import* and *Figure 203: Import Project menu - Try Import with errors*; modified third step of the import sequence. Updated *Figure 83: Clock Tree configuration view with errors*. Added mxconstants.h in *Section 6.1: STM32Cube code generation using only HAL drivers (default mode)*. Updated *Figure 590: Power Consumption Calculation example* to *Figure 599: Step 10 optimization*. Updated *Figure 600: Power sequence results after optimizations*.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
03-Feb-2016	13	4.13	Updated *Section 2.2: Key features*: – Information related to .ioc files. – Clock tree configuration – Automatic updates of STM32CubeMX and STM32Cube. Updated limitation related to STM32CubeMX C code generation in *Section 2.3: Rules and limitations*. Added Linux in *Section 3.1.1: Supported operating systems and architectures*. Updated Java Run Time Environment release number in *Section 3.1.3: Software requirements*. Updated *Section 3.2.1: Installing STM32CubeMX standalone version*, *Section 3.2.3: Uninstalling STM32CubeMX standalone version* and *Section 3.3.1: Downloading STM32CubeMX plug-in installation package*. Updated *Section 3.3.1: Running STM32CubeMX as a standalone application*. Updated *Section 4.11: Project Manager view* and *Section 4.14: Update Manager windows*. Updated *Section 5.15.1: Pinning and labeling signals on pins*. Added *Section 4.5.16: Setting HAL timebase source* Updated *Figure 143: Configuration window tabs for GPIO, DMA and NVIC settings (STM32F4 Series)*. Added note related to GPIO configuration in output mode in *Section 4.5.12: GPIO configuration window*; updated *Figure 83: GPIO configuration window - GPIO selection*. Modified *Figure 321: Power Consumption Calculator default view*, *Figure 86: Building a power consumption sequence*, *Figure 323: Step management functions*, *Figure 325: Enabling the transition checker option on an already configured sequence - All transitions valid*, *Figure 326: Enabling the transition checker option on an already configured sequence - At least one transition invalid*. Added import pinout button icon in *Section : Importing pinout*. Added *Section : Selecting/deselecting all peripherals*. Modified *Figure 331: Power Consumption Calculator view after sequence building*. Updated *Section : Managing the whole sequence (load, save and compare)*. Updated *Figure 334: Description of the Results area* and *Figure 100: Peripheral power consumption tooltip*. Updated *Figure 590: Power Consumption Calculation example* and *Figure 592: Sequence table*. Updated *Section 6.3: Custom code generation*. Updated *Figure 531: Pinout view with MCUs selection* and *Figure 532: Pinout view without MCUs selection window* in *Section 11.1: Creating a new STM32CubeMX project*. Updated *Section 11.6.2: Configuring the peripherals*. Updated *Figure 557: Project Settings and toolchain selection* and *Figure 558: Project Manager menu - Code Generator tab* in *Section 11.7.1: Setting project options*, and *Figure 559: Warning message for missing firmware package* in *Section 11.7.2: Downloading firmware package and generating the C code*.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
15-Mar-2016	14	4.14	Upgraded STM32CubeMX released number to 4.14.0. Added import of previously saved projects and generation of user files from templates in *Section 2.2: Key features*. Added MacOS in *Section 3.1.1: Supported operating systems and architectures*, *Section 3.2.1: Installing STM32CubeMX standalone version*, *Section 3.2.3: Uninstalling STM32CubeMX standalone version* and *Section 3.4.3: Running STM32CubeMX plug-in from Eclipse IDE*. Added command lines allowing the generation of user files from templates in *Section 3.3.2: Running STM32CubeMX in command-line mode*. Updated new library installation sequence in *Section 3.4.2: Updater configuration*. Updated *Figure 107: Pinout menus (Pinout tab selected)* and *Figure 108: Pinout menus (Pinout tab not selected)* in *Section 5.7.3: Pinout menu*. Modified *Table 16: Window menu*. Updated *Section 5.7: Output windows*. Updated *Figure 187: Project Settings window* and *Section 4.11.1: Project tab*. Updated *Figure 101: NVIC settings when using SysTick as HAL timebase, no FreeRTOS* and *Figure 102: NVIC settings when using FreeRTOS and SysTick as HAL timebase* in *Section 4.5.16: Setting HAL timebase source*. Updated *Figure 74: User Constants tab* and *Figure 75: Extract of the generated main.h file* in *Section 4.5.11: User Constants configuration window*. *Section 4.5.12: GPIO configuration window*: updated *Figure 83: GPIO configuration window - GPIO selection*, *Figure 84: GPIO configuration grouped by peripheral* and *Figure 85: Multiple pins configuration*. Updated *Section 4.5.14: NVIC configuration window*.
18-May-2016	15	4.15	Import project function is no more limited to MCUs of the same Series (see *Section 2.2: Key features*, *Section 5.7.1: File menu* and *Section 4.12: Import Project window*). Updated command lines in *Section 3.3.2: Running STM32CubeMX in command-line mode*. *Table 1: Command line summary*: modified all examples related to config comands as well as set dest_path <path> example. Added caution note for *Load Project* menu in *Table 13: File menu functions*. Updated *Generate Code* menu description in *Table 14: Project menu*. Updated *Set unused GPIOs* menu in *Table 15: Pinout menu*. Added case where FreeRTOS in enabled in *Section : Enabling interruptions using the NVIC tab view*. Added *Section 4.5.15: FreeRTOS configuration panel*. Updated Appendix *B.3.5: FreeRTOS* and *B.3.6: LwIP*.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
23-Sep-2016	16	4.17	Replaced *mxconstants.h* by *main.h* in the whole document. Updated *Introduction*, *Section 3.1.1: Supported operating systems and architectures* and *Section 3.1.3: Software requirements*. Added *Section 3.4.4: Installing STM32 MCU package patches*. Updated Load project description in *Table 2: Home page shortcuts*. Updated Clear Pinouts function in *Table 15: Pinout menu*. Updated *Section 4.11.3: Advanced Settings tab* to add Low Layer driver. Added *No check* and *Decimal and hexadecimal check* options in *Table 17: Peripheral and Middleware Configuration window buttons and tooltips*. Updated *Section : Tasks and Queues tab* and *Figure 98: FreeRTOS heap usage*. Updated *Figure 83: GPIO configuration window - GPIO selection*. Replaced PCC by Power Consumption Calculator in the whole document. Added *Section 6.2: STM32Cube code generation using Low Layer drivers*; updated *Table 26: LL versus HAL: STM32CubeMX generated source files* and *Table 27: LL versus HAL: STM32CubeMX generated functions and function calls*. Updated *Figure 561: Pinout view - Enabling the RTC*. Added *Section 14: Tutorial 4 - Example of UART communications with an STM32L053xx Nucleo board*. Added correspondence between STM32CubeMX release number and document revision.
21-Nov-2016	17	4.18	Removed Windows XP and added Windows 10 in *Section 3.1.3: Software requirements*. Updated *Section 3.2.3: Uninstalling STM32CubeMX standalone version*. Added setDriver command line in *Table 1: Command line summary*. Added List pinout compatible MCUs feature: – Updated *Table 15: Pinout menu*. – Added *Section 15: Tutorial 5: Exporting current project configuration to a compatible MCU* Added Firmware location selection option in *Section 4.11.1: Project tab* and *Figure 187: Project Settings window*. Added Restore Default feature: – Updated *Table 8: Peripheral and Middleware configuration window buttons and tooltips* – Updated *Figure 76: Using constants for peripheral parameter settings*.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
12-Jan-2017	18	4.19	Project import no more limited to microcontrollers belonging to the same Series: updated *Introduction*, *Figure 201: Automatic project import*, *Figure 202: Manual project import*, *Figure 203: Import Project menu - Try Import with errors* and *Figure 204: Import Project menu - Successful import after adjustments*. Modified Appendix *B.3.4: FatFs*, *B.3.5: FreeRTOS* and *B.3.6: LwIP*. Added Appendix *B.3.7: Libjpeg*.
02-Mar-2017	19	4.20	*Table 17: STM32CubeMX Chip view - Icons and color scheme*: – Updated list of alternate function example. – Updated example and description corresponding to function mapping on a pin. – Added example and description for analog signals sharing the same pin. Updated *Figure 87: Peripheral Configuration window (STM32F4 Series)*, *Figure 74: User Constants tab*, *Figure 80: Consequence when deleting a user constant for peripheral configuration*, *Figure 81: Searching for a name in a user constant list* and *Figure 82: Searching for a value in a user constant list*. Added *Section 5.3.6: SMPS feature*. Added *Section 6.4: Additional settings for C project generation*. Added STM32CubeF4 to the list of packages that include Libjpeg in *Appendix B.3.7: Libjpeg*.
05-May-2017	20	4.21	Minor modifications in *Section 1: STM32Cube overview*. Updated *Figure 40: New Project window - MCU selector* and *Figure 187: Project Settings window*. Updated description of Project Settings in *Section 4.11.1: Project tab*. Updated *Figure 199: Advanced Settings window*. In Appendix *B.3.7: Libjpeg*, added STM32CubeF2 and STM32CubeH7 in the list of software packages in which Libjpeg is embedded. Modified *Figure 697: STM32Cube Embedded Software package* look-and-feel.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
06-Jul-2017	21	4.22	Added STM32H7 to the list of supported STM32 Series. Added MCU data and documentation refresh capability in *Section 3.4: Getting updates using STM32CubeMX* and updated *Figure 18: Updater Settings window*. Added capability to identify close MCUs in *Section 4.2: New Project window*, updated *Figure 40: New Project window - MCU selector*, added *Figure 29: New Project window - MCU list with close function* and *Figure 30: New Project window - List showing close MCUs.*, updated *Figure 530: MCU selection*. Updated *Figure 50: STM32CubeMX Main window upon MCU selection*. Added Rotate clockwise/Counter clockwise and Top/Bottom view in *Table 15: Pinout menu*. Added *Section 4.1.4: Social links*. Updated *Figure 342: Configuring the SMPS mode for each step*. Updated *Section 6.2: STM32Cube code generation using Low Layer drivers*. Updated *Figure 557: Project Settings and toolchain selection*.
05-Sep-2017	22	4.22.1	Added STM32L4+ Series in *Introduction*, *Section 5.3: Power Consumption Calculator view* and *Section 6.2: STM32Cube code generation using Low Layer drivers*. Added guidelines to run STM32CubeMX on MacOS in *Section 3.3.1: Running STM32CubeMX as a standalone application*. Removed MacOS from *Section 3.4.3: Running STM32CubeMX plug-in from Eclipse IDE*. Added *Section 19.10: Ethernet configuration: why cannot I specify DP83848 or LAN8742A in some cases?*
18-Oct-2017	23	4.23	Added *Section 1: General information*. Renamed **Display close** button into **Display similar items** in *Section 4.2: New Project window*. Added **Refresh Data** and **Docs & Resources** menus in *Section 5.7.5: Help menu*. Added STM32F2, STM32F4 and STM32F7 Series in *Section 6.2: STM32Cube code generation using Low Layer drivers*. Added Appendix *B.3.8: Mbed TLS*. Updated STM32CubeMX release number corresponding to user manual revision 22.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
16-Jan-2018	24	4.24	Replaced “STM32Cube firmware package” by “STM32Cube MCU package”. Updated *Section 1: STM32Cube overview*. Updated MacOS in *Section 3.1.1: Supported operating systems and architectures*. Updated Eclipse requirements in *Section 3.1.3: Software requirements*. *Section 3.4: Getting updates using STM32CubeMX*: – updated section introduction – updated *Figure 13: Connection Parameters tab - No proxy* – *Section 3.4.3* renamed into “Installing STM32 MCU packages” and updated. – renamed *Section 3.4.4* into “Installing STM32 MCU package patches” – added *Section 3.4.5: Installing embedded software packs* – updated *Section 3.4.7: Checking for updates* Updated *Figure 42: New Project window - Board selector*. Updated *Figure 51: STM32CubeMX Main window upon board selection (peripherals not initialized)* and introductory sentence. Updated *Figure 52: STM32CubeMX Main window upon board selection (peripherals initialized with default configuration)* and introductory sentence. Added “Select additional software components” menu in *Table 14: Project menu*. “Install new libraries” menu renamed “Manage embedded software packages” and corresponding description updated in *Table 17: Help menu*. Updated *Section 3.4.6: Removing already installed embedded software packages*. Updated *Section 4.14: Update Manager windows* Added *Section 4.15: Software Packs component selection window*. Added pin stacking function in *Table 17: STM32CubeMX Chip view - Icons and color scheme*. *Section 6.2: STM32Cube code generation using Low Layer drivers*: added STM32F0, STM32F3, STM32L0 in the list of product Series supporting low-level drivers. *Section 12: Tutorial 2 - Example of FatFs on an SD card using STM32429I-EVAL evaluation board*: updated *Figure 582: Board selection* and modified step 6 of the sequence for generating a project and running tutorial 2. *Section 14: Tutorial 4 - Example of UART communications with an STM32L053xx Nucleo board*: updated *Figure 601: Selecting NUCLEO_L053R8 board*. Added *Section 16: Tutorial 6 – Adding embedded software packs to user projects*.
16-Jan-2018	24 (cont’d)	4.24	Added Appendix *B.3.9: TouchSensing* and *B.3.10: PDM2PCM*. *Section 4.5.14: NVIC configuration window*/Default initialization sequence of interrupts: changed color corresponding to interrupt enabling code from green to black bold.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
07-Mar-2018	25	4.25	Updated *Introduction*, *Section 1: STM32Cube overview*, *Section 2.3: Rules and limitations*, *Section 3.2.1: Installing STM32CubeMX standalone version*, *Section 4: STM32CubeMX user interface*, *Section 4.11.1: Project tab* and *Section 5.13.1: Peripheral and Middleware tree panel*. Minor text edits across the whole document. Updated *Table 13: File menu functions* and *Table 12: Relations between power over-drive and HCLK frequency*. Updated *Figure 40: New Project window - MCU selector*, *Figure 27: Enabling graphics choice in MCU selector*, *Figure 187: Project Settings window*, *Figure 193: Selecting a different firmware location*, *Figure 77: Enabling STemWin framework*, *Figure 116: Configuration view for Graphics*, *Figure 562: Pinout view - Enabling LSE and HSE clocks* and *Figure 563: Pinout view - Setting LSE/HSE clock frequency*. Added *Export to Excel*, *Show favorite MCUs* and *Section 4.4.16: Graphics frameworks and simulator*. Added *Section 17: Tutorial 8 – Using STemWin Graphics framework*, *Section 18: Tutorial 9: Using STM32CubeMX Graphics simulator* and their subsections. Added *Section B.3.11: Graphics*.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
05-Sep-2018	26	4.27	Updated STM32Cube logo on cover page. Replaced STMCube™ by STM32Cube™ in the whole document. Updated *Section 1: STM32Cube overview*. Updated *Figure 1: Overview of STM32CubeMX C code generation flow*. Updated *Section 2.2: Key features* to add new features: graphic simulator feature, Support of embedded software packages in CMSIS-Pack format and Contextual Help. Changed *Section 3.4* title into “Getting updates using STM32CubeMX”. Suppressed figures *Connection Parameters tab - No proxy* and *Connection Parameters tab - Use System proxy parameters*. Updated *Figure 22: Managing embedded software packages - Help menu*. In *Section 3.4.5: Installing embedded software packs*, updated step 3f of the embedded software pack installation sequence and added *Figure 27: License agreement acceptance*. *Section 4.2: New Project window*: updated *Figure 40: New Project window - MCU selector*, *Figure 41: Marking an MCU as favorite* and *Figure 42: New Project window - Board selector*. *Section 5.7.1: File menu*: added caution note for New Project in *Table 13: File menu functions*. Updated *Figure 107: Pinout menus (Pinout tab selected)* and *Figure 108: Pinout menus (Pinout tab not selected)*. *Section 4.11: Project Manager view*: – Added note related to project saving (step 3). – Updated *Figure 187: Project Settings window* – Updated *Section 4.11.1: Project tab* and *Figure 193: Selecting a different firmware location*. Added *Section 4.15.4: Component dependencies panel*, *Contextual help*, *Section 10: Support of additional software components using CMSIS-Pack standard* and *Section 17: Tutorial 7 – Using the X-Cube-BLE1 software pack*.
12-Nov-2018	27	4.28	Updated *Section 3.4.3: Installing STM32 MCU packages*, *Section 3.4.5: Installing embedded software packs*, *Section 3.4.6: Removing already installed embedded software packages*, *Section 3.4.7: Checking for updates* and the figures in it. Updated *Section 4: STM32CubeMX user interface*, its subsections and the figures and the tables in them. Updated *Section 10: Support of additional software components using CMSIS-Pack standard*, sections *11.6.1* to *11.6.5*, *Section 11.7.1: Setting project options*, *Section 11.7.2: Downloading firmware package and generating the C code*, *Section 11.8: Building and updating the C code project*, *Section 11.9: Switching to another MCU*, *Section 12: Tutorial 2 - Example of FatFs on an SD card using STM32429I-EVAL evaluation board* and the figures in it, *Section 15: Tutorial 5: Exporting current project configuration to a compatible MCU* and the figures in it, *Section 16: Tutorial 6 – Adding embedded software packs to user projects* and *Section 17: Tutorial 7 – Using the X-Cube-BLE1 software pack*.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
12-Nov-2018	27 (cont'd)	5.0	Added *Section 19: Tutorial 10: Using ST-TouchGFX framework* and its subsections. Updated *Table 27: LL versus HAL: STM32CubeMX generated functions and function calls*. Removed former *Figure 164: Enabling and configuring a CMSIS-Pack software component*, *Figure 192: FatFs peripheral instances*, *Figure 213: Project Import status*, *Figure 254: Saving software component selection as user preferences* and *Figure 268: Configuring X-Cube-BLE1*. Updated *Figure 1: Overview of STM32CubeMX C code generation flow*, *Figure 16: STM32Cube installation wizard*, *Figure 7: Closing STM32CubeMX perspective*, *Figure 9: Opening Eclipse plug-in*, *Figure 10: STM32CubeMX perspective*, *Figure 335: Overall peripheral consumption*, *Figure 504: User constant generating define statements*, *Figure 527: Selecting a CMSIS-Pack software component*, *Figure 528: Enabling and configuring a CMSIS-Pack software component*, *Figure 529: Project generated with CMSIS-Pack software component*, *Figure 530: MCU selection*, *Figure 531: Pinout view with MCUs selection*, *Figure 532: Pinout view without MCUs selection window*, *Figure 534: Timer configuration*, *Figure 535: Simple pinout configuration*, *Figure 536: Save Project As window*, *Figure 537: Generate Project Report - New project creation*, *Figure 538: Generate Project Report - Project successfully created*, *Figure 539: Clock tree view*, *Figure 544: Pinout & Configuration view*, *Figure 545: Case of Peripheral and Middleware without configuration parameters*, *Figure 546: Timer 3 configuration window*, *Figure 547: Timer 3 configuration*, *Figure 548: Enabling Timer 3 interrupt*, *Figure 549: GPIO configuration color scheme and tooltip*, *Figure 550: GPIO mode configuration*, *Figure 551: DMA parameters configuration window*, *Figure 552: Middleware tooltip*, *Figure 553: USB Host configuration*, *Figure 553: USB Host configuration*, *Figure 554: FatFs over USB mode enabled*, *Figure 555: System view with FatFs and USB enabled*, *Figure 556: FatFs define statements*, *Figure 557: Project Settings and toolchain selection*, *Figure 558: Project Manager menu - Code Generator tab*, *Figure 559: Warning message for missing firmware package*, *Figure 527: Connection Parameters tab*, *Figure 561: Updated settings with connection*, *Figure 562: Downloading the firmware package*, *Figure 563: Unzipping the firmware package*, *Figure 564: C code generation completion message*, *Figure 580: Import Project menu*, *Figure 610: Project Settings menu*, *Figure 621: Additional software components enabled for the current project*, *Figure 622: Pack software components: no configurable parameters*, *Figure 623: Pack tutorial: project settings*, *Figure 626: Embedded software packages*, *Figure 628: Installing Embedded software packages*, *Figure 629: Starting a new project - selecting the NUCLEO-L053R8 board*, *Figure 630: Starting a new project - initializing all peripherals*, *Figure 631: Selecting X-Cube-BLE1 components*, *Figure 632: Configuring peripherals and GPIOs*, *Figure 633: Configuring NVIC interrupts*, *Figure 634: Enabling X-Cube-BLE1*, *Figure 634: Enabling X-Cube-BLE1*, *Figure 635: Configuring the SensorDemo project* and *Figure 312: Graphics simulator user interface*.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
19-Feb-2019	28	5.0	Updated *Introduction*, *Section 1: STM32Cube overview*, *Section 2.2: Key features*, *Section 3.1.3: Software requirements*, *Section 3.4.3: Installing STM32 MCU packages*, *Section 4: STM32CubeMX user interface*, *Resolving pin conflicts*, *Section 4.5.10: Component configuration panel*, *Section 4.10: Clock Configuration view*, *Section 4.11: Project Manager view*, *Section 4.11.1: Project tab*, *Section 4.11.3: Advanced Settings tab*, *Using the transition checker*, *Section 9.2: STM32CubeMX Device tree generation*, *Section 6.3.2: Saving and selecting user templates*, *.extSettings file example and generated outcomes* and *Section 11.6.4: Configuring the DMAs*. Added *Section 4.6: Pinout & Configuration view for STM32 MPUs*, *Section 4.6.2: Boot stages configuration*, *Section 5: STM32CubeMX tools*, *Section 9: Device tree generation (STM32MPUs only)*, *Section B.3.11: STM32WPAN BLE/Thread (STM32WB series only)*, *Section B.3.13: OpenAmp and RESMGR_UTILITY (STM32MPUs and STM32H7 dual-core products)* and their subsections. Removed former *Section 1: General information*. Updated *Table 2: Home page shortcuts*, *Table 5: Component list, mode icons and color schemes*, *Table 6: Pinout menu and shortcuts* and title of *Table 9: Clock configuration view widgets*. Updated *Figure 187: Project Settings window*, *Figure 189: Project folder*, *Figure 193: Selecting a different firmware location*, *Figure 201: Automatic project import*, *Figure 202: Manual project import*, *Figure 203: Import Project menu - Try Import with errors*, *Figure 204: Import Project menu - Successful import after adjustments*, *Figure 215: Set unused pins window*, *Figure 216: Reset used pins window*, *Figure 302: About window*, *Figure 525: STM32CubeMX generated DTS – Extract 3*, *Figure 527: Selecting a CMSIS-Pack software component*, *Figure 528: Enabling and configuring a CMSIS-Pack software component*, *Figure 587: FATFS tutorial - Project settings* and *Figure 588: C code generation completion message*.
16-Apr-2019	29	5.1	Updated *Introduction*. *Section 3.1.3: Software requirements*, *Section 4.2: New Project window*, *MCU close selector feature*, *External clock sources*, *Importing pinout*, *Selecting/deselecting all peripherals*, *Section 4.6: Pinout & Configuration view for STM32 MPUs*, *Section 4.15: Software Packs component selection window*, *Section 5.4.1: DDR configuration*, *Section 6.2: STM32Cube code generation using Low Layer drivers*, *BLE configuration* and *Section B.3.13: OpenAmp and RESMGR_UTILITY (STM32MPUs and STM32H7 dual-core products)*. Added *Section 4.2.1: MCU selector*, *Section 4.2.2: Board selector*, *Section 4.2.4: Cross selector*, *Section 4.8: Pinout & Configuration view for STM32H7 dual-core products*, *Section 5.3.9: Example feature (STM32MPUs and STM32H7 dual-core only)* and *Section 7: Code generation for dual-core MCUs (STM32H7 dual-core product lines only)*. Removed former *Section 3.3: Installing STM32CubeMX plug-in version* and its subsections, and former *Section 3.4.3: Running STM32CubeMX plug-in from Eclipse IDE*.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
16-Apr-2019	29 (cont'd)	5.1	Updated *Table 3: Window menu*. Updated figures *27* to *42*, *Figure 199: Advanced Settings window*, figures *321* to *328*, *330* to *333* and *335* to *344*, *Figure 557: Project Settings and toolchain selection* and figures *590* to *600*, Added *Figure 37: New Project window shortcuts*, *Figure 105: STM32MPUs: assignment options for GPIOs*, *Figure 696: Resource Manager: peripheral assignment view* and *Figure 697: STM32Cube Embedded Software package*.
01-Oct-2019	30	5.2	Updated *Introduction*. *Section 2.2: Key features*, *Section 3.3.2: Running STM32CubeMX in command-line mode*, *Part number selection*, *Section 4.15: Software Packs component selection window*, *Section 4.15.1: Introduction on software components*, *Section 4.15.2: Filter panel*, *Section 4.15.3: Packs panel*, *Section 4.15.4: Component dependencies panel*, *Section 4.15.6: Updating the tree view for additional software components*, *Section 5.3: Power Consumption Calculator view* and *Section 6.2: STM32Cube code generation using Low Layer drivers*. Updated *Table 1: Command line summary*, *Table 6: Pinout menu and shortcuts*, *Table 19: Additional Software window – Packs panel icons* and *Table 20: Component dependencies panel contextual help*. Updated *Figure 33: STM32CubeMX home page*, *Figure 222: Selection of additional software components*, *Figure 223: Additional software components - Updated tree view*, *Figure 527: Selecting a CMSIS-Pack software component* and *Figure 631: Selecting X-Cube-BLE1 components*. Added *Section 4.5.8: Pinout for multi-bonding packages* and *Section 4.15.5: Details and Warnings panel*. Added *Table 18: Additional Software window – Packs panel columns*

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
13-Dec-2019	31	5.4	Updated *Introduction*, *Section 1: STM32Cube overview*, *Section 4.2: New Project window*, *MCU/MPU selection for a new project* and *Section 11.7.1: Setting project options*. Added *Section 4.9: Enabling security in Pinout & Configuration view (STM32L5 and STM32U5 series only)* with its subsections, *Section 4.10.2: Securing clock resources (STM32L5 series only)* and *Section 8: Code generation with TrustZone enabled (STM32L5 series only)*. Removed former *Section 4.4.16: Graphics frameworks and simulator*, *Section 17: Tutorial 8 – Using STemWin Graphics framework*, *Section 18: Tutorial 9: Using STM32CubeMX Graphics simulator*, *Section 19: Tutorial 10: Using ST-TouchGFX framework* and *Section B.3.11: Graphics*. Minor text edits across the whole document. Updated *Table 1: Command line summary*. Updated *Figure 68: Pinout view: MCUs with multi-bonding*, *Figure 69: Pinout view: multi-bonding with extended mode*, *Figure 105: STM32MPUs: assignment options for GPIOs*, *Figure 187: Project Settings window*, *Figure 352: DDR Suite - Connection to target*, *Figure 353: DDR Suite - Target connected*, *Figure 354: DDR activity logs*, *Figure 355: DDR interactive logs*, *Figure 356: DDR register loading*, *Figure 357: DDR test list from U-Boot SPL*, *Figure 358: DDR test suite results*, *Figure 359: DDR tests history*, *Figure 175: DDR tuning pre-requisites*, *Figure 176: DDR tuning process*, *Figure 177: Bit deskew*, *Figure 178: Eye training (centering) panel*, *Figure 179: DDR Tuning - saving to configuration*, *Figure 522: Project settings for STM32CubeIDE toolchain* and *Figure 557: Project Settings and toolchain selection*. Added *Figure 38: Enabling TrustZone®*.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
10-Jul-2020	32	6.0	Updated *Section 2.2: Key features*, *Section 3.1.1: Supported operating systems and architectures*, *Section 3.1.3: Software requirements*, *Section 3.2.1: Installing STM32CubeMX standalone version*, *Section 3.4: Getting updates using STM32CubeMX*, *Section 3.4.5: Installing embedded software packs*, *Section 4.2: New Project window*, *Export to Excel*, *Section 4.5: Pinout & Configuration view*, *Section 4.11.3: Advanced Settings tab* and *Section 18.6: Why do I get the error “Java 8 update 45” when installing “Java 8 update 45” or a more recent version of the JRE?*. Added *Section 4.2.3: Example selector*, *Section 5.1: External Tools*, *Section 19.2: Since I changed my login to access the Internet, some software packs appear not available.* and *Section 19.3: On dual-context products, why some peripherals or middleware are not available for a given context?*. Removed former *MCU selection based on graphics criteria*. Updated *Table 4: Help menu shortcuts* and *Table 17: Additional software window - Filter icons*. Updated *Figure 33: STM32CubeMX home page*, *Figure 37: New Project window shortcuts*, *Figure 42: New Project window - Board selector*, *Figure 45: Cross selector - Data refresh prerequisite*, *Figure 199: Advanced Settings window*, *Figure 219: Additional software window*, *Figure 200: Device tree generation for the Linux kernel*, *Figure 201: STM32CubeMX Device tree generation for U-boot*, *Figure 202: STM32CubeMX Device tree generation for TF-A*, *Figure 631: Selecting X-Cube-BLE1 components* and *Figure 306: Java Control Panel*.
10-Nov-2020	33	6.1	Updated *Introduction*, *Section 3.1.3: Software requirements*, *Section 3.4.7: Checking for updates*, *Section 4.15.3: Packs panel*, *Section 5.1: External Tools*, *Section 12: Tutorial 2 - Example of FatFs on an SD card using STM32429I-EVAL evaluation board* and *Section 18.6: Why do I get the error “Java 8 update 45” when installing “Java 8 update 45” or a more recent version of the JRE?*. Added *Choosing not to generate code for some peripherals or middlewares*. Updated *Table 1: Command line summary*. Updated *Figure 32: Help menu: checking for updates*, *Figure 33: STM32CubeMX home page*, *Figure 199: Advanced Settings window*, *Figure 219: Additional software window*, *Figure 303: ST Tools* and *Figure 583: SDIO peripheral configuration*.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
12-Feb-2021	34	6.2	Updated *Section 3.1.1: Supported operating systems and architectures*, *Section 3.1.3: Software requirements*, *Section 3.2.1: Installing STM32CubeMX standalone version*, *Section 3.2.2: Installing STM32CubeMX from command line*, *Section 3.2.3: Uninstalling STM32CubeMX standalone version*, *Section 3.3.2: Running STM32CubeMX in command-line mode*, *Warning:* in *Section 3.4.7: Checking for updates*, *Section 4.1: Home page*, *Section 4.15: Software Packs component selection window*, *Section 4.15.2: Filter panel*, *Section 4.15.3: Packs panel*, *Section 4.15.4: Component dependencies panel*, *Section 4.15.5: Details and Warnings panel* and *Section 12: Tutorial 2 - Example of FatFs on an SD card using STM32429I-EVAL evaluation board*. Updated *Table 6: Pinout menu and shortcuts*. Added *Figure 2: Full disk access for macOS* and *Figure 220: Component dependency resolution*. Updated *Figure 33: STM32CubeMX home page*, *Figure 38: Enabling TrustZone®*, *Figure 219: Additional software window*. Removed former *Figure 5: Auto-install command line* and former *Section 18.6: Why do I get the error “Java 8 update 45” when installing “Java 8 update 45” or a more recent version of the JRE?*.
22-Jun-2021	35	6.3	Updated *Section 3.1.1: Supported operating systems and architectures*, *Section 3.1.3: Software requirements*, *Section 4.2: New Project window*, *Section 4.3: Project page*, *Section 4.5.5: Pinout view advanced actions*, *Section 4.9: Enabling security in Pinout & Configuration view (STM32L5 and STM32U5 series only)* and code in *Section 12: Tutorial 2 - Example of FatFs on an SD card using STM32429I-EVAL evaluation board*. Added *Figure 39: Adjusting selector results* and *Section 19.1: I encountered a network connection error during a download from STM32CubeMX*. Updated *Table 1: Command line summary*, *Table 19: Additional Software window – Packs panel icons* and *Table 20: Component dependencies panel contextual help*. Updated *Figure 527: Selecting a CMSIS-Pack software component* and *Figure 631: Selecting X-Cube-BLE1 components*.
05-Nov-2021	36	6.4	Updated *Section 2.2: Key features*, *Section 3.3.1: Running STM32CubeMX as a standalone application*, *Section 3.4: Getting updates using STM32CubeMX*, *Section 4.2: New Project window*, *Enabling interruptions using the NVIC tab view*, *Section 4.9: Enabling security in Pinout & Configuration view (STM32L5 and STM32U5 series only)*, *Section 4.11.1: Project tab* and *Section 5.3.7: Bluetooth Low-Energy®/ZigBee® support (STM32WB series only)*. Added *Section 3.4.1: Running STM32CubeMX behind a proxy server* and *Section 5.3.8: Sub-GHz support (STM32WL series only)*. Updated *Figure 89: NVIC configuration tab - FreeRTOS disabled*.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
18-Feb-2022	37	6.5	Updated *Introduction* and *Section 3.1.1: Supported operating systems and architectures*. Added *Section 18: Creating LPBAM projects* with its subsections, and *Section 19.11: How to fix MX_DMA_Init call rank in STM32CubeMX generated projects?*. Minor text edits across the whole document.
14-Jun-2022	38	6.6	Updated *Introduction*, *Section 2.2: Key features*, *Section 3.3.2: Running STM32CubeMX in command-line mode*, *Boot loader (A7 FSBL) peripherals selection*, *Section 4.11.1: Project tab*, *Section 4.16: LPBAM Scenario & Configuration view*, *Section 9.1: Device tree overview*, and *Section 9.2: STM32CubeMX Device tree generation*. Updated *Table 1: Command line summary*. Updated *Figure 302: About window*. Added *Section 4.17: CAD Resources view* and *Section 18.6: LPBAM application for TrustZone activated projects*. Removed former *Section 9.2.1: Device tree generation for Linux kernel*, *Section 9.2.2: Device tree generation for U-boot*, and *Section 9.2.3: Device tree generation for TF-A*. Minor text edits across the whole document.
17-Nov-2022	39	6.7	Updated *Section 2.2: Key features* and *Section 17: Tutorial 7 – Using the X-Cube-BLE1 software pack*. Added *Section 19.12: When is the PeriphCommonClock_Config() function generated?* and *Section 19.13: How to handle thread-safe solution in STM32CubeMX and STM32CubeIDE?*. Updated *Figure 40: New Project window - MCU selector*, *Figure 41: Marking an MCU as favorite*, *Figure 29: New Project window - MCU list with close function*, *Figure 30: New Project window - List showing close MCUs*, and *Figure 302: About window*. Minor text edits across the whole document.
21-Feb-2023	40	6.8	Updated *Section 3.2.1: Installing STM32CubeMX standalone version*, *Section 3.3.2: Running STM32CubeMX in command-line mode*, *Section 3.4.1: Running STM32CubeMX behind a proxy server*, and *Section 4.11.1: Project tab*. Added *Section 4.18: Boot path* and its subsections. Removed former *Section 5.3.4: DDR tuning* and *DDR tuning tab (read-only)*. Updated *Figure 40: New Project window - MCU selector*, *Figure 187: Project Settings window*, and *Figure 659: Design check*. Minor text edits across the whole document.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
03-Jul-2023	41	6.9	Updated *Introduction*, *Section 3.1.1: Supported operating systems and architectures*, *Java™ Runtime Environment*, *Section 4.15: Software Packs component selection window*, *Section 4.18: Boot path*, *Section 4.18.2: Creating a boot path project: an example*, *Section 4.18.5: How to configure an ST-iRoT with a secure manager NS application boot path*, and note in *Section 18.4: Checking the LPBAM design*. Updated *Table 1: Command line summary*. Added note to *Section 9.2: STM32CubeMX Device tree generation*. Added figures *235* to *239* and *Figure 272: Code generated with secure manager API*. Added *Section 4.18.5: How to configure an assembled boot path*, *Section 4.19: User authentication*, *Section 4.18: STM32CubeMX Memory Management Tool* and their subsections, and *Section B.3.12: CMSIS packs selection limitation*. Updated *Figure 45: Cross selector - Data refresh prerequisite*, *Figure 231: Boot paths for STM32H57x devices*, *Figure 241: Select the STM32H5 device*, *Figure 243: Boot paths for STM32H56x devices*, figures *246* to *259*, figures *167* to *270*, figures *272* to *274*, figures *263* to *267*, *Figure 271: Secure manager API configuration*, and *Figure 302: About window*. Minor text edits across the whole document.
08-Sep-2023	42	6.9.2	Updated for the replacement of “boot path settings” with “boot path and debug authentication” in – *Section 4.18.4: How to configure an ST-iRoT boot path* – *Section 4.18.5: How to configure an ST-iRoT with a secure manager NS application boot path* – *Figure 255*, *Figure 269*, and *Figure 266* titles Updated *Figure 266: Boot path and Debug Authentication tab*. Updated figures *231* to *239* in *Section 4.18.1: Available boot paths*. Updated *Section 1: STM32Cube overview*. Minor text edits across the whole document.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
20-Nov-2023	43	6.10.0	Updated *Section 4.11: Project Manager view*, *Section 4.18.5: How to configure an ST-iRoT with a secure manager NS application boot path*, *Step 3: OEMiROT (assembled) code generation*, *Step 6: Authentication and encryption keys regeneration, option byte file generation*, and *Section 4.18: STM32CubeMX Memory Management Tool*. Added *Section 4.19.3: Password restoration*. Removed former *MCU close selector feature*. Updated *Table 21: Boot paths without TrustZone (TZEN = 0)* and *Table 22: Boot paths with TrustZone (TZEN = 1)*. Updated *Figure 235: Application boot path (OEM-uRoT assembled)*, *Figure 236: Application boot path: ST-iRoT and uRoT secure/nonsecure project*, *Figure 238: Application boot path: ST-iRoT dual figure*, *Figure 253: Project provisioning*, *Figure 255: Boot path and debug authentication panel*, *Figure 262: IDE post build commands*, *Figure 273: IDE post build commands*, *Figure 270: IDE post build commands*, figures *350* to *353*, *Figure 356: DDR register loading*, and *Figure 357: DDR test list from U-Boot SPL*. Removed former *Figure 167: Selection of the OEMiRoT_Boot project* and *Figure 195: Generated project*. Minor text edits across the whole document.
13-Mar-2024	44	6.11.0	Updated *Section 3.1.1: Supported operating systems and architectures*, *Section 3.2.1: Installing STM32CubeMX standalone version*, *Section 3.2.2: Installing STM32CubeMX from command line*, *Uninstalling STM32CubeMX on Windows*, *Feature: MMT usage, Pinout, and Configuration UI*, and *Section 4.18.5: How to configure an assembled boot path*. Added footnote to *Table 1: Command line summary*. Updated *Table 10: Clock Configuration security settings*, *Table 21: Boot paths without TrustZone (TZEN = 0)*, and *Table 22: Boot paths with TrustZone (TZEN = 1)*. Added *Section 4.18.6: How to configure OEM-uRoT (STiRot uROT) boot path*, *When using H7Rx/H7Sx with MMT*, *When using H7Rx/H7Sx*, and their subsections. Added *Figure 236: MMT view (H7Rx-H7Sx devices)* and *Figure 255: Memory assignment for context Boot H7RS*. Updated *Figure 9: Package installation*, *Figure 10: Installation script*, *Figure 11: Installation path*, *Figure 236: Application boot path: ST-iRoT and uRoT secure/nonsecure project*, *Figure 238: Application boot path: ST-iRoT dual figure*, *Figure 248: Boot path selection*, *Figure 255: Boot path and debug authentication panel*, *Figure 259: Generate the code*, *Figure 269: Boot path and Debug Authentication tab*, *Figure 266: Boot path and Debug Authentication tab*, and *Figure 275: Boot path project*. Minor text edits across the whole document.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
26-Jun-2024	45	6.12.0	Updated *Section 2.2: Key features*, *Java™ Runtime Environment*, *Section 3.4.7: Checking for updates*, *Step 5: Boot path selection*, *Section 4.6: Pinout & Configuration view for STM32 MPUs*, *Section 4.18.6: How to configure OEM-uRoT (STiRot uROT) boot path*, *Section 4.19: User authentication*, *Section 4.19.1: Login with an existing my.st.com account*, and *Section 8: Code generation with TrustZone enabled (STM32L5 series only)*. Added note to *Section 3.4.2: Updater configuration*. Added *Section 4.4: Boot chain (STM32 MPUs)*, *Section 4.7: RIF configuration*, *Section 4.18.7: How to configure ST-iRoT boot path with STM32H7RS devices*, *Section 5.5: STM32CubeMX Memory Management Tool*, and their subsections. Updated *Table 1: Command line summary* and *Table 22: Boot paths with TrustZone (TZEN = 1)*. Added *Table 23: Boot paths for STM32H7RS devices*. Added *Figure 20: Connection failure* and *Figure 31: Checking for available updates*. Updated *Figure 44: Popup window - Starting a project from an example*, *Figure 283: Project creation*, *Figure 557: Project Settings and toolchain selection*, and *Figure 645: Available IPs*. Removed former *Section 4.18: STM32CubeMX Memory Management Tool*, *Section 19: FAQ*, and their subsections. Minor text edits across the whole document.
20-Nov-2024	46	6.13.0	Updated *Section 3.1.1: Supported operating systems and architectures*, *Section 3.1.3: Software requirements*, *Section 4.7.2: RIF global configurations*, *Section 4.7.4: Peripheral instance protection*, *Section 4.7.7: Masters configuration*, *Section 4.7.9: System peripherals (STM32MP2 and STM32N6 series)*, *Section 4.7.10: Memory protection for STM32MP2 series*, *Section 4.19: User authentication*, and *Section 5.5.1: STM32H5, STM32U3, STM32U5, STM32WBA5, STM32WBA5M, and STM32WBA6 with TrustZone activated* Added *Section 4.7.10: Memory protection for STM32MP2 series*, *Section 4.7.11: Memory protection for STM32N6 series*, *Section 4.7.13: Implementation of illegal access controller (IAC) feature on STM32N6 series*, *Section 5.2: Compare Projects*, and *Section 5.5.5: STM32H7 Dual-core without Trust Zone activated*

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
24-Feb-2025	47	6.14.0	Updated *Introduction*, *Section 3.1.1: Supported operating systems and architectures*, *Section 4.7.2: RIF global configurations*, *Configuration example*, *Masters configurations for STM32MP2*, *Step 8: Code compilation*, *Step 2: Code compilation*, *Section 5.2.1: User interface of the Compare Projects tool*, *Section 5.2.2: Comparing two projects*, and *Section 5.5.1: STM32H5, STM32U3, STM32U5, STM32WBA5, STM32WBA5M, and STM32WBA6 with TrustZone activated*. Updated figures *20* to *31*, *Figure 33: STM32CubeMX home page*, figures *39* to *42*, *Figure 45: Cross selector - Data refresh prerequisite*, *Figure 115: Lock and privilege in RISUP table*, *Figure 231: Boot paths for STM32H57x devices*, *Figure 234: Application boot path (OEM-iRoT)*, *Figure 246: Configuring the project*, *Figure 247: Saving the project*, *Figure 255: Boot path and debug authentication panel*, *Figure 259: Generate the code*, *Figure 270: Select the project structure*, figures *283* to *285*, *Figure 288: Boot path and debug authentication tab*, *Figure 302: Home page without the login form*, *Figure 303: Install or remove a software package*, *Figure 305: Authentication from myST tab*, figures *307 to 309*, *Figure 313: About window*, figures *316* to *317, 319* to *325, 400* to *401*, *408* to *412*, *417* to *419*, *421 to 422*, *Figure 447: ETH configuration for STM32H7R3A8Ix*, and *Figure 530: MCU selection*. Added *Figure 109: RIF configuration extension in IPs panel for the STM32MP2 series*, *Figure 110: RIF configuration extension in IPs panel for the STM32N6 series*, *Figure 231: Boot paths for STM32H57x devices*, *Figure 234: Application boot path (OEM-iRoT)*, *Figure 319: Comparison result in Excel format - Peripherals and middleware,* and *Figure 320: Comparison result in Excel format - Project settings*. Updated *Table 3: Window menu* and *Table 22: Boot paths with TrustZone (TZEN = 1)*. Removed former *Figure 31: Library deletion progress window* and *Figure 275: Boot path project*. Removed former *Section 4.18.5: How to configure an ST-iRoT with a secure manager NS application boot path*. Added *ETH impact on MMT for STM32H7 single core* and *ETH impact on MMT when using H7RS/H7SX*. Minor text edits across the whole document.

Table 30. Document revision history (continued)

Date	Revision	STM32CubeMX release number	Changes
20-Jun-2025	48	6.15.0	Updated *Section 3.3.2: Running STM32CubeMX in command-line mode*, *RISAF 5: PCIE memory configuration*, *Section 4.11: Project Manager view*, *Section 5.5.1: STM32H5, STM32U3, STM32U5, STM32WBA5, STM32WBA5M, and STM32WBA6 with TrustZone activated*, and *Section 11.7.2: Downloading firmware package and generating the C code*. Added *Section 4.12.1: Import Project feature for STM32MCU projects* and *Section 4.12.2: Import Project feature for STM32MPU projects*. Updated *Table 1: Command line summary* and *Table 4: Help menu shortcuts*. Added note to *HSEM*. Added figures *163 to 165, Figure 188: Compiler option for CMake toolchain,* and *566 to 571*. Updated *Figure 187: Project Settings window*, figures *224 to 227, Figure 361: Regions settings to linker files ON*, figures *530 to 532,* figures *534 to 536, Figure 559: Warning message for missing firmware package,* and figures *561 to 565.* Minor text edits across the whole document.
07-Nov-2025	49	6.16.0	Updated *Video tutorials*, *Section 3.1.1: Supported operating systems and architectures*, *Section 3.2.1: Installing STM32CubeMX standalone version*, and *Section 6.2: STM32Cube code generation using Low Layer drivers*. Added *Section 5.5.8: MMT for STM32N6 products*. Added note to *Section 15: Tutorial 5: Exporting current project configuration to a compatible MCU*. Updated *Figure 196: Project Settings code generator, Figure 198: Generated project template*, *Figure 225: CAD Resources view*, figures *227* to *229*, and *610* to *611*. Removed former *Section 4.19: User authentication and Appendix C: STM32 microcontrollers naming conventions.* Minor text edits across the whole document.