AN5564

Application note

Getting started with projects based on dual-core STM32WL microcontrollers in STM32CubeIDE

Introduction

This application note describes how to get started with projects based on STM32WL Series dual-core microcontrollers in the STMicroelectronics STM32CubeIDE integrated development environment.

1 General information

STM32CubeIDE supports STM32 32-bit products based on the Arm® Cortex® processor.

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

1.1 Prerequisites

The following tools are prerequisites for understanding the tutorial in this document and developing an application based on the STM32WL Series:

- STM32CubeIDE 1.5.0 or newer
- STM32Cube_FW_WL_V1.0.0 or newer
- STM32CubeMX 6.1.0 or newer

Users are advised to keep updated with the documentation evolution of the STM32WL Series at www.st.com/en/microcontrollers-microprocessors/stm32wl-series.html.

1.2 The use cases in this document

In the STM32CubeIDE context, users have different ways to explore and get started with the development of projects based on the STM32WL Series. From the list below, select the description that best fits the use case considered and refer to the corresponding section in this application note:

- Create an STM32CubeMX project using the STM32CubeMX tool integrated inside STM32CubeIDE, or the stand-alone STM32CubeMX tool
- Import an STM32CubeIDE project from the STM32CubeWL MCU Package to learn by using an example project

1.3 Specific features of dual-core microcontrollers in the STM32WL Series

- Advanced security use cases (such as security area, isolated and protected code and secrets on locked Cortex®-M0+ side, and others)
- Applicative flexibility with both cores opened for developers
- Real time capabilities for standard sub-GHz stacks or proprietary protocols when required, with a dedicated core

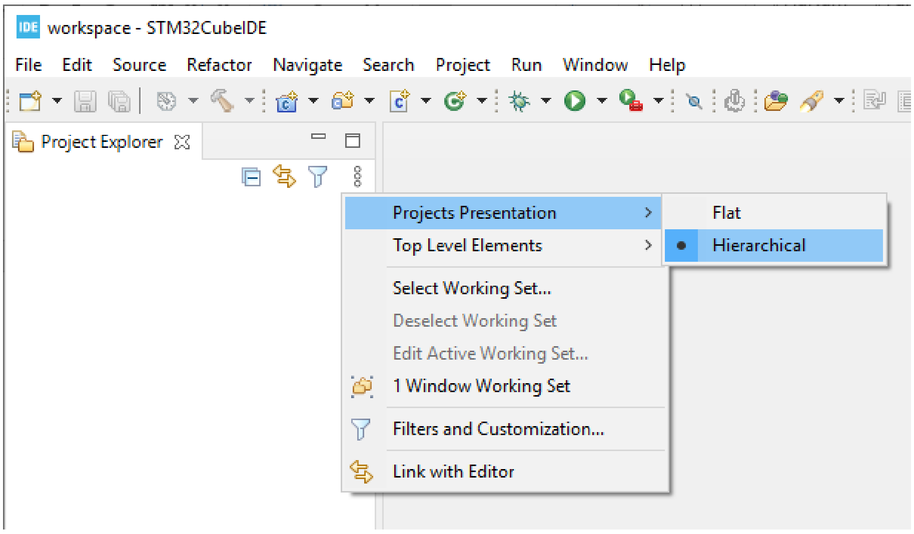
1.3.1 Dual-core STM32WL project structure

When a dual-core STM32WL project is created, its structure is automatically made hierarchical. The project structure for single-core projects is flat. When the user creates or imports a dual-core STM32WL project, it consists of one root project together with sub-projects, referred to as MCU projects, for each core.

The MCU projects are real CDT™ projects that can contain both build and debug configurations, as opposed to the root project, which is a simple container allowing common code sharing between the cores. The root project can contain neither build nor debug configurations.

If the project is not shown in a hierarchical structure, this can be changed as shown in Figure 1.

Figure 1. Setting the project hierarchical view



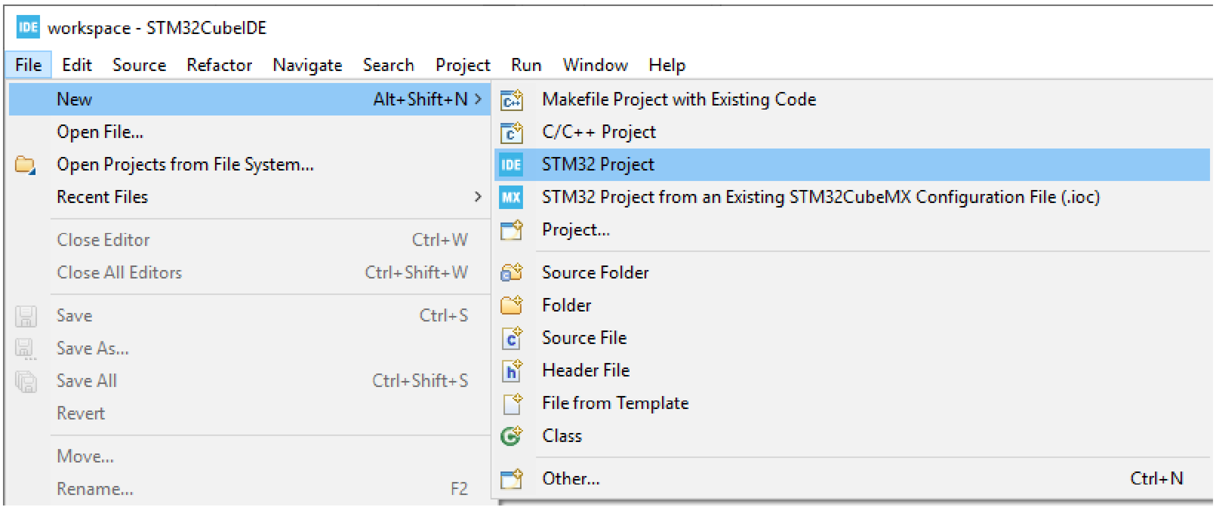
2 Create and import projects

This chapter describes how to create or import projects for dual-core microcontrollers in the STM32WL Series.

2.1 Create a new STM32 project

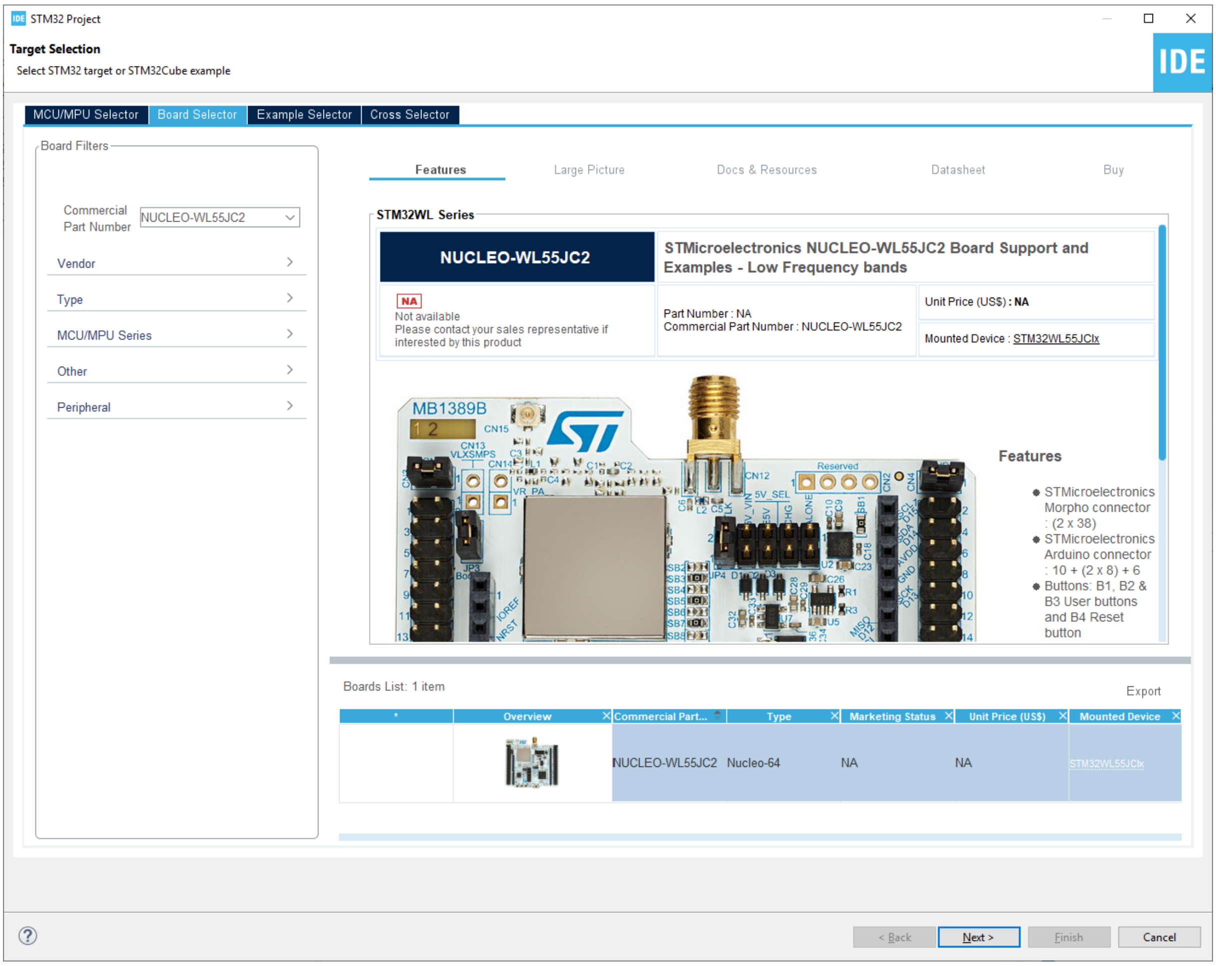
To start a new project, go to [**File**]>[**New**]>[**STM32 Project**] as shown in Figure 2.

Figure 2. **New STM32 project**



Select the desired MCU or board. In the example shown in Figure 3, the selected board is the NUCLEO-WL55JC2. Click on [**Next >**].

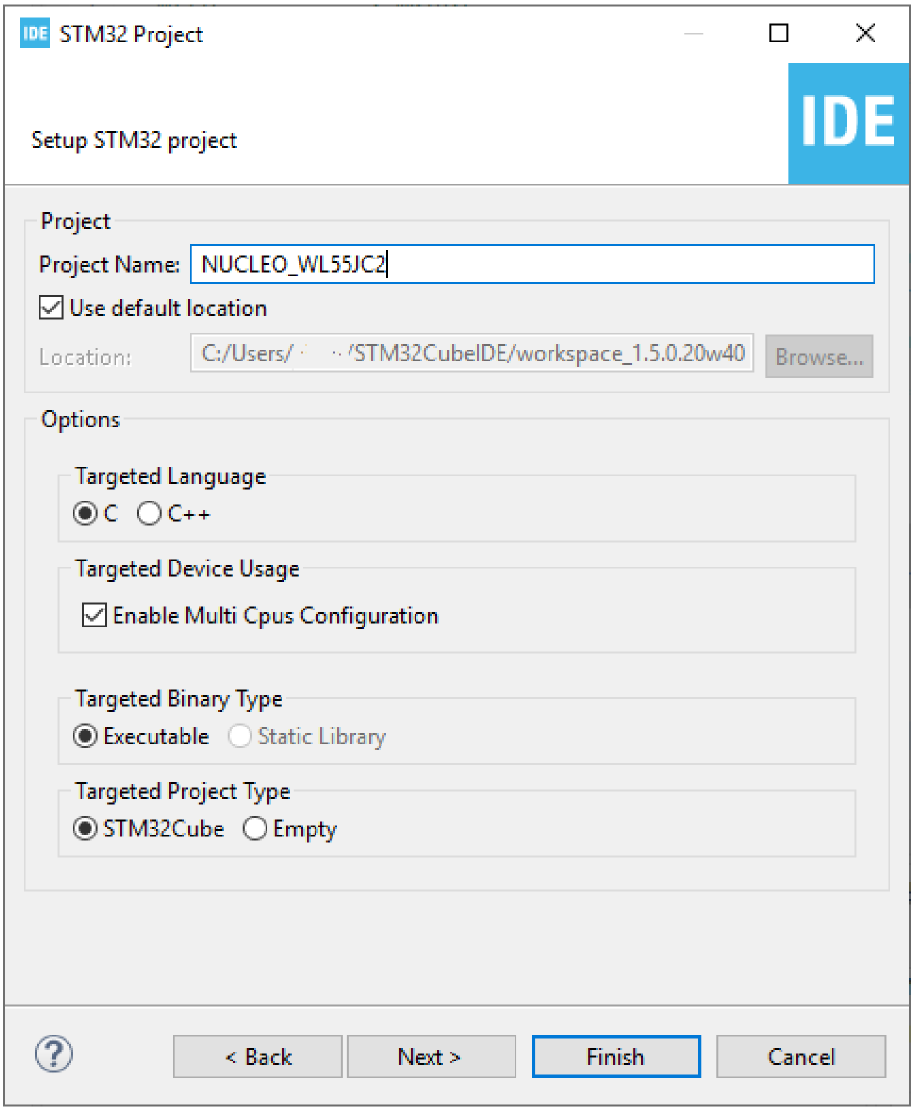
Figure 3. Board selector



After the target selection comes the project setup step shown in Figure 4. The *Targeted Project Type* setting determines whether the project gets generated by STM32CubeMX or not. An *Empty* project is a skeleton of a project that needs building upon while *STM32Cube* indicates an STM32CubeMX-managed project.

- *Empty* projects contain the bare-minimum code to build an debug an empty `main()`.
- *STM32Cube* projects are managed by STM32CubeMX. Drivers and middleware are generated in the project based on the configurations done in the `.ioc` file editor in STM32CubeIDE (the `.ioc` file editor is the integrated version of STM32CubeMX).

Figure 4. Projet setup

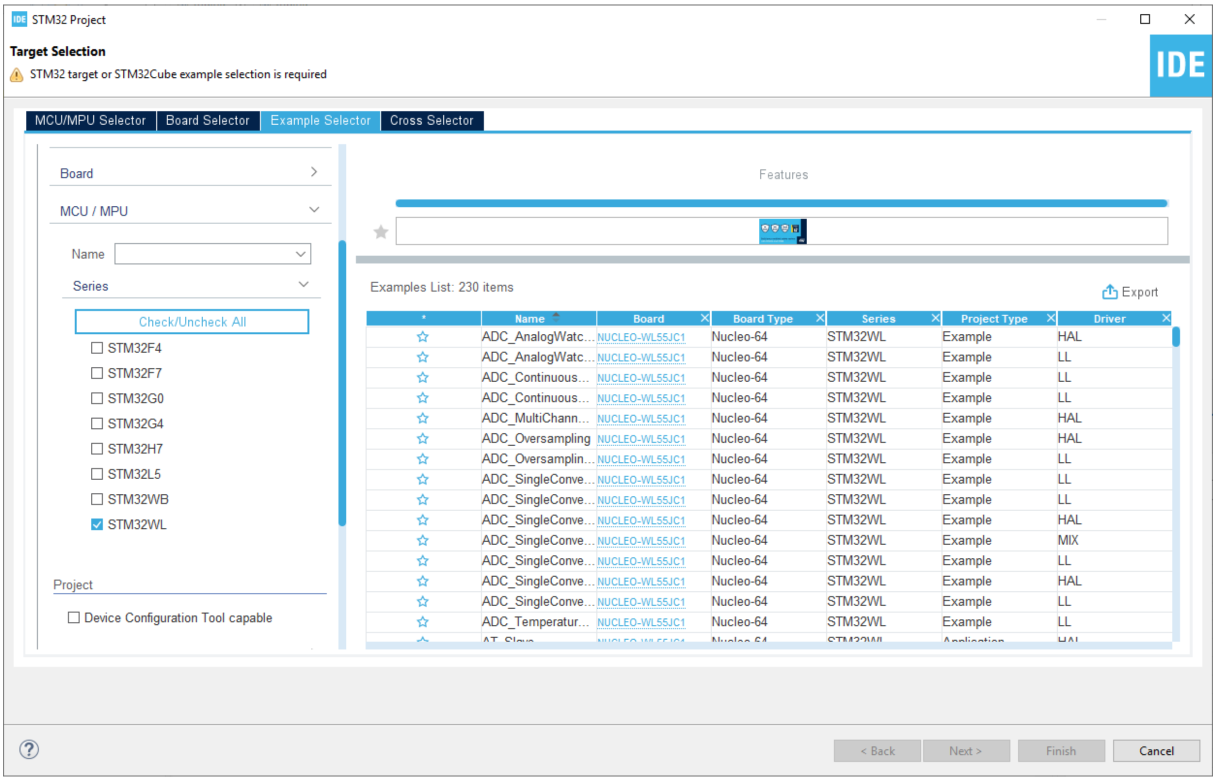


*Note: Select the [**Enable Multi Cpus Configuration**] option to allow the creation of multicore project with both the Cortex®-M0+ and Cortex®-M4. Unselect this option to create a project with the Cortex®-M4 only.*

2.2 Import a project from the STM32CubeWL MCU Package

To import the STM32Cube firmware project into STM32CubeIDE, go to [**File**]>[**New**]>[**STM32 Project**] as shown in Figure 5 and select the desired example.

Figure 5. Project example selector



*Note: Users can also import projects using the import mechanism by going to [**File**]>[**Project...**]>[**Import...**] and selecting [**Existing Projects into Workspace**].*

3 Debugging

This chapter highlights some of the points to bear in mind while debugging a device in the STM32WL Series.
In the next two sections, this application note covers the configurations needed to start debug sessions with ST-LINK GDB server and OpenOCD.

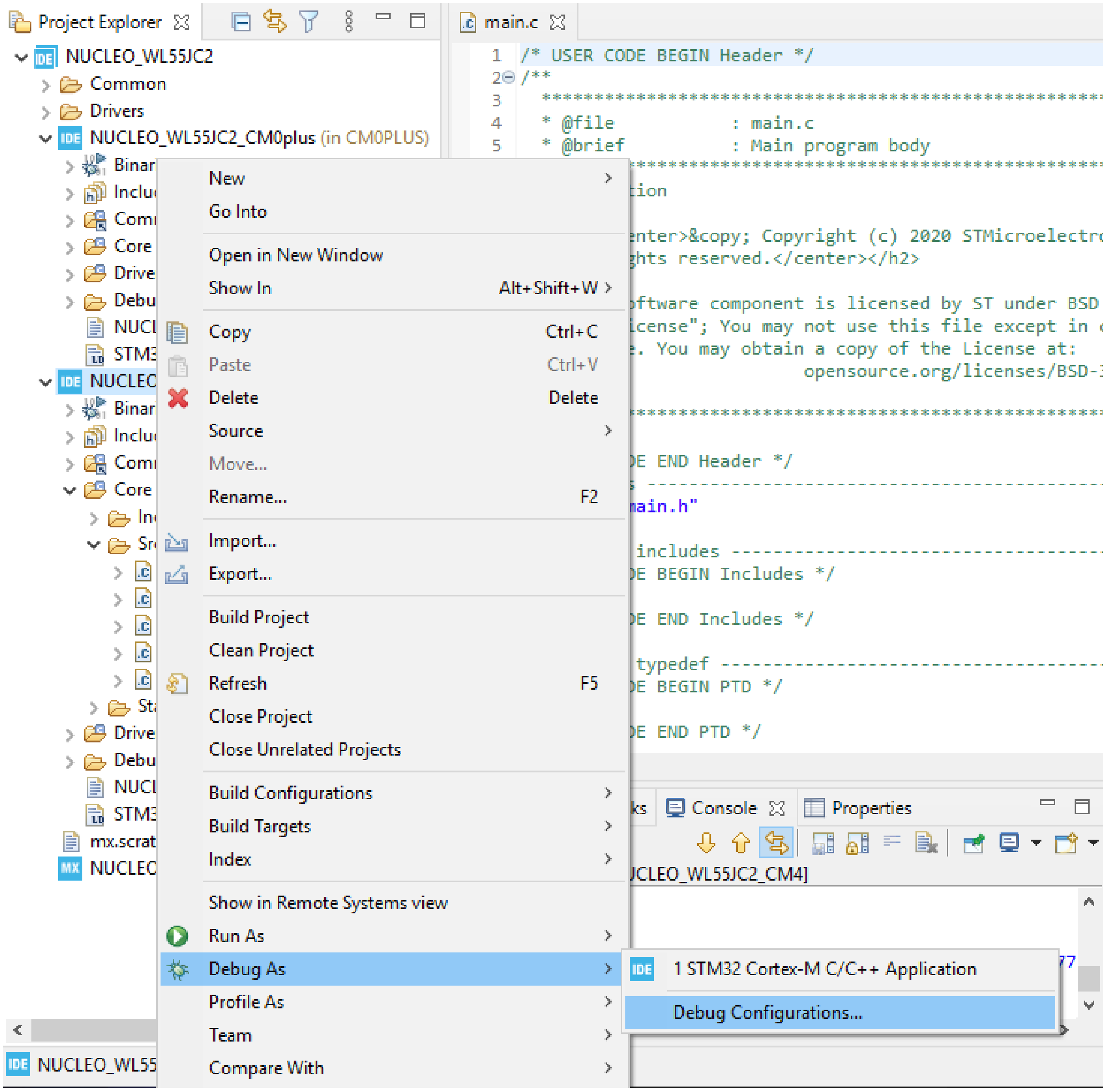
Note: *By default, the Cortex®-M0+ is not available until the C2BOOT bit is set in power control register 4 (PWR_CR4). It is the user's responsibility to enable the C2BOOT bit through the application code running on the Cortex®-M4. If there is no security enabled on the Cortex®-M0+, STMicroelectronics recommends to use access port 0 (AP0) to program both CPUs.*

3.1 Setting up with ST-LINK GDB server

To create a debug configuration using ST-LINK GDB server, perform the following steps:

1. Select the Cortex®-M4 project in the *Project Explorer* view
2. Right-click [**Debug As…**], select [**Debug Configuration…**], and then double-click on [**STM32 Cortex-M C/C++ Application**]

Figure 6. ST-LINK GDB server debug configuration (1 of 6)

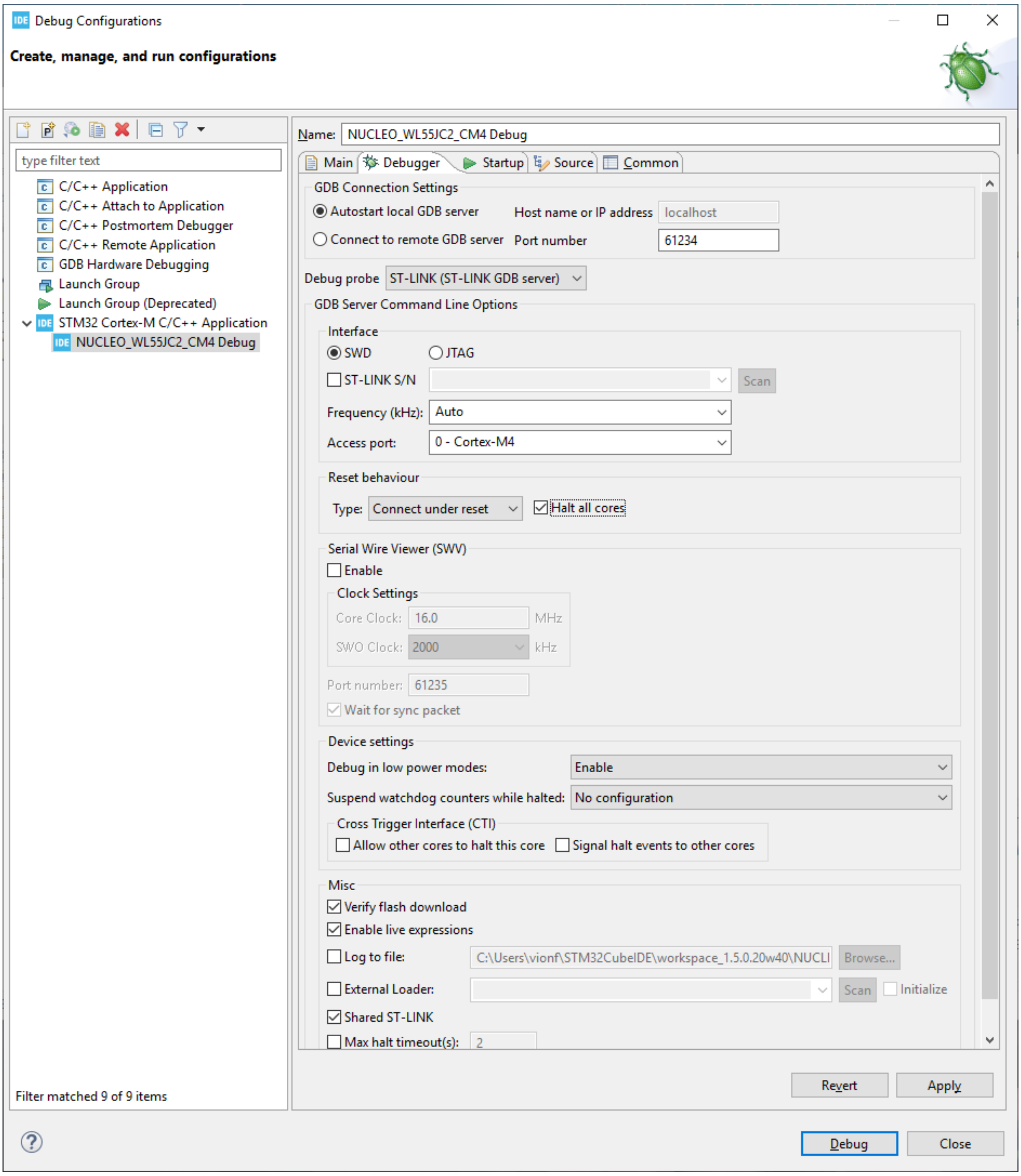


Debugger tab - Cortex®-M4

To use the ST-LINK GDB server (refer to Figure 7), make sure that:

- The type of reset behavior is selected as *Connect under reset*
- The *Halt all cores* option is enabled
- The *Shared ST-LINK* is enabled

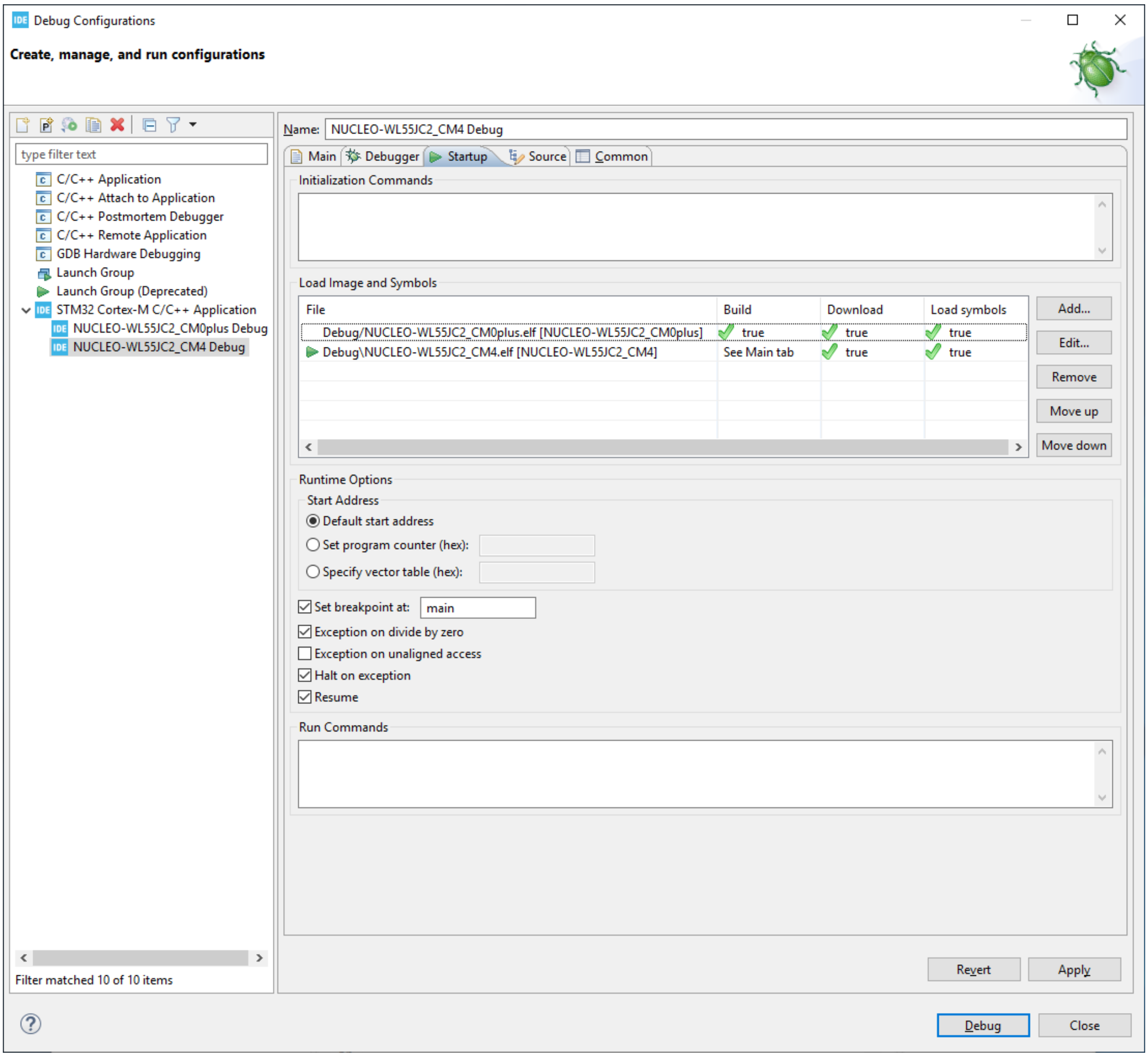
Figure 7. ST-LINK GDB server debug configuration (2 of 6)



Startup tab - Cortex®-M4

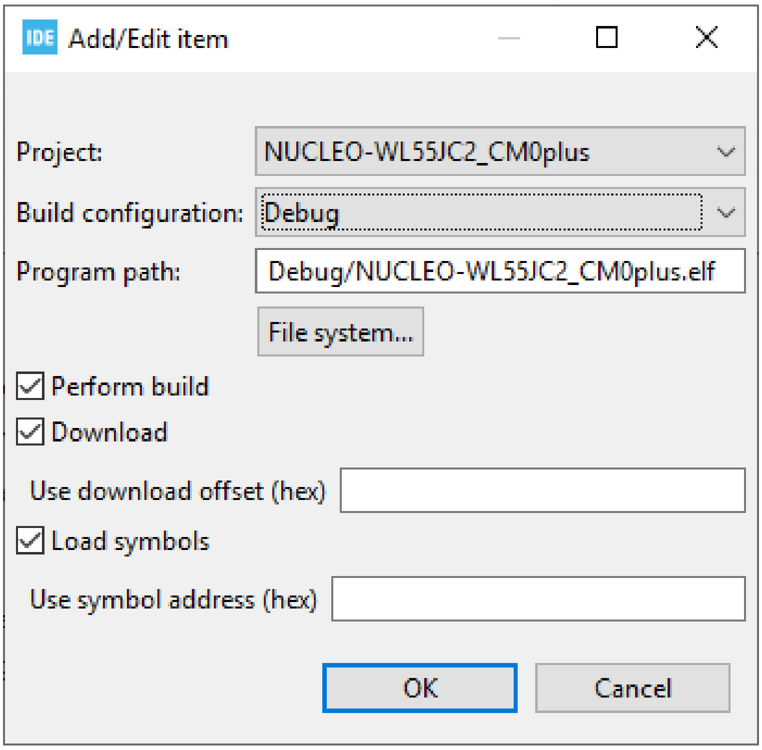
The Cortex®-M4 debug configuration is responsible for loading both the Cortex®-M4 and Cortex®-M0+ images. Go to the *Startup* tab to set this up as shown in Figure 8.

Figure 8. ST-LINK GDB server debug configuration (3 of 6)



To also download the Cortex®-M4 image, click on **[Add...]**, point to the right project and build the configuration. The result is shown in Figure 9.

Figure 9. ST-LINK GDB server debug configuration (4 of 6)



The order in the load list is very important. The debugger sets the start of execution using the entry point of the last loaded image in this list. In practice, this means that the program counter for the Cortex®-M4 is set to the location of the `Reset_Handler` of the Cortex®-M4 binary. This is indicated by the green arrow.

Note: It is not necessary to load symbols for the Cortex®-M0+ in the debug configuration of the Cortex®-M4 because they are loaded in the debug configuration of the Cortex®-M0+.

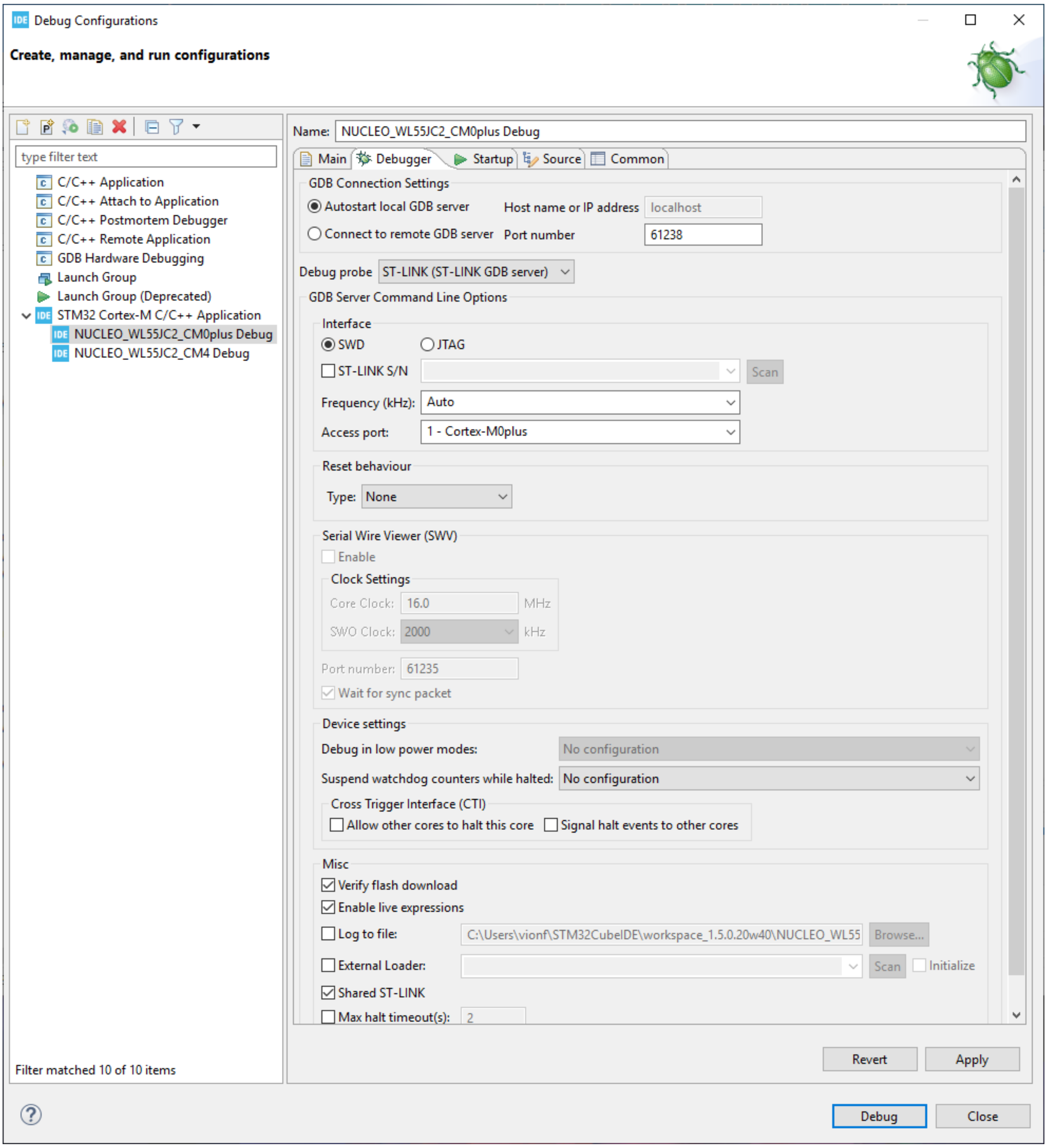
These steps conclude the debug configuration for the Cortex®-M4. The next steps present the creation of a debug configuration for the Cortex®-M0+ based on the Cortex®-M0+ project.

***Debugger* tab - Cortex®-M0+**

Contrary to the Cortex®-M4, as shown in Figure 10:

- Make sure that the *Port number* exceeds the value of the previous debug configuration by at least 3 (61238 in this example)
- Select *1 – Cortex-M0+* for [**Access port**]
- Select *None* for [**Reset behaviour**]
- Select *Shared ST-LINK*

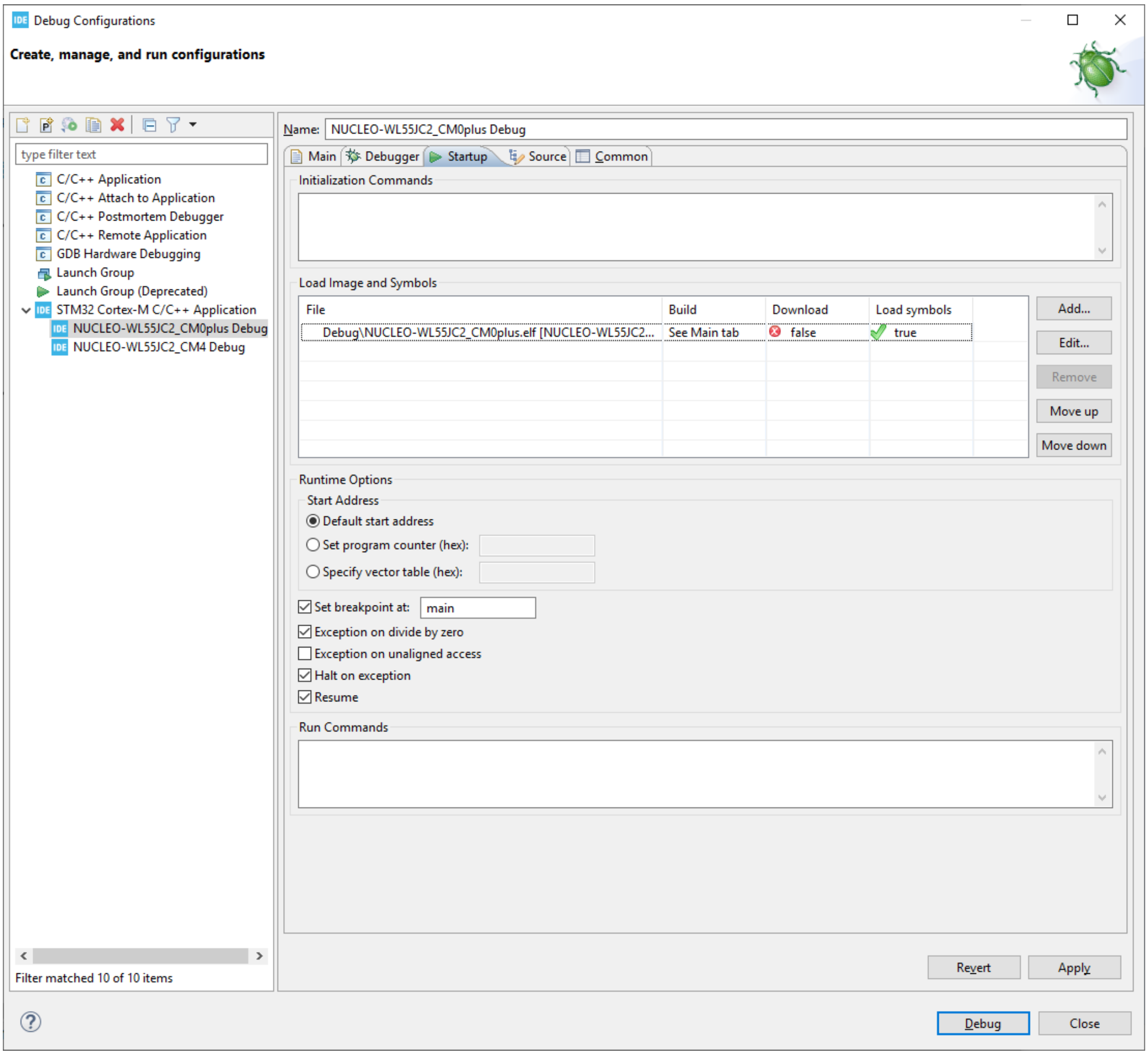
Figure 10. ST-LINK GDB server debug configuration (5 of 6)



Startup tab - Cortex®-M0+

Go to the *Startup* tab and select [**Edit…**]>[**Disable Download**]. This is required since the download is already performed by the Cortex®-M4 configuration.

Figure 11. ST-LINK GDB server debug configuration (6 of 6)



The configuration is complete.

Note: *It is possible to program the Cortex®-M0+ through access port 1:*

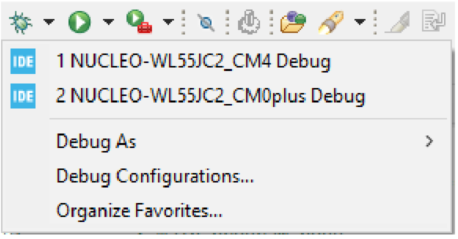
- *Delete the binary of the Cortex®-M0+ from the list in the Startup tab of the Cortex®-M4 debug configuration*
- *Enable the download of the binary on the Startup tab of the Cortex®-M0+ debug configuration*

Flash programming via access port 1 in “hot-plug” (Cortex®-M0+) fails if the existing application code on the Cortex®-M0+ enables interrupts.

3.1.1 Launching the configurations

1. Launch the Cortex®-M4 configuration to download both the Cortex®-M0+ and Cortex®-M4 images.
2. Resume the Cortex®-M4 core until the C2BOOT bit is set in power control register 4 (PWR_CR4) to enable the Cortex®-M0+.
3. Launch the Cortex®-M0+ configuration using the arrow next to the debug icon. The Cortex®-M0+ is in the running mode and the user can halt it after the debugger is started.

Figure 12. ST-LINK GDB server debug configuration launch



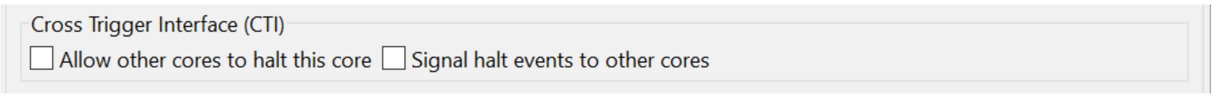
Note: After creating the debug configurations for both cores, they are not shown in the scroll-down menu if they have never been launched before. This is because the arrow provides access to the history of latest launches, with a grayed "no history" message if there are none. First-time debug launch must be done through the "Debug Configurations..." wizard.

3.1.2 Cross-trigger Interface

The cross-trigger interface is used to send halt signals from one core to the other. To enable the Cortex®-M0+ to halt the Cortex®-M4, apply the following configuration:

- In the Cortex®-M0+ debug configuration: select *Signal halt events to other cores*
- In the Cortex®-M4 debug configuration: select *Allow other cores to halt this core*

Figure 13. ST-LINK GDB server debug cross-trigger interface



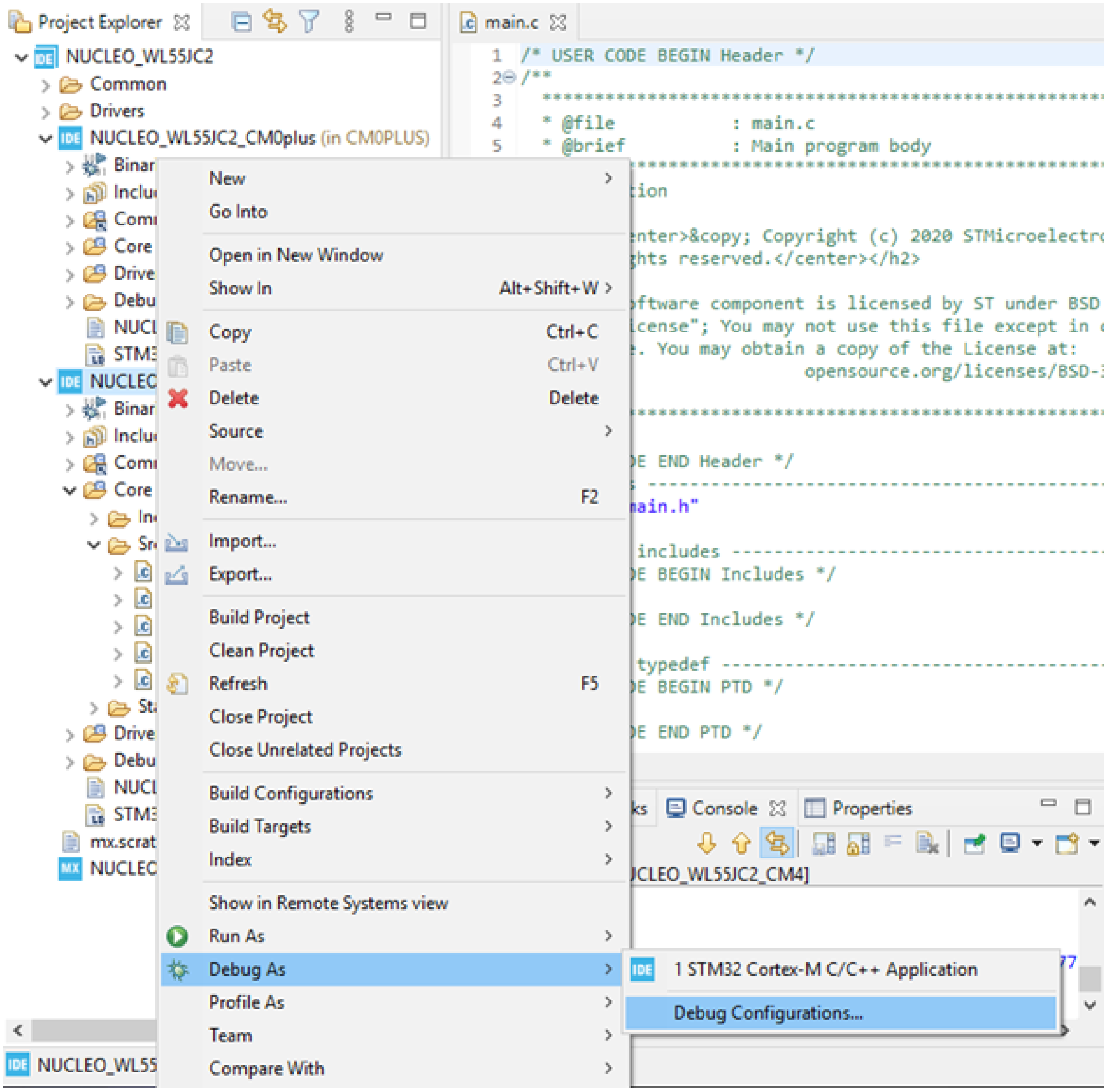
Note: Checking both checkboxes in both debug configurations enables both cores to halt each other.

3.2 Setting up with OpenOCD

To create a debug configuration using OpenOCD, perform the following steps:

1. Select the Cortex®-M4 project in the *Project Explorer* view
2. Right-click [**Debug As…**], select [**Debug Configuration…**], and then double-click on [**STM32 Cortex-M C/C++ Application**]

Figure 14. OpenOCD debug configuration (1 of 3)



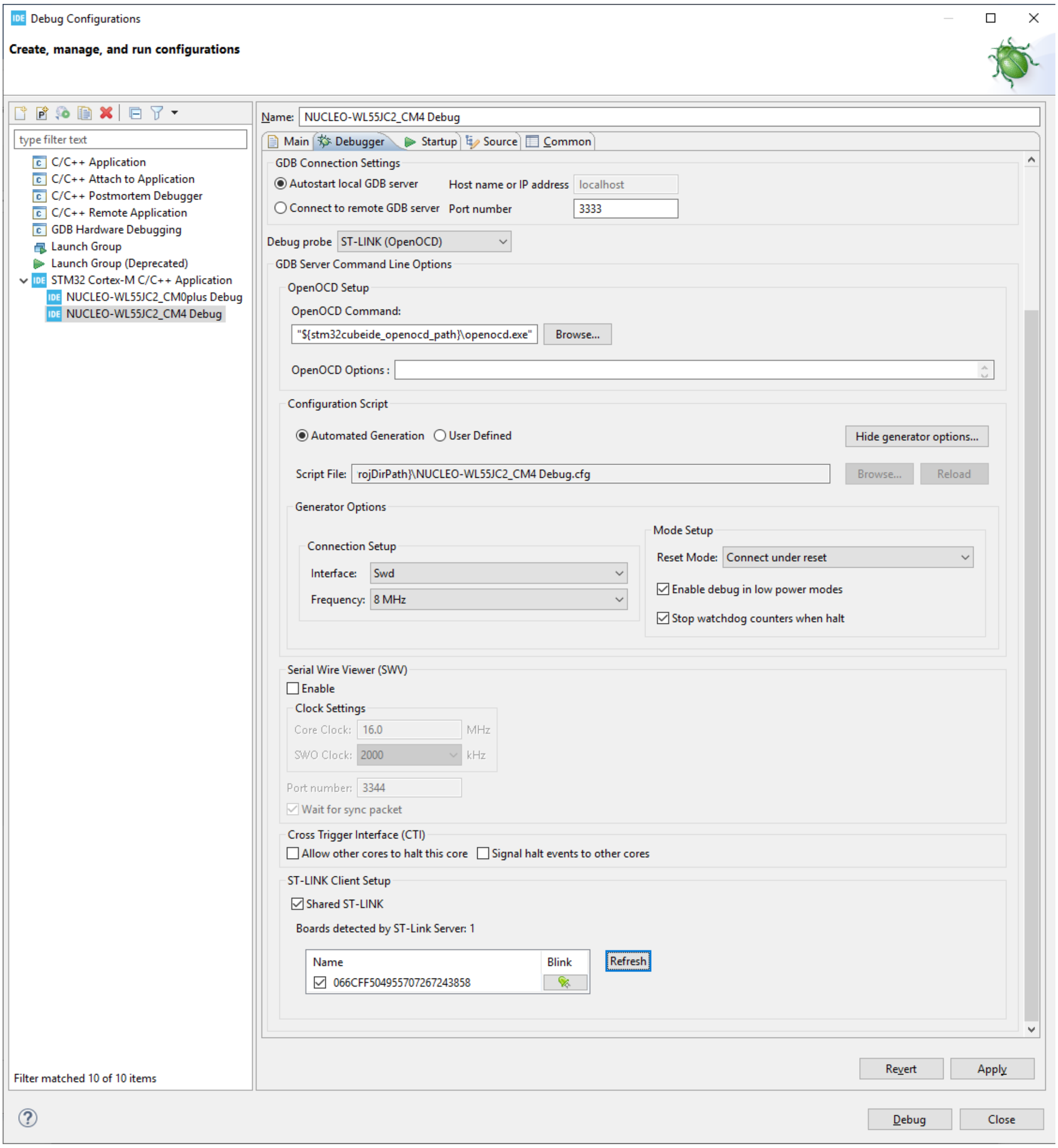
***Debugger* tab - Cortex®-M4**

Select *ST-LINK (OpenOCD)* as the [**Debug probe**]. Select *Autostart local GDB server* for the configuration that launches first, which is the Cortex®-M4 in the example in Figure 15.

Set all the default options and verify that:

- *Connect under reset* is selected as [**Reset Mode**].
- [**Shared ST-LINK**] is selected; this option is mandatory to run the multicore target.

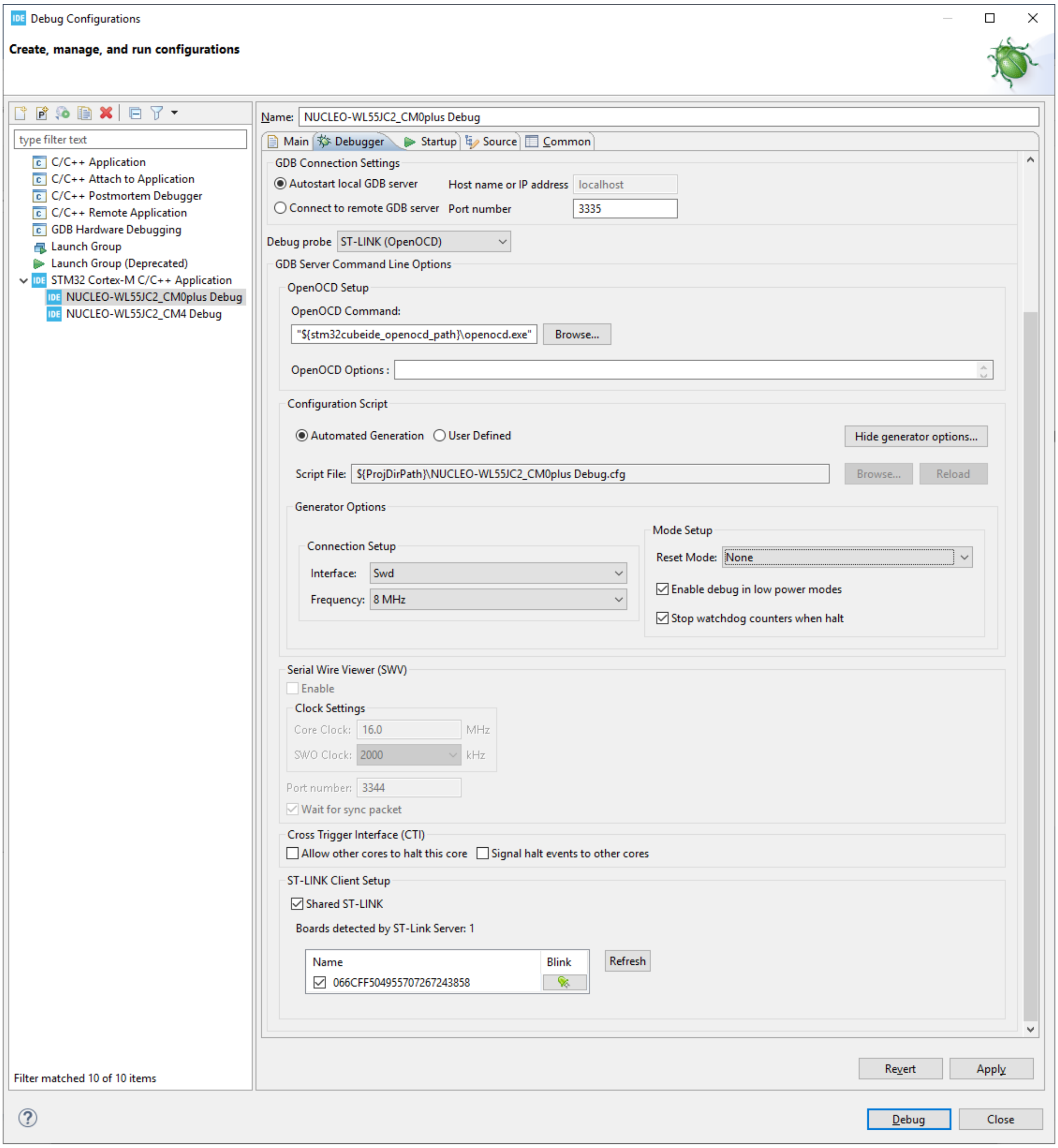
Figure 15. OpenOCD debug configuration (2 of 3)



Create the debug configuration for the other core, which is the Cortex®-M0+ in the example in Figure 16:

- Select *ST-LINK (OpenOCD)* as the [**Debug probe**]
- Select *Autostart local GDB server* as default
- Make sure that the *Port number* exceeds the value of the previous debug configuration by at least 2 (3335 in this example)
- Open [**Generator Options**] and select *None* as [**Reset Mode**]

Figure 16. **OpenOCD debug configuration (3 of 3)**



The configuration of the *Startup* tab is the same as with the ST-LINK GDB server probe for both debug configurations (refer to *Startup* tab - Cortex®-M4 and *Debugger* tab - Cortex®-M0+ in Section 3.1 Setting up with ST-LINK GDB server).

To launch the debug and enable the cross-trigger interface, refer to Section 3.1.1 Launching the configurations and Section 3.1.2 Cross-trigger Interface.

4 Limitations

STM32CubeIDE is subject to the limitations listed below:

- The Flash memory of the Cortex®-M0+ through access port 1 (AP1) is not working using STM32CubeIDE with the OpenOCD probe. There is also a limitation using the ST-LINK GDB server if the existing application code on the Cortex®-M0+ enables interrupts. To program through AP1, it is recommended to use the external tool STM32CubeProgrammer (STM32CubeProg) version 2.6.0 or above. After programming, refer to Section 3 to create the debug configurations without programming the Cortex®-M0+.
- When the security is enabled on the Cortex®-M0+, it is also recommended to use the external tool STM32CubeProgrammer to program the Cortex®-M0+ using OpenOCD.

Table 1. Programming/debugging status using access port 1 (AP1)

Initial conditions	Behaviour on Cortex®-M0+ using AP1	STM32CubeIDE probe	
		ST-LINK GDB server	OpenOCD
The whole Flash memory is empty	Download only	Use STM32CubeProgrammer standalone(1)	Use STM32CubeProgrammer standalone(1)
• Application code already loaded for the Cortex®-M4 • The Flash memory is empty for the Cortex®-M0+	Download only	Use theCortex®-M0+ debug configuration to program the Flash memory	Use STM32CubeProgrammer standalone(1)
Application code already loaded for both Cortex®-Mx	Debug only	Refer to Section 3	Refer to Section 3
The Flash memory security sector is enabled and the system is in secure mode for the Cortex®-M0+	Download only	Use theCortex®-M0+ debug configuration to program the Flash memory	Use STM32CubeProgrammer standalone(1)
The Flash memory security sector is enabled and the system is in secure mode for the Cortex®-M0+	Debug only	Refer to Section 3	Refer to Section 3

1. *STM32CubeProgrammer (STM32CubeProg) is available from www.st.com*

Revision history

Table 2. **Document revision history**

Date	Version	Changes
17-Nov-2020	1	Initial release.
14-Dec-2020	2	Added and updated the *Shared ST-LINK* configuration in *Debugger* tab - Cortex®-M4 and *Debugger* tab - Cortex®-M0+.

Contents

List of tables

List of figures